Non-Restarting SAT Solvers With Simple Preprocessing Can Efficiently Simulate Resolution

Paul Beame
Computer Science and Engineering
University of Washington
Seattle, WA 98195, USA
beame@cs.washington.edu

Ashish Sabharwal
IBM Watson Research Center
1101 Kitchawan Road
Yorktown Heights, NY 10598, USA
ashish.sabharwal@us.ibm.com

Abstract

Propositional satisfiability (SAT) solvers based on conflict directed clause learning (CDCL) implicitly produce resolution refutations of unsatisfiable formulas. The precise class of formulas for which they can produce polynomial size refutations has been the subject of several studies, with special focus on the clause learning aspect of these solvers. The results, however, assume the use of non-standard and non-asserting learning schemes, or rely on polynomially many restarts for simulating individual steps of a resolution refutation, or work with a theoretical model that significantly deviates from certain key aspects of all modern CDCL solvers such as learning only one asserting clause from each conflict and other techniques such as conflict guided backjumping and phase saving. We study non-restarting CDCL solvers that learn only one asserting clause per conflict and show that, with simple preprocessing that depends only on the number of variables of the input formula, such solvers can polynomially simulate resolution. We show, moreover, that this preprocessing allows one to convert any CDCL solver to one that is non-restarting.

Introduction

The design of practically efficient algorithms for the Boolean Satisfiability (SAT) problem has witnessed tremendous advances since the 1990s, most notably in the family of CDCL or conflict-directed clause learning solvers (Marques-Silva, Lynce, and Malik 2009). These practical advances in turn have sparked much interest in formally understanding the strengths and limitations of various key techniques employed by today's sophisticated implementations that can handle instances with several million variables. Among these techniques are *clause learning* schemes, which cache and reuse information about subformulas the solver has proved unsatisfiable, and *rapid restarts*, which attempt to prevent the solver from getting stuck in parts of the search space that are difficult to reason about.

Clause learning, with early ideas originating as far back as the 1970s (Stallman and Sussman 1977; de Kleer and Williams 1987), has been studied heavily from a practical perspective, leading to the exploration of many different learning approaches but an eventual convergence on the so-called *asserting* learning schemes (Marques-Silva and Sakallah 1996; Moskewicz et al. 2001; Zhang et al. 2001; Marques-Silva, Lynce, and Malik 2009). At the same time, restarts, with their roots in the study of heavy-tailed runtime distributions of *non-learning* search algorithms (Frost, Rish, and Vila 1997; Gomes, Selman, and Crato 1997), have also been the focus of much attention in the AI community (Hogg and Williams 1994; Gomes, Selman, and Kautz 1998; Luby, Sinclair, and Zuckerman 1993; Walsh 1999; Haim and Walsh 2009; Audemard and Simon 2012).

Clause learning and restarts are also the two key CDCL techniques that have been analyzed from a formal perspective, adding substantially to the observation that DPLL based complete SAT solvers (implicitly) produce tree-like resolution refutations (Robinson 1965) when presented with an unsatisfiable instance. However, as we will discuss shortly, the best known answer to the fundamental question of whether today's CDCL solvers can efficiently simulate any (general) resolution refutation involves a heavy use of *both* (asserting) clause learning and restarts. Are both of these techniques essential to simulating resolution?

Our main result is that with a simple preprocessing step a non-restarting CDCL solver S' can simulate the execution of any restarting CDCL solver S, with only a small overhead. In particular, both S and S' produce identical resolution refutations when the input formula F is unsatisfiable. However, our method applies not only to resolution refutations or the use of CDCL solvers on unsatisfiable formulas. We show, more generally, that our simple preprocessing applied to *any* CNF formula, together with a modified variable selection heuristic, allows one to eliminate all explicit restarts of any CDCL solver at the additive total overhead of only $O(n + R)$ variable assignments (decisions or unit propagations) where n is the number of variables and R is the number of restarts in the original CDCL execution. The idea is to interleave the execution of S on an n variable input formula with the processing of a new formula on disjoint variables that essentially implements an n-bit counter. As a consequence, we prove that with simple preprocessing, non-restarting CDCL solvers can efficiently simulate resolution.

Background and Related Work

Beame, Kautz, and Sabharwal (2003; 2004) initiated a formal study of CDCL SAT solvers, specifically seeking to understand the theoretical power of clause learning and restarts from a proof complexity perspective. One of their findings was that for any natural refinement R of (general) resolution, there exists CNF formulas F_R such that CDCL solvers—if allowed to selectively ignore unit propagations or conflicts and to perform clause learning using a specific scheme—can produce polynomial size refutations (in the associated CL proof system) of certain derived formulas F'_R that require exponential size R refutations. This result did not require the use of any restarts. However, the construction of F'_R relied on the specifics of some polynomial size resolution refutation of F_R. Further, their construction does not work for asserting clause learning schemes. Specifically, any asserting scheme would simply learn that all extension variables of F'_R must be true, reducing F'_R back to F_R.

Hertel et al. (2008) presented a clever transformation of F_R into F''_R which did not rely on knowing a resolution refutation of F_R. However, their finding was that F''_R has polynomial size proofs in a system called *pool resolution* (Van Gelder 2005). While pool resolution is a clean framework that can be used to capture certain aspects of CDCL SAT solvers, it has some features that could, at least in principle, make it much more powerful than actual CDCL solvers. First, pool resolution is not restricted to backtracking to asserting levels or, in its full generality, to always propagate a unit clause when one exists. More importantly, when simulating conflict analysis and clause learning as performed by a CDCL solver S, pool resolution effectively learns *one clause per node in the conflict graph* analyzed by S, thus learning not just one asserting clause from each conflict but up to n clauses in total.

