Extended Clause Learning

Jinbo Huang
NICTA and Australian National University
Canberra, Australia
jinbo.huang@nicta.com.au

Abstract

The past decade has seen clause learning as the most successful algorithm for SAT instances arising from real-world applications. This practical success is accompanied by theoretical results showing clause learning as equivalent in power to resolution. There exist, however, problems that are intractable for resolution, for which clause-learning solvers are hence doomed. In this paper, we present *extended clause learning*, a practical SAT algorithm that surpasses resolution in power. Indeed, we prove that it is equivalent in power to *extended resolution*, a proof system strictly more powerful than resolution. Empirical results based on an initial implementation suggest that the additional theoretical power can indeed translate into substantial practical gains.

1. Introduction

Clause learning in modern SAT solvers can be described as a loop where each iteration consists of a sequence of variable assignments followed by the learning of a clause, except for satisfiable instances where the last iteration produces a model [1, 2]. It is well understood that the derivation of each clause is in fact a sequence of resolutions, of which the learned clause is the final resolvent, and hence clause learning as a proof system cannot surpass resolution in power.

A result in the other direction, that clause learning is no less powerful than resolution, has also been gradually established in a sequence of papers [3, 4, 5, 6], the latest of which proves it without the artificial modifications of the algorithm required in earlier results. (The proof in [6] utilizes a restart after every conflict, and leaves open the question whether less frequent restarts would theoretically limit the power of clause learning.)

On the one hand, this theoretical equivalence between the two systems is positive news, as it means that the shortest resolution proofs (up to a polynomial factor) are always attainable by clause-learning SAT solvers as long as suitable choices are made in all components of the algorithm, mainly: decision, learning, backtracking (including restarts), clause deletion (clause deletion is not required for the theoretical equivalence to hold, but can help in practice). Practical solvers naturally rely on heuristics to approximate this goal, and most advances in the area boil down to new heuristics for these components, to name just one recent example for each: progress saving [7], bi-asserting clauses [8], new restart policies [9], clause deletion based on decision level span [10].

On the other hand, we are faced with the inevitable implication that clause-learning solvers, however perfect their heuristics, are doomed for problems intractable for resolution. Such problems indeed exist, the *pigeonhole principle* being one often cited example [11].

To overcome the limitations of resolution, this paper presents a new algorithm called *extended clause learning*. While clause learning introduces new clauses into the CNF, our proposed extension provides also for the introduction of *new variables* into the CNF. We prove that this new algorithm is equivalent in power, in the sense of p-simulation, to *extended resolution* [12], a proof system strictly more powerful than resolution.

To evaluate the potential practical benefits of this new algorithm, we also describe a concrete heuristic for making extensions. Although our implementation of this heuristic has only resulted in higher performance on a subset of the benchmarks used, we show that the improvement there is very significant. This provides evidence that the theoretical power of extended clause learning can indeed materialize in practice where the extension heuristic is effective, motivating a continued investigation in this new direction of research.

The remainder of the paper is organized as follows. We review clause learning as a resolution proof system, present extended clause learning, prove its equivalence to extended resolution, present and discuss empirical results, discuss some related work, and conclude the paper.

2. Clause Learning as Resolution

Given two clauses $\alpha \vee x$ and $\beta \vee \neg x$, where x is a Boolean variable and α and β are clauses, the *resolution* rule allows the derivation of the clause $\alpha \vee \beta$, called the *resolvent* of the two original clauses. Given a CNF formula represented as a set of clauses Δ, a *resolution proof* of a clause C_k is a sequence of clauses

$C_1, C_2, \ldots, C_k$ such that each clause is either a member of Δ or the resolvent of two earlier clauses in the sequence. If C_k is the empty clause, then the proof establishes unsatisfiability of Δ, and is called a *resolution refutation* of Δ.

Clause learning is an algorithm capable of establishing both satisfiability and unsatisfiability of CNF formulas, although only the latter aspect is relevant in our discussion of proof systems. Today's clause-learning SAT solvers generally implement the 3-step loop given below in Algorithm 1 [9], where we omit clause deletion as it is not relevant to our present discussion.

Algorithm 1 Clause Learning

1: set variables till a conflict is hit; return SAT if all variables are set without conflict
2: derive a new clause by resolution and add it to the CNF; return UNSAT if empty clause is derived
3: unset some variables; go back to Step 1

A number of important details about this algorithm are discussed below:

- Step 1 involves both *decisions*, where the solver selects a variable and sets it to either true or false, and *implications*, where the value of a variable is forced via unit propagation.

