Improving and Understanding the Power of Satisfaction-Driven Clause Learning

ALBERT OLIVERAS*, Technical University of Catalonia, Spain
CHUNXIAO LI, University of Waterloo, Canada
DARRYL WU, University of Waterloo, Canada
JONATHAN CHUNG, Lorica Cybersecurity, Canada
VIJAY GANESH, Georgia Institute of Technology, USA

In this paper, we explain how to improve Satisfaction-Driven Clause Learning (SDCL) SAT solvers by using a MaxSAT-based technique that enables them to learn shorter, and hence better, redundant clauses. A thorough empirical evaluation of an implementation on the MapleSAT solver shows that the resulting system solves Mutilated Chess Board (MCB) problems significantly faster than CDCL solvers, without requiring any alteration to the branching heuristic used by the underlying CDCL SAT solver.

Additionally we improve the understanding of the power of these solvers by proving that, given a refutation of a formula that consists of resolution and redundant-clause addition steps, an SDCL solver is able to produce a proof whose size is polynomial with respect to the size of the original refutation.

JAIR Associate Editor: Chu-Min LI

JAIR Reference Format:
Albert Oliveras, Chunxiao Li, Darryl Wu, Jonathan Chung, and Vijay Ganesh. 2025. Improving and Understanding the Power of Satisfaction-Driven Clause Learning. *Journal of Artificial Intelligence Research* 83, Article 30 (August 2025), 25 pages. DOI: 10.1613/jair.1.18286

1 Introduction

Despite the fact that the Boolean satisfiability (SAT) problem is well known to be NP-complete [12] and believed to be intractable in general, the use of Conflict-Driven Clause Learning (CDCL) SAT solvers is a common practice in a variety of areas such as software engineering [7], formal verification [8], security [13, 30] and AI [6]. The transition from simple tools that are only effective on toy instances to mature systems applicable to real-world problems has been possible thanks to the steady progress in improving CDCL solvers' components and heuristics [22].

However, theoretical results somehow limit the potential of these type of improvements. It is well known that CDCL SAT solvers are polynomially equivalent to resolution [24, 1], and consequently it follows that classes of formulas, such as the pigeon hole principle (PHP), that are hard for resolution are also hard for CDCL SAT solvers. A promising research direction that researchers are actively following is to design and implement solvers that correspond to stronger propositional proof systems.

*Corresponding Author.

Authors' Contact Information: Albert Oliveras, ORCID: 0000-0002-5893-1911, oliveras@cs.upc.edu, Technical University of Catalonia, Barcelona, Spain; Chunxiao Li, c279li@uwaterloo.ca, University of Waterloo, Waterloo, Canada; Darryl Wu, c279li@uwaterloo.ca, University of Waterloo, Waterloo, Canada; Jonathan Chung, ORCID: 0000-0001-5378-1136, jonathan.chung1@uwaterloo.ca, Lorica Cybersecurity, Canada; Vijay Ganesh, ORCID: 0000-0002-6029-2047, vganesh45@gatech.edu, Georgia Institute of Technology, Atlanta, Georgia, USA.

DOI: 10.1613/jair.1.18286

One such class of solvers is called Satisfaction-Driven Clause Learning (SDCL) solvers [17, 16, 14], which are based on the propagation redundancy (PR) property [15, 16]. The SDCL paradigm extends CDCL in the following way: unlike CDCL solvers, SDCL solvers may learn clauses even when an assignment trail α is consistent. To be more precise, an SDCL solver first computes a new formula $P_\alpha(F)$, known as a pruning predicate. Then, it checks the satisfiability of $P_\alpha(F)$. If it satisfiable, it means $\neg\alpha$ is *redundant* with respect to the formula, and the solver can learn the clause $(\neg\alpha)$.

A natural question to ask is whether this very concrete way of allowing an SDCL solver to add redundant clauses allows it to achieve the full power of redundant-clause addition. In a similar fashion as it was proved in [1, 24] that CDCL is equivalent to general resolution, we prove here that SDCL with no clause deletion is equivalent to a proof system that consists of only resolution steps and redundant-clause addition steps. That is, SDCL can be considered as the way to automate clause-redundant addition.

Despite this strong positive result it is still an extremely challenging task to automate SDCL. A first concern is that the satisfiability check for the formula $P_\alpha(F)$ is NP-complete and is hard to solve in general. It essentially requires the SDCL solver to call another SAT solver that we refer to as a sub-solver. Given that this sub-solver call can be expensive, one needs to be strategic about when to invoke it during the run of an SDCL solver. Second, the clauses learned by SDCL can be large, and we want to learn shorter clauses whenever possible.

To solve this second problem, we propose a novel MaxSAT encoding of the problem of "what is the smallest subset γ of trail α, such that $P_\gamma(F)$ is satisfiable", to get the shortest clause $(\neg\gamma)$ to learn. We also apply a resolution-based technique inspired by conflict analysis to further shorten the clause. We refer to the SDCL solver augmented with our MaxSAT and clause minimization technique as MapleSDCL. Our experimental evaluation shows that MapleSDCL performs well on mutilated chess board (MCB) and bipartite perfect matching problems, that are known to be hard for CDCL solvers.

1.1 Contributions

I. We make a theoretical contribution by introducing a new type of pruning predicate, the *purely positive reduct* and a proof that it allows one to detect blocked clauses. This extends the spectrum of pruning predicates with redundancy notions associated with them. We prove that checking the satisfiability of this pruning predicate can be performed in polynomial time.

II. We prove that when an assignment has a satisfiable purely positive reduct, finding a small sub-assignment with the same property is an NP-hard problem. We prove the same result for the positive reduct, which was introduced in [17] and allows one to detect set-blocked clauses. Such small assignments summarize the reasons for the redundancy and lead to learning smaller redundant clauses. In essence, we believe that this is the equivalent to conflict analysis in CDCL solvers.

III. We introduce MaxSAT encodings of the above-stated problem for both the positive reduct and the purely positive one. Experimental results show that calling a MaxSAT solver within the SDCL architecture is not as expensive as one might expect, and more importantly, significant improvements in the size of the learned redundant clauses are achievable in practice. These improvements are even larger after applying conflict analysis techniques to convert the clause into an asserting one.

IV. We show that an SDCL solver with the positive reduct and the MaxSAT based optimization procedure can solve mutilated chess board problems without the need to alter the decision heuristic used by the underlying CDCL SAT solver. This is a very important property of our approach: the chances of learning (good) redundant clauses depend much less on choosing exactly the right decision literals, thus overcoming a serious roadblock for SDCL solver design.

V. We prove that SDCL with no clause deletion polynomially simulates general resolution plus redundant-clause addition. More precisely, let us consider a redundancy notion $\mathcal{R}$ for which a pruning predicate exists,

an unsatisfiable formula F and a proof system whose only steps are resolution and the addition of $\mathcal{R}$-clauses. We prove that for any refutation Π of F, there is an execution of an SDCL solver with no clause deletion that generates a proof of unsatisfiability of F whose size is polynomial w.r.t. the size of Π.

A preliminary version of this paper [23] was accepted at SAT 2023. This document not only contains a more comprehensive presentation of our work, but it also adds important new contributions:

- Experiments with the purely positive reduct.
- Effect of using different MaxSAT solvers to solve the optimization problems derived from SDCL.
- A novel result showing the complexity of solving the purely positive reduct.
- A novel result showing the complexity of obtaning smaller blocked clauses.
- A novel result proving that SCDL can simulate a proof system that consists of resolution and redundant-clause addition.

1.2 Outline of the paper

We start in Section 2 by providing basic definitions that will be used throughout the paper. Section 3 introduces SDCL, several redundancy notions and a variety of reducts, among which there is our novel purely positive reduct. After that, in Section 4 we present our results on hardness of the redundant-clause minimization problem and explain how this problem can be encoded into MaxSAT. Section 5 reports our experimental evaluation and the conclusions we have inferred from it. After that, Section 6 proves the equivalence between SDCL and proof system consisting of resolution and redundant-clause addition. Finally, conclusions and some direction for future work are given in Section 7.

2 Preliminaries

CNF formulas. Let $\mathcal{X}$ be a finite set of propositional *variables*. A *literal* is a propositional variable (x) or the negation of one ($\neg x$). The *negation* of a literal l, denoted $\neg l$, is x if $l = \neg x$ and is $\neg x$ if $l = x$. A *clause* is a disjunction of distinct literals $l_1 \vee \ldots \vee l_n$ (interchangeably denoted with or without brackets). The *empty clause* $\square$ is a clause with 0 literals. A *CNF formula* is a conjunction of clauses $C_1 \wedge \ldots \wedge C_m$. When convenient, we consider a clause to be the set of its literals, and a CNF to be the set of its clauses. In the rest of the paper, we assume that all formulas are CNF.

Satisfaction. An *assignment* is a set of non-contradictory literals. A *total* assignment contains, for each variable $x \in \mathcal{X}$, either x or $\neg x$. Otherwise, it is a *partial* assignment. We denote by $\neg\alpha$ the clause consisting of the negation of all literals in the assignment α. An assignment α satisfies a literal l if $l \in \alpha$, it satisfies a clause C if it satisfies at least one of the literals in C, and it satisfies a formula F if it satisfies all the clauses in F. We denote these as $\alpha \models l$, $\alpha \models C$, and $\alpha \models F$, respectively. A *model* for a formula is an assignment that satisfies it. A formula with at least one model is *satisfiable*; otherwise, it is *unsatisfiable*. Given a formula F, the *SAT* problem consists of determining whether F is satisfiable. An assignment α falsifies a literal l if $\neg l \in \alpha$, falsifies a clause if it falsifies all its literals, and falsifies a formula if it falsifies at least one of its clauses. The truth values of literals, clauses, and formulas are *undefined* for an assignment if they are neither falsified nor satisfied. Given a clause C and an assignment α, we denote by $touched_\alpha(C)$ the disjunction of all literals of C that are either satisfied or falsified by α, by $untouched_\alpha(C)$ the disjunction of all undefined literals, and by $satisfied_\alpha(C)$ the disjunction of all satisfied literals.

Unit propagation. Given a formula F and an assignment α, unit propagation extends α by repeatedly applying the following rule until reaching a fixed point: if there is a clause with all literals falsified by α except one literal l, which is undefined, add l to α. If, as a result, a clause is found that is falsified by α (called *conflict*), the procedure stops and reports that a conflict clause has been found.

Formula relations. Two formulas F and G are *equisatisfiable*, if F is satisfiable if and only if G is satisfiable, and they are *equivalent*, denoted $F \equiv G$, if they are satisfied by the same total assignments. We write $F \vdash_1 G$ (F *implies* G *by unit propagation*) if for every clause $C \in G$ of the form $l_1 \vee \ldots \vee l_n$, it holds that unit propagation applied to $F \wedge \neg l_1 \wedge \ldots \wedge \neg l_n$ results in a conflict. For the particular case when G is $\square$, this means that unit propagation on F finds a conflict. We say that G is a logical consequence of F (written $F \models G$) if all models of F are models of G.

Resolution. Given two clauses $l \vee C$ and $\neg l \vee D$, the *resolution* inference rules allows us to obtain clause $C \vee D$. Unit propagation can be seen as particular case of resolution, called *unit resolution*, where one of the clauses is a single literal.

