Learning for Dynamic Subsumption

Youssef Hamadi[1] Saïd Jabbour[2]
Lakhdar Saïs[2]
[1] Microsoft Research
7 J J Thomson Avenue
Cambridge, United Kingdom
youssefh@microsoft.com
[2] Université Lille-Nord de France
CRIL - CNRS UMR 8188
Artois, F-62307 Lens
{jabbour,sais}@cril.fr

Abstract

This paper presents an original dynamic subsumption technique for Boolean CNF formulae. It exploits simple and sufficient conditions to detect, during conflict analysis, clauses from the formula that can be reduced by subsumption. During the learnt clause derivation, and at each step of the associated resolution process, checks for backward subsumption between the current resolvent and clauses from the original formula are efficiently performed. The resulting method allows the dynamic removal of literals from the original clauses. Experimental results show that the integration of our dynamic subsumption technique within the state-of-the-art SAT solvers Minisat and Rsat particularly benefits to crafted problems.

1. Introduction

The SAT problem, i.e., the problem of checking whether a set of Boolean clauses is satisfiable or not, is central to many domains in computer science and artificial intelligence including constraint satisfaction problems (CSP), planning, non-monotonic reasoning, VLSI correctness checking, etc. Today, SAT has gained a considerable audience with the advent of a new generation of SAT solvers able to solve large instances encoding real-world applications and the demonstration that these solvers represent important low-level building blocks for many important fields, e.g., SMT solving, Theorem proving, Model finding, QBF solving, etc. These solvers, called modern SAT solvers [13, 9], are based on classical unit propagation [7] efficiently combined through incremental data structures with: (i) restart policies [10, 11], (ii) activity-based variable selection heuristics (VSIDS-like) [13], and (iii) clause learning [12, 2, 13]. Modern SAT solvers can be seen as an extended version of the well known DPLL-like procedure obtained thanks to these different enhancements. It is important to note that the well known resolution rule still plays a strong role in the efficiency of modern SAT solvers which can be understood as a particular form of general resolution [3].

Indeed, conflict-based learning, one of the most important component of SAT solvers is based on resolution. We can also mention, that the well known and highly successful (`SatElite`) preprocessor is based on variable elimination through the resolution rule [16, 4]. As mentioned in [16], on industrial instances, resolution leads to the generation of many tautological resolvents. This can be explained by the fact that many clauses represent Boolean functions encoded through a common set of variables. This property of the encodings might also be at the origin of many redundant or subsumed clauses at different steps of the search process.

The utility of (`SatElite`) on industrial problems has been proved, and therefore one can wonder if the application of the resolution rule could be performed not only as a pre-processing stage but systematically during the search process. Unfortunately, dynamically maintaining a formula closed under subsumption might be time consuming. An attempt has been made recently in this direction by L. Zhang [17]. In this work, a novel algorithm maintains a subsumption-free clause database by dynamically detecting and removing subsumed clauses as they are added. Interestingly, the author mention the following perspective of research: "How to balance the runtime cost and the quality of the result for on-the-fly CNF simplification is a very interesting problem worth much further investigation".

In this paper, our objective is to design an effective dynamic simplification algorithm based on resolution. Our proposed approach aims at eliminating literals from the CNF formula by dynamically substituting smaller clauses. More precisely, our approach exploits the intermediate steps of classical conflict analysis to subsume the clauses of the formula which are used in the underlying resolution derivation of the asserting clause. Since original clauses or learnt

clauses can be used during conflict analysis both categories can be simplified. The effectiveness of our technique lies in the efficiency of the subsumption test, which is based on a simple and sufficient condition computable in constant time. Moreover, since our technique relies on the derivation of a conflict-clause, it is guided by the conflicts, and simplifies parts of the formula identified as important by the search strategy (VSIDS guidance). This dynamic process preserves the satisfiability of the formula, and with some additional bookkeeping can preserve the equivalence of the models.

The paper is organized as follows. After some preliminary definitions and notations, classical implication graph and learning schemes are presented in section 2. Then our dynamic subsumption approach is described in section 3. Finally, before the conclusion, experimental results demonstrating the performances of our approach are presented.

2. Technical background

2.1. Preliminary definitions and notations

A *CNF formula* $\mathcal{F}$ is a conjunction of *clauses*, where a clause is a disjunction of *literals*. A literal is a positive (x) or negated ($\neg x$) propositional variable. The two literals x and $\neg x$ are called *complementary*. We note by $\bar{l}$ the complementary literal of l. For a set of literals L, $\bar{L}$ is defined as $\{\bar{l} \mid l \in L\}$. A *unit* clause is a clause containing only one literal (called *unit literal*), while a binary clause contains exactly two literals. An *empty clause*, noted $\bot$, is interpreted as false (unsatisfiable), whereas an *empty CNF formula*, noted $\top$, is interpreted as true (satisfiable).

The set of variables occurring in $\mathcal{F}$ is noted $V_{\mathcal{F}}$. A set of literals is *complete* if it contains one literal for each variable in $V_{\mathcal{F}}$, and *fundamental* if it does not contain complementary literals. An *assignment* ρ of a Boolean formula $\mathcal{F}$ is function which associates a value $\rho(x) \in \{false, true\}$ to some of the variables $x \in \mathcal{F}$. ρ is *complete* if it assigns a value to every $x \in \mathcal{F}$, and *partial* otherwise. An assignment is alternatively represented by a fundamental set of literals, in the obvious way. A *model* of a formula $\mathcal{F}$ is an assignment ρ that makes the formula $true$; noted $\rho \models \Sigma$.

