Generalized Conflict-Clause Strengthening for Satisfiability Solvers

Allen Van Gelder
http://www.cse.ucsc.edu/∼avg

University of California, Santa Cruz, CA 95064

Abstract. The dominant propositional satisfiability solvers of the past decade use a technique often called conflict-driven clause learning (CDCL), although nomenclature varies. The first half of the decade concentrated on deriving the best clause from the conflict graph that the technique constructs, also with much emphasis on speed. In the second half of the decade efforts have emerged to exploit other information that is derived by the technique as a by-product of generating the conflict graph and learning a conflict clause. The main thrust has been to strengthen the conflict clause by eliminating some of its literals, a process often called conflict-clause minimization, but more accurately described as conflict-clause width reduction, or strengthening.

This paper first introduces implication sequences as a general framework to represent all the information derived by the CDCL technique, some of which is not represented in the conflict graph. Then the paper analyzes the structure of this information. The first main result is that any conflict clause that is a logical consequence of an implication sequence may be derived by a particularly simple form of resolution, known as linear input regular. A key observation needed for this result is that the set of clauses in any implication sequence is Horn-renamable. The second main result is that, given an implication sequence, and a clause C derived (learned) from it, it is NP-hard to find a minimum-cardinality subset of C that is also derivable. This is in sharp contrast to the known fact that such a minimum subset can be found quickly if the derivation is restricted to using only clauses in the conflict graph.

1 Introduction

More and more, propositional satisfiability solvers (SAT solvers, for short) are making their way into other applications as tools. The leading methodology, often called *conflict-driven clause learning* (CDCL), is well established, yet continues to evolve. The underlying idea is to derive *conflict clauses* as a by-product of failed search lines; these clauses are added to the set of clauses representing the formula to be solved. Recall that in SAT testing a *formula* is a conjunctively joined set of *clauses*, each of which is a set of disjunctively joined *literals*, abbreviated as CNF format. The number of literals in a clause is its *width*. The field has become extremely technical, but we shall try to present the main ideas of our new findings informally, to be accessible to non-specialists.

One active line of research is how to “strengthen”, or reduce the width of, such conflict clauses. We put “strengthen” in quotes because one finds several different terms in the literature, including “improve,” “subsume,” “reduce,” and “minimize.” In most cases, what is meant is to find another soundly derived clause whose literals comprise a proper subset of the literals in the conflict clause (sometimes the reference clause to be reduced is an original clause). The clause is a constraint that can be satisfied by making any one of its literals true, so reducing the number of literals creates a stronger constraint. For the strengthening to be logically sound, no solutions to the overall formula may be eliminated. The stronger constraint simply replaces the original conflict clause. Han and Somenzi provide a good review of this emerging subfield [9].

In recent papers two rather surprising observations have emerged concerning solution of typical industrial instances, with thousands of variables and over a million clauses, in some cases: (1) conflict clauses can be reduced 32% in width, on average, permitting a substantial savings in memory [2]; (2) these reductions can be discovered and applied to achieve substantial net savings in time, as well [16, 17]. the method sketched in a poster by Sörensson and Eén, The reduction method has come to be known as *recursive conflict-clause minimization.* It uses only the same clauses as were used to *derive* the conflict clause.

The subject of this paper, and some other recent papers, is how to bring additional clauses from the formula into the strengthening process, clauses which were *not* used in the derivation of the conflict clause. We call this *generalized* conflict-clause strengthening. Audemard *et al.* describe one method, which they call *inverse arcs*, to use other clauses beneficially. In their method, the newly derived conflict clause is not necessarily a subset of the original, but it removes certain literals from the original to enable a longer back-jump during the backtrack, which ensues immediately after the conflict clause is recorded [1]. Han and Somenzi go somewhat in the other direction, by using (possibly intermediate) clauses derived during conflict analysis to strengthen clauses that existed before the analysis began [9].

One motivation for considering subsets of the originally derived conflict clause (rather than including new literals, as in [1]) is that this clause is known to have a property called *1-empowering* [15]. Derivable subsets of a 1-empowering clause are also 1-empowering.

Implication sequences are introduced in Section 2 as a new formalization of part of the operation of CDCL SAT solvers. Implication sequences are supersets of the previous formalization, which we call *antecedent sequences* in this paper. The additional clauses that are included are called *volunteers.* Section 3 shows that implication sequences are Horn Renamable, after reviewing Horn clauses and Horn renamability.

The main technical results are given in Section 4. There it is shown that every clause that is a logical consequence of an implication sequence has a certain simple and short form of resolution derivation. Then it is shown that finding a minimum-cardinality conflict clause that satisfies additional natural conditions is *NP*-hard.

