

Learning for dynamic assignments reordering

Saïd Jabbour

Microsoft Research-INRIA joint centre
28, rue Jean Rostand
91893 Orsay, France
said.jabbour@inria.fr

Abstract

In this paper a new learning scheme for SAT is proposed. The originality of our approach arises from its ability to achieve clause learning even if no conflict occurs. This clearly contrasts with all the traditional learning approaches which generally refer to conflict analysis. To make such learning possible, relevant clauses, taken from the satisfied part of the formula are conjointly used with the classical implication graph to derive new and more powerful reasons for the implication of a given literal. Based on this extension a first learning scheme called Learning for Dynamic Assignments Reordering (LDAR) is proposed. It exploits the new derived reasons to dynamically reorder partial assignments. Experimental results show that the integration of LDAR within a state-of-the-art SAT solver achieves interesting improvements particularly on satisfiable instances.

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 [9, 4], are based on classical unit propagation [2] efficiently combined through incremental data structures with: (i) restart policies [6, 7], (ii) activity-based variable selection heuristics (VSIDS-like) [9], and (iii) clause learning [8, 9].

Modern SAT solvers are especially efficient with structured SAT instances coming from industrial applications. On these problems, Gomez et al. [5] have identified a heavy tailed phenomenon, i.e., different variable orderings often lead to dramatic differences in solving time. This explains the introduction of restart policies in modern SAT solvers, which attempt to discover a good variable ordering. VSIDS and other variants of activity-based heuristics, on the other hand, were introduced to avoid thrashing and to focus the search: when dealing with instances of large size, these heuristics direct the search to the most constrained parts of the formula. Restarts and VSIDS play complementary roles since they implement the two principles of, respectively, diversification and intensification. Conflict Driven Clause Learning (CDCL) is the third component, leading to non-chronological backtracking. In CDCL a central data-structure is the *implication graph*, which records the partial assignment that is under construction together with its implications. Each time a dead end is encountered a conflict clause or nogood is learnt thanks to a bottom-up traversal of the implication graph. This traversal is also used to update the activity of related variables, allowing VSIDS to always select the most active variable as the new decision point.

In this paper we deal with clause learning in modern SAT solvers. Our goal is to show that similarly to real life, one can learn not only from failures (or conflicts) but also from successful branching decisions. As clause learning usually refers to conflict analysis, our proposed framework contrasts with all the learning-based approaches proposed in SAT. In classical learning, the main goal is to compute the reason of the current conflict and then to achieve non chronological backtracking. In contrast, in our approach the main goal is to deduce new and more powerful reasons for the implication of a given literal and then to exploit such reasons to reorder (or correct) the current partial assignment.

Usually, at a given node of the search tree, when a decision literal is assigned and unit propagation is applied,

the recorded reason for an implication of a unit literal x includes at least one literal from the current decision level i . Such reasons are recorded in the implication graph. Our proposed approach aims at discovering new reasons for the implication of x , including only literals from previous levels ($j < i$). In other words, our approach is able to discover that a given literal assigned at level i , could have been deduced at some earlier step j of the search process - assuming a consistency level stronger than unit propagation. Practically, these new reasons when recorded in the implication graph might improve the conflict analysis itself e.g., by improving backjumping levels.

Contrary to the classical conflict analysis, where the asserting literal is assigned at the backjumping level to the opposite value (literal refutation), in our proposed framework the new derived reasons are used to reorder the current partial assignment by assigning some literals at earlier levels with their actual truth values. By correcting the implication level of some literals, our proposed approach tends to push the bad choices to the bottom of the tree. This process improves the quality of the current partial assignment and favors the findings of solutions.

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 approach is described in section 3. In section 4, a first integration of LDAR in modern SAT solvers is presented. Finally, before the conclusion, experimental results demonstrating the performances of our approach are presented.

2. 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 $\perp$, 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 complete and fundamental set of literals, in the obvious way. A *model* of a for-

mula $\mathcal{F}$ is an assignment ρ that makes the formula *true*; noted $\rho \models \mathcal{F}$.

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 $\eta[x, c_i, c_j] = c_i \cup c_j \setminus \{x, \neg x\}$. 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 $\mathcal{F}|_x = \{c \mid c \in \mathcal{F}, \{x, \neg x\} \cap c = \emptyset\} \cup \{c \setminus \{\neg x\} \mid c \in \mathcal{F}, \neg x \in c\}$ (that is: the clauses containing x and are therefore satisfied are removed; and those containing $\neg x$ are simplified). This notation is extended to assignments: given an assignment $\rho = \{x_1, \dots, x_n\}$, we define $\mathcal{F}|_{\rho} = (\dots((\mathcal{F}|_{x_1})|_{x_2}) \dots |_{x_n})$.
- $\mathcal{F}^*$ 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) $\mathcal{F}^* = \perp$ 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 $(\mathcal{F}|_{\bar{c}})^* = \perp$.

