Ann Oper Res (2016) 240:13–37
DOI 10.1007/s10479-015-2028-9

SI: 4OR SURVEYS

What we can learn from conflicts in propositional satisfiability

Youssef Hamadi[1,2] · **Saïd Jabbour**[3] · **Lakhdar Saïs**[3]

Published online: 1 October 2015

Abstract Learning is a general concept, playing an important role in many Artificial intelligence domains. In this paper, we address the learning paradigm used to explain failures or conflicts encountered during search. This explanation, derived by conflict analysis, and generally expressed as a new constraint, is usually used to dynamically avoid future occurrences of similar situations. Before focusing on clause learning in Boolean satisfiability (SAT), we first overview some important works on this powerful reasoning tool in other domains such as constraint satisfaction and truth maintenance systems. Then, we present a comprehensive survey of the most important works having led to what is called today—conflict driven clause learning—which is one of the key components of modern SAT solvers. We also overview some of the original extensions or variants of clauses learning. In theory, current SAT solvers with clause learning are as powerful as general resolution proof systems. In practice, real-world SAT instances with millions of variables and clauses are now in the scope of this solving paradigm.

Keywords Propositional satisfiability · Constraint solving · Reasoning · SAT

This is an updated version of the paper "Learning from Conflicts in Propositional Satisfiability" that appeared in *4OR*, 10(1), 15–32 (2012).

✉ Lakhdar Saïs
sais@cril.fr

Youssef Hamadi
youssefh@microsoft.com

Saïd Jabbour
said.jabbour@cril.fr

1 Microsoft Research, Cambridge, UK

2 LIX École Polytechnique, 91128 Palaiseau, France

3 CRIL-CNRS, Université Lille Nord de France - Artois, Rue Jean Souvraz SP18, 62307 Lens Cedex, France

1 Introduction

The SAT problem, i.e., the problem of checking whether a Boolean formula in conjunctive normal form (CNF) is satisfiable or not, is central to many domains in computer science and artificial intelligence including constraint satisfaction problems (CSP), automated planning, non-monotonic reasoning, VLSI correctness checking, etc. Today, SAT has gained a considerable audience with the advent of a new generation of solvers able to solve large instances encoding real-world problems, and the demonstration that these solvers represent important low-level building blocks for many important fields, e.g., SAT modulo theory, Theorem proving, Model checking, Quantified boolean formulas, Maximum Satisfiability, Pseudo boolean, etc. These solvers, often called *modern SAT solvers* (Moskewicz et al. 2001; Eén and Sörensson 2003), are based on the classical DPLL procedure (Davis et al. 1962) enhanced with: (1) an efficient implementation of unit propagation through incremental and lazy data structures, (2) restart policies (Gomes et al. 1998; Kautz et al. 2002), (3) activity-based variable selection heuristics (VSIDS-like) (Moskewicz et al. 2001), and (4) *clause learning* (Marques Silva and Sakallah 1996; Moskewicz et al. 2001). Clause learning is now recognized as one of the most important component of Modern SAT solvers. The main idea is that when a current branch of the search tree leads to a conflict, clause learning aims to derive a clause that succinctly expresses the causes of the conflict. Such learned clause is then used to prune the search space. Clause learning also known in the literature as conflict driven clause learning (CDCL) refers now to the most known and used First UIP learning scheme, first integrated in the SAT solver Grasp (Marques Silva and Sakallah 1996) and efficiently implemented in zChaff Moskewicz et al. (2001). Most of the SAT solvers, integrate this strong learning scheme. Theoretically, by integrating clause learning to DPLL-like procedures Davis et al. (1962), the obtained SAT solver formulated as a proof system is as powerful as general resolution (Pipatsrisawat and Darwiche 2009; Pipatsrisawat and Darwiche 2010).

This paper deals with learning from conflict, one of the most important component of modern SAT solvers. It surveys most of the works that led to the current well known First UIP learning scheme integrated in all the state-of-the-art SAT solvers. We also present some original extensions and variants of this classical learning scheme. To get a complete picture about SAT solvers, we additionally overview the other components mentioned above.

Our goal in this survey paper is to popularize this important paradigm to the operation research community, while hoping its possible integration to enhance the efficiency of branch and bound based algorithm. SAT have strong connection with operation research. Indeed, it is directly related to 0–1 integer linear programming or pseudo boolean constraints. A linear transformation for encoding as Boolean formulas in conjunctive normal form, 0/1 linear inequalities with integral coefficients was given by Warners (1998), while a clause can be seen as a simple 0/1 linear inequality. Consequently, a CNF formula can be formulated as a system of 0/1 linear inequalities and vice versa. On the solving side, branch and bound and DPLL-like procedure both develop a search tree based on the separation principle. Interestingly enough, the well known resolution principle and cutting plane rule are heavily related.

The paper is an updated version of Hamadi et al. (2012). It is organized as follows. After some preliminary definitions and notations, relationships between SAT and integer linear programming are discussed in Sect. 3. In Sect. 4, a comprehensive overview of related AI learning based approaches are briefly discussed. A survey of clauses learning based techniques are described in Sect. 5. The other components of modern SAT solvers are described before concluding.

2 Preliminary definitions and notations

A Boolean formula $\mathcal{F}$ in *Conjunctive Normal Form (CNF)* 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 denote by $\bar{l}$ the complementary literal of l. More precisely, if $l = x$ then $\bar{l}$ is $\neg x$ and if $l = \neg x$ then $\bar{l}$ is x. For a set of literals L, $\overline{L}$ is defined as $\{\bar{l} \mid l \in L\}$. Let us recall that any Boolean formula can be translated to CNF using linear Tseitin encoding (Tseitin 1968). A *unit* clause is a clause containing only one literal (called *unit literal*), while a binary clause contains exactly two literals. An *empty clause*, denoted $\bot$, is interpreted as false (unsatisfiable), whereas an *empty CNF formula*, denoted $\top$, is interpreted as true (satisfiable).

The set of variables occurring in $\mathcal{F}$ is denoted $V_{\mathcal{F}}$. 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 a function which associates a value $\rho(x)$ to some of the variables $x \in V_{\mathcal{F}}$. ρ is *complete* if it assigns a value to every $x \in V_{\mathcal{F}}$, and *partial* otherwise. An interpretation is alternatively represented by a complete and fundamental set of literals. A *model* of a formula $\mathcal{F}$ is an interpretation ρ that satisfies the formula; denoted $\rho \models \mathcal{F}$.

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 $\eta[x, c_i, c_j] = c_i \cup c_j \backslash \{x, \neg x\}$. A resolvent is called *tautological* when it contains complementary literals.
- $\mathcal{F}|_x$ denotes 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 therefore satisfied and 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 $\mathcal{F}$ 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 $\mathcal{F}$.
- $\models_*$ denotes deduction by unit propagation: $\mathcal{F} \models_* x$ means that a literal x is deduced by unit propagation from $\mathcal{F}$, i.e. $x \in \mathcal{F}^*$. We write $\mathcal{F} \models_* \bot$ if the formula is proved inconsistent by unit 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 unit propagation*.
- 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.

Let us now introduce some notations and terminology on SAT solvers based on the Davis Logemann Loveland procedure, commonly called DPLL (Davis et al. 1962). 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. 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 propagated (unit) literals at level k. Such a partial interpretation (sequence of decisions-propagations) associated to a given node of the search tree is called *partial ordered interpretation*.

Let $x \in \rho$, we denote by $l(x)$ 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 SAT and integer programming formulation

In this section, we briefly recall some connections between SAT and integer programming. Our goal is to convince the reader about the proximity of these two research domains and that cross-fertilization even if it has been widely investigated in the past remains an important and interesting research issue.

Let us identify Boolean values $false$ and $true$ with integers 0 and 1. Any CNF formula can be reformulated as a system of 0/1 linear inequalities. The following example illustrates such reformulation.

Example 1 Let $\mathcal{F}$ be a CNF formula made of the two clauses $c_1 = (x_1 \vee x_2 \vee \neg x_3)$ and $c_2 = (x_1 \vee \neg x_2)$. $\mathcal{F}$ can be reformulated as a system $S_{\mathcal{F}}$ of 0/1 linear inequalities:

$$\begin{aligned} & x_1 + x_2 + (1 - x_3) \geq 1 \quad (s_{c_1}) \\ & x_1 + (1 - x_2) \geq 1 \qquad\quad (s_{c_2}) \\ & x_1, x_2, x_3 \in \{0, 1\} \end{aligned}$$

$\mathcal{F}$ is satisfiable if and only if $S_{\mathcal{F}}$ have a feasible solution as determined by integer programming.

3.1 Branch-and-bound and DPLL

The most popular solution approach for linear 0/1 programming is branch and bound. It is very similar to the DPLL procedure, which also generates an enumeration tree by branching search. They differ mainly in that branch-and-bound solves the linear relaxation of the satisfiability problem (obtained by replacing the condition $x_i \in \{0, 1\}$ with ranges $0 \leq x_i \leq 1$) at each node of the enumeration tree whereas DPLL applies unit-propagation. There are several enhancements of this basic algorithm including branch-and-cut algorithm by Hooker and Fedjiki (1990) that uses cutting planes at several nodes of the branch-and-bound tree. For a survey on mathematical programming approaches for solving the satisfiability problem, see Chandru and Hooker (1999) and Hooker (2000).

