# Reducing hard SAT instances to polynomial ones

Olivier Fourdrinoy Éric Grégoire Bertrand Mazure Lakhdar Saïs
CRIL CNRS & IRCICA
Université d'Artois
Rue Jean Souvraz SP18
F-62307 Lens Cedex France
{fourdrinoy, gregoire, mazure, sais}@cril.fr

## Abstract

*This last decade, propositional reasoning and search has been one of the hottest topics of research in the A.I. community, as the Boolean framework has been recognized as a powerful setting for many reasoning paradigms thanks to dramatic improvements of the efficiency of satisfiability checking procedures. SAT, namely checking whether a set of propositional clauses is satisfiable or not, is the technical core of this framework. In the paper, a new linear-time pre-treatment of SAT instances is introduced. Interestingly, it allows us to discover a new polynomial-time fragment of SAT that can be recognized in linear-time, and show that some benchmarks from international SAT competitions that were believed to be difficult ones, are actually polynomial-time and thus easy-to-solve ones.*

## 1 Introduction

These last decade, propositional reasoning and search has been one of the hottest topics of research in the A.I. community, as the Boolean framework has been recognized as a powerful setting for many reasoning paradigms thanks to dramatic improvements of the efficiency of satisfiability checking procedures. SAT, namely checking whether a set of propositional clauses is satisfiable or not, is the technical core of this framework. In the paper, a new linear-time pre-treatment of SAT instances is introduced. Interestingly, it allows us to discover a new polynomial-time fragment of SAT that can be recognized in linear-time, and show that some benchmarks from international SAT competitions that were believed to be difficult ones, are actually polynomial-time and thus easy-to-solve ones. Many approaches have been proposed to solve hard SAT instances. Direct approaches have focused on the development of -logically complete or not- algorithms. Local-search techniques (e.g. [21]) and elaborate variants of the Davis-Loveland-Logemann's DPLL procedure [8] (e.g. [18, 12]) allow many families of difficult instances to be solved. Indirect approaches aim at solving instances, using either approximation or compilation techniques (see e.g. [7, 3, 20]). In particular, compilation techniques, which were developed in the more general framework of propositional deduction, aim at transforming the set of Boolean clauses into a deductively equivalent form that belongs to a polynomial fragment, making use of a -possibly exponential-transformation schema and by ensuring that the compiled form remains tractable in size. Finally, other approaches have concentrated on discovering and studying fragments of SAT that can be recognized and solved in polynomial time (see e.g. [10, 5, 2, 4]). The contribution of this paper pertains to these three families of approaches. A new pre-treatment of SAT instances is introduced: it can be performed before some direct approaches are run. It can be interpreted as an attempt to compile the SAT instance into an easier-to-solve one. However, contrary to usual compilation techniques, the transformation process remains a polynomial-time one, and no guarantee is provided that the resulting set of clauses belongs to a polynomial fragment. However, this pre-treatment can prove valuable in showing that some instances are actually polynomial ones or in making the further solving step become more efficient. Finally, a new polynomial fragment of SAT, called U-Horn SAT (*Horn modulo Unit propagation*), is put in light. Interestingly, it can be recognized using the proposed polynomial-time pre-treatment. In other words, SAT instances that can be mapped to Horn SAT using our approach belong to the U-Horn SAT fragment. Roughly, this pre-treatment is as follows. The focus is on the unit propagation mechanism (in short UP), which is a linear-time deductive mechanism. Given a polynomial fragment (e.g. the Horn one), any SAT instance can be divided into two subsets of clauses: the first one contains clauses that belong to the targeted polynomial fragment whereas the second cone contains clauses that do not belong to it. For each of these latter clauses, we attempt to discover one sub-clause belonging to the polynomial fragment, using UP. In case of success, this sub-clause can replace the initial one, and increase the size of the polynomial subset. In case of failure, we also check whether the clause itself is an UP consequence of the instance or not. The paper is organized as follows. In the next section, the basic formal background is provided, together with the description of propositional fragments that will be mentioned in the paper. Then, the pre-treatment is described, before extensive experimental studies are reported and analyzed. Then, it is shown how this approach extends some previous related works and other SAT-related approaches exploiting the unit propagation mechanism.