2.2. DPLL search

Let us now introduce some notations and terminology on SAT solvers based on the Davis, Putnam, Logemann and Loveland procedure, commonly called DPLL [2]. DPLL 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). The current decision level is the highest decision level in the assignment stack (i.e., which represents the current branch of the search tree). 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 $\langle (x_k^i), x_{k_1}^i, x_{k_2}^i, \dots, x_{k_{n_k}}^i \rangle$ 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 a propagated (unit) literals at level i . Let $x \in \rho$, we note $l(x)$ the assignment level of x , $d(\rho, i) = x$ if x is the decision literal assigned at level i . For a clause α , $l(\alpha)$ is defined as the maximum level of its assigned literals. For a given level i , we define ρ^i as the projection of ρ to literals assigned at a level $\leq i$.

2.3. Conflict analysis using implication graphs

Implication graphs capture the variable assignments ρ made during the search, both by branching and by propagation. This standard representation is a convenient way to analyze conflicts. In classical SAT solvers, whenever a literal y is propagated, we keep a reference to the clause which triggers the propagation of y , which we note $\text{imp}(y)$. The clause $\text{imp}(y)$, called implication of y , is in this case of the form $(x_1 \vee \dots \vee x_n \vee y)$ where every literal x_i is false under the current partial assignment ($\rho(x_i) = \text{false}, \forall i \in 1..n$), while $\rho(y) = \text{true}$. When a literal y is not obtained by propagation but comes from a decision, $\text{imp}(y)$ is undefined, which we note for convenience $\text{imp}(y) = \perp$.

When $\text{imp}(y) \neq \perp$, we denote by $\text{exp}(y)$ the set $\{\bar{x} \mid x \in \text{imp}(y) \setminus \{y\}\}$, called set of *explanations* of y . In other words if $\text{imp}(y) = (x_1 \vee \dots \vee x_n \vee y)$, then the explanations are the literals $\bar{x}_i$ with which $\text{imp}(y)$ becomes the unit clause $\{y\}$. Note that for all $i \in 1..n$ we have $l(\bar{x}_i) \leq l(y)$ and there exists $i \in 1..n$ such that $l(\bar{x}_i) = l(y)$, i.e., all the explanations of the deduction come from a level at most as high. When $\text{imp}(y)$ is undefined we define $\text{exp}(y)$ as the empty set. The explanations can alternatively be seen as an implication graph, in which the set of predecessors of a node corresponds to the set of explanations of the corresponding literal:

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

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

For the sake of simplicity exp is omitted in the remainder of this paper, and an implication graph is simply noted as $\mathcal{G}_{\mathcal{F}}^{\rho}$.

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: $\mathcal{F} \supseteq \{c_1, \dots, c_9\}$

$$\begin{array}{ll} (c_1)x_6 \vee \neg x_{11} \vee \neg x_{12} & (c_2)\neg x_{11} \vee x_{13} \vee x_{16} \\ (c_3)x_{12} \vee \neg x_{16} \vee \neg x_2 & (c_4)\neg x_4 \vee x_2 \vee \neg x_{10} \\ (c_5)\neg x_8 \vee x_{10} \vee x_1 & (c_6)x_{10} \vee x_3 \\ (c_7)x_{10} \vee \neg x_5 & (c_8)x_{17} \vee \neg x_1 \vee \neg x_3 \vee x_5 \vee x_{18} \\ (c_9)\neg x_3 \vee \neg x_{19} \vee \neg x_{18} \end{array}$$

$\rho = \{\langle \dots \neg x_6^1 \dots \neg x_{17}^1 \rangle \langle (x_8^2) \dots \neg x_{13}^2 \dots \rangle \langle (x_4^3) \dots x_{19}^3 \dots \rangle \dots \langle (x_{11}^5) \dots \rangle\}$. The current decision level is 5 and $\rho(\mathcal{F}) = \text{false}$.

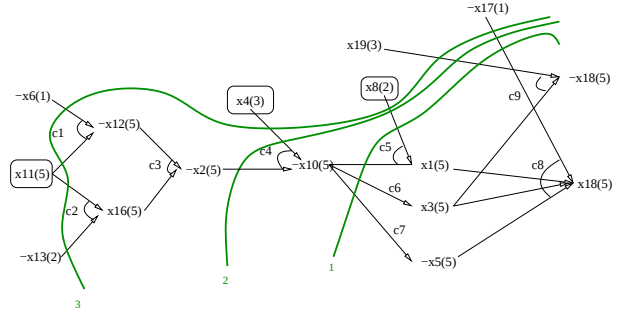


Figure 1. Implication Graph $\mathcal{G}_{\mathcal{F}}^{\rho} = (\mathcal{N}, \mathcal{E})$

Conflict analysis is the result of the application of resolution from the top (sources) to the bottom (sinks) of the implication graph using the different clauses of the form $(\text{exp}(y) \vee y)$ implicitly encoded at each node $y \in \mathcal{N}$. We call this process a conflict resolution proof. Using the example 1, we now illustrate the classical conflict driven clause learning scheme and its underlying concepts of asserting unit clauses and unique implication point. The traversal of the graph $\mathcal{G}_{\mathcal{F}}^{\rho}$ allows us to generate the following four clauses:

