Understanding the Power of Clause Learning

Paul Beame* **Henry Kautz*** **Ashish Sabharwal***

Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350
{beame,kautz,ashish}@cs.washington.edu

Abstract

Efficient implementations of DPLL with the addition of clause learning are the fastest complete satisfiability solvers and can handle many significant real-world problems, such as verification, planning, and design. Despite its importance, little is known of the ultimate strengths and limitations of the technique. This paper presents the first precise characterization of clause learning as a proof system, and begins the task of understanding its power. In particular, we show that clause learning using any non-redundant scheme and unlimited restarts is equivalent to general resolution. We also show that without restarts but with a new learning scheme, clause learning can provide exponentially smaller proofs than regular resolution, which itself is known to be much stronger than ordinary DPLL.

1 Introduction

In recent years the task of deciding whether a CNF propositional logic formula is satisfiable has gone from a problem of theoretical interest to a practical approach for solving real-world problems. SAT procedures are now a standard tool for hardware verification, including verification of super-scalar processors [Velev and Bryant, 2001; Biere *et al.*, 1999]. Open problems in group theory have been encoded and solved using satisfiability [Zhang and Hsiang, 1994]. Other applications of SAT include circuit diagnosis and experiment design [Konuk and Larrabee, 1993; Gomes *et al.*, 1998b].

The most surprising aspect of such relatively recent practical progress is that the best complete satisfiability testing algorithms remain variants of the DPLL procedure [Davis and Putnam, 1960; Davis *et al.*, 1962] for backtrack search in the space of partial truth assignments. The main improvements to DPLL have been better branch selection heuristics (*e.g.* , [Li and Anbulagan, 1997]), and extensions such as randomized restarts [Gomes *et al.*, 1998a] and clause learning. One can argue that clause learning has been the most significant of these in scaling DPLL to realistic problems.

Clause learning grew out of work in AI on explanation-based learning (EBL), which sought to improve the performance of backtrack search algorithms by generating explanations for failure (backtrack) points, and then adding the explanations as new constraints on the original problem [de Kleer and Williams, 1987; Stallman and Sussman, 1977; Genesereth, 1984; Davis, 1984]. For general constraint satisfaction problems the explanations are called "conflicts" or "no goods"; in the case of Boolean CNF satisfiability, the technique becomes clause learning. A series of researchers [Bayardo Jr. and Schrag, 1997; Marques-Silva and Sakallah, 1996; Zhang, 1997; Moskewicz *et al.*, 2001; Zhang *et al.*, 2001] showed that clause learning can be efficiently implemented and used to solve hard problems that cannot be approached by any other technique.

Despite its importance there has been little work on formal properties of clause learning, with the goal of understanding its fundamental strengths and limitations. A likely reason for such inattention is that clause learning is a rather complex rule of inference – in fact, as we describe below, a complex family of rules of inference. A contribution of this paper is that we provide a precise specification of clause learning.

Another problem in characterizing clause learning is defining a formal notion of the strength or power of a reasoning method. This paper uses the notion of proof complexity [Cook and Reckhow, 1977], which compares inference systems in terms of the sizes of the shortest proofs they sanction. A family of formulas C provides an *exponential separation* between systems S_1 and S_2 if the shortest proofs of formulas in C in system S_1 are exponentially smaller than the corresponding shortest proofs in S_2. From this basic propositional proof complexity point of view, only families of unsatisfiable formulas are of interest, because only proofs of unsatisfiability can be large; minimum proofs of satisfiability are linear in the number of variables of the formula. Nevertheless, Achlioptas *et al* [2001] have shown how negative proof complexity results for unsatisfiable formulas can be used to derive time lower bounds for specific inference algorithms running on satisfiable formulas as well.

Proof complexity does not capture everything we intuitively mean by the power of a reasoning system, because it says nothing about how difficult it is to find shortest proofs. However, it is a good notion with which to begin our analysis, because the size of proofs provides a lower-bound on the running time of any implementation of the system. In the systems we consider, a branching function, which determines

* Research supported by NSF Grant ITR-0219468

which variable to split upon or which pair of clauses to resolve, guides the search. A negative proof complexity result for a system tells us that a family of formulas is intractable even with a perfect branching function; likewise, a positive result gives us hope of finding a branching function.

A basic result in proof complexity is that general resolution is exponentially stronger than the DPLL procedure [Bonet *et al.*, 2000; Ben-Sasson *et al.*, 2000]. This is because the trace of DPLL running on an unsatisfiable formula can be converted to a tree-like resolution proof of the same size, and tree-like proofs must sometimes be exponentially larger than the DAG-like proofs generated by general resolution. Although resolution can yield shorter proofs, in practice DPLL is better because it provides a more efficient way to search for proofs. The weakness of the tree-resolution proofs that DPLL finds is that they do not reuse derived clauses. The conflict clauses found when DPLL is augmented by clause learning correspond to reuse of derived clauses in the associated resolution proofs and thus to more general forms of resolution proofs. An intuition behind the results in this paper is that the addition of clause learning moves DPLL closer to general resolution while retaining its practical efficiency.

It has been previously observed that clause learning can be viewed as adding resolvents to a tree-like proof [Marques-Silva, 1998]. However, this paper provides the first mathematical *proof* that clause learning is exponentially stronger than tree-like resolution. Further, we provide a family of formulas that exponentially separates clause learning from regular resolution, a system that is known to be intermediate in strength between general and tree resolution. The proof uses a new clause learning scheme called FirstNewCut that we introduce. We also show that combining clause learning with restarts is as strong as general resolution.

