Trade-offs Between Time and Memory in a Tighter Model of CDCL SAT Solvers

Jan Elffers[1], Jan Johannsen[2], Massimo Lauria[3], Thomas Magnard[4], Jakob Nordström[1], and Marc Vinyals[1]

[1] KTH Royal Institute of Technology, Stockholm, Sweden
{elffers,jakobn,vinyals}@kth.se
[2] Ludwig-Maximilians-Universität München, Munich, Germany
Jan.Johannsen@ifi.lmu.de
[3] Universitat Politècnica de Catalunya, Barcelona, Spain
lauria@cs.upc.edu
[4] École Normale Supérieure, Paris, France
magnard@clipper.ens.fr

Abstract. A long line of research has studied the power of conflict-driven clause learning (CDCL) and how it compares to the resolution proof system in which it searches for proofs. It has been shown that CDCL can polynomially simulate resolution even with an adversarially chosen learning scheme as long as it is asserting. However, the simulation only works under the assumption that no learned clauses are ever forgotten, and the polynomial blow-up is significant. Moreover, the simulation requires very frequent restarts, whereas the power of CDCL with less frequent or entirely without restarts remains poorly understood. With a view towards obtaining results with tighter relations between CDCL and resolution, we introduce a more fine-grained model of CDCL that captures not only time but also memory usage and number of restarts. We show how previously established strong size-space trade-offs for resolution can be transformed into equally strong trade-offs between time and memory usage for CDCL, where the upper bounds hold for CDCL without any restarts using the standard 1UIP clause learning scheme, and the (in some cases tightly matching) lower bounds hold for arbitrarily frequent restarts and arbitrary clause learning schemes.

1 Introduction

For two decades the dominant strategy for solving the Boolean satisfiability problem (SAT) in practice has been *conflict-driven clause learning (CDCL)* [5, 27, 28]. Although SAT is an NP-complete problem, and is hence widely believed to be intractable in the worst case, CDCL SAT solvers have turned out to be immensely successful over a wide range of application areas. An important problem is to understand how such SAT solvers can be so efficient and what theoretical limits exist on their performance.

Previous Work At the core, CDCL searches for proofs in the proof system *resolution* [14]. While pre- and inprocessing techniques can, and sometimes do, go significantly beyond resolution (incorporating, e.g., solving of linear equations mod 2 and reasoning with cardinality constraints), understanding the power of even just the fundamental CDCL algorithm seems like an interesting and challenging problem in its own right. Three crucial aspects of CDCL solvers, which are the focus of our work, are running time, memory usage, and restart policy.

In resolution, time is modelled by the size/length complexity measure, in that lower bounds on proof size yield lower bounds on the running time of CDCL solvers. Resolution proof size is a well-studied measure. It is not hard to show that it need never be larger than exponential in the formula size, and such exponential lower bounds were shown already in, e.g., [20, 25, 31].

Another more recently studied measure is *(clause) space*, measured as the number of clauses needed in memory while verifying a proof.[5] We remark that although the study of space was originally motivated by SAT solving concerns, it is not a priori clear to what extent this abstract space measure corresponds to CDCL memory usage. Space need never be more than linear in the worst case [24], even though such proofs might have exponential size, and optimal linear lower bounds on space were obtained in [1, 10, 24].

More interesting than such space bounds is perhaps what can be said regarding simultaneous optimization of time and space, which is the setting in which SAT solvers operate. There are strong trade-offs [6, 9, 11] showing that this is not possible in general. What this means is that one can find formulas for which (a) there are short proofs and (b) also space-efficient proofs but (c) no proof can get close to being simultaneously both size- and space-efficient.

Regarding restarts, such a concept does not quite make sense for resolution proofs and so has not been studied in that context as far as we are aware.

It is natural to ask to what extent upper and lower bounds for resolution apply to CDCL. By comparison, it is well understood that the *DPLL* method [22, 23] searches for proofs in *tree-like resolution*, which incurs an exponential loss in performance as compared to general resolution. There has been a long line of research investigating how CDCL compares to general resolution, e.g., [7, 18, 26, 32], culminating in the result by [30] that CDCL viewed as a proof system polynomially simulates resolution with respect to size/time. The nonconstructive part of this result is that variable decisions are not done according to some concrete heuristic but are provided as helpful advice to the solver. This limitation is probably inherent, since a fully algorithmic result would have unexpected implications in complexity theory [2]. It is worth noting, however, that in independent work [3] showed that for resolution proofs where all clauses have constant size, using a *random* variable selection heuristic will yield a constructive polynomial-time simulation.

[5] We mention for completeness that there is also a *total space* measure counting the number of literals in memory, which has been studied in, e.g., [1, 13, 15, 16], but for our purposes clause space seems like a more relevant measure to focus on.

One strength of [3, 30] is that the results hold for any learning scheme as long as it is *asserting* (an assumption that anyway lies at the heart of the CDCL algorithm). The results also have a few less desirable aspects, however:

– The simulations require very frequent restarts. Only the first conflict after each restart is useful, and after that one has to wait for the next restart to make any further progress.
– There is also a large polynomial blow-up in the simulations, which means that for practical purposes these simulations are far too inefficient to yield really concrete insights into CDCL performance as compared to resolution.
– Finally, and most seriously, the results crucially rely on the assumption that no learned clause is ever forgotten. This is unrealistic, as typically around 90–95% of learned clauses are erased during CDCL search and this is absolutely essential for performance.

It would be desirable to obtain results relating CDCL and resolution that also take the above aspects into account.

Addressing one of these concerns, a more fine-grained study of the power of CDCL without restarts has been conducted in, e.g., [8, 17, 18, 19]. One problematic aspect here is that the models studied appear to be quite far from actual CDCL behaviour. Some papers assume non-standard and rather artificial preprocessing steps. Others study CDCL models that do not enforce that unit clauses are propagated or that do not trigger conflict analysis as soon as a clause is falsified. In the latter case, as a result one gets very limited restrictions on what the clause learning schemes are, and it is hard even to talk about what "conflict analysis" is supposed to mean in this context. This is not an issue for results establishing lower bounds limiting what CDCL can do—here a stronger model of CDCL only makes the results stronger—but for upper bounds the results become too optimistic, indicating that the theoretical CDCL model can do much better than what seems possible in practice. As a case in point, there are currently no known separations between general resolution and CDCL without restarts, but part of the reason for this appears to be that the models of CDCL without restarts are clearly too strong to be realistic.