3.2 Resolution and cutting planes

As mentioned previously, a modern SAT solver formulated as a proof system is equivalent to general resolution (Pipatsrisawat and Darwiche 2009). Resolution (Quine 1955; Robinson 1965) is a well known satisfiability procedure, that recursively applies the resolution rule (see Sect. 2) on two selected clauses containing two complementary literals and adding the obtained resolvents to the formula. This simple technique is complete for refutation. If the CNF formula is unsatisfiable, the resolution procedure is able to generate an empty clause in a finite but exponential number of steps in general. Applying subsumption at each step of the resolution procedure leads to a complete approach for satisfiability checking. In spite

of its exponential worst case time and spatial complexity, resolution plays an important role in current SAT solvers. As we can see later, conflict driven clause learning is based on a restricted form of resolution.

In integer programming, resolution corresponds to cutting planes (Hooker and Fedjiki 1990). A cutting plane is an inequality verified by all the 0/1 solutions of the problem. As we can see in the following example, a cutting plane can be generated by taking a nonnegative linear combination of the inequalities in the linear relaxation, including the bounds $0 \leq x_i \leq 1$, and to round up any fractions that result in the coefficient or right hand side. Such a cut is called a rank one cut.

Example 2 Let us take again the formula $\mathcal{F}$ given in the Example 1 and its corresponding set of inequalities $S_{\mathcal{F}}$.

$$\begin{array}{llll} x_1 + & x_2 + (1 - x_3) & \geq 1 & \left(\frac{1}{2}\right) \\ x_1 + (1 - x_2) & & \geq 1 & \left(\frac{1}{2}\right) \\ & (1 - x_3) & \geq 0 & \left(\frac{1}{2}\right) \\ \hline x_1 + & (1 - x_3) & \geq \frac{1}{2} & \end{array}$$

We recall that applying the resolution rule on the clauses c_1 and c_2 leads to the resolvent $\eta[x_2, c_1, c_2] = (x_1 \vee \neg x_3)$. This resolvent corresponds to the rank one cutting plane $x_1 + (1 - x_3) \geq \frac{1}{2}$ which is equivalent to $x_1 + (1 - x_3) \geq 1$ obtained by rounding up the right hand side. This resolvent is generated by combining the linear inequalities s_{c_1}, s_{c_2} with the bound $(1 - x_3) \geq 0$ where the two sides of each inequality is multiplied by $\frac{1}{2}$.

Further connexion between resolution and cutting planes can be found in Hooker and Fedjiki (1990) and Hooker (1989).

4 An historical overview of learning based approaches

Learning from conflicts appeared first in the context of constraint satisfaction problem solving (Stallman and Sussman 1977; Gasching 1979). Such approaches usually refer to non-chronological backtracking and nogood recording. A backtrack-style procedure performs a depth-first search, successively instantiating variables of the constraint network to values in order to build a solution, and backtracks when necessary, in order to escape from a conflict. In general, the current branch of the search tree might contain many irrelevant variables between the level of conflict and its real cause. Chronological backtracking may lead the solver to trashing by rediscovering the same conflict over and over again in different settings of irrelevant variables. One has to decide how far to backtrack and potentially, what to learn from the conflict. In 1977, Stallman and Sussman (1977) proposed a rule-based system for computer-aided circuit analysis. The system threads deduced facts with justifications which mention the antecedent facts and the rule used. To reduce the search space, these justifications are exploited by the system during the conflict analysis to reduce the search space. This leads to an effective control of combinatorial search which is called dependency-directed backtracking. This method is used in truth maintenance systems (Doyle 1979; McAllester 1980), and has been improved or simplified in the context of logic programming and constraint satisfaction problems by various researchers (Bruynooghe 1981; Dechter 1986, 1990).

As the learnt clauses database is of exponential size, several strategies have been proposed to reduce such spatial complexity. Most of them try to avoid such major drawback by either introducing limited and (1) relevant based learning scheme or by achieving only (2) non-chronological backtracking (without nogood recording) referred as intelligent backtracking.

For example, relevance-bounded learning (Bayardo and Miranker 1996) and conflict directed backjumping (CBJ) (Prosser 1993) belong respectively to the first and the second category. We can also cite other approaches that reduce space complexity by introducing bounds on the size of the constraints recorded (Dechter 1990; Frost and Dechter 1994), and those that record constraints of arbitrary size, but delete nogoods which become irrelevant to the current portion of the search space (Ginsberg 1993).

The efficiency of clause learning currently integrated in all the state-of-the-art SAT solvers is made possible thanks to strong learning schemes, lazy data structures and constraint database management strategies. Among these chronologically ordered contributions, we can cite the important works by Marques Silva and Sakallah (1996), Bayardo and Schrag (1997), Zhang (1997), Moskewicz et al. (2001) and Zhang et al. (2001).

5 Classical clause learning: a formal description

Clause learning plays a critical role in the success of modern complete SAT solvers. The main idea is that when a current branch leads to a conflict, clause learning aims to derive a clause that succinctly expresses the causes of conflict. Such learned clause is then used to prune the search space. Clause learning also known in the literature as conflict driven clause learning (CDCL) refers now to the most known and used First UIP learning scheme, first integrated in the SAT solver Grasp (Marques Silva and Sakallah 1996) and efficiently implemented in zChaff Moskewicz et al. (2001). Most of the SAT solvers, usually called modern SAT solvers, integrate this strong learning scheme. Theoretically, by integrating clause learning to DPLL-like procedures (Davis et al. 1962), the obtained SAT solver formulated as a proof system is as powerful as general resolution (Pipatsrisawat and Darwiche 2009). Several works have been conducted in order to improve this simple learning scheme. Such improvements, include the minimization process introduced by Sörensson and Biere (2009) and Hamadi et al. (2009a) to reduce the size of the learnt clauses, the extended learning schemes proposed in Audemard et al. (2008a) and other clauses learning variants (Pipatsrisawat and Darwiche 2008; Sabharwal et al. 2012; Hamadi et al. 2009a; Jabbour 2009).

Prior to the formal description of the above classical clause learning scheme, we first formalize the graph-based framework used to learn conflict clauses.

An implication graph (Marques Silva and Sakallah 1996) is a representation which captures the variable assignments ρ (interpretation) made during 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 denote $\overrightarrow{imp}(y)$. The clause $\overrightarrow{imp}(y)$ 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 interpretation $(\rho(x_i) = \mathit{false}, \forall i \in 1..n)$, while $\rho(y) = \mathit{true}$. When a literal y is not obtained by propagation but comes from a decision, $\overrightarrow{imp}(y)$ is undefined, which we denote for convenience $\overrightarrow{imp}(y) = \bot$.

When $\overrightarrow{imp}(y) \neq \bot$, we denote by $exp(y)$ the set $\{\overline{x} \mid x \in \overrightarrow{imp}(y) \setminus \{y\}\}$, called set of *explanations* of y. In other words if $\overrightarrow{imp}(y) = (x_1 \vee \cdots \vee x_n \vee y)$, then the explanations are the literals $\overline{x_i}$ with which $\overrightarrow{imp}(y)$ 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 $\overrightarrow{imp}(y)$ is undefined we define $exp(y)$ as the empty set. In an implication graph, the set of predecessors of a node corresponds to the set of explanations of the corresponding literal:

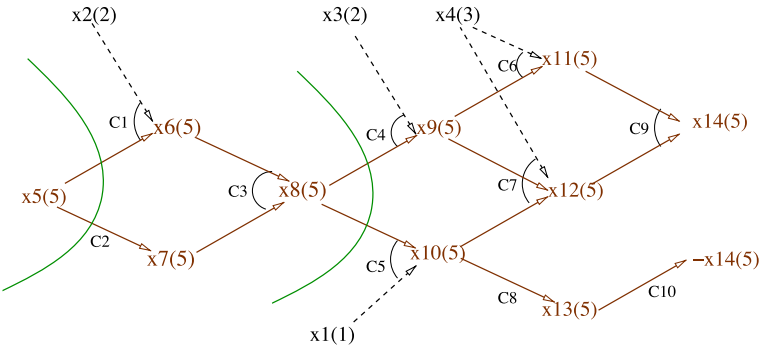

Fig. 1 Implication Graph $\mathcal{G}^{\rho}_{\mathcal{F}} = (\mathcal{N}, \mathcal{E})$

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

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

For simplicity reason, we denote $\mathcal{G}^{(\rho,exp)}_{\mathcal{F}}$ simply as $\mathcal{G}^{\rho}_{\mathcal{F}}$ in the rest of this paper.

Example 3 $\mathcal{G}^{\rho}_{\mathcal{F}}$, shown in Fig. 1 is an implication graph for the formula $\mathcal{F}$ and the partial assignment ρ given below : $\mathcal{F} \supseteq \{c_1, \ldots, c_{10}\}$

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

$\rho = \{\langle \neg x_1^1 \ldots \rangle \langle (x_2^2) \ldots x_3^2 \ldots \rangle \langle (x_4^3) \ldots \rangle \ldots \langle (x_5^5) \ldots \rangle\}$. The current decision level is 5.

5.1 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 an asserting clause derivation. Let us now formally define the concept of asserting clause, asserting clause derivation and unique implication point.

Definition 2 (*Asserting clause*) A conflict 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 denote in short $\mathcal{A}(c)$.

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

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

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

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 asserting clause derivation*) An asserting clause derivation $\pi(\sigma_k) = \langle \sigma_1, \sigma_2, \ldots \sigma_k \rangle$ is called elementary iff $\exists i < k$ s.t. $\pi(\sigma_i) \subset \pi(\sigma_k)$ is also an asserting clause derivation.

Using the Definitions 3 and 4, we can now define the concepts of Unique Implication Point (UIP) and First UIP Marques-Silva and Sakallah (1999):

Definition 5 (*Unique Implication Point* (*UIP in short*)) A node $x \in \mathcal{N}$ is a UIP iff there exists an asserting clause derivation $\langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ 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* asserting clause derivation; i.e. $\exists \pi(\sigma_k) = \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ an elementary asserting clause derivation 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 assigned at the conflict level m.