## 2 Technical background

Let $\mathcal{L}$ be a standard Boolean logical language built on a finite set of Boolean variables, noted $a$, $b$, $c$, 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 $\mathcal{L}$ 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 Boolean variable that can negated. Clauses will be represented by the set of literals that they contain. For example, the clause $C = a \vee b \vee \neg c \vee \neg d$ will be represented by the set $\{a, b, \neg c, \neg d\}$. A clause is said to be positive (resp. negative) if it contains no negative (resp. positive) literal. The size of a clause is the number of literals in it. Unit clauses contain exactly one literal whereas binary ones contain at most two literals. The empty clause is denoted by $\perp$. A clause $C$ is a sub-clause of a clause $D$ iff $C \subseteq D$. For example, the resolvent of $C_1 = (p \vee \alpha)$ and $C_2 = (\neg p \vee \beta)$ is defined as $Res(C_1, C_2) = (\alpha \vee \beta)$ (Resolution rule); it is a logical consequence of $C_1$ and $C_2$. SAT is the NP-complete problem that consists in checking whether a set of Boolean clauses (also called CNF) is satisfiable or not, i.e. whether there exists an interpretation that satisfies all clauses in the set or not. A central deductive mechanism in this paper is the unit propagation mechanism (in short UP). UP is a linear time process that recursively simplifies a SAT instance by propagating the constraints expressed by unit clauses. Let $\Sigma$ be a SAT instance, $UP(\Sigma)$ is defined as the formula obtained by unit propagation. A clause $C$ is a UP consequence of $\Sigma$; noted $\Sigma \models^* C$, iff $UP(\Sigma \wedge \neg C)$ allows to derive the empty clause. A clause $C'$ is called a sub-clause of $C$ if $C' \subset C$. A sub-clause $C'$ of $C$ is called maximal if $|C| - |C'| = 1$. Some fragments of $\mathcal{L}$ exhibit polynomial-time algorithms for SAT. Among them, let us mention the Horn fragment, which is made of Horn clauses only. A Horn (resp. reverse Horn) clause contains at most one positive (resp. negative) literal. Binary and renamable Horn clauses also form polynomial fragments: renamable Horn clauses are clauses that can be transformed into Horn ones by systematically replacing some negative literals by new Boolean variables. Let us also mention Dalal and Etherington's hierarchy of classes [5] and the class of Q-Horn [2] formulas, which strictly contains all binary, Horn reverse, and renamable-Horn clauses. All of them can be recognized and solved in polynomial time. A polynomial fragment of $\mathcal{L}$ of special interest in this paper is Quad, introduced by Dalal [4]. Quad is based on a tractable fragment called $Root$. A formula $\Sigma$ is in class $Root$, if either (1) $\Sigma$ contains the empty clause, or (2) $\Sigma$ contains no positive clause, or (3) $\Sigma$ contains no negative clause, or (4) all clauses of $\Sigma$ are binary. A formula $\Sigma$ is in class Quad[4] if either (1) $UP(\Sigma)$ belongs to Root, or (2) for the first max sub-clause $C'$ of the first clause $C \in UP(\Sigma)$ for which $UP(\Sigma \wedge \neg C')$ is in class Root: (a) either $UP(\Sigma \wedge \neg C')$ is unsatisfiable, or (b) the formula $(\Sigma \setminus \{C\}) \cup \{C'\}$ is in class Quad. As mentioned by Dalal, Quad depends on the considered ordering of clauses. Different orderings might lead to different Quad classes.

## 3 A new pre-treatment

The central idea is to reduce the non-polynomial fragment of the SAT instance $\Sigma$ through the use of UP. $\Sigma$ can be divided in two parts. The first one is formed by polynomial-time detectable clauses w.r.t. a given polynomial fragment, like Horn, reverse-Horn or strictly positive formulas, etc. The second one contains the remaining clauses. When the second part is empty, $\Sigma$ is polynomial. Different forms of sub-clause deduction can be defined depending on the targeted polynomial class. For example, if binary clauses are considered then sub-clause deduction of binary clauses will concern clauses whose length is larger than two. In the following, we instantiate this general approach by selecting the Horn fragment as the polynomial target. First, let us introduce some necessary definitions.

**Definition 1 (U-Horn clause)**
*Let* $C = \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_p\}$*, with* $n \geq 0$ *and* $p > 1$ *a clause of* $\Sigma$*.* $C$ *is called a U-Horn clause of* $\Sigma$ *iff* $\exists C' = \{\neg n_1, \ldots, \neg n_n, p_i\}$ *a sub-clause of* $C$*, s.t.* $\Sigma \models^* C'$ *or* $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n\}$*.*

**Property 1** *If* $C \in \Sigma$ *is a U-Horn clause and* $C' \subset C$ *is a Horn clause s.t.* $\Sigma \models^* C'$ *then* $\Sigma$ *is satisfiable if and only if* $(\Sigma \setminus \{C\}) \cup \{C'\}$ *is satisfiable.*

The above property states that when a clause $C$ of $\Sigma$ is U-Horn, it can be replaced in $\Sigma$ by a Horn clause without any change in the satisfiability status of $\Sigma$.