CDCL. The Conflict-Driven Clause Learning (CDCL) algorithm is the most successful procedure to-date for determining whether certain types of industrial formulas are satisfiable [22]. Let F denote such a formula. The CDCL procedure starts with an empty assignment α, which is extended and reduced in a last-in first-out (LIFO) way, by the following three steps until the satisfiability of the formula is determined (see Algorithm 1 removing lines 9-12):

(1) Unit propagation is applied.
(2) If a conflict is found, a *conflict analysis* procedure [31] derives a clause C (called a *lemma*) which is a logical consequence of F. If C is the empty clause, we can conclude that F is unsatisfiable. Otherwise, it is guaranteed that by removing enough literals from α, a new unit propagation is possible due to C. This process is called *backjump*. Additionally, lemma C is conjuncted (*learnt*) with F, and the procedure returns to step (1).
(3) If no conflict is found in unit propagation, either α is a total assignment (and hence it satisfies the formula), or an undefined literal is chosen and added to α (the branching step). The choice of this literal, called a *decision literal*, is determined by sophisticated heuristics [4] that can have a huge impact on performance of the CDCL procedure.

MaxSAT. Given a formula F, the *MaxSAT* problem [20] consists in finding the assignment that satisfies the maximum number of clauses of F. Sometimes the clauses in F are split into *hard* and *soft clauses*, and in this case, the *Partial MaxSAT* problem consists in finding the assignment that satisfies all hard clauses and the maximum number of soft clauses.

3 Propagation Redundancy and SDCL

Despite their success on a variety of real-world applications [27, 25, 5, 18], CDCL SAT solvers have well-known limitations. techniques have established the polynomial equivalence between CDCL and general resolution [24, 1]. An important consequence of this equivalence is that if an unsatisfiable formula does not have a polynomial size proof by resolution, no run of CDCL can determine the unsatisfiability of the formula in polynomial time.

3.1 Propagation Redundancy

This limitation has pushed researchers to develop extensions of CDCL that allow the resultant method to simulate proof systems that are stronger than resolution. One such proof system is extended resolution [28]: by allowing the introduction of new variables to resolution, it can produce polynomial size proofs of the pigeon-hole principle [11], which requires exponential-size resolution proofs otherwise. However, the addition of new variables implies an increase of the search space of the formula. A newer and promising direction [15, 16] avoids the addition of new variables and is instead based on the well-known notion of redundancy:

Definition 1. *A clause C is* **redundant** *with respect to a formula F if F and $F \wedge C$ are equisatisfiable.*

Note that this definition does not provide any insight on how to derive redundant clauses in practice. For that purpose, let us first consider the following definition.

Definition 2. *Given an assignment α and a clause C, we define $\mathbf{C}|_\alpha = \top$ if $\alpha \models C$; otherwise $\mathbf{C}|_\alpha$ is the clause consisting of all literals of C that are undefined in α. For a formula F, we define the formula $\mathbf{F}|_\alpha = \{C|_\alpha \mid C \in F \text{ and } \alpha \not\models C\}$.*

We can now provide a more useful characterization of redundancy:

Theorem 1 ([15], Theorem 1). *A non-empty clause C is redundant with respect to a formula F if and only if there exists an assignment ω such that $\omega \models C$ and $F \wedge \neg C \models F|_\omega$.*

From a practical point of view, this characterization does not help much yet, because even if we know ω (known as the *witness*) it is hard to check whether the property holds. This is why a more limited notion of redundancy has been defined [15]:

Definition 3. *A clause C is* **propagation redundant (PR)** *with respect to a formula F if there exists an assignment ω such that $\omega \models C$ and $F \wedge \neg C \vdash_1 F|_\omega$*

Since $F \wedge \neg C \vdash_1 F|_\omega$ implies $F \wedge \neg C \models F|_\omega$, any PR clause is redundant. Hence, we can add PR clauses to our formula in order to make it easier to solve without affecting its satisfiability. Note that, in general, ω can contain any literal of F. If we force ω to assign all variables in C but no other variable, we obtain weaker but simpler notions of redundancy: if we force ω to satisfy exactly one literal of C, we obtain *literal-propagation redundant (LPR)* clauses; if we allow ω to satisfy more than one literal of C, we obtain *set-propagation redundant (SPR)* clauses. Obviously, any LPR clause is SPR, and any SPR clause is PR, but none of these three notions are equivalent as the following examples show.

Example 1 ([15])**.** Let $F = \{x \vee y,\ \ x \vee \neg y \vee z,\ \ \neg x \vee z,\ \ \neg x \vee u,\ \ x \vee \neg u\}$ and $C = x \vee u$. The witness $\omega = \{x, u\}$ satisfies C and, since $F|_\omega = \{z\}$, it holds that $F \wedge \neg C \vdash_1 F|_\omega$, that is, unit propagation on $F \wedge \neg x \wedge \neg u \wedge \neg z$ results in a conflict. Hence, C is SPR w.r.t. F.

However, it is not LPR. The reason is that there are only two possible witnesses that satisfy exactly one literal of C: $\omega_1 = \{x, \neg u\}$ and $\omega_2 = \{\neg x, u\}$. But we have that both $F|_{\omega_1}$ and $F|_{\omega_2}$ contain, among others, the empty clause. Hence, $F \wedge \neg C \vdash_1 F|_{\omega_1}$ and $F \wedge \neg C \vdash_1 F|_{\omega_2}$ require that unit propagation on $F \wedge \neg C$, that is, $F \wedge \neg x \wedge \neg u$, results in a conflict, which is not the case.

Example 2 ([15])**.** Let $F = \{x \vee y, \neg x \vee y, \neg x \vee z\}$ and $C = (x)$. If we consider the witness $\omega = \{x, z\}$, we have that $F|_\omega = \{y\}$. It is obvious that $\omega \models C$ and also $F \wedge \neg x \vdash_1 y$. Thus, C is PR w.r.t. F. However it is not SPR because the only possible witness would be $\omega_1 = \{x\}$, but $F|_{\omega_1} = \{y, z\}$ and it does not hold that $F \wedge \neg x \vdash_1 z$.

3.2 SDCL and Reducts

It was proved in [15] that the proof system that combines resolution with the addition of PR clauses admits polynomial-sized proofs for the pigeon hole principle. Hence, we would like to develop a procedure that determines which PR clauses the CDCL solver should and when this addition should be done. This is clearly a non-trivial task that was addressed with the development of Satisfiability-Driven Clause Learning (SDCL) [17]. The key notion in this new solving paradigm is the one of *pruning predicate*:

Definition 4. *Let F be a formula and α an assignment. A* **pruning predicate** *for F and α is a formula $P_\alpha(F)$ such that if it is satisfiable, then the clause $\neg\alpha$ is redundant w.r.t. F.*

SDCL extends CDCL in the following way (See also Algorithm 1). Before making a decision, a pruning predicate for the assignment α and formula F is constructed. If satisfiable, we can learn $\neg\alpha$ and use it for backjump and

```
1  α := ∅
2  while true do
3      α := unitPropagate(F, α)
4      if conflict found then
5          C := analyzeConflict()
6          F := F ∧ C
7          if C is the empty clause then return UNSAT
8          α := backjump(C, α)
9      else if P_α(F) is satisfiable then
10         C := analyzeWitness()
11         F := F ∧ C
12         α := backjump(C, α)
13     else
14         if all variables are assigned then return SAT
15         α := α ∪ Decide()
16     end
17 end
```

Algorithm 1: The SDCL algorithm. Note that removing lines 9-12 results in the CDCL algorithm.

continuing the search, hence pruning away the search tree without needing to find a conflict. This leads to the simple code in Algorithm 1, where removing lines 9 to 12 results in the standard CDCL algorithm, and where we can assume, for simplicity, that $analyzeWitness()$ returns $\neg\alpha$. More sophisticated versions of $analyzeWitness$ are discussed in the next Section.

We can understand SDCL as a parameterized algorithm, since the use of different pruning predicates $P_\alpha(F)$ leads to distinct types of SDCL algorithms with possibly different underlying proof systems. In the following, we summarize the contributions of [17, 14] and explain the different pruning predicates and the corresponding redundancy notions they are related to.

Definition 5. *Given formula F and a (partial) assignment α, the* **positive reduct** $p_\alpha(F)$ *is the formula $\neg\alpha \wedge G$, where $G = \{touched_\alpha(D) \mid D \in F \text{ and } \alpha \models D\}$.*

That is, we only consider clauses satisfied by α, and among them, only the literals that are assigned. Although the results in [17] imply that $p_\alpha(F)$ is a valid pruning predicate, we also provide here a simple proof. The proof illustrates the underlying idea behind SDCL: whenever the solver has a partial assignment α with $p_\alpha(F)$ satisfiable, if there is a model of F extending α, there is also a model satisfying $\neg\alpha$. Hence, it is safe to prune the current branch and instead look for a model satisfying $\neg\alpha$.

Theorem 2. *Given a formula F and a (partial) assignment α, if $p_\alpha(F)$ is satisfiable, then the clause $\neg\alpha$ is redundant with respect to F.*

Proof. Let us assume that $p_\alpha(F)$ is satisfiable. We only need to prove that if F is satisfiable, $F \wedge \neg\alpha$ is also satisfiable. Consider a total model β of F and a model γ of $p_\alpha(F)$. If $\beta \models \neg\alpha$ we are done. Otherwise, we know that $\beta = \alpha$ on the variables defined in α.

We can now modify β so that it coincides with γ on all variables of α, and is unchanged otherwise. This new assignment, let us call it $\hat{\beta}$, satisfies $\neg\alpha$ because $\gamma \models \neg\alpha$. To prove that $\hat{\beta}$ also satisfies F, let us take a clause $D \in F$. If $\alpha \models D$, then $touched_\alpha(D)$ is a clause in $p_\alpha(F)$ and hence is satisfied by γ and also by $\hat{\beta}$, because $touched_\alpha(D)$

only contains variables in α. Since $touched_\alpha(D)$ is a subclause of D, we have that $\hat{\beta} \models D$. If $\alpha \not\models D$, since $\beta \models D$ and we know that $\beta = \alpha$ on the variables defined in α, then β satisfies a literal of D not defined in α. Since $\hat{\beta}$ differs from β only on variables defined in α, we have that $\hat{\beta} \models D$. □

A precise characterization of the redundancy achieved by $p_\alpha(F)$ is given in [17]: $p_\alpha(F)$ is satisfiable if and only if $\neg\alpha$ is set-blocked in F.

Definition 6. *A clause C is* **set-blocked** *in a formula F if there exists a subset $L \subseteq C$ such that, for every clause D containing the negation of some literal in L, the clause $(C \setminus L) \vee \neg L \vee D$ contains two complementary literals.*

The results in [17] imply that a proof system based on resolution and set-blocked clauses has polynomial size proofs for the pigeon hole principle. It is also known [15] that set-blocked clauses are a particular case of SPR clauses. If one wants to obtain the full power of SPR clauses, the following pruning predicate is needed:

Definition 7. *Given formula F and a (partial) assignment α, the* **filtered positive reduct** *$f_\alpha(F)$ is the formula $\neg\alpha \wedge G$, where $G = \{touched_\alpha(D) \mid D \in F \text{ and } F \wedge \alpha \not\vdash_1 untouched_\alpha(D)\}$.*

Again, a precise characterization of the power of $f_\alpha(F)$ is known [14]: $f_\alpha(F)$ is satisfiable if and only if $\neg\alpha$ is SPR with respect to F. Despite being harder to compute than $p_\alpha(F)$, the fact that $f_\alpha(F)$ is a subset of the clauses in $p_\alpha(F)$ makes it easier to check for satisfiability. Finally, another pruning predicate is given in [14] that achieves the full power of PR clauses, but it is not considered to be practical. We close this sequence of pruning predicates and their corresponding redundancy characterization with a novel pruning predicate and its corresponding redundancy notion.

