Factoring Learned Clauses

Florian Pollitt
University Freiburg, Freiburg, Germany

Zachary Battleman
Carnegie Mellon University, Pittsburgh, Pennsylvania, US

Mathias Fleury
University Freiburg, Freiburg, Germany

Yakir Vizel
Technion, Israel Institute of Technology, Israel

Marijn J. H. Heule
Carnegie Mellon University, Pittsburgh, Pennsylvania, US

Armin Biere
University Freiburg, Freiburg, Germany

Randal E. Bryant
Carnegie Mellon University, Pittsburgh, Pennsylvania, US

Abstract

Modern SAT solvers are based on the conflict-driven clause learning (CDCL) paradigm, which can be simulated by the resolution proof system. This limits solver effectiveness on instances known to be hard for resolution. Certain approaches, such as parity reasoning, have been shown to be effective in this context, but are hard to integrate with CDCL, in particular, with mainstream proof certificates. The powerful yet simple Extended Resolution (ER) proof system provides an alternative but is not widely used in SAT solving despite having proof certificates for decades and using it effectively remains an open challenge. This paper revisits previous work on ER, which factors out repeated parts of learned clauses during conflict analysis, and explores how their original strategy benefits from 15 years of improvements in the state-of-the-art solver CaDiCaL. We further propose a new, less intrusive inprocessing approach based on factoring XOR and ITE gates from learned clauses globally. Previous work on bounded variable addition focused on AND gates and original clauses only. Our experimental evaluation shows substantial improvements on hard combinatorial benchmark families without performance degradation on the SAT Competition.

2012 ACM Subject Classification Theory of computation → Automated reasoning

Keywords and phrases SAT solving, Extended Resolution, CDCL, Inprocessing

Digital Object Identifier 10.4230/LIPIcs.SAT.2026.36

Supplementary Material *Dataset (Log Files)*: `https://doi.org/10.5281/zenodo.20154935` [38]

Acknowledgements Supported by the state of Baden-Württemberg through bwHPC, the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG, and a gift from Intel Corporation. The research leading to these results is also partially funded by the Israel Science Foundation (ISF), grant no. 2875/21 and by the European Union (ERC, StrongMC, 101231745). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

1 Introduction

One of the greatest strengths of the conflict-driven clause learning (CDCL) paradigm [34] is its flexibility and extensibility. It accommodates a wide range of optimizations and

© Florian Pollitt, Zachary Battleman, Mathias Fleury, Yakir Vizel, Marijn J. H. Heule, Armin Biere, and Randal E. Bryant;
licensed under Creative Commons License CC-BY 4.0

heuristics during search [3, 36, 37]. This is perhaps best exemplified by the development of "inprocessing" [27]—algorithms for the simplification and transformation of formulas that exploit not only the structure of the original formula, but also information learned throughout the solver's execution. Although CDCL is founded on resolution, a relatively weak proof system [17], inprocessing techniques enable the controlled application of stronger proof systems, often yielding substantial performance gains. In this paper, we focus on extended resolution (ER) [41] as one such stronger proof system. For background on SAT, proof systems, and inprocessing, we refer the reader to the *Handbook of Satisfiability* [10].

Our first contribution revisits the idea introduced in GlucosER [2], implemented within the modern solver CaDiCaL [8] (Sect. 2). GlucosER is built around a specific pattern of extended resolution: when two clauses are learned in sequence of the form $\overline{\ell_1} \vee C$ and $\overline{\ell_2} \vee C$, it introduces the definition $z \leftrightarrow \ell_1 \vee \ell_2$. We revisit this idea and investigate how over 15 years of solver advances have impacted the effectiveness of this technique.

Our second contribution is motivated by recent work on ER in the context of SAT-based model checking [32]. The key insight in that work is to reencode *clauses that are learned* during the execution of the PDR model checking algorithm, thereby compressing the proofs. This reencoding is done by identifying AND and XOR gates, resulting in the algorithm PdrER. Building on their insight, we introduce CaDiCaL-FX, an extension of bounded variable addition (BVA) [33] with advanced XOR and ITE gate detection. Unlike BVA, CaDiCaL-FX crucially applies gate detection to *learned* clauses as well (Sect. 3). Together, the ideas of reencoding clauses, and critically doing so on all clauses, yield significant improvements in solver performance on hard SAT instances. Moreover, when the technique is not effective, using this approach does not harm the overall performance of the SAT solver.

In analyzing the proofs produced by CaDiCaL-FX, we developed a formalism capable of generating polynomial-length proofs for Tseitin formulas via XOR gate factorization combined with bounded variable elimination [20] in *preprocessing* (Sect. 4). Achieving this requires tuning solver heuristics and a shared bound between these two techniques.

Our experiments (Sect. 5) show that GlucosER's definition-introduction strategy does not benefit dramatically from modern solver improvements in general. However, the overhead remains acceptable, and we observe large performance gains on specific benchmark families known to be challenging for CDCL. Notably, our new tool CaDiCaL-ER is the only tool able to solve many of the hardest mutilated-chessboard instances, suggesting that GlucosER's strategy does genuinely benefit from modern solving techniques, beyond what CaDiCaL's general improvements alone would explain. Our gate factorization technique, CaDiCaL-FX, achieves comparable gains across most of the same benchmark families, but crucially, has little-to-no overhead in the general case, making it a practical drop-in enhancement for modern SAT solvers and we plan to include it in the next CaDiCaL release.

2 CaDiCaL-ER, CaDiCaL with Extended Resolution

Extended resolution (ER) [41] expands the resolution proof system with the ability to add definitions of the form $z \leftrightarrow \ell_1 \wedge \ell_2$ for a fresh variable z. Other extension rules with different gate definitions are equivalent; for example, GlucosER and CaDiCaL-ER use $z \leftrightarrow \ell_1 \vee \ell_2$. While a simple addition, introducing new variables makes the proof system much more powerful. There are many examples of problems that have exponential lowerbounds on resolution proofs, but polynomial proofs with ER [14, 17, 22]. Intuitively, the definitions allow the proofs to be greatly reduced by removing the need for "duplicate" steps. However, despite ER's theoretical benefits, using definitions effectively remains an open challenge.

The first CDCL SAT solver to use ER was GlucosER [2]. To explore how its strategy would work in a modern SAT solver, specifically given the 15 years of development of additional solving techniques, we reimplemented GlucosER's definition learning strategy in CaDiCaL-ER, extending CaDiCaL [8] . Whenever two consecutive clauses of the form $\overline{\ell_1} \vee C$ and $\overline{\ell_2} \vee C$ are learned, it introduces the definition $z \leftrightarrow \ell_1 \vee \ell_2$ Future learned clauses of the form $\ell_1 \vee \ell_2 \vee C$ are replaced by $z \vee C$. This technique is integrated into conflict analysis during CDCL as a recursive post-processing step for learned clauses [2].

To make the approach practical, we must carefully manage the database of definitions. If too many definitions are deleted or too few are added, then the solver does not benefit from this technique. However, if too many definitions are added or too few are deleted, the solver becomes overwhelmed by both useless definitions whose propogations do not result in conflicts and long propagation chains of definitions involving other definitions. While the overhead introduced by this technique can be large, especially if the definition database is not tuned properly, this strategy is very effective on certain benchmark families. Surprisingly, despite introducing only disjunction definitions, the technique most consistently exhibits improvements on formulas with many XORs.

While the strategy in CaDiCaL-ER and GlucosER is fundamentally the same, the implementations crucially differ in how they manage the clause databases. Specifically, we utilize CaDiCaL's bounded variable elimination (BVE) [20] to remove added definitions. If irredundant clauses [27] do not contain z often enough, the solver can immediately eliminate the extension variable z by identifying the definition $z \leftrightarrow \ell_1 \vee \ell_2$, encoded as $\overline{z} \vee \ell_1 \vee \ell_2$, $z \vee \overline{\ell_1}$, and $z \vee \overline{\ell_2}$. In order to prevent the immediate deletion of extension variables, CaDiCaL-ER freezes them, preventing them from being considered for elimination. Then, when the clause database is reduced, CaDiCaL-ER evaluates which variables need to be deleted based on their VSIDS scores and unfreezes them. At this point, the unmodified BVE removes variables and clauses during regularly scheduled elimination rounds. Finally, after each elimination round, CaDiCaL-ER cleans the database of substitutions, clearing eliminated definitions that can no longer be used. It also must do bookkeeping after inprocessing rounds that modify the representation of variables, such as equivalent literal substitution.