1.1 Terminology and Notation

Let $\mathcal{V}$ be a set of propositional variables. Propositional variables may take on the truth values *true* (or 1) and *false* (or 0). A *literal* is either v or its negation, $\overline{v}$, where v is a variable in $\mathcal{V}$ or $v = \bot$, which denotes *false*, but is treated as a positive literal to make the notation more uniform. Instead of $\overline{\bot}$ we write $\top$, for readability. We consider $\overline{\overline{v}}$ to be synonymous with v. (To distinguish propositional variables from literals, usually letters near the middle of the alphabet (p, q, r, etc.) denote literals, and letters near the end of the alphabet denote propositional variables.)

A *clause* is a disjunctively connected set of literals, and is non-tautological unless specified otherwise. The literals comprising a clause may be shown between square brackets. The *width* of a clause is the number of literals in it. A *CNF formula* (*formula* for short) is a conjunctively connected set of clauses.

An *assignment* is a partial function from variables to truth values. It is often represented by a set of literals that are assigned *true* by the assignment. A *total assignment* assigns values to all variables. An assignment $\mathcal{A}$ is said to *satisfy* a literal p if $\mathcal{A}$ assigns *true* to p, and it is said to *falsify* p if it assigns *true* to $\overline{p}$. The terminology extends to logical expressions in the natural way. A formula is said to *satisfiable* if it is satisfied by some assignment; otherwise the formula is *unsatisfiable*.

Resolution is denoted as follows. For two clauses, $C_1 = [r, p_1, \ldots, p_k]$ and $C_2 = [\overline{r}, q_1, \ldots, q_j]$, r is called the *clashing literal* and *resolution on* r yields the *resolvent*: $\mathbf{res}_r(C_1, C_2) = [p_1, \ldots, p_k, q_1, \ldots, q_j]$, which must not be tautological, unless stated otherwise. A *resolution proof* is a sequence of resolutions whose operand clauses are in the formula under consideration or derived earlier in the proof. A *resolution refutation* (*refutation* for short) is a resolution proof that derives an empty clause.

Unit-clause propagation consists of doing all possible resolutions in which at least one operand is a unit clause. The effect is to reduce the width of the second operand by one, which may result in a new unit clause, whose effects are similarly propagated. If the second operand is also a unit clause, the empty clause is derived.

In the course of unit-clause propagation, the first clause that shrinks to width one or zero is called the *antecedent* of the associated unit literal in many papers. (Some papers use the term "reason" instead of "antecedent".)

2 Implication Sequences

Although CDCL solvers have many technical details, the part we are concerned with can be described in terms of implication sequences, which are composed of propagation sequences. We define propagation sequences and implication sequences abstractly, but the action of a CDCL solver actually creates such sequences.

Definition 2.1 An *assumption clause* is a special clause that serves only a notation purpose, of the form $[q, \top]$. This records that literal q is assumed to be *true*, and is assigned *true* at this point in whatever sequence contains the clause. For uniformity, q is called the *satisfied literal* of such a clause. In addition, $[\top]$ is a placeholder assumption clause that assumes nothing.

A *unit clause* is a clause in which all literals except one have been assigned *false* (falsified). The remaining literal is called the *implied literal*, as well as the *satisfied literal* of this clause. The complements of the falsified literals in the clause are called *reason literals* for this clause.

A *falsified clause* is a clause in which all literals have been assigned *false*. However, for uniformity, we add $\bot$ as an extra literal, and call it the *implied literal* and *satisfied literal*, so that a falsified clause can be processed as though it were a unit clause. □

Definition 2.2 A *propagation sequence* is a sequence of clauses C_i, $i = 1, \ldots, m$, that begins with an assumption clause and continues with zero or more standard clauses that have become unit clauses or falsified clauses. The unit-clause propagation begins with the assumption, as well as variable assignments that were made prior to the propagation sequence, as its unit clauses. The clauses C_i, for $i > 1$, appear in the propagation sequence in the order they were found to be unit or falsified. To some extent, this order is solver dependent. A propagation sequence ends when no further unit clauses or falsified clauses can be derived by unit-clause propagation.

The sequence may not be unique, but once C_1 is chosen, the set of clauses in the propagation sequence is unique. If no falsified clause is derived, then the final assignment, as a set of literals, is unique. Assignments made in one propagation sequence carry over into subsequent propagation sequences that are part of the same implication sequence, which is defined next. □

Definition 2.3 An *implication sequence* is a sequence of one or more propagation sequences in which the last propagation sequence contains at least one falsified clause, and no earlier propagation sequence contains a falsified clause. Each propagation sequence is usually called a *level* (or *decision level*) in the implication sequence, with level numbers beginning at one for the first assumption (and zero before any assumption). An implication sequence may also be viewed as the *concatenation* of its propagation sequences; which view is taken should be clear from the context. Within an implication sequence, clauses (other than assumption clauses) are named as follows: (1) The clause that is earliest in the implication sequence among those that contain q as their satisfied literal is called the *antecedent* of q, and is said to *satisfy* q. If the antecedent is not an assumption clause it also is said to *imply* q (the word *force* is sometimes seen). (2) Other clauses that contain q as their (only) satisfied literal are called *volunteers*.[1] These clauses are said to *re-imply* q.