**Definition 2 (U-redundant [13])**
*A clause $C$ of $\Sigma$ is called U-redundant iff $\Sigma \setminus \{C\} \models^* C$ .*

**Property 2** *If $C \in \Sigma$ is a U-redundant clause then $\Sigma$ is satisfiable iff $\Sigma \setminus \{C\}$ is satisfiable.*

Thus, U-redundant clauses can be safely removed from $\Sigma$. Let us now introduce two properties, leading to the introduction of two new additional reduction operators, namely $U\text{-}NRes$ and $U\text{-}PRes$, respectively.

**Property 3** *Let $C = \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_p\}$ be a clause of $\Sigma$. If $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n\}$ or $\exists p_i \in C$ s.t. $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, p_i\}$ and $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, \neg p_i\}$, then $\Sigma \models \{\neg n_1, \ldots, \neg n_n\}$.*

**Definition 3 (U-NRes)**
*When a clause $C = \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_p\}$ of $\Sigma$ satisfies Property 3, $U\text{-}NRes(C)$ is defined as $\{\neg n_1, \ldots, \neg n_n\}$.*

**Property 4** *Let $C = \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_p\}$ be a clause of $\Sigma$. If $\exists p_i \in C$ s.t. $\Sigma \not\models^* \{\neg n_1, \ldots, \neg n_n, p_i\}$ and $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, \neg p_i\}$, then $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_p\}$.*

**Definition 4 (U-PRes)**
*When a clause $C = \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_p\}$ of $\Sigma$ satisfies Property 4 w.r.t. the literals $p_i$ to $p_j$, $U\text{-}PRes(C)$ is defined as $\{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_{i-1}, p_{j+1}, \ldots, p_p\}$.*

Based on the previous properties, a new tractable class called U-Horn SAT is extracted. Algorithm 1 describes how this class can be recognized.

**Algorithm 1**: isU-Horn

**Input**: a SAT instance $\Sigma$
**Output**: $true$ if $\Sigma$ is U-Horn; $false$ otherwise

1 **begin**
2 $\Sigma' \leftarrow \{C | C \in \Sigma$ s.t. isHorn$(C)\}$;
3 **forall** $C \in \Sigma$ *s.t.* $C = \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_p\}$,
4 *with* $n \geq 0$ *and* $p > 1$ **do**
5 **if** $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n\}$ **then**
6 $\Sigma' \leftarrow \Sigma' \cup \{\{\neg n_1, \ldots, \neg n_n\}\}$;
7 **else**
8 $\Sigma'' \leftarrow \emptyset$ ; $C' \leftarrow C$;
9 **forall** $p_i \in C$ **do**
10 **if** $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, p_i\}$ **then**
11 **if** $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, \neg p_i\}$ **then**
12 $\Sigma' \leftarrow \Sigma' \cup \{\{\neg n_1, \ldots, \neg n_n\}\}$ ;
13 **else**
14 $\Sigma'' \leftarrow \Sigma'' \cup \{\{\neg n_1, \ldots, \neg n_n, p_i\}\}$;
15 **else if** $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, \neg p_i\}$ **then**
16 $C' \leftarrow C' \setminus \{p_i\}$;
17 **if** $\{\neg n_1, \ldots, \neg n_n\} \not\subset \Sigma'$ **then**
18 **if** $\Sigma'' = \emptyset$ **then** $\Sigma' \leftarrow \Sigma' \cup \{C'\}$;
19 **else** $\Sigma' \leftarrow \Sigma' \cup \Sigma''$;
20 $\Sigma' \leftarrow redundancyUP(\Sigma')$;
21 **return** isHorn$(\Sigma')$;
22 **end**

After all Horn clauses have been recorded in $\Sigma'$, all remaining clauses $C$ are tested successively (line 3). According to Property 3, when the negative part of $C$ is UP-derivable from $\Sigma$, this negative part is considered as an additional Horn clause and recorded in $\Sigma'$ (lines 5 and 6). Else, the second part of Property 3 is implemented in lines 10 to 12. The tests of lines 10 and 15 translate Property 4. In order to obtain $U\text{-}PRes(C)$, the tests in lines 17 to 19 allow the insertion within $\Sigma'$ of the smallest clause (w.r.t. its number of positive literals). In line 20, a call is made to a procedure described in [13] to get rid of redundant clauses modulo PU. Finally, the initial formula $\Sigma$ is U-Horn if and only if the simplified formula $\Sigma'$ is Horn.