- Step 2 adds a derived clause to the CNF on reaching a conflict. Recall that a conflict is a clause of the CNF whose literals have all been set to false (in Step 1). The derivation starts with this clause, and consists in resolving it with clauses, one at a time, that were responsible for forcing the values of its literals in unit propagation. At some point this process is terminated and the final resolvent returned as the learned clause.

- Step 3 undoes some variable assignments, usually in reverse chronological order (i.e., latest assignment undone first) so that a prefix of the assignment sequence carries over to the next iteration. This is the step commonly known as *backtracking*, with a *restart* being a special case (where the prefix contains only assignments forced by the current unit clauses).

It is easy to see that by keeping track of the derivations in Step 2, a resolution refutation can be produced, with little overhead, when unsatisfiability is returned. This implies that clause learning cannot be more powerful than resolution. In particular, if a polynomial-sized resolution refutation does not exist for the given CNF, then Algorithm 1 will not terminate in polynomial time.

For the other direction, Pipatsrisawat and Darwiche [6] proved that Algorithm 1 in fact p-simulates resolution (i.e., can induce a resolution refutation where the size of the refutation and the time to produce it are both polynomial in the size of any given resolution refutation of the same CNF) and that the result holds even if the solver is restricted to learn only one particular type of clauses, known as *asserting* clauses (which is usually the case in practice).

Having thus reviewed the equivalence between clause learning and resolution, we now proceed to present the core contribution of this paper, an extension to clause learning that results in a strictly more powerful proof system.

3. An Extension

Our motivation for extending clause learning stemmed from a proof system that extends resolution, described next.

3.1. Extended Resolution (ER)

Tseitin [12] introduced the *extension* rule that allows new variables to be introduced in a resolution refutation along with their "definitions." Specifically, given CNF Δ and a variable x not appearing in Δ, the following clauses (the overline denotes literal complement: $\overline{x} = \neg x, \overline{\neg x} = x$, for variable x),

$$x \vee \overline{l_1}, x \vee \overline{l_2}, \neg x \vee l_1 \vee l_2,$$

or more compactly,

$$x \leftrightarrow l_1 \vee l_2,$$

can be added to the CNF for any literals l_1 and l_2 whose variables appear in Δ (the definition of extension in [12] involves negative literals and that in [13] uses conjunction instead of disjunction; they are equivalent to each other while ours is equivalent to both).

The resulting proof system is known as *extended resolution* (ER), where a refutation is a sequence of extensions followed by a normal resolution refutation.

That ER is at least as powerful as resolution is straightforward, as one can always ignore the extension rule. Cook [14] gave a polynomial-sized ER proof of the pigeonhole principle (in a standard Boolean encoding), a problem for which it was subsequently shown that no polynomial-sized resolution proof exists [11]. This establishes that ER is strictly more powerful than resolution.

3.2. *Extended Clause Learning (ECL)*

If one were to simply add the same extension rule to clause learning, so that the CNF could be suitably augmented before Algorithm 1 started, one would indeed immediately obtain an algorithm equivalent to ER in power. Such an algorithm, however, would offer no help in selecting suitable literals for each extension, just as the resolution rule itself, for example, offers no help in selecting suitable clauses to resolve.

Hence we would like a mechanism to guide the execution of each extension (while not preventing short proofs to be found), in much the same way as clause learning serves as a mechanism to guide the sequence of resolutions. It turns out, as our empirical results will help illustrate, that the decision heuristic can provide an effective candidate for this task.

First of all, we will allow extensions involving two or more literals, i.e., $x \leftrightarrow \alpha$ where x is a fresh variable and α is any clause of size ≥ 2. We envisage that this will enhance flexibility in practical solvers, although it does not affect the power of the proof system (as a "larger" extension can always be replaced by a few two-literal extensions and resolutions).

Second, to make an extension $x \leftrightarrow \alpha$, we will only look for an α whose negation is a subset of the solver's assignment stack when a conflict is reached (hence the selection of α will be partly guided by the decision heuristic). This implies that we will interleave extensions and learning (i.e., resolutions) in the actual execution of the algorithm. The resulting proofs, however, remain valid ER proofs as they can always be rearranged so that all extensions appear before resolutions.