We are not aware of any work on models measuring not only time but also memory consumption in a proof system formalizing CDCL. As discussed above, one can define a space measure for resolution proofs, but it is not clear what relation, if any, there is between this space measure and the size of the clause database during CDCL execution.

Our Contributions In this work, we present a proof system that tightly models running time, memory usage, and restarts in CDCL. The model draws heavily on [3, 30], combined with ideas from [18] to capture memory and restarts. Indeed, we do not claim any key new technical insights for this part of our work, but rather it is more a matter of carefully studying previous models and painstakingly putting the pieces together to get as clean and simple a proof system as possible that is nevertheless significantly "closer to the metal" than in previous papers.

Our CDCL proof system enforces unit propagation and triggers conflict analysis directly at a conflict. It can incorporate any asserting learning scheme (as long as it is based on resolution derivations from the current conflict and reason clauses), and this scheme is specified explicitly as a parameter. Right from the definitions one obtains natural measures of time, memory usage, and restarts. Variable decisions are still provided externally, just as in [3, 30], but in principle one could also plug in, say, the most commonly used *VSIDS (variable state independent decaying sum)* decision scheme with *phase saving* and analyse what proofs can be generated using these heuristics (though this is not the focus of our current work). Since we are now managing the database of learned clauses explicitly, we also have to specify a clause database reduction policy. In this paper, the decisions about which clauses to delete are also provided to the solver, but the model allows to plug in a concrete reduction policy as well.

We argue that the proof system we present faithfully models possible execution traces during CDCL search. Some interesting questions to study in this model are as follows:

1. Do upper and lower bounds on resolution size and space transfer to this CDCL proof system?
2. How does CDCL compare to general resolution if we want efficient simulations with respect to *both* time and space, and in addition aim for at most constant-factor blow-ups rather than arbitrary polynomial blow-ups?
3. What is the power of CDCL without restarts compared to the subsystems of tree-like resolution or so-called *regular resolution*? (Briefly, regularity is the somewhat SAT solver-like restriction on resolution that along each path in the proof any variable is branched over only once.)

The worst-case upper bounds on size and space in resolution carry over to time and memory usage in CDCL, and it turns out that this can in fact be read off from [29], although that paper uses quite a different language. More interestingly, we show that there is a straightforward translation from CDCL to resolution that preserves both time and space, and so we obtain that all size and space lower bounds previously established for resolution apply also to CDCL (which, in particular, was not at all obvious for space).

This means that the lower bounds on time-space trade-offs in [6, 9, 11] also hold for CDCL. But this does not yet yield true trade-offs, since for such results we also want *upper bounds*. That is, we want to show that CDCL can find time- or space-efficient proofs optimizing just one of these measures in isolation. It is known how to construct such proofs in resolution, but these proofs are not obviously CDCL-like. Since SAT solving was mentioned as a motivation for [6, 9, 11] it is a relevant question whether the size-space trade-offs shown in these papers correspond to anything one could expect to see in practice, or whether the size- and space-efficient proofs have such peculiar structure that nothing similar can be found by CDCL proof search.

The main contribution of our work is to address the question of whether true time-space trade-offs can be established for CDCL. Finding an answer turns out to be surprisingly technically challenging, and we are not able to prove the known trade-offs for exactly the same formulas as in [6, 9, 11] However, for many of the formulas it is possible to modify them slightly to obtain CDCL trade-offs with essentially the same parameters. An additional feature of these trade-offs is that all our upper bounds hold for CDCL *without any restarts* using the standard *1UIP (first unique implication point)* learning scheme, while the (often tightly matching) lower bounds hold for arbitrarily frequent restarts and arbitrarily chosen clause learning schemes (even non-asserting ones).

We leave as open problems whether CDCL with 1UIP clause learning and with or without restarts can simulate or be separated from general or regular resolution, respectively. While those problems still look quite challenging, we hope and believe that it should be possible to make progress by investigating them in a model that more closely resembles what happens during CDCL proof search in practice, such as the model presented in this paper.

Organization of This Paper In Section 2 we describe our proof system modelling CDCL. Section 3 gives an overview of our time-space trade-off results. Since the proofs are quite long and technical, however, we have to defer essentially all of them to the full-length version of the paper, and in this extended abstract we only sketch the proof of a simpler (but still nontrivial) trade-off for CDCL *with* restarts. We make some concluding remarks in Section 4.

2 Modelling CDCL as a Proof System

We start by describing our model of CDCL and how it is formalized as a proof system. As already mentioned, this is very much inspired by [3, 30], but with ideas added from [18]. We want to remark right away that we describe the model at a level of detail that might seem excessive to SAT practitioners familiar with CDCL. We do so precisely because a serious issue with many contributions on the theoretical side has been that they fail to get crucial details of the model right, as discussed in the introduction.[6]

Preliminaries Let us first fix some standard notation and terminology. A *literal* a over a Boolean variable x is either x itself or its negation $\overline{x}$ (a *positive* or *negative* literal, respectively). A *clause* $C = a_1 \vee \cdots \vee a_k$ is a disjunction of literals, where the clause is *unit* if it contains only one literal. A *CNF formula* F is a conjunction of clauses $F = C_1 \wedge \cdots \wedge C_m$. We think of clauses and formulas as sets, so that the order of elements is irrelevant and there are no repetitions.

A *resolution derivation* of C from F is a sequence of clauses $(C_1, C_2, \ldots, C_\tau)$ such that $C_\tau = C$ and every C_i is either a clause in F (an *axiom*) or is derived

[6] Indeed there were issues with the model we presented in the conference version of this paper as well. Our description of the behaviour of the solver after a restart was not matching exactly what actual solvers seem to do in practice. Our trade-offs deal with CDCL proofs without restarts, therefore the correctness of the results was not compromised.

from clauses C_j, C_k with $j, k < i$, by the *resolution rule*

$$\frac{C \vee x \qquad D \vee \overline{x}}{C \vee D}\ , \tag{1}$$

where we say that $C \vee x$ and $D \vee \overline{x}$ are *resolved* over x. A derivation is *trivial* if all variables resolved over are distinct and each C_i either is an axiom or is derived from a resolution rule application where one of the resolved clauses is an axiom. A *resolution refutation* of, or *resolution proof* for, an unsatisfiable formula F is a derivation of the empty clause $\bot$ (containing no literals) from the axioms in F. The *length* or *size* of a proof is the number of clauses in it counted with repetitions. The space of a proof at step t is the number of clauses at steps $\leq t$ that are used in applications of the resolution rule at steps $\geq t$. The space of a proof is obtained by measuring the space at each step and taking the maximum.