## 4 Experimental results

In order to assess the practical interest of this pretreatment, a variant of Algorithm 1 (without line 16) has been implemented and experimented. Due to computational reasons, it does not manipulate two CNF ($\Sigma$ and $\Sigma'$) as shown in Algorithm 1 but makes use of a same CNF $\Sigma$. Consequently, the reduced CNF depends on the order according to which the clauses are considered, and running the program once does not guarantee that all possible simplifications are made, whereas Algorithm 1 ensures this last point. To perform all possible simplifications, the program must be iterated until no new Horn clause is produced. The program has been run on various benchmarks from the DIMACS depository [9] and from the last SAT competitions (`www.satcompetition.org`). All experimentations have been conducted on an Intel(R) Xeon(TM) CPU 3.00GHz with 2Go of memory under Linux CentOS release 4.1. Interestingly enough, some instances were reduced to polynomial-time ones, running the program just once. In Table 1, all instances belonging to the U-Horn class are given: 99 instances belonging to U-Horn class have been found within (almost) 1600 tested instances. For each instance, its name, its size (#var. and #cla.), the number of propagations (#UP) and the time spent in seconds to reduce the instance to U-Horn are given. When the program is iterated until no new Horn clause is produced, 28 additional

| CNF instances | # var. | # cla. | #UP | time (s.) | CNF instances | # var. | # cla. | #UP | time (s.) |
|---|---|---|---|---|---|---|---|---|---|
| **aim-100-1_6-yes1-4** | 100 | 160 | 179 | 0 | IBM_FV_2004_rule_batch. . . | | | | |
| **aim-100-2_0-yes1-2** | 100 | 200 | 456 | 0 | IBM_. . .04_SAT_dat.k15 | 15300 | 65598 | 397812 | 0.25 |
| **aim-100-6_0-yes1-1** | 100 | 600 | 2502 | 0 | IBM_. . .05_SAT_dat.k15 | 25128 | 134922 | 1708357 | 1.22 |
| **aim-100-6_0-yes1-2** | 100 | 600 | 2534 | 0 | IBM_. . .15_SAT_dat.k100 | 226970 | 893496 | 2432156 | 2.46 |
| **aim-100-6_0-yes1-3** | 100 | 600 | 777 | 0 | IBM_. . .15_SAT_dat.k15 | 30790 | 119911 | 184301 | 0.19 |
| **aim-100-6_0-yes1-4** | 100 | 600 | 568 | 0 | IBM_. . .15_SAT_dat.k20 | 42330 | 165416 | 252596 | 0.26 |
| **aim-200-6_0-yes1-2** | 200 | 1200 | 6113 | 0.01 | IBM_. . .15_SAT_dat.k25 | 53870 | 210921 | 329216 | 0.33 |
| **aim-200-6_0-yes1-4** | 200 | 1200 | 696 | 0 | IBM_. . .15_SAT_dat.k30 | 65410 | 256426 | 413391 | 0.42 |
| **aim-50-2_0-yes1-2** | 50 | 100 | 218 | 0 | IBM_. . .15_SAT_dat.k35 | 76950 | 301931 | 506031 | 0.5 |
| **aim-50-2_0-yes1-3** | 50 | 100 | 250 | 0 | IBM_. . .15_SAT_dat.k40 | 88490 | 347436 | 606086 | 0.6 |
| **aim-50-2_0-yes1-4** | 50 | 100 | 156 | 0 | IBM_. . .15_SAT_dat.k45 | 100030 | 392941 | 714746 | 0.71 |
| **aim-50-6_0-yes1-1** | 50 | 300 | 516 | 0 | IBM_. . .15_SAT_dat.k50 | 111570 | 438446 | 830681 | 0.83 |
| **aim-50-6_0-yes1-2** | 50 | 300 | 692 | 0 | IBM_. . .15_SAT_dat.k55 | 123110 | 483951 | 955361 | 0.99 |
| **aim-50-6_0-yes1-3** | 50 | 300 | 440 | 0 | IBM_. . .15_SAT_dat.k60 | 134650 | 529456 | 1087176 | 1.07 |
| **aim-50-6_0-yes1-4** | 50 | 300 | 1621 | 0 | IBM_. . .15_SAT_dat.k65 | 146190 | 574961 | 1227876 | 1.22 |