More specifically, whenever a conflict is reached after a set (sequence) of assignments (literals) $A = \{l_1, \ldots, l_k\}, k > 2$, have been made, we may decide to make an extension. If we do, we select a subset $\{l_{i_1}, \ldots, l_{i_r}\}$ of A, and make the extension $x \leftrightarrow \overline{l_{i_1}} \vee \ldots \vee \overline{l_{i_r}}$, where x is a fresh variable.

At this point readers familiar with SAT solver implementations may ask how the attributes (value, decision level, antecedent, etc) of the new variable x should be set. Our answer is: don't bother—simply have the solver perform a restart after each extension.

We shall refer to the resulting algorithm as *extended clause learning* (ECL), summarized in Algorithm 2, where lines 4–6 carry out the extensions.

3.3. *A Concrete Heuristic for Extensions*

For an initial empirical study of ECL, we propose a concrete heuristic for extensions that has a potentially desirable side effect—that of shortening learned

Algorithm 2 Extended Clause Learning

1: set variables till a conflict is hit; return SAT if all variables are set without conflict
2: derive a new clause by resolution; return UNSAT if empty clause is derived
3: **if** an extension is desired **then**
4: select $\{l_{i_1}, \ldots, l_{i_r}\}, r \geq 2$, from assignment stack
5: add clauses $x \leftrightarrow \overline{l_{i_1}} \vee \ldots \vee \overline{l_{i_r}}$ for new variable x
6: unset all variables except those forced by unit propagation; go back to Step 1
7: **else**
8: add the clause derived on line 2 to the CNF
9: **end if**
10: unset some variables; go back to Step 1

clauses. Specifically, when we decide to make an extension (after a conflict) we will use literals that appear in the learned clause (note that a learned clause can only contain literals whose negations are on the assignment stack). In other words, we will split the learned clause γ (of size > 2) into $\alpha \vee \beta$ such that α has size ≥ 2 and β is not empty, and learn the clauses $x \vee \beta, x \leftrightarrow \alpha$ instead of γ, where x is a fresh variable. This has the effect of combining an extension and the learning of a clause into a single step, and shortening the learned clause by $|\alpha| - 1$.

In the experiments to be reported later in the paper, we made an extension whenever the learned clause had size > 30, and selected among the learned clause the two earliest-set literals as α.

4. Equivalence of ECL and ER

It is not always the case that a clause-learning solver (Algorithm 1) can make the required assignments $\{\overline{l_1}, \overline{l_2}\}$ to allow any given extension $x \leftrightarrow l_1 \vee l_2$ to be made in the ECL framework, because there will generally be forced assignments via unit propagation. However, we will show that this becomes the case once all *useless* extensions have been removed from a given ER refutation.

Intuitively, a useless extension is one where the truth of the new variable would be implied by refutation using only unit propagation. The definition below captures this idea. For any set of clauses Δ, we will write $\Delta \vdash l$ (resp. $\Delta \vdash \mathsf{false}$) to mean that unit propagation on Δ produces literal l (resp. the empty clause).

Definition 1. *An extension $x \leftrightarrow l_1 \vee l_2$ on CNF Δ is* <u>*useless*</u> *if* $\Delta \cup \{\overline{l_1}, \overline{l_2}\} \vdash \mathsf{false}$.

We will show that useless extensions can indeed be dispensed with, where we will make use of the following lemma.

Lemma 1. *For CNF Δ and literals l_1, l_2, if $\Delta \cup \{\overline{l_1}, \overline{l_2}\} \vdash \mathsf{false}$, then one of the following four clauses can be derived from Δ by resolution in $O(n)$ steps where n is the number of variables of Δ: $l_1 \vee l_2$, l_1, l_2,* false.

Proof Without loss of generality, assume that Δ does not contain $\overline{l_1}$ or $\overline{l_2}$ (simply remove them if they do appear in Δ).

If $\mathsf{false} \in \Delta$, we are done; otherwise there exists a clause $\alpha_0 \in \Delta$ whose literals are all falsified under unit propagation on $\Delta \cup \{\overline{l_1}, \overline{l_2}\}$. Each literal l of α_0, unless $l = l_1$ or $l = l_2$, has been falsified because of an *antecedent* clause $\beta_0 \vee \bar{l}$ (where β_0 can be empty) in Δ that is or has become unit clause $\bar{l}$ under unit propagation. Resolving this antecedent with α_0 gives a new clause α_1 that satisfies the same property. Hence a sequence of clause $\alpha_0, \alpha_1, \ldots, \alpha_i$ can be produced by resolution until none of α_i's literals have an antecedent in Δ. Since l_1 and l_2 are the only two literals that do not have an antecedent in Δ, α_i must be one of those four clauses listed in the lemma.