A Formal Description of CDCL A CDCL solver running on a formula F decides variable assignments and propagates values that follow from such assignments until a clause is falsified, at which point a learned clause is added to the clause database $\mathcal{D}$ (where we always have $F \subseteq \mathcal{D}$) and the search backtracks. A key concept is the current partial assignment maintained by the solver together with some book-keeping why variables were set this way, which we refer to as the *trail*. This is a sequence $s = (x_1 = b_1/*, x_2 = b_2/*, \ldots, x_\ell = b_\ell/*)$ where all variables are distinct and where $* = \mathsf{d}$ indicates that the assignment is a decision and $* = C$ that it was propagated by the clause C. We write $s_{\leq j}$ and $s_{<j}$ to denote the subsequences that are the prefixes of length j and $j-1$ of s, respectively. We denote the empty trail by ϵ.

The *decision level* of an assignment $x_j = b_j/*$ is the number of decision assignments in $s_{\leq j}$. The decision level of a (non-empty) trail is that of its last assignment. Identifying a trail s with the partial assignment it defines, we write $C{\restriction_s}$ to denote the clause C *restricted by* s, which is the trivially true clause if s satisfies C and otherwise C with all literals falsified by s removed, and this notation is extended to sets of clauses by taking unions. If a trail s falsifies a clause C, we say that C is *asserting* if it has a unique variable at the maximum decision level of s. If so, the second largest decision level represented in C is the *assertion level* of C.

A trail $s = (x_1 {=} b_1/*, \ldots, x_\ell {=} b_\ell/*)$ is *legal* with respect to a formula F and clause database $\mathcal{D} \supseteq F$ if the following holds:

- $\mathcal{D}{\restriction_{s_{<\ell}}}$ does not contain the empty clause;
- if the jth element of s is $x_j {=} b_j/\mathsf{d}$, then $\mathcal{D}{\restriction_{s_{<j}}}$ does not contain a unit clause;
- if the jth element of s is $x_j {=} b_j/C$, then C is contained in $\mathcal{D}$ and has the property that $C{\restriction_{s_{<j}}}$ is unit and is satisfied by setting $x_j = b_j$.

This captures properties that must hold during CDCL search, and so in what follows trails are implicitly required to be legal unless otherwise specified.

At each point in time, the solver is in a *CDCL state* $(F, \mathcal{D}, s)$, where at the beginning $\mathcal{D} = F$ and $s = \epsilon$. It is convenient to describe the solver as being in one of the four modes **Default** (where it starts), **Unit**, **Conflict**, or **Decision**, where transitions are performed as described below (guided by plug-in components that specify the detailed behaviour; also to be discussed in what follows):

Default If s falsifies a clause in $\mathcal{D}$, the solver moves to **Conflict**, otherwise it checks that all variables in F have been assigned, and in that case the solver halts and outputs `SAT` together with the assignment s. Otherwise, if $\mathcal{D}\restriction_s$ contains a unit clause, the solver transits to **Unit** mode. If none of the previous cases applies, the solver uses its *restart policy* to decide whether to restart, i.e., to set $s = \epsilon$ and to move to **Default**. At last, if none of the others cases applies, solver uses its *clause database reduction policy* to decide whether to shrink $\mathcal{D}$ to $\mathcal{D}' \subsetneq \mathcal{D}$, where $\mathcal{D}'$ must still contain F and all clauses mentioned in the current trail s, after which it moves to **Decision**.

Conflict If s has decision level 0, the solver outputs `UNSAT`. Otherwise it applies the *learning scheme* to derive an asserting clause C and then *backjumps* by updating the state to $(F, \mathcal{D} \cup \{C\}, s')$ (where s' is the prefix of s that contains all assignments with decision level less than or equal to the assertion level of C), and shifts to **Unit** mode.

Unit The solver uses the *unit propagation scheme* to pick a clause C in $\mathcal{D}$ such that $C\restriction_s$ is unit, extends s with the assignment $x = b/C$ that satisfies $C\restriction_s$, and moves to **Default** mode.

Decision The solver uses the *decision scheme* to determine an assignment $x = b/\mathsf{d}$ with which to extend the trail and moves to **Default** mode.

We say that a CDCL state $(F, \mathcal{D}, s)$ is *stable* if, when solver is in **Default** mode, it causes neither a conflict, a unit propagation nor to output `SAT`. We say it is a *conflict state* if it causes a move from **Default** to **Conflict**. We remark that CDCL solvers typically apply restarts and database reductions only in the first stable state after a conflict. However, it is not hard to see that from a proof complexity point of view the solver does not get any stronger by allowing these steps to be performed at any stable state, and since this simplifies the description we have done so above.

In order to obtain a concrete CDCL implementation, one needs to instantiate the components referred to above. Let us briefly discuss how this can be done.

For the *clause learning scheme* the assumption is that the clause is derivable in resolution from the clause falsified (the *conflict clause*) and the clauses causing unit propagations (the *reason clauses*) and that the learned clause is always asserting. For our upper bounds we use the 1UIP learning scheme from [33], which is simply a trivial resolution derivation from the conflict clause and the reason clauses processed in reverse order up to the first point when there remains only one variable of maximal decision level in the clause.[7]

The *restart policy* determines when the solver should clear the trail and start over from the beginning (but keeping the clause database as it is). From a theoretical point of view adding more frequent restarts can only make the solver more powerful. Hence, in order to obtain the strongest possible result we want to

[7] In fact, our results hold for any UIP scheme, but for simplicity we focus on 1UIP, which is anyway dominant in practice.

prove our upper bounds on CDCL with a strict no-restarts policy and our lower bounds in a setting with no restrictions on restarts.