To illustrate the previous definitions, let us consider again the Example 3.

The traversal of the graph $\mathcal{G}^{\rho}_{\mathcal{F}}$ allows us to generate two asserting clauses corresponding to the two possible UIPs (see Fig. 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, first green line).

- $\sigma_1 = \eta[x_{14}, c_9, c_{10}] = (\neg x^5_{11} \vee \neg x^5_{12} \vee \neg x^5_{13})$
- $\sigma_2 = \eta[x_{13}, \sigma_1, c_8] = (\neg x^5_{10} \vee \neg x^5_{11} \vee \neg x^5_{12})$
- $\sigma_3 = \eta[x_5, \sigma_2, c_7] = (\neg x^3_4 \vee \neg x^5_9 \vee \neg x^5_{10} \vee \neg x^5_{11})$
- ....
- $\sigma_6 = \Delta_1 = \eta[x_9, \sigma_5, c_4] = (\neg x^1_1 \vee \neg x^2_3 \vee \neg x^3_4 \vee \neg x^5_8)$

As we can see, σ_6 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_8) which is assigned a the current level 5; $\pi(\Delta_1) = \langle \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_6 = \Delta_1 \rangle$ is an elementary asserting clause derivation (the intermediate clauses of $\pi(\Delta_1)$ contain more than one literal of the current decision level 5), and x_8 is a first UIP.

If we continue such a process, we obtain an additional asserting clause $\Delta_2 = (\neg x^1_1 \vee \neg x^2_2 \vee \neg x^2_3 \vee \neg x^3_4 \vee \neg x^5_5)$, corresponding to a second UIP x^5_5; which is the last UIP since it corresponds to the decision literal (see cut 2 in Fig. 1, second green line).

Property 1 (Asserting clause and backjumping) Let $\pi(c = (\alpha \vee x))$ an asserting clause derivation, and $i = jump(c)$. We can safely backjump to level i and consider the partial assignment $\rho^i \cup \{x\}$

Proof As $\alpha \subset \overline{\rho^i}$ the clause c is such that $(\mathcal{F} \wedge c)|_{\rho^i} \models_* x$, which implies that $(\mathcal{F} \wedge c)|_{\rho^i} \models x$. Furthermore c is a resolvent so $\mathcal{F}$ and $\mathcal{F} \wedge c$ are equivalent wrt. satisfiability; therefore $\mathcal{F}|_{\rho^i} \models x$. This means that at level i the assignment of x to *false* leads to a conflict. Consequently, at level i, the literal x must be assigned to $true$.

Property 2 Let $\pi(c = (\alpha \vee x))$ an asserting clause derivation, and $i = jump(c)$. Then x can be deduced by unit propagation 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 et al. (1996) and Li and Anbulagan (1997).

Grasp Marques Silva and Sakallah (1996) and zChaff Moskewicz et al. (2001) use the First UIP learning scheme, the first cut encountered by a bottom up traversal of the implication graph. Other learning schemes have been proposed by other authors. The decision scheme proposed by Zhang et al. (2001), achieve a complete traversal of the implication graph until reaching the sources of the graph, leading to an asserting clause containing only the negation of decisions literals. Relsat (Bayardo and Schrag 1997) uses the last UIP learning scheme (see Definition 7), the asserting clause have exactly one literal from the current decision level corresponding to the negation of the last decision literal. Zhang et al. (2001) present a comparison of all these and other learning schemes and conclude that First UIP is the best in practice.

An important property that any asserting clause satisfies is the 1-empowering property introduced in Pipatsrisawat and Darwiche (2010). A clause c implied by F is *1-empowering* if it allows unit propagation to derive a new implication that would be impossible to derive without c. Let us consider an example from Pipatsrisawat and Darwiche (2010). Let $F = (a \vee b \vee c) \wedge (a \vee b \vee \neg c) \wedge (a \vee \neg b \vee c) \wedge (a \vee \neg b \vee \neg c) \wedge (\neg c \vee d) \wedge (c \vee e)$. The clause $(a \vee b)$ is 1-empowering with respect to F because $F \wedge \neg b \not\models_* a$. The literal a is called an empowering literal of c with respect to F. Contrary, $(d \vee e)$, which is implied by F, is not 1-empowering (i.e., is absorbed), because $F \wedge \neg d \models_* e$ and $F \wedge \neg e \models_* d$. Let us give a formal definition of this important property.

Definition 8 (*1-empowerment* Pipatsrisawat and Darwiche (2010)) Let $(\alpha \vee l)$ be a clause where l is a literal and α is a sub-clause (a disjunction of literals). The clause is 1-empowering with respect to F via l iff

1. $F \models (\alpha \vee l)$: the clause is implied by F.
2. $F \wedge \bar{\alpha} \not\models_* l$: the literal l cannot be derived from $F \wedge \bar{\alpha}$ using unit propagation.

Any asserting clause is 1-empowering with respect to the formula (original formula augmented with the previous learnt clauses) at the time of its derivation with its asserting literal

as an empowering literal. This property plays an important role on the power of clause learning.

In Audemard et al. (2008a, b), the authors prove that the asserting clause learned using the first UIP scheme is optimal with respect to both the backjumping level and to the number of different levels it contains. Let us describe these important properties that explain why the first UIP usually considered in modern SAT solvers is more powerful than the other UIPs.

Given a clause c we denote by $levels(c)$ the measure defined as the set of different levels present in the clause c, i.e. $levels(c) = \{l(x) \mid x \in c\}$. This measure was first introduced in Audemard et al. (2008b).

Property 3 (*Optimality w.r.t. Levels*) In an asserting clause derivation $\pi(\sigma_k) = \langle \sigma_1, \ldots, \sigma_k \rangle$ we have for all $i < k, levels(\sigma_i) \subseteq levels(\sigma_{i+1})$.

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 at least another literal y such that $l(y) = l(x)$, and eventually other literals z with (1) $l(z) \in levels(\sigma_i)$ or (2) $l(z) \notin levels(\sigma_i)$. In the first case, the substitution of x in σ_i by $exp(x)$ leads to σ_{i+1} with $levels(\sigma_{i+1}) \supseteq levels(\sigma_i)$.

Property 4 (*Optimality wrt. backjumping level* (Audemard et al. 2008a, b)) Let $\pi(c = (\alpha, x))$ be an elementary asserting clause derivation and in which x is the first UIP. The back-jumping level of c is optimal: any asserting clause derivation $\pi(c')$ 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 $\pi(\sigma_k) = \langle \sigma_1, \ldots, \sigma_k = c' \rangle$ be an asserting clause derivation. Let i be the index of the first asserting clause in this proof. Because σ_i is an asserting clause we have $jump(c) = jump(\sigma_i)$ and because of Property 3 we have $jump(\sigma_i) \leq jump(\sigma_k) = jump(c')$.

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. Note that these optimality results have been proved for the first time in Audemard et al. (2008b).

5.2 Extended clauses learning

These optimality results motivated the extension of the implication graph proposed in Audemard et al. (2008a). The extended graph is obtained by considering additional arcs, called inverse arcs, derived from the satisfied clauses of the formula, which are usually ignored by conflict analysis. Traditionally, when a unit literal y is produced at a given level, the reasons (e.g. x_1, x_2) of such deduction is recorded. Such an implication is obtained from the clause $(\neg x_1 \vee \neg x_2 \vee y)$ of the original formula. Let us note that x_1 and x_2 are assigned at levels smaller or equal to the assignment level of y. Suppose that the clause $(x_1 \vee \neg y)$ belong to the formula. This mean that $x_1 = true$ is also deduced from the assignment $y = true$. In this case, an "arc" was revealed for free that could as well be used to extend the graph. The extension proposed in Audemard et al. (2008a) encode this new reason (called inverse arcs) because x_1 assigned at level i is implied by a literal y assigned at greater or equal level. This extension allows the derivation of better asserting clauses in term of size and backjumping level.

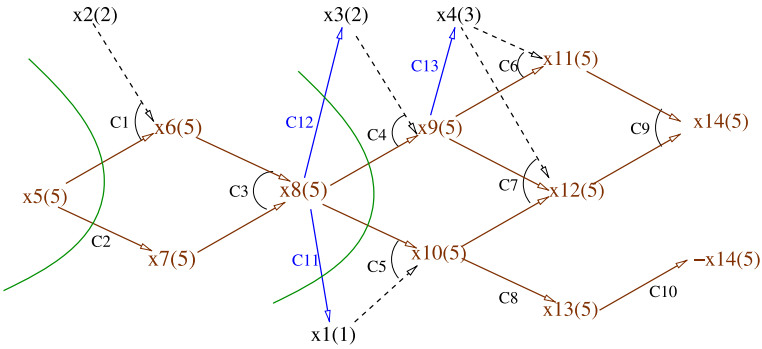

Fig. 2 Implication graph with inverse arcs : $\mathcal{G}s^{\rho}_{\mathcal{F}'} = (\mathcal{N}', \mathcal{E}')$

To explain the idea behind the proposed extension, let us consider, again, the formula $\mathcal{F}$ given in the Example 3. We consider the same partial interpretation and we define a new formula $\mathcal{F}'$ as follow : $\mathcal{F}' \supseteq \{c_1, \ldots, c_{10}\} \cup \{c_{11}, c_{12}, c_{13}\}$ where $c_{11} = (\neg x_8 \vee x_1)$, $c_{12} = (\neg x_8 \vee x_3)$ and $c_{13} = (\neg x_9 \vee x_4)$

The three added clauses are satisfied under the partial interpretation ρ. c_{11} is satisfied by x_1 assigned at level 1, c_{12} is satisfied by x_3 at level 2, and c_{13} is satisfied by x_4 at level 3. This is shown in the extended implication graph (see Fig. 2) by the blue arcs. Let us now illustrate the usefulness of the proposed extension. Let us consider again the the asserting clause Δ_1 corresponding to the classical first UIP scheme. The following strong asserting clause can be derived.