3 Advanced Gate Factorization

Viewing the CNF formula as logical gates is a very powerful approach. Gates are often encoded using Tseitin transformation [41]. By reverse-engineering the encoding from the CNF, one can reason on a higher abstraction level. For example, clausal congruence closure [9], partially responsible for Kissat winning the SAT Competition 2024 [25], is inseparable from this view. Similarly, bounded variable addition (BVA) [33] can be modeled as factorization of logical AND gates by introducing new variables.

▶ **Example 1.** We can factor the cube $(a \wedge d)$ from the CNF $(a \vee b \vee c) \wedge (b \vee c \vee d)$ into the NNF $(a \wedge d) \vee (b \vee c)$. In order to transform it back into CNF we introduce the definition $x \leftrightarrow a \wedge d$ and substitute, which yields $(\overline{x} \vee a) \wedge (\overline{x} \vee d) \wedge (x \vee b \vee c)$. The definition clause $(x \vee \overline{a} \vee \overline{d})$ is redundant (blocked) and can be omitted.

We extend BVA by XOR and ITE (*if-then-else*) gate detection in CaDiCaL-FX, similar to the work on PdrER [32], which reformulates the CNF based on AND, XOR, and HA (half adder) gates. Contrary to the original BVA approach, we apply factorization as *inprocessing* to the entire clause database during the search, including learned clauses. As it turns out, doing so is crucial for solving some XOR-heavy instance families. In fact, it is stronger then

ExtractPairs (occurrence lists *occs*, global *marks* and *counts*, literal a, vectors *vars* and *pairs*)

```
1  for all clauses C ∈ occs(a)
2      mark literals in C using marks
3      for all clauses D ∈ occs(ā)
4          if size of D is different from C continue
5          if number of unmarked literals in D is not two continue   // a is unmarked
6          Let ℓ be the other unmarked literal in D
7          increase count of abs(ℓ)                                  // phase-oblivious counts
8          record abs(ℓ) in vars
9          record tuple (C, D, ℓ, 0) in pairs                        // forth entry for later
10     unmark literals in C using marks
```

Algorithm 1 At first, we extract all possible pairs of clauses matching our pattern from the occurrence lists of our target literal a with $\overline{a}$. Since the main logic potentially requires matching all clauses from a with all clauses from $\overline{a}$ we cannot avoid a quadratic behaviour. However, we can use various filters to speed up matching and abort early.

simply reverse-engineering Tseitin transformations and does not require these structures to be present in the original formula.

The key to advanced gate factorization is to observe changes to the CNF representation of gates in a single variable elimination step [19]. Doing such an **elimination step in reverse** corresponds to *factoring out* a common pattern from the CNF (cf. Example 2). While reverting a variable elimination step is challenging to automate when generating proofs, we can do it with gates because resolving two gate clauses on the output variable always results in a tautology [20]. Introducing a definition for a gate boils down to blocked clause addition, a generalization of extended resolution [29], compatible with incremental reasoning [21]. We can easily generate proofs for these blocked clauses in the DRAT and LRAT formats [18]. Then, we can use resolution and clause deletion to get the desired CNF transformation.

▶ **Example 2.** Let us assume that the CNF formula containes two XOR gates $x = a \oplus b$ and $x = c \oplus d$ with clausal encoding $(\overline{x} \vee a \vee b), (\overline{x} \vee \overline{a} \vee \overline{b}), (x \vee \overline{a} \vee b), (x \vee a \vee \overline{b})$ and $(\overline{x} \vee c \vee d), (\overline{x} \vee \overline{c} \vee \overline{d}), (x \vee \overline{c} \vee d), (x \vee c \vee \overline{d})$. Eliminating x (*merge*, cf. Fig. 4) results in $(a \vee b \vee \overline{c} \vee d), (\overline{a} \vee \overline{b} \vee \overline{c} \vee d), (a \vee b \vee c \vee \overline{d}), (\overline{a} \vee \overline{b} \vee c \vee \overline{d}), (\overline{a} \vee b \vee \overline{c} \vee \overline{d}), (a \vee \overline{b} \vee \overline{c} \vee \overline{d}), (\overline{a} \vee b \vee c \vee d), (a \vee \overline{b} \vee c \vee d)$. XOR factorization (*split*, cf. Fig. 4) reverts this by grouping consecutive clauses into pairs (with the same polarities on $c \vee d$), introducing $x = a \oplus b$ with ER and substituting this definition into the clause pairs.

PrefilterCandidates (vector *vars*, global *counts*)

```
1  for all vars first ∈ vars
2      if counts of first smaller than threshold   // threshold = factorbound + 5
3      then first is not a candidate
4      else add first to prefiltered-var-candidates
5  reset all counts    // for reuse in Alg 3
```

Algorithm 2 Since we are using the number of clauses as a heuristic, we define a threshold to rule out candidate variables with an insufficient number of pairs before the precise counting and grouping. We use the total number of pairs with a specific variable generated from Alg. 1 in *counts* as an overapproximation for the best match involving this variable.

```
    GroupPairs (literal a, vectors PrefilterCandidates and pairs, global marks and counts)
 1  Let countmax, literals best1 and best2 be initialized to zero
 2  for all variables first ∈ PrefilterCandidates
 3      for all C, D, ℓ, other ∈ pairs   // other is zero initialized from before
 4          if abs(ℓ) not equal first continue
 5          mark literals in D using marks
 6          Set other to the other unmarked literal in C         // different from a
 7          unmark literals in D using marks
 8          add abs(other) to candidates
 9          if ℓ is positive then increase counts(other)         // ℓ and other or ℓ̄ and other̄
10          else increase counts(other̄)                          // ℓ̄ and other or ℓ and other̄
11      for all literals other ∈ candidates
            // optional random tiebreaker if either count is equal to countmax
12          if counts(other) is greater countmax
13          then countmax = counts(other), best1 = first, best2 = other
14          if counts(other̄) is greater countmax
15          then countmax = counts(other̄), best1 = first, best2 = other̄
16          in any case, reset counts of other and other̄
    // the extracted pattern corresponds to an XOR if best1 equals best2̄, otherwise an ITE
17  remove C, D, ℓ, other ∈ pairs where ℓ, other match neither best1, best2 nor best1̄, best2̄
```

Algorithm 3 We are grouping pairs based on which gate definition they belong to, count the size of each group and keep track of the largest group, which will generate the best factorization. We have delayed identifying the second branch of the ITE gate for recorded pairs, which we now need to extract from the clauses. In general, ITEs are phase sensitive, meaning that the clausal encodings for $x = \text{ITE}(c, t, f)$ and $x = \text{ITE}(c, t, \overline{f})$ are not equal, but for $x = \text{ITE}(c, t, f)$ and $\overline{x} = \text{ITE}(c, \overline{t}, \overline{f})$ they are. We account for this when generating groups and identify the best group for each variable.

Since CaDiCaL-FX is an extension of BVA, we can apply this algorithm in the same context, using the same data structures. In the implementation, the solver state is transformed from watches [35] to full occurrence lists occs, i.e., every literal is mapped to a list of all clauses containing said literal (as opposed to just two per clause). This structure allows us to restrict our pair extraction algorithm to the occurrence lists of a single variable a, i.e., clauses in $\text{occs}(a)$ and $\text{occs}(\overline{a})$. Doing this for all variables will find every possible factorization. Unlike the phase-sensitive BVA, it does not matter whether we consider a or $\overline{a}$, so we add a filter to skip a literal when its negation has already been scheduled.

Pairing up clauses for XOR gates can be reduced to matching a clause from $\text{occs}(a)$ with a clause from $\text{occs}(\overline{a})$ with a common prefix C, as well as an additional literal b with clashing polarity. For ITE gates, we are looking for the same prefix, but the additional literal b is not restricted to clashing polarity, but can be two different literals instead. Note that we can always handle XOR as a special case of $\text{ITE}(c, t, f)$ where $t = \overline{f}$. Combining the extraction for these two gates leads to the following multi-phase algorithm for a variable a:

Extraction of all possible pairs (Alg. 1): Fixing a clause C in $\text{occs}(a)$, search for all clauses D in $\text{occs}(\overline{a})$ that differ by one literal in addition to a. This is efficiently done using marks, a global Boolean flag for each literal, which allows marking literals of C. Finding the corresponding pair then boils down to iterating all clauses in $\text{occs}(\overline{a})$ and counting differences (i.e. unmarked literals). This does not require sorting beforehand. Clauses of different

