A Generalized Framework for Conflict Analysis

Gilles Audemard[1], Lucas Bordeaux[2], Youssef Hamadi[2], Said Jabbour[1], and Lakhdar Sais[1]

[1] CRIL-CNRS, Université d'Artois
rue Jean Souvraz SP18
F-62307 Lens Cedex France
{audemard,jabbour,sais}@cril.fr

[2] Microsoft Research
7 J J Thomson Avenue
Cambridge, United Kingdom
{lucasb,youssefh}@microsoft.com

Abstract. This paper makes several contributions to Conflict Driven Clauses Learning (CDCL), which is one of the key components of modern SAT solvers. First, we show that, given an implication graph, asserting clauses derived using the first Unique Implication Point principle (First UIP) are optimal in terms of back-jumping. Secondly we propose an extended notion of implication graph containing additional arcs, called inverse arcs. These are obtained by taking into account the satisfied clauses of the formula, which are usually ignored by conflict analysis. This extension captures more conveniently the whole propagation process, and opens new perspectives for CDCL-based approaches. Among other benefits, our extension of the implication graph leads to a new conflict analysis scheme that exploits the additional arcs to back-jump to higher levels. Experimental results show that the integration of our generalized conflict analysis scheme within state-of-the-art solvers consistently improves their performance.

1 Introduction

This paper makes a number of contributions to *Conflict-Driven Clause-Learning (CDCL)*, which is one of the key components of modern SAT solvers [8, 5]. In the CDCL approach a central data-structure is the *implication graph*, which records the partial assignment that is under construction together with its implications. This data-structure enables conflict analysis, which, in turn, is used for intelligent backtracking, for clause learning, and for adjusting the weights of the heuristic. An important observation is that the implication graph built in the traditional way is "incomplete" in that it only gives a partial view of the actual implications between literals. For instance, let us consider what happens when a literal y is deduced at a given level. Such an implication is always obtained because a clause, say $(\neg x_1 \vee \neg x_2 \vee y)$, has become unit under the current partial assignment. The partial interpretation therefore includes the assignments $x_1 = \mathit{true}$ and $x_2 = \mathit{true}$, which took place at levels smaller than or equal to the assignment level of y. When y is deduced the reasons of this deduction are recorded, i.e. we add to the implication graph edges between each of the reasons x_1, x_2 and the deduction y. Suppose now that another clause $(x_1 \vee \neg y)$ also belongs to the formula. This means that $x_1 = \mathit{true}$ is also a consequence of the assignment $y = \mathit{true}$. It would be logically correct to add an edge (y, x_1) to the graph, but this is not usually done, because in traditional implication graphs the solver only keeps track of the first explanation that is encountered for the deduced literal. This strategy is obviously very much dependent on the particular order

in which clauses are propagated. We propose to consider an extended notion of implication graph in which a deduced literal can have several explanations, in a sense that is made formal later in the paper. Contrary to the traditional case, some of these edges may go backwards with respect to the levels: such was the case in our example for the edge (y, x_1), where y was at a level higher than x_1. Such edges are called *inverse arcs*. In this paper, a strong extension of the classical Conflict Clause Driven Learning based scheme is proposed, based on the generalized notion of implication graph that we presented informally. First, we prove that given an implication graph generated according to a given partial assignment, the conflict clause generated using the first Unique Implication Point principle (First UIP) is *optimal* in terms of back-jumping level. Motivated by this optimality result, we formalize our notion of extended implication graph and study its use for conflict analysis. The "inverse arcs" present in our extended graph allow us to detect that even some *decision literals* admit a reason, something which is ignored when using classical implication graphs. This finding opens new perspectives in SAT learning-based techniques. The extended implication graph can be used in different ways to learn more powerful conflict clauses. For example, the additional edges can be used to minimize the asserting clause obtained using the classical learning scheme or to improve its back-jumping capabilities. The inverses edges can also be used as alternative reasons of the process generation of the conflict clause itself.

The paper is organized as follows. After some preliminary definitions and notations (section 2), classical implication graph and learning schemes are presented in section 3. Then the optimality (in terms of backjumping level) of the asserting clause generated according to the first unique implication point is proved in section 4. In section 5, our extended implication graph is then presented and some of its possible features are informally described. Then, a detailed description of the first extension of the classical learning scheme that aims to derive more powerful conflict clauses in term of back-jumping is given in section 6. In section 6.1, the integration of our extended learning scheme to the state-of-the-art satisfiability solvers (MiniSat and Rsat) is described. Experimental results on many classes of SAT instances showing the interesting improvements obtained by our proposed approach are given before concluding.

2 Preliminary definitions and notations

A *CNF formula* $\mathcal{F}$ is a set (interpreted as a conjunction) of *clauses*, where a clause is a set (interpreted as 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 $\bar{l}$ 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 *interpretation* ρ of a boolean formula $\mathcal{F}$ is associates a value $\rho(x)$ 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 interpretation is alternatively represented by a complete and fundamental set of literals, in the obvious way. A *model* of a formula $\mathcal{F}$ is an interpretation ρ that satisfies the formula; noted $\rho \vDash \Sigma$. 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 and are therefore satisfied are removed; and those containing $\neg x$ are simplified). This notation is extended to interpretations: given an interpretation $\rho = \{x_1, \ldots, x_n\}$, 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 $\mathcal{F}$ do 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 $\mathcal{F}$.
- $\models_*$ denotes deduction by unit propagation: $\mathcal{F} \models_* x$ means that a literal x is deduced by propagation from $\mathcal{F}$, i.e. $x \in \mathcal{F}^*$. We write $\mathcal{F} \models_* \bot$ if the formula is proved inconsistent by propagation. In particular, note that if we have a clause c such that $\mathcal{F} \wedge \overline{c} \models_* \bot$, then c is a consequence of $\mathcal{F}$. We say that c is deduced *by refutation*.

Let us now introduce some notations and terminology on SAT solvers based on the Davis Logemann Loveland procedure, commonly called DPLL [2]. DPLL is a backtrack search procedure; at each node the assigned literals (decision literal and the propagated ones) are labeled with the same *decision level* starting from 1 and increased at each branching branching. The current decision level is the highest decision level in the assignment stack. 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 k and each $x_{k_j}^i$ for $1 \leq j \leq n_k$ represents a propagated (unit) literals at level k. 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 given level i, we define ρ^i as the projection of ρ to literals assigned at a level $\leq i$.