Although this paper focuses on basic proof complexity, we briefly indicate how the understanding we gain through this kind of analysis may lead to practical applications. For example, our proofs describe an improvement to the clause learning rules previously suggested in the literature, and suggest an approach to leveraging the structure of a problem encoded as a CNF formula in order to create a branching heuristic that takes the greatest advantage of clause learning. As an example, we apply these ideas to certain natural satisfiable and unsatisfiable formulas where we obtain significant speed-ups over existing methods.

2 Preliminaries

A CNF formula F is an AND ($\wedge$) of *clauses*, where each clause is an OR ($\vee$) of *literals*, and a literal is a variable or its negation ($\neg$). It is natural to think of F as a set of clauses and each clause as a set of literals. A clause that is a subset of another is called its *subclause*.

Let ρ be a partial assignment to the variables of F. The *restricted formula* $F|\rho$ is obtained from F by replacing variables in ρ with their assigned values. F is said to be *simplified* if all clauses with at least one TRUE literal are deleted, all occurrences of FALSE literals are removed from clauses, and the resulting formula, if different, is simplified recursively.

2.1 The DPLL Procedure

The basic idea of the Davis-Putnam-Logemann-Loveland (DPLL) procedure [Davis and Putnam, 1960; Davis *et al.*, 1962] for testing satisfiability of CNF formulas is to *branch* on variables, setting them to TRUE or FALSE, until either an initial clause is *violated* (*i.e.* has all literals set to FALSE) or all variables have been set. In the former case, we backtrack to the last branching decision whose other branch has not been tried yet, reverse the decision, and proceed recursively. In the latter, we terminate with a satisfying assignment. If all possible branches have been unsuccessfully tried, the formula is declared unsatisfiable. To increase efficiency, *pure literals* (those whose negation does not appear) and *unit clauses* (those with only one unset literal) are immediately set to true.

Definition 1. A *branching sequence* for a CNF formula F is a sequence $\sigma = (l_1, l_2, \ldots, l_k)$ of literals of F, possibly with repetitions. DPLL on F *branches according to* σ if it always picks the next variable v to branch on in the literal order given by σ, skips it if v is currently assigned a value, and branches further by setting the chosen literal to FALSE otherwise.

In this paper, we will use the term DPLL to denote the basic branching and backtracking procedure described above. It will, for instance, not include extensions such as learning conflict clauses or restarting, but will allow intelligent branching heuristics. Note that this is in contrast with the occasional use of the term DPLL to encompass practically all branching and backtracking approaches, including those involving learning.

2.2 Resolution

Resolution is a simple proof system that can be used to prove unsatisfiability of CNF formulas. The *resolution rule* states that given clauses $(A \vee x)$ and $(B \vee \neg x)$, we can derive clause $(A \vee B)$ by *resolving on* x. A *resolution derivation* of C from a CNF formula F is a sequence $\pi = (C_1, C_2, \ldots, C_s \equiv C)$ where each clause C_i is either a clause of F (an *initial* clause) or derived by applying the resolution rule to C_j and C_k, $j, k < i$ (a *derived* clause). The *size* of π is s, the number of clauses occurring in it. We will assume that each $C_j \neq C$ in π is used to derive at least one other clause $C_i, i > j$. Any derivation of the empty clause Λ from F, also called a *refutation* or *proof* of F, shows that F is unsatisfiable.

Despite its simplicity, resolution is not efficiently implementable due to the difficulty of finding good choices of clauses to resolve; natural choices typically yield huge storage requirements. Various restrictions on the structure of resolution proofs lead to less powerful but easier to implement variants such as tree-like, regular, linear and positive resolution. *Tree-like* resolution uses non-empty derived clauses exactly once in the proof and is equivalent to an optimal DPLL procedure. *Regular* resolution allows any variable to be resolved upon at most once along any "path" in the proof from an initial clause to Λ. Both these variants are sound and complete but differ in efficiency – regular resolution is known to be exponentially stronger than tree-like [Bonet *et al.*, 2000; Ben-Sasson *et al.*, 2000], and general resolution is exponentially stronger than regular [Alekhnovich *et al.*, 2002].

Definition 2. A resolution derivation $(C_1, C_2, \ldots, C_k)$ is *trivial* iff all variables resolved upon are distinct and each

$C_i, i \geq 3$, is either an initial clause or is derived by resolving C_{i-1} with an initial clause.

A trivial derivation is tree-like as well as regular. Moreover, the condition that each derived clause C_i use C_{i-1} in its derivation makes it *linear*. As we will see, trivial derivations correspond to conflicts in clause learning algorithms.

3 Clause Learning

Clause learning proceeds by following the normal branching process of DPLL until there is a "conflict" after unit propagation. If this conflict occurs without any branches, the formula is declared unsatisfiable. Otherwise, the "conflict graph" is analyzed and the "cause" of the conflict is learned in the form of a "conflict clause." We now backtrack and continue as in ordinary DPLL, treating the learned clause just like initial ones. A clause is said to be *known* at a stage if it is either an initial clause or has already been learned. The learning process is expected to save us from redoing the same computation when we later have an assignment that causes conflict due in part to the same reason.