If there is more than one unit clause that can propagate in **Unit** mode, the *unit propagation scheme* determines in which order the clauses are chosen. Typically this will depend somewhat randomly on low-level implementation details, and therefore we try to prove our upper and lower bounds for the settings when the order chosen is maximally unhelpful and maximally helpful, respectively.

The *decision scheme* is used to choose the next variable to assign when there are no unit propagations. The dominant heuristic in practice is VSIDS [28], but for our theoretical analysis we follow [3, 30] by allowing the decisions to be chosen externally by a helpful oracle and fed to the solver.

The *database reduction policy*, finally, regulates when and how to forget learned clauses. Making this aspect explicit is the main difference between our work and [3, 30]—the latter papers crucially need the unrealistic assumption that no learned clauses may ever be erased. In principle, here one could plug in, say, the *literal block distance (LBD)* heuristic in [4] to decide which clauses to throw away or keep, but in this work we will let this, too, be part of the external input provided to construct a CDCL proof.

Formalizing CDCL as a Proof System In order to construct a proof system corresponding to CDCL, we will simply let the proofs be execution traces that contain enough information to allow efficient verification that they are consistent with the detailed description of the CDCL model above. More formally, we say that a *CDCL trace* π is an ordered sequence of the following types of elements:

- decisions $x_i = b/\mathsf{d}$;
- unit propagations $x_i = b/C$ (with reason C);
- learned clauses $\mathsf{add}\, C/\sigma_C$ (with conflict analysis σ_C);
- deletions of clauses $\mathsf{del}\, C$;
- restarts R.

Given a CDCL model with components as above partially or fully specified, a trace π is *legal*, or is a *CDCL proof*, if it is consistent with an execution of the CDCL model as described above.

We say that a CDCL trace is a *CDCL proof of unsatisfiablity* or *CDCL refutation* of F if it is legal and makes the CDCL solver output `UNSAT`, and that it is a *CDCL proof of satisfiablity* if the output is `SAT`. It should be clear that if the components specified are efficiently computable, then CDCL traces are efficiently verifiable and constitute a proof system in the sense of [21] (and since all traces we construct will be legal, we will sometimes use the words "trace" and "proof" as synonyms).

The *time* of a CDCL proof π is the number of elements in the sequence plus the sum of the length of all conflict analysis resolution derivations σ_C, i.e., the total number of variable decisions, propagations, and steps in conflict analysis. The *space* of the proof at a given point in time is the number of learned clauses $|\mathcal{D} \setminus F|$, i.e., the number of statements $\mathsf{add}\, C/\sigma_C$ minus the number of statements $\mathsf{del}\, C$ up to that point, and the space of a proof is obtained by taking the maximum over all time steps in it.

These measures are intended to capture the execution time and memory usage of a CDCL solver execution described by the trace π, and in addition we want them to translate to length and space bounds for resolution. This is indeed the case, as we state in the next theorem (the proof of which is provided in the full-length version of this paper).

Theorem 1. *If there is a CDCL proof with some learning scheme using trivial resolution (in particular, 1UIP) refuting a CNF formula F in time τ and space s, then F has a resolution refutation in length at most τ and space $s + \mathrm{O}(1)$.*

The meaning of this theorem is that all lower bounds on length and space in resolution automatically carry over to impossibility results for conflict-driven clause learning. These results hold even for very general models of CDCL, with arbitrarily frequent restarts and arbitrarily smart decision and database reduction heuristics, as long as the clause learning scheme is realistic. In order to prove upper bounds for CDCL, however—i.e., showing that proof search can (at least sometimes) be performed in a time- and space-efficient manner compared to the best-case resolution proof scenario—we have to work harder.

3 Overview of Time-Space Trade-off Results

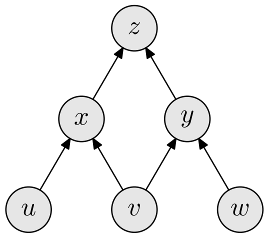
In this section we survey the kind of CDCL time-space trade-offs obtained in this paper, and discuss some of the challenges that have to be overcome when establishing such results.

Statement of Trade-off Theorems Our first set of trade-off results are for formulas defined in terms of pebble games as described in [12]. Given a directed acyclic graph (DAG) G with source vertices S and a unique sink vertex z, and with all non-sources having fan-in 2, we identify vertices with variables and define the *pebbling formula* Peb_G to consist of the following clauses:

- for all $s \in S$, the unit clause s (*source axioms*),
- for all w with predecessors u, v, the clause $\overline{u} \vee \overline{v} \vee w$ (*pebbling axioms*),
- for the sink z, the unit clause $\overline{z}$ (*sink axiom*).

These formulas are not too interesting, since it is easy to see that they are solved immediately by unit propagation, but if we replace each variable by an exclusive or of two new, fresh variables, and then expand out to CNF we obtain a *XORified pebbling formula* $Peb_G^{\oplus}$ as in Figure 1b. Given the right kind of graphs, [11] showed that such formulas have strong trade-offs between length and space in resolution, and we are able to lift most of these results to CDCL. We give two examples of such results below.

Theorem 2 (Robust trade-offs (informal)). *There are XORified pebbling formulas F_n of size $\Theta(n)$ such that:*

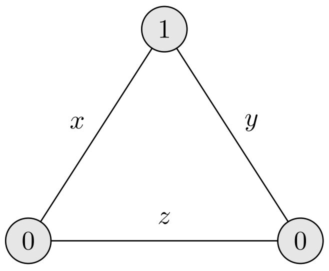


(a) Pyramid graph Π_2 of height 2.