3 Conflict analysis using Implication Graphs

Implication graphs are a representation which captures the variable assignments ρ made suring the search, both by branching and by propagation. This representation is a convenient way to analyse conflicts. In classical SAT solvers, whenever a literal y is propagated, we keep a reference to the clause at the origin of the propagation of y, which we note $\overrightarrow{cla}(y)$. The clause $\overrightarrow{cla}(y)$ 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 ($\rho(x_i) = \mathit{false}, \forall i \in 1..n$),

while $\rho(y) = \mathit{true}$. When a literal y is not obtained by propagation but comes from a decision, $\overrightarrow{cla}(y)$ is undefined, which we note for convenience $\overrightarrow{cla}(y) = \bot$.

When $\overrightarrow{cla}(y) \neq \bot$, we denote by $exp(y)$ the set $\{\overline{x} \mid x \in \overrightarrow{cla}(y) \setminus \{y\}\}$, called set of *explanations* of y. In other words if $\overrightarrow{cla}(y) = (x_1 \vee \cdots \vee x_n \vee y)$, then the explanations are the literals $\overline{x_i}$ with which $\overrightarrow{cla}(y)$ becomes the unit clause $\{y\}$. Note that for all i we have $l(\overline{x_i}) \leq l(y)$, i.e., all the explanations of the deduction come from a level at most as high. When $\overrightarrow{cla}(y)$ is undefined we define $exp(y)$ 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 ordered interpretation, and let exp denote a choice of explanations for the deduced literals in ρ. The implication graph associated to $\mathcal{F}$, ρ and exp is $(\mathcal{N}, \mathcal{E})$ where:*

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

Example 1. $\mathcal{G}^{\rho}_{\mathcal{F}}$, shown in Figure 1 is an implication graph for the formula $\mathcal{F}$ and the partial assignment ρ given below : $\mathcal{F} \supseteq \{c_1, \ldots, c_9\}$

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

$\rho = \{\langle \ldots \neg x^1_6 \ldots \neg x^1_{17} \rangle \langle (x^2_8) \ldots \neg x^2_{13} \ldots \rangle \langle (x^3_4) \ldots x^3_{19} \ldots \rangle \ldots \langle (x^5_{11}) \ldots \rangle\}$. The current decision level is 5.

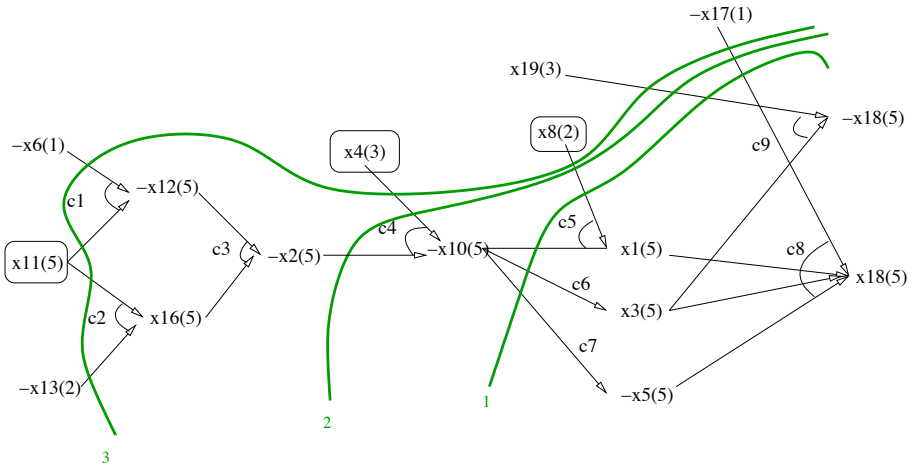


Fig. 1. Implication Graph $\mathcal{G}^{\rho}_{\mathcal{F}} = (\mathcal{N}, \mathcal{E})$

3.1 Generating asserting clauses

In this section, we formally describe the classical learning schemes used in modern SAT solvers. In the following definitions, we consider $\mathcal{F}$ a CNF formula, ρ a partial assignment such that $(\mathcal{F}|_\rho)* = \bot$ and $\mathcal{G}^\rho_\mathcal{F} = (\mathcal{N}, \mathcal{E})$ the associated implication graph. Assume that the current decision level is m. As a conflict is reached, then $\exists x \in$ st. $\{x, \neg x\} \subset \mathcal{N}$ and $l(x) = m$ or $l(\neg x) = m$. Conflict analysis is based on applying resolution from the top to the bottom of the implication graph using the different clauses of the form $(\overline{exp(y)} \vee y)$ implicitly encoded at each node $y \in \mathcal{N}$. We call this process a conflict resolution proof. Let us now formally define the concept of asserting clause, conflict resolution proof and unique implication point.

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, which we note in short $\mathcal{A}(c)$.*

We define $jump(c) = max\{l(\neg y) \mid y \in \alpha\}$.

Definition 3 (Conflict resolution proof). *A conflict resolution proof π is a sequence of clauses $\langle \sigma_1, \sigma_2, \ldots \sigma_k \rangle$ satisfying the following conditions :*

1. *$\sigma_1 = \eta[x, \overrightarrow{cla}(x), \overrightarrow{cla}(\neg x)]$, 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 $\overrightarrow{cla}(\overline{y})$ is defined. We then have $y \in \sigma_{i-1}$ and $\overline{y} \in \overrightarrow{cla}(\overline{y})$: the two clauses resolve. The clause σ_i is defined as $\eta[y, \sigma_{i-1}, \overrightarrow{cla}(\overline{y})]$;*
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: $\mathcal{F} \models \sigma_i$.

Another important remark is that in modern SAT solvers, the literals y used in the condition 2 of definition 3 are restricted to those of the current decision level.