Definition 8. *Given formula F and a (partial) assignment α, the* **purely positive reduct** *$pp_\alpha(F)$ is the formula $\neg\alpha \wedge G$, where $G = \{satisfied_\alpha(D) \mid D \in F \text{ and } \alpha \models D\}$.*

Since all clauses in $pp_\alpha(F)$ are subclauses of clauses in $p_\alpha(F)$, whenever $pp_\alpha(F)$ is satisfiable, $p_\alpha(F)$ is also satisfiable. This proves that $pp_\alpha(F)$ is a pruning predicate, but we can be more precise about the notion of redundancy it corresponds to.

Definition 9. *We say that a literal $l \in C$* **blocks** *C in F if an only if for every clause D in F containing literal $\neg l$, resolution between C and D gives a tautology. A clause C is* **blocked** *in F if and only if there exists some literal $l \in C$ that blocks C in F.*

Theorem 3. *Given a formula F and an assignment α, the formula $pp_\alpha(F)$ is satisfiable if and only if the clause $\neg\alpha$ is blocked in F.*

Proof. *Left to right:* let β be a model of $pp_\alpha(F)$. Since $\beta \models \neg\alpha$, we can take any literal $\neg l$ in $\neg\alpha$ satisfied by β. We now prove that $\neg l$ blocks $\neg\alpha$ in F. Let us consider a clause of the form $l \vee C \in F$. Since $l \in \alpha$ we have that $\alpha \models l \vee C$, and hence there is a clause of the form $l \vee satisfied_\alpha(C)$ in $pp_\alpha(F)$. Since $\beta \models pp_\alpha(F)$ and $\beta \models \neg l$, necessarily $\beta \models satisfied_\alpha(C)$. This means that C contains a literal from α different from l, and hence if we apply resolution between the clause $\neg\alpha$ and $l \vee C$ we obtain a tautology.

Right to left: Assume w.l.o.g. that the clause $\neg\alpha$ is blocked w.r.t. $\neg l$ in F. We prove that $\hat{\alpha} := \alpha \setminus \{l\} \cup \{\neg l\}$ is a model of $pp_\alpha(F)$. It is obvious that $\hat{\alpha}$ satisfies the clause $\neg\alpha \in pp_\alpha(F)$. Any other clause $D \in pp_\alpha(F)$ is of the form $satisfied_\alpha(C)$ for some $C \in F$ such that $\alpha \models C$. There are now in principle two cases:

i. if D is not the unit clause l, it necessarily contains a literal from α different from l, and hence $\hat{\alpha}$ satisfies it.
ii. If D is the unit clause l, this means that clause $C \in F$ does not contain any literal from α except for l. Thus, applying resolution between $\neg\alpha$ and C cannot give a tautology, contradicting the fact that $\neg\alpha$ is blocked w.r.t $\neg l$ in F. Hence, this case cannot take place.

□

In general, checking the satisfiability of a reduct can be an NP-hard problem. We now prove that the satisfiability of $pp_\alpha(F)$ can be checked in polynomial time.

Theorem 4. *Given a formula F and an assignment α, the formula $pp_\alpha(F)$ is unsatisfiable if and only all literals in α are unit clauses of $pp_\alpha(F)$.*

Proof. *Right to left.* Since $\neg\alpha$ is a clause of $pp_\alpha(F)$, if all literals in α are unit clauses in $pp_\alpha(F)$, it is clearly unsatisfiable.

Left to right. Let us assume that not all literals of α are unit clauses of $pp_\alpha(F)$ and prove that $pp_\alpha(F)$ is satisfiable. Let $l \in \alpha$ be a literal that is not a unit clause in $pp_\alpha(F)$. We claim that the assignment $\beta := (\alpha \setminus \{l\}) \cup \{\neg l\}$ satisfies $pp_\alpha(F)$. This is easy to see: first of all, β satisfies $\neg\alpha$. Any other clause C of $pp_\alpha(F)$ is a non-empty disjunction of literals in α. Since we know that C cannot be the unit clause l, it necessarily contains a literal of α different from l, and hence β satisfies it. □

Corollary 1. *For any formula F and assignment α, checking the satisfiability of $pp_\alpha(F)$ can be done in polynomial time.*

We want to remark that this result is not totally unexpected. The definition of blocked clause already gives us a polynomial procedure for checking whether a clause is blocked. The above theorem presents another polynomial procedure for that.

We finish this section with one important remark about the computation of reducts in SDCL: we need to add all already computed redundant clauses in the reduct computation when trying to find additional ones. Let us show why not doing this is incorrect. Given the satisfiable formula $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_4)$, the SDCL solver might first build the assignment $\alpha = \{x_1, \neg x_2\}$. Its positive reduct is $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$, which is satisfiable, and hence we learn the redundant clause $\neg x_1 \vee x_2$. If the solver now builds the assignment $\{\neg x_1, x_2\}$, the positive reduct w.r.t F is $(x_1 \vee \neg x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$, which is again satisfiable and allows us to learn the clause $x_1 \vee \neg x_2$. However, adding the two learned redundant clauses to F makes it unsatisfiable. The solution is to build the second positive reduct w.r.t. F conjuncted with the first learned redundant clause. The corresponding reduct is $(x_1 \vee \neg x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$, which is now unsatisfiable and hence does not allow us to learn the second redundant clause.

A natural question that arises now is whether we also need to add all clauses that were derived using CDCL-style conflict analysis in a reduct. The answer is that we do not need to do so. The reason is that, given two formulas $G_1 \equiv G_2$, it holds that C is redundant w.r.t. G_1 if and only if C is redundant w.r.t G_2. Now, if the current formula that the SDCL solver has in its database is $F \wedge L \wedge R$, where F is the original formula, L are the lemmas derived by CDCL-style conflict analysis and R are the learned redundant clauses, it holds that $F \wedge L \wedge R \equiv F \wedge R$. Therefore, it is sufficient to compute redundant clauses w.r.t. $F \wedge R$ only. Having said that, it is better to compute redundant clauses w.r.t $F \wedge R \wedge U$, where U denotes CDCL-derived unit clauses, because it results in smaller reducts and faster sub-solver calls. Note that for correctness, clauses in R are never deleted. This design decision prevents us from using off-the-shelf proof checkers like dpr-trim[1]. However, as we mention at the end of Section 5, this checker can be easily adapted.

4 Minimizing SDCL Learned Clauses

In Algorithm 1, we considered the function *AnalyzeWitness* to always return $\neg\alpha$, which was correct due to the results presented in Section 3. However, adding the negation of the whole assignment results in a very large clause, and it is not a surprise that this is far from being useful in practice. Already in [17] it was proven that

[1] https://github.com/marijnheule/dpr-trim

one can learn a much shorter clause: the negation of all decisions in α. We provide a simple proof that we use to justify that learning other clauses is also correct:

Theorem 5. *Let F be a formula and C a clause that is redundant with respect to F. Any clause D obtained via resolution steps from $F \wedge C$ is also redundant with respect to F.*

Proof. Let us assume that F is satisfiable and prove that $F \wedge D$ also is. Since C is redundant w.r.t. F we know that $F \wedge C$ is satisfiable. We know that resolution generates logical consequences, and hence any model of $F \wedge C$ is also a model of $F \wedge C \wedge D$, which proves that $F \wedge D$ is satisfiable. □

It is well known that, if α is an assignment, starting from $\neg\alpha$ one can apply a series of resolution steps in order to derive a clause that only consists of decisions. If $\neg\alpha$ is redundant, the theorem proves that the decision-only clause is also redundant. However, learning the negation of all decisions is not the ideal situation for at least two reasons. The first one is that, according to experience from CDCL SAT solving, forcing the solver to learn clauses that only contain decisions leads to very poor performance in practice. It is certainly true that these clauses are small, but that is probably their only good property. The second reason is that not all decisions in α need to be present in the redundant clause. Similarly to what happens in CDCL, where usually not all decisions are responsible for a conflict, here not all decisions are responsible for the pruning predicate to be satisfiable. In order to fix these two issues, we modify *AnalyzeWitness* so that it finds the smallest subset $\gamma \subseteq \alpha$ for which $P_\gamma(F)$ is satisfiable. This allows us to learn the hopefully much shorter clause $\neg\gamma$ and address one of the open problems mentioned in [16]: "checking if a subset of a conflict clause is propagation redundant with respect to the formula under consideration."

Example 3. Consider $F = (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_5) \wedge (\neg x_1 \vee \neg x_4 \vee x_5) \wedge (x_2 \vee x_4 \vee \neg x_5) \wedge (x_3 \vee x_6 \vee \neg x_5)$ and assignment $\alpha = \{x_1, x_4, x_5, \neg x_2\}$, where x_5 is the only non-decision. The positive reduct $p_\alpha(F)$ is $(\neg x_1 \vee \neg x_4 \vee \neg x_5 \vee x_2) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_5) \wedge (\neg x_1 \vee \neg x_4 \vee x_5) \wedge (x_2 \vee x_4 \vee \neg x_5)$ and is satisfiable. Hence we could learn the redundant clause $\neg x_1 \vee \neg x_4 \vee x_2$ consisting of the negation of the decisions. However the subset $\gamma = \{x_1, \neg x_2\} \subseteq \alpha$ also has satisfiable positive reduct: $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$ and hence we could learn the shorter clause $\neg x_1 \vee x_2$.

4.1 Hardness of Minimization for $pp_\alpha(F)$

Unfortunately, as we prove in this section, the problem of finding a small subassignment whose reduct is satisfiable is difficult even for the purely positive reduct. This is somehow surprising because the satisfiability of this reduct can be determined by a very simple polynomial-time procedure. Since finding a small subassignment for which the purely positive reduct is satisfiable is equivalent to finding a small subclause that is blocked, we state our results as follows: given a blocked clause, finding the smallest subclause that is still blocked is hard. We start by defining a decisional version of this problem.