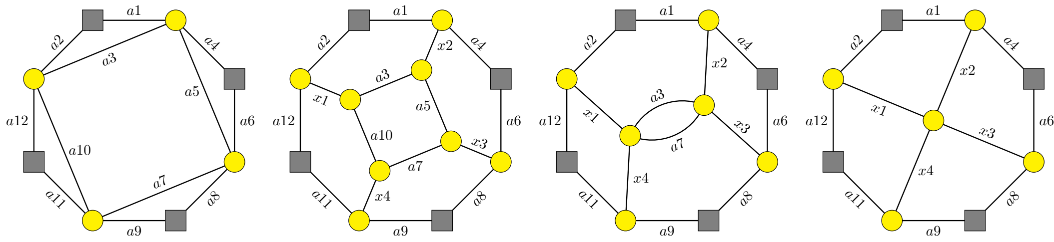


Figure 4 The plot visualizes the elimination of a *four-cycle* in a Tseitin formula (cf. Tab. 5) from left to right in three steps (*split*, *merge*, *collapse*). Each yellow node corresponds to an XOR on the variables of incident edge labels (the grey nodes are not touched by the procedure). First, all *degree-four* nodes in the cycle are *split* inward with XOR factorization, introducing the new variables $x1$, $x2$, $x3$ and $x4$. Then, neighboring *degree-three* nodes in the inner cycle are *merged* by variable elimination on $a5$ and $a10$, contracting the cycle. Finally, the last two *degree-four* nodes forming a *two-cycle* with two shared edges are *collapsed* by eliminating either $a3$ or $a7$, resulting in a reduction in clauses and removal of the cycle. On larger cycles, it will take additional split and merge iterations before collapsing the cycle.

sizes are skipped as they can never be pairs (Lines 1–5). We also extract some additional information: For all pairs C, D, count the variables that are different in D using a global counter and record them separately (Lines 8 and 9). The counts are used (and reset) when prefiltering (Alg. 2) and the pairs are required for grouping (Alg. 3).

Grouping of pairs (Alg. 3): Our goal is to apply the best factorization to the current clause set with respect to formula size. We want to identify the two literals t, f with the most pairs, which will correspond to the gate $\text{ITE}(a, t, f)$ (or an XOR gate if $t = \overline{f}$). These can be extracted from the additional information we obtained previously by iterating through each candidate variable and counting the number of pairs it occurs in (Lines 2–10). We can group pairs with the same literals in opposite polarities, as these belong to the same gate (i.e. t, f with $\overline{t}, \overline{f}$, but not $\overline{t}, f$ or $t, \overline{f}$, Lines 9 and 10). Normalizing for f, we can generate a phase-sensitive count for all possible t and record the best pair t, f. Doing this for all f yields the globally best candidate pair (Lines 11–16). Tiebreaking for these pairs is relevant. While deterministic tiebreaking seems to work better in general, a random tiebreaker is necessary for CaDiCaL-PP (*cf.* Sect. 4). After extracting the best candidate pair we finalize our set of clauses and mark them to avoid duplicates. If some clause D appears in the formula twice, they can be matched with the same clause C. This is usually avoided through subsumption, but can sometimes happen depending on the inprocessing schedule or specific options.

Factoring and proof: First, we introduce the four gate clauses using a fresh variable x. Then, for every pair $C \vee c \vee \overline{t}, C \vee \overline{c} \vee \overline{f}$ we use the two clauses containing $\overline{x}$ to resolve to a single clause $C \vee \overline{x}$. We do the same for the pattern $D \vee c \vee t, D \vee \overline{c} \vee f$, but using the two clauses containing x instead. Finally, we can add the new clauses (as irredundant clauses [27]) and delete all clause pairs from our database. The derivation produces some intermediate clauses that need to be recorded for the DRAT (or LRAT) proof, but they can be deleted immediately, without learning them in the solver.

4 Polynomial Extended Resolution Proofs of Tseitin Formulas

Tseitin formulas [41] are defined as a conjunction of clausal XOR constraints on edge-labeled graphs. Each node corresponds to one such constraint on the edge-label incidence set. When the parity modulo two across all constraints is equal to one, they are unsatisfiable and often

hard for resolution, especially for certain graph families, such as expander graphs [42]. Gate factorization with XORs enables polynomial-sized proofs on Tseitin formulas.

Using a series of *split*, *merge*, and *collapse* operations on the underlying graph, one can transform it into a single-node graph by removing *cycles* (Fig. 4). These operations can be captured by XOR gate factorization and elimination. The single-node graph corresponds to an empty clause (or formula for satisfiable instances). In general, one can always transform a graph into a single node without generating nodes of degree five or higher. This polynomial procedure yields a polynomial extended resolution proof of the underlying graph (as the size of the ER-proof for a merge or split operation depends on the size of the XORs).

Nodes of degree two or less can always be merged without increasing the degree of the other nodes, whereas degree three nodes can be merged into a degree four node, and a degree four node can be split into two degree three nodes. These operations allow contracting any cycle in $\mathcal{O}(d \log d)$ steps until two degree four nodes share two edges, which can then be collapsed. The contracting will result in binary trees of depth $\mathcal{O}(\log d)$ with a total of $\mathcal{O}(d \log d)$ nodes with degree three. The structure can be reverted to get a path with d nodes.

This allows reducing arbitrary graphs to a single node in at most $\mathcal{O}(n^2 \log n)$ steps by picking any cycle in the graph and transforming it into a path with one less edge but the same number of nodes in $\mathcal{O}(n \log n)$ steps. We created a new family of benchmarks that we call *Tseitin Donut* (*cf.* Fig. 5). They are nearly identical to the Tseitin Grid formulas; however, instead of a grid, we connect the borders of the grid to get a torus. On these grids, we can transform a four-cycle without changing the structure to get many long cycles of degree two nodes that we can simply eliminate. This proof only takes $\mathcal{O}(n)$ steps.

While these proofs can be obtained by our technique, the solver requires a reduction of clauses after factoring (to make sure BVE will not undo it) but merging size three XORs as well as splitting size four XORs does not reduce the number of clauses in the CNF. Therefore, the technique does not help solving these instances if restricted to irredundant clauses (as for original BVA). Only when factoring irredundant and redundant clauses together, can we solve these instances effectively. We believe that learned clauses steer the solver towards helpful factorizations, thereby compressing the proof. We leave it to future work to explore theoretical reasons for the empirical success of this approach, presented in the next section.

By changing the solver heuristics through settings (CaDiCaL-PP in Sect. 5), one can split and merge these nodes. However, to reduce cycles we also need to add random tiebreaking for factoring, which in principle allows us to solve all Tseitin instances with preprocessing. Progress can easily be seen from the number of clauses, which is only reduced when successfully merging degree-four nodes (or collapsing degree-two nodes), while the number of variables jumps back and forth. Interestingly, the resulting proofs scale roughly quadratically in the number of clauses, independent of the underlying graph structure. The runtime is roughly cubic, due to the overhead of extracting XORs from scratch every time.

Not all problems can be solved through preprocesing only. Some Tseitin families scale a lot better with the default heuristics. The Tseitin donuts can be solved up to size 200×200 squares, while the preprocessing-only version struggles starting from size 40×40.

5 Experiments

The experiments were run on Bridges-2 at the Pittsburgh Supercomputing Center [11] with AMD EPYC 7742 CPUs, 5000 s time limit and 256 GB of memory shared among 64 solvers.

After analyzing SAT competition instances where our techniques succeeded, we present families (Tab. 5) that are either interesting from a proof complexity standpoint, e.g., contain

Table 5 Features of selected benchmark families ([∗] this paper). We consider the anniversary track from the SAT Competition 2022 as well as hard combinatorial instances (Mutilated Chessboard and Matching of Properly Intersecting Intervals), XOR-heavy instances motivated by hardware verification (Crux- and XOR Miter) and a wide range of Tseitin formulas with different characteristics.

	Name	Source	Type	Graph structure	Difficulty
(AT)	Anniversary Track	[6]	Composite	n.a.	wide range
(MC)	Mutilated Chessboard	[24]	Parity argument	n.a.	hard
(MI)	Matching Intervals	[4]	Matching problem	n.a.	hard
(CM)	Cruxmiter	[28]	Multiplier	n.a.	medium
(XM)	XOR Miter	[∗], [16]	Tseitin/Miter	DAG	easy
(TS)	Tseitin SC	[26]	Tseitin Formula	Different Families	easy
(UL)	Urquhart (Li)	[31]	Tseitin Formula	Expander Graph	hard
(US)	Urquhart (Simon)	[15]	Tseitin Formula	Random Graph	hard
(RT)	d-Regular Tseitin	[30]	Tseitin Formula	Random Graph	medium
(TD)	Tseitin Donut	[∗]	Tseitin Formula	Torus	easy
(TC)	Tseitin Cube	[∗]	Tseitin Formula	Hyper Torus	hard

many XORs, or for which our techniques are especially effective. We compare the following solvers: CaDiCaL, GlucosER, CaDiCaL-ER (Sect. 2), CaDiCaL-FX (Sect. 3) and CaDiCaL-PP (Sect. 4), which results from CaDiCaL-FX with the following options:

- -P10000: Runs 10000 rounds of preprocessing, effectively disables search.
- --factorbound=0 --elimboundmax=0: Limits clauses that factorization and elimination are allowed to add to the formula to zero, consistently enables splitting XORs of size four and merging of XORs of size three.
- --factorxorrand=1: Enables random tiebreaking during factorization that is necessary for breaking cycles in Tseitin graphs.
- --factorthresh=0 --factordelay=0: Disables heuristics that detect and avoid factorization taking up too much of the total running time.