- $\sigma_1 = \eta[x_{18}, c_8, c_9] = (x_{17}^1 \vee \neg x_1^5 \vee \neg x_3^5 \vee x_5^5 \vee \neg x_{19}^3)$
- $\sigma_2 = \eta[x_1, \sigma_1, c_5] = (x_{17}^1 \vee \neg x_3^5 \vee x_5^5 \vee \neg x_{19}^3 \vee \neg x_8^2 \vee x_{10}^5)$
- $\sigma_3 = \eta[x_5, \sigma_2, c_7] = (x_{17}^1 \vee \neg x_3^5 \vee \neg x_{19}^3 \vee \neg x_8^2 \vee x_{10}^5)$
- $\sigma_4 = \Delta_1 = \eta[x_3, \sigma_3, c_6] = (x_{17}^1 \vee \neg x_{19}^3 \vee \neg x_8^2 \vee x_{10}^5)$

The clause Δ_1 is an *asserting unit clause* since all its literals are assigned *false* before the current level except one (x_{10}) which is assigned at the current level 5. From this first asserting unit clause, we deduce that one can safely back-jump to level 3 (the maximum level of the remaining literals in Δ_1) and assign the literal x_{10} (called asserting literal) to *true*. The node $\neg x_{10}$ in the graph (see cut 1 in Figure. 1) is called a first *Unique Implication Point (UIP)*. All the intermediate clauses σ_1 , σ_2 and σ_3 contain more than one literal of the current decision level 5. If we continue such a process, we obtain the two additional asserting unit clauses $\Delta_2 = (x_{17}^1 \vee \neg x_8^2 \vee \neg x_{19}^3 \vee \neg x_4^3 \vee x_2^5)$ and $\Delta_3 = (x_{17}^1 \vee x_6^1 \vee \neg x_8^2 \vee x_{13}^2 \vee \neg x_4^3 \vee \neg x_{19}^3 \vee \neg x_{11}^5)$ corresponding to the second $\neg x_2^5$ (cut 2 in Figure. 1) and third $\neg x_{11}^5$ (cut 3 in Figure. 1) UIP respectively. The node $\neg x_{11}$ is the last UIP since it corresponds to a decision literal (see cuts 2 and 3 in Fig. 1).

Let us note that for an asserting unit clause $c = (\alpha \vee x)$ deduced by conflict analysis at level m , one can deduce by unit propagation the literal x at decision level $i < m$ where $i = l(\alpha)$ i.e., $(\mathcal{F}|_{\rho^i}) \wedge \bar{x} \models_* \perp$. As x is assigned to false at the current decision level m , conflict analysis aims to cor-

rect this assignment by deriving a better explanation for the literal x .

3. Deriving new explanations

In this section, we show that new explanations for implied literals (unit propagated literals) can be generated even if a partial assignment does not lead to a conflict. These new capabilities offered by our proposed approach are motivated in the following example.

3.1. Motivating example

Example 2 Let $\mathcal{F}'$ the new formula obtained from $\mathcal{F}$ (see example 1) by removing the clause c_9 and adding the clause $c_{10} = (x_6 \vee \neg x_{10} \vee x_{18})$. The implication graph $\mathcal{G}_{\mathcal{F}'}$ is the a sub-graph of $\mathcal{G}_{\mathcal{F}}$ (see Figure 1) induced by the set of nodes $\mathcal{N} \setminus \{\neg x_{18}, x_{19}\}$. Clearly, the partial assignment ρ does not lead to a conflict at the decision level 5. We can remark, that the clause c_{10} is not recorded in the implication graph $\mathcal{G}_{\mathcal{F}'}$, as it contains two literals $\neg x_{10}$ and x_{18} assigned to true. Moreover, the literal x_{18} is assigned by unit propagation at level 5 due to the clause c_8 and ρ . Let us now illustrate how one can deduce a new reason (clause) asserting that x_{18} can be assigned at level less than 5. First starting from the clause $\text{imp}(x_{18})$, the following resolvents are generated :

- $\sigma'_1 = \eta[x_1, c_8, c_5] = (x_{17}^1 \vee \neg x_8^2 \vee x_{10}^5 \vee \neg x_3^5 \vee x_5^5 \vee x_{18}^5)$
- $\sigma'_2 = \eta[x_3, \sigma'_1, c_6] = (x_{17}^1 \vee \neg x_8^2 \vee x_{10}^5 \vee x_5^5 \vee x_{18}^5)$
- $\sigma'_3 = \Delta'_1 = \eta[x_5, \sigma'_2, c_7] = (x_{17}^1 \vee \neg x_8^2 \vee x_{10}^5 \vee x_{18}^5)$

The clause Δ'_1 contains only two literals x_{10} and x_{18} from the current decision level 5 and all its literals are assigned false except x_{18} which is assigned to true. Now, if we apply one additional resolution step between Δ'_1 and $c_{10} = (x_6 \vee \neg x_{10} \vee x_{18}) \in \mathcal{F}'$, we obtain a new asserting unit clause $\Delta''_1 = (x_6^1 \vee x_{17}^1 \vee \neg x_8^2 \vee x_{18}^5)$. This last clause expresses that the literal x_{18} assigned at level 5 can be assigned at level 2. Indeed, as $\mathcal{F}' \models \Delta''_1$ and the two literals x_6 and x_{17} (resp. the literal $\neg x_8$) are assigned false at level 1 (resp. level 2), we deduce that the literal x_{18} assigned at level 5 can be assigned by unit propagation at level 2 thanks to the new reason Δ''_1 .