[1] In gardening lexicon, a *volunteer* is a plant that was not intentionally planted but is not objectionable, whereas a *weed* is objectionable.

Notice that $[q]$ is a unit clause that implies or re-implies q, while $[q, \top]$ denotes an "assumption" (decision or guess) to make q *true* in the computation, but has no logical effect on whether the sequence is satisfiable. □

Note that many solvers stop processing before an implication sequence is complete, if a falsified clause is discovered, and many do not record volunteers. However, the assignments that *were* recorded, and their order, determine, at least implicitly, which clauses are in the implication sequence, as defined.

Example 2.1. This example illustrates the definition of implication sequence, using these clauses, which are part of a formula.

$$\begin{array}{lllllllll} C_1 &=& [\overline{v},\, \overline{x},\, \overline{y},\, \overline{z}] & C_2 &=& [y,\, \overline{u},\, \overline{w},\, \overline{x}] & C_3 &=& [z,\, \overline{x},\, \overline{v}] \\ C_4 &=& [w,\, \overline{v},\, \overline{y}] & C_5 &=& [u,\, \overline{v},\, \overline{w}] & C_6 &=& [x,\, \overline{t},\, \overline{u}] \end{array} \tag{1}$$

The following is a possible implication sequence, with one level per line. The implied or re-implied literal is shown in parentheses for each clause.

$$\begin{array}{l} 1\ [v, \top] \\ 2\ [w, \top]\,, \quad C_5(u) \\ 3\ [t, \top]\,, \quad C_6(x) \quad C_2(y) \quad C_3(z) \quad C_4(w) \quad C_1(\bot) \end{array} \tag{2}$$

C_5 becomes a unit clause at level 2 with u as the implied literal, and v and w as reason literals. C_4 is a volunteer because it re-implies w. It appears in the sequence at a point where all of its literals are assigned. The order in which C_4 and C_3 appear depends on the solver, as they are both eligible as soon as y is assigned *true*. This example is continued in Example 2.2. □

2.1 DPLL and Implication Sequences

Before the modern era of SAT solving the predominant solver methodology was a backtracking search that came to be called DPLL, or a variant of that procedure. "DPLL"" stands for Davis, Putnam, Logemann, and Loveland, who originated the procedure in two classical papers [6, 5]. We briefly review this for unsatisfiable formulas in terms of implication sequences.

DPLL builds an implication sequence as just described, and in addition keeps track of whether each assumption is a *left branch* or a *right branch* in the search tree of assignments that it is exploring. When an implication sequence is concluded with a falsified clause on a left branch with assumption p, the procedure retracts the entire propagation sequence including p, and starts a new propagation sequence with the right-branch assumption $\overline{p}$. Every left-branch assumption is followed up with the complementary right-branch assumption. In Example 2.1, the level-3 propagation sequence would be retracted and an alternative level-3 propagation sequence would be initiated with the assumption $\overline{t}$.

DPLL is naturally expressed with a recursive procedure. Early attempts to enhance DPLL used essentially the same backtracking method, and attempted to prune the search by deriving various clauses.

2.2 CDCL and Implication Sequences

CDCL began with GRASP [14], was soon improved by Chaff [13], and quickly became the dominant SAT solving methodology of the modern era. Many papers mistakenly describe this method as DPLL enhanced with clause learning. Although DPLL can be "annotated" to derive the same clauses as GRASP, it might be forced also to derive exponentially many *additional* clauses. Therefore, as claimed by the original GRASP authors, the way CDCL derives and uses conflict clauses makes it an essentially different method. Stepping through the process with an appropriate example quickly illustrates the difference.

Example 2.2. We continue with Example 2.1 at the conclusion of its implication sequence, (1). The immediate goal is to derive a *conflict clause* that has exactly one literal that was falsified during the latest propagation sequence, which is level 3 in the example (see (2)). We illustrate the *1-UIP scheme*, which is most popular. A sequence of resolutions begins with the falsified clause, C_1, and works backwards through *antecedent* clauses that are also at level 3.

$$\begin{aligned} D_1 = \mathbf{res}_y(C_2,\, C_1) \;&=\; [\bot, \overline{v},\, \overline{x},\, \overline{u},\, \overline{w},\, \overline{z}] \\ D_2 = \mathbf{res}_z(C_3,\, D_1) \;&=\; [\bot, \overline{v},\, \overline{x},\, \overline{u},\, \overline{w}] \end{aligned}$$

The literal x is called the *first unique implication point* (1-UIP) and D_2 is called the *1-UIP conflict clause* because D_2 has $\overline{x}$ as its only literal that was assigned on level 3. D_2 is called an *asserting clause* because, after all level-3 assignments are retracted, D_2 becomes a unit clause. The CDCL solver now "learns" D_2, that is, D_2 is now considered part of the formula.