Now note that in choosing a literal from any α_j for resolution, we can always select one whose value has been forced at the latest time in unit propagation (as is the practice in actual clause-learning SAT solvers). This will ensure that no variable is resolved twice in the resolution sequence (literals introduced into α_{j+1} from the antecedent after one resolution must have been forced at strictly earlier times and therefore no variable can be reintroduced into the clause once it has been resolved away). Hence the length of the sequence must be $O(n)$. $\square$

We are now ready to prove that disallowing useless extensions does not affect the power of ER.

Lemma 2. *For any ER refutation Π of CNF Δ, there exists an ER refutation of Δ without useless extensions and of length polynomial in $|\Pi|$.*

Proof Let $x \leftrightarrow l_1 \vee l_2$ be the last useless extension, X the set of all extensions before it, and Y the rest of the proof following it, in the given ER refutation. We have $\Delta \cup X \cup \{\overline{l_1}, \overline{l_2}\} \vdash \mathsf{false}$.

It follows from Lemma 1 that the clause $l_1 \vee l_2$, or one of its literals or the empty clause, can be derived from $\Delta \cup X$ by resolution in $O(n)$ steps, where n is the number of variables of $\Delta \cup X$. Without loss of generality, assume that $l_1 \vee l_2$ is derived (derivation of any subset of this clause is stronger and leads to a similar or simpler proof). Let Z be the resolution derivation of size $O(n)$ that ends in $l_1 \vee l_2$.

First consider the case where variable x does not appear in subsequent extensions. Since $l_1 \vee l_2$ is already derived, any clause $\beta \in Y$ as the resolvent of some

clause α and the clause $\neg x \vee l_1 \vee l_2$ (from the extension on x) can be safely replaced by β' as the resolvent of α and $l_1 \vee l_2$ (where the original resolution was on l_1 or l_2) or simply by $\beta' = l_1 \vee l_2$ (where the original resolution was on $\neg x$). In either case β' is a subset of β and does not contain $\neg x$. The substitution of β' for β can then be propagated in all resolutions that use β, and so on. Since the clause $\neg x \vee l_1 \vee l_2$ is the only source of the literal $\neg x$, once all possible substitutions have occurred, the resulting Y' does not contain any occurrences of $\neg x$. Clauses in Y' that contain the positive literal x can never contribute to the derivation of the empty clause (because the x would never be resolved away due to the total absence of $\neg x$ in Y'), and can therefore all be safely removed. The resulting Y'' does not mention variable x at all, and does not rely on clauses from $x \leftrightarrow l_1 \vee l_2$. Hence XZY'' (after pushing all extensions to the front) is an ER refutation of Δ not containing the useless extension $x \leftrightarrow l_1 \vee l_2$.

Now consider the other case where variable x appears in a subsequent extension either positively ($y \leftrightarrow x \vee l_3$) or negatively ($y \leftrightarrow \neg x \vee l_3$), which we will remove before removing x from the proof. The positive case is in fact not possible, as it would mean that the extension on y is also useless, contradicting the fact that $x \leftrightarrow l_1 \vee l_2$ is the last useless extension. It remains to show how to remove $y \leftrightarrow \neg x \vee l_3$. Recall that $l_1 \vee l_2$ is already derived. Resolving it with $x \leftrightarrow l_1 \vee l_2$ produces unit clause x, which together with $y \leftrightarrow \neg x \vee l_3$ would produce $y \leftrightarrow l_3$. It is then easy to see that we can simply remove $y \leftrightarrow \neg x \vee l_3$ from Y, and replace all occurrences of y with l_3, $\neg y \vee \neg x$ with $\overline{l_3}$, and the remaining $\neg y$ with $\overline{l_3}$, to obtain Y' such that $X \cup \{x \leftrightarrow l_1 \vee l_2\} \cup Y'$ continues to be an ER refutation of Δ. We then fall back on the case covered in the previous paragraph to remove x.