If a given CNF formula F is unsatisfiable, clause learning terminates with a conflict without any branches. Since all clauses used in this conflict themselves follow directly or indirectly from F, this failure of clause learning in finding a satisfying assignment constitutes a logical proof of unsatisfiability of F. Our bounds compare the size of such a proof with the size of a (possibly restricted) resolution proof of unsatisfiability of F.

Variations of such conflict driven learning [Bayardo Jr. and Schrag, 1997; Marques-Silva and Sakallah, 1996; Zhang, 1997; Moskewicz *et al.*, 2001; Zhang *et al.*, 2001] include different ways of choosing the clause to learn and possibly allowing multiple clauses to be learned from a single conflict. Although many such algorithms have been proposed and demonstrated to be empirically successful, a theoretical discussion of the underlying concepts and structures needed for our bounds is lacking. The rest of this section focuses on this formal framework.

Definition 3. A *clause learning proof* π of an unsatisfiable CNF formula F under scheme $\mathcal{S}$ and induced by branching sequence σ is the result of applying DPLL with unit propagation on F, branching according to σ, and using scheme $\mathcal{S}$ to learn conflict clauses such that at the end of this process, there is a conflict without any branches. The *size* of the proof, $size(\pi)$, is $|\sigma|$.

All clause learning algorithms discussed in this paper are based on *unit propagation*, which is the process of repeatedly applying the unit clause rule followed by formula simplification until no clause with exactly one unassigned literal remains. In this context, it is convenient to work with residual formulas at different stages of DPLL. Let ρ be the partial assignment at some stage of DPLL on formula F. The *residual formula* at this stage is obtained by simplifying $F|\rho$ and applying unit propagation.

When using unit propagation, variables assigned values through the actual branching process are called *decision* variables and those assigned values as a result of unit propagation are called *implied* variables. *Decision and implied literals* are analogously defined. Upon backtracking, the last decision variable no longer remains a decision variable and might instead become an implied variable depending on the clauses learned so far. The *decision level of a decision variable* x is one more than the number of current decision variables at the time of branching on x. The *decision level of an implied variable* is the maximum of the decision levels of decision variables used to imply it. The *decision level* at any step of the underlying DPLL procedure is the maximum of the decision levels of all current decision variables.

3.1 Implication Graph and Conflicts

Unit propagation can be naturally associated with an *implication graph* that captures all possible ways of deriving all implied literals from decision literals.

Definition 4. The *implication graph* G at a given stage of DPLL is a directed acyclic graph with edges labeled with sets of clauses. It is constructed as follows:

1. Create a node for each decision literal, labeled with that literal. These will be the indegree zero root nodes of G.
2. While there exists a known clause $C = (l_1 \vee \dots l_k \vee l)$ such that $\neg l_1, \dots, \neg l_k$ label nodes in G,
   (a) Add a node labeled l if not already present in G.
   (b) Add edges $(l_i, l), 1 \leq i \leq k$, if not already present.
   (c) Add C to the label set of these edges. These edges are thought of as grouped together and associated with clause C.
3. Add to G a special node Λ. For any variable x which occurs both positively and negatively in G, add directed edges from x and $\neg x$ to Λ.

Since all node labels in G are distinct, we identify nodes with the literals labeling them. Any variable x occurring both positively and negatively in G is a *conflict variable*, and x as well as $\neg x$ are *conflict literals*. G contains a *conflict* if it has at least one conflict variable. DPLL at a given stage has a *conflict* if the implication graph at that stage contains a conflict. A conflict can equivalently be thought of as occurring when the residual formula contains the empty clause Λ.

By definition, an implication graph may contain many conflict variables and several ways of deriving any single literal. To better understand and analyze a conflict, we work with a subgraph, called the *conflict graph* (see Figure 1), that captures only one among possibly many ways of reaching a conflict from the decision variables. The choice of the conflict graph is part of the strategy of the solver. It can also be thought of as giving power to clause learning by adding nondeterminism.

Definition 5. A *conflict graph* H is any subgraph of an implication graph with the following properties:

1. H contains Λ and exactly one conflict variable.
2. All nodes in H have a path to Λ.
3. Every node l in H other than Λ either corresponds to a decision literal or has precisely the nodes $\neg l_1, \neg l_2, \dots, \neg l_k$ as predecessors where $(l_1 \vee l_2 \vee \dots \vee l_k \vee l)$ is a known clause.

Consider the implication graph at a stage where there is a conflict and fix a conflict graph contained in that implication graph. Pick any cut in the conflict graph that has all decision variables on one side, called the *reason side*, and Λ as well as at least one conflict literal on the other side, called the *conflict side*. All nodes on the reason side that have at least one edge going to the conflict side form a *cause* of the conflict. The negations of the corresponding literals forms the *conflict clause associated with this cut*.

Proposition 1. *Every conflict clause corresponds to a cut in a conflict graph that separates decision variables from Λ and a conflict literal.*

Proof. Let S denote the set containing the negations of the literals of a given conflict clause C and $pred(S)$ be the set of all predecessors of these literals in the underlying implication graph. Let T denote the set containing all literals obtained by unit propagation after setting literals in S to TRUE. Since C is a conflict clause, T must contain a conflict literal. Consider the subgraph $G_{S,T}$ of the implication graph induced by Λ and the literals in $S \cup pred(S) \cup T$, but having no edges going from $pred(S)$ to T. Fix any conflict graph that is a subgraph of $G_{S,T}$. The cut in this conflict graph with T as the conflict side has C as the conflict clause. $\square$

Proposition 2. *If there is a trivial resolution derivation of a clause C from a set of clauses F, then setting all literals of C to* FALSE *leads to a conflict.*

Proof. Let $\pi = (C_1, C_2, \ldots, C_k \equiv C)$ be a trivial resolution derivation of C from F. Assume without loss of generality that clauses in π are ordered so that all initial clauses precede any derived clause. We give a proof by induction on the number of derived clauses in π.

