A Restriction of Extended Resolution for Clause Learning SAT Solvers*

Gilles Audemard
Univ. Lille-Nord de France
CRIL/CNRS UMR8188
Lens, F-62307, France
audemard@cril.fr

George Katsirelos
Univ. Lille-Nord de France
CRIL/CNRS UMR8188
Lens, F-62307, France
gkatsi@gmail.com

Laurent Simon
Univ. Paris-Sud
LRI/CNRS UMR 8623 / INRIA Saclay
Orsay, F-91405, France
simon@lri.fr

Abstract

Modern complete SAT solvers almost uniformly implement variations of the clause learning framework introduced by Grasp and Chaff. The success of these solvers has been theoretically explained by showing that the clause learning framework is an implementation of a proof system which is as poweful as resolution. However, exponential lower bounds are known for resolution, which suggests that significant advances in SAT solving must come from implementations of more powerful proof systems.

We present a clause learning SAT solver that uses extended resolution. It is based on a restriction of the application of the extension rule. This solver outperforms existing solvers on application instances from recent SAT competitions as well as on instances that are provably hard for resolution, such as XOR-SAT instances.

Introduction

There has been significant progress in the practice of satisfiability testing in the past decade. The turning point was the clause learning (CDCL) framework introduced in GRASP (Marques Silva and Sakallah 1999), as well as the algorithmic improvements and heuristics proposed in Chaff (Moskewicz et al. 2001). The reengineering of Minisat (Eén and Sörensson 2003) then helped identify the critical components of modern SAT solvers. The substantial advantage of CDCL over previous algorithms has been explained on the theoretical side by showing that not only are such algorithms not restricted by the weak tree resolution proof system (Beame, Kautz, and Sabharwal 2004), as was the case for the DPLL class of algorithms (Davis, Logemann, and Loveland 1962; Bayardo and Schrag 1997), but they are actually as powerful as unrestricted resolution.

Although current solvers are very effective for instances generated from formal verification problems (Prasad, Biere, and Gupta 2005), new instances arising from areas such as cryptography and bioinformatics (SAT Competition 2009) have been quite challenging. In order to achieve significant speed-ups, we propose in this paper to extend the CDCL algorithm with the ability to use a more powerful underlying proof system. A natural candidate for this is Extended Resolution (ER). It involves a conceptually simple addition to the Resolution system already used in CDCL: the ability to add arbitrary lemmas to the formula being considered. Despite this simplicity, ER is as powerful as the most powerful known family of proof systems, Extended Frege Systems (Urquhart 2001). In fact, no family of instances has been shown to be hard for ER.

However, there is a significant obstacle in implementing a solver based on ER: choosing the right lemmas to add to the formula is hard, as the number of choices is large and there seems to be no guide as to which lemmas might be relevant. So while ER has been used to construct short proofs of problems that are hard for RES (e.g., (Cook 1976)), no solver has been proposed that implements it. There have, however, been several attempts to use alternate proof systems: for example, BDD-based SAT solvers can be seen as constructing ER proofs (Sinz and Biere 2006; Chatalic and Simon 2001), while ER has also been used to exploit symmetry (Schaafsma, Heule, and van Maaren 2009). The challenge is to design an algorithm and a set of heuristics that implements unrestricted ER and performs in practice similarly to CDCL solvers.

In our work, we propose a simple restriction of ER which reduces the number of choices for new lemmas based on the proof that has been generated so far. This is a different approach to those mentioned earlier, as the link to ER is explicit and may potentially be as powerful as unrestricted ER. Another innovation of our design is that we treat new variables as an "any-space" resource similar to clauses: we extend often but discard unused variables so as not exhaust the available space or to allow the cost of unit propagation caused by the new variables to to dominate the runtime.

This paper is organized as follows: we first recall the background of CDCL solvers, proof systems and extended resolution. Then, we describe the restriction of ER that we use and show how it can improve on resolution. We then describe how this restriction of ER can be efficiently embedded in CDCL solvers. We experimentally show that adding ER rules to CDCL solvers significantly impoves their performance on large set of challenging benchmarks.

*This work is (partially) supported by ANR UNLOC project: ANR 08-BLAN-0289-01.