Definition 10. *Min-Blocked-Clause: given a formula F, a clause C that is blocked in F and an integer $k \geq 0$, we want to know whether there exists a clause $C' \subseteq C$ of size at most k that is blocked in F.*

In order to prove that Min-Blocked-Clause is NP-hard, we use the following well-known NP-hard problem:

Definition 11. *Hitting-Set: given a set S, a collection T of subsets of S and an integer $k \geq 0$, we want to determine whether there exists a subset $S' \subseteq S$ of size at most k such that every subset in T has non-empty intersection with S'.*

Theorem 6. *Min-Blocked-Clause is NP-hard.*

Proof. Let us consider (S, T, k) an instance of Hitting-Set, that is, S is a set $\{p_1, p_2, \cdots, p_N\}$, $T = \{T_1, T_2, \cdots, T_m\}$ is a collection of subsets of S with T_i of the form $\{x_1^i, x_2^i, \cdots, x_{\ell_i}^i\}$, and an integer k. We construct the following

instance $(F, C, \hat{k})$ of Min-Blocked-Clause: let F contain, for $1 \leq i \leq N$, the unit clause p_i and the clause $p_0 \vee x_1^i \vee x_2^i \cdots \vee x_{\ell_i}^i$. The clause C is $\neg p_0 \vee \neg p_1 \cdots \vee \neg p_N$, which is blocked w.r.t F due to blocking literal $\neg p_0$. This is true because applying resolution with a clause $p_0 \vee x_1^i \vee x_2^i \cdots \vee x_{\ell_k}^i$ results in a tautology since all x_k^i belong to $\{p_1, \cdots, p_N\}$. Finally, we set $\hat{k} := k + 1$ This reduction clearly runs in polynomial time and has polynomial size. Let us now prove that (S, T, k) is a positive instance of Hitting-Set if and only if $(F, C, \hat{k})$ is a positive instance of Min-Blocked-Clause.

Left to right. Let $S' \subseteq S$ be a subset of size $\leq k$ that has non-empty intersection with every subset in T. We claim that $\neg p_0 \vee \neg S'$ is a subclause of C with size $\leq k + 1$ that is blocked in F, being $\neg p_0$ its blocking literal. It is trivial to check that it is a subset of $C = \neg p_0 \vee \neg p_1 \vee \cdots \vee \neg p_N$ of size $\leq k + 1$. Finally, we know that it is blocked in F because the only clauses containing literal p_0 are of the form $C_i := p_0 \vee x_1^i \vee x_2^i \cdots \vee x_{\ell_i}^i$ for some $1 \leq i \leq N$. Since $T_i = \{x_1^i, x_2^i, \cdots, x_{\ell_i}^i\}$ has non-empty intersection with S', resolution between $\neg p_0 \vee \neg S'$ and C_i is a tautology.

Right to left. Let us assume that there is a clause $C' \subseteq C$ of size $k + 1$ that is blocked in F. Note that the only possible blocking literal is $\neg p_0$, because of the existence of unit clauses p_i for $1 \leq i \leq N$ in F. Thus, C' is of the form $\neg p_0 \vee D$. If we let S' be the set consisting of the negation all literals in D, we have that S' is set of size $\leq k$ with $S' \subseteq S$ and non-empty intersection with any T_i for $1 \leq i \leq N$. Checking that is size is at most k and the fact that $S' \subseteq S$ is easy. Now, consider any i and take T_i, which is of the form $\{x_1^i, x_2^i, \cdots, x_{\ell_i}^i\}$. Since F contains clause $p_0 \vee x_1^i \vee x_2^i \cdots \vee x_{\ell_k}^i$ and $C' = \neg p_0 \vee D$ is blocked in F with blocking literal p_0, resolution between these two clauses must be a tautology. This means that T_i and S' have non-empty intersection. □

The fact that this problem is NP-hard should not prevent us from trying to solve it. One natural attempt is to cast its optimization version as a MaxSAT problem. In Section 4.3 we explain how, given a formula F and an assignment α, we can define a partial MaxSAT formula $mpp_\alpha(F)$ whose optimal solution corresponds to a smallest $\gamma \subseteq \alpha$ such that $pp_\gamma(F)$ is satisfiable. This, in turn, will lead us to a smallest clause $\neg\gamma \subseteq \neg\alpha$ that is blocked in F.

4.2 Hardness of Minimization for $p_\alpha(F)$

In the following, we prove an equivalent negative result for the positive reduct. Let us first formalize it as a decision problem.

Definition 12. *Trail-Minimization: given a formula F, an assignment α and an integer $k \geq 0$, we want to know whether there is a subset $\gamma \subseteq \alpha$ of size k such that $p_\gamma(F)$ is satisfiable.*

Note that in the rest of the paper we focus on the positive reduct of the formula, and hence our approach allows us to obtain (short) set-blocked clauses. Before introducing the NP-hard problem that we use to prove the NP-hardness of Trail-Minimization, we need one definition:

Definition 13. *Given α and β two assignments over the same variables $\{x_1, \ldots, x_n\}$, we say that $\beta < \alpha$ if $\beta \neq \alpha$ and for each $\neg x_i \in \alpha$ we also have that $\neg x_i \in \beta$.*

In other words, $\beta < \alpha$ if, considering an assignment as a sequence of n bits, the sequence of bits of β is pointwise smaller than the one of α.

Definition 14. *Smaller-Model[19]: given a formula F and a total model α of F, we want to know whether there is a total model β of F such that $\beta < \alpha$.*

In [19] it is proved that Smaller-Model is NP-hard. Hence, a polynomial reduction from Smaller-Model to Trail-Minimization proves that the latter is also NP-hard.

Theorem 7. *Trail-Minimization is NP-hard.*

Proof. Given (F, α) an instance of Smaller-Model, we can partition $\alpha = \alpha^+ \cup \alpha^-$, where α^+ contains all positive literals in α, and α^- contains all negative literals. The reduction amounts to constructing an instance of Trail-Minimization as follows: the formula is $\hat{F} := F \cup \alpha^-$ (that is, we add to F all negative literals in α as unit clauses), the assignment is $\hat{\alpha} := \alpha$ and the integer $k := |\alpha|$. This can be computed in polynomial time and has polynomial size. Let us now check that (F, α) is a positive instance of Smaller-Model if and only if $(\hat{F}, \hat{\alpha}, k)$ is a positive instance of Trail-Minimization.

Left to right: we know, by definition of Smaller-Model, that there exists an assignment $\beta \models F$ such that $\beta < \alpha$. Since obviously $\hat{\alpha} \subseteq \hat{\alpha}$ and $|\hat{\alpha}| = k$, if we prove that $\beta \models p_{\hat{\alpha}}(\hat{F})$ we are done. Clause $\neg\alpha$ is satisfied by β because $\beta \neq \alpha$. Since $\hat{\alpha}$ is a total assignment, we have that $touched_{\hat{\alpha}}(C) = C$ for any clause $C \in \hat{F}$, hence any clause in $p_{\hat{\alpha}}(\hat{F})$ is either (i) a unit clause consisting of a literal in α^-, which is satisfied by β because $\beta < \alpha$ implies that $\beta \supseteq \alpha^-$ or (ii) a clause $C \in F$, which is of course satisfied by β since β is a model of F.

Right to left: the only subset of $\hat{\alpha}$ of size k is $\hat{\alpha}$ itself. Let us assume that $p_{\hat{\alpha}}(\hat{F})$ is satisfied by a model β. Since $\neg\alpha$ is a clause in $p_{\hat{\alpha}}(\hat{F})$, we know that $\beta \neq \alpha$. Also, since $p_{\hat{\alpha}}(\hat{F})$ contains all negative literals of α as unit clauses, we know that $\beta \supseteq \alpha^-$. Altogether, this proves that $\beta < \alpha$. The only missing piece is to prove that β satisfies F. This is not difficult to see: since α is a model of F, and $touched_{\alpha}(C) = C$ for any clause C, all clauses in F belong to $p_{\hat{\alpha}}(\hat{F})$ and β necessarily satisfies them. □

4.3 A MaxSAT Encoding for Redundant Clause Minimization

Knowing that Trail-Minimization and Min-Blocked-Clause are difficult optimization problems, and being somehow similar to SAT, it is very natural to try solving them using MaxSAT. We now first describe in detail the encoding for Trail-Minimization. Later, we present the encoding of Min-Blocked-Clause as a particular case.

Given a formula F, and an assignment α, we describe a partial MaxSAT formula $mp_{\alpha}(F)$ whose solutions correspond to a smallest $\gamma \subseteq \alpha$ such that $p_{\gamma}(F)$ is satisfiable. Before formally defining $mp_{\alpha}(F)$, let us explain the intuition behind it. The main idea is that we have to determine which literals we can remove from α, giving a new assignment γ, such that $p_{\gamma}(F)$ is satisfiable. For each literal l in α, we add an additional variable r_l that indicates whether l is removed. Hence, a truth assignment over these variables induces an assignment $\gamma \subseteq \alpha$. The key point is to construct a formula such that when restricted with r_l's, it is essentially equivalent to $p_{\gamma}(F)$.

More formally, let us assume $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$. Note that α is a set of literals, not variables. We introduce three sets of additional variables:

- $\{r_1, r_2, \ldots, r_m\}$: indicate whether α_i is **r**emoved from α.
- $\{p_1, p_2, \ldots, p_m\}$: replace "**p**ositive" occurrences of α_i in $p(F, \alpha)$.
- $\{n_1, n_2, \ldots, n_m\}$: replace "**n**egative" occurrences of α_i in $p(F, \alpha)$.

Given a clause C, we denote by $\hat{C}$ the result of replacing in C, for $i = 1 \ldots m$, every occurrence of literal α_i by p_i and every occurrence of literal $\neg\alpha_i$ by n_i.

Our Partial MaxSAT formula $mp_{\alpha}(F)$ contains the following hard formulas (that can be easily converted into clauses), that enforce the semantics of the r, p and n variables:

$$\begin{array}{rcll} r_i & \rightarrow & \neg p_i & \text{for all } i = 1 \dots m \\ r_i & \rightarrow & \neg n_i & \text{for all } i = 1 \dots m \\ \neg r_i & \rightarrow & p_i = \alpha_i & \text{for all } i = 1 \dots m \\ \neg r_i & \rightarrow & n_i = \neg\alpha_i & \text{for all } i = 1 \dots m \end{array}$$

Intuitively, if r_i is false, and hence we do not remove α_i from the assignment, then p_i is equivalent to α_i and n_i is equivalent to $\neg\alpha_i$. Otherwise, if r_i is removed, we force p_i and n_i to be false.

The rest of $mp_\alpha(F)$ is constructed by iterating over all clauses of $p_\alpha(F)$. For each clause $C \in p_\alpha(F)$, we add a set of hard clauses to $mp_\alpha(F)$, constructed as follows. If C is the clause $\neg\alpha$, we add the hard clause $\widehat{\neg\alpha}$. Otherwise C is of the form $S \vee D$, where S is the non-empty set of literals satisfied by α and D contains the remaining literals, which are touched, but not satisfied by α. The clauses to be added are:

$$\{\hat{S} \vee \hat{D} \vee r_i \mid i = 1 \dots m \text{ and } \alpha_i \in S\}$$

The idea here is that if we remove all literals in S from α, then C would not be satisfied and hence it should not appear in the positive reduct. The addition of the r_i's in the clauses guarantee that, if all of them are removed, and hence all r_i's are set to true, these clauses are all satisfied by the r_i's and hence they do not constrain the formula at all. On the other hand, if some literal in S is not removed, then the corresponding r_i is false and we essentially have the clause $\hat{S} \vee \hat{D}$, that is what we wanted to impose.