For the base case, π does not have any derived clauses. Consequently, $C_k \in F$. Let $C_k = (l_1 \vee l_2 \vee \ldots \vee l_q)$ and ρ be the partial assignment that sets all $l_i, 1 \leq i \leq q$, to FALSE. Unit propagation using clause C_k with $l_i, 1 \leq i \leq q-1$, set to FALSE derives the literal l_q. Since $\neg l_q$ is a decision literal of ρ, l_q serves as a conflict variable and we have a conflict.

When π does have derived clauses, C_k, by triviality of π, must be derived by resolving C_{k-1} with a clause in F. Assume without loss of generality that $C_{k-1} \equiv (A \vee x)$ and the clause from F used in this resolution step is $(B \vee \neg x)$, where both A and B are subclauses of C_k. ρ falsifies all literals of B, implying $x =$ FALSE by unit propagation. This in turn results in falsifying all literals of C_{k-1} because all literals of A are also set to FALSE by ρ. Now $(C_1, \ldots, C_{k-1})$ is a trivial resolution derivation of C_{k-1} from F with one less derived clause than π, and all literals of C_{k-1} are falsified. By induction, this must lead to a conflict. $\square$

Proposition 3. *Any conflict clause can be derived from known clauses using a trivial resolution derivation.*

Proof. In the light of Proposition 1, assume that we have a conflict clause associated with a cut σ in a fixed conflict graph. Let $V_{conflict}(\sigma)$ denote the set of variables on the conflict side of σ, but including the conflict variable only if it occurs both positively and negatively on the conflict side. We will prove by induction on $|V_{conflict}(\sigma)|$ the stronger statement that the conflict clause associated with a cut σ has a trivial derivation resolving precisely on the variables in $V_{conflict}(\sigma)$.

For the base case, $V_{conflict}(\sigma) = \phi$ and the conflict side contains only Λ and a conflict literal, say x. The cause associated with this cut consists of node $\neg x$ that has an edge to Λ, and nodes $\neg l_1, \neg l_2, \ldots, \neg l_k$ corresponding to a known clause $C_x = (l_1 \vee l_2 \vee \ldots \vee l_k \vee x)$ that each have an edge to x. The conflict clause for this cut is simply the known clause C_x itself, having a length zero trivial derivation.

When $V_{conflict}(\sigma) \neq \phi$, pick a node y on the conflict side all whose predecessors are on the reason side. Let the conflict clause be $C = (l_1 \vee l_2 \vee \ldots \vee l_p)$ and assume without loss of generality that the predecessors of y are $\neg l_1, \neg l_2, \ldots, \neg l_k$ for some $k \leq p$. By definition of unit propagation, $C_y = (l_1 \vee l_2 \vee \ldots \vee l_k \vee y)$ must be a known clause. Obtain a new cut σ' from σ by pulling node y to the reason side. The new associated conflict clause must be of the form $C' = (\neg y \vee D)$, where D is a subclause of C. Now $V_{conflict}(\sigma') \subset V_{conflict}(\sigma)$. Consequently, by induction, C' must have a trivial resolution derivation from known clauses resolving precisely upon the variables in $V_{conflict}(\sigma')$. Recall that no variable occurs twice in a conflict graph except the conflict variable. Hence $V_{conflict}(\sigma')$ has exactly all variables of $V_{conflict}(\sigma)$ except y. Using this trivial derivation of C' and finally resolving C' with the known clause C_y on variable y gives us a trivial derivation of C from known clauses. This completes the inductive step. $\square$

3.2 Different Learning Schemes

Different cuts separating decision variables from Λ and a conflict literal correspond to different learning schemes (see Figure 1. One can also create learning schemes based on cuts not involving conflict literals at all [Zhang *et al.*, 2001], but their effectiveness is not clear. These will not be considered here.

It is insightful to think of the *non-deterministic* scheme as the most general learning scheme. Here we pick the cut non-deterministically, choosing, whenever possible, one whose associated clause is not already known. Since we can repeatedly branch on the same last variable, non-deterministic learning subsumes learning multiple clauses from a single conflict as long as the sets of nodes on the reason side of the corresponding cuts form a (set-wise) decreasing sequence. For simplicity, we will assume that only one clause is learned from any conflict.

In practice, however, we employ deterministic schemes. The *decision* scheme [Zhang *et al.*, 2001], for example, uses the cut whose reason side comprises all decision variables. `rel-sat` [Bayardo Jr. and Schrag, 1997] uses the cut whose conflict side consists of all implied variables at the current decision level. This scheme allows the conflict clause to have exactly one variable from the current decision level, causing an automatic flip in its assignment upon backtracking.