The following notations will be heavily used throughout the paper:

- $\eta[x, c_i, c_j]$ denotes the *resolvent* between a clause c_i containing the literal x and c_j a clause containing the opposite literal $\neg x$. In other words $\eta[x, c_i, c_j] = c_i \cup c_j \backslash \{x, \neg x\}$. A resolvent is called *tautological* when it contains opposite literals.
- $\mathcal{F}|_x$ will denote the formula obtained from $\mathcal{F}$ by assigning x the truth-value *true*. Formally $\mathcal{F}|_x = \{c \mid c \in \mathcal{F}, \{x, \neg x\} \cap c = \emptyset\} \cup \{c \backslash \{\neg x\} \mid c \in \mathcal{F}, \neg x \in c\}$ (that is: the clauses containing x are removed; and those containing $\neg x$ are simplified). This notation is extended to assignments: given an assignment $\rho = \{x_1, \ldots, x_n\}$, we define $\mathcal{F}|_\rho = (\ldots((\mathcal{F}|_{x_1})|_{x_2})\ldots|_{x_n})$.
- $\mathcal{F}^*$ denotes the formula $\mathcal{F}$ closed under unit propagation, defined recursively as follows: (1) $\mathcal{F}^* = \mathcal{F}$ if $\mathcal{F}$ does not contain any unit clause, (2) $\mathcal{F}^* = \bot$ if $\mathcal{F}$ contains two unit-clauses $\{x\}$ and $\{\neg x\}$, (3) otherwise, $\mathcal{F}^* = (\mathcal{F}|_x)^*$ where x is the literal appearing in a unit clause of $\mathcal{F}$. A clause c is deduced by unit propagation from $\mathcal{F}$, noted $\mathcal{F} \models^* c$, if $(\mathcal{F}|_{\bar{c}})^* = \bot$.

Let c_1 and c_2 be two clauses of a formula $\mathcal{F}$. We say that c_1 (respectively c_2) subsume (respectively is subsumed) c_2 (respectively by c_1) iff $c_1 \subseteq c_2$. If c_1 subsume c_2, then $c_1 \models c_2$ (the converse is not true). Also $\mathcal{F}$ and $\mathcal{F} - c_2$ are equivalent with respect to satisfiability.

2.2. DPLL search

DPLL [7] is a tree-based backtrack search procedure; at each node of the search tree, the assigned literals (decision literal and the propagated ones) are labeled with the same *decision level* starting from 1 and increased at each decision (or branching). After backtracking, some variables are unassigned, and the current decision level is decreased accordingly. At level i, the current partial assignment ρ can be represented as a sequence of decision-propagation of the form $\langle (x_k^i), x_{k_1}^i, x_{k_2}^i, \ldots, x_{k_{n_k}}^i \rangle$ where the first literal x_k^i corresponds to the decision literal x_k assigned at level i and each $x_{k_j}^i$ for $1 \leq j \leq n_k$ represents unit propagated literals at level i. Let $x \in \rho$, we note $l(x)$ the assignment level of x. For a clause α, $l(\alpha)$ is defined as the maximum level of its assigned literals.

2.3. Conflict analysis using implication graphs

Implication graphs is a standard representation conveniently used to analyze conflicts in modern SAT solvers. Whenever a literal y is propagated, we keep a reference to the clause which triggers the propagation of y, which we note $imp(y)$. The clause $imp(y)$, called implication of y, is in this case of the form $(x_1 \vee \cdots \vee x_n \vee y)$ where every literal x_i is false under the current partial assignment ($\rho(x_i) = \textit{false}, \forall i \in 1..n$), while $\rho(y) = \textit{true}$. When a literal y is not obtained by propagation but comes from a decision, $imp(y)$ is undefined, which we note for convenience $imp(y) = \bot$. When $imp(y) \neq \bot$, we denote by $exp(y)$ the set $\{\overline{x} \mid x \in imp(y) \setminus \{y\}\}$, called set of *explanations* of y. When $imp(y)$ is undefined we define $exp(y)$ as the empty set.

Definition 1 (Implication Graph) *Let $\mathcal{F}$ be a CNF formula, ρ a partial assignment, and let exp denotes the set of explanations for the deduced (unit propagated) literals in ρ. The implication graph associated to $\mathcal{F}$, ρ and exp is $\mathcal{G}_{\mathcal{F}}^{\rho,exp} = (\mathcal{N}, \mathcal{E})$ where:*

- *$\mathcal{N} = \rho$, i.e., there is exactly one node for every literal, decided or implied;*
- *$\mathcal{E} = \{(x, y) \mid x \in \rho, y \in \rho, x \in exp(y)\}$*

In the rest of this paper, for simplicity reason, exp is omitted, and an implication graph is simply noted as $\mathcal{G}_{\mathcal{F}}^{\rho}$. We also note m as the conflict level.