Anniversary Track (AT). A great benefit of CaDiCaL-FX is that it improves performance on many families of formulas with little-to-no overhead cost. This is contrasted with CaDiCaL-ER and GlucosER, which experience an unavoidable overhead from learning ER definitions on-the-fly, rather than in controlled inprocessing phases. Many definitions are not particularly valuable and add cost during propagation. In Fig. 6, there is a bias upward, suggesting most formulas do not benefit from CaDiCaL-ER and the overhead is, on average, around a factor of 1.5×. Figure 8 shows that CaDiCaL-FX induces very little overhead on formulas where it is not useful, making it a drop-in upgrade with little downside.

Mutilated Chessboard (MC). The Mutilated Chessboard problem is a classic, difficult problem for automated reasoning tools [24]. The problem asks whether it is possible to use 2×1 dominoes to tile an $n \times n$ chessboard with its top-left and bottom-right corners removed. While there is a well-known parity-based refutation, automated reasoning tools struggle to uncover the abstractions required for such an argument. In fact, it is known that the shortest resolution proofs are exponential in length [1]. As can be seen in Tab. 7, CaDiCaL-ER demonstrates significantly improved performance on (MC) compared to any of the others.

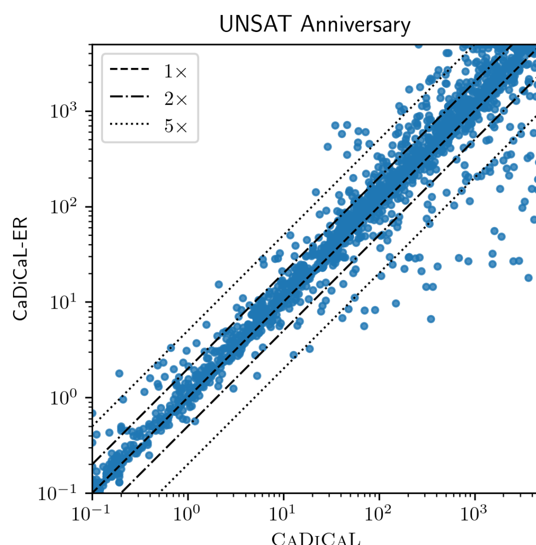


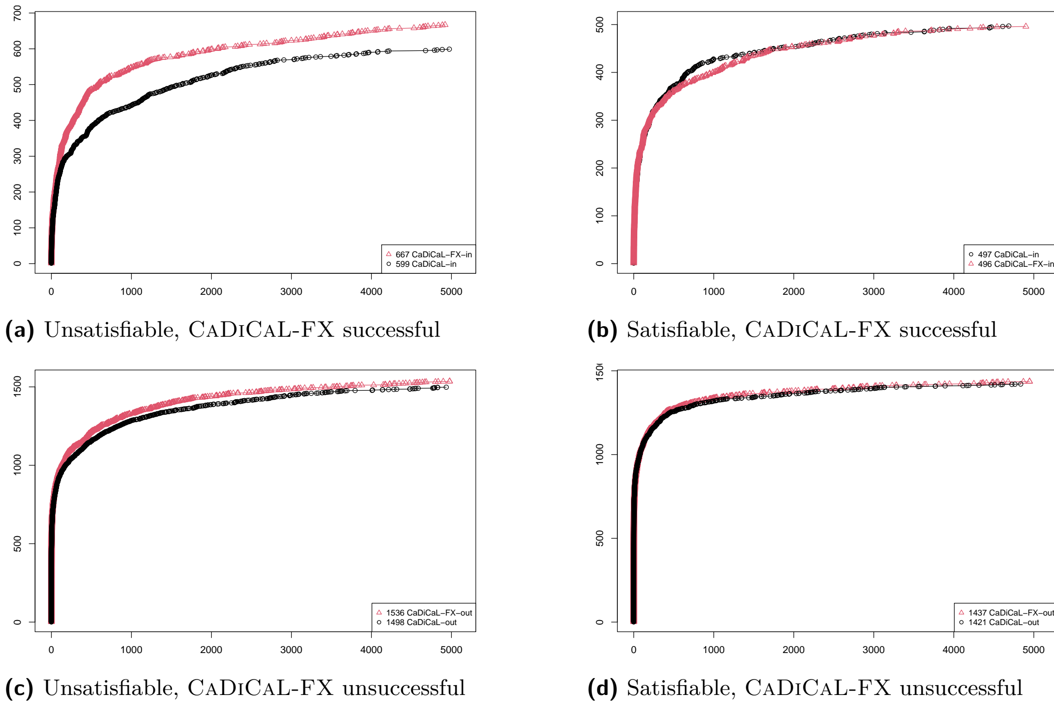
Figure 6 We show a comparison of CaDiCaL-ER (par2 1907224) with CaDiCaL (par2 2067776) on the unsatisfiable instances of the anniversary track (AT). While certain families benefit from the ER technique, the majority incur overhead between 1 and 2× with some families having more than that. However, far more families are improved significantly than are hurt significantly.

Matching of Properly Intersecting Intervals (MI). Buss et al. [13] have observed that ER techniques are effective on what we are calling "Intervals" formulas [4]. The problem is from a competitive programming competition: you are given a sequence $a_0, \ldots, a_{2n}$, initialized as 0s, and a sequence of operations $\{(l_i, r_i)\}_i$ with $0 \leq l_i < r_i \leq 2n$ and $1 \leq i \leq n$. The operation (l_i, r_i) assigns value i to all a_j with $l_i \leq j < r_i$. The goal is to perform each operation exactly once, in any order, to maximize the number of a_i such that $a_i \neq a_{i+1}$. To encode this as a boolean formula, the problem is reduced to a bipartite maximum matching problem. The problem is then made UNSAT by removing all, but one, right endpoint. A more detailed description can be found in the proceedings of the 2023 SAT Competition [4].

Much like the Mutilated Chessboard problem (MC), neither CaDiCaL nor CaDiCaL-FX are able to solve any instances (Tab. 12). However, GlucosER is able to solver more instances than CaDiCaL-ER. We do not know why GlucosER excels on these formulas, but interestingly, even when CaDiCaL is configured to have settings to mirror GlucosER to the best of our ability, primarily via the removal kjof all pre- and inprocessing techniques (keeping only BVE which is necessary for our ER procedure), CaDiCaL-ER does not solve any instance. We leave investigating this as future work.

Table 7 Runtime in seconds on (MC) family. They challenge all solvers except CaDiCaL-ER. This demonstrates that for certain families of formulas, the improvement observed in CaDiCaL-ER over GlucosER is not simply due to general improvements in CaDiCaL.

n	CaDiCaL	CaDiCaL-FX	GlucosER	CaDiCaL-ER
17	2616	3369	901	90
18	1485	815	3442	143
19	> 5000	4069	> 5000	203
20	> 5000	> 5000	> 5000	245



(a) Unsatisfiable, CaDiCaL-FX successful

(b) Satisfiable, CaDiCaL-FX successful

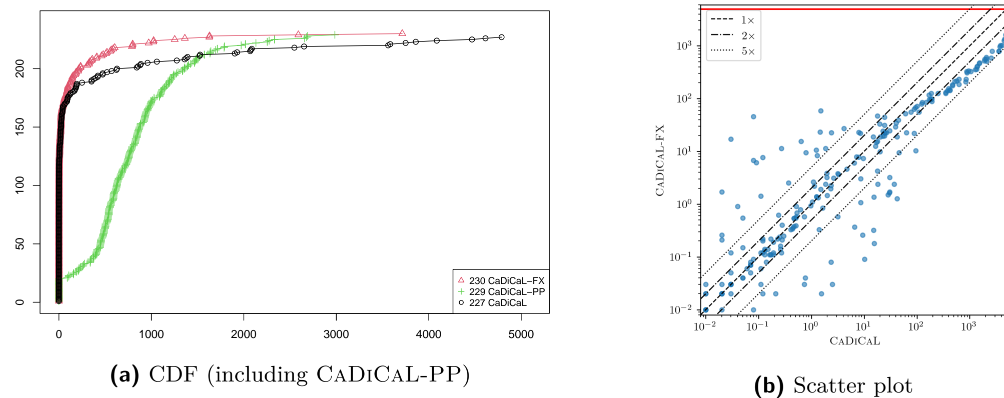
(c) Unsatisfiable, CaDiCaL-FX unsuccessful

(d) Satisfiable, CaDiCaL-FX unsuccessful

Figure 8 We split the instances from (AT) into two sets, where CaDiCaL-FX successfully extracted XORs or ITEs (name-in, on top), and where it did not (name-out, at the bottom). The plots show a significant improvement for CaDiCaL-FX on unsatisfiable instances while not degrading performance on satisfiable instances nor instances where it was never successful. The combined par2 scores of CaDiCaL-FX is 13857999 vs. 15086406 on CaDiCaL.