$$\begin{aligned}
& (u_1 \lor u_2) \\
\land\ & (\overline{u}_1 \lor \overline{u}_2) \\
\land\ & (v_1 \lor v_2) \\
\land\ & (\overline{v}_1 \lor \overline{v}_2) \\
\land\ & (w_1 \lor w_2) \\
\land\ & (\overline{w}_1 \lor \overline{w}_2) \\
\land\ & (u_1 \lor \overline{u}_2 \lor v_1 \lor \overline{v}_2 \lor x_1 \lor x_2) \\
\land\ & (u_1 \lor \overline{u}_2 \lor v_1 \lor \overline{v}_2 \lor \overline{x}_1 \lor \overline{x}_2) \\
\land\ & (u_1 \lor \overline{u}_2 \lor \overline{v}_1 \lor v_2 \lor x_1 \lor x_2) \\
\land\ & (u_1 \lor \overline{u}_2 \lor \overline{v}_1 \lor v_2 \lor \overline{x}_1 \lor \overline{x}_2) \\
\land\ & (\overline{u}_1 \lor u_2 \lor v_1 \lor \overline{v}_2 \lor x_1 \lor x_2) \\
\land\ & (\overline{u}_1 \lor u_2 \lor v_1 \lor \overline{v}_2 \lor \overline{x}_1 \lor \overline{x}_2) \\
\land\ & (\overline{u}_1 \lor u_2 \lor \overline{v}_1 \lor v_2 \lor x_1 \lor x_2) \\
\land\ & (\overline{u}_1 \lor u_2 \lor \overline{v}_1 \lor v_2 \lor \overline{x}_1 \lor \overline{x}_2) \\
\land\ & (v_1 \lor \overline{v}_2 \lor w_1 \lor \overline{w}_2 \lor y_1 \lor y_2) \\
\land\ & (v_1 \lor \overline{v}_2 \lor w_1 \lor \overline{w}_2 \lor \overline{y}_1 \lor \overline{y}_2) \\
\land\ & (v_1 \lor \overline{v}_2 \lor \overline{w}_1 \lor w_2 \lor y_1 \lor y_2) \\
\land\ & (v_1 \lor \overline{v}_2 \lor \overline{w}_1 \lor w_2 \lor \overline{y}_1 \lor \overline{y}_2) \\
\land\ & (\overline{v}_1 \lor v_2 \lor w_1 \lor \overline{w}_2 \lor y_1 \lor y_2) \\
\land\ & (\overline{v}_1 \lor v_2 \lor w_1 \lor \overline{w}_2 \lor \overline{y}_1 \lor \overline{y}_2) \\
\land\ & (\overline{v}_1 \lor v_2 \lor \overline{w}_1 \lor w_2 \lor y_1 \lor y_2) \\
\land\ & (\overline{v}_1 \lor v_2 \lor \overline{w}_1 \lor w_2 \lor \overline{y}_1 \lor \overline{y}_2) \\
\land\ & (x_1 \lor \overline{x}_2 \lor y_1 \lor \overline{y}_2 \lor z_1 \lor z_2) \\
\land\ & (x_1 \lor \overline{x}_2 \lor y_1 \lor \overline{y}_2 \lor \overline{z}_1 \lor \overline{z}_2) \\
\land\ & (x_1 \lor \overline{x}_2 \lor \overline{y}_1 \lor y_2 \lor z_1 \lor z_2) \\
\land\ & (x_1 \lor \overline{x}_2 \lor \overline{y}_1 \lor y_2 \lor \overline{z}_1 \lor \overline{z}_2) \\
\land\ & (\overline{x}_1 \lor x_2 \lor y_1 \lor \overline{y}_2 \lor z_1 \lor z_2) \\
\land\ & (\overline{x}_1 \lor x_2 \lor y_1 \lor \overline{y}_2 \lor \overline{z}_1 \lor \overline{z}_2) \\
\land\ & (\overline{x}_1 \lor x_2 \lor \overline{y}_1 \lor y_2 \lor z_1 \lor z_2) \\
\land\ & (\overline{x}_1 \lor x_2 \lor \overline{y}_1 \lor y_2 \lor \overline{z}_1 \lor \overline{z}_2) \\
\land\ & (z_1 \lor \overline{z}_2) \\
\land\ & (\overline{z}_1 \lor z_2)
\end{aligned}$$

(b) XORified pebbling formula $Peb^{\oplus}_{\Pi_2}$.

Fig. 1. Example pebbling formula for the pyramid of height 2.



(a) Labelled triangle graph.

$$\begin{aligned}
& (x \lor y) \\
\land\ & (\overline{x} \lor \overline{y}) \\
\land\ & (x \lor \overline{z}) \\
\land\ & (\overline{x} \lor z) \\
\land\ & (y \lor \overline{z}) \\
\land\ & (\overline{y} \lor z)
\end{aligned}$$

(b) Corresponding Tseitin formula.

Fig. 2. Example Tseitin formula.

- *CDCL with 1UIP learning and no restarts can refute F_n in time* $\mathrm{O}(n)$ *and space* $\mathrm{O}(n/\log n)$ *simultaneously.*
- *CDCL with 1UIP learning and no restarts can refute F_n in space* $\mathrm{O}\big((\log n)^2\big)$ *and time* $n^{\mathrm{O}(\log n)}$ *simultaneously.*
- *Any CDCL refutation of F_n in space* $\mathrm{o}(n/\log n)$ *requires time at least* $n^{\Omega(\log\log n)}$ *regardless of learning scheme and restart policy.*

Theorem 3 (Exponential trade-offs (informal)). *There are XORified pebbling formulas F_n of size $\Theta(n)$ such that:*

- *CDCL with 1UIP learning and no restarts can refute F_n in time* $\mathrm{O}(n)$ *and space* $\mathrm{O}\big(\sqrt[4]{n}\big)$ *simultaneously.*
- *CDCL with 1UIP learning and no restarts can refute F_n in space* $\mathrm{O}\big(\sqrt[8]{n}\big)$ *and time* $n^{\mathrm{O}(\sqrt[8]{n})}$ *simultaneously.*
- *Any CDCL refutation of F_n in space* $\mathrm{O}\big(n^{1/4-\epsilon}\big)$ *for $\epsilon > 0$ requires exponential time regardless of learning scheme and restart policy.*

The other formula family considered in this paper are *Tseitin formulas*, which are defined in terms of undirected graphs with vertices labelled 0/1 in such a way that the total sum of all vertex labels is odd. The variables of the formula are the edges of the graph. For every vertex we add a constraint saying that the parity of the number of true edges incident to the vertex is equal to the vertex label. Summing over all vertices, each edge is counted exactly twice and hence the total number of true edges must be even. But this contradicts that the sum of the labels is odd, and thus the formulas are unsatisfiable. Figure 2b gives an example Tseitin formula generated from the labelled graph in Figure 2a.

Using Tseitin formulas over long, skinny grids, we can build on [9] to obtain the following trade-off, which applies even for superlinear space.