Definition 4 (Elementary conflict resolution proof). *A conflict resolution proof $\pi = \langle \sigma_1, \sigma_2, \ldots \sigma_k \rangle$ is called elementary iff $\nexists i < k$ s.t. $\langle \sigma_1, \sigma_2, \ldots \sigma_i \rangle$ is also a conflict resolution proof.*

Using the definitions 3 and 4, we can now define the concepts of Unique Implication Point (UIP) and First UIP:

Definition 5 (Unique Implication Point (UIP in short)). *A node $x \in \mathcal{N}$ is a UIP iff there exists a conflict resolution proof $\langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ st. $\overline{x} \in \sigma_k$ and $l(x)$ is equal to the current decision level, m. (Note that σ_k, being assertive, has exactly one such x.)*

Definition 6 (First Unique Implication Point). *A node $x \in \mathcal{N}$ is a First UIP iff it is obtained from an* elementary *proof; i.e. $\exists \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ an elementary conflict resolution proof st. $\overline{x} \in \sigma_k$ and $l(x) = m$.*

Definition 7 (Last Unique Implication Point). *The last UIP is defined as the literal $d(\rho, m)$, i.e. the decision literal at the level of failure m.*

To illustrate the previous definitions, let us consider again the example 1.

The traversal of the graph $\mathcal{G}^{\rho}_{\mathcal{F}}$ allow us to generate three asserting clauses corresponding to the three possible UIPs (see figure 1). Let us illustrate the conflict resolution proof leading to the first asserting clause Δ_1 corresponding to the first UIP (see cut 1 in Fig. 1).

- $\sigma_1 = \eta[x_{18}, c_8, c_9] = (x^1_{17} \vee \neg x^5_1 \vee \neg x^5_3 \vee x^5_5 \vee \neg x^3_{19})$
- $\sigma_2 = \eta[x_1, \sigma_1, c_5] = (x^1_{17} \vee \neg x^5_3 \vee x^5_5 \vee \neg x^3_{19} \vee \neg x^2_8 \vee x^5_{10})$
- $\sigma_3 = \eta[x_5, \sigma_2, c_7] = (x^1_{17} \vee \neg x^5_3 \vee \neg x^3_{19} \vee \neg x^2_8 \vee x^5_{10})$
- $\sigma_4 = \eta[x_3, \sigma_3, c_6] = (x^1_{17} \vee \neg x^3_{19} \vee \neg x^2_8 \vee x^5_{10})$

As we can see, σ_4 gives us a first asserting clause (that we'll also name Δ_1) because all of its literals are assigned before the current level except one (x_{10}) which is assigned a the current level 5; $\langle \sigma_1, \sigma_2, \sigma_3, \sigma_4 \rangle$ is an elementary conflict resolution proof (the intermediate clauses of π contain more than one literal of the current decision level 5), and $\neg x_{10}$ is a first UIP.

If we continue such a process, we obtain the two additional asserting clauses $\Delta_2 = (x^1_{17} \vee \neg x^3_{19} \vee \neg x^2_8 \vee \neg x^3_4 \vee x^5_2)$, corresponding to a second UIP $\neg x^5_2$; and $\Delta_3 = (x^1_{17} \vee \neg x^3_{19} \vee \neg x^2_8 \vee \neg x^3_4 \vee x^2_{13} \vee x^1_6 \vee \neg x^5_{11})$, corresponding respectively to a 3rd UIP ($\neg x^5_{11}$) which is the last UIP since it corresponds to the decision literal (see cuts 2 and 3 in Fig. 1).

Property 1 (Asserting clause and backjumping). Let $c = (\alpha \vee x)$ an asserting clause deduced by conflict analysis, and $i = max\{l(\neg y) \mid y \in \alpha\}$. We can safely backjump to level i and consider the partial assignment $\rho^i \cup \{x\}$

Property 2. Let $c = (\alpha \vee x)$ be an asserting clause deduced by conflict analysis, and $i = max\{l(\neg y) \mid y \in \alpha\}$. Then x can be deduced by refutation at level i, i.e. $(\mathcal{F} \wedge \overline{x})|_{\rho^i} \models_* \bot$.

Proof. The assignment of all literals $y \in \alpha$ to *false* leads exactly to the same conflict by unit propagation (conflict side of the implication graph)

The above property shows that the asserting literal can be deduced by refutation at level i. This mean that if we apply some lookahead local processing at that level, we could derive it before actually reaching the conflict. The main difficulty, however, lies in how to select efficiently the literals to process by unit propagation. An attempt to answer this question can be found for example in the approaches proposed by Dubois etal. [3] and by Li and Anbulagan [6].

This simple property shows that modern SAT solvers explore a binary search tree, and they can be considered as a variant of DPLL procedure. This point of view is not shared by all the SAT researchers.

4 Optimality results of the first UIP asserting clause

To motivate our proposed extension, in this short section we prove a fundamental property about the optimality of the asserting clause generated according to the first UIP learning scheme: it is optimal in terms of backjumping level[1]. This simple property explains why the first UIP usually considered in classical learning schemes is more powerful than the the other UIPs.

Given a clause c we denote by $levels(c)$ the set of different levels present in the clause c, i.e. $levels(c) = \{l(x) \mid x \in c\}$.

Property 3. In a conflict resolution proof $\langle c_1, \ldots, c_k \rangle$ we have for all $i < k$, $levels(c_i) \subseteq levels(c_{i+1})$.

Proof. Each σ_{i+1} is obtained from σ_i by selecting a literal $x \in \sigma_i$ that is not a decision, and replacing this literal by its set of explanations $exp(x)$. These explanations always contain another literal y such that $l(y) = l(x)$.

Property 4. Let $c = (\alpha, x)$ be an asserting clause obtained by conflict resolution and in which x is the first UIP. The back-jumping level of c is optimal: any asserting clause c' obtained by conflict resolution is such that $jump(c') \geq jump(c)$.

Proof. It can be shown that all asserting clauses containing the first UIP have the same backjump level (there could be several such clauses, in general, under our definitions, and their sets of $levels$ may be incomparable). Let $\langle c_1, \ldots, c_k \rangle$ be a conflict resolution proof of $c' = c_k$. Let i be the index of the first asserting clause in this proof. Because c_i is an asserting clause we have $jump(c) = jump(c_i)$ and because of property 3 we have $jump(c_i) \leq jump(c_k)$.