| **cnf-r1-b3-k1.2** | 660004 | 5281 | 56944 | 0.21 | IBM_. . .15_SAT_dat.k70 | 157730 | 620466 | 1375571 | 1.38 |
| **cnf-r1-b4-k1.1** | 397893 | 7089 | 105048 | 0.18 | IBM_. . .15_SAT_dat.k75 | 169270 | 665971 | 1532291 | 1.53 |
| **cnf-r1-b4-k1.2** | 922148 | 6818 | 60079 | 0.29 | IBM_. . .15_SAT_dat.k80 | 180810 | 711476 | 1695866 | 1.69 |
| **cnf-r2-b2-k1.2** | 406052 | 6064 | 54402 | 0.15 | IBM_. . .15_SAT_dat.k85 | 192350 | 756981 | 1868606 | 1.88 |
| **cnf-r2-b3-k1.2** | 668180 | 9169 | 100807 | 0.27 | IBM_. . .15_SAT_dat.k90 | 203890 | 802486 | 2048061 | 2.06 |
| **cnf-r2-b4-k1.1** | 406052 | 12784 | 178182 | 0.25 | IBM_. . .15_SAT_dat.k95 | 215430 | 847991 | 2236821 | 2.26 |
| **cnf-r2-b4-k1.2** | 930282 | 12464 | 175575 | 0.37 | IBM_. . .22_SAT_dat.k10 | 18919 | 77414 | 596987 | 0.4 |
| jnh10 | 100 | 850 | 6737 | 0.02 | IBM_. . .22_SAT_dat.k15 | 29833 | 122814 | 1249118 | 0.96 |
| jnh11 | 100 | 850 | 11187 | 0.02 | IBM_. . .22_SAT_dat.k20 | 40753 | 168249 | 1845706 | 1.48 |
| **jnh12** | 100 | 850 | 5323 | 0.01 | iso-brn005.shuffled | 1130 | 9866 | 13572 | 0.02 |
| jnh13 | 100 | 850 | 4940 | 0.01 | f19-b21-s0-0 | 746 | 3517 | 23805 | 0.03 |
| jnh14 | 100 | 850 | 3362 | 0.01 | f27-b10-s0-0 | 193 | 1113 | 8268 | 0.01 |
| jnh15 | 100 | 850 | 7544 | 0.01 | f27-b1-s0-0 | 193 | 1113 | 9401 | 0.01 |
| jnh18 | 100 | 850 | 16943 | 0.03 | f27-b2-s0-0 | 193 | 1113 | 5614 | 0.01 |
| jnh19 | 100 | 850 | 10836 | 0.02 | f27-b3-s0-0 | 193 | 1113 | 8716 | 0.01 |
| jnh202 | 100 | 800 | 4641 | 0.01 | f27-b4-s0-0 | 193 | 1113 | 5992 | 0.01 |
| jnh203 | 100 | 800 | 18563 | 0.03 | f27-b5-s0-0 | 193 | 1113 | 5626 | 0.01 |
| jnh208 | 100 | 800 | 16108 | 0.03 | f27-b8-s0-0 | 193 | 1113 | 7702 | 0.01 |
| jnh20 | 100 | 850 | 8478 | 0.02 | f27-b9-s0-0 | 193 | 1113 | 8684 | 0.01 |
| jnh211 | 100 | 800 | 3030 | 0.01 | f83-b11-s0-0 | 1000 | 43900 | 318968 | 0.74 |
| jnh214 | 100 | 800 | 12131 | 0.02 | f83-b14-s0-0 | 1000 | 43540 | 811348 | 1.61 |
| jnh215 | 100 | 800 | 10558 | 0.02 | f83-b17-s0-0 | 1000 | 43900 | 180456 | 0.37 |
| jnh216 | 100 | 800 | 12821 | 0.02 | par8-1-c | 64 | 254 | 5613 | 0 |
| jnh2 | 100 | 850 | 2201 | 0 | **par8-1** | 350 | 1149 | 9224 | 0 |
| jnh302 | 100 | 900 | 246 | 0 | **par8-2** | 350 | 1157 | 7641 | 0 |
| jnh303 | 100 | 900 | 13452 | 0.03 | **par8-4-c** | 67 | 266 | 6216 | 0 |
| jnh304 | 100 | 900 | 1720 | 0 | **par8-4** | 350 | 1155 | 10248 | 0.01 |
| jnh305 | 100 | 900 | 5348 | 0.01 | **par8-5** | 350 | 1171 | 7978 | 0 |
| jnh307 | 100 | 900 | 2211 | 0 | pitch.boehm | 1192 | 6361 | 656 | 0.01 |
| jnh308 | 100 | 900 | 15155 | 0.03 | qg5-10.shuffled | 1000 | 43900 | 318968 | 0.69 |
| jnh309 | 100 | 900 | 2460 | 0.01 | qg6- 10.shuffled | 1000 | 43540 | 811348 | 1.62 |
| jnh310 | 100 | 900 | 3054 | 0.01 | qg7-10.shuffled | 1000 | 43900 | 180456 | 0.37 |
| jnh4 | 100 | 850 | 5955 | 0.01 | 3col20_5_5.shuffled | 40 | 176 | 774 | 0 |
| jnh5 | 100 | 850 | 4151 | 0.01 | 3col20_5_6.shuffled | 40 | 176 | 656 | 0 |
| jnh8 | 100 | 850 | 4749 | 0.01 | 3col20_5_7.shuffled | 40 | 176 | 903 | 0 |
| jnh9 | 100 | 850 | 3099 | 0.01 | 3col20_5_9.shuffled | 40 | 176 | 438 | 0 |