Cruxmiter (CM). The "crux" of multiplier verification is conjectured to be the parity reasoning needed to handle differently ordered adders [28]. The hope with our ER approaches is that it can identify and factor common XOR structures among those adders. We do observe a speedup with CaDiCaL-FX compared to CaDiCaL that scales with difficulty (Fig. 9b) while CaDiCaL-PP does not speed up solving (Fig. 9a), even when only counting search (CDCL) time without any pre- nor inprocessing (*cf.* 10).

XOR Miter (XM). The XOR miters (or rpar [16]) are simple instances that are still challenging for CDCL. They compute the parity of n input bits with linear XOR chains in two different orders and compare them through a miter construction, i.e., an XOR on the output. Since they are encoded with XORs only, they can be seen as a Tseitin formula on a DAG-like structure. When using the same input order on both sides of the miter, the instances are solved by clausal congruence closure [9]. When randomizing the order, they tend to be hard for resolution. Surprisingly, instances with *reversed* order are also solved instantly by CaDiCaL, though not through congruence closure. Instead, bounded variable elimination was able to reduce the graph structure to a single node using only *merge* operations that do not increase size (*cf.* Sect. 4). This observation was partially responsible for the development of the polynomial proofs and CaDiCaL-PP. We built our own generator [7] for these different versions. Since the instances scale only linearly with the input size CaDiCaL-PP can solve them almost instantly up to $n = 200$ (*cf.* Fig 15b).



(a) CDF (including CaDiCaL-PP)

(b) Scatter plot

Figure 9 CDF and scatter plot of the (CM) family with CaDiCaL versus CaDiCaL-FX (below the diagonal is better). We observe an increasing speedup which scales with the difficulty of the instance. The CaDiCaL-PP configuration is not able to improve on CaDiCaL-FX, as the richer Boolean structure of the benchmark family requires some search to prove UNSAT. The par2 scores on (CM) are 29666 (CaDiCaL-FX), 101248 (CaDiCaL) and 191132 (CaDiCaL-PP).

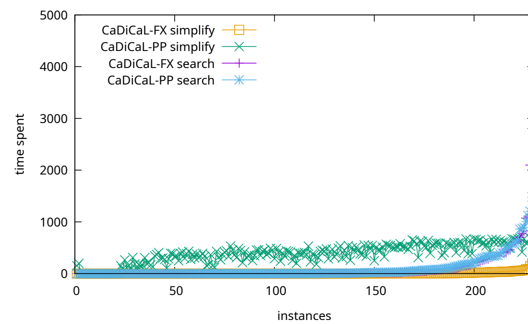
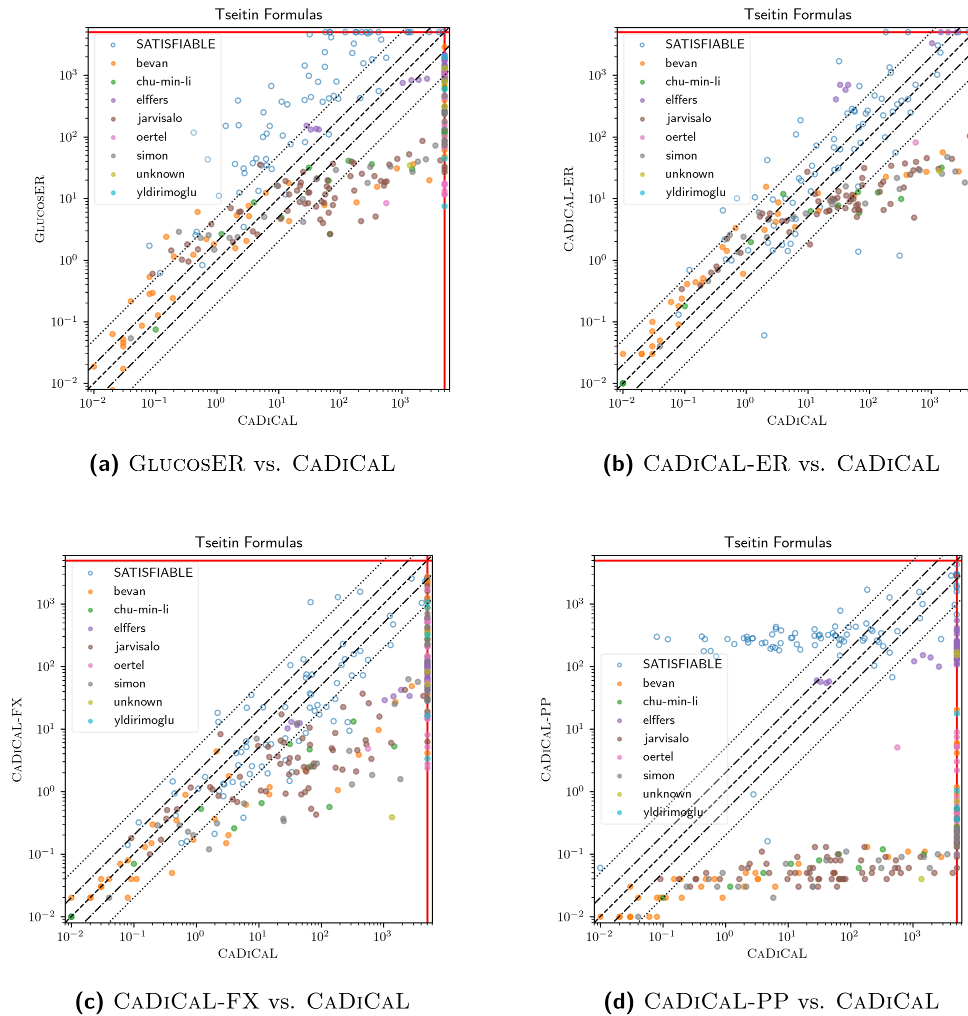


Figure 10 Isolating the *search* and *simplify* (pre- and inprocess) times for CaDiCaL-FX and CaDiCaL-PP on the (CM) shows that preprocessing does not help on this familiy, but add unnecessary overhead.

SAT Competition Tseitin Formula Family (TS). We consider the “SAT Competition Tseitin Formula” family as defined by the Global Benchmark Database [26]. While all of these families share the parity constraint essential to Tseitin formulas, the exact graph structure differs among different benchmarks. We see a general improvement on Tseitin formulas as expected and it is interesting to note that different benchmark families respond differently to the various techniques. The results are shown in Fig. 11.

For satisfiable formulas, no technique demonstrates a clear improvement. CaDiCaL-PP demonstrates a performance degradation on easier formulas due to the cost of preprocessing. The unsatisfiable formulas, on the other hand, tell a more interesting story. GlucosER (Fig. 11a) and CaDiCaL-ER (Fig. 11b) demonstrate a comparable speedup on the same families, except for those submitted by Elffers [5] (purple) which consist of long and narrow “Tseitin Donuts” as described in the next section. CaDiCaL-FX (Fig. 11c) improves performance dramatically as compared to CaDiCaL-ER or GlucosER across the board. CaDiCaL-PP (Fig. 11d), though, is the most interesting. Nearly every formula that CaDiCaL is able to solve is solved almost instantly by CaDiCaL-PP. On those where CaDiCaL times out,



(a) GlucosER vs. CaDiCaL

(b) CaDiCaL-ER vs. CaDiCaL

(c) CaDiCaL-FX vs. CaDiCaL

(d) CaDiCaL-PP vs. CaDiCaL

Figure 11 Comparing different ER techniques on the SAT Competition Tseitin Formulas (TS), split by author (below the diagonal is better). On unsatisfiable instances, the base solver CaDiCaL (par2 1293447) is consistently outperformed by the other solvers using ER. We observe performance improvements across the board, with CaDiCaL-PP (par2 248424) being the clear winner, able to instantly solve most formulas solved by CaDiCaL. On satisfiable instances, GlucosER (par2 584198) and CaDiCaL-PP demonstrate a performance degradation, while CaDiCaL-ER (par2 698112) and CaDiCaL-FX (par2 266626) are noisy with no clear trend.