We want to note that we can obtain a smaller formula by, instead of adding multiple clauses of the form of $\hat{S} \vee \hat{D} \vee r_i$, introducing one auxiliary variable a_C for each clause $C = S \vee D$ and adding the clauses:

$$\begin{array}{ll} \hat{S} \vee \hat{D} \vee a_C & \\ \neg r_i \rightarrow \neg a_C & \text{for all } i = 1 \dots m \text{ such that } \alpha_i \in S \end{array}$$

Apart from these hard clauses and the hard ones imposing the semantics of r, p and n, our formula $mp_\alpha(F)$ is completed with the set of soft unit clauses $\{r_i \mid i = 1 \dots m\}$, expressing that we want to remove as many literals as possible while still satisfying the rest of the formula, which are hard clauses.

Example 4. Let us revisit Example 3, where $\alpha = \{x_1, x_4, x_5, \neg x_2\}$ had $p_\alpha(F)$ satisfiable, but there was a smaller subset $\gamma = \{x_1, \neg x_2\}$ for which $p_\gamma(F)$ was also satisfiable. We use this example to illustrate our encoding. Let us consider that the variables related with x_i are p_i, n_i, r_i for $i \in \{1, 2, 4, 5\}$. We only show the hard clauses in $mp_\alpha(F)$ that are constructed from $p_\alpha(F)$. The implications defining the semantics of p, n, r are ignored, as well as the soft clauses, since those should be easy to understand.

$\mathbf{p_\alpha(F)}$	$\mathbf{mp_\alpha(F)}$
$\neg x_1 \vee \neg x_4 \vee \neg x_5 \vee x_2$	$n_1 \vee n_4 \vee n_5 \vee n_2$
$x_1 \vee x_2 \vee x_4$	$p_1 \vee n_2 \vee p_4 \vee r_1$
	$p_1 \vee n_2 \vee p_4 \vee r_4$
$x_1 \vee \neg x_2 \vee x_5$	$p_1 \vee p_2 \vee p_5 \vee r_1$
	$p_1 \vee p_2 \vee p_5 \vee r_2$
	$p_1 \vee p_2 \vee p_5 \vee r_5$
$\neg x_1 \vee \neg x_4 \vee x_5$	$n_1 \vee n_4 \vee p_5 \vee r_5$
$x_2 \vee x_4 \vee \neg x_5$	$n_2 \vee p_4 \vee n_5 \vee r_4$

Note that any literal x_1 is replaced by p_1 since $x_1 \in \alpha$. On the other hand, any literal x_2 is replaced by n_2 because $\neg x_2 \in \alpha$. The interesting fact is that if we set r_4, r_5 to true and r_1, r_2 to false, which means that we are removing x_4 and x_5 from α (hence obtaining γ) and propagate the implications, the hard clauses in $mp_\alpha(F)$ become equivalent to $p_\gamma(F)$. For example, take the first clause $n_1 \vee n_4 \vee n_5 \vee n_2$. Setting r_4 and r_5 to true causes the implications to

unit propagate $\neg n_4$ and $\neg n_5$. Hence the clause is equivalent to $n_1 \vee n_2$. However, setting r_1 to false makes $n_1 = \neg x_1$ and setting r_2 to false makes $n_2 = x_2$. All in all, the clause is equivalent to $\neg x_1 \vee x_2$, which is the first clause of $p_\gamma(F)$.

If we take clause $n_2 \vee p_4 \vee n_5 \vee r_4$ we can see that it is satisfied due to r_4 and thus is not constraining the other variables. This is as expected, because if we remove x_4 from α, it no longer satisfies the clause $x_2 \vee x_4 \vee \neg x_5$ and hence it should not appear in the reduct.

Finally, the last case is a clause like $p_1 \vee n_2 \vee p_4 \vee r_1$. In this case r_1 is false and hence the last literal in the clause disappears. Also, since r_4 is true, it makes p_4 false due to the implications, and r_1, r_2 being false unit propagates $p_1 = x_1$ and $n_2 = x_2$, hence making the clause equivalent to $x_1 \vee x_2$, which is precisely the clause that appears in $p_\gamma(F)$. All in all, if we set the r variables to the appropriate values we can obtain the positive reduct of any subset of α. Below, we formally prove that this encoding is indeed correct.

Theorem 8. *Given a formula F and an assignment $\alpha = \{\alpha_1, \ldots, \alpha_m\}$, it holds that the smallest subset $\gamma \subseteq \alpha$ such that $p_\gamma(F)$ is satisfiable has size $m - k$ if and only if the optimal solution to $mp_\alpha(F)$ satisfies k soft clauses.*

Proof. We prove something slightly stronger: there exists $\gamma \subseteq \alpha$ of size $m - k$ such that $p_\gamma(F)$ is satisfiable if and only if there exists an assignment that satisfies all hard clauses in $mp_\alpha(F)$ and exactly k soft clauses.

Left to right: let us consider $\gamma \subseteq \alpha$ of size $m - k$ with $p_\gamma(F)$ satisfiable, and let δ be a model for it. We build an assignment the satisfies all hard clauses in $mp_\alpha(F)$ and exactly k soft clauses as follows. The first remark is that $mp_\alpha(F)$ only consists of the variables r_i, p_i, n_i and the ones appearing in α_i, for $i = 1 \ldots m$ and hence we have to build an assignment β over those. For $i = 1 \ldots m$ we add r_i to β if $\alpha_i \notin \gamma$, and add $\neg r_i$ otherwise. Since there are k literals α_i not belonging to γ, it is clear that β satisfies exactly k soft clauses. The assignment β is completed by making it coincide with δ on the variables of γ and take arbitrary values for the variables of $\alpha \setminus \gamma$. If we now unit propagate these values on the implications that define the semantics of r, p and n, we complete β to define values for all p_i and n_i.

Let us now see that β satisfies the hard clauses in $mp_\alpha(F)$. The implications defining the semantics of the variables are obviously satisfied. Clause $\widehat{\neg\alpha}$ is also satisfied: we know that this clause is of the form $n_1 \vee n_2 \vee \ldots \vee n_m$. Since $\delta \models \neg\gamma$, there is a literal $\alpha_t \in \gamma$ such that $\delta \models \neg\alpha_t$. By the definition of β, we know that $\neg r_t \in \beta$ and hence the formula $\neg r_t \rightarrow n_t = \neg\alpha_t$ propagates n_t to be true in β and hence satisfy $\widehat{\neg\alpha}$.

Let us now take another clause $C \in mp_\alpha(F)$, which is necessarily of the form $\hat{S} \vee \hat{D} \vee r_i$, with $S \vee D \in p_\alpha(F)$ and r_i be such that $\alpha_i \in S$. If $\beta \models r_i$ we are done. Otherwise, it is because $\alpha_i \in \gamma$. Hence, the clause $S \vee D$ is satisfied by γ due to literal $\alpha_i \in S$ and $p_\gamma(F)$ contains the clause $touched_\gamma(S \vee D)$. Thus, $\delta \models touched_\gamma(S \vee D)$. Let us consider that case where δ satisfies $\alpha_j \in touched_\gamma(S \vee D)$ (the other case is that is satisfies some $\neg\alpha_j$ and the proof is similar). Since $\alpha_j \in \gamma$, we have that $r_j \notin \beta$ and the formula $\neg r_j \rightarrow p_j = \alpha_j$ guarantees that $\beta \models p_j$. We only have to realize that p_j is a literal in $\hat{S} \vee \hat{D}$, to conclude that $\beta \models C$.

Right to left: let us consider an assignment β that satisfies all hard clauses in $mp_\alpha(F)$ and exactly k soft clauses. We build a subset $\gamma \subseteq \alpha$ of size $m - k$ such that $p_\gamma(F)$ is satisfiable. As expected, γ is constructed by removing from α all α_i such that $\beta \models r_i$. It is obvious that $|\gamma| = m - k$, because β satisfies exactly k unit clauses of the form r_i.

In order to prove that $p_\gamma(F)$ is satisfiable, let us build an assignment δ that coincides with β over all variables in γ and prove that it is a model. The first clause in $p_\gamma(F)$ to consider is $\neg\gamma$. Since $\beta \models n_1 \vee \ldots \vee n_m$ and it satisfies the clauses $r_i \rightarrow \neg n_i$ it necessarily satisfies some n_i such that r_i is false. Due to the clause $\neg r_i \rightarrow n_i = \neg\alpha_i$, it also satisfies $\neg\alpha_i$. Since r_i is false in β we have that $\alpha_i \in \gamma$ and hence, by the definition of δ, it satisfies $\neg\alpha_i$. This proves that $\delta \models \neg\gamma$.

Let us now take another clause in $p_\gamma(F)$, which is necessarily of the form $touched_\gamma(C)$ for some $C \in F$ such that $\gamma \models C$. Since α is a superset of γ, obviously $\alpha \models C$, and hence a clause of the form $touched_\alpha(C)$ belongs to

$p_\alpha(F)$. This clause in $p_\alpha(F)$ is of the form $S \vee D$, with S containing all literals satisfied by α, and thus we have in $mp_\alpha(F)$ hard clauses of the form $\hat{S} \vee \hat{D} \vee r_i$ for every i with $\alpha_i \in S$. If $\gamma \models C$ it is because it satisfies some $\alpha_i \in C$ with r_i being false in β. Hence, the existence of the clause $\hat{S} \vee \hat{D} \vee r_i$ implies that $\beta \models \hat{S} \vee \hat{D}$. We know that $\hat{S} \vee \hat{D}$ is a disjunction of positive p's and n's literals. Let us assume that it satisfies some p_k (the case n_k is similar). Note that r_k has to be false because otherwise the implications force p_k to be false. Hence β satisfies some α_k such that r_k is false and hence $\alpha_k \in \gamma$, which means that δ also satisfies α_k because they coincide over γ. Since $\alpha_k \in \gamma$, it belongs to $touched_\gamma(C)$ which is the clause that we wanted δ to satisfy. □

As we promised at the beginning of this section, we now define a partial MaxSAT formula $mpp_\alpha(F)$ whose optimal solution corresponds to a smallest $\gamma \subseteq \alpha$ such that $pp_\gamma(F)$ is satisfiable. This, in turn, will lead us to a smallest clause $\neg\gamma \subseteq \neg\alpha$ that is blocked in F, that is, solving the Min-Blocked-Clause problem.

$mpp_\alpha(F)$ is almost identical to $mp_\alpha(F)$. Exactly the same fresh variables are introduced, as well as the implications that define its semantics. Soft clauses are also identical. The only difference is that, instead of iterating over clauses in $p_\alpha(F)$ to construct the remaining hard clause, we now iterate over clauses in $pp_\alpha(F)$, as one would expect. For each such clause, we follow the same procedure as in the construction of $mp_\alpha(F)$, with the only difference that we know that the set D is always empty. However, this has no impact on the encoding.

Theorem 9. *Given a formula F and an assignment $\alpha = \{\alpha_1, \ldots, \alpha_m\}$, it holds that the smallest subset $\gamma \subseteq \alpha$ such that $pp_\gamma(F)$ is satisfiable has size $m - k$ if and only if the optimal solution to $mpp_\alpha(F)$ satisfies k soft clauses.*

Proof. The proof follows exactly along the same line of the one for Theorem 8. We reproduce and adapt it here again for completeness. Again, we prove something slightly stronger: there exists $\gamma \subseteq \alpha$ of size $m - k$ such that $pp_\gamma(F)$ is satisfiable if and only if there exists an assignment that satisfies all hard clauses in $mpp_\alpha(F)$ and exactly k soft clauses.