This nice flipping property holds in general for all *unique implication points* (UIPs) [Marques-Silva and Sakallah, 1996]. A UIP of an implication graph is a node at the current decision level d such that any path from the decision variable at level d to the conflict variable as well as its negation must go through it. Intuitively, it is a *single* reason at level d that causes the conflict. Whereas `rel-sat` uses the decision variable as the obvious UIP, `GRASP` [Marques-Silva and Sakallah, 1996] and `zChaff` [Moskewicz *et al.*, 2001] use *FirstUIP*, the one that is "closest" to the conflict variable. `GRASP` also learns multiple clauses when faced with a conflict. This makes it typically require fewer branching steps but possibly slower because of the time lost in learning and unit propagation.

The concept of UIP can be generalized to decision levels other than the current one. The *1UIP scheme* corresponds to learning the FirstUIP clause of the current decision level, the *2UIP scheme* to learning the FirstUIP clauses of both the current level and the one before, and so on. Zhang *et al* [2001] present a comparison of all these and other learning schemes and conclude that 1UIP is quite robust and outperforms all other schemes they consider on most of the benchmarks.

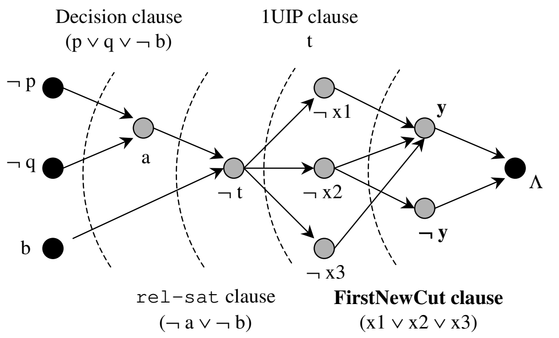

Figure 1: A conflict graph depicting various learning schemes

The FirstNewCut Scheme

We propose a new learning scheme called *FirstNewCut* whose ease of analysis helps us demonstrate the power of clause learning. We would like to point out that we use this scheme here only to prove our theoretical bounds. Its effectiveness on other formulas has not been studied yet.

The key idea behind FirstNewCut is to make the conflict clause as relevant to the current conflict as possible by choosing a cut close to the conflict literals. This is what the FirstUIP scheme also tries to achieve in a slightly different manner. For the following definitions, fix a cut in a conflict graph and let S be the set of nodes on the reason side that have an edge to some node on the conflict side (S is the reason side *frontier* of the cut). Let C_S be the conflict clause associated with this cut.

Definition 6. *Minimization* of conflict clause C_S is the process of repeatedly identifying, if one exists, a node $v \in S$, all of whose predecessors are also in S, moving it to the conflict side, and removing it from S.

Definition 7. *FirstNewCut scheme*: Start with a cut whose conflict side consists of Λ and a conflict literal. If necessary, repeat the following until the associated conflict clause, after minimization, is not already known: pick a node on the conflict side, pull all its predecessors except those that correspond to decision variables into the conflict side. Finally, learn the resulting new minimized conflict clause.

This scheme starts with the cut that is closest to the conflict literals and iteratively moves it back toward the decision variables until a new associated conflict clause is found. This backward search always halts because the cut with all decision variables on the reason side is certainly a new cut. Note that there are potentially several ways of choosing a literal to move the cut back, leading to different conflict clauses. The FirstNewCut scheme, by definition, always learns a clause not already known. This motivates the following:

Definition 8. A clause learning scheme is *non-redundant* if on a conflict, it always learns a clause not already known.

3.3 Fast Backtracking and Restarts

Most clause learning algorithms use *fast backtracking* where one uses the conflict graph to backtrack not only the last branching decision but also all other recent decisions that did not contribute to the conflict [Stallman and Sussman, 1977]. This adds power to clause learning because the current conflict might use clauses learned earlier as a result of branching on the apparently redundant variables. Hence, fast backtracking in general cannot be replaced by a "good" branching sequence that does not produce redundant branches. For the same reason, fast backtracking cannot either be replaced by simply learning the decision scheme clause. However, the results we present in this paper are independent of whether or not fast backtracking is used.

Restarts allow clause learning algorithms to arbitrarily restart their branching process. All clauses learned so far are however retained and now treated as additional initial clauses [Baptista and Silva, 2000]. As we will show, unlimited restarts can make clause learning very powerful at the cost of adding non-determinism. Unless otherwise stated, clause learning proofs will be assumed to allow no restarts.

4 Clause Learning and General Resolution

Lemma 1. *Let F be a CNF formula over n variables. If F has a general resolution proof of size s, then it also has a clause learning proof of size at most ns using any non-redundant learning scheme and at most s restarts.*

Proof. Let $\pi = (C_1, C_2, \ldots, C_s \equiv \Lambda)$ be a general resolution proof of F where each C_i is either an initial clause or derived by resolving two clauses C_j and C_k, $j, k < i$, occurring earlier in π. If π contains a derived clause C_i whose strict subclause C'_i can be derived by resolving two previously occurring clauses, then we can replace C_i with C'_i, do trivial simplifications on further derivations that used C_i and obtain another proof π' of F of size at most s. Doing this repeatedly will remove all such redundant clauses and leave us with a simplified proof no larger in size. Hence we will assume without loss of generality that π has no such clause.

A clause learning proof of F can be constructed by choosing derived clauses of π in order, learning each of them, and restarting. Suppose every clause C_i, $i < p$, is already known and we are at decision level zero. This is trivially true when $C_1, \ldots, C_{p-1}$ are initial clauses. If $p = s$, there are two known clauses x and $\neg x$ whose resolution generates $C_s \equiv \Lambda$. In this case we have a conflict from the known clauses at decision level zero and our clause learning proof is complete.