Theorem 4 (Superlinear space trade-offs (informal)). *For a Tseitin formula $F_{w,\ell}$ over a grid graph with w rows and ℓ columns, $1 \leq w \leq \ell^{1/4}$, and with double edges between every two vertices at horizontal distance one or vertical distance one, it holds that*

- *CDCL with 1UIP learning and no restarts can refute $F_{w,\ell}$ in time* $\mathrm{O}(2^{5w}\ell)$ *and space* $\mathrm{O}(2^{2w})$.
- *CDCL with 1UIP learning and no restarts can refute $F_{w,\ell}$ in space* $\mathrm{O}(w\log(\ell))$ *and time* $\mathrm{O}\big(\ell^{\mathrm{O}(w)}\big)$.
- *For any CDCL refutation in time τ and space s, regardless of learning scheme and restart policy, it holds that*

$$\tau = \left(\frac{2^{\Omega(w)}}{s}\right)^{\Omega\left(\frac{\log\log\ell}{\log\log\log\ell}\right)} .$$

Proof Techniques and Technical Challenges All the trade-offs stated in Theorems 2, 3, and 4 are known to hold for resolution, and so by Theorem 1 we immediately obtain that the lower bounds carry over to CDCL. What we need to show in order to establish these theorems is that CDCL can find proofs that match the upper bounds in resolution.

The general idea how we would like to do this is clear: given a resolution proof $\pi = (C_1, C_2, \ldots, C_\tau)$, we should force the CDCL solver to efficiently learn the clauses C_i one by one, making sure at all times that the clause database size is comparable to the space complexity of the resolution proof. This seems hard to do, however, and somewhat ironically what causes trouble for us are the unit propagations that otherwise make CDCL so efficient. To illustrate the problem, suppose that we have learned $C \vee x$ and $D \vee \overline{x}$ and now want to learn their resolvent $C \vee D$. It would be nice to decide on all literals in $C \vee D$ being false, after which we could get a conflict on x. But there might be other clauses in the database that propagate literals to "wrong values" before we manage to falsify all of $C \vee D$, and if so the CDCL search will veer off in another direction and we will not be able to learn this resolvent.

This highlights two technical difficulties that we need to be able to deal with:

– Not only do we have to decide on variables in the right order, but we have to make sure that no other unexpected (and unwanted) propagations occur.
– In contrast to resolution, where having more clauses at your disposal never hurts, keeping too many learned clauses in the clause database can actually hinder the CDCL search. This is also a striking contrast to [3, 30], where a key technical lemma is precisely that having more clauses in the database can only be helpful.

We do not know how to simulate general resolution efficiently with respect to length and space simultaneously, even using ever so frequent restarts. And an additional problem is that we want to know—in order to better understand basic CDCL reasoning with unit propagation and conflict analysis—whether for the pebbling and Tseitin formulas presented above CDCL can find efficient proofs *even without restarts*. This makes our task substantially more complicated.

If we allow suitably frequent restarts, however, it is not too hard to show that CDCL can efficiently simulate the "canonical" resolution proofs for these formulas. To give at least some flavour of the technical arguments needed to reason about the CDCL proof system, we conclude this section with a description of how this result can be proven for pebbling formulas.

Pebbling Formula Upper Bound for CDCL with Restarts A pebbling formula encodes the *black pebble game* played on a DAG G, where we start with G being empty and want to finish with a pebble on the sink z. A vertex can be pebbled if its predecessors have pebbles (vacuously true for sources), and pebbles can always be removed. The *time* of a pebbling is the number of moves before z is reached and the *space* is the maximum number of pebbles on vertices of G at any point.

Resolution can simulate such pebblings by deriving, whenever a vertex w is pebbled, the two *pebble clauses* $w_1 \vee w_2$ and $\overline{w}_1 \vee \overline{w}_2$ saying that the exclusive or $w_1 \oplus w_2$ is true, and by erasing these clauses whenever a pebble is removed. For a source vertex the pebble clauses are already available as source axioms

```
   Input: a black pebbling $\mathcal{P}$
1  foreach (move,w) in $\mathcal{P}$ where w is not a source or the sink do
2      if move is Add then
3          HalfPebble ($w$, 0)
4          HalfPebble ($w$, 1)
5      else
6          del $w_1 \vee w_2$
7          del $\overline{w}_1 \vee \overline{w}_2$
8  PebbleSink
```

Fig. 3. Procedure `Pebble` ($\mathcal{P}$)

in the formula (see Figure 1b), and it is not hard to show that resolution can efficiently propagate exclusive ors from predecessors to successors. Once pebble clauses have been derived for the sink z, contradiction immediately follows from the sink axioms.

We want to mimic this in CDCL as described in the algorithm `Pebble` in Figure 3 producing a CDCL trace. For a pebble placement, we want to learn first $w_1 \vee w_2$, corresponding to "half of the pebble" on w, and then $\overline{w}_1 \vee \overline{w}_2$. How to do this is described in the procedure `HalfPebble` in Figure 4, where the notation x^b for $b \in \{0, 1\}$ is used as a compact way of denoting $x^1 = x$ and $x^0 = \overline{x}$.

When a pebble is placed on w in the pebbling, we let the CDCL solver make the decisions $(w_1 = 0/\mathsf{d}, w_2 = 0/\mathsf{d})$ with the goal of learning $w_1 \vee w_2$. Then we decide values for the variables of the predecessors u and v of w, and since there are clauses in memory encoding $u_1 \oplus u_2$ and $v_1 \oplus v_2$ this will provoke repeated conflicts until finally the clause $w_1 \vee w_2$ is learned. Since this only involves a constant number of variables, the time and space required for this is constant, and our goal can be achieved, e.g., as described in `FindConflicts` in Figure 5.

But now we run into problems. At this point the CDCL solver will backjump to the decision $w_1 = 0$, where the learned clause $w_1 \vee w_2$ asserts $w_2 = 1$. As the next step, we want to generate conflicts that lead to the "second half" of the pebble $\overline{w}_1 \vee \overline{w}_2$ being learned, but there is no way this can happen since the decision $w_1 = 0$ is on the trail and the clause $\overline{w}_1 \vee \overline{w}_2$ is thus satisfied. Moreover, if the solver is not allowed to restart, then this satisfying assignment is fixed on the trail, and no new conflict could possibly cause a backjump to before this assignment. Therefore, the solver is forced to continue the proof search elsewhere. This turns out to be a major obstacle, which we are able to circumvent only by substantial extra work involving reordering the pebbling and using a different algorithm. Unfortunately the technical arguments are rather intricate and the algorithm too long to describe; we refer to the full-length version for the details.