From the above example and contrary to classical learning scheme, we can make the following preliminary observations:

- σ'_3 is not a conflict clause i.e., $\rho(\sigma'_3) = \text{true}$
- the literal x_{18} is not refuted i.e., the deduced clause Δ''_1 mentions that x_{18} must be assigned the same value true at level 2 instead of level 5

3.2. Formal presentation

Let us now give a formal presentation of our learning for new implication reasons approach. In all the following definitions and properties, we consider $\mathcal{F}$ a CNF formula, ρ a partial assignment such that $(\mathcal{F}|_\rho)^* \neq \perp$ and m the current decision level.

Definition 2 (Bi-Asserting clause) A clause $c = (\alpha \vee x \vee y)$ is called a Bi-Asserting binary clause iff $\rho(\alpha) = \text{false}$, $l(\alpha) < m$ and $l(x) = l(y) = m$.

A similar notion of bi-asserting clause is first defined in [11] by Knot Pipatsrisawat and Adnan Darwiche. There, a Bi-asserting clause is defined as a conflict-clause with two literals assigned at the conflict level, whereas in our Definition 2, we do not make any assumption on the value assignments of the two literals x and y . Following our Definition 2, the clause σ'_3 (example 2) satisfied by x_{18} is a Bi-Asserting clause. In summary, and unlike the work presented in [11] our Definition is not related to any conflict-clause generation. It can be derived by a Bi-Asserting clause resolution proof (Definition 3) at each node of the search tree, even if the implication graph is not a conflict graph.

Definition 3 (Bi-Asserting resolution proof) Let x be a literal such that $l(x) = m$ and $\text{imp}(x)$ is defined. A Bi-Asserting resolution proof $\pi_b(x)$ is a sequence of clauses $\langle \sigma_1, \sigma_2, \dots, \sigma_k \rangle$ satisfying the following conditions :

1. $\sigma_1 = \text{imp}(x)$
2. σ_i , for $i \in 2..k$, is built by selecting a literal $y \in \sigma_{i-1}$. The clause σ_i is then defined as $\eta[y, \sigma_{i-1}, \text{imp}(\bar{y})]$;
3. σ_k is moreover a Bi-Asserting clause.

Obviously, as the literal $x \in \sigma_1$ can not be eliminated by resolution, we have $x \in \sigma_k$. Indeed, if such elimination is possible, this means that both $\text{imp}(x)$ and $\text{imp}(\bar{x})$ are defined. This contradicts the hypothesis that $(\mathcal{F}|_\rho)^* \neq \perp$. Also, x is the unique literal of σ_k assigned to true.

In [1], we have shown that the asserting literal x generated using the classical conflict analysis can be deduced by unit propagation at level i where i is the backjumping level associated to the conflict (or asserting unit) clause $(\alpha \vee x)$ i.e. $\mathcal{F}|_{\rho^i} \models_* x$. Similarly, for an asserting binary clause $(\alpha \vee y \vee x)$, the binary clause $(y \vee x)$ (see Property 1 below) can also be deduced by unit propagation at level $i = l(\alpha)$.

Property 1 Let $\pi_b(x) = \langle \sigma_1, \sigma_2, \dots, \sigma_k \rangle$ be a Bi-Asserting resolution proof such that $\sigma_k = (\alpha \vee y \vee x)$ where $i = l(\alpha)$. Then we have $\mathcal{F}|_{\rho^i} \models_* (y \vee x)$

Proof : By definition of the Bi-Asserting resolution proof, $\pi_b(x)$ is derived by a traversal of the implication graph associated to $\mathcal{F}$ and ρ from the node x . It is easy

to see that the assignment of $\neg y$ at level i leads to the propagation of the literal x . Indeed, all the clauses used in the Bi-Asserting resolution proof $\pi_b(x)$ contain only literals from $\bar{\alpha}$ and literals propagated at the current decision level m . Consequently, we have $\mathcal{F}|_{\rho^i} \wedge \neg y \models^* x$. Then $\mathcal{F}|_{\rho^i} \wedge \neg y \wedge \neg x \models^* \perp$.