Buss, Hoffmann, and Johannsen (2008) introduced the RTL family of proof systems to capture CDCL solvers and showed several interesting results about them, such as the strength of limited width variants of RTL. Using this system, Buss and Bonet (2012) and Buss and Kolodziejczyk (2012) recently showed that every family of formulas currently known to separate resolution from some natural refinement of it admits polynomial size refutations in the weakest RTL variant, regWRTI. However, RTL based proof systems, including regWRTI, share the drawback of pool resolution that they effectively allow learning and re-using up to n clauses in situations where a CDCL solver would be allowed to learn only one, very specific kind of clause, namely an asserting clause. Further, they also allow the *weakening rule* of resolution, which essentially amounts to letting the solver selectively ignore unit propagations or conflicts.

The model that comes perhaps the closest to how modern CDCL solvers work—and the model we use for our results—is the one proposed by Pipatsrisawat and Darwiche (2009; 2011). While pool resolution and RTL systems do not explicitly model the highly interconnected notions of decision level, unit propagation, and 1-UIP or asserting clauses, Pipatsrisawat and Darwiche showed that, with enough restarts, CDCL solvers learning only one asserting clause per conflict and being free to employ other common search strategies such as conflict-guided backjumping, can simulate any resolution refutation with only a polynomial overhead. However, unlike the previously mentioned results, their result relies heavily on the ability of CDCL solvers to restart. In particular, in the worst case, the CDCL solver restarts $O(n^2)$ times per learned clause, and generates $O(n^4)$ new clauses per clause, in the resolution proof it tries to simulate.

Main Idea

Our main contribution is a combination and extension of the above ideas to develop a novel way of preprocessing an input formula F to produce $\tilde{F}$ such that a non-restarting CDCL solver S' operating on $\tilde{F}$ can be seen as interleaving two executions, one of which mimics a counter and the other precisely mimics the steps of a restarting solver S that S' tries to simulate without restarts. $\tilde{F}$ is obtained by conjoining F with a counting formula G. G is defined on a set of variables disjoint from F and, like the construction of Beame, Kautz, and Sabharwal (2004), is trivially satisfiable. The counting aspect of G is inspired by the construction of Hertel et al. (2008) but is substantially simpler. Moreover, not only does G not rely on the knowledge of a short refutation of F, it does not even rely on F except for the number of variables in it. Finally, as in the model of Pipatsrisawat and Darwiche (2011), S' learns one asserting clause per conflict. It may optionally use fast backjumping or phase saving.

We show that S' operating on $\tilde{F}$ can simulate S operating on any satisfiable or unsatisfiable formula F, with little overhead. If F is unsatisfiable and S produces a resolution refutation P of F, then S' also produces P. Combining this result with the construction of Pipatsrisawat and Darwiche (2011) allows us to prove that S' can efficiently simulate any resolution refutation.

Our correctness analysis for simulating resolution is an extension of ideas from Pipatsrisawat and Darwiche (2011), Atserias, Fichte, and Thurley (2009; 2011), and Hertel et al. (2008). The first two papers provide the concept of *empowerment* or *absorption* of clauses w.r.t. unit propagation, which is the key to reason about resolution derivations in the presence of dynamic aspects of CDCL solvers such as eager unit propagation, learning asserting clauses, backjumping, etc. Following Pipatsrisawat and Darwiche (2011), the goal when trying to simulate a resolution refutation P should not be to derive every individual clause in P but rather to derive clauses that progressively absorb all clauses in P. Our construction and analysis follow the same structure, except for the fact that we do not allow the solver to restart. Instead, borrowing ideas from Hertel et al. (2008), we interleave the execution on F with an execution on G, which helps simulate restarts through a counter.

Preliminaries

We are interested in Conjunctive Normal Form (CNF) formulas over propositional or Boolean variables taking values in $\{0, 1\}$. The value is also referred to as the variable's *polarity* or *phase*. A literal is a variable x or its negation $\overline{x}$, with $\overline{x} = 0$ iff $x = 1$. A clause is a disjunction ($\vee$) of literals, and a CNF formula F is a conjunction ($\wedge$) of clauses. We will write $\ell \in C$ if ℓ is a literal appearing in clause C. If $\ell \in C$ and $\overline{\ell} \in C'$, then C and C' are said to *clash* on the literal ℓ. $|C|$ denotes the number of literals in C. C is referred to as a unit clause if $|C| = 1$. $\bot$ denotes the empty (and thus unsatisfiable) clause.

For a literal ℓ, the *simplified formula* $F|_\ell$ is obtained by removing from F all clauses containing ℓ and removing $\overline{\ell}$ from all clauses of F that contain $\overline{\ell}$. A partial assignment σ to the variables of F can be thought of as the set of literals that σ sets to 1. $F|_\sigma$ is defined as the formula obtained by simplifying F w.r.t. all literals in σ. *Unit propagation* applied to F w.r.t. σ extends σ by adding to it all literals in unit clauses of $F|_\sigma$, simplifying F_σ further w.r.t. these newly added literals, and repeating until no more unit clauses are identified. Unit propagation is said to infer a *conflict* if it identifies a unit clause ℓ in $F|_\sigma$ such that $\overline{\ell} \in \sigma$.

F is said to absorb a clause C if F and $F \wedge C$ have identical inference power w.r.t. unit propagation, i.e., whatever one can derive from $F \wedge C$ using unit propagation one can also derive from F itself. Formally:

Definition 1. A clause C is said to be *absorbed* by a CNF formula F if for every literal $\ell^* \in C$, unit propagation on $F|_{\{\overline{\ell}:\ell\in C\setminus\{\ell^*\}\}}$ either infers a conflict or infers ℓ^*.