Algorithm 1 Conflict Driven - Clause Learning algorithm

```
 1: procedure CDCL(φ)
 2:     loop
 3:         Propagate
 4:         if Clause C is FALSE then
 5:             C′ = AnalyzeConflict(C)
 6:             if C′ = ∅ then
 7:                 Report UNSAT and exit
 8:             Store C′
 9:             Backtrack until C′ is not FALSE
10:         else
11:             if All variables are assigned then
12:                 Report SAT and exit
13:             Pick a branching literal l
14:             Assert l in a new decision level
```

Background

The SAT problem consists of finding an assignment to all variables of a propositional formula ϕ, expressed in conjuctive normal form (CNF) so that all clauses of ϕ is satisfied. In particular, the instance ϕ has n Boolean variables $x_1, \ldots, x_n$. For each variable x_i, there exist two *literals*, x_i and $\overline{x}_i$. We write l for an arbitrary literal and $\overline{l}$ for its negation. The CNF formula is a conjunction of disjunctions, and each disjunction is called a *clause*. As a matter of convenience, we view a CNF formula as a set of clauses and each clause as a set of literals. We assume the reader is familiar with the notions of backtracking search and unit propagation.

In this paper, we deal with CDCL algorithms (Marques Silva and Sakallah 1999; Moskewicz et al. 2001; Eén and Sörensson 2003). These complete solvers are extensions of the well known DPLL procedure. They incorporate lazy data structures, backjumping, nogoods, dynamic heuristics and restarts. Algorithm 1 gives the general scheme of a CDCL solver. At each step of the main loop, it performs unit propagation (line 3). Then, if a conflict occurs (line 4), it computes a *nogood* (also called asserting clause) by resolution (Zhang et al. 2001) (line 5) and stores it. It backjumps (lines 8–9) until the nogood becomes unit and performs propagation again. This may create a new conflict, in which case it repeats the backjumping process. If no conflict is detected, it branches on a new variable (lines 13–14). Search finishes when all variables are assigned (lines 11–12) or when the empty clause is derived (lines 6–7).

Proof systems

A propositional refutation proof system is a polynomial time algorithm V, such that for all propositional formulas ϕ, ϕ is unsatisfiable iff there exists a string P (a *proof* of unsatisfiability or *refutation* of ϕ) such that V accepts the input (ϕ, P). Propositional proof systems have been studied in proof complexity as one way to resolve the NP vs $coNP$ problem, as one can show that $NP = coNP$ iff there exists a polynomially bounded propositional proof system, i.e., for every unsatisfiable formula ϕ, the size of the smallest P such that V accepts (ϕ, P) is polynomial in the size of ϕ (Cook and Reckhow 1974). Proof complexity provides the tools to study the size of the proofs admitted by a single proof system as well as the relations between different systems. In the following, we say that a proof system V admits *short* proofs for a family of instances $\mathcal{I}$ if the size of the minimum proof grows polynomially with the size of the instances. We then say that $\mathcal{I}$ is *easy* for V. Conversely, V admits superpolynomially (exponentially) *long* proofs for a family $\mathcal{I}$ when the size of the minimum proof grows superpolynomially (exponentially) with the size of the instance and $\mathcal{I}$ is superpolynomially (exponentially) *hard* for V.

Given two propositional proof systems V_1 and V_2, we say that V_1 (strongly) *p-simulates* V_2 iff there exists a polynomial-time computable function f such that V_2 accepts (ϕ, P) iff V_1 accepts $(\phi, f(P))$. Two proof systems are said to be polynomially equivalent if they can p-simulate each other. There exists a superpolynomial (exponential) separation between V_1 and V_2 if V_1 can p-simulate V_2 and there exists a family of instances $\mathcal{I}$ that is easy for V_1 and superpolynomially (exponentially) hard for V_2.