Note that the first UIP is the only UIP with these two optimality guarantees: one can construct examples on which the asserting clauses containing any other UIP have a strictly higher backjump level.

5 Extended implication graph

In this section, we explain how extra edges coming from clauses that are satisfied can be added to the conflict graph and be exploited advantageously [2]. In particular, these edges sometimes allow us to improve the backjump level. As the previous section showed, this is something which is simply impossible if we do not reconsider the notion of implication graph.

[1] From private discussions [7] it emerges that this result is part of the "folklore" known to some researchers of the community but which, to the best of our knowledge, is however not stated formally in the literature.

[2] An extended notion of implication graph, in which several explanation clauses are maintained for each literal of the current level, has been considered in the literature by [1]. The aim of this work was, however, theoretical, while our goal is to show that inverse edges also have a practical value.

5.1 Illustration of the notion

In modern SAT solvers, clauses containing a literal x that is implied at the current level are essentially ignored by the propagation. More precisely, because the solver does not maintain the information whether a given clause is satisfied or not, a clause containing x may occasionally be considered by the propagation, but only when another literal y of the clause becomes false. When this happens the solver typically skips the clause. However, in cases where x is true *and all the other literals are false*, an "arc" was revealed for free that could as well be used to extend the graph. Such arcs are those we propose to use in our extension.

To explain our idea let us consider, again, the formula $\mathcal{F}$ given in the example 1. We consider the same partial instantiation and we define a new formula $\mathcal{F}'$ as follow : $\mathcal{F}' \supseteq \{c_1, \ldots, c_9\} \cup \{c_{10}, c_{11}, c_{12}\}$ where $c_{10} = (\neg x_{19} \vee x_8), c_{11} = (x_{19} \vee x_{10})$ and $c_{12} = (\neg x_{17} \vee x_{10})$

The three added clauses are satisfied under the instantiation ρ. c_{10} is satisfied by x_8 assigned at level 2, c_{11} is satisfied by x_{19} at level 3, and c_{12} is satisfied by $\neg x_{17}$ at level 1. This is shown in the extended implication graph (see Figure 2) by the doted edges. Let us now illustrate the usefulness of our proposed extension. Let us consider again the the asserting clause Δ_1 corresponding to the classical first UIP. We can generate the following strong asserting clause.

- $c_{13} = \eta[x_8, \Delta_1, c_{10}] = (x_{17}^1 \vee \neg x_{19}^3 \vee x_{10}^5)$
- $c_{14} = \eta[x_{19}, c_{13}, c_{11}] = (x_{17}^1 \vee x_{10}^5)$
- $\Delta_1^s = \eta[x_{17}, c_{14}, c_{12}] = x_{10}^5$

In this case we backtrack to the level 0 and we assign x_{10} to *true*. Indeed $\mathcal{F}' \models x_{10}$.

As we can see Δ_1^s subsumes Δ_1. If we continue the process we also obtain other strong asserting clauses $\Delta_2^s = (\neg x_4^3 \vee x_2^5)$ and $\Delta_3^s = (\neg x_4^3 \vee x_{13}^2 \vee x_6^1 \vee \neg x_{11}^5)$ which subsume respectively Δ_2 and Δ_3. This first illustration gives us a new way to minimize the size of the asserting clauses.

If we take a look to the clauses used in the classical implication graph $\mathcal{G}_{\mathcal{F}}^{\rho}$ (figure 1) all have the following properties: (1) $\forall x \in \mathcal{N}$ the clause $c = (\overline{exp(x)} \vee x)$ is satisfied by only one literal i.e. $\rho(x) =$ *true* and $\forall y \in exp(x)$, we have $\rho(y) =$ *true* and (2) $\forall y \in exp(x)$, $l(\neg y) \leq l(x)$. Now in the extended implication graph (figure 2) the added clauses satisfy property (1) and, in addition, the property (2') $\exists y \in exp(x)$ st. with $l(\neg y) > l(x)$.

Let us now explain briefly how the extra arcs can be computed. Usually unit propagation does not keep track of implications from the satisfiable sub-formula. In this extension the new implications (deductions) are considered. For instance in the previous example, when we deduce x_{19} at level 3, we "rediscover" the deduction x_8 (which was a choice (a decision literal) at level 2). Similarly, for $\neg x_{10}$ deduced at level 5, we "rediscover" the deductions x_{19} (made at level 3) and $\neg x_{17}$ (made at level 1).

Our proposal keeps track of these re-discoveries. Note something unusual: even a decision literal (choice point) can now have incoming edges. This is the case for x_8 for instance.

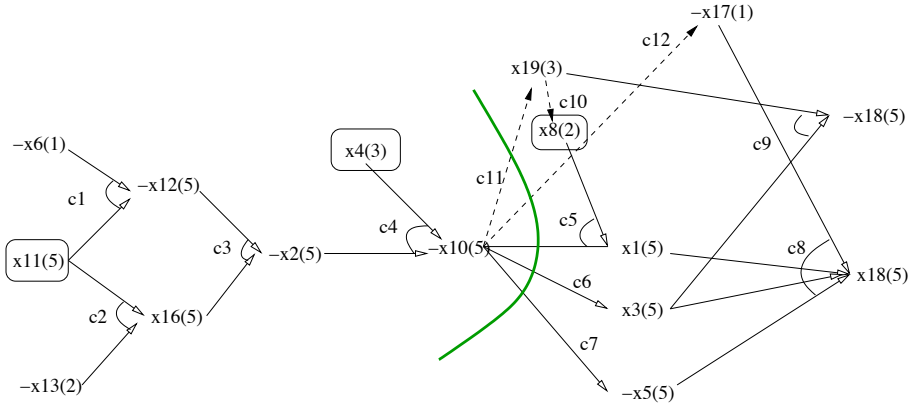


Fig. 2. Implication graph with inverse edges : $\mathcal{G}s^{\rho}_{\mathcal{F}'} = (\mathcal{N}', \mathcal{E}')$

Before introducing the formal definition of our extended Implication Graph, we introduce the concept of inverse implication (inverse edge).