Example 1 *$\mathcal{G}_{\mathcal{F}}^{\rho}$, shown in Figure 1 is an implication graph for the formula $\mathcal{F}$ and the partial assignment ρ given below : $\mathcal{F} \supseteq \{c_1, \ldots, c_{12}\}$*

$(c_1) \neg x_1 \vee \neg x_{11} \vee x_2$ $(c_2) \neg x_1 \vee x_3$
$(c_3) \neg x_2 \vee \neg x_{12} \vee x_4$ $(c_4) \neg x_1 \vee \vee \neg x_3 \vee x_5$
$(c_5) \neg x_4 \vee \neg x_5 \vee \neg x_6 \vee x_7$ $(c_6) \neg x_5 \vee \neg x_6 \vee x_8$
$(c_7) \neg x_7 \vee x_9$ $(c_8) \neg x_5 \vee \neg x_8 \vee \neg x_9$
$(c_9) \neg x_{10} \vee \neg x_{17} \vee x_1$ $(c_{10}) \neg x_{13} \vee \neg x_{14} \vee x_{10}$
$(c_{11}) \neg x_{13} \vee x_{17}$ $(c_{12}) \neg x_{15} \vee \neg x_{16} \vee x_{13}$

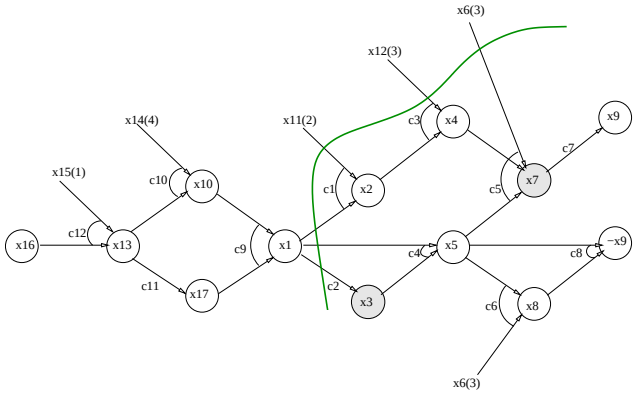

Figure 1. Implication Graph $\mathcal{G}_{\mathcal{F}}^{\rho} = (\mathcal{N}, \mathcal{E})$

$\rho = \{\langle \ldots x_{15}^1 \ldots \rangle \langle (x_{11}^2) \ldots \ldots \rangle \langle (x_{12}^3) \ldots x_6^3 \ldots \rangle$
$\langle (x_{14}^4), \ldots \rangle \langle (x_{16}^5), x_{13}^5, \ldots \rangle\}$. *The conflict level is 5 and $\rho(\mathcal{F}) = false$.*

Definition 2 (Asserting clause) *A clause c of the form $(\alpha \vee x)$ is called an asserting clause iff $\rho(c) = false$, $l(x) = m$ and $\forall y \in \alpha, l(y) < l(x)$. x is called asserting literal.*

Conflict analysis is the result of the application of resolution starting from the conflict clause using different implications implicitly encoded in the implication graph. We call this process an asserting clause derivation (in short ACD).

Definition 3 (Asserting clause derivation) *An asserting clause derivation π is a sequence of clauses $\langle \sigma_1, \sigma_2, \ldots \sigma_k \rangle$ satisfying the following conditions :*

1. *$\sigma_1 = \eta[x, imp(x), imp(\neg x)]$, where $\{x, \neg x\}$ is the conflict.*
2. *σ_i, for $i \in 2..k$, is built by selecting a literal $y \in \sigma_{i-1}$ for which $imp(\overline{y})$ is defined. We then have $y \in \sigma_{i-1}$ and $\overline{y} \in imp(\overline{y})$: the two clauses resolve. The clause σ_i is defined as $\eta[y, \sigma_{i-1}, imp(\overline{y})]$;*
3. *σ_k is, moreover an asserting clause.*

Note that every σ_i is a resolvent of the formula $\mathcal{F}$: by induction, σ_1 is the resolvent between two clauses that directly belong to $\mathcal{F}$; for every $i > 1$, σ_i is a resolvent between σ_{i-1} (which, by induction hypothesis, is a resolvent) and a clause of $\mathcal{F}$. Every σ_i is therefore also an *implicate* of $\mathcal{F}$, that is: $\mathcal{F} \models \sigma_i$. By definition of the implication graph, we also have $\mathcal{F}' \models^* \sigma_i$ where $\mathcal{F}' \subset \mathcal{F}$ is the set of clauses used to derive σ_i. Indeed, we have $(\mathcal{F}'|_{\bar{\sigma}_i})^* = \bot$.

Let us consider again the example 1. The traversal of the graph $\mathcal{G}_{\mathcal{F}}^{\rho}$ (see Fig. 1) leads to the following asserting clause derivation: $\langle \sigma_1, \ldots, \sigma_7 \rangle$ where $\sigma_1 = \eta[x_9, c_7, c_8] = (\neg {x_5}^5 \vee \neg {x_7}^5 \vee \neg {x_8}^5)$ and $\sigma_7 = (\neg {x_{11}}^2 \vee \neg {x_{12}}^3 \vee \neg {x_6}^3 \vee \neg {x_1}^5)$. The clause σ_7 is the first encountered resolvent that contains only one literal from the current decision level. Let us note that this resolvent is false under the interpretation ρ. Consequently, the literal $\neg x_1$ is implied at level 3. SAT solvers add such a clause (σ_7) to the learnt database, backjump to level 3 and assign the asserting literal $\neg x_1$ to true. The node x_1 corresponding to the asserting literal $\neg x_1$ is called the first Unique Implication Point (First UIP). For more details about CDCL based learning schemes, we refer the reader to [12, 13, 1].