- $\sigma'_1 = \eta[x_1, \Delta_1, c_{11}] = (\neg x_3^2 \vee \neg x_4^3 \vee \neg x_8^5)$
- $\sigma'_2 = \eta[x_3, \sigma'_1, c_{12}] = (\neg x_4^3 \vee \neg x_8^5)$
- $\sigma'_3 = \eta[x_4, \sigma'_2, c_{13}] = (\neg x_9^5 \vee \neg x_8^5)$
- $\sigma'_4 = \eta[x_9, \sigma'_3, c_4] = (\neg x_3^2 \vee \neg x_8^5)$
- $\Delta_1^s = \eta[x_3, \sigma'_4, c_{12}] = (\neg x_8^5)$

In this case we backtrack to the level 0 and we assign $\neg x_8$ to *true*. Indeed $\mathcal{F}' \models \neg x_8$.

As we can see Δ_1^s subsumes Δ_1. If we continue the process we also obtain an other strong asserting clause $\Delta_2^s = (\neg x_2^2 \vee \neg x_5^5)$ which subsume Δ_2. This example illustrates how inverse arcs can be used to derive strong asserting clauses i.e. with small sizes and high buckjumping levels.

If we take a look to the clauses used in the classical implication graph $\mathcal{G}^{\rho}_{\mathcal{F}}$ (Fig. 2) 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. $\rho(x) = \mathit{true}$ and $\forall y \in exp(x)$, we have $\rho(y) = \mathit{true}$ and (2) $\forall y \in exp(x)$, $l(\neg y) \leq l(x)$. Now in the extended implication graph (Fig. 2) the added clauses satisfy property (1) and, in addition, a property (3) $\exists y \in exp(x)$ st. $l(\neg y) > l(x)$.

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_9 at level 5, we "rediscover" the deduction x_4 (which was a choice (a decision literal) at level 3). Similarly, for x_8 deduced at level 5, we "rediscover" the deductions x_3 (made at level 2) and x_1 (made at level 1).

This proposal keeps track of these re-discoveries. Note something unusual: even a decision literal (choice point) can now have incoming arcs. This is the case for x_4 for instance.

It is important to note that the proposed extension can leads to implications graphs that are not acyclic. In Audemard et al. (2008a), a new concept of joint inverse arc to extend the classical conflict resolution is proposed. The extended learning scheme, exploit classical learning to generate the first asserting clause, then inverse arcs are used to eliminate the literals with higher levels. Some additional conditions are used to avoid cycles in the extended implication graph. For more details, on the practical integration to SAT solvers, we refer the reader to Audemard et al. (2008a).

5.3 Learning Bi-asserting clauses

Another Clause learning scheme is also proposed by Pipatsrisawat and Darwiche (2008). The authors exploit a new class of conflict clauses, called bi-asserting clauses, which is a relaxation of asserting clauses. It is defined as conflict clause with two literals from the current decision level.

Definition 9 [*Bi-Asserting clause* (Pipatsrisawat and Darwiche 2008)] A clause $c = (\alpha \vee x \vee y)$ is called a Bi-Asserting clause if $\rho(c) = \mathit{false}, l(\alpha) < m$ and $l(y) = l(x) = m$.

In Pipatsrisawat and Darwiche (2008), the authors propose an efficient way to derive empowering bi-asserting clauses, but only with respect to the clauses used in its derivation. Indeed, the 1-empowerment property is checked on the fly, by simply checking if the resolution derivation of such clause involves a merging resolution (Andrews 1968). A resolution between two clauses $(a \vee \alpha)$ and $(\neg a \vee \beta)$ is a merging resolution if $\alpha \cap \beta \neq \emptyset$.

This new class of bi-asserting clauses tends to be much shorter and have smaller assertion levels than asserting clauses for the same conflict. The exploitation of such kind of clauses is particularly suitable for solving unsatisfiable SAT instances.

Recently Jabbour et al. (2013), proposed another interesting approach to derive a new class of bi-asserting clauses. These clauses are obtained by traversing the implication graph separately from x and $\neg x$. These new kinds of bi-asserting clauses are much shorter and tend to induce more implications than the classical bi-asserting clauses.

Let us, illustrate these new kind of bi-asserting clauses using a simple example. Let $\mathcal{F}$ be a CNF formula, ρ a partial assignment. Assume that we have an implication graph $\mathcal{G}^{\rho}_{\mathcal{F}}$ associated to $\mathcal{F}$ and ρ. For simplicity reason, Fig. 3 depicts the implication graph restricted to the sub-graph induced by the nodes between the conflict and the first UIP.
Assume that the conflict level is 5.

By traversing the implication graph separately from x_{14} and $\neg x_{14}$ until the first UIP x_1, we derive the new bi-asserting clauses $\pi_1, \pi_2, \pi_3, \pi_4$ as follows:

- $\sigma_1 = \eta[x_{10}, c_{13}, c_9] = \neg x_{15} \vee \neg x_{11} \vee \neg x_8 \vee x_{14}$
- $\pi_1 = \eta[x_{11}, \sigma_1, c_{10}] = \neg x_{15} \vee \neg x_8 \vee x_{14}$
- $\sigma_2 = \eta[x_4, c_7, c_3] = \neg x_{17} \vee \neg x_5 \vee \neg x_2 \vee x_8$
- $\pi_2 = \eta[x_5, \sigma_2, c_4] = \neg x_{17} \vee \neg x_2 \vee x_8$
- $\sigma_3 = \eta[x_{12}, c_{14}, c_{11}] = \neg x_{13} \vee \neg x_9 \vee \neg x_{14}$
- $\pi_3 = \eta[x_{13}, \sigma_3, c_{12}] = \neg x_{15} \vee \neg x_9 \vee \neg x_{14}$
- $\sigma_4 = \eta[x_6, c_8, c_5] = \neg x_3 \vee \neg x_7 \vee x_9$
- $\pi_4 = \eta[x_7, \sigma_4, c_6] = \neg x_{16} \vee \neg x_3 \vee x_9$

The new bi-asserting clauses $\pi_1, \pi_2, \pi_3, \pi_4$ form a set of connected clauses. More precisely, the clause π_2 is connected with π_1 by the variable x_8, π_1 is connected with π_3 by the variable x_{14} and π_3 is connected with π_4 by the variable x_9. As we can observe the set of bi-asserting clauses form a chain of connected clauses. It is important to note that the classical bi-asserting clauses that can be obtained from the implication graph of Fig. 3 are:

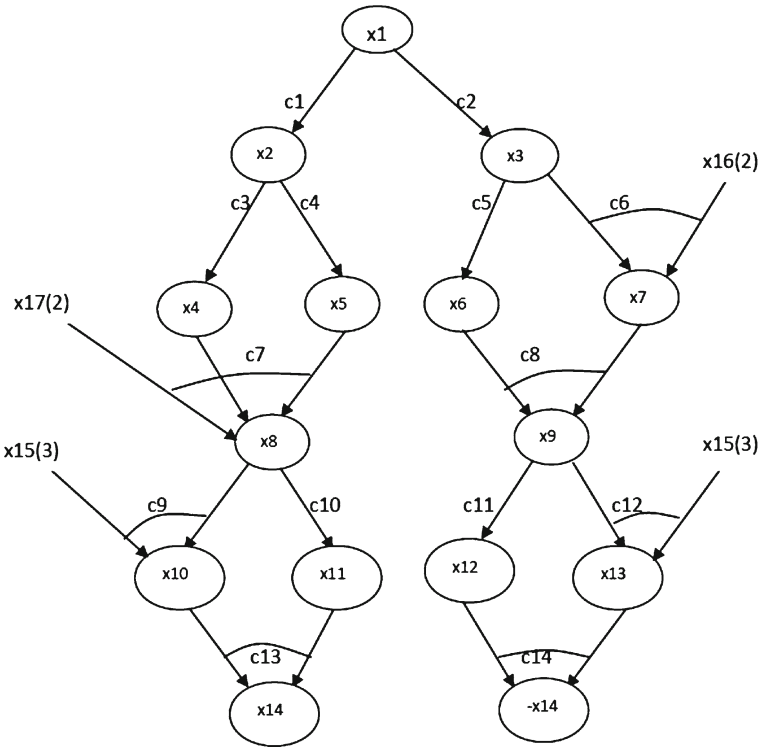

Fig. 3 Sub Implications Graph $\mathcal{G}^{\rho}_{\mathcal{F}} = (\mathcal{N}, \mathcal{E})$

- $\pi'_1 = \neg x_{15} \vee \neg x_8 \vee \neg x_9$,
- $\pi'_2 = \neg x_{15} \vee \neg x_{17} \vee \neg x_2 \vee \neg x_9$,
- $\pi'_3 = \neg x_{15} \vee \neg x_{16} \vee \neg x_8 \vee \neg x_3$,
- $\pi'_4 = \neg x_{15} \vee \neg x_{17} \vee \neg x_{16} \vee \neg x_2 \vee \neg x_3$.

Let us note that by classical clause learning (First UIP scheme), one can derive the asserting clause $(\neg x_{15} \vee \neg x_{17} \vee \neg x_{16} \vee \neg x_1)$. The back-jumping level is 3.