Let us illustrate the Property 1 using the example 2. From the Bi-Asserting clause $\sigma'_3 = (x_{17}^1 \vee \neg x_8^2 \vee x_{10}^5 \vee x_{18}^5)$, it is easy to show that the clause $(x_{10}^5 \vee x_{18}^5)$ can be deduced by unit propagation at level 2 i.e. $\mathcal{F}'|_{\rho^2} \models^* (x_{10} \vee x_{18})$. From the formula $\mathcal{F}'$ and ρ , we can see that the formula $\mathcal{F}'|_{\rho^2}$ contains the following clauses $\{(x_{10} \vee x_{18}), (x_{10} \vee x_3), (x_{10} \vee \neg x_5), (\neg x_1 \vee \neg x_3 \vee x_5 \vee x_{18})\}$ (see the implication graph $\mathcal{G}_{\mathcal{F}'|_{\rho^2}}^p$). Then, we have $\mathcal{F}'|_{\rho^2} \wedge \neg x_{10} \models^* x_{18}$. Consequently, $\mathcal{F}'|_{\rho^2} \wedge \neg x_{10} \wedge \neg x_{18} \models^* \perp$.

To derive a new reason for the implication of a given literal x . In the first step, we generate a Bi-Asserting resolution proof $\pi_b(x) = \langle \sigma_1, \sigma_2, \dots, \sigma_k = (\alpha \vee y \vee x) \rangle$, and in the second step we eliminate y from σ_k using resolution. Such resolution process, called asserting resolution proof, is defined as follows:

Definition 4 (Asserting resolution proof) Let $\pi_u(x)$ be a sequence of clauses $\langle \sigma_1, \dots, \sigma_{k-1}, \sigma_k \rangle$. $\pi_u(x)$ is an asserting resolution proof if it satisfies the following two conditions :

1. $\langle \sigma_1, \dots, \sigma_{k-1} = (\alpha \vee y \vee x) \rangle$ is a Bi-Asserting resolution proof;
2. $\exists c = (\beta \vee \neg y \vee x) \in \mathcal{F}$ a Bi-Asserting clause and $\sigma_k = \eta[y, \sigma_{k-1}, c] = (\gamma \vee x)$ where $\gamma = \alpha \cup \beta$.

It follows from the Definition 2 that all the literals of α and β are assigned to false before level m . Consequently, the resolvent σ_k is an asserting unit clause. This means that when $\pi_u(x)$ exists, we deduce that the literal x propagated at level m can be assigned to true at level $k < m$ where $k = l(\gamma)$. This result is stated in the following property:

Property 2 Let x be a literal such that $l(x) = m$ and $\text{imp}(x)$ is defined. If $\pi_u(x) = \langle \sigma_1, \dots, \sigma_{k-1}, \sigma_k = (\gamma \vee x) \rangle$ is an asserting resolution proof then $\mathcal{F}|_{\rho^i} \models^* x$ where $i = l(\gamma)$.

Proof : First, as $\langle \sigma_1, \dots, \sigma_{k-1} \rangle$ is a Bi-Asserting resolution proof, the resolvent σ_{k-1} is of the form $(\alpha \vee y \vee x)$ where $l(\alpha) \leq i$. Using the Property 1, we deduce that $\mathcal{F}|_{\rho^i} \models^* (y \vee x)$, then $\mathcal{F}|_{\rho^i} \wedge \neg x \models^* y$. Secondly, by definition of $\pi_u(x)$, the resolvent σ_k is obtained by resolution on y between σ_{k-1} and $c \in \mathcal{F}$ of the form $(\beta \vee \neg y \vee x)$ where $l(\beta) \leq i$. As $\forall z \in \beta \rho(z) = \text{false}$, then $(\neg y \vee x) \in \mathcal{F}|_{\rho^i}$ and $\mathcal{F}|_{\rho^i} \wedge \neg x \models^* \neg y$. Consequently, $\mathcal{F}|_{\rho^i} \wedge \neg x \models^* \perp$.

In the Property 2, we have shown that the literal x can be deduced by unit propagation at level i . However, checking if each unassigned literal can be deduced by unit prop-

Algorithm 1: LDAR solver

Input: A CNF formula $\mathcal{F}$;
Output: *true* if $\mathcal{F}$ is satisfiable; *false* otherwise

```

1 begin
2   currentLevel = 0,  $\Delta = \emptyset$ ;
3   while (true) do
4     if (propagate()  $\neq$  false) then
5       if (currentLevel  $\neq$  0) then return false;
6        $c = (\alpha \vee \neg d) = \text{analyze}()$ ;
7        $\Delta = \Delta \cup c$ ;
8       backJumpLevel =  $l(\alpha)$ ;
9       backtrack(backJumpLevel+1);
10      (newJumpLevel,  $\Delta'$ ) =
        learnForNewReasons(backJumpLevel+1);
11       $\Delta = \Delta \cup \Delta'$ ;
12      backtrack(newJumpLevel);
13    else
14      currentLevel++;
15      if (decide()  $\neq$  false) then
16        return true
17 end

```

agation at each node of the search tree might be time consuming. In the next section we show how some of these deductions can be obtained on the fly.