Otherwise, let $C_p = (l_1 \vee l_2 \vee \ldots \vee l_k)$ and assume without loss of generality that C_p is derived by resolving two known clauses $(A \vee x)$ and $(B \vee \neg x)$, where both A and B are subclauses of C_p. Our clause learning proof will choose to branch on and set all of $l_1, l_2, \ldots, l_k$ to FALSE. This will falsify both A and B and thus imply both x and $\neg x$ after unit propagation, resulting in a conflict. It is easy to see that the only new clause that can be learned from this conflict is C_p. Accordingly, we will use any non-redundant learning scheme, learn C_p and restart to get back to decision level zero. Due to the non-redundancy of C_p assumed earlier in the proof, there is no premature conflict until *all* of $l_1, l_2, \ldots, l_k$ have been branched upon. This makes sure that our clause learning proof proceeds as described above and we learn precisely the clause C_p.

We learn at most s clauses and each learning stage requires branching on at most n variables and exactly one restart. This gives the desired bounds on the size of the constructed clause learning proof and the number of restarts it uses. □

Lemma 2. *Let F be a CNF formula over n variables. If F has a clause learning proof of size s using any learning scheme and any number of restarts, then F also has a general resolution proof of size at most ns.*

Proof. Given a clause learning proof π of F, a resolution proof can be constructed by sequentially deriving all clauses that π learns, which includes the empty clause Λ. From Proposition 3, all these derivations are trivial and hence require at most n steps each. Consequently, the size of the resulting clause learning proof is at most ns. Note that since we derive clauses of π individually, restarts in π do not change the construction. □

Combining Lemmas 1 and 2, we immediately get

Theorem 1. *Clause learning with any non-redundant scheme and unlimited restarts is equivalent to general resolution.*

Note that this theorem strengthens the result from [Baptista and Silva, 2000] that clause learning together with restarts is *complete*. Our theorem makes the stronger claim that clause learning with restarts can find proofs that are as short as those of general resolution.

5 Clause Learning and Regular Resolution

Here we prove that clause learning even without restarts is exponentially stronger than regular resolution on some formulas. We do this by first introducing a way of extending any CNF formula based on a given resolution proof of it. We then show that if a formula exponentially separates general resolution from regular resolution, its extension exponentially separates clause learning from regular resolution. Finally, we cite specific formulas called GT'_n that satisfy this property.

5.1 The Proof Trace Extension

Definition 9. Let F be a CNF formula and π be a resolution refutation of it whose last step resolves v with $\neg v$. Let $S = \pi \setminus (F \cup \{\neg v, \Lambda\})$. The *proof trace extension* $PT(F, \pi)$ of F is a CNF formula over variables of F and new trace variables t_C for clauses $C \in S$. The clauses of $PT(F, \pi)$ are all initial clauses of F together with a trace clause $(\neg x \vee t_C)$ for each clause $C \in S$ and each literal $x \in C$.

We first show that if a formula has a short general resolution refutation, then the corresponding proof trace extension has a short clause learning proof. Intuitively, the new trace variables allow us to handle every resolution step of the original proof individually, effectively letting us *restart* after learning each derived clause.

Lemma 3. *Let F be any CNF formula and π be a resolution refutation of it. Then $PT(F, \pi)$ has a clause learning proof of size less than $size(\pi)$ using the FirstNewCut scheme and no restarts.*

Proof. Let v be the variable resolved upon in the last step of the resolution proof π and $S = \pi \setminus (F \cup \{\neg v, \Lambda\})$. Let $(C_1, C_2, \ldots, C_k \equiv v)$ be the subsequence of π that has precisely the clauses in S. Assume without loss of generality as in the proof of Lemma 1 that π does not contain a derived clause C_i whose strict subclause C'_i can be derived by resolving two previously occurring clauses. We claim that the branching sequence $(t_{C_1}, t_{C_2}, \ldots, t_{C_k})$ induces a clause learning proof of F using the FirstNewCut scheme.

Suppose $C_1 = (x_1 \vee x_2 \vee \ldots \vee x_l)$ and assume without loss of generality that it was derived by resolving two previous clauses $(A \vee y)$ and $(B \vee \neg y)$, where both A and B are subclauses of C_1. The first branch $t_{C_1} =$ FALSE followed by unit propagation results in implied literals $\neg x_1, \neg x_2, \ldots, \neg x_l$ using trace clauses $(\neg x_i \vee t_{C_1})$. Further unit propagation using A and B implies y as well as $\neg y$ and we have a conflict. The cut in the conflict graph containing y and $\neg y$ on the conflict side and everything else on the reason side makes us learn C_1 as the FirstNewCut conflict clause. We now backtrack the branch on t_{C_1} and continue. Subsequent branches on $t_{C_2}, t_{C_3}, \ldots, t_{C_k}$ similarly make us learn clauses $C_2, C_3, \ldots, C_k$.

We have now learned (or have as initial clauses) the clause v as well as the two clauses used to derive $\neg v$ in π. These generate Λ in the residual formula, leading to a conflict without any branches and finishing the clause learning proof. Note that the maximum decision level in this proof is 1. □

Lemma 4. *Let F be a CNF formula over n variables that has a polynomial (in n) size general resolution proof π but requires exponential size regular resolution proofs. Then $PT(F, \pi)$ has a polynomial size clause learning proof using the FirstNewCut scheme and no restarts, but requires exponential size regular resolution proofs.*

Proof. Lemma 3 immediately implies that $PT(F, \pi)$ has a polynomial size FirstNewCut clause learning proof.