**Table 1. U-Horn instances**

instances are reduced to U-Horn. They are given in Table 2 where "removed cla" (resp. "removed var") represents the ratio (in percents) of clauses (resp. variables) removed by the method and where "#lit" represents the total number of litterals that have been removed. Even when this pre-treatment does not conduct the instance to be reduced to a polynomial-time one, the global size of the instance is often decreased in a significant manner, whereas its polynomial subpart is increased accordingly. Interestingly, this reduction appears valuable from a global problem-solving point of view. In Table 3, the time required to solve instances using Minisat [12] with the time spent by a combination of the pre-treatment with Minisat are compared. In this table, the columns "Minisat" represent the time consumed by Minisat to solve the original instance ("original") and the simplified one ("simplified"); and the columns "%profit" represents the gain (in percents) obtained by the pre-treatment when the simplification time is taken into account either together with the satisfiability checking time ("total") or not ("partial").

## 5 Related works

The U-Horn SAT class exhibits a limited similarity with Dalal's Quad fragment [4]. Indeed, both approaches make use of a sub-clauses deduction procedure, using unit propagation inference rules. However, the approach in this paper differs from Dalal's one in several ways. First, it re-

| | instance size | | removed | | | | |
|---|---|---|---|---|---|---|---|
| CNF Instances | #var. | #cla. | cla | var | #lit. | #UP | time (s.) |
| een-tipb-sr06-par1 | 163647 | 484831 | 94% | 95% | 252004 | 68362283 | 38.98 |
| ezfact16_10.shuffled | 193 | 1113 | 26% | 34% | 335 | 5614 | 0.01 |
| ezfact16_3.shuffled | 193 | 1113 | 37% | 44% | 479 | 5992 | 0.01 |
| f32-b2-s0-0 | 40 | 176 | 70% | 69% | 178 | 941 | 0 |
| f32-b4-s0-0 | 40 | 176 | 85% | 77% | 163 | 919 | 0 |
| f33-b9-s0-0 | 80 | 346 | 88% | 80% | 391 | 5867 | 0 |
| f6-b2-s2-20 | 478 | 1007 | 95% | 92% | 532 | 14216 | 0 |
| IBM_FV_2004_rule_batch_03_SAT_dat.k30 | 29079 | 118925 | 44% | 55% | 31075 | 1393665 | 1.07 |
| IBM_FV_2004_rule_batch_05_SAT_dat.k10 | 15399 | 81447 | 87% | 93% | 33203 | 1252239 | 0.76 |
| IBM_FV_2004_rule_batch_05_SAT_dat.k20 | 34863 | 188452 | 74% | 82% | 72024 | 7798669 | 5.49 |
| IBM_FV_2004_rule_batch_05_SAT_dat.k25 | 44598 | 241982 | 67% | 75% | 86760 | 18851484 | 16.38 |
| IBM_FV_2004_rule_batch_05_SAT_dat.k30 | 54333 | 295512 | 60% | 67% | 99477 | 31503131 | 26.84 |
| IBM_FV_2004_rule_batch_06_SAT_dat.k15 | 17501 | 75616 | 43% | 49% | 18130 | 1278040 | 1.07 |
| IBM_FV_2004_rule_batch_06_SAT_dat.k20 | 23826 | 103226 | 71% | 78% | 41764 | 12961178 | 10.17 |
| IBM_FV_2004_rule_batch_10_SAT_dat.k15 | 40278 | 159501 | 33% | 35% | 26022 | 8285670 | 6.89 |
| IBM_FV_2004_rule_batch_1_11_SAT_dat.k10 | 28280 | 111519 | 47% | 49% | 25573 | 58410957 | 42.46 |
| IBM_FV_2004_rule_batch_18_SAT_dat.k10 | 17141 | 69989 | 48% | 55% | 19878 | 13050828 | 8.7 |
| IBM_FV_2004_rule_batch_19_SAT_dat.k10 | 21823 | 83902 | 24% | 31% | 13250 | 298260 | 0.26 |
| IBM_FV_2004_rule_batch_19_SAT_dat.k15 | 34697 | 134023 | 17% | 22% | 14917 | 508638 | 0.47 |
| IBM_FV_2004_rule_batch_19_SAT_dat.k20 | 47577 | 184178 | 17% | 23% | 23258 | 14607263 | 12.98 |
| IBM_FV_2004_rule_batch_20_SAT_dat.k10 | 17567 | 72087 | 36% | 41% | 14004 | 5226452 | 3.63 |
| IBM_FV_2004_rule_batch_21_SAT_dat.k10 | 15919 | 65180 | 35% | 39% | 11897 | 267966 | 0.21 |
| IBM_FV_2004_rule_batch_21_SAT_dat.k15 | 25213 | 103881 | 25% | 28% | 13564 | 471438 | 0.39 |
| IBM_FV_2004_rule_batch_21_SAT_dat.k20 | 34513 | 142616 | 26% | 30% | 21454 | 9624852 | 7.38 |
| IBM_FV_2004_rule_batch_22_SAT_dat.k25 | 51673 | 213684 | 24% | 27% | 28739 | 30219471 | 22.32 |
| IBM_FV_2004_rule_batch_23_SAT_dat.k10 | 18612 | 76086 | 41% | 48% | 16035 | 69713 | 0.09 |
| IBM_FV_2004_rule_batch_27_SAT_dat.k10 | 6477 | 27070 | 62% | 70% | 10054 | 3826810 | 2.15 |
| rip08.boehm | 471 | 263 | 92% | 59% | 145 | 8728 | 0.01 |
| x6dn.boehm | 521 | 1255 | 86% | 84% | 1022 | 137818 | 0.07 |