3. Learning for dynamic subsumption

In this section, we show how classical learning can be adapted for an efficient dynamic subsumption of clauses. In our approach, we exploit the intermediate steps or resolvents generated during the classical conflict analysis to subsume some of the clauses used in the underlying resolution derivation of the asserting clause. Obviously, it would be possible to consider the subsumption test between each generated resolvent and the whole set of clauses. However, this could be very costly in practice. Let us, illustrate some of the main features of our proposed approach.

3.1. Motivating example

Let us reconsider again the example 1 and the implication graph $\mathcal{G}_{\mathcal{F}}^{\rho}$ (figure 1). The asserting clause derivation leading to the asserting clause Δ_1 is described as follows:
$\pi = \langle \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_7 = \Delta_1 \rangle$

- $\sigma_1 = \eta[x_9, c_7, c_8] = (\neg {x_8}^5 \vee \neg {x_7}^5 \vee \neg {x_5}^5)$
- $\sigma_2 = \eta[x_8, \sigma_1, c_6] = (\neg {x_6}^3 \vee \neg {x_7}^5 \vee \neg {x_5}^5)$
- $\sigma_3 = \eta[x_7, \sigma_2, c_5] = (\neg {x_6}^3 \vee \neg {x_5}^5 \vee \neg {x_4}^5) \subset c_5$ (subsumption)
- ...
- $\Delta_1 = \sigma_7 = \eta[x_2, \sigma_6, c_1] = (\neg {x_{11}}^2 \vee \neg {x_{12}}^3 \vee \neg {x_6}^3 \vee \neg {x_1}^5)$

As we can see the asserting clause derivation π includes the resolvent $\sigma_3 = (\neg {x_6}^3 \vee \neg {x_5}^5 \vee \neg {x_4}^5)$ which subsumes the clause $c_5 = (\neg x_6 \vee \neg x_5 \vee \neg x_4 \vee x_7)$. Consequently, the literal x_7 is eliminated from the clause c_5. In general, the resolvent σ_3 can subsume other clauses from the implication graph that include the literals $\neg x_6$, $\neg x_5$ and $\neg x_4$.

3.2. Dynamic subsumption: a general formulation

Let us now give a formal presentation of our dynamic subsumption approach.

Definition 4 (F-subsumption modulo UP) *Let $c \in \mathcal{F}$. c is $\mathcal{F}$-subsumed modulo Unit Propagation iff $\exists c' \subset c$ such that $\mathcal{F}|_{\bar{c'}} \models^* \perp$*

Given two clauses c_1 and c_2 from $\mathcal{F}$ such that c_1 subsumes c_2, then c_2 is $\mathcal{F}$-subsumed modulo UP.

As explained before, subsuming clauses during search might be time consuming. In our proposed framework, to reduce the computational cost, we restrict subsumption checks to the intermediate resolvents σ_i and the clauses of the form $imp(y)$ used to derive them (clauses encoded in the implication graph).

Definition 5 *Let $\mathcal{F}$ be a formula and $\pi = \langle \sigma_1 \dots \sigma_k \rangle$ an asserting clause derivation. For each $\sigma_i \in \pi$, we define $\mathcal{C}_{\sigma_i} = \{imp(y) \in \mathcal{F} | \exists j \leq i$ st. $\sigma_j = \eta[y, imp(y), \sigma_{j-1}]\}$ as the set of clauses of $\mathcal{F}$ used for the derivation of σ_i.*

Property 1 *Let $\mathcal{F}$ be a formula and $\pi = \langle \sigma_1, \sigma_2, \dots, \sigma_i, \dots, \sigma_k \rangle$ an asserting clause derivation. If σ_i subsumes a clause c of $\mathcal{C}_{\sigma_k}$ then $c \in \mathcal{C}_{\sigma_i}$.*

Proof : As $\sigma_{i+1} = \eta[y, imp(y), \sigma_i]$ where $\neg y \in \sigma_i$, we have $\sigma_i \not\subset imp(y)$. The next resolution steps can not involve clauses containing the literal $\neg y$. Otherwise, the literal y in the implication graph will admit more than one possible explanation, which is not possible by definition of the implication graph. Consequently, σ_i can not subsume clauses from $\mathcal{C}_{\sigma_k} - \mathcal{C}_{\sigma_i}$.

Property 2 *Let $\mathcal{F}$ be a formula and π an asserting clause derivation. If $\sigma_i \in \pi$ subsumes a clause c of $\mathcal{C}_{\sigma_i}$ then c is $\mathcal{C}_{\sigma_i}$-subsumed modulo UP.*

Proof : As $\sigma_i \in \pi$ is derived from $\mathcal{C}_{\sigma_i}$ by resolution, then $\mathcal{C}_{\sigma_i} \models \sigma_i$. By definition of an asserting clause derivation and implication graphs, we also have $\mathcal{C}_{\sigma_i} \models^* \sigma_i$ (see section 2.3). As σ_i subsumes c ($\sigma_i \subseteq c$), then $\mathcal{C}_{\sigma_i} \models^* c$.

The Property 2 shows that if a clause c encoded in the implication graph is subsumed by σ_i, such subsumption can be captured by subsumption modulo UP, while the Property 1 mention that subsumption checks of σ_i can be restricted to clauses from $\mathcal{C}_{\sigma_i}$. Consequently, a possible general dynamic subsumption approach can be stated as follows: Let $\pi = \langle \sigma_1, \dots, \sigma_i, \dots, \sigma_k \rangle$ be an asserting resolution derivation. For each resolvent $\sigma_i \in \pi$, we apply subsumption checks between σ_i and all the clauses in $\mathcal{C}_{\sigma_i}$.