So far, this could fit into the framework of DPLL, but now the CDCL difference emerges. All assignments made on level 3 are retracted. D_2 is now a unit clause, as one literal became unassigned. Instead of starting another propagation sequence with some assumption, the level-2 propagation sequence is continued with the new unit clause D_2 and implied literal $\overline{x}$.

$$\begin{array}{l} 1\; [v, \top] \\ 2\; [w, \top]\,, \quad C_5(u) \quad D_2(\overline{x}) \quad \ldots \end{array} \tag{3}$$

Notice that $\overline{x}$ is not the complement of any previous assumption. If $\overline{x}$ causes further unit (or empty) clauses to be derived, they append to the level-2 propagation sequence. If unit-clause propagation dies out without falsifying a clause, then a new propagation sequence, with a new assumption literal, is initiated.

In standard CDCL, volunteers are ignored. Thus the position of C_4 on level 3 does not matter. The continuations in Example 2.3 and Example 2.4 illustrate issues that must be considered if volunteers are to be incorporated into the clause-learning process. The *inverse arcs* technique [1] was a first step in this direction. □

Example 2.3. The formula and implication sequence are the same as in Example 2.2. This example shows that volunteers can create a cyclic structure that complicates correct reasoning.

The conflict clause derived from the above implication sequence is

$$D_2 = [\overline{v},\, \overline{w},\, \overline{u},\, \overline{x}]\,.$$

As things stand now, backtracking will go to level 2, where D_2 has one unassigned literal, but cannot go further due to the presence of $\overline{u}$ and $\overline{v}$. Can D_2 be strengthened to permit backtracking to level 1?

Clause C_4 meets all the criteria of Audemard *et al.* [1] for a usable inverse arc: the reason literal y appears as an implied literal at level 3, the level of the conflict, and its antecedent, C_2, participated in the derivation of the conflict clause; the reason literal v appears at level 1, which *precedes* the level in which w became satisfied; finally, w was satisfied at level 2, the current backtrack level.

The motivation is that resolving w out of the conflict clause makes progress toward permitting a longer back jump, while introducing $\overline{y}$ might not be a problem because $\overline{y}$ was able to be resolved out during the derivation of the conflict clause.

However, care must be taken to actually perform the steps, and not simply delete w, assuming the steps will succeed as hoped. (In the `minisat2` conflict-clause reduction, literals are simply deleted, and this is sound because only antecedents are used.) The derivation may continue:

$$\begin{aligned}
D_3 = \mathbf{res}_u(C_5,\, D_2) &= [\bot, \overline{v},\, \overline{x},\, \overline{w}] \\
D_4 = \mathbf{res}_w(C_4,\, D_3) &= [\bot, \overline{v},\, \overline{x},\, \overline{y}] \\
D_5 = \mathbf{res}_y(C_2,\, D_4) &= [\bot, \overline{v},\, \overline{x},\, \overline{u},\, \overline{w}]
\end{aligned}$$

D_4 re-introduced $\overline{y}$ at level 3, so it is not an asserting clause, like D_3 is. The extra level-3 literal had to be resolved out using C_2. But the resolvent D_5 is just the same clause as D_2, so the procedure is in a cycle. Indeed, $[\overline{v},\, \overline{x}]$ would be an unsound derivation. A more favorable case is shown in Example 2.4. □

Example 2.4. A slight change to the clauses in Example 2.3 illustrates how a volunteer *can* be useful. Clause C_7 replaces clause C_4.

$$\begin{array}{lll}
C_1 = [\overline{v},\, \overline{x},\, \overline{y},\, \overline{z}] & C_2 = [y,\, \overline{u},\, \overline{w},\, \overline{x}] & C_3 = [z,\, \overline{x},\, \overline{v}] \\
C_7 = [w,\, \overline{v},\, \overline{z}] & C_5 = [u,\, \overline{v},\, \overline{w}] & C_6 = [x,\, \overline{t},\, \overline{u}]
\end{array}$$

We assume the same implication sequence as earlier examples, but with C_7 in the place of C_4. The conflict clause D_2 is the same, since its derivation ignores volunteers. C_7 also meets all the criteria of Audemard *et al.* [1] for a usable inverse arc (z plays the former role of y). The derivation may continue:

$$\begin{aligned}
D_3 = \mathbf{res}_u(C_5,\, D_2) &= [\bot, \overline{v},\, \overline{x},\, \overline{w}] \\
D_6 = \mathbf{res}_w(C_7,\, D_3) &= [\bot, \overline{v},\, \overline{x},\, \overline{z}] \\
D_7 = \mathbf{res}_z(C_3,\, D_6) &= [\bot, \overline{v},\, \overline{x}]
\end{aligned}$$