CaDiCaL-PP still demonstrates a dramatic improvement.

Expander Graphs (UL) and (US). The “Urquhart Problems” first introduced by Urquhart [42] are Tseitin problems based on expander graphs. Two families of benchmarks, developed by Chatalic and Simon [15], as well as Li [31], are known to be difficult for current CDCL solvers, but are less of a challenge for other automated reasoning methods [12]. Indeed, it is known that resolution proofs require exponential length for these problems [42]. Since Simon’s graphs are randomized and are expanders with high probability, we generated 5 graphs of each size parameter. Li’s graphs, on the other hand, are “true” expanders and considered to be the harder of the two. The data, as shown in Tab. 12, reflects exactly

Table 12 Solved instances on (MI) and (UL) families. The (US) family is specified by a size parameter as well as five seeds per size. We show the maximum size solved and the number of seeds solved for this size. While CaDiCaL-FX (and CaDiCaL-PP) fall behind on (MI), CaDiCaL-FX has comparable performance to GlucosER and CaDiCaL-ER on (UL) with a slight edge on (US). The CaDiCaL-PP configuration is the clear winner on both families. Note that CaDiCaL-PP also solved only 4/5 of the $m = 27$ seeds.

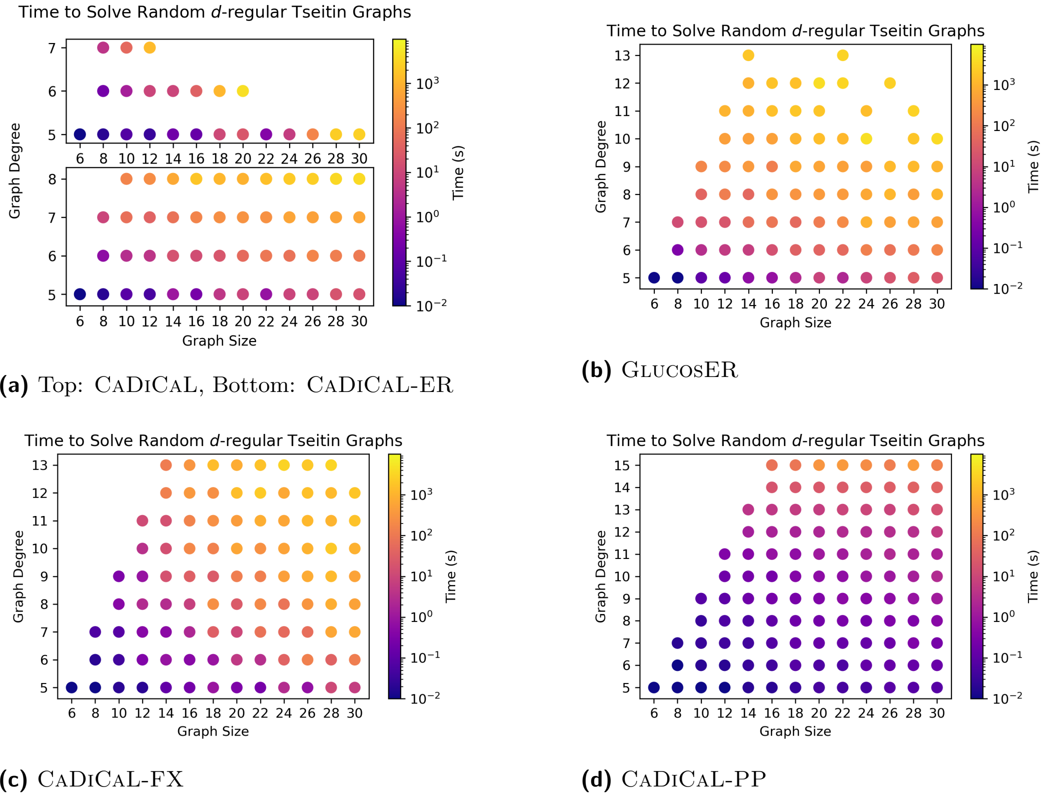
	Solved Instances (within a time limit of 5000 s)				
	CaDiCaL	CaDiCaL-FX	CaDiCaL-PP	GlucosER	CaDiCaL-ER
(MI)	0	0	n.a.	14	9
(UL)	0	5	18	6	4

	Max Size Parameter Solvable (seeds solved)				
(US)	4 (4/5)	8 (2/5)	28 (1/5)	7 (3/5)	7 (4/5)

the story we expect. CaDiCaL is unable to solve many instances, and the addition of ER reasoning shows marginal improvement, but ultimately CaDiCaL-PP is far more successful on these benchmarks. This makes sense, as it is essentially a dedicated solving mechanism independent of the resolution of CDCL. A similar story can be seen for the Simon graphs: extended resolution demonstrates marginal improvements, but only the dedicated XOR-factoring procedure is able to make a significant dent in the benchmark.

d-Regular Tseitin (RT). One of the simplest forms of Tseitin problems are random ones on d-regular graphs. For this benchmark, we use those provided by the CNFgen tool [30]. Specifically, we generated one random graph for each n and d for even n with $6 \leq n \leq 30$ and $5 \leq d < n$. The results are in Fig. 13. It is unsurprising that CaDiCaL struggles on these problems and that the addition of extended resolution via our various tools helps, not unlike the Urquhart instances. However, what is interesting is how the different tools exhibit different scaling properties. CaDiCaL struggles in scaling with both the graph size (x-axis) as well as the graph degree (y-axis). CaDiCaL-ER remedies this a bit and has much less of an issue scaling the graph size. However, it struggles to scale the graph degree, though it is able to consistently do so up to degree eight. GlucosER is able to push the frontier of solved graph degrees a bit higher, but lacks robustness. It is the only tool with “gaps” in the difficulties of problems it is able to solve e.g. it can solve a 10-regular graph on 28 vertices, but not on 26. CaDiCaL-FX, while across the board better than CaDiCaL-ER and GlucosER, scales with graph size proportionally more than CaDiCaL-ER does. That is, the impact of the increasing graph size is more noticeable in CaDiCaL-FX than CaDiCaL-ER. Finally, CaDiCaL-PP is able to take the best of both worlds. It has no issue whatsoever scaling to larger graphs, and is able to solve problems of higher degree than any other tool. It is likely that the reason that CaDiCaL-FX and CaDiCaL-PP cannot go even higher in degree is due to the exponential encoding in the CNFgen tool. For example, nodes of degree 15 are encoded using $2^{14} = 16\,384$ clauses. With that many clauses, it is very expensive for CaDiCaL-FX or CaDiCaL-PP to reencode the formula as the algorithm scales quadratically.

Tseitin Cube (TC) and Tseitin Donut (TD). The donut is a $n \times n$-grid with connections between opposite edges (modeling this in 3D looks like a donut) with $3 \leq n \leq 200$ (198



(a) Top: CaDiCaL, Bottom: CaDiCaL-ER

(b) GlucosER

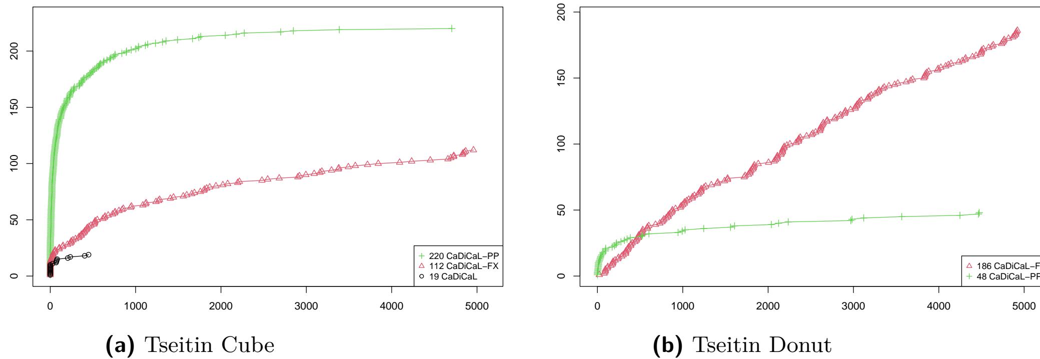
(c) CaDiCaL-FX

(d) CaDiCaL-PP

Figure 13 We generated random Tseitin formulas (RT) from random d-regular graphs on (even) n vertices with $6 \leq n \leq 30$ and $5 \leq d < n$ using the CNFGen [30]. Dots not shown on the graphs were unable to be solved. Missing dots to the left do not exist (there are no d-regular graphs on n vertices for $d \geq n$). Par2 scores are 262123 (CaDiCaL-PP), 471279 (CaDiCaL-FX), 620998 (GlucosER), 932696 (CaDiCaL-ER) and 1172175 (CaDiCaL).