The most widely used propositional proof system is RES (Robinson 1965). A proof in RES is a sequence of clauses $C_1, \ldots, C_m, C_{m+1}, \ldots, C_s = \square$, where $\square$ is the empty clause. The clauses $C_1, \ldots, C_m$ are the formula ϕ. Clauses after that are produced using the *resolution rule*. Specifically, the clause C_k is generated from the clauses C_i and C_j, where $i < j < k$, iff there exists a literal l with $C_i \equiv l \vee \alpha$, $C_j \equiv \overline{l} \vee \beta$, $C_k \equiv \alpha \vee \beta$ and $l \in \alpha \implies \overline{l} \notin \beta$. In this case, we write $C_k = C_i \otimes_l C_j$ and call C_k the *resolvent* of C_i and C_j. While not necessary for either completeness or succinctness, RES often contains the weakening rule which allows us to derive $C_k \equiv \alpha \vee \beta$ from $C_i \equiv \alpha$, where $i < k$ and $l \in \alpha \implies \overline{l} \notin \beta$.

The trace of execution of a CDCL solver can itself be seen as a proof of unsatisfiability. In fact, the operation of CDCL solvers has been formalized as a proof system in a series of papers (Beame, Kautz, and Sabharwal 2004; Van Gelder 2005; Hertel et al. 2008; Pipatsrisawat and Darwiche 2009), and this proof system has been shown to p-simulate RES. More precisely, (Hertel et al. 2008) showed that CDCL p-simulates RES if it is allowed to perform some preprocessing (called effectively p-simulation (Hertel, Hertel, and Urquhart 2007)), while (Pipatsrisawat and Darwiche 2009) showed that CDCL with restarting p-simulates RES. This is significant because all previous SAT algorithms, including DPLL, are known to be exponentially weaker than resolution. For example, DPLL can be p-simulated by a restriction of resolution called tree resolution, which is only allowed to use a non-input clause once to derive new clauses, and which is exponentially weaker than RES.

Unfortunately, RES has its own limitations. Many kinds of reasoning, such as counting and modulo counting cannot be efficiently performed using resolution. This has led to the discovery of several families of instances which were proven to be exponentially hard for RES. The first family for which exponential lower bounds were shown is instances which encode the pigeonhole principle (Haken 1985), followed by many others and a more general result (Ben-Sasson and Wigderson 2001). These limitations suggest that significant improvement in the performance of SAT solvers must come

by implementing solvers which cannot be p-simulated by resolution. This is the aim of this paper.

Extended Resolution

Extended resolution (ER) (Tseitin 1983) is a generalization of resolution with one additional rule: at any point, a fresh variable y (one that does not appear in ϕ) may be introduced along with the clauses that encode $y \Leftrightarrow l_1 \vee l_2$ for any pair of literals l_1 and l_2 of ϕ. A proof in ER is then a sequence $C_1, \ldots, C_m, \{C|E\}_{m+1}, \ldots, C_s = \Box$, where C_i is a resolution step and E_i is an extension step. In fact, an ER refutation can introduce all variables in the beginning and then proceed as a resolution proof over the extended formula, so the proof can be written as $C_1, \ldots, C_m, E_{m+1}, \ldots, E_{m+N}, C_{m+N+1}, \ldots, C_s = \Box$.

Although it is a seemingly simple extension of RES, ER is as powerful as Extended Frege Systems (Urquhart 2001), for which no lower bounds are currently known. In fact it has been demonstrated that ER can naturally capture the inductive arguments that are used to refute formulas that are hard for RES (Cook 1976; Krishnamurthy 1985).

Despite the apparent simplicity of ER and its similarity to RES, no solver has been proposed so far that is based on ER. The reason for this is that it is hard to devise a heuristic that introduces variables that are likely to help find a short refutation. We propose a particular view of the reason why ER can help produce shorter proofs. We will then attempt to use this view to design heuristics that guide a CDCL solver to introduce fresh variables during search.

Suppose l_1 and l_2 are literals of different variables and we have derived in our proof the clauses $C_i = \overline{l}_1 \vee \alpha$ and $C_j = \overline{l}_2 \vee \alpha$. Further suppose that these clauses are each resolved with the same sequence of clauses $C_{m_1}, \ldots, C_{m_k}$ to derive $C'_i = \overline{l}_1 \vee \beta$ and $C'_j = \overline{l}_2 \vee \beta$. In this case, we claim that it can be beneficial to extend the formula with the variable $x \Leftrightarrow l_1 \vee l_2$. Then, we can resolve C_i and C_j with $\overline{x} \vee l_1 \vee l_2$ to get $C_k = \overline{x} \vee \alpha$. The clause C_k can then be resolved with $C_{m_1}, \ldots, C_{m_k}$ to derive $\overline{x} \vee \beta$. Thus, any steps that are common in the derivations of C'_i and C'_j will not be replicated, thereby *compressing* the proof.