Given two clauses $D = (x \vee C)$ and $D' = (\overline{x} \vee C')$ such that C and C' do not clash on any literal, one can apply the *resolution rule* (Robinson 1965) to resolve D with D' over x and derive the resolvent $(C \vee C')$ which is satisfied by a truth assignment if and only if both D and D' are.

Definition 2. A *resolution derivation* P of a clause C from a CNF formula F is a sequence $C_1, C_2, \ldots, C_s$ such that $C_s = C$ and each $C_i, 1 \leq i \leq s$, is either a clause of F or is derived by applying the resolution rule to clauses C_j and C_k such that $j, k < i$. P is *minimal* if every clause $C_i, 1 \leq i < s$, is used in the derivation of some clause $C_j, j > i$. The *length* of P, denoted *length*(P), is s. The *size* of P, denoted *size*(P), is $\sum_{i=1}^{s} |C_i|$.

Definition 3. A *resolution refutation* P of a CNF formula F is a resolution derivation of $\bot$ from F.

CDCL SAT Solvers. Our goal is to analyze the strength of Conflict-Directed Clause Learning (CDCL) SAT solvers as combinatorial algorithms that search for resolution refutations of unsatisfiable formulas. In the interest of space, we refer the reader to several surveys that discuss CDCL solvers in detail, in particular the relevant chapter in the Handbook of Satisfiability (Marques-Silva, Lynce, and Malik 2009) and the model of Pipatsrisawat and Darwiche (2011) that we closely follow. We provide an intuitive description below, highlighting the key aspects and terms we will need.

Suppose we have a CDCL solver S and a CNF formula F over variables $V, |V| = n$. S maintains a partial assignment σ and a *decision level* d. Initially, σ is empty and $d = 0$. First, S applies unit propagation to F w.r.t. the empty σ at the current decision level, 0. If unit propagation infers a conflict, S halts and declares F to be unsatisfiable. Otherwise, S repeats the following process involving *branching decisions*, *unit propagation*, and *conflict analysis* until unit propagation at decision level 0 infers a conflict. To branch, S increments d by 1, selects a literal ℓ such that $\ell, \overline{\ell} \notin \sigma$, extends σ by adding ℓ to it, and associates ℓ with decision level d. Then S applies unit propagation on F w.r.t. σ and associates all literals newly added to σ also with decision level d. If there is no conflict, then S proceeds as follows: if $|\sigma| < n$ it repeats the branching process, otherwise it declares F to be satisfiable with σ as a satisfying assignment. More interestingly, if unit propagation infers a conflict, there must be a *conflict clause* C_{conf} in F such that σ falsifies C_{conf} and, furthermore, C_{conf} has at least one literal associated with decision level d. At this point, S performs *conflict analysis* to derive a *learned clause* C_L that is not in F and has the property that unit propagating $F \wedge \bigwedge_{\ell\in C_L} \overline{\ell}$ w.r.t. the empty partial assignment infers a conflict. Conflict analysis also yields a *backtrack level* $b < d$. How C_L and b are determined is something we will discuss shortly. At this point, S *learns* C_L by adding it to F, backtracks to decision level b by removing from σ all literals associated with decision levels greater than b, and applies unit propagation taking the learned clause C_L into account. If there is an immediate conflict, it repeats the conflict analysis process, otherwise it repeats the branching decision process.

Our main interest is in learning strategies that derive an *asserting clause* C_L when encountering a conflict at decision level d. C_L is called *asserting* if it has exactly one literal associated with decision level d. It is always possible to learn such a clause; e.g., the clause containing the negations of the literals in σ that are set by branching decisions is one such clause. The backtracking level b associated with such a clause is 0 if C_L is a unit clause, and it is the second largest decision level associated with the literals in C_L (thus $b \leq d - 1$). In general, b can be much smaller than $d - 1$, so backtracking to level b is sometimes referred to as non-chronological backtracking or *backjumping*. Suppose that ℓ is the unique literal in C_L associated with decision level d. Then, when S backjumps to decision level b, all other literals of C_L must still be falsified, making unit propagation "assert" or infer ℓ. b is therefore referred as the *asserting level* of C_L, and ℓ as the *asserted literal*.

Learned clauses can be associated with *cuts* in an underlying *conflict graph* G (Marques-Silva and Sakallah 1996; Beame, Kautz, and Sabharwal 2004). Nearly all of today's CDCL solvers learn asserting clauses, in fact a particular kind of asserting clause called the *1-UIP* clause, which corresponds to the asserting cut that is "closest" to the conflict end of G. A precise understanding of 1-UIP clauses is not necessary for our results.

S is said to use *chronological backtracking* if it always backtracks to decision level $d-1$, irrespective of the decision level b provided by conflict analysis.

S is said to *restart* if, rather than backtracking to the decision level b provided by conflict analysis, S backtracks all the way to decision level 0, empties out σ, retains all clauses learned so far, and starts the search again.

S is said to use *phase saving* if it maintains a phase assignment $\rho \in \{0, 1\}^n$ (initialized arbitrarily) that stores the last phase (if any) assigned by it through a branching decision or unit propagation to each variable x, and it always prefers this phase when choosing between x and $\overline{x}$ during a branching decision.

Simulating Restarts with Preprocessing

In the rest of this paper, unless otherwise specified, CDCL solvers will be assumed to learn one asserting clause per conflict, employ backjumping to the asserting level, and not use phase saving. The last two conditions are merely for clarity of exposition. As we will later argue, our results hold irrespective of whether or not the solver uses backjumping or phase saving.