**Table 2. Reduction of SAT instances using several runs**

mains independent from the considered literals ordering. Secondly, a single polynomial fragment is considered instead of several ones in Dalal's work, which as a consequence does not deliver a linear-time pre-treatment. Finally, the use of other treatments based on the removal of redundant clauses [13] and of other reductions operations in our pre-treatment makes the two classes incomparable ones. Obviously enough, the idea of pre-treating SAT instances is not a new one. Many modern SAT solvers include some pre-treatment techniques. For instance, C-SAT [11] made a restricted use of resolution as a polynomial-time pre-treatment, and some DPLL algorithms start with local search runs that, when they fail to prove consistency, are exploited in the further complete search [17]. More recently, Satellite, which is the pre-treatment used in one of the state-of-the-art satisfiability solver, simplifies the instance using variable elimination [1]. Due to its linear-time character, the unit propagation algorithm has been exploited in several ways in the context of SAT, in addition to being a key component of DPLL-like procedures. For example, C-SAT and Satz used a local treatment during important steps of the exploration of the search space, based on UP, to derive implied literals and detect local inconsistencies, and guide the selection of the next variable to be assigned [11, 16]. In [15], a double UP schema is explored in the context of SAT solving. In [19, 14], UP has been used as an efficient tool to detect functional dependencies in SAT instances. The UP technique has also been exploited in [6] in order to derive subclauses by using the UP implication graph of the SAT instance, and speed up the resolution process.

# 6 Conclusions and perspectives

In this paper, a new linear-time pre-treatment technique for SAT instances has been introduced. It is based on the efficiency of the unit propagation algorithm, which is exploited in order to attempt to increase the polynomial subpart of the targeted SAT instances. Interestingly enough, benchmarks from the SAT competitions that were so far believed to be hard-to-solve problems have been proved to be polynomial SAT instances, and solved accordingly. As such, the pre-treatment is also valuable in that it often increases the efficiency of the satisfiability checking global process. We plan to extend this technique w.r.t. other polynomial fragments of SAT in the future.


# Acknowledgments

This research has been supported in part by the CNRS, the EC under a Feder grant and the *Région Nord/Pas-de-Calais*.


# References


[1] A. Biere and N. Eén. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of the*