In fact, we can generalize this argument. Suppose the clauses C_i and C_j differ in more than l_1 and l_2 but are instead of the form $C_i = \overline{l}_1 \vee \alpha \vee \beta$ and $C_j = \overline{l}_2 \vee \alpha \vee \gamma$, where $\alpha \vee \beta \vee \gamma$ is not a tautology. If there exists a sequence of clauses $C_{m_1}, \ldots, C_{m_k}$ such that C_i and C_j are both resolved with $C_{m_1}, \ldots, C_{m_k}$ to derive $C'_i = \overline{l}_1 \vee \delta$ and $C'_j = \overline{l}_1 \vee \delta$ where $\beta \subseteq \delta$ and $\gamma \subseteq \delta$, we can again apply the transformation outlined above to avoid replicating the resolution sequence $C_{m_1}, \ldots, C_{m_k}$.

Also note that in this scheme, the clauses C_i and C_j need not be input clauses but can be derived clauses. This is important from a practical point of view, because it allows us to introduce variables not only by examining the input formula, but also by examining the resolution proof, as it is being generated. The following formalizes this scheme as a proof system.

Definition 1 (Local Extended Resolution) *LER is the propositional proof system which includes the resolution rule as well as the extension rule that, at step k we can introduce the variable $z \Leftrightarrow l_1 \vee l_2$ if there exist clauses $C_i \equiv \overline{l}_1 \vee \alpha$ and $C_j \equiv \overline{l}_2 \vee \beta$ with $i < k, j < k$, where α and β are disjunctions such that $l \in \alpha \implies \overline{l} \notin \beta$.*

The difference of LER from ER is that it requires clauses of the appropriate form to be derived before a variable can be introduced. Such clauses might not be derived at all in an ER proof. The advantage of LER is that proofs in it are much less unpredictable than those in ER. In particular, each introduced variable is related to the proof that is being generated. This mitigates somewhat the non-deterministic nature of the proof system.

The restriction of LER is not too severe. In fact, it is easy to verify that some ER proofs of problems that are hard for RES can be easily transformed into LER proofs, including Cook's ER refutation of the pigeonhole principle. This shows that LER is exponentially more powerful than resolution. On the other hand, it may not always be easy to derive the two clauses that are needed for each extended variable to be introduced. This can make the minimum LER proof larger than the minimum ER proof. We do not currently know whether there exist instances that are hard for LER but easy for ER or whether LER can p-simulate ER.

Given that CDCL solvers have been described as resolution engines (Huang 2007), we can empirically claim that we have enough information during the execution of a CDCL solver to implement a formalism like LER and extend the formula with variables that are likely to help shorten the proof.

This insight is exploited in the next section to describe a particular implementation of this idea.

A CDCL solver based on local extensions

The scheme described in the previous section restricts the number of pairs of literals that need to be considered for extension. However, even under this restriction, the number of choices in many problems is too large to consider systematically extending the formula with all candidates. We therefore design heuristics here whose main aim is to be conservative: we aim to maintain the good performance of CDCL solvers on application instances, even at the cost of not achieving all the benefits of the LER system in practice, and failing to solve tricky formulas efficiently. This means that we need to control very strictly the number of fresh variables that we will add.

In general, most of the reasoning performed by CDCL solvers is during conflict analysis. This includes clause learning, as pointed out in Algorithm 1, but also updates of heuristic scores and evaluation restart strategies. This is also the logical choice for detecting candidate pairs of literals, as we can examine each clause as it is produced.

Fast (but incomplete) detection of interesting pairs. The most important restriction and main difference of our implementation to the LER system is that we restrict our attention to pairs of clauses of the form $\overline{l}_1 \vee \alpha, \overline{l}_2 \vee \alpha$. This restriction

can potentially reduce the power of the proof system implemented by our solver. However, in the alternative case, the number of extensions performed would then be too large. Then, the task of keeping the number of extended variables manageable would be more challenging.