In the following, we show that we can reduce further the number of clauses to be checked for subsumption by considering only a subset of $\mathcal{C}_{\sigma_i}$. Obviously, as σ_i is a resolvent of an asserting clause derivation π, then there exists two paths from the conflict nodes x and $\neg x$ respectively, to one or more nodes of the implication graph associated to the literals of σ_i assigned at the conflict level. Consequently, we derive the following property:

Property 3 *Let π be an asserting clause derivation, $\sigma_i \in \pi$ and $c \in \mathcal{C}_{\sigma_i}$. If σ_i subsumes c, then there exists two paths from the conflict nodes x and $\neg x$ respectively, to one or more nodes of the implication graph associated to the literals of c assigned at the conflict level.*

The proof of the property is immediate since $\sigma_i \subset c$. As this property is true for σ_i which is derived by resolution from the two clauses involving x and $\neg x$. Then it is also, true for its supersets (c).

For a given σ_i, the Property 3 leads us to another restriction of the set of clauses to be checked for subsumption. Indeed, we only need to consider the set of clauses $\mathcal{P}_{\sigma_i}$, linked (by paths) to the two conflicting literals x and $\neg x$.

We illustrate this characterization using the example 1 (see. also Figure 1). Let $\pi = \langle \sigma_1, \dots, \sigma_7 \rangle$ where $\sigma_7 = (\neg x_{11} \vee \neg x_{12} \vee \neg x_6 \vee \neg x_1)$. We have $\mathcal{C}_{\sigma_7} = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8\}$ and $\mathcal{P}_{\sigma_7} = \{c_1, c_2, c_4, c_5, c_6, c_8\}$. Indeed, from the nodes associated to the clause c_3 we only have one path to the node x_9. Consequently, the clause c_3 might be discarded from the set of clauses to be checked for subsumption. Similarly, the clause c_7 is only linked to the node x_9. Then c_7 is not considered for subsumption tests.

Property 4 *Given an asserting clause derivation $\pi = \langle \sigma_1, \dots, \sigma_k \rangle$. The time complexity of our general dynamic subsumption approach is in $O(|\mathcal{C}_{\sigma_k}|^2)$.*

Proof : From the definition of $\mathcal{C}_{\sigma_i}$, we have $|\mathcal{C}_{\sigma_i}| = i + 1$. In the worst case, we need to consider $i + 1$ subsumption

checks. Then for all σ_i with $1 \leq i \leq k$, we have to check $\sum_{1 \leq i \leq k}(i+1) = \frac{k \times (k+3)}{2}$. As $k = |\mathcal{C}_{\sigma_k}|$, then the worst case complexity is in $O(|\mathcal{C}_{\sigma_k}|^2)$.

The worst case complexity is quadratic even if we consider $\mathcal{P}_{\sigma_k} \subset \mathcal{C}_{\sigma_k}$.

3.3. Dynamic subsumption on the fly

In section 3.2, we have presented the general approach for dynamic subsumption. Its complexity is quadratic in the number of clauses used in the derivation of an asserting clause. As stated, in the introduction, to design an efficient dynamic simplification technique, one need to balance the run time cost and the quality of the simplification. In this section, we propose a restriction of the general dynamic subsumption scheme, called dynamic susbumption on the fly, which applies subsumption only between the current resolvent σ_i and the last clause from the implication graph used for its derivation. More precisely, suppose $\sigma_i = \eta[y, c, \sigma_{i-1}]$, we only check subsumption between σ_i and c.

The following property gives a sufficient condition under which y can be removed form c

Property 5 *Let π be an asserting clause derivation, $\sigma_i \in \pi$ such that $\sigma_i = \eta[y, c, \sigma_{i-1}]$. If $\sigma_{i-1} - \{y\} \subseteq c$, then c is subsumed by σ_i.*

Proof : Let $c = (\neg y \vee \alpha)$ and $\sigma_{i-1} = (y \vee \beta)$. Then $\sigma_i = (\alpha \vee \beta)$. As $\sigma_{i-1} - \{y\} \subseteq c$, then $\beta \subseteq \alpha$. So, $\sigma_i = \alpha$ which subsumes $(\neg y \vee \alpha) = c$.

Considering modern SAT solvers that include conflict analysis, the integration of this new dynamic subsumption approach can be done with negligible additional cost. Indeed, by using a simple counter during the conflict analysis procedure, we can verify the sufficient condition given in the Property 5 with a constant complexity. Indeed, at each step of the asserting clause derivation, we generate the next resolvent σ_i from a clause c and a resolvent σ_{i-1}. In the classical implementation of conflict analysis, one can check in constant time if a given literal is present in the current resolvent. Consequently, during the visit of the clause c, we additionally compute the number n of literals of c that belong to σ_{i-1}. If $n \geq |\sigma_{i-1}| - 1$ then c is subsumed by $\sigma_i = \eta[y, c, \sigma_{i-1}]$.

4. Experiments

The experiments were done on a large panel of crafted and industrial problems coming from the last competitions. All the instances were simplified by the `SatElite` preprocessor [8]. We implemented our dynamic subsumption approach in Minisat [9] and Rsat [15] and made a comparison between the original solvers and the ones enhanced with dynamic subsumption. All the tests were made on a Xeon 3.2GHz (2 GB RAM) cluster. Results are reported in seconds.