Left to right: let us consider $\gamma \subseteq \alpha$ of size $m - k$ with $pp_\gamma(F)$ satisfiable, and let δ be a model for it. We build an assignment the satisfies all hard clauses in $mpp_\alpha(F)$ and exactly k soft clauses as follows. The first remark is that $mpp_\alpha(F)$ only consists of the variables r_i, p_i, n_i and the ones appearing in α_i, for $i = 1 \ldots m$ and hence we have to build an assignment β over those. For $i = 1 \ldots m$ we add r_i to β if $\alpha_i \notin \gamma$, and add $\neg r_i$ otherwise. Since there are k literals α_i not belonging to γ, it is clear that β satisfies exactly k soft clauses. The assignment β is completed by making it coincide with δ on the variables of γ and take arbitrary values for the variables of $\alpha \setminus \gamma$. If we now unit propagate these values on the implications that define the semantics of r, p and n, we complete β to define values for all p_i and n_i.

Let us now see that β satisfies the hard clauses in $mpp_\alpha(F)$. The implications defining the semantics of the variables are obviously satisfied. Clause $\widehat{\neg\alpha}$ is also satisfied: we know that this clause is of the form $n_1 \vee n_2 \vee \ldots \vee n_m$. Since $\delta \models \neg\gamma$, there is a literal $\alpha_t \in \gamma$ such that $\delta \models \neg\alpha_t$. By the definition of β, we know that $\neg r_t \in \beta$ and hence the implication $\neg r_t \rightarrow n_t = \neg\alpha_t$ propagates n_t to be true in β and hence satisfy $\widehat{\neg\alpha}$.

Let us now take another clause $C \in mpp_\alpha(F)$, which is necessarily of the form $\hat{S} \vee r_i$, with $S \in pp_\alpha(F)$ and r_i be such that $\alpha_i \in S$. If $\beta \models r_i$ we are done. Otherwise, it is because $\alpha_i \in \gamma$. Hence, the clause S is satisfied by γ due to literal $\alpha_i \in S$ and $pp_\gamma(F)$ contains the clause $satisfied_\gamma(S)$. Thus, $\delta \models satisfied_\gamma(S)$, and let α_j be the literal satisfied by δ. Since $\alpha_j \in \gamma$, we have that $r_j \notin \beta$ and the formula $\neg r_j \rightarrow p_j = \alpha_j$ guarantees that $\beta \models p_j$. We only have to realize that p_j is a literal in $\hat{S}$ to conclude that $\beta \models C$.

Right to left: let us consider an assignment β that satisfies all hard clauses in $mpp_\alpha(F)$ and exactly k soft clauses. We build a subset $\gamma \subseteq \alpha$ of size $m - k$ such that $pp_\gamma(F)$ is satisfiable. As expected, γ is constructed by removing from α all α_i such that $\beta \models r_i$. It is obvious that $|\gamma| = m - k$, because β satisfies exactly k unit clauses of the form r_i.

Function `analyzeWitness` *(Assignment* $\alpha = (\alpha_1, \cdots, \alpha_m)$*, Formula* F*)*:
  $M := MaxSAT(mp_\alpha(F))$ `// M is a solution, not necessarily optimal`
  $C := \bigvee\{ \neg\alpha_i \mid i \in \{1 \dots m\}$ and r_i false in $M\}$
  return *conflictAnalysis(C)*

Algorithm 2: *analyzeWitness* procedure using Max-SAT based minimization for the positive reduct $p_\alpha(F)$.

In order to prove that $pp_\gamma(F)$ is satisfiable, let us build an assignment δ that coincides with β over all variables in γ and prove that it is a model. The first clause in $pp_\gamma(F)$ to consider is $\neg\gamma$. Since $\beta \models n_1 \vee \ldots \vee n_m$ and it satisfies the clauses $r_i \rightarrow \neg n_i$ it necessarily satisfies some n_i such that r_i is false. Due to the clause $\neg r_i \rightarrow n_i = \neg\alpha_i$, it also satisfies $\neg\alpha_i$. Since r_i is false in β we have that $\alpha_i \in \gamma$ and hence, by the definition of δ, it satisfies $\neg\alpha_i$. This proves that $\delta \models \neg\gamma$.

Let us now take another clause in $pp_\gamma(F)$, which is necessarily of the form $satisfied_\gamma(C)$ for some $C \in F$ such that $\gamma \models C$. Since α is a superset of γ, obviously $\alpha \models C$, and hence a clause of the form $satisfied_\alpha(C)$ belongs to $pp_\alpha(F)$. This clause in $pp_\alpha(F)$ is of the form S, with S containing all literals satisfied by α, and thus we have in $mpp_\alpha(F)$ hard clauses of the form $\hat{S} \vee r_i$ for every i with $\alpha_i \in S$. Since $\gamma \models C$, it satisfies some $\alpha_i \in C$ with r_i being false in β. Hence, the existence of the clause $\hat{S} \vee r_i$ implies that $\beta \models \hat{S}$. We know that $\hat{S}$ is a disjunction of positive p's literals. Let p_k be one of the literals satisfied by β. Note that r_k has to be false because otherwise the implications force p_k to be false. Hence β satisfies some α_k such that r_k is false and hence $\alpha_k \in \gamma$, which means that δ also satisfies α_k because they coincide over γ. Since $\alpha_k \in \gamma$, it belongs to $satisfied_\gamma(C)$ which is the clause that we wanted δ to satisfy. □

□

4.4 Practical Remarks

The previous encoding would allow us to learn the redundant clause $C := \neg\gamma$. However, SDCL (see Algorithm 1) requires C to be asserting (that is, containing exactly one literal of the last decision level, and hence allowing it to unit propagate after backjumping). In order to achieve this property, we first observe that, being the negation of a subset of the current assignment, clause $\neg\gamma$ is a conflict. Hence, we can apply standard CDCL conflict analysis to it and obtain a clause that is asserting. For those familiar with SMT, this is essentially what DPLL(T)-based SMT solvers do when they analyze theory conflicts. Thanks to Theorem 5, we can guarantee that the final clause we obtain in this process is redundant and hence can be safely added. Moreover, as can be seen in Section 5, the size of this clause tends to be even smaller than $\neg\gamma$. In addition, this method allows us to learn clauses that are stronger than set-blocked clauses. A version of the *analyzeWitness* procedure used in Algorithm 1 that exploits Max-SAT based minimization is described in Algorithm 2.

Example 5. Let us consider $F = (\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_2) \wedge (x_0 \vee x_2) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (x_3 \vee x_6 \vee \neg x_5)$. Assume the SAT solver builds the assignment, from left to right, $\{\mathbf{x_0}, x_1, \neg x_2, x_4, \mathbf{x_5}\}$ where literals in bold are decisions. If we pick $\gamma = \{x_0, x_1, \neg x_2\}$, we can see that its reduct $(\neg x_0 \vee \neg x_1 \vee x_2) \wedge (\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_2) \wedge (x_0 \vee x_2)$ is satisfiable. This mean that we can learn $\neg x_0 \vee \neg x_1 \vee x_2$. Now, in two resolution steps with the reasons of x_1 and $\neg x_2$ which are $\neg x_0 \vee x_1$ and $\neg x_0 \vee \neg x_2$, respectively, we can derive the redundant clause $\neg x_0$. However, assignment x_0 does not have satisfiable positive reduct. In fact, clause $\neg x_0$ is not even SPR. It can be checked that it is indeed PR (a possible witness is $w = \{\neg x_0, x_2, x_3\}$). This shows that by combining the positive reduct with posterior resolution steps, we can obtain clauses with stronger redundancy properties than set-blocked clauses, which is the one obtained by using the positive reduct alone.

One final question that we want to address is whether, in an SDCL implementation, we should (i) first ask a SAT solver whether $p_\alpha(F)$ is satisfiable, and then, if this is the case, ask a MaxSAT solver to possibly find a smaller $\gamma \subseteq \alpha$ for which $p_\gamma(F)$ is also satisfiable, or (ii) directly ask a MaxSAT solver whether there exists a subset of $\gamma \subseteq \alpha$ for which $p_\gamma(F)$ is satisfiable. The following result sheds some light on this:

Proposition 1. *Given a formula F and an assignment α, if $mp_\alpha(F)$ has some solution, then $p_\alpha(F)$ is satisfiable.*

Proof. By Theorem 8, if $mp_\alpha(F)$ has some solution satisfying k soft clauses, we can build an assignment $\gamma \subseteq \alpha$ for which $p_\gamma(F)$ is satisfiable, and let β be a model for it. It is now easy to prove that $\delta := \beta \cup \alpha \backslash \gamma$ is a model for $p_\alpha(F)$. The first clause in $p_\alpha(F)$ is $\neg\alpha$, of which the clause $\neg\gamma \in p_\gamma(F)$ is a subclause and hence $\beta \models \neg\alpha$. This implies that $\delta \models \neg\alpha$. Now, any other clause C in $p_\alpha(F)$ is of the form $touched_\alpha(D)$ for some $D \in F$ such that $\alpha \models D$. Expressing $touched_\alpha(D)$ as $touched_\gamma(D) \vee touched_{\alpha\backslash\gamma}(D)$ helps in our reasoning. If $\gamma \models D$ then $touched_\gamma(D) \in p_\gamma(F)$ and hence β satisfies it. In this case $\delta \models C$. Otherwise, $\gamma \not\models D$ but since $\alpha \models D$ it has to be that $\alpha\backslash\gamma \models D$. This implies that $\alpha\backslash\gamma \models touched_{\alpha\backslash\gamma}(D)$ and hence $\delta \models C$. □

This result shows that by directly calling the MaxSAT solver on $mp_\alpha(F)$, the solver cannot learn more redundant clauses than if we call the SAT solver on $p_\alpha(F)$. Hence, it makes sense to first call the SAT solver, which should be faster and then, only if $p_\alpha(F)$ has been found to be satisfiable, call the MaxSAT solver to possibly learn a shorter redundant clause. If we make an analogy with CDCL, checking $p_\alpha(F)$ for satisfiable would be the equivalent of unit propagation and solving the MaxSAT formula $mp_\alpha(F)$ the equivalent of conflict analysis.

As expected, this strategy also applies when using the purely positive reduct, as the following result indicates.

Proposition 2. *Given a formula F and an assignment α, if $mpp_\alpha(F)$ has some solution, then $pp_\alpha(F)$ is satisfiable.*

Proof. The proof is identical to the proof of Theorem 2, replacing $touched$ by $satisfied$. □

5 Experimental Evaluation

5.1 Implementation

We implemented SDCL with the clause minimization techniques described in the previous section on top of the SAT solver MapleSAT [21]. In order to solve the MaxSAT queries, we have used EvalMaxSAT [2], an efficient solver that provides a very convenient C++ API.

The changes in Algorithm 1 are limited to $analyzeWitness$. Once we know that $p_\alpha(F)$ is satisfiable, we construct $mp_\alpha(F)$ and obtain the optimal solution with EvalMaxSAT. This induces a clause $\neg\gamma$, to which standard CDCL conflict analysis is applied in order to derive an asserting clause, which is learned and used to backjump.