A number of further restrictions were made to the application of the extension rule. Intuitively, the heuristics of CDCL solvers are tuned to keep the solver exploring the same search space despite restarts, so it is likely that pairs of clauses that match the scheme $\overline{l}_1 \vee \alpha$, $\overline{l}_2 \vee \alpha$ will be discovered close to each other. Therefore, it would be sufficient to examine only a small window of recent clauses to detect many such pairs of clauses. In fact, we push this reasoning to its extreme and set the window size to 1: we consider only successive learnt clauses. We can then assume that the literals $\overline{l}_1$ and $\overline{l}_2$ are in fact the UIP literals discovered during conflict analysis. Thus, our candidate generation scheme is: if successive clauses learnt by the solver have the form $\overline{l}_1 \vee \alpha$, $\overline{l}_2 \vee \alpha$, we introduce the variable $z \Leftrightarrow l_1 \vee l_2$, if it has not already been introduced.

Successive applications of the ER rule. If n successive learnt clauses have the form $l_i \vee C$ $(1 \leq i \leq n)$ then we add the rules $z_1 \Leftrightarrow \overline{l}_1 \vee \overline{l}_2$, but also all $z_i \Leftrightarrow \overline{l}_i \vee \overline{l}_{i+1}$ for $2 \leq i < n$. Another alternative may be to add $z_i \Leftrightarrow \overline{z}_i \vee \overline{l}_{i+1}$ instead of $z_i \Leftrightarrow \overline{l}_i \vee \overline{l}_{i+1}$ but we found empirically that this performed worse.

Extended variables in new clauses. As soon as a fresh variable $z \Leftrightarrow l_1 \vee l_2$ is introduced, we have to ensure that we replace new clauses in the remaining proof that match the form $l_1 \vee l_2 \vee \beta$ with $z \vee \beta$. This systematic reduction is important because this allows the new variable z to be propagated even when the learnt clause $l_1 \vee l_2 \vee \beta$ would not have been. For example, if all literals in β are false, the clause $l_1 \vee l_2 \vee \beta$ is not unit, but $z \vee \beta$ is. Empirically, we discovered that if we do not perform this reduction step, then z will almost never occur in conflict analysis, so z will not be used in the resolution proof that is produced. Note that we restrict the application of this reduction step to clauses learnt after we introduce z, mostly for efficiency.

To support this reduction step, we maintain a hash table of pairs of extended literals, and we probe the hash table each time a conflict is performed to replace pairs of literals by their extended variable. It has to be noticed that the use of such a hash table implies a special case were the reduction introduces a choice. Suppose we learn the clause $C \equiv l_1 \vee l_2 \vee l_3 \vee \beta$ and we have previously introduced the two new variables z_1 and z_2 such that $z_1 \Leftrightarrow l_1 \vee l_2$ and $z_2 \Leftrightarrow l_2 \vee l_3$. Then our iterative procedure of reduction will have to make a choice between the two clauses $C_1 \equiv z_1 \vee l_3 \vee \beta$ or $C_2 \equiv l_1 \vee z_2 \vee \beta$. This choice is handled by prefering to use extended variables with a higher VSIDS score at the time of the reduction. Note that although this manipulation is crucial for the performance of the solver, it is only a heuristic that encourages resolution on introduced variables, and does not affect the theoretical properties of the solver.

Contrary to the case of clauses of the form $l_1 \vee l_2 \vee \beta$, there is no need to compress any new pair of clauses of the form $\overline{l}_1 \vee C$, $\overline{l}_2 \vee C$ by substituting them with the single clause $\overline{z} \vee C$. If either $\overline{l}_1$ or $\overline{l}_2$ are propagated according to the two previous clauses, then z will be propagated, and thus potentially used during conflict analysis (and thus resolution).