In Fig. 4, we show that the original implications graph (Fig. 3) can be rewritten more compactly. Let us now consider that the new bi-asserting clauses π_1, π_2, π_3 and π_4 are added to the learnt clauses database and assume that at level 3 the literal x_2 is assigned to $true$. It is easy to see that all the literals x_8, x_{14}, $\neg x_9$, $\neg x_3$ and $\neg x_1$ (see Fig. 4) together with the literals x_4, x_5, x_{10}, x_{11} are implied by unit propagation. These last unit propagated literals are implied thanks to the original clauses of the formula c_3, c_4, c_9 and c_{10}. However, if we consider the original implication graph (Fig. 3), we derive by unit propagation the same set of literals except $\neg x_9$, $\neg x_3$ and $\neg x_1$.

Suppose now that all the classical bi-asserting clauses π'_1, π'_2, π'_3 and π'_4 are added to the learnt clauses database, assigning x_2 to $true$ at level 3 leads to the same set of implications by unit propagation. However, if we decide to assign x_{14} to $true$ at level 3, using the new bi-asserting clauses we derive $\neg x_9$, $\neg x_3$ and $\neg x_1$ while with the classical bi-asserting clauses no literal is implied by unit propagation. Another important difference that can be made is that the derivation of all possible classical bi-asserting clauses is quadratic in the worst case, while the new bi-asserting clauses can be derived in linear time (see the Fig. 4). Consequently,

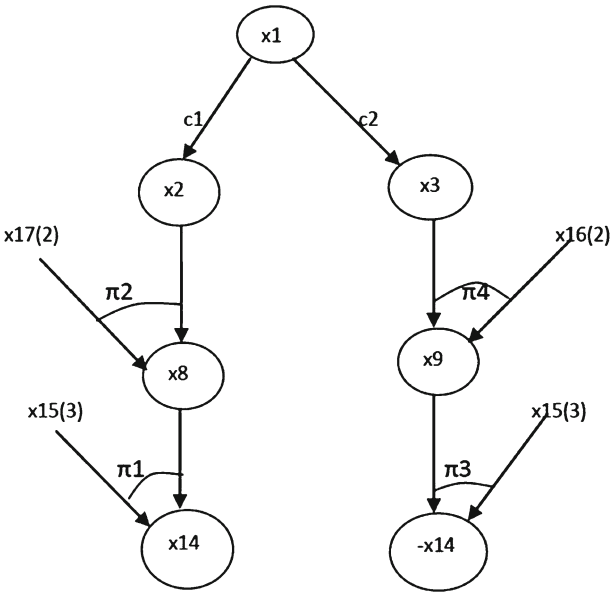

Fig. 4 New and compact implication graph obtained from $\mathcal{G}^{\rho}_{\mathcal{F}} = (\mathcal{N}, \mathcal{E})$ using the new bi-asserting clauses

searching for all classical bi-asserting clauses takes more time than for the new proposed bi-asserting clauses. Moreover, the new bi-asserting clauses are much shorter than classical ones.

From this illustrative example, we see that adding the new bi-asserting clauses to the clauses database allows to derive more implications than with classical bi-asserting clauses. We can observe that the proposed bi-asserting clauses establish a link between the asserting literal and the conflicting literals x_{14} and $\neg x_{14}$. In Jabbour et al. (2013), the authors present formally such separate conflict analysis to derive bi-asserting clauses. From the experiments presented by the authors, integrating such clauses in SAT solvers, improve their performance particularly on crafted instances.

5.4 Learning back-clauses

As mentioned previously, First Unique Implication Point (First UIP) is the traditional learning scheme. Obviously, an implications graph might contains several unique implication points from the first to the last UIP corresponding to the asserting clause involving the last decision literal of the current branch. We argued in the previous section that the first UIP is better than others with respect to both the buckjumping level and to the number of different levels involved in the clause. However, deriving other clauses connecting all these implications points, will enable unit propagation to make all inferences that all traditional asserting clauses based on all the UIPs. In Sabharwal et al. (2012), the authors proposed a learning scheme that record these kind of clauses, called back-clauses. The idea of learning back-clauses during search is first introduced in Marques Silva and Sakallah (1996). In Sabharwal et al. (2012), it is also shown that adding back-clauses is stronger than adding all asserting clauses corresponding to all UIPs.

Let us illustrate this notion using the example depicted in Fig. 1 admitting only two UIPs. From the example we can derive two asserting clauses, $\Delta_1 = (\neg x_1^1 \vee \neg x_3^2 \vee \neg x_4^3 \vee \neg x_8^5)$ and

$\Delta_2 = (\neg x_1^1 \vee \neg x_2^2 \vee \neg x_3^2 \vee \neg x_4^3 \vee \neg x_5^5)$ corresponding to the first and last UIPs respectively. A back-clause, can be derived by resolution starting from the first UIP node (x_8), leading to the clause $B = (\neg x_2^2 \vee \neg x_5^5 \vee x_8^5)$. We observe that the back-clause B contains two literals from the current conflict level, it can be seen as a non conflicting bi-asserting clause, satisfied by the current partial interpretation ($\rho(x_8) = true$). Let us now, explain why recording such back-clauses, can leads to more propagation at the buckjumping level 3 ($jump(\Delta_1) = 3$). Suppose that, after such conflict, we record both the classical asserting clause Δ_1 corresponding to the first UIP and the back-clause B. At the backjumping level 3, the asserting literal $\neg x_8$ is implied, thanks to the asserting clause Δ_1. By adding the back-clause B, the literal $\neg x_5$ is propagated. This last literal can not be deduced by unit propagation even if we record all the asserting clauses Δ_1 and Δ_2. Consequently, connecting the different asserting clauses using back clauses, allows further propagations.

5.5 Learning for dynamic subsumption

Usually, clauses learning is used to analyse the reason of the conflict. We now describe, an original approach introduced in Hamadi et al. (2009a) that exploit clause learning to dynamically subsume some clauses of the formula. The approach exploits 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.

Let us, illustrate some of the main features of this approach using the Example 3 and the implication graph $\mathcal{G}_{\mathcal{F}}^{\rho}$ (Fig. 1). The asserting clause derivation leading to the asserting clause Δ_1 is described as follows:
$\pi(\Delta_1) = \langle \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_7 = \Delta_1 \rangle$

- $\sigma_1 = \eta[x_{14}, c_9, c_{10}] = (\neg x_{11}^5 \vee \neg x_{12}^5 \vee \neg x_{13}^5)$
- $\sigma_2 = \eta[x_{13}, \sigma_1, c_8] = (\neg x_{10}^5 \vee \neg x_{11}^5 \vee \neg x_{12}^5)$
- $\sigma_3 = \eta[x_{12}, \sigma_2, c_7] = (\neg x_4^3 \vee \neg x_9^5 \vee \neg x_{10}^5 \vee \neg x_{11}^5)$
- $\sigma_3 = \eta[x_{11}, \sigma_2, c_7] = (\neg x_4^3 \vee \neg x_9^5 \vee \neg x_{10}^5 \vee \neg x_{11}^5)$
- $\sigma_4 = \eta[x_{10}, \sigma_2, c_7] = (\neg x_4^3 \vee \neg x_9^5 \vee \neg x_{10}^5) \subset c_7$ (subsumption)
- $\ldots$
- $\sigma_7 = \Delta_1 = \eta[x_9, \sigma_5, c_4] = (\neg x_1^1 \vee \neg x_3^2 \vee \neg x_4^3 \vee \neg x_8^5)$

As we can see the asserting clause derivation $\pi(\Delta_1)$ includes the resolvent $\sigma_4 = (\neg x_4 \vee \neg x_9 \vee \neg x_{10})$ which subsumes the clause $c_7 = (\neg x_4 \vee \neg x_9 \vee \neg x_{10} \vee x_{12})$. Consequently, the literal x_{12} is eliminated from the clause c_7. In general, the resolvent σ_4 can subsume other clauses from the implication graph that include the literals $\neg x_4$, $\neg x_9$ and $\neg x_{10}$.

Let us now give the formal presentation of of this dynamic subsumption approach.

Definition 10 (*F-subsumption modulo UP*) Let $c \in \mathcal{F}$. c is $\mathcal{F}$-subsumed modulo unit propagation iff $\exists c' \subset c$ such that $\mathcal{F}|_{\vec{c'}} \models^* \perp$

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.

Subsuming clauses during search might be time consuming. To reduce the computational cost, the authors restrict the subsumption checks to the intermediate resolvents σ_i and to the clauses of the form $\overrightarrow{imp}(y)$ used to derive them (clauses encoded in the implication graph).

Definition 11 Let $\mathcal{F}$ be a formula and $\pi(\sigma_k) = \langle \sigma_1 \dots \sigma_k \rangle$ an asserting clause derivation. For each $\sigma_i \in \pi$, we define $\mathcal{C}_{\sigma_i} = \{\overrightarrow{imp}(y) \in \mathcal{F} | \exists j \leq i \text{ st. } \sigma_j = \eta[y, \overrightarrow{imp}(y), \sigma_{j-1}]\}$ as the set of clauses of $\mathcal{F}$ used for the derivation of σ_i.

Property 5 Let $\mathcal{F}$ be a formula and $\pi(\sigma_k) = \langle \sigma_1, \sigma_2, \dots, \sigma_i, \dots, \sigma_k \rangle$ 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 $\sigma_{i+1} = \eta[y, \overrightarrow{imp}(y), \sigma_i]$ where $\neg y \in \sigma_i$, we have $\sigma_i \not\subset \overrightarrow{imp}(y)$. 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 6 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 Sect. 5). As σ_i subsumes c ($\sigma_i \subseteq c$), then $\mathcal{C}_{\sigma_i} \models^* c$.