This time, $\overline{z}$ at level 3 has been re-introduced in D_6, making it non-asserting, so C_3 must be used to resolve out the extra level-3 literal, producing D_7. D_7 is asserting and is stronger than the previously derived asserting clauses, D_2 and D_3. The end result is that D_7 is soundly derived as the conflict clause, and a back-jump to level 1 is possible. That is, after retracting all assignments made at the current level 3, the procedure determines that none of the assignments at level 2 influence D_7, so these are all retracted, as well, and D_7 is added as an additional unit clause at level 1, with $\overline{x}$ as the satisfied literal.

$$1\ [v, \top]\,, \quad D_7(\overline{x}) \quad \ldots \tag{4}$$

Notice that $\overline{x}$ is not the complement of any previous assumption. If $\overline{x}$ causes further unit (or empty) clauses to be derived, they append to the level-1 propagation sequence. Thus the CDCL procedure has departed decisively from the DPLL framework. In fact, it would be perfectly proper for the next assumption literal to be w or t again. □

Examples 2.3 and 2.4 demonstrate the importance of discovering cycles, if volunteers are to be included in conflict-clause reduction.

2.3 Traditional Use of Implication Sequences

In the standard methodology originated in GRASP [14], and continued in Chaff [13], Minisat [7], and other solvers, a conflict graph is constructed using only antecedents, besides one chosen falsified clause. Several papers formalize this technique [18, 3]. For any literal that has been assigned false, there is precisely one antecedent in which its complement is the (true) implied literal (the antecedent might be an assumption clause). The antecedent necessarily precedes all occurrences of this false literal.

Definition 2.4 Let $C = \{C_i\}$ be an implication sequence of clauses. Let C_A be the subsequence of decisions and antecedents, and let C_V be the subsequence of volunteer clauses. We call C_A an *antecedent sequence* to distinguish it from the implication sequence. We suppose that the final decision in C led to one or more falsified clauses, the earliest being in C_A. Any additional falsified clauses are in C_V. □

It is an easy matter to define an acyclic graph in which satisfied literals are vertices, with $\perp$ being the satisfied literal of the chosen falsified clause. If q is a vertex and its antecedent is $[q, \overline{p_1}, \ldots, \overline{p_k}]$, there are directed edges from q to the vertices for $p_1, \ldots, p_k$; if q is an assumption, there are no outgoing edges.[2] Vertices are included in the conflict graph only if they are reachable from the $\perp$ vertex.

We have defined an antecedent sequence to be an implication sequence in which all volunteers have been discarded. Since we have a one-to-one correspondence between vertices and antecedents, we might regard the antecedents as

[2] This edge orientation is opposite that seen in several *papers*, but is consistent with the *solvers*' actual data structure.

being the vertices, instead of the satisfied literals being the vertices. Then the vertices of the conflict graph comprise a subset of the antecedent sequence, which in turn is a subset of the full implication sequence.

3 Implication Sequences are Horn Renamable

The key insight for this paper is that the set of clauses in any implication sequence is Horn renamable. Thus the rich body of theory for Horn-clause reasoning can be brought to bear. Recall that a *Horn clause* has one or zero positive literals. A *Horn set* is a set of Horn clauses. A set of clauses is called *Horn renamable* if flipping the polarities of all occurrences of certain variables turns it into a Horn set. It is known from the early days of theorem proving [10, 4, 12] that:

Theorem 3.1 *Positive unit resolution* is complete for Horn sets; that is, the empty clause is derivable from a Horn set if and only if it is derivable by a resolution proof in which one operand is always a positive unit clause. □

Corollary 3.2 *Unit resolution* is complete for renamable Horn sets; that is, the empty clause is derivable from a renamable Horn set if and only if it is derivable by a resolution proof in which one operand is always a unit clause. □

The following simple lemma may be known to some researchers, at least for antecedent sequences.[3] We state it here for self-containment and because it appears not to be widely known and is so far unpublished.

Lemma 3.3 An implication sequence is Horn renamable.

Proof: Flip every negative satisfied literal and flip every negative assumption literal. Now every clause is a Horn clause whose positive literal is its satisfied literal. ■

It is unnecessary to do this flipping in the actual computation, but for convenience of presentation, we assume without loss of generality that satisfied literals are always positive. (The attentive reader may have noticed this in the examples; Lemma 3.3 justifies the practice.)

4 Conflict-Clause Strengthening Problem

Let $C = C_1, C_2, \ldots, C_m$ be an implication sequence of clauses. As in Definition 2.4, let C_A be the antecedent sequence of C; that is, the subsequence of decisions and antecedents. Let C_V be the subsequence of volunteer clauses. We

[3] Previous papers use the term "implication graph" for the graph associated with antecedents, but we avoid this term because our "implication sequence" includes volunteers.

suppose that the final decision in C led to one or more falsified clauses, the earliest being in C_A. Any additional falsified clauses are in C_V.