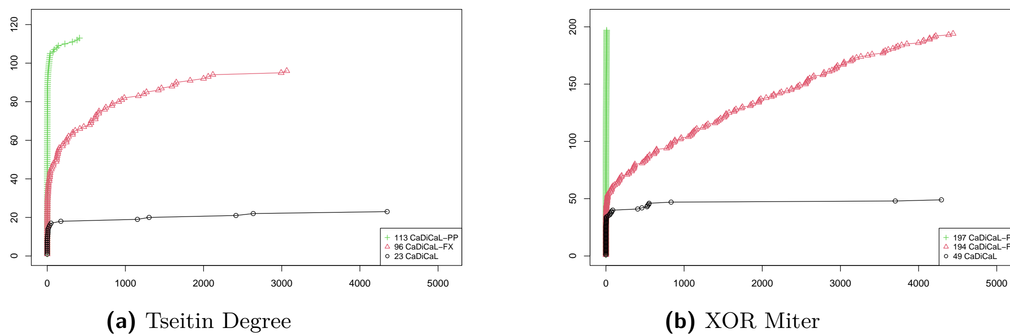
instances). Tseitin Cube is the 3D version of that with three parameters n, m, o (length, width, height) with $2 \leq n \leq m \leq o \leq 11$ (220 instances). The results are in Fig. 14 where we see that preprocess easily solves (TC) but CaDiCaL-FX is much better on (TD).

Preprocessing. Plotting the number of clauses produced by factoring XORs (linearly related to proof size and the number of *splits*, but ommiting *merges*) versus the number of clauses in the original instance shows that CaDiCaL-PP operates independently of the underlying graph structure (Fig. 16), similar to the general algorithm that is sketched in Section 4. The outlier is the (TD) family. These can, in theory, be solved by preprocessing in a linear number of steps. We postulate that this structure is used to some degree in the random approach of CaDiCaL-PP, leading to smaller proof sizes. The (TC) instances are encoded using XORs of size six, which can be reencoded to XORs of size four while halving the number of clauses. We plot the clauses after reencoding as this more closely relates to the difficulty of the instance. Reecoding this way will always happen in the first preprocessing round. The (RT) family is missing on this plot because the instances generally contain a few large XORs with an exponential encoding so the initial reencoding time dominates solving time. We also cannot reliably reencode larger instances in one go with the default restrictions to factor.



(a) Tseitin Cube **(b)** Tseitin Donut

Figure 14 The CaDiCaL-PP solver outperforms CaDiCaL-FX on Tseitin Cubes (TC), while struggling on the comparatively larger but easier donuts (TD). CaDiCaL times out on all but the smallest instances. The par2 scores (TC)/(TD) are: CaDiCaL-PP: 64033/1474075, CaDiCaL-FX: 1241489/459474, CaDiCaL: 2011618/1910000.



(a) Tseitin Degree **(b)** XOR Miter

Figure 15 The left plot shows a comparison of CaDiCaL (par2 1172175) with CaDiCaL-FX (par2 471279) and CaDiCaL-PP (par2 262123) on the Tseitin Degree fromulas generated from CNFgen. CaDiCaL-PP outperforms CaDiCaL-FX and both struggle with the huge encoding of large degree XORs by CNFgen in both cases the runtime of reencoding the instances from the exponential XOR encoding to XORs size four with additional variables dominates the total solving time. The right plot shows the same solvers on random XOR miters from our generator (par2 for (XM) in the same order: 1492235, 278477, 496).

Related Approaches. We looked at tools utilizing the Propagation Redundancy (PR) [23] proof system. PR shares similar theoretical benefits with ER and, in recent years, there has been much work on effectively using PR clauses in practice [39, 40]. One example is CAutiCaL [40] which modifies CaDiCaL by learning PR clauses based on Conditional Autarkies as a preprocessing step. CAutiCaL demonstrated poor performance on Tseitin Formulas, solving none of the (TD) (*cf.* Fig. 14b) or (UL) formulas (*cf.* Tab. 12), and solving only the easiest few of the (TC) (*cf.* Fig. 14a) and (US) formulas (*cf.* Tab. 12). It was similarly unable to solve any of the (MI) instances (*cf.* Tab. 12). While it did demonstrate improved performance on small (MC) (*cf.* Tab. 7) formulas solving the order 18 formula in only 417 seconds compared to CaDiCaL's 1485, it took a similar amount of time to CaDiCaL on the order 17 formula, and was unable to solve any larger instances. Finally, results on Miter instances were largely unchanged, with CAutiCaL solving one more (XM) instance (*cf.* Fig. 15b) than CaDiCaL and the same number of (CM) instances (*cf.* Fig. 9) with no clear performance improvement. This shows that while PR has similar theoretical benefits to ER, in practice, techniques based on them demonstrate performance improvements on

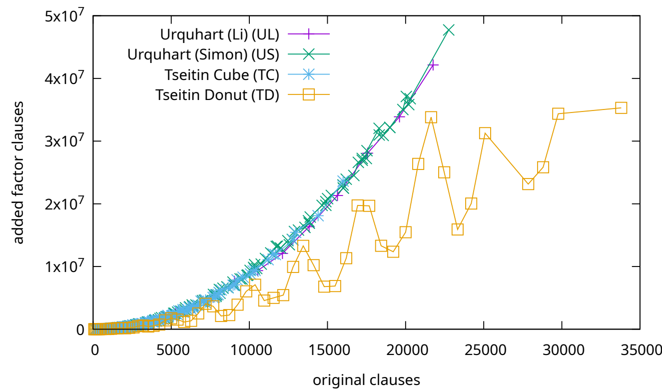


Figure 16 Comparing formula size in clauses to the number of clauses added by factoring, linearly related to the number of *split* operations (cf. Sect. 4). We can observe a clear quadratic dependency in the two, independent from the underlying graph structure. The exception to this are the Tseitin Donuts which have linear proofs in our split-and-merge paradigm.

different sets of benchmarks.

We also investigated a related approach involving a different strategy for learning ER definitions, dubbed *Dual Implication Points* [13] that has only recently come to our attention and has not yet been published. We observed similar results to ours on certain benchmarks, particularly those involving XORs. Notably, it performed quite well on (UL) formulas, solving more formulas than our techniques except for CaDiCaL-PP. On (US) formulas (*cf.* Tab. 12), however, it was able to solve two of the $m = 8$ formulas, but none of the $m = 7$, and only one $m = 6$ formula. It was the only tool to demonstrate such behavior, being unable to solve almost all easier benchmarks than its maximum capability. We were also only able to use it to solve 2 intervals formulas (*cf.* Tab. 12), and none of the Mutilated Chessboard formulas (*cf.* Tab. 7). While it seems to struggle with robustness, the *Dual Implication Point* strategy performs very well on the Urquhart (Li) formulas, arguably the hardest formulas we benchmarked against, and we leave it to future work to determine how to leverage this success on other benchmark families.

6 Conclusion

This paper presents two applications of extended resolution in CDCL SAT solving. The first revisits GlucosER, implemented as CaDiCaL-ER in the modern award-winning SAT Solver CaDiCaL. The second application investigates factorization of learned clauses during inprocessing with two variants. The first interleaves factorization with CDCL search (CaDiCaL-FX), while the other applies factorization during preprocessing (CaDiCaL-PP). All of these techniques perform well on a similar set of instances, namely Tseitin formulas. While there are some non-Tseitin instances where CaDiCaL-ER performs very well, the performance penalty across a wider range of instances makes it hard to justify integrating it into CaDiCaL. However, gate factorization with CaDiCaL-FX performs well enough to be used unconditionally on all instances of the SAT Competition.

As future work, we would like to include CaDiCaL-FX into the next CaDiCaL release and check the performance on incremental benchmarks, where, especially for hardware model checking benchmarks, there could be many XOR gates. We further want to improve our understanding and investigate the impact of ITE gate factoring in general.