In both cases we have removed one useless extension from the proof while introducing no more than $O(n)$ (which is also $O(|\Pi|)$) extra resolutions into the proof. The lemma is proved by repeating the process until no useless extensions remain, with an increase of at most $O(|\Pi|^2)$ in the number of resolutions. $\square$

With this lemma, we can now prove that ECL and ER are indeed equivalent in power in the sense of p-simulation, given that clause learning and resolution have that same relation [6]. Recall that a proof system S p-simulates a proof system T if every refutation in T can be turned in polynomial time into a refutation of the same CNF in S. More formally, S p-simulates T if there is a function g, computable in polynomial time, such that for every refutation Π in T, $g(\Pi)$ is a refutation of the same CNF in S [15] (Pipatsrisawat and Darwiche [6] used a weaker notion of p-simulation, but gave an additional theorem so that the main result holds as if this definition of p-simulation is used).

We are now ready to state our main theorem:

Theorem 1. *ECL and ER p-simulate each other.*

Proof That ER p-simulates ECL is straightforward as the extensions in ECL can be freely made in ER (after replacing any large extension with 2-literal ones plus a few additional resolutions) and the rest of ECL is pure clause learning and hence can be p-simulated by resolution.

To prove that ECL p-simulates ER, we need only show that given any ER refutation Π of any CNF Δ, the extensions in Π can be generated in polynomial time by some execution of Algorithm 2. The theorem will then follow given the known fact that clause learning p-simulates resolution. By virtue of Lemma 2, we assume that Π contains no useless extensions.

Let $x \leftrightarrow l_1 \vee l_2$ be the first extension in Π. Given that this extension is not useless, unit propagation on Δ does not produce l_1; hence Algorithm 2 is free to set $\overline{l_1}$ as its first decision assignment, unless $\overline{l_1}$ is already set by unit propagation, in which case we simply skip this decision.

Unit propagation after the setting of $\overline{l_1}$ cannot produce l_2, or the extension $x \leftrightarrow l_1 \vee l_2$ would be useless; hence Algorithm 2 is free to set $\overline{l_2}$ as its next decision assignment, unless $\overline{l_2}$ is already set by unit propagation, in which case we simply skip the decision.

We now have both $\overline{l_1}$ and $\overline{l_2}$ on our assignment stack; hence Algorithm 2 (lines 4 & 5) is free to generate the extension $x \leftrightarrow l_1 \vee l_2$ (after some further assignments where necessary to reach a conflict). This clearly takes polynomial time. The theorem is proved by repeating this process for each subsequent extension in Π. $\square$

5. Empirical Results

In this section we report on an empirical evaluation of our first implementation of ECL, based on the heuristic described earlier that uses part of a learned clause to define each new variable. The reported results shall serve as evidence that the additional theoretical power of ECL can in fact translate into substantial practical gains where the extension heuristic is effective.

It is well known that SAT solver performance can be quite sensitive to the problem or even problem instance. We have hence taken care to ensure that our choice of SAT solver and benchmarks are such that we are comparing with reasonably the best performance of clause learning on some of the most challenging

problems. Specifically, we used as benchmarks a large set of instances, both satisfiable and unsatisfiable, created from a variety of applications and distributed by Miroslav Velev at http://www.miroslav-velev.com/sat_benchmarks.html. This is only one of the standard sources of benchmarks in recent SAT competitions and SAT races, but is one that presents particular difficulties even for the best SAT solvers, including MiniSat 2.0 [16, 17], which solved only 102 out of the 251 instances under a one-hour time limit [18]. In addition, we have included a set of crafted unsatisfiable instances (gt-ordering) that have proved to be empirically difficult for current clause-learning SAT solvers. These are "based on the ordering principle that any partial order on the set $\{1, 2, \ldots, n\}$ must have a maximal element," and have been submitted to the SAT competition by the authors of [3].

We based our implementation of ECL on Tinisat [18], which is a small solver but is particularly competitive on the Velev benchmarks, solving 183 of the instances compared with 102 for MiniSat 2.0 [18]. We inserted code allowing Tinisat to make an extension after each conflict according to the heuristic. By way of variable and value ordering (i.e., the decision heuristic), on making each extension $x \leftrightarrow l_1 \vee l_2$, we set the scores for the new variable x "by inheritance," as follows:

$$score(x) = \frac{score(l_1) + score(l_2)}{2},\ score(\bar{x}) = \frac{score(\bar{l_1}) + score(\bar{l_2})}{2}.$$

Everything else in Tinisat was left unchanged.