Let γ_0 be the conflict clause derived by the CDCL solver using the 1-UIP scheme [14, 18, 3], or any scheme that derives asserting clauses (recall Section 2.2). That is, γ_0 is derived from the conflict graph based on clauses in C_A reachable from the falsified clause; we call these clauses C^*_A. We know that adding $\neg(\gamma_0)$ as unit clauses to C^*_A makes an inconsistent set. The *conflict-clause strengthening problem* is to find another, stronger, conflict clause, $\gamma \subset \gamma_0$, where subset is strict. There are several versions, depending on what is allowed.

Suppose the problem is cast as finding a minimum-width $\gamma \subseteq \gamma_0$ that is logically implied by C^*_A, or equivalently, such that adding $\neg(\gamma)$ as unit clauses to C^*_A causes inconsistency. Then it is a "folklore theorem" that this problem can be solved in P-time and that the minimum-width clause is unique [16, 17]. The procedure implemented in MiniSat 2.0 [8] is believed to achieve this. This procedure is now called *recursive conflict-clause reduction*.

A more ambitious goal is to require that $\gamma \subseteq \gamma_0$ be the minimum-width clause that is logically implied by *all of* C. That is, by including the volunteer clauses in C_V, a smaller subset of γ_0 may be logically implied, or equivalently, a smaller subset of $\neg(\gamma_0)$ may be sufficient to produce inconsistency, as illustrated in Example 2.4. We now define this problem formally in the *NP* framework as a decision problem.

Definition 4.1 The decision form of the *general minimum conflict clause problem* is defined as follows.

Input: An implication sequence C (Definition 2.3), a conflict clause γ_0 as described above, and a positive integer K.

Question: Is there a clause $\gamma \subset \gamma_0$ with at most K literals such that $\neg(\gamma) \cup C$ is inconsistent?

Note that $\neg(\gamma)$ is treated as a set of unit clauses in this notation. □

Before addressing the complexity of the strengthening problem, we show in Section 4.1 that *any* clause that is a logical consequence of an implication sequence C has a simple, short derivation of a particular kind.

4.1 Implication Sequences and Linear Input Regular Derivations

The property stated in the next theorem is known for *antecedent* sequences (i.e., C^*_A in the above discussion), due to Beame *et al.* [3]. The next theorem shows that it holds for entire implication sequences. The proof idea reduces the problem to one covered by Beame *et al.*.

Definition 4.2 A *linear input regular* (LIR) resolution derivation is a sequence in which each derived clause after the first uses an "input" clause as the first operand and the previous derived clause as the second operand, and does not resolve on any literal more than once. (The terminology follows Biere [2], but such derivations were less descriptively called "trivial resolutions" by Beame *et al.* [3].) An "input" clause is one that was in the original formula or was derived before the present derivation began. □

Theorem 4.3 Let C be an implication sequence and let the clause γ be a logical consequence of the clauses in C. (Note that assumption clauses do not play any role in determining logical consequences.) Then γ (or a subset of γ) can be derived by a LIR resolution from C.

Proof: Assume W.L.O.G. (in view of Lemma 3.3) that C and γ are Horn. Add $\neg(\gamma)$ to the antecedents and volunteers of C, and find a refutation by *positive* unit resolution, which is known to be complete for Horn clauses. (Theorem 3.1). Derive each positive unit clause only once. The result is a conflict graph in which every implied literal has a unique antecedent. For purposes of forming the conflict graph, every positive literal of $\neg(\gamma)$ is treated as a decision, i.e., it has no antecedent. If γ has a positive literal x, then $[\overline{x}] \in \neg(\gamma)$ is treated as a unit clause in the input clause set. Following the terminology and results of Beame *et al.* [3], the conflict graph has a cut in which the literals of $\neg(\gamma)$ comprise the "reason" side of the cut and the remaining literals comprise the "conflict" side of the cut. Therefore, γ can be derived by LIR.

Note that if γ has a positive literal x it becomes a *negative* unit clause $[\overline{x}]$. Although it cannot play the role of the required positive unit clause for resolution, eventually the positive unit clause $[x]$ gets implied, and then the two can resolve. This completes the proof. ■

The implication of this theorem is that conflict clauses that are derivable from implication sequences that include volunteers have short, non-redundant, derivations. For example, the procedures described by Audemard *et al.* [1] for using "inverse arcs" apparently involve redundant derivations, as illustrated in Example 2.4. The above theorem tells us that resolving on the same literal more than once is unnecessary if a proper order is used.