For the other part, we use a simple reduction argument. Suppose $PT(F, \pi)$ has a regular resolution refutation π' of size s. Consider the restriction ρ that sets every trace variable of this formula to TRUE. ρ keeps original clauses of F intact and trivially satisfies all trace clauses, thereby reducing the initial clauses of $PT(F, \pi)$ to precisely F. Recall that regularity of resolution proofs is preserved under arbitrary restrictions. Consequently, applying ρ to π' gives us a regular resolution refutation of F of size at most s. By the assumption in the Lemma, s must be exponential in n. Note that $PT(F, \pi)$ itself is of size polynomial in n because of our choice of π. Hence s is also exponential in the size of $PT(F, \pi)$ itself. □

5.2 The GT'_n Formulas

We use the proof trace extension of an explicit family of unsatisfiable CNF formulas called GT'_n to obtain an exponential separation between regular resolution and clause learning. Note that in place of GT'_n, we could also have used any other formulas satisfying the conditions on F in Lemma 4, such as the modified pebbling formulas of Alekhnovich *et al* [2002].

The GT'_n formulas are based on the ordering principle that any partial linear ordering on the set $\{1, 2, \ldots, n\}$ must have a maximal element. The original formulas, called GT_n, were first considered by Krishnamurthy [1985] and later used by Bonet and Galesi [1999] to show the optimality of the size-width relationship of resolution proofs. Recently, Alekhnovich *et al* [2002] used GT'_n to show an exponential separation between general and regular resolution. We refer the reader to this paper for exact specification of the GT'_n formulas. For our bound, we only need the following result:

Lemma 5 ([Alekhnovich *et al.*, 2002]). *GT'_n has a polynomial size general resolution refutation but requires exponential size regular resolution proofs.*

Let $\pi_{GT'}$ be the polynomial size resolution refutation of GT'_n described in [Alekhnovich *et al.*, 2002]. It follows from Lemmas 4 and 5 that $PT(GT'_n, \pi_{GT'})$ has a polynomial size clause learning proof using the FirstNewCut scheme but requires exponential size regular resolution proofs. Hence,

Theorem 2. *There exist CNF formulas on which clause learning using the FirstNewCut scheme and no restarts provides exponentially smaller proofs than regular resolution.*

6 Experimental Results

Table 1 reports the performance of variants of `zChaff` on *grid pebbling formulas*. We conducted experiments on a 1600 MHz Linux machine with memory limit set to 512MB. Base code of `zChaff` was extended to allow a (partial) branching sequence to be specified as part of the input. We used the more popular 1UIP learning scheme of `zChaff` instead of FirstNewCut because for these formulas, both schemes provide small proofs.

Pebbling formulas based on pebbling graphs are known to be hard for tree-like resolution but easy for regular resolution [Ben-Sasson and Wigderson, 1999]. We refer the reader to this paper for exact specification of the formulas. For our experiments, we worked with a uniform, grid like version, where every non-leaf node has precisely two predecessors. For a k layer graph, the corresponding formula has $O(k^2)$ variables and clauses. These formulas are minimally unsatisfiable. We also used a satisfiable version obtained by deleting a randomly chosen clause. The branching sequence was generated based on depth first traversal of the underlying pebbling graph. Results are reported for `zChaff` with no learning or specified branching sequence (DPLL), with specified branching sequence only, with clause learning only (original `zChaff`), and both.

	Formula		*Runtime* (seconds)	
Solver	*layers*	*variables*	*unsat.*	*satisfiable*
	5	30	0.24	0.12
DPLL	6	42	110	0.02
	7	56	> 24 hrs	0.07
	8	72	> 24 hrs	> 24 hrs
Branch	5	30	0.20	0.00
sequence	6	42	105	0.00
only	7	56	> 24 hrs	0.00
	9	90	> 24 hrs	> 24 hrs
Clause	20	420	0.12	0.05
learning	40	1,640	59	36
only	60	3,660	65	14
(original	65	4,290	‡	47
`zChaff`)	70	4,970	‡	‡
Clause	40	1,640	0.04	0.04
learning	100	10,100	0.59	0.62
and	500	250,500	254	288
branch	1,000	1,001,000	4,251	5,335
sequence	1,500	2,551,500	21,103	‡

Table 1: `zChaff` on pebbling formulas. ‡ denotes out of memory

7 Discussion and Open Problems

This paper has begun the task of formally understanding the power of clause learning from a proof complexity perspective. We have seen that clause learning can be more powerful than even regular resolution, and that learning with restarts yields general resolution.

In practice, a solver must employ good branching heuristics as well as implement a powerful proof system. Our result that modified pebbling formulas have small clause learning proofs depends critically upon the solver choosing a branching sequence that solves the formula in “bottom up” fashion, so that the learned clauses have maximal reuse. As we describe in a subsequent paper [Sabharwal *et al.*, 2003], this branching sequence can be efficiently generated by a combination of breadth-first and depth-first traversals of the original pebbling graph even for more general classes of pebbling formulas. As shown in Table 1, one needs both clause learning as well as a good branching sequence to efficiently solve large problem instances.