The Property 6 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 5 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 $\pi(\sigma_k) = \langle \sigma_1, \dots, \sigma_i, \dots, \sigma_k \rangle$ be an asserting resolution derivation. For each resolvent $\sigma_i \in \pi(\sigma_k)$, 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 $\pi(\sigma_k)$, 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 7 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.

Proof 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 7 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 3 (see also Fig. 1). Let $\pi(\sigma_k) = \langle \sigma_1, \dots, \sigma_6 \rangle$ where $\sigma_6 = (\neg x_1 \vee \neg x_2 \vee \neg x_4 \vee \neg x_8)$. We have $\mathcal{C}_{\sigma_6} = \{c_4, c_5, c_6, c_7, c_8, c_9, c_{10}\}$ and $\mathcal{P}_{\sigma_6} = \{c_4, c_5, c_7, c_8\}$. Indeed, from the nodes associated to the clause c_6 we only have one path to the node x_{14}. Consequently, the clause c_6 might be discarded from the set of clauses to be checked for subsumption. Similarly, the clause c_9 and c_{10} are only linked to one node of the set $\{x_{14}, \neg x_{14}\}$. Then c_9 and c_{10} are not considered for subsumption tests.

Property 8 Given an asserting clause derivation $\pi(\sigma_k) = \langle \sigma_1, \dots, \sigma_k \rangle$. 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 $\mathcal{P}_{\sigma_k} \subset \mathcal{C}_{\sigma_k}$.

To design an efficient dynamic simplification technique, one need to balance the run time cost and the quality of the simplification. In Hamadi et al. (2009a), a restriction of the general dynamic subsumption scheme, called dynamic subsumption 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 $\sigma_i = \eta[y, c, \sigma_{i-1}]$, subsumption checks are only performed between σ_i and c. The following property gives a sufficient condition under which y can be removed form c

Property 9 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 dynamic subsumption approach is done with negligible additional cost. Indeed, by using a simple counter during the conflict analysis procedure, one can verify the sufficient condition given in the Property 9 in constant time. Indeed, at each step of the asserting clause derivation, the next resolvent σ_i is generated 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 only need to 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 $\sigma_i = \eta[y, c, \sigma_{i-1}]$.

5.6 Learning from success

In Jabbour (2009), Said Jabbour showed that new explanations for implied (unit propagated) literals can be generated even if a partial assignment does not lead to a conflict. Let us explain this issue, using an example.

Example 4 Let $\mathcal{F}'$ the new formula obtained from $\mathcal{F}$ (see Example 3) by removing the clause c_{10} and adding the clause $c_{11} = (\neg x_2 \vee x_8 \vee x_{14})$. The implication graph $\mathcal{G}^{\rho}_{\mathcal{F}'}$ is the a sub-graph of $\mathcal{G}^{\rho}_{\mathcal{F}}$ (see Fig. 1) induced by the set of nodes $\mathcal{N} \backslash \{\neg x_{14}, x_{13}\}$. Clearly, the partial assignment ρ does not lead to a conflict at the decision level 5. We can remark, that the clause c_{11} is not recorded in the implication graph $\mathcal{G}^{\rho}_{\mathcal{F}'}$ as it contains two literals x_8 and x_{14} assigned to $true$. Moreover, the literal x_{14} is assigned by unit propagation at level 5 due to the clause c_9 and ρ. Let us now illustrate how one can deduce a new reason (clause) asserting that x_{14} can be assigned at level less than 5. A traversal of the implication graph starting from the node x_{14}, leads to the clause : $\Delta = (\neg x_1^1 \vee \neg x_3^2 \vee \neg x_4^3 \vee \neg x_8^5 \vee x_{14}^5)$.

The clause Δ contains only two literals $\neg x_8$ and x_{14} from the current decision level 5 and all its literals are assigned $false$ except x_{14} which is assigned to $true$. Now, if we apply one additional resolution step between Δ and $c_{11} = (\neg x_2 \vee x_8 \vee x_{14}) \in \mathcal{F}'$, we obtain a new asserting clause $\Delta' = (\neg x_1^1 \vee \neg x_2^2 \vee \neg x_3^2 \vee \neg x_4^3 \vee x_{14}^5)$. This last clause expresses that the literal x_{14} assigned at level 5 can be assigned at level 3. Indeed, as $\mathcal{F}' \models \Delta'$ and the literals $\neg x_1$, $\neg x_2$, $\neg x_3$ and $\neg x_4$ are assigned false at levels less than 3, we deduce that the literal x_{14} assigned at level 5 can be assigned by unit propagation at level 3 thanks to the new reason Δ'.

In Jabbour (2009), this idea is used to reorder the current partial assignment thanks to the new derived better reasons for unit propagated literals. These reasons allows to assign such literals at previous levels.

6 Modern SAT solvers

In this section we first give a *brief description* of the other components of modern SAT solvers not covered in the previous sections. Then we present the CDCL-based SAT algorithm. This will give to the reader the whole picture about modern SAT solvers including the learning component. For a more detailed description, the reader can see Sais (2008) and Biere et al. (2009).

Modern SAT solvers are especially efficient with structured SAT instances coming from industrial applications. On application domains, Gomes et al. (2000) 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. Variable State Independent Decaying Sum (VSIDS) and other variants of *activity-based heuristics*, on the other hand, were introduced to avoid thrashing and to focus the search on 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. The activity of the variables are updated during the learning process, allowing VSIDS to always select the most active variable as the new decision point. As the number of learned clauses might be exponential in the worst case, several *deletion strategies* are proposed to keep a learnt clauses database of manageable size. On the other hand, to allow solving large SAT instance, the introduction of *lazy data-structure*, called watched literals is crucial for the efficiency of modern SAT solvers. Reducing the size of the input formula in a *preprocessing* step is another important component, usually integrated in all the state-of-the-art SAT solvers.

6.1 Restarts strategies

Restarts policy were introduced for the first time as early as 1994 in Crawford and Baker (1994). When a solver restarts, the current partial interpretation is left and the search starts again at the root of the search tree, while maintaining several informations cumulated from the previous runs (e.g. learned clauses, variables activities). Restarts are presented by Gomes et al. (1998) to eliminate the heavy tailed phenomena observed on many families of SAT instances. Indeed, this observed phenomena demonstrates that on many instances different variables ordering might lead to dramatic performance variation of a given SAT solver. Restarts aim to eliminate the long runs by increasing the probability to fall on the best variables ordering. In modern SAT solvers, restarts dive the search to the most actives variables and aims to compact the assignment stack and to improve the order of assumptions.

Different restart policies have been previously presented. Most of them are static, and the cutoff value follows different evolution schemes (e.g. arithmetic, geometric, Luby). To ensure the completeness of the SAT solver, in all these restarts policies, the cutoff value in terms of the number of conflicts increases over the time. The performance of these different policies clearly depends on the considered SAT instances. More generally, rapid restarts [e.g. Luby et al. (1993)] perform well on industrial instances, however on hard SAT instances

slow restarts is more suitable. Generally, it is hard to say in advance which policy should be used on which problem class (Huang 2007). More recently, several dynamic and adaptive restarts policies are proposed, the cutoff value is computed during search using observed informations on the running behavior of the solver (Biere 2008; Pipatsrisawat and Darwiche 2009; Hamadi et al. 2009b).

6.2 Activity based heuristics

Branching heuristics aim to select the next variable to assign at each node of the search tree. Such variable selection strategy induces an assignment ordering which is known to be crucial for the efficiency of SAT solvers. Many heuristics have been proposed in the literature, most of them are based on syntactical arguments such as the number of occurrences of the variables, the size of the clauses, etc. One of the most popular strategies, is the dynamic largest individual sum (DLIS) heuristic (Marques-silva 1999), that choose the variable and the assignment that directly satisfy the largest number of clauses. Other sophisticated heuristic functions have been proposed, one can cite the Jeroslow and Wang (1990) heuristic and many other variants which select the variable occurring most frequently in short clauses. All these heuristics need to maintain such arguments during search, which can be very costly on large SAT instances.

Recently, the Chaff SAT solver proposed the use of the variable state independent decaying sum (VSIDS) heuristic (Moskewicz et al. 2001). An activity is associated to each variable. Each time a variable occurs in a recorded learnt conflict clause, its activity is increased. Additionally, to give more importance to the variables involved in the most recent learnt clauses, the activity of all the variables are multiplied by a constant less than 1. MiniSAT uses similar ideas (Een and Sörensson 2005). However, the activity update is not limited to the variables occurring in the final learnt conflict clause, but considers all the variables occurring in any clause used in the conflict analysis.

Activity based heuristics take their origin from the principle used in the break-out local search method (Morris 1993), where the score of falsified clauses is increased during the search and used to escape from local minima. Brisoux et al. (1999), proposed an extension of the clause-weighting scheme to DPLL-like techniques. Every time a conflict is reached, the score of the falsified clauses is increased. The next variable is chosen according to its occurrence in most falsified clauses. The MiniSAT heuristic (Een and Sörensson 2005) updates the activity of the variables encountered between the conflict side and the first UIP, while in Brisoux et al. (1999), only the variables of the two last clauses at the origin of the contradiction are updated. VSIDS and MiniSAT heuristics are cheap to maintain and they are shown to be very effective on a variety of problems.