Example 4.1. Again consider the clauses and the same implication sequence as in Example 2.4, where the use of the volunteer C_7 was successful, but required resolving on some literals more than once.

$$C_1 = [\overline{v}, \overline{x}, \overline{y}, \overline{z}] \quad C_2 = [y, \overline{u}, \overline{w}, \overline{x}] \quad C_3 = [z, \overline{x}, \overline{v}]$$
$$C_7 = [w, \overline{v}, \overline{z}] \quad C_5 = [u, \overline{v}, \overline{w}] \quad C_6 = [x, \overline{t}, \overline{u}]$$

Here is a linear input regular derivation from C_1, the falsified clause:

$$\begin{aligned} D_1 = \mathbf{res}_y(C_2, C_1) &= [\bot, \overline{v}, \overline{x}, \overline{u}, \overline{w}, \overline{z}] \\ D_8 = \mathbf{res}_u(C_5, D_1) &= [\bot, \overline{v}, \overline{x}, \overline{w}, \overline{z}] \\ D_9 = \mathbf{res}_w(C_7, D_8) &= [\bot, \overline{v}, \overline{x}, \overline{z}] \\ D_{10} = \mathbf{res}_z(C_3, D_9) &= [\bot, \overline{v}, \overline{x}] \end{aligned}$$

The key difference from Example 2.4 is that resolution on $\overline{z}$ at level 3 was delayed, so that it did not need to be re-introduced. □

After the initial conflict clause has been derived, there are several published methods for reducing it. The method used in MiniSat 2.0 amounts to doing additional resolutions (possibly redundantly) on literals that were implied in earlier propagation sequences, yielding a subset of the original conflict clause [8, 16]. It is now known that the redundancy is efficiently avoidable [17]. Volunteer clauses are not used. (Although Theorem 4.3 guarantees that a LIR proof exists, even when volunteer clauses are included, it does not tell how to find it.)

Definition 4.4 Given a set of Horn clauses H, define directed edges between variables by $v \rightarrow w$ whenever H contains some clause in which v occurs positively and w occurs negatively. If the resulting graph is acyclic, then H is said to be acyclic. A Horn renamable set of clauses is acyclic if it is acyclic Horn after some renaming. □

Antecedent sequences are always acyclic. Although implication sequences often are not acyclic, Theorem 4.3 guarantees that any logical consequence can be derived from a subset of clauses that *is* acyclic.

Audemard *et al.* [1] described a method for using certain volunteers to resolve away literals that were assigned in the propagation sequence at the *backtrack level*, to enable longer back jumping. Their method might resolve on literals more than once, and might produce a a conflict clause that is not a subset of the original.

The general conflict-clause strengthening problem addressed in this paper has the goal of reducing the final conflict clause to be a small subset of the original, using volunteers in some cases, to achieve greater reductions than are possible with antecedents alone.

4.2 The General Minimum Conflict Clause Problem Is *NP*-Complete

Our next result is that the decision form of the general minimum conflict clause problem, stated in Definition 4.1, is *NP*-complete. That is, finding a minimum-cardinality subset $\gamma \subset \gamma_0$, where γ_0 is a conflict clause derived from a general implication sequence, is *NP*-hard. This finding stands in sharp contrast with the fact that the problem can be solved a low-degree polynomial time for implication sequences *without* volunteers. Kleine Büning and Lettmann give a theorem with somewhat the same flavor [11, Problem MI, p. 245], but Theorem 4.6 below is not a corollary, because it requires that (A) the input clauses comprise an implication sequence C that could be generated by a CDCL solver and (B) the clause to be minimized must be a subset of a specified conflict clause γ_0, that could be derived by the same CDCL solver, rather than being any subset of variables. CDCL-derivable conflict clauses are not arbitrary; it is known that they have a property called 1-*empowering* [15]. Thus Theorem 4.6 has several additional restrictions not found in "Problem MI." The proof uses reduction from the well known Hitting Set problem, whose formal definition follows.

Definition 4.5 The decision form of the *Hitting Set problem* is:
Input: A collection of sets S_i, $i = 1, \ldots, m$ whose union is $U = \{x_j \mid j = 1, \ldots, n\}$ and an integer M such that $0 < M < n$.
Question: Is there a subset $H \subset U$ with at most M elements such that H intersects each S_i? □

Theorem 4.6 The general minimum conflict clause (GMCC) problem is *NP*-complete. The problem remains *NP*-complete if the implication sequence C is restricted to be an acyclic clause set, as defined in Definition 4.4.

Proof: The problem is in *NP* because, if a clause γ is presented as a certificate, then the set of clauses $\neg(\gamma) \cup C$ is renamable-Horn, so can be checked for inconsistency with unit-clause propagation in P-time. To show *NP*-hardness, reduce from Hitting Set (Definition 4.5).