Reducing unintended side effects. At each conflict, if an LER rule is triggered, we may add the 3 clauses $\overline{z} \vee l_1 \vee l_2$, $z \vee \overline{l}_1$ and $z \vee \overline{l}_2$. We need to ensure at this stage that no new clause is made unit at a higher decision level than the current backjump level, otherwise we have to jump back to that level and update the implication stack accordingly. However, because of our restrictions, this case will never happen. Indeed, consider again two successive learnt clauses $\overline{l}_1 \vee \alpha$ and $\overline{l}_2 \vee \alpha$. We add the three clauses that encode $z \Leftrightarrow l_1 \vee l_2$. Then, because l_1 and l_2 were UIP literals, $\overline{z} \vee l_1 \vee l_2$ will be TRUE and z will be propagated to TRUE according to $z \vee \overline{l}_1$. Hence, our restriction over detected pairs of clauses also ensures that the desirable property of learnt clauses that they assert a literal on backtracking still holds even when introducing new variables and additional clauses at a given conflict. In particular, we don't force the CDCL solver to restart earlier, or to particularly reorder its decision dependencies.

Removing unused extended variables In CDCL solvers, the learnt clause database is regularly reduced, in order to keep the cost of unit propagation from dominating the runtime. The same argument applies for introduced variables and their corresponding clauses. We keep track of clauses that encode the LER rule for any given variable for future deletion. Then, whenever we reduce the database of learnt clauses, we also remove those extended variables that have a low VSIDS score, indicating that they are not used in the resolution proof.

Empirical evaluation

In this section, we report on the empirical evaluation of our ideas with implementations based on two CDCL solvers, MINISAT and GLUCOSE. The modified versions embedding our LER mechanisms are reported here as MINISATER and GLUCOSER, respectively. We also compare against PRECOSAT, the other winner of the application track of the last competition, along with GLUCOSE. Finally, note that our version of MINISAT slightly differs from the publicly available one, as we changed the restart strategy to follow the Luby sequence (starting at 32) and also added phase saving. Both modifications improve the performance of MINISAT on industrial problems, and are now used in most CDCL solvers.

We report results on two families of benchmarks: (1) industrial/application benchmarks from the SAT07 and SAT09 competitions (SAT Competition 2009) and (2) some additional benchmarks known to be hard for resolution-based algorithms (pigeonhole problems and Urquhart problems (Urquhart 1987)) or experimentally proven hard for recent

	SAT 07	SAT 09
PRECOSAT	167 (91 - 76)	211 (129 - **82**)
GLUCOSE	185 (111 - 74)	211 (132 - 79)
GLUCOSER	**191** (**113** - **78**)	**213** (**133** - 80)
MINISAT	142 (81 - 61)	190 (118 - 72)
MINISATER	146 (85 - 61)	198 (123 - 75)

Table 1: Performance of solvers with and without the LER rule. Instances come from the application category of the SAT 2007 (234 instances) and SAT 2009 (292 instances) competitions. For each solver, we report the number of solved instances with, in parenthesis, the number of UNSAT and SAT instances.

solvers. We used a farm of quad-core Intel XEON X5550 – 2.66 GHz with 32 GB RAM. Time limit is set to 5000 seconds. Results are reported in seconds.

The tunable parameters of our solver were experimentally set as follows: we only introduce variables when the pair of most recently learned clauses have size at least 4. Every time the clause database is reduced, we attempt to remove half the introduced variables.

Results from previous competitions

Table 1 summarizes our results on industrial benchmarks from previous contests. Those sets of benchmarks are really hard, and even a small increase in the number of solved instances is a clear win (let us point out that PRECOSAT and GLUCOSE were ex-aequo in the number of solved benchmarks in the category SAT +UNSAT). The challenge was here to increase the reasoning power of CDCL solvers while keeping their overall good performance.

Results on known "hard" instances

Table 2 highlights results obtained from two sets of challenging benchmarks. The first are instances that are known hard for resolution, encodings of the pigeonhole principle and the Urquhart instances. We notice that on pigeonhole problems, MINISATER does not improve on the base solver, and in fact MINISAT is the best solver for these instances. However, we offer two counterpoints here. We attempted to generate an easy instance for MINISAT by generating not only the clauses that encode the pigeonhole principle but also all the extended variables that are used in a minimal resolution refutation of PHP (Cook 1976). Somewhat surprisingly, these instances were hard for MINISAT as well. This implies that in this case extending the formula is not enough. Instead, we also need to modify the branching heuristic. However, this runs counter to our goal of maintaining the performance of the base solver in application instances. The second counterpoint that we offer is that MINISATER and GLUCOSER improve significantly on MINISAT and GLUCOSE when solving instances that encode the functional pigeonhole principle. Unfortunately, even in this case the performance of the solver does not scale polynomially with the size of the problem.