We describe how a non-restarting CDCL SAT solver using a simple preprocessing step can efficiently simulate the behavior of a CDCL solver that employs as many as $2^m - 1$ restarts. The simulation uses the following formula:

Definition 4. Let $Y = \{u, v, y_0, \dots y_{m-1}, z_0, \dots, z_{m-1}\}$ be a set of $2m + 2$ Boolean variables. The CNF formula $G_m(Y)$ with $4m + 1$ clauses over Y is defined as:

$$G_m(Y) = (u \vee \overline{z_0} \vee \dots \vee \overline{z_{m-1}}) \wedge \bigwedge_{i=0}^{m-1} \begin{pmatrix} (u \vee y_i \vee z_i \vee v) \\ \wedge (u \vee y_i \vee z_i \vee \overline{v}) \\ \wedge (u \vee \overline{y_i} \vee z_i \vee v) \\ \wedge (u \vee \overline{y_i} \vee z_i \vee \overline{v}) \end{pmatrix}$$

We first see how a CDCL solver S operating on $G_m(Y)$ acts like an m-bit counter. We can think of y as an m-bit number that starts at 0 and is incremented by 1 after every 2 conflicts when S operates on $G_m(Y)$. Later, we will interleave the execution of S on $G_m(Y)$ with the execution of S on $F_n(X)$, and the counter will be incremented by 1 (and the number of conflicts involving $G_m(Y)$ will increase by 2) each time S would have restarted on $F_n(X)$. To understand the operation of the counter, we use the following definition for clauses that are equivalent w.r.t. unit propagation.

Definition 5. We say that two clauses C and D are *1-equivalent* with respect to a CNF formula F' if and only if for every partial assignment σ and literal ℓ, ℓ follows by unit propagation from $(F' \wedge C)|_\sigma$ if and only ℓ follows by unit propagation from $(F' \wedge D)|_\sigma$. When 1-equivalence is defined between potential learned clauses at the same conflict in a CDCL solver, the set F' is implicitly fixed to be the conjunction of the input clauses plus all prior learned clauses. It is easy to see that the unit propagations made and conflicts found by a CDCL solver do not distinguish between 1-equivalent learned clauses.

Lemma 1. *Consider the execution of any non-restarting CDCL solver S on input $G_m(Y)$ that chooses decision variables $u, y_{m-1}, \dots, y_0, v$ in order, each with initial phase 0 at every option, and learns one asserting clause per conflict. For $0 \le J \le 2^m - 1$, write J in binary as $J_{m-1} \dots J_0$, let $\nu_2(J) = \min\{i \mid J_i = 1\}$, and write $\vee y_J$ for the disjunction consisting of all y_i such that $J_i = 0$. Then,*

- *the $(2J + 1)$-st conflict in the execution of S occurs when $u = 0, y_{m-1} = J_{m-1}, \dots, y_0 = J_0, v = 0$, and the asserting clause learned by S is 1-equivalent to $(u \vee (\vee y_J) \vee v)$, which we denote by $D_{J,v}$, and*
- *after the $(2J+2)$-nd conflict, the asserting clause learned by S is 1-equivalent to $(u \vee (\vee y_J))$, which we denote by D_J.*

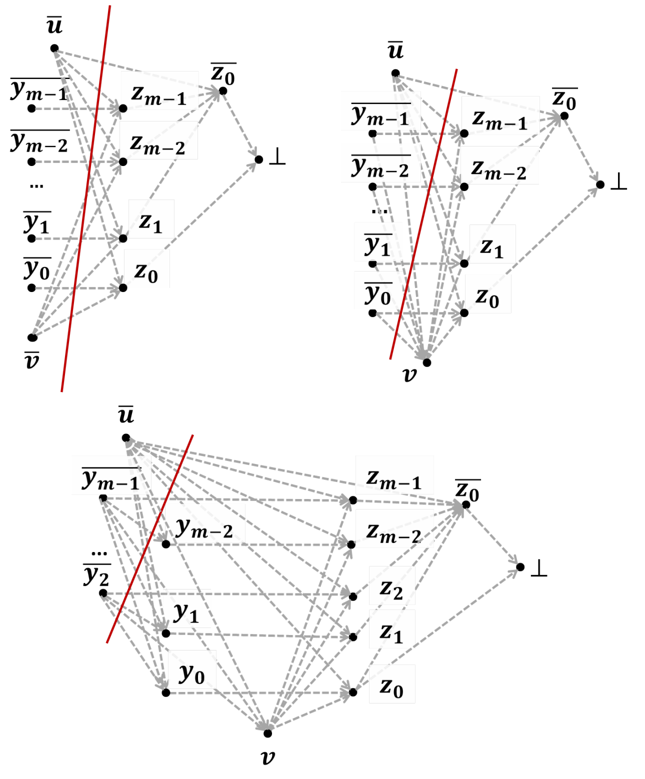

Figure 1: Conflict graphs for (a) 1st conflict (b) 2nd conflict (c) conflict $2J$ for some J.

Moreover, the $(2J + 2)$-nd conflict occurs after at most $m + 4J + 2$ variable assignments are made either via decisions or unit propagation.