Of course, pebbling graphs, which correspond to problems involving precedence of tasks, represent a narrow domain of applicability. However, logic encodings of many kinds of real-world problems, such as planning graphs [Kautz and Selman, 1996], exhibit layered structure not unlike pebbling graphs. An important direction of our current research is to generate branching sequences that allow clause learning to work well ofr the general classes of structures that arise in encodings of particular problem domains.

Different learning schemes are likely to vary in their effectiveness when used on formulas from different domains. We introduced FirstNewCut as a new scheme and used it in this paper to derive our theoretical results. How well it performs on encodings of real-world problems is still open. It would be interesting to know if there is a class of practical problems on which FirstNewCut works better than other schemes.

This paper inspires but leaves open several interesting questions of proof complexity. We have shown that with arbitrary restarts, clause learning is as powerful as general resolution. However, judging when to restart and deciding what branching sequence to use after restarting adds more non-determinism to the process, making it harder for practical implementations. Can clause learning with no or limited restarts also simulate general resolution efficiently? We showed that there are formulas on which clause learning is much more efficient than regular resolution. In general, can every small regular refutation be converted into a small clause learning proof? Or are regular resolution and clause learning incomparable?

References

[Achlioptas *et al.*, 2001] D. Achlioptas, P. Beame, and M. Molloy. A sharp threshold in proof complexity. In *33rd STOC*, 337–346, 2001.

[Alekhnovich *et al.*, 2002] M. Alekhnovich, J. Johannsen, T. Pitassi, and A. Urquhart. An exponential separation between regular and general resolution. In *34th STOC*, 448–456, 2002.

[Baptista and Silva, 2000] L. Baptista and J. P. M. Silva. Using randomization and learning to solve hard real-world instances of satisfiability. In *Prin. and Prac. of Const. Prog.*, 489–494, 2000.

[Bayardo Jr. and Schrag, 1997] R. J. Bayardo Jr. and R. C. Schrag. Using CST look-back techniques to solve real-world SAT instances. In *14th AAAI*, 203–208, 1997.

[Ben-Sasson and Wigderson, 1999] E. Ben-Sasson and A. Wigderson. Short proofs are narrow – resolution made simple. In *31st STOC*, 517–526, 1999.

[Ben-Sasson *et al.*, 2000] E. Ben-Sasson, R. Impagliazzo, and A. Wigderson. Near-optimal separation of treelike and general resolution. Tech. Rep. TR00-005, Elec. Colloq. in Comput. Compl., http://www.eccc.uni-trier.de/eccc/, 2000.

[Biere *et al.*, 1999] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *36th DAC*, 317–320, 1999.

[Bonet and Galesi, 1999] M. L. Bonet and N. Galesi. A study of proof search algorithms for resolution and polynomial calculus. In *40th FOCS*, 422–432, 1999.

[Bonet *et al.*, 2000] M. L. Bonet, J. L. Esteban, N. Galesi, and J. Johansen, On the relative complexity of resolution refinements and cutting planes proof systems, SIAM J. Comput., 30 (2000), 1462–1484.

[Cook and Reckhow, 1977] S. A. Cook and R. A. Reckhow, The relative efficiency of propositional proof systems, J. Symb. Logic, 44 (1977), 36–50.

[Davis and Putnam, 1960] M. Davis and H. Putnam, A computing procedure for quantification theory, CACM, 7 (1960), 201–215.

[Davis *et al.*, 1962] M. Davis, G. Logemann, and D. Loveland, A machine program for theorem proving, CACM, 5 (1962), 394–397.

[Davis, 1984] R. Davis, Diagnostic reasoning based on structure and behavior, J. AI, 24 (1984), 347–410.

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

[Genesereth, 1984] R. Genesereth, The use of design descriptions in automated diagnosis, J. AI, 24 (1984), 411–436.

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

[Gomes *et al.*, 1998b] C. P. Gomes, B. Selman, K. McAloon, and C. Tretkoff. Randomization in backtrack search: Exploiting heavy-tailed profiles for solving hard scheduling problems. In *4th Int. Conf. Art. Intel. Planning Syst.*, 1998.

[Kautz and Selman, 1996] H. A. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *13th AAAI*, 1194–1201, 1996.

[Konuk and Larrabee, 1993] H. Konuk and T. Larrabee. Explorations of sequential ATPG using boolean satisfiability. In *11th VLSI Test Symposium*, 85–90, 1993.

[Krishnamurthy, 1985] B. Krishnamurthy, Short proofs for tricky formulas, Acta Inf., 22 (1985), 253–274.

[Li and Anbulagan, 1997] C. M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *IJCAI (1)*, 366–371, 1997.

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

[Marques-Silva, 1998] J. Marques-Silva. An overview of backtrack search satisfiability algorithms. In *5th Symp. on AI and Math.*, 1998.

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

[Sabharwal *et al.*, 2003] A. Sabharwal, P. Beame, and H. Kautz. Using problem structure for efficient clause learning. To appear in SAT, 2003.

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

[Velev and Bryant, 2001] M. Velev and R. Bryant. Effective use of boolean satisfiability procedures in the formal verification of superscalar and vliw microprocessors. In *38th DAC*, 226–231, 2001.

[Zhang and Hsiang, 1994] H. Zhang and J. Hsiang. Solving open quasigroup problems by propositional reasoning. In *Proceedings of the International Computer Symp.*, 1994.

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

[Zhang, 1997] H. Zhang. SATO: An efficient propositional prover. In *Int. Conf. on Auto. Deduc., LNAI*, vol. 1249, 272–275, 1997.