If we instead give the solver the option to restart at this point, it can clear the trail and also forget all unnecessary clauses. This means that the decisions $(w_1 = 1/\mathsf{d}, w_2 = 1/\mathsf{d})$ can be made, after which the clause $\overline{w}_1 \vee \overline{w}_2$ is learned in

```
Input: A vertex $w$, a Boolean $b$
1 Decide $w_1 = b/\mathsf{d}$
2 Decide $w_2 = b/\mathsf{d}$
3 FindConflicts $(w, b)$
4 Learn $w_1^{1-b} \lor w_2^{1-b}$ and assert $w_2 = 1-b/\mathsf{u}$
5 Restart $\mathsf{R}$
6 foreach clause $C \in \mathcal{D} \setminus F$ such that $|C| > 2$ do
7     $\mathsf{del}\, C$
```

Fig. 4. Procedure `HalfPebble` (w, b)

```
Input: A vertex $w$ with predecessors $u$ and $v$, a Boolean $b$
1 Decide $u_1 = 0/\mathsf{d}$
2 Propagate $u_2 = 1/u_1 \lor u_2$
3 Decide $v_1 = 0/\mathsf{d}$
4 Learn $w_1^{1-b} \lor w_2^{1-b} \lor u_1 \lor \overline{u}_2 \lor v_1$
5 Assert $v_1 = 1/w_1^{1-b} \lor w_2^{1-b} \lor u_1 \lor \overline{u}_2 \lor v_1$
6 Learn $w_1^{1-b} \lor w_2^{1-b} \lor u_1$
7 Assert $u_1 = 1/w_1^{1-b} \lor w_2^{1-b} \lor u_1$
8 Propagate $u_2 = 0/\overline{u}_1 \lor \overline{u}_2$
9 Decide $v_1 = 0/\mathsf{d}$
10 Learn $w_1^{1-b} \lor w_2^{1-b} \lor \overline{u}_1 \lor u_2 \lor v_1$
11 Assert $v_1 = 1/w_1^{1-b} \lor w_2^{1-b} \lor \overline{u}_1 \lor u_2 \lor v_1$
```

Fig. 5. Procedure `FindConflicts` (w, b)

the same way as above. To conclude, we again trigger a restart and erase all auxiliary clauses that are no longer needed.

Pebble removals are very straightforward to simulate: the only condition that could stop us from erasing the clauses $w_1 \lor w_2$ and $\overline{w}_1 \lor \overline{w}_2$ is if they are reasons for propagated literals on the trail, but since we have just made a restart the trail is empty. Formalizing the arguments above, we obtain the following lemma.

Lemma 5. *If $\mathcal{P}$ is a black pebbling of G in space s and time τ, then there is a CDCL proof of $Peb_G^{\oplus}$ with restarts using the 1UIP learning scheme and any unit propagation scheme in space at most $2s + 3$ and time $\mathrm{O}(\tau)$.*

Proof. Given a pebbling $\mathcal{P}$ we generate a trace as described by the procedure `Pebble`. Note that this procedure maintains the invariant that the pebble clauses for a non-source vertex w are in the clause database if and only if there is a pebble on w. No other clauses are in memory. The space bound follows from this invariant, and the time bound holds by construction.

It remains to check that the trace thus generated is legal. Observe that the clauses in memory only propagate if at least one variable from each vertex they mention is set. Since the decision sequence mentions at most three vertices at the same time, we only need to reason about clauses that mention these vertices.

The correctness of `FindConflicts` is straightforward to verify, since the order of unit propagations can be seen to be uniquely determined. At the end of `FindConflicts`, the assignments to w_1 and w_2 are decisions and all predecessor variables u_1, u_2, v_1, v_2 are set by unit propagation. Since one of the conflicting clauses, a pebbling axiom, contains the decision variables w_1 and w_2, they have to appear in any conflict clause. The remaining variables involved in the conflict have maximal decision level, so they cannot appear in an asserting clause because w_2 already appears. Therefore, we learn the clause $w_1^{1-b} \lor w_2^{1-b}$ and assert $w_2 = 1 - b/\mathtt{u}$. We only erase clauses after a restart, so no erased clauses can be reasons for unit propagations. This concludes the proof.

4 Concluding Remarks

In this paper, we present a proof system that closely models conflict-driven clause learning (CDCL) and yields natural measures not only of running time but also of memory usage and number of restarts. To the best of our knowledge, previous papers considered either zero restarts or very frequent restarts, and none of the models captured space. We show that lower bounds on proof size and space in resolution carry over to this CDCL proof system. Furthermore, we establish that currently known trade-offs between size and space in resolution can be transformed into essentially equally strong trade-offs between time and memory usage for CDCL, where the upper bounds are achieved by CDCL without any restarts using the standard 1UIP clause learning scheme, and the lower bounds apply even for arbitrarily frequent restarts and arbitrary clause learning schemes.

The focus of our work is theoretical, namely to see if CDCL proof search is in principle subject to the kind of trade-offs shown previously for the resolution proof system in which it searches for proofs. Since the answer turns out to be yes, an interesting direction for future work would be to investigate experimentally whether anything like these time-space trade-offs show up also in practice.

Two other interesting problems are whether CDCL with 1UIP can simulate general resolution efficiently with respect to both time and space (measuring time only, a polynomial simulation follows from [30]), and whether CDCL with 1UIP and without restarts can simulate or be separated from regular resolution. If one believes that a separation should be more likely, a first step could be to revisit the formulas in [17, 19] and study them in our model, which is much closer to actual CDCL search and where proving lower bounds might therefore be easier. It should be said, though, that both of these problems still look like formidable challenges.