Proof. For simplicity, we denote the clauses of $G_m(Y)$ by $G_0 = (u \vee \overline{z_0} \vee \dots \vee \overline{z_{m-1}})$, $G_{i11} = (u \vee y_i \vee z_i \vee v)$, $G_{i10} = (u \vee y_i \vee z_i \vee \overline{v})$, $G_{i01} = (u \vee \overline{y_i} \vee z_i \vee v)$, and $G_{i00} = (u \vee \overline{y_i} \vee z_i \vee \overline{v})$. We start with $J = 0$. After the solver sets decision variables $u, y_{m-1}, \dots, y_0$ to 0 in order, setting v to 0 yields each z_i by unit propagation using clauses G_{i11}, which results in a conflict with clause G_0. The conflict graph is shown in Figure 1(a) (where the final conflicting literal is shown as z_0 w.l.o.g.). Since all propagated literals are at decision level $m + 2$ (associated with v), the only asserting clause to learn from this conflict is the one that contains the negation of the decision variables, namely $D_{0,v}$, with asserting level $m + 1$. Learning $D_{0,v}$ and backtracking to level $m + 1$ causes a propagation of v after the $y_0 = 0$ assignment, and then unit propagation with each G_{i10} again yields a conflict with clause G_0. The conflict graph is shown in Figure 1(b) and again because all propagated literals are at decision level $m + 1$ (associated with y_0), the only absorbing clause to learn is the one that contains the negation of the decision variables, namely D_0, with asserting level m.

In a more general situation for conflicts $2J + 1$ and $2J + 2$, assume by the inductive hypothesis that the solver has learned clauses that are 1-equivalent to $D_{0,v}, D_0, \dots, D_{J-1,v}, D_{J-1}$. Let $i^* = \nu_2(J)$. For each

$i \geq i^*$ for which $J_i = 1$, define $J^{[i]} < J$ to be the integer whose binary expansion agrees with that of J for all $i' \in [i+1, m-1]$ and ends in 01^i. In particular, observe that using the order $y_{m-1}, \ldots, y_0$, for all i with $J_i = 1$, if we set $y_{i'} = 0$ for each $i' \geq i$ such that $J_{i'} = 0$ then unit propagation using the learned clause 1-equivalent to $D_{J^{[i]}}$ will infer $y_i = 1$. Therefore, setting the decision variables $u, (y_i)_{J_i=0}$, in order, all to 0, will, by unit propagation, yield the partial assignment with $y_i = J_i$ for all i. Now setting decision variable v to 0 will yield a conflict (using clauses G_0 and G_{i01} or G_{i11} depending on the assignment to y_i) and any asserting clause that might be learned must involve all the decision variables and hence contain all of $D_{J,v}$ plus some subset of the $\overline{y_i}$ for those y_i that were set to 1 by unit propagation from previously learned clauses and the decision variables. Such an asserting clause is clearly 1-equivalent to $D_{J,v}$ given the learned clauses so far. This in turn immediately propagates v and then using clauses G_0 and G_{i00} or G_{i10} produces a conflict that infers a clause 1-equivalent to asserting clause D_J as shown in the example in Figure 1(c). (The cut shown there is precisely D_J, but the actual cut may also include some $\overline{y}_i$ for some of the unit-propagated y_i literals.) We observe that the asserting level is the one associated with variable y_{i^*} for $i^* = \nu_2(J)$. The solver then backtracks to this level, sets decision variables $y_i = 0$ for $i < i^*$, and continues inductively with the new assignment.

Observe that there is always a decision on v at conflicts $2J + 1$ and $2J + 2$. Further, at conflict $2J + 2$, the value of y_0 is always changed, the value of y_1 is changed when $J+1$ is even, the value of y_2 is changed when $J+1$ is a multiple of 4, and, in general, the value of y_i is changed when $J + 1$ is a multiple of 2^i. Therefore the total number of variable assignments (either via decision or unit propagation) up through $2J+2$ conflicts is at most $m+2J+2+\sum_{i=0}^{m-1} \lfloor (J+1)/2^i \rfloor \leq m + 4J + 2$. ☐

Remark 1. In the above proof, literals at all decision levels are involved in all conflicts. Hence, the asserting level is always one less than the current decision level, making *backjumping* behave the same as chronological backtracking.

Remark 2. S operating on $G_m(Y)$ acts like a counter even if S uses *phase saving*, albeit the semantics of the underlying counter changes to that of a *Gray code* (Gray 1953). In a Gray code counter, the bits change in exactly one position when the counter is incremented. The position that changes is precisely the least significant bit that yields a previously unseen number when flipped. E.g., a 3-bit Gray code counter proceeds like 000, 001, 011, 010, 110, 111, 101, and 100. The only change in the operation of S when it uses phase saving is that upon backjumping to the asserting level after the $(2J + 2)$-nd conflict, rather than proceeding to set y_i to 0 for $i < i^*$, it will set y_i to their saved phases. It can be verified that the asserted literal will correspond to the bit in the Gray code that flips when the counter increments from J to $J + 1$. The overall structure of the argument remains intact, only the polarities of some of the literals in clauses $D_{J,v}$ and D_J change.

Let $F_n(X)$ be a CNF formula defined over n variables $X = \{x_1, x_2, \ldots, x_n\}$ such that $X \cap Y = \phi$. The preprocessing of $F_n(X)$ that we referred to earlier simply conjoins it with $G_m(Y)$, for an appropriately chosen m. Our main result is that the execution of a CDCL solver employing R restarts on $F_n(X)$ can be simulated with very little overhead by a non-restarting CDCL solver operating on $F_n(X) \wedge G_m(Y)$ as long as $m \geq \lceil \log_2(R + 1) \rceil$.