This general idea is refined in different directions. First of all, we do not apply this procedure before every decision. Without redundant-clause minimization, this might be a bad decision design, since the length of the learned clause coincides with the decision level, and hence we should apply it as soon as possible. With clause minimization enabled, applying this procedure at high decision levels can still give short redundant clauses. Since, as a consequence of Proposition 1, we know that long assignments are more likely to produce redundant clauses, it makes sense to delay the check until the assignment is large enough. However, there is a certain trade-off because at high decision levels, $p_\alpha(F)$ and $mp_\alpha(F)$ are larger formulas and hence can be more difficult to solve. Our strategy relies on having a variable *dl_call* that corresponds to the decision level at which lines 9-12 of Algorithm 1 are executed. That is, if we are not at decision level *dl_call* those lines are ignored. This variable is updated as follows: when line 9 is executed, we compute the number of times $p_\alpha(F)$ has been found satisfiable and compare it with the total number of satisfiability checks. If this ratio is lower than a certain amount (15% in our implementation), we increment *dl_call* by one; if it is higher, we decrement it by one. The rationale for this strategy is to achieve a predefined ratio of successful calls but not invoking the technique too often.

The second refinement is that our final asserting clause is not always shorter than the clause obtained by negating all decisions. In those rare situations (it happens in around 1% of the learned redundant clauses), we

learn the only-decisions clause. A final refinement consists of only learning clauses of size at most 3. In our SDCL implementation, we cannot delete redundant clauses we have learned in SDCL unless we also delete all CDCL learned clauses that have been derived using them. This is why we have to be very cautious and only learn high-quality redundant clauses.

One final remark is that, unlike previous SDCL implementations [17, 14], we have not modified the decision heuristics of the solver. We believe that, due to our conflict minimization techniques, picking the exact right variable at low decision levels is not so critical.

5.2 Experimental Results

We have evaluated our system on the benchmarks used in [14, 26]. In order to assess the impact of our Max-SAT based minimization technique, we have presented in Figure 1 results about one execution of our system on a mutilated chess board benchmark of size 20. Data for other benchmarks follow along the same lines. On the left-hand histogram, a bar over the x-point 30 with height 10 means that, in 10% of the calls to minimization, the percentage (Size MaxSAT clause / Size Only-decisions clause)*100 is between 30% and 35%. That is, the size of the MaxSAT clause was around one third the size of the only-decisions clause. The histogram on the right plots the same data, but comparing the final asserting clause with respect to the only-decisions clause. One can observe that the percentage of reduction is important and comes from the MaxSAT invocation as well as from the subsequent conflict analysis call that returns the final asserting clause.

We also studied the cost of calling the SAT solver for checking the satisfiability of $p_\alpha(F)$ and the MaxSAT solver for processing $mp_\alpha(F)$. Our experiments revealed that the cost of the SAT solver call never exceeds 2% of the total runtime, whereas the calls to MaxSAT are more expensive and they can account for almost 30% of the total runtime.

Finally, we present in Table 1 results on the performance of our system compared to others. The 10 instances of the form mchess-n represent mutilated chess board problems of size n: given an $n \times n$ grid alternating black and white squares where two opposite corners have been removed, the question is whether or not the board can be covered with 2×1 dominoes. The problem is unsatisfiable: every domino covers a black and a white cell, but since the two removed cells have the same color, there are strictly less cells of this color than of the other one. The 4 instances of the form rnd-* encode the problem of finding a perfect matching in a randomly generated bipartite graph [9]. Finally, we include 9 additional instances of the form mchess$n$-p%, where a mutilated chess board problem of size $n \times n$ is viewed as a perfect matching problem on a bipartite graph. Additionally, a certain percentage p of a set of symmetry-breaking constraints are added [10].

We want to remark that no change to the decision heuristic of the baseline solver has been made. We chose Kissat [3] as a representative of a state-of-the-art CDCL SAT solver; SaDiCaL [17, 14] as the only other existing SDCL system; and our system MapleSDCL. For SaDiCaL, we used two versions, one using the positive reduct and one using the filtered positive reduct. Regarding MapleSDCL, we present four configurations: CDCL corresponds to the standard MapleSAT solver, implementing CDCL; SDCL represents a configuration using the positive reduct but no MaxSAT-based minimization, that is, learning the only-decisions clause. The two columns SDCL-min use the MaxSAT-based minimization presented in this paper for the positive and the purely positive reduct. The table reports the number of seconds needed to solve each benchmark for each system. Due to the use of internal time limits in EvalMaxSAT, the exact behavior of SDCL-min is not reproducible. In order to have a higher confidence in its results we have run it 10 times on each benchmark. For this system, the number in parenthesis corresponds to the number of executions that solved that instance within the time limit of 7200 seconds, and the runtime is the average over those successful executions.

Table 2 contains additional data of the SDCL-min configuration using the positive reduct, on the same instances. The three columns below the *Calls* label contain (i) the total number of satisfiability checks of $p_\alpha(F)$, (i) the

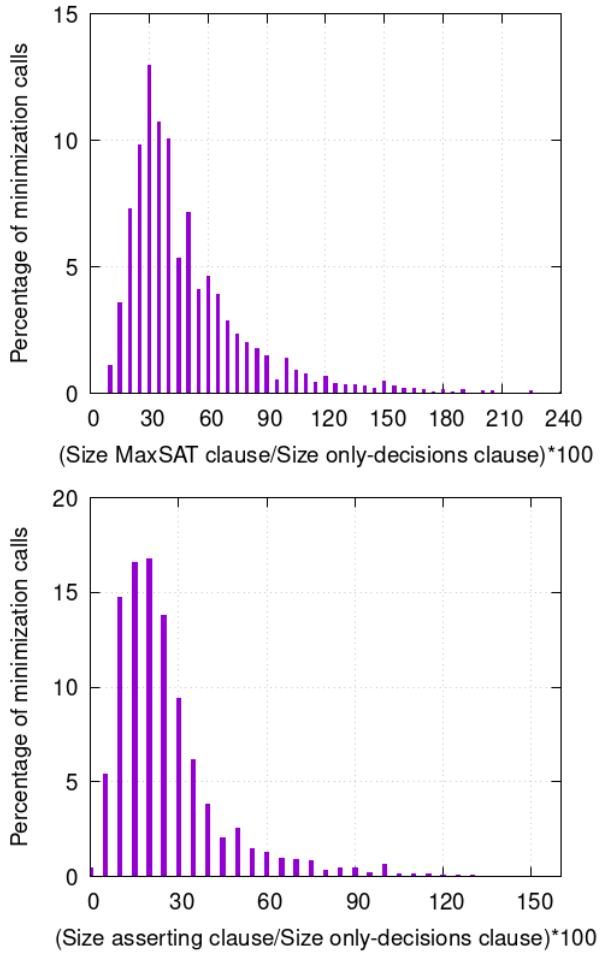

Fig. 1. Distribution of the amount of minimization achieved in the clause returned by the MaxSAT solver (left) and in the final asserting clause (right).

number of such checks that were satisfiable and hence, for which a MaxSAT solver call was performed, and (iii) the number of learned redundant clauses, respectively. It is remarkable that despite the big impact of learning redundant clauses, as shown in Table 1, our system only learns a very limited number of them. Another conclusion we can draw is that, as expected, around 15% of the calls are satisfiable and require an additional MaxSAT call. The last 5 columns of the table show the impact of changing the MaxSAT solver being used. For this purpose, we chose the best performing solvers of the unweighted category of the 2024 MaxSAT Evaluation[2]: EvalMaxSAT, CASHWMaxSAT-DisjCad, UWrMaxSat and MaxCDCL. For EvalMaxSAT, we used the version submitted to the evaluation as well as a version compiled as a library (EvalLIB in the table). Communication with the other solvers was done via files. For this particular type of application, EvalMaxSAT and MaxCDCL are clearly the most competitive solvers. More importantly, the choice of the MaxSAT solver has a big impact on the overall runtime.

[2]https://maxsat-evaluations.github.io/2024/

For pigeon-hole problems, we observed the same behavior reported in [14]: dedicated decision heuristics are needed to outperform CDCL. Unfortunately, these heuristics are not very stable as they do work if the formula is scrambled. More concretely, with such dedicated decision heuristics, SDCL with no minimization solves pigeon-hole instance of size 50 in about 10 minutes, whereas CDCL cannot solve size 13 in that amount of time. If one uses general-purpose decision heuristics like VSIDS, SDCL does not provide an advantage w.r.t. CDCL. Lots of redundant clauses are learned, but they do no seem to help reducing the search space.

Tseitin formulas [29], and other benchmarks from the SAT competition used in [26] are out of reach of our system, probably due to the fact that our current minimization procedure uses the positive reduct, and not the filtered one. We found out that our system was not able to learn almost any redundant clause in those instances. An interesting observation is that there is a strong correlation between the number of redundant clauses that SDCL learns and the improvement w.r.t. CDCL. In almost no instance our SDCL-based system learns redundant clauses but still performs worse than CDCL-based implementations. This makes it possible to build an adaptive system that disables all the SDCL-related overhead when no redundant clauses are being learned.

Overall, we observed that our technique gives important benefit on mutilated chess board and bipartite perfect matching problems, outperforming all other competitors. Regarding the use of the purely positive reduct, this has only some positive impact on the perfect matching problems, but is still worse than using the positive reduct. On mutilated chess problems, the use of the purely positive reduct did not allow our system to learn any redundant clause, showing the inability to detect blocked clauses to be added. In any case, as far as we know, it is an open problem to show whether these problems have polynomial-size proofs by adding blocked clauses. We want to remark that we performed experiments with the preprocessing-based technique of [26] for detecting PR clauses. Its performance is comparable to our minimization-based technique for mutilated chess problems and better for perfect matching problems. However, our goal was to show how far the SDCL framework can be improved, and we believe that results confirm that there is still a large space for improvement.

Finally, we would like to mention that MapleSDCL is able to produce proofs that are checkable with **dpr-trim**. However, this checker assumes that PR clauses are computed with respect to the current formula, including all learned lemmas. As already explained, we compute clauses that are PR with respect to $F \wedge R \wedge U$, where F is the initial formula, R contains all redundant clauses we have learned, and U is the set of all CDCL-like unit lemmas. This has forced us to add simple 6 lines of code to the checker that control which clauses have to be used when checking that the added PR clauses are correct.

6 Equivalence between SDCL and Proof Systems

In the following let us consider $\mathcal{R}$ to be a redundancy notion chosen among blocked, set-blocked, SPR or PR. In Section 3 we have seen that for each of these redundancy notions there is a reduct $p_{\alpha,\mathcal{R}}(F)$ that characterizes it. That is, $\neg\alpha$ is an $\mathcal{R}$-clause w.r.t. F if and only if $p_{\alpha,\mathcal{R}}(F)$ is satisfiable.

The main result of this section is to prove that SDCL with no clause deletion, and using the reduct $p_{\alpha,\mathcal{R}}(F)$ polynomially simulates the proof system consisting of general resolution and $\mathcal{R}$-clause addition.

We start by proving the following simple property:

Lemma 1. *Given a formula F, if C is an $\mathcal{R}$-clause w.r.t. F, then $C \vee D$ is also an $\mathcal{R}$-clause w.r.t. F.*

Proof. Proofs for the cases of SPR and PR can be found in [16], Theorems 8 and 11. For blocked (and set-blocked clauses) this is trivial because the literal (or the set of literals) that block C also block $C \vee D$.