4. Learning for dynamic assignments re-ordering

In this section, we present how our learning for new reasons scheme (Asserting resolution proofs) can be used for dynamic assignments reordering in modern SAT solvers. The improved SAT solver, noted LDAR, is sketched in Algorithm 1. For simplicity reason, the algorithm does not mention the classical restart component which is an important part of modern solvers. The main functions involved in this algorithm are the following:

- *propagate*: achieves unit propagation and returns *false* in case of conflict, and *true* otherwise.
- *analyze*: returns the learnt clause c computed using classical conflict analysis (see section 2.3). The asserting literal is denoted d .
- *learnForNewReasons*: applies our learning for new reasons scheme presented in Algorithm 2.
- *backtrack*: backtracks to the level given in parameter.
- *decide*: selects and assigns the next decision literal. The function returns *true*, if such a literal exists, and *false* otherwise i.e. the formula is satisfied (all the literals are assigned).

As we have shown in the previous section, learning for new reasons can be applied at each node of the search tree. In other words, for each unit literal w propagated at a given level i , one can apply learning for new reasons to search for an implication (or asserting unit clause) of w with level $j < i$. However, in practice, a systematic application might

Algorithm 2: learnForNewReasons

Input: *currentLevel*: application level of learning for new reasons
Output: *newJumpLevel*: the highest implication level found;
R: a set of new reasons;

```
1 begin
2   y = decisionLiteral(currentLevel);
3   U = UPLiterals(currentLevel);
4   newJumpLevel = currentLevel - 1, R = ∅;
5   for (w ∈ U) do
6     if (∃c = (β ∨ y ∨ w) ∈ F s.t. ρ(β) = false,
7        ρ(w) = ρ(y) = true) then
8       Let πu(w) = ⟨σ1, ..., σk-1 = (α ∨ ¬y ∨ w), σk⟩
9       where σk = (γ ∨ w) = η[y, σk-1, c] s.t. γ = α ∪ β;
10      R = R ∪ σk;
11      newJumpLevel = min(l(γ), newJumpLevel);
12   return (newJumpLevel, R);
13 end
```

decrease the performance of the solver. In Algorithm 1, we apply learning for new reasons (line 10) only when a conflict occurs, i.e., if the call to *propagate()* returns *false*. More precisely, we first use the classical conflict analysis component (line 6). The generated conflict clause *c* is then added to the learnt database Δ (line 7). Then, the algorithm backtracks to *backJumpLevel* + 1 (line 9) and learning for new reasons is only applied at that particular level. By restricting such learning process to the literals (e.g. *w*) propagated at level *backJumpLevel* + 1, we guarantee that any new implication level *i* for *w* is less or equal to the current backjumping level. A set of new reasons Δ' and new level (called *newJumpLevel*) is returned by the function *learnForNewReasons* (line 10). The new backjumping level is computed by taking the best level (the smallest one) of the new reasons Δ' . Before backtracking to that new level (line 12), the set of new reasons Δ' is added to the learnt database (line 11). If the *learnForNewReasons* procedure does not find any new implications, *newJumpLevel* is equal to *backJumpLevel*, and the algorithm follows the classical CDCL learning scheme.

The function *learnForNewReasons* is detailed in Algorithm 2. First, let *y* (resp. *U*) be the decision literal (resp. the set of unit propagated literals) assigned (resp. propagated) at *currentLevel* set to *backJumpLevel* + 1 (line 2 and line 3). For each literal *w* ∈ *U*, if a Bi-Asserting clause of form $c = (\beta \vee y \vee w) \in \mathcal{F}$ s.t. $\rho(\beta) = \text{false}$, $\rho(x) = \rho(y) = \text{true}$ is found, then an asserting resolution proof $\pi_u(w) = \langle \sigma_1, \dots, \sigma_k \rangle$ is computed (see line 7). The new implication σ_k of *w* is added to the nogood database (line 8) and the *newJumpLevel* is updated (line 9). When all the literals propagated at the level *backJumpLevel* + 1 are processed, the best implication level (*newJumpLevel*) and the set of new reasons are returned (line 10), and Algorithm 1 can backtrack to this new level (line 12).

To reorder the current partial assignment, we make use of progress saving [10]. Indeed, during backjumping, all the decision literals between the *backJumpLevel* (computed

with classical learning) and the *newJumpLevel* (computed by *learnForNewReasons*) are stored. The function *decide()* selects them in priority in order to maintain approximately the same branch. The unit literals with new implication level are assigned at the right level (the new computed one) thanks to the recorded reasons (see line 12 in Algorithm 2). This process defines our Learning for Dynamic Assignments Reordering (LDAR) approach.

Figure 2 illustrates the differences between CDCL (left side) and LDAR (right side). Let $(\alpha \vee \neg d)$ be the asserting clause obtained by classical conflict analysis. In the CDCL approach the algorithm backjumps to *backJumpLevel* = *l*(α) and assigns the asserting literal $\neg d$. In the LDAR approach, the algorithm, first backjumps to *backJumpLevel* + 1 and applies learning for new reasons on all the literals propagated at that level (literal *w* in the Figure). All the new reasons for these literals are recorded in the nogood database. Assuming that the best backjumping level (*newJumpLevel*) corresponds to the new level of implication for *w*, the algorithm backjumps to *newJumpLevel* assigns the literal *w* and the other literals at the right level thanks to the recorded new reasons. Finally, search continues while assigning in priority the decisions literals recorded between *backJumpLevel* and *newJumpLevel* only.