Theorem 1. *For every execution of a CDCL solver S that learns one asserting clause per conflict and uses at most $R \leq 2^m - 1$ restarts on CNF formula $F_n(X)$ on n variables, there is a non-restarting CDCL solver S' that learns one asserting clause per conflict, that on input $F_n(X) \wedge G_m(Y)$ makes precisely the same literal selection decisions as S on X, that makes precisely the same set of inferences involving the variables in X as S does on input $F_n(X)$, and that in total makes at most an additional $\mathrm{O}(m + R)$ variable assignments on Y.*

Proof. The solver S' on $F_n(X) \wedge G_m(Y)$ will exactly simulate S on $F_n(X)$ except that it will interleave the execution of a CDCL solver as shown in Lemma 1 with the actions of S on $F_n(X)$. Note that since the X and Y variables never occur in a clause together, the learned clauses on $F_n(X)$ and on $G_m(Y)$ will depend only on $F_n(X)$ or on $G_m(Y)$, respectively. In particular, no learned clause will have variables from both X and Y.

1. S' begins by branching on decision variables $u, y_{m-1}, \ldots, y_0$, in order, with preferred phase 0.
2. It then simulates S on $F_n(X)$ until S would restart, or until it learns a unit clause on the X variables and wants to backjump to decision level 0. In the former case, let this be the $(J + 1)$-st time S wants to restart and proceed to step 3. In the latter case, S' backjumps to decision level 0 and repeats the same branching decisions (and corresponding propagations) on the Y variables as their current phase.
3. Instead of restarting, S' makes a branching decision on $\overline{v}$. By Lemma 1, this causes a conflict on $G_m(Y)$ and S' learns a clause 1-equivalent to D_{2J+1} involving only the Y variables. The asserting level b of this learned clause clearly must be the level of some Y variable. S' backjumps to level b. This has the effect of retracting all decisions on X variables just as S would in a restart.
4. S' immediately realizes another conflict, learns a clause 1-equivalent to D_{2J+2} as in Lemma 1, and continues to operate on $G_m(Y)$ until just after the value of y_0 is set. S' now returns to step 2.

After R restarts, the total number of variable assignments on Y is at most $m + 2 + 4R$ by Lemma 1. ☐

Remark 3. Even if S' does not employ *backjumping*, the simulation will still work. In this case, S' will learn clauses on Y variables 1-equivalent to D_{2J+1} and D_{2J+2} in steps 2 and 3 while still at the decision level of some X variable. However, having learned the clause 1-equivalent to D_{2J+2}, which involves only Y literals all of which are falsified by the current assignment, S' will continue to infer immediate conflicts and keep backtracking chronologically until it reaches the asserting level b referenced in step 3 above.

Remark 4. We note that every clause of $G_m(Y)$ contains the pure literal u. Therefore if pure literal elimination were run during preprocessing by a CDCL solver then it would simply remove the clauses of $G_m(Y)$. However, it is not hard to modify $G_m(Y)$ by appending extra clauses H obtained by adding $\overline{u}$ to an arbitrary satisfiable formula on a new small set of extra variables that does not contain any pure literals. The pure literal preprocessing would then not prune the clauses of $G_m(Y)$ and the initial branch setting $u = 0$ would immediately satisfy these extra clauses H which would eliminate their impact on the CDCL execution in any of our arguments.

Simulating Resolution Without Restarts

We begin with the observation that if a given refutation P is too large, the simulation can simply discard it and instead start with an obvious tree-like resolution refutation with at most $2^n - 1$ derived clauses.

Lemma 2. *Let P be a given resolution refutation of $F_n(X)$. If length$(P) > (2^n - 1)/n^2$, then there is a non-restarting CDCL solver that, even without employing clause learning, when run on $F_n(X)$ produces a tree-like resolution refutation after fewer than $n^2 \cdot$ length(P) branching decisions.*

Proof. Let S be a non-restarting CDCL solver that simply discards P and instead simulates an obvious tree-like resolution refutation P' (w.r.t. any fixed variable order) of $F_n(X)$. P' essentially enumerates the 2^n possible variable assignments σ and identifies a clause of $F_n(X)$ (or a learned clause, if S uses clause learning) falsified by each such σ. P' will need no more than $2 \cdot (2^{n-1} - 1) = 2^n - 2$ branching decisions as the $(n-1)$-st assignment for any σ will either satisfy all clauses of $F_n(X)$ or unit propagate the remaining variable. By the precondition on the length of P, the number of branching decisions is thus smaller than $n^2 \cdot$ *length*(P). □

In the rest of this section, we will thus assume w.l.o.g. that *length*$(P) \leq (2^n - 1)/n^2$. Pipatsrisawat and Darwiche (2011) provide a CDCL simulation of P, henceforth referred to as the *PD simulation*, that uses $R \leq n^2 \cdot$*length*(P) restarts and has size $\mathrm{O}(n^4 \cdot$ *length*$(P))$. By the above observation regarding *length*(P), we have $R \leq (2^n - 1)$. Applying Theorem 1 using a counter with $m = n$ bits thus yields the following:

Corollary 1. *Let P be a resolution refutation of $F_n(X)$. Then there is a non-restarting CDCL solver that infers any one asserting clause per conflict when run on formula $F_n(X) \wedge G_n(Y)$ produces a resolution refutation of $F_n(X)$ of size $\mathrm{O}(n^4 \cdot$ length$(P))$.*

We can strengthen this by using a variant of the PD simulation together with a somewhat sharpened analysis.