Our hardware was a computer cluster featuring a number of CPUs running Linux at 2.4GHz with 4GB of RAM. A one-hour time limit was imposed on each run of a solver.

Table 1 summarizes the overall results of Tinisat vs. the version with extensions, referred to as TinisatX. The SatELite preprocessor [19] was used for both solvers and the preprocessing time is included in the reported timings. The set of Velev benchmark families are the same as in [18], excluding the easy ones (those with no unsolved instances). For each benchmark family and solver, we report the number of instances solved and the total time (in seconds) spent on the solved instances.

Although this implementation of ECL did not result in a consistent improvement across all instances, we would like to zoom in on those cases where improvement did occur, to ascertain the extent to which the extensions positively affected the solver's behavior, in both the overall running time and number of conflicts (i.e., number of leaves of the search tree, an indication of the search tree size as well as proof size in unsatisfiable cases).

Table 1: Overall Results

Benchmark Family	# of Instances	# Solved (Time on Solved)	
		Tinisat	TinisatX
engine-unsat-1.0	10	7 (1434s)	4 (1615s)
fvp-sat.3.0	20	17 (1376s)	20 (501s)
fvp-unsat.3.0	6	0	2 (3196s)
liveness-sat-1.0	10	8 (8422s)	2 (1200s)
liveness-unsat-1.0	12	4 (812s)	4 (417s)
liveness-unsat-2.0	9	3 (6s)	3 (243s)
npe-1.0	6	4 (1634s)	3 (317s)
pipe-ooo-unsat-1.0	15	7 (4552s)	5 (1458s)
pipe-ooo-unsat-1.1	14	8 (2512s)	6 (3983s)
pipe-unsat-1.0	13	9 (4988s)	8 (2827s)
pipe-unsat-1.1	14	9 (2721s)	9 (4549s)
vliw-sat-2.0	9	8 (2808s)	4 (153s)
vliw-sat-2.1	10	6 (7789s)	5 (509s)
vliw-unsat-2.0	9	3 (4337s)	3 (3026s)
vliw-unsat-3.0	2	0	0
vliw-unsat-4.0	4	1 (608s)	0
gt-ordering	7	0	7 (39s)

On the level of benchmark families, there are five where TinisatX performed substantially better:

- **fvp-sat.3.0**: 3 previously unsolved instances (bug05, bug16, bug17) are now solved, in a total of only 362 seconds. For the remaining 17 instances, solved by both, the total number of conflicts went down from 2,280,038 to 54,195, a reduction by a factor of 42.

- **fvp-unsat.3.0**: 2 instances (01, 04) are now solved, taking 3196 seconds, where none could be solved before.

- **liveness-unsat-1.0**: Both solved 4 instances (sharing 3), but the time went down by half for TinisatX, and the number of conflicts went down from 501,049 to 92,577.

- **vliw-unsat-2.0**: Both solved (the same) 3 instances, but the time went down by 30% for TinisatX, and the number of conflicts went down from 2,852,442 to 315,442.

- **gt-ordering**: None could be solved previously, and all 7 are now solved, in a total of only 39 seconds.

Table 2: Additional Positive Cases

Benchmark	Number of		Tinisat			TinisatX			
	Variables	Clauses	Decisions	Conflicts	Time	Decisions	Conflicts	Extensions	Time
npe-1.0/1dlx_c_bug	2974	34392	2730	749	0.3s	2020	160	15	0.2s
pipe-ooo-unsat-1.1/3pipe	1784	23493	31710	10517	0.8s	48611	7225	1278	0.7s
pipe-ooo-unsat-1.1/4pipe	3823	65464	132770	39507	6.9s	300024	15960	4298	4.8s
pipe-unsat-1.0/3pipe	1973	26217	25777	9144	0.7s	40075	5313	836	0.5s
pipe-unsat-1.0/4pipe	4286	77140	102329	32871	5.6s	269456	14126	3116	3.8s
pipe-unsat-1.0/5pipe	7945	184641	292251	92737	35s	1108221	29820	10513	30s
pipe-unsat-1.1/4pipe	3685	65057	114091	37336	6.5s	215317	14792	3410	3.7s
pipe-unsat-1.1/5pipe	6662	146522	257183	65869	19s	877258	27911	7677	18s
vliw-sat-2.0/bug2	200536	8005366	711711	1133	44s	352031	35	4	32s
vliw-sat-2.0/bug5	200503	8005129	6834344	409740	894s	1814177	953	239	50s
vliw-unsat-2.0/iq1	16889	244145	2669266	592335	475s	5089176	61510	23881	114s
vliw-unsat-2.0/iq2	32773	518020	5980951	1011625	1577s	13077261	94932	46533	566s
Total	483,833			2,306,563	3064s		272,737	101,800	824s