We maintain additionally to the classical clause $\overrightarrow{cla}(x)$ a new clause $\overleftarrow{cla}(x)$ of the form $(x \vee y_1 \vee \cdots \vee y_n)$. This clause is selected so that $\rho(y_i) = \mathit{false}$ for $i \in 1..n$; $\rho(x) = \mathit{true}$; and $\exists i.\ l(y_i) > l(x)$. This clause can be undefined in some cases (which we note $\overleftarrow{cla}(x) = \bot$). Several clauses of this form can be found for each literal, in which case one is selected arbitrarily: one can choose to consider the first one in the ordering. (It is easy to define a variant where we would take into account all of them, in which case $\overleftarrow{cla}(x)$ is a set of clauses; but we won't develop this variant).

We denote by $\overleftarrow{exp}(x)$ the set $\{\overline{y} \mid y \in \overleftarrow{cla}(x) \setminus \{x\}\}$, and, for clarity, by $\overrightarrow{exp}(x)$ the set that was previously noted exp. An extended implication graph is defined as follows (note that this graph is now not acyclic in general):

Definition 8 (Extended Implication Graph). *Let $\mathcal{F}$ be a CNF formula and ρ an ordered partial interpretation. We define the extended implication Graph associated to $\mathcal{F}$ and ρ as $\mathcal{G}s^{\rho}_{\mathcal{F}} = (\mathcal{N}, \mathcal{E} \cup \mathcal{E}')$ where,*

- $\mathcal{N} = \rho$
- $\mathcal{E} = \{(x, y) \mid x \in \rho, y \in \rho, x \in \overrightarrow{exp}(y)\}$, $\mathcal{E}' = \{(x, y) \mid x \in \rho, y \in \rho, x \in \overleftarrow{exp}(y)\}$

6 Learning to back-jump: a first extension

In this section, we describe a first possible extension of CDCL based approach using extended implication graph. Our proposed approach makes an original use of inverses arcs to back-jump farther i.e. to improve the back-jumping level of the classical asserting clauses.

Let us illustrate the main idea behind our proposed extension. Our approach works in three steps. In the first step (1) : an asserting clause, say $\sigma_1 = (\neg x^1 \vee \neg y^3 \vee \neg z^7 \vee \neg a^9)$ is learnt using the usual learning scheme where 9 is the current decision level. As

$\rho(\sigma_1) = false$, usually we backtrack to level $jump(\sigma_1) = 7$. In the second step (2): our approach aims to eliminate the literal $\neg z^7$ from σ_1 using the new arcs of the extended graph. Let us explain this second and new processing. Let $c = (z^7 \vee \neg u^2 \vee \neg v^9)$ such that $\rho(z) = true$, $\rho(u) = true$ and $\rho(v) = true$. The clause c is an inverse arc i.e. the literal z assigned at level 7 is implied by the two literals u and v respectively assigned at level 2 and 9. From c and σ_1, a new clause $\sigma_2 = \eta[z, c, \sigma_1] = (\neg x^1 \vee \neg u^2 \vee \neg y^3 \vee \neg v^9 \vee \neg a^9)$ is generated by resolution. We can remark that the new clause σ_2 contains two literals from the current decision level 9. In the third step (3), using classical learning, one can search from σ_2 for another asserting clause σ_3 with only one literal from the current decision level. Let us note that the new asserting clause σ_3 might be worse in terms of back-jumping level. To avoid this main drawback, the inverse arc c is chosen if the two following conditions are satisfied : i) the literals of c assigned at the current level (v^9) has been already visited during the first step and ii) all the other literals of c are assigned before the level 7 i.e. level of z. In this case, we guaranty that the new asserting clause satisfy the following property : $jump(\sigma_3) \leq jump(\sigma_1)$. Interestingly enough, the asserting literal of σ_3 is $\neg a$.

One can iterate the previous process on the new asserting clause σ_3 to eliminate the literals of σ_3 assigned at level $jump(\sigma_3)$.

Let us now introduce more formally this approach. In the following definitions, we define the concept of joint inverse arc to extend the classical conflict resolution.

Definition 9 (Joint inverse arc). *Let $\pi = \langle \sigma_1, \sigma_2, \ldots \sigma_k \rangle$ be a conflict resolution proof. A clause $c \in \mathcal{F}$ is called a joint inverse arc of π if it satisfies the following conditions:*

1. *c is the inverse arc associated with one of the literals of σ_k whose level is the backjumping level: $\exists x \in \sigma_k$ st. $c = \overleftarrow{cla}(\overline{x})$ and $l(x) = jump(\sigma_k)$;*
2. *$\forall y \in c$, we have either $l(y) = m$ or $l(y) < jump(\sigma_k)$.*

When there exists a joint inverse arc c for a conflict resolution proof π, this means that we can use it to *extend* this proof: we obtain a clause $\sigma_{k+1} = \eta[x, \sigma_k, c]$, from which we start a new proof $\pi' = \langle \sigma_{k+1} \ldots \sigma_\iota \rangle$. (Intuitively c "joins" π and π'.) The whole sequence if called joint conflict resolution proof:

Definition 10 (Joint conflict resolution proof). *A joint conflict resolution proof is a sequence of clauses $\langle \sigma_1 \ldots \sigma_k, \sigma_{k+1}, \ldots \sigma_\iota \rangle$ such that*

- *$\langle \sigma_1 \ldots \sigma_k \rangle$ is a conflict resolution proof;*
- *$\sigma_{k+1} = \eta[x, \sigma_k, c]$, where c is a joint inverse arc;*
- *The clauses $\sigma_{k+2}, \ldots \sigma_\iota$ go ahead like a new conflict resolution proof, i.e. they respect the conditions (2) and (3) of Def. 3.*

Clearly, from the general definition 10, one can derive new asserting clauses. The join step (resolving the two clauses σ_k and c) leads us to a new clause where one of the literals with the greatest level is eliminated. From the new generated clause σ_{k+1} we can derive by resolution (using the arcs of the classical implication graph) another asserting clause σ_ι. To make sure that the new asserting clause σ_ι admits a back-jumping level lower or equal to $jump(c_k)$, a new condition on the joint inverse arc is introduced:

Definition 11. *Let $\pi = \langle \sigma_1, \sigma_2, \ldots \sigma_k \rangle$ be a conflict resolution proof. We define $resLit(\pi)$ as the set of literals of the last decision level m used in some resolvent of π.*

Property 5. Let $\pi^e = \langle \sigma_1 \ldots \sigma_k, \sigma_{k+1}, \ldots \sigma_\iota \rangle$ be a joint conflict resolution proof where k is the position of the join with the inverse arc c. If $\forall x \in c$ st. $l(x) = m$, we have $x \in resLit(\pi_1) \cup \{\mathcal{A}(\sigma_k)\}$ then $jump(\sigma_\iota) \leq jump(\sigma_k)$.