Theorem 2. *Let P be a resolution refutation of $F_n(X)$. Then a resolution refutation of $F_n(X)$ of length $\mathrm{O}(n^2 \cdot$ size$(P))$ may be found using only $\mathrm{O}(n^2 \cdot$ size$(P))$ variable assignments by a non-restarting CDCL solver that infers any one asserting clause per conflict when run on formula $F_n(X) \wedge G_n(Y)$.*

Algorithm 1: p-simulation of a resolution refutation by a CDCL solver with simple preprocessing

```
Input  : CNF formula $F_n(X)$ with a resolution
         refutation $P = (C_1, \ldots, C_s)$
Output: A resolution refutation $Q$ of $F_n(X)$
1  begin
2      $F' \leftarrow F_n(X) \wedge G_n(Y)$
3      $Q \leftarrow$ a sequence with all clauses of $F_n(X)$
4      $J \leftarrow 0$
5      for $k$ in $1..s$ do
6          if $C_k$ is absorbed by $F'$ then
7              continue
8          Continue CDCL execution on the $Y$ variables
               as in Lemma 1 until $y_0$ is assigned a value
9          Let $d$ be the current decision level
10         while $C_k$ is not absorbed by $F'$ do
11             Let $\ell \in C_k$ be a literal witnessing that $C_k$ is
                   not absorbed by $F'$
12             repeat
13                 Branch on $\overline{\ell'}$ for all $\ell' \in C_k \setminus \{\ell\}$
14                 Branch on $\overline{\ell}$ at decision level $d + |C_k|$
15                 Realize a conflict involving only $X$
16                 Derive an asserting clause $C$ over $X$
17                 Append to $Q$ resolution proof of $C$
                       from $F'$ given by the conflict graph
18                 $F' \leftarrow F' \wedge C$
19                 $b \leftarrow$ asserting level of $C$ (0 or $> d$)
20                 Backjump to decision level $b$
21             until $C_k$ is absorbed by $F'$ w.r.t. $\ell$
22             if $b > d$ then
23                 Branch on $\overline{v}$ at level $b + 1$ (continues
                       execution on $G_n(Y)$ as in Lemma 1)
24                 Learn clause 1-equivalent to $D_{2J+1}$
                       with asserting level $d$
25                 Backtrack to decision level $d$
26                 Learn clause 1-equivalent to $D_{2J+2}$
                       with asserting level $d - 1$
27                 Backtrack to decision level $d - 1$
28                 $J \leftarrow J + 1$
29     Backtrack to decision level 0; Realize a conflict
30     return $Q$
31 end
```

Proof. If *length*$(P) > (2^n - 1)/n^2$, the result follows from Lemma 2. Otherwise we proceed as follows. We assume w.l.o.g. that P is minimal. We observe that since $G_n(Y)$ is satisfiable and on disjoint variables from $F_n(X)$, any minimal resolution refutation of $F_n(X) \wedge G_n(Y)$ must be a refutation of $F_n(X)$.

The CDCL solver execution is given by Algorithm 1. The simulation begins (Line 2) by setting F' to $F_n(X) \wedge G_n(Y)$. The idea, following the PD simulation, is to have F' absorb each of the inferred clauses one by one. When the outer for-loop (Line 5) of the algorithm finishes, F' must have absorbed the empty clause and hence produced a resolution refutation of $F_n(X) \wedge G_n(Y)$ and thus of $F_n(X)$.

We start (Line 8) by branching on the Y variables as in Lemma 1 until y_0 is assigned a value at some decision level d. In order to absorb a clause C_k that is currently not absorbed by F', the simulation identifies a literal $\ell \in C$ witnessing that C_k is not yet absorbed (Line 11) and then branches negatively on all literals of C_k other than ℓ (Line 13). Since C is not absorbed, making these $|C_k| - 1$ branching decisions will be possible in spite of eager unit propagation, will not cause a conflict by unit propagation, and will allow branching on $\overline{\ell}$. This last branch (Line 14), however, *will* cause a conflict because $C_1, \ldots C_{k-1}$ are already absorbed by F'. From this conflict, S will learn an asserting clause C (Line 16) whose resolution derivation involves ℓ. Since these clauses are only derived from $F_n(X)$, C must be a clause over X. If the asserted literal in C is different from ℓ and C_k remains unabsorbed w.r.t. ℓ, this process of branching on $\overline{\ell}$ is repeated (Lines 12-21) no more than $n - |C_k| + 1$ times until ℓ becomes the asserted literal (this holds because all asserted variables here must be distinct and cannot belong to $C_k \setminus \{\ell\}$). S will backtrack to some asserting level b (Line 19), which must either be the level of some X variable or be 0 (if S learns a unit clause). At this point, we have absorbed C_k w.r.t. ℓ. If $b > d$, however, the "leftover" branching decisions on the X variables of C_k may interfere with the derivation of clauses needed to absorb C_k w.r.t. other literals or to absorb $C_{k+1}, \ldots C_s$. In order to avoid this situation, the PD simulation simply restarts the solver and forcibly returns to decision level 0. Since S is a non-restarting CDCL solver, our construction employs the counter involving the Y variables and a decision setting $v = 0$ (Line 23) to backtrack over any leftover branching decisions on the X variables that remain after absorbing C_k. This increments the counter and brings us back to decision level $d - 1$ (Lines 24-27).

Repeating this process (Lines 10-28) for other literals of C_k witnessing that C_k is not yet absorbed will eventually result in C_k being absorbed. Thus, after no more than $(n - |C_k| + 1)|C_k|$ repetitions, each of which involves at most $|C_k|$ branching decisions or at most n resolution steps, we are able to absorb C_k. This yields a total of at most $n \sum_{k=1}^{s} |C_k|^2$ branching decisions and at most $n^2 \sum_{k=1}^{s} |C_k| = n^2 \cdot size(P)$ resolution steps, to absorb all clauses of P.

The total number R of restarts needed to be simulated is at most $n^2 \cdot length(P)$ which, by our earlier argument, is at most $2^n - 1$. Hence, $\log_2(R+1) \leq n$ and, from Theorem 1, it suffices to use $G_n(Y)$ as the counter for $F_n(X)$. □

Concluding Remarks

Known efficient simulations of resolution refutations by modern CDCL SAT solvers that learn one asserting clause per conflict have so far relied heavily on the ability of such solvers to restart several times per resolution derivation step that they are trying to simulate. Our result shows that simple preprocessing and an associated branching heuristic can replace the need for these solvers to explicitly restart. As a consequence, from a theoretical standpoint, such solvers are in fact powerful enough to efficiently simulate resolution refutations without relying on restarts at all.