6.3 Lazy data structures—watched literals

The two watched literals scheme introduced in Zchaf Moskewicz et al. (2001), now integrated in most SAT solvers allows efficient unit propagation. This lazy data structure is an improvement of the one introduced earlier by Zhang (1997) in the solver Sato. The idea behind two watched literals, is to maintain an invariant for each active clause by marking or "watching" two special literals not assigned the value $false$. A clause containing such two watched literals cannot be involved in unit propagation. Such invariant is maintained with a very small overhead, while no update is needed in case of backtracking. This leads to a substantial saving of computation time. For more details on how such invariant is maintained see Moskewicz et al. (2001) and Een and Sörensson (2005).

6.4 Learnt database deletion strategies

Conflict driven clause learning is recognized as a powerful technique both in theory and practice. However, the set of clauses that can be derived from conflicts is of exponential size in the worst case. In practice, the number of such clauses can sometimes be greater than the number of clauses of the original formula. Several strategies have been designed to cope with this combinatorial explosion problem. To maintain a relevant learnt clauses database of polynomial size - and consequently a unit propagation of reasonable cost - all these strategies dynamically reduce the learnt database by deleting clauses considered to be irrelevant to the next search steps. The most popular strategy considers a learnt clause as irrelevant if its activity or its involvement in conflict analysis is marginal. Several clauses deletion strategies have been integrated in different SAT solvers (Marques-Silva and Sakallah 1999; Moskewicz et al. 2001; Eén and Sörensson 2003). In most cases, these strategies prefer to keep a learnt clauses database, with smaller, recent and active clauses. In Audemard et al. (2011), a new dynamic management policy of the learnt clauses database is introduced. It is based on a dynamic freezing and activation principle of the learnt clauses. At a given search state, it activates the most promising learnt clauses while freezing irrelevant ones. In this way, previously learned clauses can be discarded for the current step, but may be activated again in future steps of the search process. This policy tries to exploit pieces of information gathered from the past to deduce the relevance of a given clause for the remaining search steps.

In Guo et al. (2014), we proposed two new measures to predict the quality of learned clauses. The first one is based on the backtracking level (BTL), while the second is based on a notion of distance, defined as the difference between the maximum and minimum assignment levels of the literals involved in the learned clause.

More recently, Jabbour et al. (2014) revisited size-bounded learning strategies proposed more than fifteenth years ago (Bayardo and Schrag 1997; Bayardo and Miranker 1996; Marques Silva and Sakallah 1996) and show that these strategies remains a competitive with the state-of-the-art ones. They also demonstrate that adding randomization to size bounded learning is a nice way to achieve controlled diversification. It allows to favor short clauses, while maintaining a small fraction of large clauses necessary for deriving resolution proofs on some SAT instances.

Finally, size-bounded clause sharing strategies are also considered in several portfolio and divide and conquer based parallel SAT solvers (e.g. Hamadi et al. 2009b; Hamadi and Wintersteiger 2013; Hamadi et al. 2008, 2011).

6.5 Preprocessing

Many preprocessing techniques have been proposed over the years. In this section, we particularly limit our presentation to the most popular and useful ones. Among them, `SatElite` (Eén and Biere 2005) is clearly the most efficient and used preprocessing technique. Once again, it is based on variable elimination through the resolution rule. `SatElite` is an extension of NIVER (non increase variable elimination by resolution) originally proposed in Subbarayan and Pradhan (2004). The extension includes variable elimination by substitution using boolean functions, subsumption and self-subsuming resolution. As mentioned by Subbarayan and Pradhan (2004), 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 Zhang (2005). Another original technique called learning for dynamic subsumption is proposed in Hamadi et al. (2009a). At each conflict, it exploits clause learning to subsume clauses of the original formula and of the learnt clauses database. Designing strong preprocessing techniques able to greatly reduce the size of the formula remains an important research issue [see for example recent works by Heule et al. (2010, 2011), and Piette et al. (2008)].

Algorithm 1: CDCL-based SAT solver

Input: CNF formula $\mathcal{F}$
Output: A model of $\mathcal{F}$ or unsat if $\mathcal{F}$ is unsatisfiable

```
2  D ← ∅;                                          /* decision literals */
3  Δ ← ∅;                                  /* learnt clauses database */
4  while (true) do
5  |  S ← F ∧ Δ ∧ D;
6  |  if (S* = ⊥) then
7  |  |  if ((D = ⟨⟩)) then return unsat;
8  |  |  α ← learningFromConflict(S) ;
9  |  |  m ← assertion level of α;
10 |  |  D ← D^m;
11 |  |  Δ = Δ ∧ α ;
12 |  else
13 |  |  if (timeToReduce()) then
14 |  |  |  Δ ← reduceLearntDB();
15 |  |
16 |  |  if (timeToRestart()) then
17 |  |  |  D ← ∅;
18 |  |  |  S ← F ∧ Δ ∧ D;
19 |  |  ℓ ←decide();
20 |  |  if (ℓ = null) then
21 |  |  |  return D;
22 |  |  D ← D ∧ ℓ;
23 |  |
24
```

6.6 CDCL-based SAT algorithm

Let us now give a general formulation of a CDCL-based SAT solver (Algorithm 1) (Pipatsrisawat and Darwiche 2009). As we can see the algorithm starts with an empty set of decision literals and an empty learnt clauses database (lines 1 and 2). At each iteration, the current formula $\mathcal{S}$ is closed under unit propagation (line 5). In case of conflict (lines 6–10), if the set of decisions literals is empty (line 6), the formula is answered unsatisfiable, otherwise a new asserting clause is derived by learning from conflict (line 7). In this last case, the algorithm backtracks to the assertion level m (line 9) and adds the asserting clause to the learnt clauses database (line 10). If no conflict occurs (lines 12–21), the algorithm either reduces the learnt database (lines 12–13) and/or restarts the search process (lines 15–17) using reduction and restarts policies respectively. Then a new decision literal is selected according to

VSIDS heuristics and polarity functions (Een and Sörensson 2005; Pipatsrisawat and Darwiche 2007). The chosen literal is then added to the set of decisions with its associated level (line 21). If all the variables are assigned (line 19–20) a model is found and the formula is answered satisfiable.

Modern SAT solvers are based on the general scheme depicted in Algorithm 1. Several variants of this basic scheme are designed and implemented. For more details about these solvers and their performances, see the annual SAT competition or SAT race evaluation (http://www.satcompetition.org/).

7 General conclusion

In this survey, we presented learning from conflict, one of the most powerful reasoning technique in propositional satisfiability. Learning from conflict is a general concept used to explain failures or conflicts encountered during search. This explanation, derived by conflict analysis, and generally expressed as a new constraint, is usually used to dynamically avoid future occurrences of similar situations. After a brief discussion of most of the related works in SAT and other domains, we formally described clause learning currently integrated in most of the state-of-the-art modern SAT solvers. To give a more comprehensive view, other important components of SAT solvers such as restarts, activity based heuristics, learnt clauses database deletion strategies, lazy data structures and preprocessing are also sketched. Finally a general SAT algorithm is given to illustrate the interaction between all these strong and powerful components.

The proximity between propositional satisfiability and 0/1 linear programming, suggests that cross-fertilization remains an important issue that will benefits to both domains. We hope that this survey will give to the operation research community a comprehensive vision about clause learning and satisfiability solving.

References

Andrews, P. B. (1968). Resolution with merging. *Journal of the ACM*, *15*(3), 367–381.

Audemard, G., Bordeaux, L., Hamadi, Y., Jabbour, S., & Sais, L. (2008). A generalized framework for conflict analysis. In *Proceedings of the eleventh international conference on theory and applications of satisfiability testing (SAT'2008)*, (pp. 21–27).

Audemard, G., Bordeaux, L., Hamadi, Y., Jabbour, S., & Sais, L. (2008). A generalized framework for conflict analysis. Technical report MSR-TR-2008-34, Microsoft research.

Audemard, G., Lagniez, J., Mazure, B., & Sais, L. (2011). On freezing and reactivating learnt clauses. In *Proceedings of the 14th international conference on theory and applications of satisfiability testing (SAT'2011)*, (pp. 188–200).

Bayardo, R. J., & Miranker, D. P. (1996). A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem. In *Proceedings of the thirteenth national conference on artificial intelligence (AAAI'96)*, (pp. 298–304).

Bayardo, R. J., Jr., & Schrag, R. C. (1997) Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the fourteenth national conference on artificial intelligence (AAAI-97)*, (pp. 203–208).

Biere, A., Biere, A., Heule, M., van Maaren, H., & Walsh, T. (2009). *Handbook of satisfiability: Volume 185 Frontiers in artificial intelligence and applications*. Amsterdam: IOS Press.

Biere, A. (2008). Adaptive restart strategies for conflict driven sat solvers. In *Proceedings of the 11th international conference on theory and applications of satisfiability testing (SAT'08)*, (pp. 28–33).

Brisoux, L., Gregoire, E., & Sais, L. (1999). Improving backtrack search for SAT by means of redundancy. In *Proceedings of the international syposium on methodologies for intelligent systems (ISMIS'99)*, (pp. 301–309).