Proof. Let us sketch the proof. As π_1 is a conflict resolution proof, $\sigma_k = (\alpha \vee y \vee x)$ where $\mathcal{A}(\sigma_k) = x$ and $l(\neg y) = jump(\sigma_k)$, is an asserting clause. Without loss of generality, suppose that $\forall z \in \alpha, l(\neg z) < l(\neg y)$. As π^e is a joint conflict resolution proof, then $\exists c = (\beta \vee \neg y)$ a joint inverse arc and $\sigma_{k+1} = \eta[y, \sigma_k, c] = (\alpha \vee \beta \vee x)$. Now, from σ_{k+1}, we apply resolution only on the set of literals $\beta_1 = \beta \cap resLit(\pi_1)$. As these literals are previously used in the conflict resolution proof π_1, then from σ_{k+1} we can derive by following the same arcs (by resolution) the asserting clause σ_ι of the form $(\alpha \vee \beta \backslash \beta_1 \vee x)$ or $(\alpha \vee \beta \backslash \beta_1 \vee y \vee x)$. In the first case, $jump(\sigma_\iota) < jump(\sigma_k)$. From the hypothesis and the definition of c (see condition 2), we deduce that the literals of α and $\beta \backslash \beta_1$ are assigned at smaller level than $l(\neg y)$ respectively. In the second case, the literal y to be eliminated from σ_k appears in σ_ι. Consequently, the two asserting clauses admit the same back-jumping level $l(y)$. Suppose that, the asserting clause σ_k contains several literals with the same level as y. In this case, by eliminating only one literal, we obtain the same back-jumping level.

From the property 5, we have shown how to improve the back-jumping by eliminating one literal of the greatest level. Obviously, when the asserting clause contains several literals of the greatest level, we can either iterate the processing on the new asserting clause to eliminate another literal. Interestingly enough, on all literals of the greatest level are eliminated, one can process the other literals with higher level and then achieve successive back-jumping from a single conflict. However such iterating process can be time consuming in practice.

In the following section we address the practical choice and restrictions that we have adopted to show the feasibility of considering inverse arcs.

6.1 Practical integration to modern sat solvers

Our extended learning scheme can be crafted to any CDCL based solver. Let us recall the three important components of modern SAT solvers : (1) the conflict analysis, (2) heuristics VSIDS, literal activity and nogood data base updating (3) restart policy. In this paper our proposed extension is integrated to the two state-of-the art satisfiability solvers MiniSat and Rsat. The modifications mainly concern the two first components as explained below:

1. *Conflict analysis* : At each conflict, classical learning is applied and an asserting clause $\sigma_1 = (\alpha \vee y^i \vee x^m)$ with x the asserting literal and $i = jump(\sigma_1)$, is generated with an elementary conflict resolution proof π_1 (first UIP principle). Before back jumping to level i, we try to extended to form a joint conflict resolution proof leading to a new asserting clause σ_2. In our implementation, this extension is only

applied if σ_1 contains only one literal of level i. In general, one can iterate the processing on all the literals of level i. Another restriction is that the inverse arcs must contain only literals of level m or j with $j < i$ i.e. $x \in resLit(\sigma_1) \cup \{\mathcal{A}(\sigma_1)\}$. Without this restriction, one can generate other asserting clauses that might be better or worse in term of back-jumping level.

2. *Literals activity and learnt data base management*: First the activity of the new variables introduced by the inverse arcs are also updated. Secondly, the two asserting clauses are added to the nogood data base except when the new one subsume the first asserting clause. In this last case, the classical asserting clause is not added.

7 Experiments

The experimental results reported in this section are obtained on a Xeon 3.2 GHz (2 GB RAM) and performed on a large panel of SAT instances (286) coming from SAT RACE2006 and SAT07 (industrial). All instances are simplified by the satelite preprocessor [4]. Time limit is fixed to 1800 seconds and results are reported in seconds. We implement our proposed extension (section 6.1) to Minisat [5] and Rsat [9] and make a comparison between original solvers and extended ones (called Minisat^E and Rsat^E).

The scatter plot (in log scale) given in figure 3 illustrates the comparative results of Minisat (figure left) and Rsat (figure right) with respect to their extended versions. The x-axis (resp. y-axis) corresponds to the cpu time tx (resp. ty) obtained on the extended solver (resp. original solver). Each dot with (tx, ty) coordinates corresponds to a SAT instance. Dots above (resp. below) the diagonal indicate that the extended solver is faster (resp. slower) than the original ones. Figure 3 exhibits clearly that inverse edges speed-up Minisat. On Rsat the improvements are less impressive. But, and it is important, Rsat^E is able to solve 5 more instances than original Rsat (whereas Minisat^E solves 1 instances less than the original one).

Figure 4 shows the time T (figure on the left-hand side) and the cumulated time CT (figure on the right hand side) needed to solve a given number of instances ($\#instances$). T and CT represent respectively the number of instances with running time less than T seconds and the number of solved instances if we consider that all the instances are run sequentially within a time limit of T seconds. This global view clearly shows that as T or CT increase the extended versions of the solvers solve more instances than the original ones.

Finally, Table 1 highlights the results obtained on a representative sample of instances. For each instance, we report the running time (in seconds) for each of the four versions : MiniSat, Rsat, Minisat^E ad Rsat^E.