The picture is more clearly positive in Urquhart instances. MINISATER and GLUCOSER solve instances that appear well out of reach of the other solvers, even if it seems that the performance does not scale polynomially.

The most important part of Table 2 is a set of hard benchmarks from previous competitions, most of them unsolved during the given contest. These clearly demonstrates the ability of GLUCOSER, and to a lesser extent MINISATER, to solve instances that are out of the reach of previous solvers.

Some important points to note in these results is that first, the solver introduces a large number of new variables – roughly one for every 1000 conflicts. Second, (not shown in the table) the solver branches on the new variables frequently. This means that they have a high VSIDS score, which in turn means that they are used in resolution, thus the proof produced is an LER proof, not simply a resolution proof. This implies firstly that even our restriction of the LER rule finds enough learned clauses to extend the formula significantly, and secondly that LER admits significantly smaller proofs for these instances.

Conclusions

We have proposed a restriction of extended resolution with two important advantages over unrestricted ER: first, the variables that are introduced are related to the proof as it gets generated. Second, the number of pairs of literals that are candidates for extension is significantly reduced. Both of these advantages mean that it is easier to design heuristics for a solver based on this proof system. We built such a solver and showed that it outperfoms state-of-the-art CDCL solvers on benchmarks from recent competitions. Our implementation was based on MINISAT and GLUCOSE, but the techniques we describe can be embedded easily in any CDCL solver.

Several open questions remain from this work. From the practical point of view, we would like to design heuristics for more aggressively introducing new variables without introducing a significant runtime overhead. This would mean considering more pairs of clauses beyond the window of the last few conflicts, but also generalizing the candidates to clauses that do not match exactly. From the theoretical point of view, we would like to determine exactly where LER places in the proof complexity hierarchy. If it can be shown that it p-simulates ER, this would be a significant step towards understanding the power of ER and harnessing it in SAT solvers.

Acknowledgements. We thank Nicolas Prcovic for suggesting "pre-extending" pigeonhole instances.

References

Bayardo, R. J., and Schrag, R. C. 1997. Using csp look-back techniques to solve real-world sat instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 203–207.

Beame, P.; Kautz, H.; and Sabharwal, A. 2004. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research* 22:319–351.

Ben-Sasson, E., and Wigderson, A. 2001. Short proofs are narrow—resolution made simple. *J. ACM* 48(2):149–169.

benchmark	SAT ?	PRECOSAT	GLUCOSE	GLUCOSER	MINISAT	MINISATER
hole-11	UNSAT	90s / 1,314	78s / 552	127s / 848 / 930	7s / 148	20s / 359 / 2195
hole-12	UNSAT	963s / 8,112	1,167s / 4,588	208s / 1,333 / 1,334	40s / 405	363s / 1,825 / 6,927
Urq3_5	UNSAT	142s / 4,570	21s / 791	8s / 235 / 1,478	398s / 2,119	10s / 318 / 2,702
Urq4_5	UNSAT	–	3,106s / 28,181	26s / 527 / 3,103	–	11s / 333 / 3,191
Urq5_5	UNSAT	–	–	545s / 5,021 / 7,298	–	107s / 1,488 / 7,503
aloul-chnl11-13	UNSAT	–	–	310s / 1,794 / 1,233	130s / 808	88s / 859 / 4,665
SAT_dat.k75†	UNSAT	–	–	614s / 5,544 / 23,042	–	–
dated-5-19-u†	UNSAT	–	–	3,127s / 9,063 / 9,202	–	–
9dlx_vliw_at_b_iq8†	UNSAT	–	4,322s / 6,672	3,609s / 4,733 / 14,490	–	–
sortnet-8-ipc5-h19-sat†	SAT	3,395s / 2,179	–	3,672s / 7,147 / 6,921	–	–
vmpc_34	SAT	–	2,616s / 15,833	1,565s / 10,022 / 31,576	–	–
rbcl_xits_08	UNSAT	1,286s / 3,600	–	3,067s / 8,069 / 8,916	–	–
simon-s02b-k2f-gr-rcs-w8	UNSAT	–	–	3,622s / 7,904 / 13,567	–	3,797s / 5,121 / 30,890