A more specialized question along the same lines, but still quite intriguing, is what can be said if VSIDS and phase saving is plugged into our CDCL model. The VSIDS heuristic seems like an important part of what makes CDCL SAT solvers so successful in practice, and yet there are also theoretical combinatorial formulas where it seems to be less useful. It would be interesting if one could find explicit examples of formulas where VSIDS in combination with phase saving goes provably wrong compared to the best possible resolution proof, causing a large polynomial or even superpolynomial blow-up in proof size.

Acknowledgements We are grateful to the anonymous *SAT* conference reviewers for detailed comments that helped improve the exposition in this paper.

The third author performed this work while at KTH Royal Institute of Technology, and most of the work of the second and fourth author was done while visiting KTH. The first, third, fifth, and sixth author were funded by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no. 279611 as well as by Swedish Research Council grant 621-2012-5645. The third author was also supported by the European Research Council under the European Union's Horizon 2020 Research and Innovation Programme / ERC grant agreement no. 648276.

References

[1] Alekhnovich, M., Ben-Sasson, E., Razborov, A.A., Wigderson, A.: Space complexity in propositional calculus. SIAM Journal on Computing 31(4), 1184–1211 (2002), preliminary version in *STOC '00*

[2] Alekhnovich, M., Razborov, A.A.: Resolution is not automatizable unless W[P] is tractable. SIAM Journal on Computing 38(4), 1347–1363 (Oct 2008), preliminary version in *FOCS '01*

[3] Atserias, A., Fichte, J.K., Thurley, M.: Clause-learning algorithms with many restarts and bounded-width resolution. Journal of Artificial Intelligence Research 40, 353–373 (Jan 2011), preliminary version in *SAT '09*

[4] Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI '09). pp. 399–404 (Jul 2009)

[5] Bayardo Jr., R.J., Schrag, R.: Using CSP look-back techniques to solve real-world SAT instances. In: Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97). pp. 203–208 (Jul 1997)

[6] Beame, P., Beck, C., Impagliazzo, R.: Time-space tradeoffs in resolution: Superpolynomial lower bounds for superlinear space. In: Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC '12). pp. 213–232 (May 2012)

[7] Beame, P., Kautz, H., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. Journal of Artificial Intelligence Research 22, 319–351 (Dec 2004), preliminary version in *IJCAI '03*

[8] Beame, P., Sabharwal, A.: Non-restarting SAT solvers with simple preprocessing can efficiently simulate resolution. In: Proceedings of the 28th National Conference on Artificial Intelligence (AAAI '14). pp. 2608–2615. AAAI Press (Jul 2014)

[9] Beck, C., Nordström, J., Tang, B.: Some trade-off results for polynomial calculus. In: Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC '13). pp. 813–822 (May 2013)

[10] Ben-Sasson, E., Galesi, N.: Space complexity of random formulae in resolution. Random Structures and Algorithms 23(1), 92–109 (Aug 2003), preliminary version in *CCC '01*

[11] Ben-Sasson, E., Nordström, J.: Understanding space in proof complexity: Separations and trade-offs via substitutions. In: Proceedings of the 2nd Symposium on Innovations in Computer Science (ICS '11). pp. 401–416 (Jan 2011)

[12] Ben-Sasson, E., Wigderson, A.: Short proofs are narrow—resolution made simple. Journal of the ACM 48(2), 149–169 (Mar 2001), preliminary version in *STOC '99*

[13] Bennett, P., Bonacina, I., Galesi, N., Huynh, T., Molloy, M., Wollan, P.: Space proof complexity for random 3-CNFs. Tech. Rep. 1503.01613, arXiv.org (Apr 2015)

[14] Blake, A.: Canonical Expressions in Boolean Algebra. Ph.D. thesis, University of Chicago (1937)

[15] Bonacina, I.: Total space in resolution is at least width squared. In: Proceedings of the 43rd International Colloquium on Automata, Languages and Programming (ICALP '16) (Jul 2016), to appear

[16] Bonacina, I., Galesi, N., Thapen, N.: Total space in resolution. In: Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS '14). pp. 641–650 (Oct 2014)

[17] Bonet, M.L., Buss, S., Johannsen, J.: Improved separations of regular resolution from clause learning proof systems. Journal of Artificial Intelligence Research 49, 669–703 (2014)

[18] Buss, S.R., Hoffmann, J., Johannsen, J.: Resolution trees with lemmas: Resolution refinements that characterize DLL-algorithms with clause learning. Logical Methods in Computer Science 4(4:13) (Dec 2008)

[19] Buss, S.R., Kołodziejczyk, L.: Small stone in pool. Logical Methods in Computer Science 10 (Jun 2014)

[20] Chvátal, V., Szemerédi, E.: Many hard examples for resolution. Journal of the ACM 35(4), 759–768 (Oct 1988)

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

[22] Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Communications of the ACM 5(7), 394–397 (Jul 1962)

[23] Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM 7(3), 201–215 (1960)

[24] Esteban, J.L., Torán, J.: Space bounds for resolution. Information and Computation 171(1), 84–97 (2001), preliminary versions of these results appeared in *STACS '99* and *CSL '99*

[25] Haken, A.: The intractability of resolution. Theoretical Computer Science 39(2-3), 297–308 (Aug 1985)

[26] Hertel, P., Bacchus, F., Pitassi, T., Van Gelder, A.: Clause learning can effectively P-simulate general propositional resolution. In: Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI '08). pp. 283–290 (Jul 2008)

[27] Marques-Silva, J.P., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computers 48(5), 506–521 (May 1999), preliminary version in *ICCAD '96*

[28] Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference (DAC '01). pp. 530–535 (Jun 2001)

[29] Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). Journal of the ACM 53(6), 937–977 (2006)

[30] Pipatsrisawat, K., Darwiche, A.: On the power of clause-learning SAT solvers as resolution engines. Artificial Intelligence 175, 512–525 (Feb 2011), preliminary version in *CP '09*

[31] Urquhart, A.: Hard examples for resolution. Journal of the ACM 34(1), 209–219 (Jan 1987)

[32] Van Gelder, A.: Pool resolution and its relation to regular resolution and DPLL with clause learning. In: Proceedings of the 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '05). Lecture Notes in Computer Science, vol. 3835, pp. 580–594. Springer (2005)

[33] Zhang, L., Madigan, C.F., Moskewicz, M.W., Malik, S.: Efficient conflict driven learning in Boolean satisfiability solver. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '01). pp. 279–285 (Nov 2001)