In the remaining families, there are still many cases where ECL resulted in a significant improvement, either in the number of conflicts or overall running time. Table 2 presents a collection of these, where we report the size of the CNF (after preprocessing), the number of decisions and conflicts and the running time for both solvers, and in addition the number of extensions made (i.e., number of new variables invented) for TinisatX.

The data in Table 2 give an additional indication of the extent of improvement that the particular heuristic we implemented was able to achieve where it was effective. Overall (bottom row of the table), the number of conflicts went down by an order of magnitude, and the running time by a factor of 3.7. We also note that in these cases the number of extensions made was roughly 21% of the original number of variables on average (for the crafted gt-ordering benchmarks, this percentage was 351%).

As a more direct confirmation of the contribution of extensions to these performance gains, we have also examined how often decisions and resolutions were made on extension variables. For the set of instances in Table 2, 1.8% of the decisions and 5.1% of the resolutions were made on extension variables; for the gt-ordering family, the corresponding percentages are 2.1% and 1.3%.

6. Related Work

A proof system called *augmented resolution* was studied in [20]. This system requires a different modeling language from CNF, where clauses are augmented

with groups of permutations; hence it's not directly comparable to our present work, which concerns proof systems based on CNF.

SAT algorithms and ER have previously been studied in the same context [21, 22]. This work concerns a class of SAT algorithms based on encoding clauses into binary decision diagrams and conjoining them symbolically (with or without variable elimination) to decide satisfiability. As far as we are aware, the power of this class of algorithms as a proof system, particularly in relation to that of clause learning, is currently an open question. The work of [21, 22] shows how the traces of these algorithms can be converted into ER proofs; it does not propose a new SAT algorithm.

Since the initial submission of the present paper, work has been published [23] showing a restricted form of ER to be effective on a set of SAT competition benchmarks. This can be viewed as a new heuristic for extensions and the results as additional evidence that ER can benefit practical SAT solvers. We note that the heuristic of [23] is overall much less aggressive than the one we have used in the present paper (roughly one extension for "every 1000 conflicts" in [23] versus 2.7 conflicts as seen in Table 2).

7. Conclusion

We have presented a practical SAT algorithm, called extended clause learning (ECL), that is strictly more powerful than the current clause learning framework which has largely remained unchanged for the past decade. We formally proved the equivalence in power between ECL and extended resolution in the sense of p-simulation, and empirically evaluated ECL using a particular heuristic for extensions that also helps shorten learned clauses. While improvement was only observed on a subset of the benchmarks used, its magnitude was quite substantial in terms of the reduction in both the search tree size and overall running time.

Based on these theoretical and empirical results, we conclude that ECL has a potential to bring about a new generation of practical SAT solvers that may go beyond the current clause learning solvers in efficiency and scalability at least for certain families of problems.

Despite the existing evidence of the practical benefits of ER, however, it should be noted that a more powerful proof system is often accompanied by an increased difficulty in finding optimal proofs. Under some generally held assumptions, both resolution and ER, in particular, are known to be not *automatizable*, meaning that no algorithm can be guaranteed to find a proof in time polynomial in the size of the shortest proof of a given formula, in either the resolution or ER proof system

[24, 25, 26]. As a result, heuristics for clause learning and ECL are likely to remain effective only for restricted classes of problems.

Acknowledgements

We thank the reviewers for their comments on earlier drafts of this paper and pointers to additional references. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[1] J. Marques-Silva, K. Sakallah, GRASP—A new search algorithm for satisfiability, in: Proceedings of the International Conference on Computer Aided Design (ICCAD), 1996, pp. 220–227.

[2] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an efficient SAT solver, in: Proceedings of the 38th Design Automation Conference (DAC), 2001, pp. 530–535.