For the two extended versions of the Minisat and Rsat solvers, the back-jumping level is improved on 1% to 9% of conflicts. Despite, this low percentage, our extension is clearly helpful. Let us recall that we apply the joint conflict resolution proof only if the asserting clause contains one literal of the back-jumping level. The obtained results suggest that the inverse arcs are very useful. Furthermore, on many classes of instances, their exploitation speeds-up the running time by more than one order of magnitude. Rsat^E is clearly the best on the `ibm`, `Velev`, `total` and `simon` families whereas Minisat^E achieves significant improvements on `manol`, `vmpc`, `gold`, `APro` families.

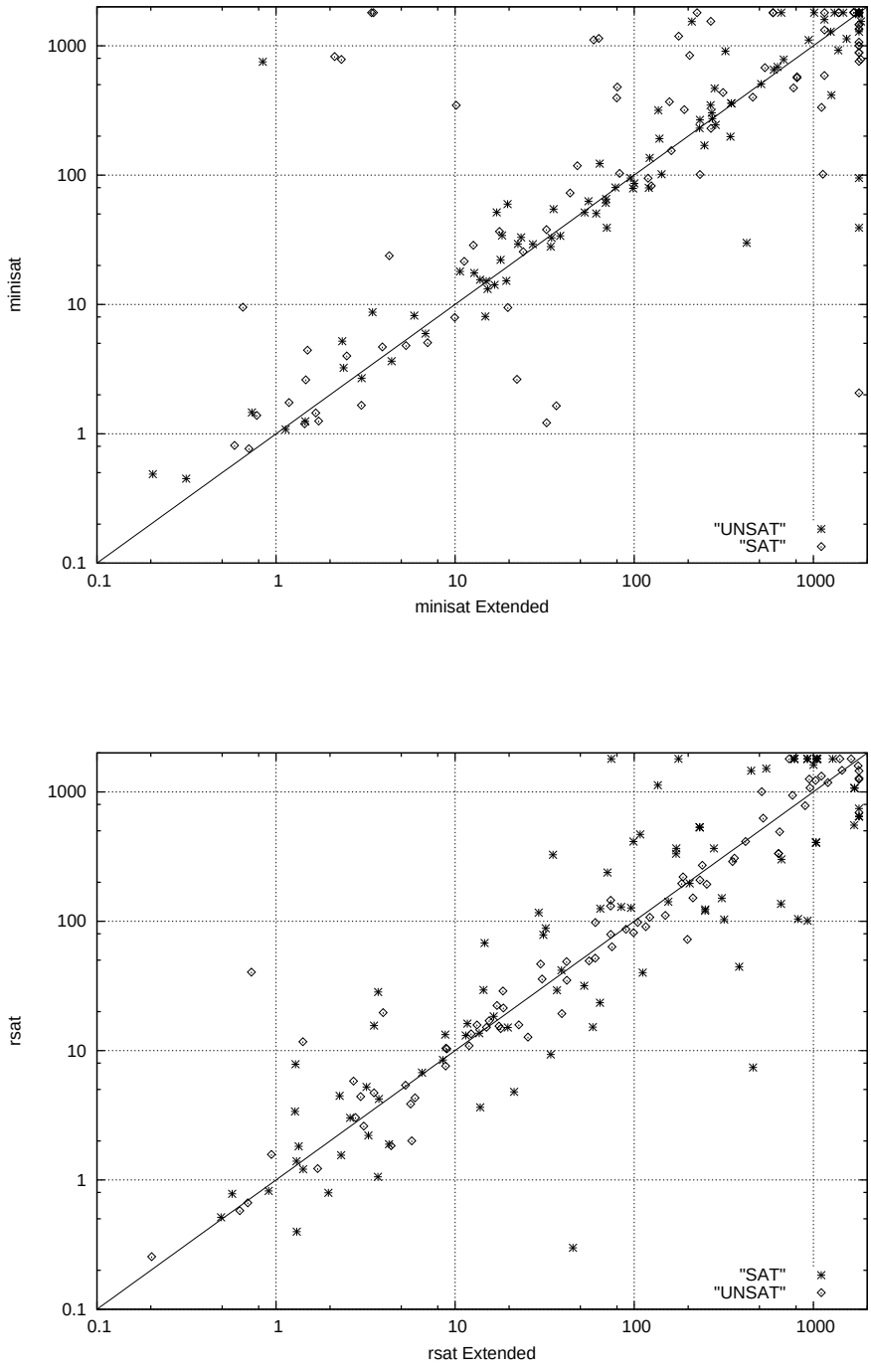
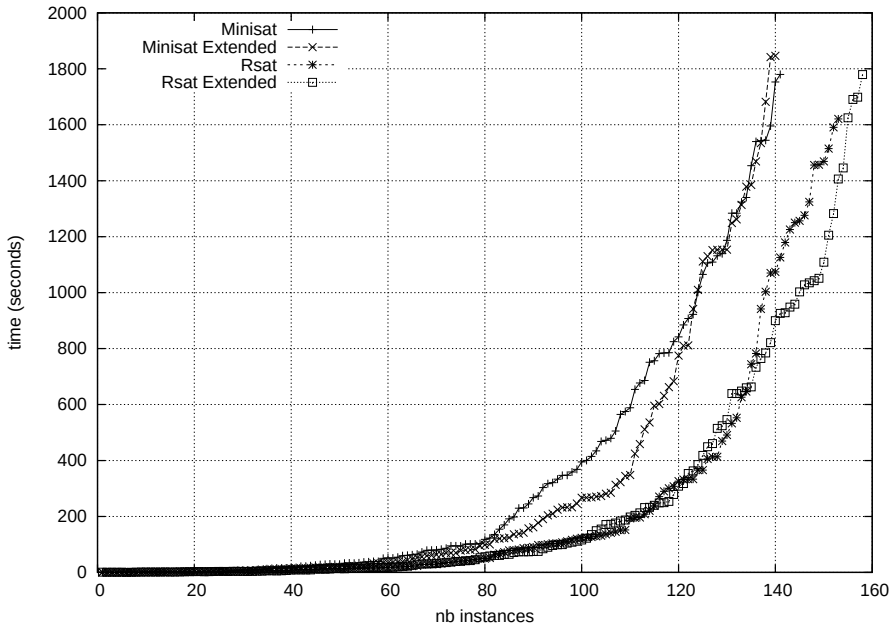