The result holds for all CDCL solvers that learn (at least) one asserting clause from each conflict, irrespective of whether or not they employ other common techniques such as backjumping and phase saving. As remarked briefly earlier, the construction can be extended to withstand simplification techniques such as pure literal elimination.

The kind of preprocessing used in our non-restarting simulation is admittedly of a rather different nature than the commonly used preprocessing techniques geared towards increasing the efficiency of SAT solvers. It adds a counting formula on a brand new set of variables such that branching first on (a subset of) these new variables enables the solver to freely backtrack out of any "leftover" partial assignment after realizing a conflict.

While our simulation result answers an open theoretical question, research on effective restart strategies continues to be relevant for CDCL solvers in practice. The question of whether modern CDCL solvers can polynomially simulate resolution proofs without restarting and without preprocessing the input formula remains open.

Acknowledgments

The work of the first author was supported by NSF grant CCF-1217099. Part of this research was conducted when the authors were visiting the BIRS Workshop on Theoretical Foundations of Applied SAT Solving during January 2014.

References

Atserias, A.; Fichte, J. K.; and Thurley, M. 2009. Clause-learning algorithms with many restarts and bounded-width resolution. In *12th SAT*, volume 5584 of *LNCS*, 114–127.

Atserias, A.; Fichte, J. K.; and Thurley, M. 2011. Clause-learning algorithms with many restarts and bounded-width resolution. *JAIR* 40:353–373.

Audemard, G., and Simon, L. 2012. Refining restarts strategies for SAT and UNSAT. In *18th CP*, volume 7514 of *LNCS*, 118–126.

Beame, P.; Kautz, H.; and Sabharwal, A. 2003. Understanding the power of clause learning. In *18th IJCAI*, 1194–1201.

Beame, P.; Kautz, H.; and Sabharwal, A. 2004. Understanding and harnessing the potential of clause learning. *JAIR* 22:319–351.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2009. *Handbook of Satisfiability*. IOS Press.

Buss, S. R., and Bonet, M. L. 2012. An improved separation of regular resolution from pool resolution and clause learning. In *15th SAT*, volume 7313 of *LNCS*, 244–57.

Buss, S. R., and Kolodziejczyk, L. 2012. Small stone in pool. Manuscript, 2012.

Buss, S. R.; Hoffmann, J.; and Johannsen, J. 2008. Resolution trees with lemmas: Resolution renements that characterize DLL-algorithms with clause learning. *Logical Methods in Computer Science* 4(4):13.

de Kleer, J., and Williams, B. C. 1987. Diagnosing multiple faults. *AI J.* 32(1):97–130.

Frost, D.; Rish, I.; and Vila, L. 1997. Summarizing csp hardness with continuous probability distributions. In *14th AAAI*, 327–333.

Gomes, C. P.; Selman, B.; and Crato, N. 1997. Heavy-tailed distributions in combinatorial search. In *3rd CP*, volume 1330 of *LNCS*, 121–135.

Gomes, C. P.; Selman, B.; and Kautz, H. 1998. Boosting combinatorial search through randomization. In *15th AAAI*, 431–437.

Gray, F. 1953. Pulse code communication. US Patent #2632058, (filed 1947).

Haim, S., and Walsh, T. 2009. Restart strategy selection using machine learning techniques. In *12th SAT*, volume 5584 of *LNCS*, 312–325.

Hertel, P.; Bacchus, F.; Pitassi, T.; and Van Gelder, A. 2008. Clause learning can effectively p-simulate general propositional resolution. In *23rd AAAI*, 283–290.

Hogg, T., and Williams, C. P. 1994. Expected gains from parallelizing constraint solving for hard problems. In *12th AAAI*, 331–336.

Luby, M.; Sinclair, A.; and Zuckerman, D. 1993. Optimal speedup of Las Vegas algorithms. *Inf. Process. Lett.* 47(4):173–180.

Marques-Silva, J. P., and Sakallah, K. A. 1996. GRASP – a new search algorithm for satisfiability. In *ICCAD*, 220–227.

Marques-Silva, J. P.; Lynce, I.; and Malik, S. 2009. CDCL solvers. In Biere et al. (2009). chapter 4, 131–154.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *38th DAC*, 530–535.

Pipatsrisawat, K., and Darwiche, A. 2009. On the power of clause-learning SAT solvers with restarts. In *15th CP*, volume 5732 of *LNCS*, 654–668.

Pipatsrisawat, K., and Darwiche, A. 2011. On the power of clause-learning SAT solvers as resolution engines. *AI J.* 175(2):512–525.

Robinson, J. A. 1965. A machine-oriented logic based on the resolution principle. *J. Assoc. Comput. Mach.* 12(1):23–41.

Stallman, R. M., and Sussman, G. J. 1977. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *AI J.* 9:135–196.

Van Gelder, A. 2005. Pool resolution and its relation to regular resolution and DPLL with clause learning. In *12th Intl. Conf. Logic for Prog., AI, and Reason.*, volume 3835 of *LNCS*, 580–594.

Walsh, T. 1999. Search in a small world. In *16th IJCAI*, 1172–1177.

Zhang, L.; Madigan, C. F.; Moskewicz, M. H.; and Malik, S. 2001. Efficient conflict driven learning in a Boolean satisfiability solver. In *ICCAD*, 279–285.