[3] P. Beame, H. A. Kautz, A. Sabharwal, Towards understanding and harnessing the potential of clause learning, Journal of Artificial Intelligence Research 22 (2004) 319–351.

[4] S. R. Buss, J. Hoffmann, J. Johannsen, Resolution trees with lemmas: Resolution refinements that characterize DLL algorithms with clause learning, Logical Methods in Computer Science 4 (4).

[5] P. Hertel, F. Bacchus, T. Pitassi, A. V. Gelder, Clause learning can effectively p-simulate general propositional resolution, in: Fox and Gomes [27], pp. 283–290.

[6] K. Pipatsrisawat, A. Darwiche, On the power of clause-learning SAT solvers with restarts, in: I. P. Gent (Ed.), Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP), Vol. 5732 of Lecture Notes in Computer Science, Springer, 2009, pp. 654–668.

[7] K. Pipatsrisawat, A. Darwiche, A lightweight component caching scheme for satisfiability solvers, in: Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT), 2007, pp. 294–299.

[8] K. Pipatsrisawat, A. Darwiche, A new clause learning scheme for efficient unsatisfiability proofs, in: Fox and Gomes [27], pp. 1481–1484.

[9] J. Huang, The effect of restarts on the efficiency of clause learning, in: M. M. Veloso (Ed.), Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), 2007, pp. 2318–2323.

[10] G. Audemard, L. Simon, Predicting learnt clauses quality in modern SAT solvers, in: C. Boutilier (Ed.), Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI), 2009, pp. 399–404.

[11] A. Haken, The intractability of resolution, Theoretical Computer Science 39 (1985) 297–308.

[12] G. S. Tseitin, On the complexity of derivations in the propositional calculus, Studies in Mathematics and Mathematical Logic Part II (1968) 115–125.

[13] S. A. Cook, R. A. Reckhow, On the lengths of proofs in the propositional calculus (preliminary version), in: Proceedings of the Sixth Annual ACM Symposium on Theory of Computing (STOC), ACM, 1974, pp. 135–148.

[14] S. A. Cook, A short proof of the pigeon hole principle using extended resolution, SIGACT News 8 (4) (1976) 28–32. doi:http://doi.acm.org/10.1145/1008335.1008338.

[15] S. A. Cook, R. A. Reckhow, The relative efficiency of propositional proof systems, Journal of Symbolic Logic 44 (1) (1979) 36–50.

[16] N. Eén, N. Sörensson, An extensible SAT-solver, in: Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT), 2003, pp. 502–518.

[17] N. Eén, N. Sörensson, MiniSat—A SAT solver with conflict-clause minimization, in: SAT 2005 Competition, 2005.

[18] J. Huang, A case for simple SAT solvers, in: C. Bessiere (Ed.), Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP), Vol. 4741 of Lecture Notes in Computer Science, Springer, 2007, pp. 839–846.

[19] N. Eén, A. Biere, Effective preprocessing in SAT through variable and clause elimination, in: Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT), 2005, pp. 61–75.

[20] H. E. Dixon, M. L. Ginsberg, D. K. Hofer, E. M. Luks, A. J. Parkes, Implementing a generalized version of resolution, in: Proceedings of the 19th AAAI Conference on Artificial Intelligence (AAAI), 2004, pp. 55–60.

[21] C. Sinz, A. Biere, Extended resolution proofs for conjoining BDDs, in: Proceedings of the First International Computer Science Symposium in Russia (CSR), 2006, pp. 600–611.

[22] T. Jussila, C. Sinz, A. Biere, Extended resolution proofs for symbolic SAT solving with quantification, in: Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT), 2006, pp. 54–60.

[23] G. Audemard, G. Katsirelos, L. Simon, A restriction of extended resolution for clause learning SAT solvers, in: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI), 2010.

[24] J. Krajíček, P. Pudlák, Some consequences of cryptographical conjectures for S_2^1 and EF, Information and Computation 140 (1) (1998) 82–94.

[25] M. L. Bonet, T. Pitassi, R. Raz, On interpolation and automatization for Frege systems, SIAM Journal on Computing 29 (6) (2000) 1939–1967.

[26] M. Alekhnovich, A. Razborov, Resolution is not automatizable unless W[P] is tractable, in: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, Washington, DC, USA, 2001, p. 210.

[27] D. Fox, C. P. Gomes (Eds.), Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI), July 13-17, 2008, AAAI Press, 2008.