References

1. Michael Alekhnovich. Mutilated chessboard problem is exponentially hard for resolution. *Theoretical Computer Science*, 310(1):513–525, 2004. URL: https://www.sciencedirect.com/science/article/pii/S0304397503003955, doi:10.1016/S0304-3975(03)00395-5.
2. Gilles Audemard, George Katsirelos, and Laurent Simon. A restriction of extended resolution for clause learning SAT solvers. *Proceedings of the AAAI Conference on Artificial Intelligence*, 24(1):15–20, Jul. 2010. doi:10.1609/aaai.v24i1.7553.
3. Gilles Audemard and Laurent Simon. On the Glucose SAT solver. *Int. J. Artif. Intell. Tools*, 27(1):1840001:1–1840001:25, 2018. doi:10.1142/S0218213018400018.
4. Tomas Balyo, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2023: Solver, Benchmark and Proof Checker Descriptions*. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2023.
5. Tomáš Balyo, Marijn J. H. Heule, and Matti Juhani Järvisalo. *Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions*. Department of Computer Science Series of Publications B. University of Helsinki, Finland, 2016.
6. Tomas Balyo, {Marijn J.H.} Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2022: Solver and Benchmark Descriptions*. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2022.
7. Armin Biere. Generator a miter between two random parity checking circuits., 2026. URL: https://github.com/arminbiere/genxormiter.
8. Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2024. doi:10.1007/978-3-031-65627-9_7.
9. Armin Biere, Katalin Fazekas, Mathias Fleury, and Nils Froleyks. Clausal congruence closure. In Supratik Chakraborty and Jie-Hong Roland Jiang, editors, *27th International Conference on Theory and Applications of Satisfiability Testing, SAT 2024, August 21-24, 2024, Pune, India*, volume 305 of *LIPIcs*, pages 6:1–6:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.SAT.2024.6.
10. Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021. doi:10.3233/FAIA336.
11. Shawn T. Brown, Paola Buitrago, Edward Hanna, Sergiu Sanielevici, Robin Scibek, and Nicholas A. Nystrom. *Bridges-2: A Platform for Rapidly-Evolving and Data Intensive Research*, pages 1–4. Association for Computing Machinery, New York, NY, USA, 2021.
12. Randal E. Bryant and Marijn J. H. Heule. Generating extended resolution proofs with a BDD-based SAT solver. *ACM Trans. Comput. Log.*, 24(4):31:1–31:28, 2023. doi:10.1145/3595295.
13. Sam Buss, Jonathan Chung, Vijay Ganesh, and Albert Oliveras. Extended resolution clause learning via dual implication points. *CoRR*, abs/2406.14190, 2024. arXiv:2406.14190, doi:10.48550/ARXIV.2406.14190.
14. Sam Buss and Jakob Nordström. Proof complexity and SAT solving. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 233–350. IOS Press, 2021. doi:10.3233/FAIA200990.
15. Philippe Chatalic and Laurent Simon. ZRES: the old Davis-Putman procedure meets ZBDD. In David A. McAllester, editor, *Automated Deduction - CADE-17, 17th International Conference on Automated Deduction, Pittsburgh, PA, USA, June 17-20, 2000, Proceedings*, volume 1831 of *Lecture Notes in Computer Science*, pages 449–454. Springer, 2000. doi:10.1007/10721959_35.

16 Leroy Chew and Marijn J. H. Heule. Sorting parity encodings by reusing variables. In *Theory and Applications of Satisfiability Testing – SAT 2020: 23rd International Conference, Alghero, Italy, July 3–10, 2020, Proceedings*, page 1–10, Berlin, Heidelberg, 2020. Springer-Verlag. doi:10.1007/978-3-030-51825-7_1.

17 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979. doi:10.2307/2273702.

18 Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt Jr., Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In Leonardo de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 220–236. Springer, 2017. doi:10.1007/978-3-319-63046-5_14.

19 Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960. doi:10.1145/321033.321034.

20 Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing*, pages 61–75, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

21 Katalin Fazekas, Florian Pollitt, Mathias Fleury, and Armin Biere. Incremental inprocessing rules beyond resolution. In Jochen Hoenicke, Mikoláš Janota, Aina Niemetz, and Sophie Tourret, editors, *Proceedings 16'th Pragmatics of SAT International Workshop (POS'25), co-located with the 28th International Conference on Theory and Applications of Satisfiability Testing (SAT'23)*, number 4008 in CEUR Workshop Proceedings, pages 190–200, 2025. URL: https://ceur-ws.org/Vol-4008.

22 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985. Third Conference on Foundations of Software Technology and Theoretical Computer Science. doi:10.1016/0304-3975(85)90144-6.

23 Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Short proofs without new variables. In Leonardo de Moura, editor, *Automated Deduction – CADE 26*, pages 130–147, Cham, 2017. Springer International Publishing.

24 Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Clausal proofs of mutilated chessboards. In Julia M. Badger and Kristin Yvonne Rozier, editors, *NASA Formal Methods*, pages 204–210, Cham, 2019. Springer International Publishing.

25 Marijn J.H. Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2024: Solver, Benchmark and Proof Checker Descriptions*. Department of Computer Science Report Series B. Department of Computer Science, University of Helsinki, Finland, 2024.

26 Ashlin Iser and Christoph Jabs. Global benchmark database. In Supratik Chakraborty and Jie-Hong Roland Jiang, editors, *27th International Conference on Theory and Applications of Satisfiability Testing, SAT 2024, Pune, India, August 21-24, 2024*, volume 305 of *LIPIcs*, pages 18:1–18:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.SAT.2024.18.

27 Matti Järvisalo, Marijn J. H. Heule, and Armin Biere. Inprocessing rules. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning*, pages 355–370, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

28 Daniela Kaufmann, Manuel Kauers, Armin Biere, and David Cok. Arithmetic verification problems submitted to the SAT Race 2019. In Marijn Heule, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Race 2019 – Solver and Benchmark Descriptions*, volume B-2019-1 of *Department of Computer Science Series of Publications B*, page 49. University of Helsinki, 2019.

29 Oliver Kullmann. On a generalization of extended resolution. *Discret. Appl. Math.*, 96-97:149–176, 1999. doi:10.1016/S0166-218X(99)00037-2.

30 Massimo Lauria, Jan Elffers, Jakob Nordström, and Marc Vinyals. CNFgen: A generator of crafted benchmarks. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of*

Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings, volume 10491 of *Lecture Notes in Computer Science*, pages 464–473. Springer, 2017. doi:10.1007/978-3-319-66263-3_30.

31 Chu Min Li. Equivalent literal propagation in the DLL procedure. *Discret. Appl. Math.*, 130(2):251–276, 2003. doi:10.1016/S0166-218X(02)00407-9.

32 Andrew Luka and Yakir Vizel. Property directed reachability with extended resolution. In Ruzica Piskac and Zvonimir Rakamaric, editors, *Computer Aided Verification - 37th International Conference, CAV 2025, Zagreb, Croatia, July 23-25, 2025, Proceedings, Part I*, volume 15931 of *Lecture Notes in Computer Science*, pages 258–280. Springer, 2025. doi:10.1007/978-3-031-98668-0_13.

33 Norbert Manthey, Marijn Heule, and Armin Biere. Automated reencoding of Boolean formulas. In Armin Biere, Amir Nahir, and Tanja E. J. Vos, editors, *Hardware and Software: Verification and Testing - 8th International Haifa Verification Conference, HVC 2012, Haifa, Israel, November 6-8, 2012. Revised Selected Papers*, volume 7857 of *Lecture Notes in Computer Science*, pages 102–117. Springer, 2012. doi:10.1007/978-3-642-39611-3_14.

34 João Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 133–182. IOS Press, 2nd edition, 2021. doi:10.3233/FAIA200987.

35 Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535. ACM, 2001. doi:10.1145/378239.379017.

36 Alexander Nadel and Vadim Ryvchin. Chronological backtracking. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 111–121. Springer, 2018. doi:10.1007/978-3-319-94144-8_7.

37 Chanseok Oh. Between SAT and UNSAT: the fundamental difference in CDCL SAT. In Marijn Heule and Sean A. Weaver, editors, *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, volume 9340 of *Lecture Notes in Computer Science*, pages 307–323. Springer, 2015. doi:10.1007/978-3-319-24318-4_23.

38 Florian Pollitt, Zachary Battleman, Mathias Fleury, Yakir Vizel, Marijn Heule, Armin Biere, and Randal Bryant. Facrtoring-learned-clauses-artifact, May 2026. doi:10.5281/zenodo.20154935.

39 Joseph E. Reeves, Marijn J. H. Heule, and Randal E. Bryant. Preprocessing of propagation redundant clauses. *Journal of Automated Reasoning*, 67(3):31, Sep 2023. doi:10.1007/s10817-023-09681-3.

40 Amar Shah, Twain Byrnes, Joseph Reeves, and Marijn J. H. Heule. Learning short clauses via conditional autarkies. In *2025 Formal Methods in Computer-Aided Design (FMCAD)*, pages 1–11, 2025. doi:10.34727/2025/isbn.978-3-85448-084-6_17.

41 G. S. Tseitin. *On the Complexity of Derivation in Propositional Calculus*, pages 466–483. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983. doi:10.1007/978-3-642-81955-1_28.

42 Alasdair Urquhart. The complexity of propositional proofs. *The Bulletin of Symbolic Logic*, 1(4):425–467, 1995. URL: http://www.jstor.org/stable/421131.