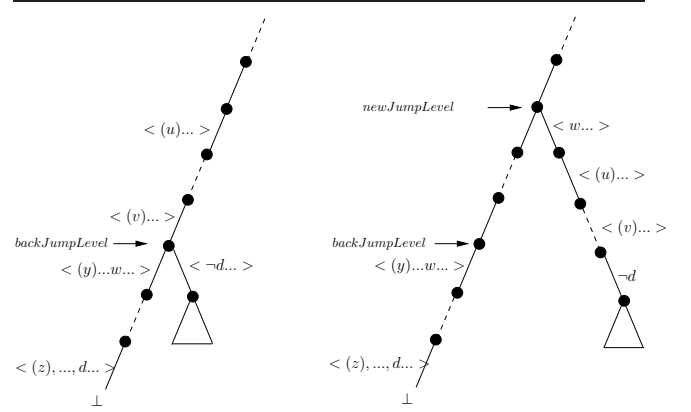


Figure 2. CDCL vs LDAR

5. Experiments

We used a large set of industrial problems coming from the SAT-Race 2006 and SAT-Competition 2007. We did not consider the latest SAT-Race (2008) which shares 80% of its problems with the 2007 SAT-Competition. All the instances were simplified by the Satellite preprocessor [3]. The CPU time limit was fixed to 1800 seconds and results are reported in seconds. We implemented our LDAR extension (section

4) in Minisat and made a comparison between the original Minisat solver and Minisat enhanced with LDAR. All the tests were made on a Xeon 3.2GHz (2 GB RAM) cluster.

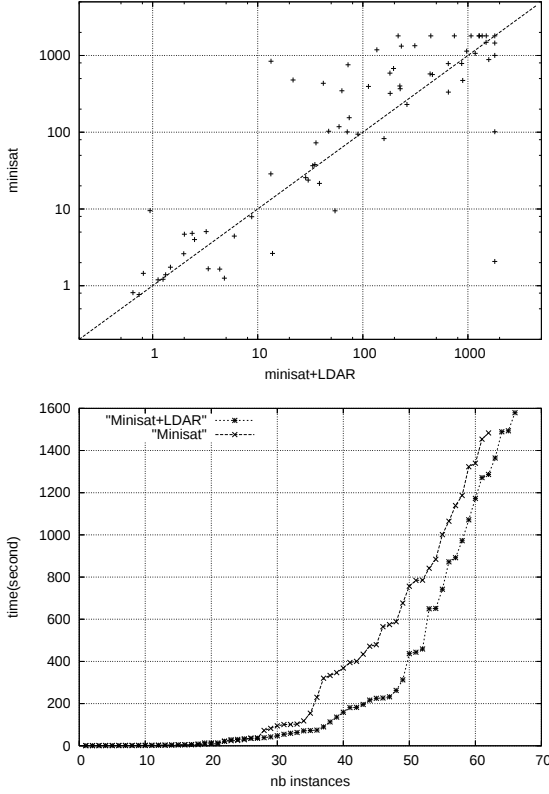


Figure 3. Satisfiable instances

The scatter plots (in log scale) given in Figures 3 and 4 illustrate the comparative results of Minisat and Minisat+LDAR on satisfiable and unsatisfiable instances respectively. The x-axis (resp. y-axis) corresponds to Minisat+LDAR (resp. Minisat). In these plots, the CPU times t_x and t_y obtained by the solvers on a particular instance represent a (t_x, t_y) dot.

As expected, Figure 3 shows that on satisfiable instances, learning for dynamic assignments reordering (LDAR) achieves interesting speedups with respect to Minisat. The majority of the dots in the figure are clearly above the diagonal i.e., Minisat+LDAR is better. This is not surprising since our proposed approach aims to correct the level (not the truth value) of an implied literal. It differs from classical conflict analysis, where we both correct the assignment level (backjumping) and the value assignment of the asserting literal. On the other hand, for unsatisfiable instances (see. Figure 4), the scatter plot does not allow a clear separation between the solvers.