Table 2: Results on a selection of challenging instances. For each instance, we report whether it is satisfiable, the time needed by each solver to solve it, the number of conflicts (in thousands) and the number of extended vars (after the "/" , if extended resolution is used), or '–' if the solver timed out. Instances marked with †were not solved by any solver in the 07 and 09 competition. Other problems were solved by at most 5 solvers.

Chatalic, P., and Simon, L. 2001. Multiresolution for SAT checking. *International Journal on Artificial Intelligence Tools* 10(4):451–481.

Cook, S. A., and Reckhow, R. A. 1974. On the lengths of proofs in the propositional calculus (preliminary version). In *STOC*, 135–148. ACM.

Cook, S. A. 1976. A short proof of the pigeon hole principle using extended resolution. *SIGACT News* 8(4):28–32.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A machine program for theorem proving. *Communications of the ACM* 5:394–397.

Eén, N., and Sörensson, N. 2003. An extensible SAT-solver. In *proceedings of SAT*, 502–518.

Haken, A. 1985. The intractability of resolution. *Theoretical Computer Science* 39:297–308.

Hertel, P.; Bacchus, F.; Pitassi, T.; and Gelder, A. V. 2008. Clause learning can effectively p-simulate general propositional resolution. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-2008)*, 283–290.

Hertel, A.; Hertel, P.; and Urquhart, A. 2007. Formalizing dangerous sat encodings. In *Proceedings of SAT*. 159–172.

Huang, J. 2007. The effect of restarts on the efficiency of clause learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 2318–2323.

Krishnamurthy, B. 1985. Short proofs for tricky formulas. *Acta Informatica* 22(3):253–275.

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

Moskewicz, M.; Madigan, C.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 39th Design Automation Conference*, 530–535.

Pipatsrisawat, K., and Darwiche, A. 2009. On the power of clause-learning SAT solvers with restarts. In Gent, I. P., ed., *Principles and Practice of Constraint Programming - CP 2009*, volume 5732. Berlin, Heidelberg: Springer Berlin Heidelberg. chapter 51, 654–668.

Prasad, M.; Biere, A.; and Gupta, A. 2005. A survey of recent advances in SAT-based formal verification. *journal on Software Tools for Technology Transfer* 7(2):156–173.

Robinson, J. A. 1965. A machine oriented logic based on the resolution principle. *Journal of the ACM* 12(1):23–41.

2009. `http://www.satcompetition.org`.

Schaafsma, B.; Heule, M.; and van Maaren, H. 2009. Dynamic symmetry breaking by simulating zykov contraction. In Kullmann, O., ed., *Theory and Applications of Satisfiability Testing - SAT 2009*, volume 5584. Berlin, Heidelberg: Springer Berlin Heidelberg. chapter 22, 223–236.

Sinz, C., and Biere, A. 2006. Extended resolution proofs for conjoining bdds. In Grigoriev, D.; Harrison, J.; and Hirsch, E. A., eds., *CSR*, volume 3967 of *Lecture Notes in Computer Science*, 600–611. Springer.

Tseitin, G. 1983. On the complexity of proofs in propositional logics. In Siekmann, J., and Wrightson, G., eds., *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, volume 2. Springer-Verlag.

Urquhart, A. 1987. Hard examples for resolution. *JACM* 34(1):209–219.

Urquhart, A. 2001. The complexity of propositional proofs. 332–342.

Van Gelder, A. 2005. Pool resolution and its relation to regular resolution and DPLL with clause learning. In *Logic for Programming, Artificial Intelligence, and Reasoning*, Lecture Notes in Computer Science. chapter 40, 580–594.

Zhang, L.; Madigan, C.; Moskewicz, M.; and Malik, S. 2001. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of IEEE/ACM International Conference on Computer Design (ICCAD)*, 279–285.