Fig. 3. Results on SAT'07 (industrial) and SATRACE06 benchmarks



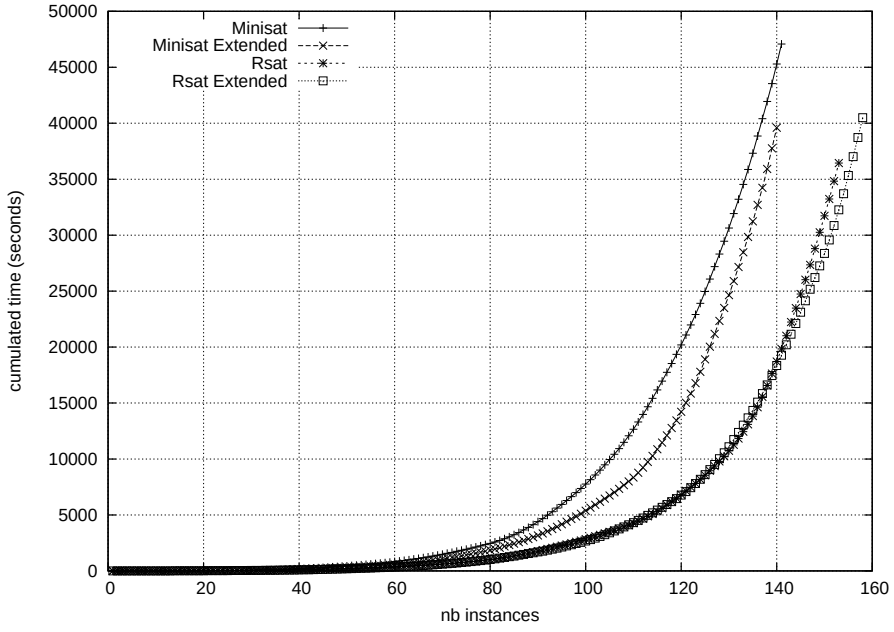


Fig. 4. Time and cumulated time for solving a given number of instances

8 Conclusion

In this paper, we have proposed a generalized framework for conflict analysis. This generalization is obtained by an original extension of the classical implication graph usually used in CDCL based solvers. This extension, motivated by the optimality result of the classical asserting clause (Firs UIP) in terms of back-jumping level, is obtained by considering some clauses (called inverse arcs) that come from the satisfiable part of the formula. Several learning schemes can be defined from this extension. The first possible extension of learning that improves the classical asserting clauses in term of back-jumping level shows the great potential of the new framework. Despite the different restrictions, its integration achieves interesting improvements of the state-of-the art satisfiability solvers (Rsat and MiniSat) on the industrial instances taken from the last SAT competition and SAT race. In future works, we plan to define other learning schemes, for example, by considering the inverse arcs as an alternative at each node of the classical conflict resolution proof. That is to say, to improve the different resolvent generated during the conflict resolution proof in term of back-backjumping and/or size. Finally, we also plan to exploit the inverse arcs to minimize the asserting clauses.

References

1. P. Beame, H. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. *J. of Artificial Intelligence Research*, 22:319–351, 2004.
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. 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.
4. 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.
5. Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, 2002.
6. Chu Min 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.
7. J. Marques-Silva. Personal communication.
8. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of (DAC'01)*, pages 530–535, 2001.
9. Knot Pipatsrisawat and Adnan Darwiche. Rsat 2.0: Sat solver description. Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA, 2007.

instance	SAT?	Rsat^E	Rsat.	Minisat^E	Minisat
AProVE07-02	N	–	–	683.21	782.07
AProVE07-21	N	1406.02	–	511.14	505.41
AProVE07-22	N	232.68	208.17	136.19	316.98
clauses-6	Y	171.21	331.93	810.24	564.78
cube-9-h11-sat	Y	1282.38	–	774.78	472.25
dated-5-15	N	958.61	1074.07	–	1752.71
goldb-heqc-frg2mul	N	187.42	219.99	281.12	468.28
goldb-heqc-i8mul	N	1108.60	1324.17	941.759	1105.80
goldb-heqc-term1mul	N	–	1250.23	–	–
ibm-2002-19r-k100	Y	95.94	126.84	157.31	368.49
ibm-2002-21r-k95	Y	64.68	125.04	268.14	229.77
ibm-2002-26r-k45	N	3.96	19.65	0.84	751.33
IBM_04_30_dat.k8	Y	1042.88	–	1682.10	–
IBM_2004_30_SAT_dat.k100	Y	784.22	–	–	–
IBM_2004_30_SAT_dat.k55	Y	231.99	532.94	–	–
IBM_2004_30_SAT_dat.k60	Y	1698.74	1070.30	595.85	–
IBM_2004_30_dat.k80	Y	925.87	–	–	–
IBM_2004_30_dat.k85	Y	1042.88	–	1682.10	–
IBM_2004_30_dat.k90	Y	1051.16	–	–	–
manol-pipe-c7nidw	N	254.00	193.00	1469.20	–
manol-pipe-f7idw	N	1624.76	–	1010.01	–
manol-pipe-g10bidw	N	–	–	1313.43	–
manol-pipe-g10id	N	73.73	131.10	209.71	1539.87
mizh-md5-47-3	Y	107.98	469.35	1111.21	333.34
mizh-md5-47-4	Y	135.36	1125.72	267.50	1544.60
mizh-sha0-35-4	Y	278.60	365.84	1153.66	1323.37
partial-5-13-s	Y	546.21	1514.82	–	–
schup-l2s-guid-1-k56	N	524.18	626.08	322.68	907.43
simon-s02b-dp11u10	N	515.00	1003.18	284.89	244.64
simon-s02b-r4b1k1.1	Y	1002.87	1620.79	458.83	400.30
total-10-17-s	Y	98.98	413.28	–	–
total-10-19-s	Y	74.46	–	–	–
total-5-17-s	Y	14.59	67.85	1384.98	–
velev-pipe-sat-1.1-b7	Y	70.85	238.16	224.04	–
velev-pipe-uns-1.0-9	N	764.74	941.76	–	–
velev-pipe-uns-1.1-7	N	733.14	–	–	–
vmpc_30	Y	176.49	–	–	–

Table 1. Highlighted results