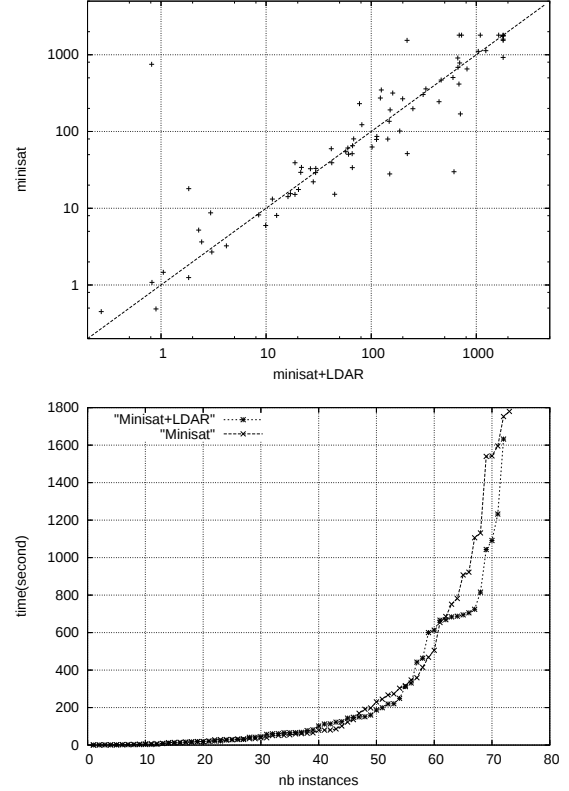


Figure 4. Unsatisfiable instances

Figure 3 and 4 (bottom side) show the *time* needed to solve a given number of instances. This global view confirms that our approach is better than Minisat on satisfiable instances in term of time needed to solve a given number of instances. Again, on unsatisfiable instances the difference in performance is less obvious but remains in favour of Minisat+LDAR which globally performs better.

Finally, Table 1 highlights the results obtained by Minisat+LDAR and Minisat, for each family of instances. For each family (first column), we report the number of instances (second column), the average running time on the solved instances and the number of solved instances (shown in parenthesis). The results are given on satisfiable (SAT), unsatisfiable (UNSAT) and on all the instances (SAT+UNSAT). For each family, best results in terms of the average runtime and number of solved instances are highlighted in bold font. As expected, Minisat+LDAR performs well on satisfiable families. For instance, on the *safe - ** family, Minisat+LDAR achieves on average a speed-up of 64.

In summary our first integration of LDAR in a modern SAT solver is very promising. We have shown that LDAR is complementary to CDCL since it is able to reuse its analysis of the implication graph to eventually build on it and

		SAT		UNSAT		SAT-UNSAT	
families	# inst.	Minisat+LDAR	Minisat	Minisat+LDAR	Minisat	Minisat+LDAR	Minisat
mizh.*	10	501(10)	720(10)	–	–	501(10)	720(10)
manol.*	20	–	–	336(17)	344(14)	336(17)	344(14)
partial.*	20	1272(1)	–	–	–	1271(1)	–
total.*	20	262(7)	94(4)	66(3)	66(3)	203(10)	82(7)
vmp_grieu.*	12	462(4)	801(4)	–	–	462(4)	801(4)
APro.*	16	13(1)	2(1)	323(7)	557(9)	284(8)	502(10)
dated.*	20	160(9)	151(9)	135(2)	661(3)	156(11)	279(12)
clause.*	4	233(4)	303(4)	–	–	233(4)	303(4)
cube.*	4	891(1)	472(1)	59(1)	60(1)	475(2)	266(2)
gold.*	7	–	–	597(4)	577(4)	597(4)	577(4)
safe.*	4	13(1)	841(1)	–	–	13(1)	841(1)
ibm.*	20	98(10)	121(10)	36(9)	102(9)	69(19)	112(19)
IBM.*	35	1295(4)	1113(3)	331(1)	359(1)	1102(5)	924(4)
simon.*	6	149(2)	277(2)	204(3)	128(3)	182(5)	188(5)
block.*	2	–	–	27(2)	27(2)	27(2)	27(2)
dspam.*	2	–	–	315(2)	34(2)	315(2)	34(2)
schup.*	2	71(1)	100(1)	666(1)	907(1)	368(2)	504(2)
sort.*	5	312(1)	1339(1)	1232(1)	1131(1)	772(2)	1235(2)
velev.*	12	–	2(1)	25(3)	–	25(3)	15(4)

Table 1. Highlighted results

achieve better performances on satisfiable instances while not degrading the performance on unsatisfiable ones.

6. Conclusion

In this paper a new learning scheme for SAT is proposed. Its originality arises from its ability to derive better implications for the unit propagated literals. This can be seen as an original way to correct the current partial assignment. Whereas, in most of the SAT solvers the satisfied part of the formula is clearly ignored, our approach suggests that such clauses usually connected to the remaining part of the formula can be advantageously exploited during search. Our first proposed integration of LDAR to one of the state-of-the-art SAT solver (Minisat) clearly shows interesting improvements on many classes of industrial problems. Interestingly, we have shown the relationships between the asserting unit clause resolution proof and the deduction based on unit propagation. As our approach aims to rearrange the current partial assignment, we plan to investigate the role of restarts policies in this context. Another interesting path for future research is to investigate how other strong local consistencies can be achieved using (extended) learning based approaches.

References

- [1] G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, and L. Saïs. Generalized framework for conflict analysis. In *proceedings of eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT'2008)*, 2008.
- [2] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.
- [3] N. Eén 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.
- [4] 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.
- [5] C. P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tail phenomena in satisfiability and constraint satisfaction. *Journal of Automated Reasoning*, pages 67 – 100, 2000.
- [6] 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.
- [7] 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.
- [8] 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.
- [9] M. W. Moskwicz, 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.
- [10] 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.
- [11] Knot Pipatsrisawat and Adnan Darwiche. A new clause learning scheme for efficient unsatisfiability proofs. In *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI'08)*, pages 1481–1484, 2008.