4.1. Crafted problems

During these experiments, the CPU time limit was fixed to 3 hours. These problems are hand made and many of them are designed to beat all the existing DPLL solvers. They contain for example Quasi-group instances, forced random SAT instances, counting, ordering and pebbling instances, social golfer problems, etc.

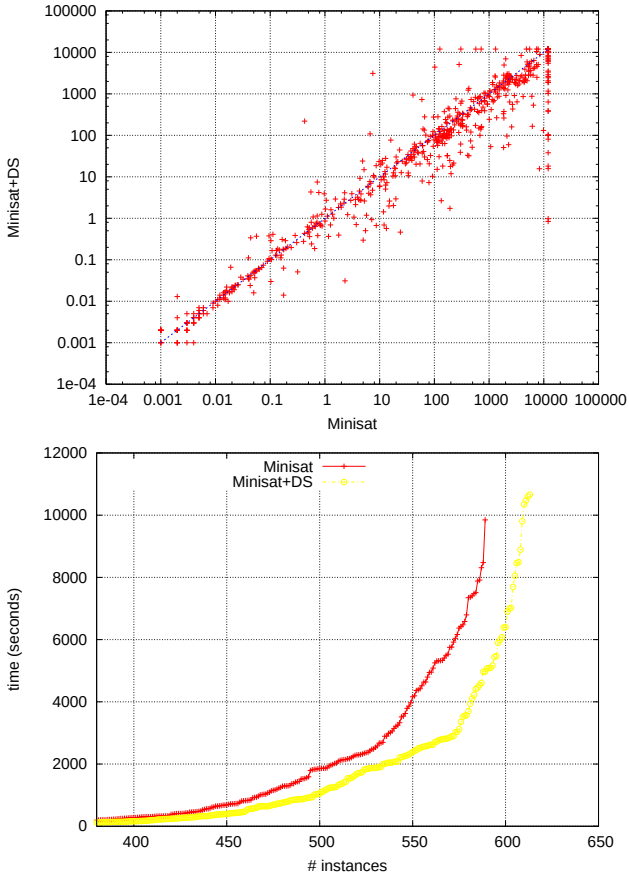

Figure 2. Crafted problems: `Minisat` vs `Minisat+DS`

The log-scaled scatter plot (in log scale) given in the left part of Figure 2 details the results for `Minisat` and `Minisat+DS` on each crafted instance. The x-axis (resp. y-axis) corresponds to the CPU time tx (resp. ty) obtained by `Minisat` (resp. `Minisat+DS`). Each dot with (tx, ty) coordinates, corresponds to a SAT instance. Dots below

(resp. above) the diagonal indicate instances where the subsumption is more efficient i.e. $ty < tx$. This figure clearly shows the computational gain obtained thanks to our efficient dynamic subsumption approach. By automatically counting the points we found that 365 instances are solved more efficiently through dynamic subsumption. In some cases the gain is up to 1 order of magnitude. Of course, there exists instances where subsumption decreases the performances of `Minisat` (178 instances).

The right part of the Figure 2 shows the same results with a different representation which gives for each technique the number of solved instances (# instances) in less than t seconds. This Figure confirms the efficiency of our dynamic subsumption approach on these problems. On several classes the number of removed literals is very important e.g., `x_*`, `QG_*`, `php_*`, `parity_*`. On the `genurq_*`, `mod_*`, and `urquhart_*` the problem is simplified during each conflict analysis.

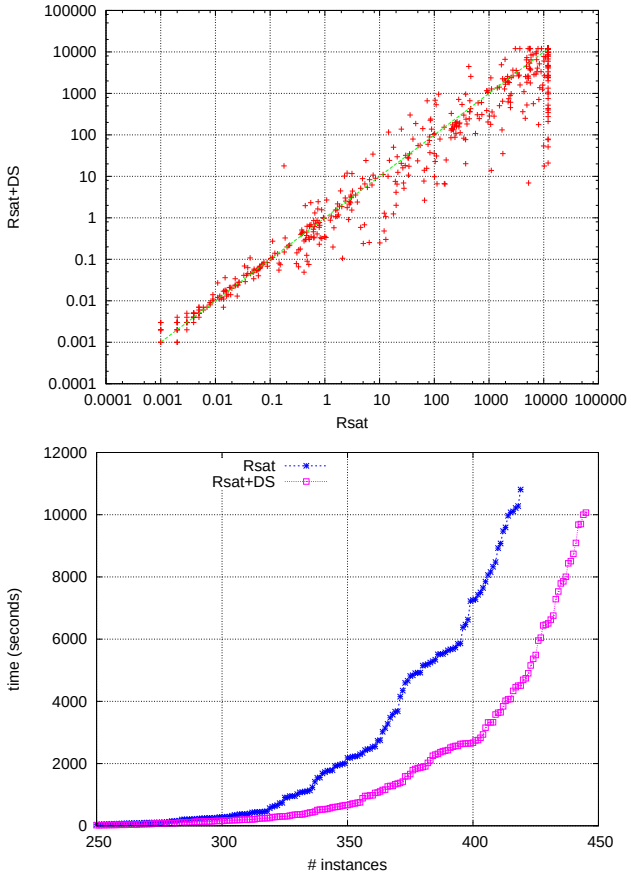

Figure 3. Crafted problems: `Rsat` vs `Rsat+DS`

Figure 3 shows results for `Rsat` and `Rsat+DS`. Overall we can see that the addition of our dynamic subsumption process to `Rsat` improves the performance. The fine analysis of the left part of Figure 3 showed that `Rsat+DS` solves 327 instances more efficiently than `Rsat`, which solves 219 problems more efficiently than its opponent.