Using the notation in the definition, the transformation arranges that each $x_j \in U$ is an assumption and γ_0 contains each $\overline{x_j}$, as well as some "control" literals. Clauses of the form $[s_i, \overline{x_j}]$ are generated to specify set membership, i.e., $x_j \in S_i$ in the Hitting Set instance. Control variables y_1, y_2, y_3, and z ensure that the desired conflict clause γ_0 is derived by a CDCL solver. Additional "control" clauses, including one volunteer clause, ensure that a sufficiently small-width γ is logically implied if and only if the x_j that occur in $\neg(\gamma)$ provide a sufficiently small H. The formal details follow.

Transform a Hitting Set instance $(\{S_i\}, M)$ into a GMCC instance (C, γ_0, K) with the following steps:
(1) Output the following propositional clause sequence over the variables x_j, $j = 1, \ldots, n$; s_i, $i = 1, \ldots, m$; y_k, $k = 1, 2, 3$; and z.

S-clauses:	For each x_j in order, $j = 1, \ldots, n$: output the decision clause $[x_j, \top]$, then, for each S_i such that $x_j \in S_i$, output $[s_i, \overline{x_j}]$.
decision y-clause:	output $[y_1, \top]$.
first z-clause:	output $[z, \overline{y_1}, \overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}]$.
second y-clause:	output $[y_2, \overline{y_1}]$.
volunteer z-clause:	output $[z, \overline{y_2}, \overline{s_1}, \overline{s_2}, \ldots, \overline{s_m}]$.
third y-clause:	output $[y_3, \overline{y_2}]$.
all-negative clause:	output $[\overline{y_1}, \overline{y_3}, \overline{z}]$.

The above clauses comprise the sequence C, which is easily seen to be an implication sequence.
(2) Output the 1-UIP conflict clause $\gamma_0 = [\overline{y_1}, \overline{x_1}, \ldots, \overline{x_n}]$.
(2) Output $K = M + 1$.

The output C, is clearly an acyclic Horn clause set, and can clearly be computed in time quadratic in the length of the Hitting set instance. It is straightforward to show that (C, γ_0, K) is a yes instance of GMCC if and only if $(\{S_i\}, M)$ has a hitting set of size at most $M = K - 1$. ■

Keep in mind that the sequence C is not the whole formula presented to the CDCL solver, just one "run" to a conflict clause. In general, given any specific deterministic solver of this class, the transformation can be tweaked and the rest of the formula can be specified to force the solver into the desired sequence of decisions, implications and re-implications.

5 Conclusion

We considered the structure of the set of *all* fully assigned clauses at the time that a conflict-driven clause-learning (CDCL) solver derives (learns) a conflict clause. These clauses can be organized into an implication sequence that faithfully represents the actions of a CDCL solver, such as GRASP, Chaff, Minisat, and others. However, these solvers ignore the information available in many of these clauses, which we name "volunteers." We showed that the set of clauses in an implication sequence is always Horn renamable. It followed from this that *any* clause that is logically implied by the clauses of the implication sequence has a linear input regular derivation (Definition 4.2). We also showed that in this environment trying to squeeze a derived clause, such as a conflict clause, down to its absolutely minimum width is *NP*-hard.

Acknowledgment We thank the anonymous referees for helpful comments.

References

1. G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, and L. Sais. A generalized framework for conflict analysis. In *SAT*. Springer, 2008.
2. A. Biere. Picosat essentials. *J. Satisfiability, Boolean Modeling and Comp.*, 4:75–97, 2008.
3. Paul Beame, Henry Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *J. Artificial Intelligence Research*, 22, 2004.
4. C-L. Chang and R.C-T. Lee. *Symbolic Logic and Mechanical Theorem Proving.* 1973.
5. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.
6. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.
7. N. Eén and N. Sörensson. An Extensible SAT-solver. In *SAT*, LNCS 2919, Springer, 2003.
8. Niklas Eén and Niklas Sörensson. MiniSat v.1.13 – a SAT solver with conflict-clause minimization. Poster at SAT, 2005.
9. Hyojung H. and F. Somenzi. On-the-fly clause improvement. In *SAT*, LNCS 5584, 2009.
10. L. J. Henschen and L. Wos. Unit refutations and Horn sets. *JACM*, 21, 1974.
11. H. Kleine Büning and T. Lettmann. *Propositional Logic: Deduction and Algorithms.* 1999.
12. D. W. Loveland. *Automated Theorem Proving: a Logical Basis.* North-Holland, 1978.
13. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *39th Design Automation Conference*, June 2001.
14. J. P. Marques-Silva and K. A. Sakallah. GRASP–a search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48:506–521, 1999.
15. K. Pipatsrisawat and A. Darwiche. A new clause learning scheme for efficient unsatisfiability proofs. In *AAAI*, 2008.
16. N. Sörensson and A. Biere. Minimizing learned clauses. In *SAT*, LNCS 5584, 2009.
17. Allen Van Gelder. Improved conflict-clause minimization leads to improved propositional proof traces. In *SAT*, LNCS 5584, Springer, 2009.
18. L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *ICCAD*, Nov. 2001.