Bruynooghe, M. (1981). Solving combinatorial search problems by intelligent backtracking. *Information Processing Letters*, *12*(1), 36–39.
Chandru, V., & Hooker, J. (1999). *Optimization methods for logical inference*. Hoboken: Wiley.
Crawford, J. M., & Baker, A. B. (1994). Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of the twelth national conference on artificial intelligence (AAAI'94)*, (pp. 1092–1097).
Davis, M., Logemann, G., & Loveland, D. W. (1962). A machine program for theorem-proving. *Communications of the ACM*, *5*(7), 394–397.
Dechter, R. (1986). Learning while searching in constraint-satisfaction-problems. In *Proceedings of the fifth national conference on artificial intelligence (AAAI-86)*, (pp. 178–185).
Dechter, R. (1990). Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, *41*(3), 273–312.
Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence*, *12*(3), 231–272.
Dubois, O., André, P., Yassine Boufkhad, Y., & Carlier, Y. (1996). Second DIMACS implementation challenge: Cliques, coloring and satisfiability, In *DIMACS series in discrete mathematics and theoretical computer science, chapter SAT versus UNSAT* (Vol. 26, pp. 415–436). American Mathematical Society.
Een, N., & Sörensson, N. (2005). Minisat—a sat solver with conflict-clause minimization. In *Proceedings of the eighth international conference on theory and applications of satisfiability testing (SAT'05)*.
Eén, N., & Biere, A. (2005). 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)*, (pp. 61–75).
Eén, N., & Sörensson, N. (2003). An extensible sat-solver. In *Proceedings of the sixth international conference on theory and applications of satisfiability testing (SAT'03)*, (pp. 502–518).
Frost, D., & Dechter, R. (1994). Dead-end driven learning. In *Proceedings of the twelth national conference on artificial intelligence (AAAI-94)*, (pp. 294–300).
Gasching, J. (1979).*Performance measurement and analysis of certain search Algorithms*. PhD thesis, Department of Computer science, Carnegie Mellon University.
Ginsberg, M. L. (1993). Dynamic backtracking. *Journal of Artificial Intelligence Research (JAIR)*, *1*, 25–46.
Gomes, Carla P., Selman, Bart, Crato, Nuno, & Kautz, Henry A. (2000). Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, *24*(1/2), 67–100.
Gomes, C. P., Selman, B., & Kautz, H. A. (1998). Boosting combinatorial search through randomization. In *Proceedings of the fifteenth national conference on artificial intelligence (AAAI-98)*, (pp. 431–437).
Guo, L., Jabbour, S., Lonlac, J., & Sais, L. (2014). Diversification by clauses deletion strategies in portfolio parallel SAT solving. In *Proceedings of the 26th IEEE international conference on tools with artificial intelligence (ICTAI'2014), Limassol, Cyprus, November 10–12, 2014*, (pp. 701–708).
Hamadi, Y., Jabbour, S., Piette, C., & Sais, L. (2011). Deterministic parallel DPLL. *Journal on Satisfiability, Boolean Modeling and Computation—JSAT*, *7*(4), 127–132.
Hamadi, Y., Jabbour, S., & Sais, L. (2008). ManySAT: Solver description. Technical report MSR-TR-2008-83, Microsoft research.
Hamadi, Y., Jabbour, S., & Sais, L. (2009a). Learning for dynamic subsumption. In *Proceedings of the 21st IEEE international conference on tools with artificial intelligence (ICTAI'09)*, (pp. 328–335).
Hamadi, Y., Jabbour, S., & Sais, L. (2009b). ManySAT: A parallel SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation—JSAT*, *6*, 245–262.
Hamadi, Y., Jabbour, S., & Sais, L. (2012). Learning from conflicts in propositional satisfiability. *A Quarterly Journal of Operations Research—4OR*, *10*(1), 15–32.
Hamadi, Y., & Wintersteiger, C. M. (2013). Seven challenges in parallel SAT solving. *AI Magazine*, *34*(2), 99–106.
Heule, M., Järvisalo, M., & Biere, A. (2010). Clause elimination procedures for cnf formulas. In *Proceedings of the 17th international conference on logic for programming, artificial intelligence, and reasoning (LPAR'10)*, (pp. 357–371).
Heule, M., Järvisalo, M., & Biere, A. (2011). Efficient cnf simplification based on binary implication graphs. In *Proceedings of the 14th international conference on theory and applications of satisfiability testing (SAT'11)*, (pp. 201–215).
Hooker, J. (2000). *Logic-based methods for optimization: Combining optimization and constraint satisfaction*. Hoboken: Wiley.
Hooker, J. N. (1989). Input proofs and rank one cutting planes. *INFORMS Journal on Computing*, *1*(3), 137–145.
Hooker, J. N., & Fedjiki, C. (1990). Branch-and-cut solution of inference problems in propositional logic. *Annals of Mathematics and Artificial Intelligence*, *1*, 123–139.

Huang, J. (2007). The effect of restarts on the efficiency of clause learning. In *Proceedings of the 20th international joint conference on artificial intelligence (IJCAI'07)*, (pp. 2318–2323).

Jabbour, S. (2009). Learning for dynamic assignments reordering. In *21st IEEE international conference on tools with artificial intelligence (ICTAI'2009), Newark, New Jersey, USA, 2–4 November 2009*, (pp. 336–343).

Jabbour, S., Lonlac, J., & Sais, L. (2013). Adding new bi-asserting clauses for faster search in modern sat solvers. In A. M. Frisch and P. Gregory (Eds.), *Proceedings of the tenth symposium of abstraction, reformulation, and approximation (SARA'2013)*. AAAI.

Jabbour, S., Lonlac, J., Sais, L., & Salhi, Y. (2014). Revisiting the learned clauses database reduction strategies. *CoRR*, arXiv:1402.1956

Jeroslow, R. G., & Wang, J. (1990). Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, *1*, 167–187.

Kautz, H. A, Horvitz, E., Ruan, Y., Gomes, C. P., & Selman, B. (2002). Dynamic restart policies. In *Proceedings of the eighteenth national conference on artificial intelligence (AAAI-02)*, (pp. 674–681).

Li, C. M., & Anbulagan, A. (1997). Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the fifteenth international joint conference on artificial intelligence (IJCAI'97)*, (pp. 366–371).

Luby, M., Sinclair, A., & Zuckerman, D. (1993). Optimal speedup of las vegas algorithms. *Information Processing Letters*, *47*, 173–180.

Marques-Silva, J. (1999). The impact of branching heuristics in propositional satisfiability algorithms. In *Proceedings of the 9th Portuguese conference on artificial intelligence (EPIA'99)*, (pp. 62–74).

Marques-Silva, J. P., & Sakallah, K. A. (1999). Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, *48*(5), 506–521.

McAllester, D. A. (1980). An outlook on truth maintenance. AI Memo 551, MIT AI Laboratory.

Morris, P. (1993). The breakout method for escaping from local minima. In *Proceedings of the eleventh national conference on artificial intelligence (AAAI'93)*, (pp. 40–45).

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., & Malik, S. (2001). Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th design automation conference (DAC'01)*, (pp. 530–535).

Piette, C., Hamadi, Y., & Sais, L. (2008). Vivifying propositional clausal formulae. In *Proceedings of the 18th European conference on artificial intelligence (ECAI'08)*, (pp. 525–529).

Pipatsrisawat, K., & Darwiche, A. (2007). A lightweight component caching scheme for satisfiability solvers. In *Proceedings of the 10th international conference on theory and applications of satisfiability testing (SAT'07)*, (pp. 294–299).

Pipatsrisawat, K., & Darwiche, A. (2008). A new clause learning scheme for efficient unsatisfiability proofs. In *Proceedings of the twenty-third AAAI conference on artificial intelligence (AAAI'08)*, (pp. 1481–1484).

Pipatsrisawat, K, & Darwiche, A. (2009). On the power of clause-learning sat solvers with restarts. In *Proceedings of the fifteenth international conference on principles and practice of constraint programming (CP'09)*, (pp. 654–668).

Pipatsrisawat, K., & Darwiche, A. (2009). Width-based restart policies for clause-learning satisfiability solvers. In *Proceedings of the 12th international conference on theory and applications of satisfiability testing (SAT'09)*, (pp. 341–355).

Pipatsrisawat, K., & Darwiche, A. (2010). On modern clause-learning satisfiability solvers. *Journal of Automated Reasoning*, *44*(3), 277–301.

Prosser, P. (1993). Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, *9*, 268–299.

Quine, W. V. (1955). A way to simplify truth functions. *The American Mathematical Monthly*, *62*(9), 627–631.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of The ACM*, *12*, 23–41.

Sabharwal, A., Samulowitz, H., & Sellmann, M. (2012). Learning back-clauses in sat. In Alessandro, C., & Roberto S. (Eds.), *Proceedings of the 15th international confernce on theory and applications of satisfiability testing (SAT'12)*, volume 7317 of lecture notes in computer science, (pp. 498–499). Berlin Heidelberg: Springer.

Sais, L. (2008). Probleme SAT: Progrès et Défis. Probleme SAT: Progrès et Défis, London.

Silva, J. M., & Sakallah, K. A. (1996). Grasp—a new search algorithm for satisfiability. In *Proceedings of international conference on computer aided design (ICCAD'96)*, (pp. 220–227).

Sörensson, N., & Biere, A. (2009). Minimizing learned clauses. In *Proceedings of the 12th international conference on theory and applications of satisfiability testing (SAT'09)*, (pp. 237–243).

Stallman, R. M., & Sussman, G. J. (1977). Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, *9*(2), 135–196.

Subbarayan, S., & Pradhan, D. K. (2004). 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)*.

The international sat competition and sat race web site. http://www.satcompetition.org/.

Tseitin, G. S. (1968). On the complexity of derivations in the propositional calculus. In H.A.O. Slesenko (Ed.), *Structures in constructives mathematics and mathematical logic, Part II*, (pp. 115–125).

Warners, J. P. (1998). A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, *68*(2), 63–69.

Zhang, H. (1997). SATO: An efficient propositional prover. In W. McCune (Ed.), *Proceedings of the 14th international conference on automated deduction (CADE'97)*, *LNAI*, (Vol. 1249, pp. 272–275). Berlin: Springer.

Zhang, L. (2005). 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)*, (pp. 482–489).

Zhang, L., Madigan, C. F., & Moskewicz, M. H. (2001). Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of the IEEE/ACM international conference on computer aided design (ICCAD'01)*, (pp. 279–285).