Interestingly we can remark that the performances of `Rsat` and `Rsat+DS` is worse than the ones of `Minisat` and `Minisat+DS`. This comes from the rapid restart strategy used by this algorithm which does not pay off on crafted problems.

4.2. Industrial problems

With these problems, the time limit was set to 3 hours. Table 1 provides detailed results on the SAT industrial problems from the Sat-competition 2007 and Sat-Race 2008. The first column represents the instances families. The second column (#inst.) indicates the total number of instances in each family. Following columns present results for respectively `Rsat`, `Rsat+DS`, `Minisat`, and `Minisat+DS`. In each of these columns the first number represents the number of instances solved, and the number in parenthesis represents the number of instances solved more quickly than the opponent. The last row of the table gives the total of each column. We can see that `Rsat+DS` and `Minisat+DS` are in general faster and solves more problems than `Rsat` and `Minisat` respectively.

families	# inst.	`Rsat`	`Rsat` + `DS`	`Minisat`	`Minisat` + `DS`
aloul_*	1	–	–	–	**1(1)**
IBM_*	53	15(7)	**17(10)**	15(**10**)	15(7)
APro_*	16	12(**7**)	12(5)	**14**(6)	13(**8**)
mizh_*	10	10(**7**)	10(3)	10(5)	10(5)
partial_*	20	6(2)	**7(5)**	1(0)	**2(2)**
total_*	20	13(6)	13(**8**)	**10**(5)	9(**6**)
dated_*	20	15(6)	**16(10)**	**14(10)**	13(4)
braun_*	7	4(1)	4(**3**)	5(2)	5(**3**)
velev_*	10	2 (0)	2(**2**)	**2(2)**	1(0)
sort_*	5	2(**2**)	2(0)	2(0)	2(**2**)
manol_*	10	8(3)	8(**5**)	8(**5**)	**9**(4)
vmpc_*	10	9(1)	9(**8**)	6(2)	**7(5)**
clause_*	5	3(**2**)	3(1)	3(0)	3(**3**)
cube_*	4	4(2)	4(2)	4(1)	4(**3**)
gold_*	4	2 (2)	2 (**0**)	2(0)	2(**2**)
safe_*	4	2(0)	2(**2**)	1(0)	1(**1**)
simon_*	5	5 (**4**)	5 (1)	5(**3**)	5(2)
block_*	2	2(**2**)	2(0)	2(0)	2(**2**)
dspam_*	10	10(5)	10(5)	10(5)	10(5)
schup_*	3	3(2)	3(1)	3(**2**)	3(1)
post_*	10	8(3)	8(**5**)	5(3)	**6**(3)
ibm_*	20	20(6)	20(**14**)	19(6)	19(**13**)
Total	249	155(70)	**159(90)**	141(67)	142(**82**)

Table 1. Industrial problems

Table 2, focuses on some industrial families. In these families, the speed-ups are relatively important. For in-

instance	Rsat	Rsat + DS	Minisat	Minisat + DS
vmpc_24	43	**8**	**82**	210
vmpc_25	39	**1**	830	**318**
vmpc_26	182	**69**	1239	**1235**
vmpc_27	593	**327**	1159	**637**
vmpc_28	**173**	488	**3859**	5478
vmpc_29	2598	**1302**	–	**1252**
vmpc_30	366	**105**	3111	**2039**
vmpc_33	5540	**1562**	–	–
vmpc_31	–	–	–	–
vmpc_34	3366	**944**	–	–
partial-5-11-s	931	**176**	–	**2498**
partial-5-13-s	503	**71**	3248.38	**669**
partial-5-15-s	**737.064**	825	–	–
partial-5-19-s	1134	**498**	–	–
partial-5-17-s	**7437.82**	10610	–	–
partial-10-11-s	1242	**875**	–	–
partial-10-13-s	–	**3237**	–	–
ibm-02-04r-k80	90	**33**	**113**	152
ibm-02-11r1-k45	67	**29**	102	**65**
ibm-02-18r-k90	265	**157**	1044	**769**
ibm-02-20r-k75	**36**	185	2112	**668**
ibm-02-22r-k60	738	**691**	5480	**3434**
ibm-02-22r-k75	363	**349**	1109	**688**
ibm-02-22r-k80	**285**	298	894	**642**
ibm-02-23r-k90	1477	**965**	7127	**2670**
ibm-02-24r3-k100	273	**256**	**133**	249
ibm-02-25r-k10	**3104**	3118	**2877**	3172
ibm-02-29r-k75	353	**248**	**272**	1107
ibm-02-30r-k85	3853	**592**	–	–
ibm-02-31_1r3-k30	1203	**652**	1150	**998**
ibm-04-01-k90	114	**30**	**251**	726
ibm-04-1_11-k80	394	**222**	559	**329**
ibm-04-23-k100	**326**	687	3444	**2743**
ibm-04-23-k80	**465**	563	2060	**1584**
ibm-04-29-k25	290	**210**	1061	**1017**
ibm-04-29-k55	533	**16**	558	**124**
ibm-04-3_02_3-k95	**1**	2	**1**	2

Table 2. Zoom on industrial families

stance, if we consider the vmpc family, we can see that our dynamic simplification allows a one order of magnitude improvement with Rsat+DS (instances 24, and 25). On the same family, on the 9 solved instances by Rsat+DS and Rsat, Rsat+DS is better on 8 instances. While Minisat+DS is better than Minisat on 5 instances among the 7 solved instances.