| | Minisat | | % profit | | instance size | | removed | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CNF Instance | Original | Simplified | partial | total | #var | #cla | var | cla | #lit | #UP | time (s.) |
| f2clk_40 | 293.4 | 265.19 | 9 | 7 | 27568 | 80439 | 36% | 36% | 13619 | 5009951 | 5.52 |
| f3-b29-s0-10 | 76.72 | 26.58 | 65 | 65 | 2125 | 12677 | 24% | 35% | 3520 | 127851 | 0.18 |
| f28-b4-s0-0 | 3.15 | 0.04 | 98 | 96 | 769 | 4777 | 13% | 22% | 927 | 45554 | 0.08 |
| f81-b3-s0-0 | 2081.34 | 1654.61 | 20 | 17 | 33385 | 163232 | 26% | 30% | 24855 | 36519004 | 63.69 |
| fifo8_100 | 14.31 | 11.12 | 22 | -48 | 64762 | 176313 | 42% | 46% | 42718 | 8435460 | 9.98 |
| fifo8_200 | 43.74 | 77.5 | -78 | -134 | 129762 | 353513 | 37% | 40% | 76361 | 18309357 | 24.51 |
| fifo8_300 | 349.92 | 152.11 | 56 | 45 | 194762 | 530713 | 35% | 39% | 109878 | 28532439 | 39.94 |
| fifo8_400 | 500.73 | 428.59 | 14 | 3 | 259762 | 707913 | 34% | 38% | 143413 | 38349604 | 55.85 |
| IBM_03_SAT_dat.k60 | 28.33 | 11.99 | 57 | 17 | 59649 | 244535 | 22% | 27% | 33386 | 12915029 | 11.33 |
| IBM_03_SAT_dat.k90 | 195.45 | 173.44 | 11 | 1 | 90219 | 370145 | 17% | 20% | 38507 | 21481064 | 18.32 |
| IBM_05_SAT_dat.k100 | 204.54 | 28.02 | 86 | 21 | 190623 | 1044932 | 21% | 27% | 167316 | 143847825 | 133.17 |
| IBM_05_SAT_dat.k60 | 55.59 | 10.52 | 81 | -37 | 112743 | 616692 | 31% | 37% | 123076 | 73459545 | 65.46 |
| IBM_16_1_SAT_dat.k95 | 14.62 | 2.18 | 85 | 29 | 50492 | 203817 | 23% | 26% | 24509 | 9440470 | 8.16 |
| ip50 | 92.63 | 307.54 | -233 | -266 | 66131 | 214786 | 36% | 44% | 47569 | 23195373 | 30.61 |
| logistics-rotate-09t6 | 80.07 | 6.5 | 91 | -55 | 8186 | 887558 | 15% | 30% | 908 | 157186029 | 117.23 |

**Table 3. Some typical instances**


*Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, 2005.

[2] E. Boros, P. L. Hammer, and X. Sun. Recognition of q-horn formulae in linear time. *Discrete Applied Mathematics*, 55(1):1–13, 1994.

[3] M. Cadoli and F. M. Donini. A survey on knowledge compilation. *AI Communications-The European Journal for Artificial Intelligence*, 10(3-4):137–150, 1997.

[4] M. Dalal. An almost quadratic class of satisfiability problems. In W. Wahlster, editor, *Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI'96)*, pages 355–359, Budapest (Hungary), 1996. John Wiley & Sons, Ltd.

[5] M. Dalal and D. W. Etherington. A hierarchy of tractable satisfiability problems. *Information Processing Letters*, 44(4):173–180, 1992.

[6] S. Darras, G. Dequen, L. Devendeville, B. Mazure, R. Ostrowski, and L. Saïs. Using Boolean constraint propagation for sub-clauses deduction. In *Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP'05)*, pages 757–761, 2005.

[7] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal Artificial Intelligence Research (JAIR)*, 17:229–264, 2002.

[8] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[9] Second challenge on satisfiability testing, 1993.

[10] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing satisfiability of propositional horn formulae. *Journal of Logic Programming*, pages 267–284, 1984.

[11] O. Dubois, P. André, Y. Boufkhad, and Y. Carlier. *Second DIMACS implementation challenge: cliques, coloring and satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, chapter SAT vs. UNSAT, pages 415–436. American Mathematical Society, 1996.

[12] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518, 2003.

[13] O. Fourdrinoy, É. Grégoire, B. Mazure, and L. Saïs. Eliminating redundant clauses in sat instances. In *Proceedings of the The Fourth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'07)*, LNCS 4510, pages 71–83, 2007.

[14] É. Grégoire, R. Ostrowski, B. Mazure, and L. Saïs. Automatic extraction of functional dependencies. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, LNCS 3542, pages 122–132, 2004.

[15] D. Le Berre. Exploiting the real power of unit propagation lookahead. In *Proceedings of the Fourth International Conference on Theory and Applications of Satisfiability Testing (SAT'01)*, Boston University, Massachusetts, USA, June 2001.

[16] C. M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 366–371, 1997.

[17] B. Mazure, L. Saïs, and É. Grégoire. Boosting complete techniques thanks to local search. *Annals of Mathematics and Artificial Intelligence*, 22:309–322, 1998.

[18] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 530–535, 2001.

[19] R. Ostrowski, B. Mazure, L. Saïs, and É. Grégoire. Eliminating redundancies in SAT search trees. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'2003)*, pages 100–104, Sacramento, November 2003.

[20] B. Selman and H. A. Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, 1996.

[21] B. Selman, H. J. Levesque, and D. G. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 440–446, 1992.