families	avg #subsumed clauses	avg. # conflicts
genurq_*	89669	131089
marg_*	279524	1.2e+06
bevhcube3_*	43855	159525
urqh_*	1.70612e+06	7.7e+06
hcb3_*	706560	2855021
icosahedron_*	257070	1.1e+06
mod_*	3.97111e+06	2.2e+07
pebbling	6994	88106
SGI_*	8418	1.4e+06
counting	89246	7.3e+06
QG _*	4861	1.1e+06
ezfact_*	7573	4e+06
clus_*	2040	417749

Table 3. Statistics on crafted families

In table 3 and 4, using Minisat+DS we give some statistics on the application of our learning for subsumption approach on crafted and industrial SAT families respectively. On each family, the average number of subsumed clauses (avg. #subsumed clauses) and the average number of conflicts are given (avg. #conflicts). Clearly, these statistics show that the average number of subsumed clauses is more important on crafted instances than on industrial ones. This is consistent with the experimental results presented previously. Even if the average number of subsumed clauses is less significant on industrial families (except on aloul_*), we still observe interesting improvements in term of cpu time. Note that similar statistics are observed using Rsat+DS.

families	avg. #subsumed clauses	avg. #conflicts
aloul_*	2863024	20242749
IBM_*	4038	1015876
AProVE_*	4388	1.8e+06
mizh_*	11473	851772
partial_*	423.5	302606
total_*	266	173394
dated_*	504	389092
braun_*	15482	4.8e+06
velev_*	230	126327
sortnet_*	5006	914436
manol_*	23857	1.7e+06
vmpc_*	2288	733818
clause_*	271	45112
cube_*	795	278352
gold_*	5336.5	4820315
safe_*	4650	229703
simon_*	3131	728076
blocks_*	226	23453
dspam_*	3990	767798
schup_*	584	286181
post_*	3706	952420
ibm_*	3114	605552

Table 4. Statistics on industrial families

Overall, our experiments allow us to demonstrate two things. First our technique does not degrade and often improves the performance of DPLLs on industrial problems. Second, it enhances the applicability of these algorithms on classes of problems which are made to be challenging for them. Since the implementing of our algorithm is rather simple, we think that overall, it represents an interesting contribution for the robustness of modern DPLLs.

5. Related Works

In Darras et al. [6], the authors proposed a preprocessing based on unit propagation for sub-clauses deduction. Considering the implication graph generated by the constraint propagation process as a resolution tree, the proposed approach deduces sub-clauses from the original formula. However, their proposed dynamic version is clearly time consuming. The experimental evaluation is only given in term of number of nodes.

In [17], an algorithm for maintaining a subsumption-free

CNF clause database is presented. It efficiently detects and removes subsumption when a learnt clause is added. Additionally, the algorithm compacts the database greedily by recursively applying resolutions in order to decrement the size of the database.

Conflict-clause shrinking was introduced in Jerusat [14]. It is also implemented in PicoSAT [5]. It removes literals from learnt clauses by resolving recursively with clauses of the implication graph. Remark that in the previous experiments, our base solvers `Minisat` and `Rsat` implement this technique.

6. Conclusion

This paper presents a new subsumption technique for Boolean CNF formulae. It makes an original use of learning to reduce original or learnt clauses. At each conflict, and during the asserting clause derivation process, subsumption between the generated resolvents and some clauses encoded in the implication graph is checked using an efficient sufficient condition. Interestingly, since our subsumption technique relies on the clauses used in the derivation of an asserting clause, it tends to simplify parts of the formula identified as important by the activity-based search strategy.

Experimental results show that the integration of our method within two state-of-the-art SAT solvers `Minisat` and `Rsat` particularly benefits to crafted problems and achieves interesting improvements on several industrial families.

As a future work, we plan to investigate how to efficiently extend our approach to achieve exhaustive clauses subsumption. Another interesting path of research would be to exploit our subsumption framework to fine tune the activity based strategy. Indeed, each time a literal is eliminated, this mean that a new conflict clause is derived and all the resolvents used in such derivation are useless and can be dropped from the implication graph.

References

[1] G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, and L. Sais. Generalized framework for conflict analysis. In *Proceedings of the eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, pages 21–27, 2008.

[2] Roberto J. Bayardo, Jr. and Robert C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 203–208, 1997.

[3] Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Understanding the power of clause learning. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1194–1201, 2003.

[4] A Biere and N. En. Effective preprocessing in sat through variable and clause elimination. In *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, pages 61–75, 2005.

[5] Armin Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT'08)*, 4(1):75–97, 2008.

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

[8] N. En and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, pages 61–75, 2005.

[9] Niklas En and Niklas Srensson. An extensible sat-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518, 2002.

[10] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 431–437, 1998.

[11] H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, and B. Selman. Dynamic restart policies. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI'02)*, pages 674–682, 2002.

[12] Joao P. Marques-Silva and Karem A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, 1996.

[13] 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.

[14] Alexander Nadel. Backtrack search algorithms for propositional logic satisfiability : Review and innovations. Master's thesis, Master Thesis, the Hebrew University, 2002.

[15] Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, pages 294–299, 2007.

[16] Sathiamoorthy Subbarayan and Dhiraj K. Pradhan. NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, pages 276–291, 2004.

[17] Lintao Zhang. On subsumption removal and on-the-fly cnf simplification. In *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, pages 482–489, 2005.