□

Definition 15. *An $\mathcal{R}$-derivation of a clause C from a CNF F consisting of m clauses is a sequence of clauses $\Pi = (C_1, C_2, \cdots, C_k)$ such that its first m clauses are the clauses of F, C_k is C, and any other clause C_i is either (i) the*

Table 1. Performance of different systems on mutilated chess board and bipartite perfect matching problems. Times are in seconds.

				MapleSDCL			
		SaDiCaL		Positive			Pure.pos
Benchmark	**Kissat**	Positive	Filtered	CDCL	SDCL	SDCL-min	SDCL-min
mchess14	5	5206	2.5	10.4	6.4	**2.3** (10)	10.7 (10)
mchess15	95.6	>7200	10	48.5	21.7	**5** (10)	50 (10)
mchess16	177	>7200	14.2	380	169	**7.4** (10)	395 (10)
mchess17	966	>7200	48.4	5038	460	**21** (10)	4242 (10)
mchess18	506	>7200	**54.1**	>7200	3160	56 (10)	>7200 (0)
mchess19	>7200	>7200	>7200	>7200	2998	**100** (10)	>7200 (0)
mchess20	>7200	>7200	>7200	>7200	>7200	**301** (10)	>7200 (0)
mchess21	>7200	>7200	>7200	>7200	>7200	**1118** (10)	>7200 (0)
mchess22	>7200	>7200	>7200	>7200	>7200	**5077** (8)	>7200 (0)
mchess23	>7200	>7200	>7200	>7200	>7200	**5861** (6)	>7200 (0)
mchess16-25%	114	>7200	>7200	24.5	35.8	**18.1** (10)	26.2 (10)
mchess16-35%	57	>7200	>7200	31.3	21.1	**16.1** (10)	19.3 (10)
mchess16-45%	19	>7200	>7200	14.1	21.6	**18.1** (10)	12.2 (10)
mchess18-25%	346	>7200	>7200	502	549	**163** (10)	>7200 (0)
mchess18-35%	361	>7200	>7200	883	1159	**112** (10)	>7200 (0)
mchess18-45%	**131**	>7200	>7200	1715	724	184 (10)	>7200 (0)
mchess20-25%	1428	>7200	>7200	>7200	>7200	**1142** (10)	>7200 (0)
mchess20-35%	2131	>7200	>7200	>7200	>7200	**1518** (10)	>7200 (0)
mchess20-45%	2777	>7200	>7200	>7200	>7200	**2346** (10)	>7200 (0)
rnd-Mix-17	>7200	>7200	>7200	2837	1698	**203** (10)	2253 (10)
rnd-Mix-18	>7200	>7200	>7200	>7200	>7200	**1986** (10)	>7200 (0)
rnd-n17	>7200	>7200	>7200	1266	614	**126** (10)	760 (10)
rnd-n18	>7200	>7200	>7200	>7200	>7200	**2064** (10)	>7200 (0)

resolvent of two clauses preceding C_i or a (ii) an $\mathcal{R}$-clause w.r.t. $C_1 \wedge C_2 \wedge \cdots \wedge C_{i-1}$. An $\mathcal{R}$-derivation of the empty clause from F is called an $\mathcal{R}$-refutation of F.

The following three definitions are borrowed from [24].

Definition 16. *Given a formula F, a clause C is 1-provable w.r.t F iff $F \wedge \neg C \vdash_1 \square$.*

A alternative widely adopted name for this concept is to say the C is *RUP* (for Reverse Unit Propagation) in F.

Definition 17. *A clause $C \vee l$ is 1-empowering w.r.t formula F, and l is said to be its 1-empowering literal, iff*
(1) $F \models C \vee l$
(2) $F \wedge \neg C \not\vdash_1 \square$
(3) $F \wedge \neg C \not\vdash_1 l$

This definition implies that the addition of $C \vee l$ to F improves the power of unit propagation on F. That is, the assignment $\neg C$ now unit propagates l, whereas it was not possible before.

Table 2. Impact of using different MaxSAT solvers to solve the optimizations problems arising from redundant clause minimization. Times are in seconds.

	Calls			MaxSAT solver				
Benchmark	Total	SAT	Learn	EvalLIB	Eval	CASH	UWr	MaxCDCL
mchess14	931	138	36	**2.7**	13.2	16.9	5.8	3.5
mchess15	1759	263	61	**4.2**	30.8	32.0	9.0	7.5
mchess16	1937	298	52	**7.1**	40.5	53.3	16.2	10.2
mchess17	4948	741	93	22.4	116	148	42.5	**20.5**
mchess18	8988	1349	127	**27.6**	218	218	62.9	50.7
mchess19	14725	2207	144	83.8	548	560	110	**82.8**
mchess20	20023	3004	159	**210**	1325	1594	425	232
mchess21	47744	7160	194	**1148**	4045	3391	1557	1265
mchess22	177027	26552	275	2928	7200	5594	**2391**	4445
mchess23	301845	45283	350	**5855**	7200	7200	7200	7200
mchess16-25%	2491	372	70	19.5	168	60.5	21.1	**18.5**
mchess16-35%	2083	312	43	16.1	126	51.7	20.3	**10.1**
mchess16-45%	2139	319	46	18.3	141	64.5	23.2	**16.6**
mchess18-25%	7388	1109	177	136	857	403	125	**98**
mchess18-35%	8504	1276	116	118	1322	635	149	**94.6**
mchess18-45%	15136	2272	132	262	939	724	167	**72.5**
mchess20-25%	42540	6379	293	1037	3705	1833	1035	**928**
mchess20-35%	43372	6505	299	2045	7200	7200	1132	**1127**
mchess20-45%	47699	7152	291	**801**	5107	4214	3117	814
rnd-Mix-17	16495	2473	320	**256**	518	707	334	286
rnd-Mix-18	126028	18931	584	2196	5104	4214	1669	**1461**
rnd-n17	15258	2294	286	**117**	354	795	169	155
rnd-n18	96638	14497	565	2035	3755	5321	**1470**	2539

Definition 18. *Given a clause C and a formula F, we say that C is absorbed by F iff $F \models C$ and C is not 1-empowering w.r.t F.*

Intuitively, this means that adding an absorbed clause C to F does not increase unit propagation power. That is, learning C is, to some extent, useless from the CDCL point of view.

Lemma 2 ([24], Lemma 1). *Let F be a formula, and $C \vee l$ and $D \vee \neg l$ be two clauses that are logical consequences of F but are not 1-empowering w.r.t. F. Then $C \vee D$ is 1-provable w.r.t. F.*

Proof. If after asserting $\neg C$ or $\neg D$, unit propagation on F derives a conflict, $C \vee D$ is trivially 1-provable w.r.t F. Otherwise, since $C \vee l$ is not 1-empowering, l cannot be an empowering literal and hence $F \wedge \neg C \vdash_1 l$. A similar reasoning applied to clause $D \vee \neg l$ proves that $F \wedge \neg D \vdash_1 \neg l$. Altogether, $F \wedge \neg(C \vee D) \vdash_1 \square$ and hence $C \vee D$ is 1-provable w.r.t F. □

Lemma 3 (adaptation of Proposition 2 in [24]). *Every $\mathcal{R}$-derivation of a clause C from F that is not 1-provable w.r.t F contains either (i) a clause that is 1-provable and 1-empowering w.r.t F, or (ii) an clause that is $\mathcal{R}$ but not 1-provable w.r.t F.*

Proof. Let $(C_1, C_2, \cdots, C_k = C)$ the $\mathcal{R}$-derivation of C and let C_i be the first clause in the derivation that is not 1-provable. Note that this clause must exist because C_k is not 1-provable. C_i cannot be a clause of F, because all clauses of F are 1-provable. Hence, either:

- C_i is the resolvent of C_j and C_k, for some $j, k < i$. If C_j and C_k are both logical consequences of F, we can apply Lemma 2 to prove that either C_j or C_k are 1-empowering. Since both of them are 1-provable, we have found a clause fulfilling the properties of case (i).
  If, w.l.o.g., C_j is not a logical consequence of F, there must be some C_ℓ with $\ell \leq j$ in the derivation that is not a logical consequence of F and that was added due to an $\mathcal{R}$-clause addition step. This is because a derivation with only resolution steps only produces clauses that are logical consequences. But we know that C_ℓ is 1-provable w.r.t. F because C_i is the first clause in the derivation that is not 1-provable. This is a contradiction because any clause that is not a logical consequence of F is not 1-provable.
- C_i is an $\mathcal{R}$-clause w.r.t $C_1 \wedge \cdots \wedge C_{i-1}$. Since we know that C_i is not 1-provable, we have found a clause fulfilling the properties of case (ii).

□

Definition 19. *An SDCL-extended branching heuristic is a sequence of literals and the special symbols* **R** *and* **P**. *If symbol* **P** *is not present, we call it extended branching heuristic.*

An SDCL-extended branching heuristic tells an SDCL solver on which literals to branch, when to **R**estart and when to **P**rune the search by checking the satisfiability of a reduct.

Proposition 3 (Proposition 3 in [24]). *Let F be a CNF with n variables and C a clause that is 1-empowering and 1-provable w.r.t. F. For any asserting learning scheme LS, there exists an extended branching scheme σ such that:*

(1) The clause database of running CDCL with learning scheme LS using σ on formula F absorbs C.
(2) The size of the derivation of C is $O(n^4)$.

Note that running CDCL on an extended resolution sequence is equivalent to running SDCL on that sequence, because in the absence of **P** steps, SDCL is identical to CDCL.

All the ingredients given so far allow us to prove the main theorem of this section: SDCL, with no clause deletion, and using the reduct $p_{\alpha,\mathcal{R}}(F)$, polynomially simulates the proof system consisting of general resolution plus $\mathcal{R}$-clause addition.

Theorem 10. *Given an $\mathcal{R}$-refutation Π of a formula F, there exists an $\mathcal{R}$-refutation that can be produced by an SDCL solver that uses the $p_{\alpha,\mathcal{R}}(F)$ reduct and performs no clause deletions whose size is at most polynomially longer than the size of Π.*

Proof. Our aim is to show how to create an SDCL-extended branching sequence such that induces SDCL to derive the empty clause and produce an $\mathcal{R}$-refutation of F of size at most $O(n^4 \cdot |\Pi|)$, where n is the number of variables in F.

In the following, let us consider $\mathcal{S}$ the set of clauses that the SDCL solver has in its clause database. Initially, $\mathcal{S}$ is F.

If $\square$ is 1-provable w.r.t $\mathcal{S}$, unit propagation refutes the formula. Hence, the SDCL solver produces a refutation of size at most linear in the number of variables consisting of only unit resolution steps.

Otherwise, since $\mathcal{S}$ always contains F, we can consider Π to be a derivation of $\square$ from $\mathcal{S}$. Lemma 3 guarantees that Π contains a clause C that either (i) is 1-provable and 1-empowering w.r.t $\mathcal{S}$, or (ii) is an $\mathcal{R}$-clause w.r.t. $\mathcal{S}$ but not 1-provable w.r.t $\mathcal{S}$. We will prove that in both cases, there is an SDCL-extended branching sequence that leads the solver to absorb C.

In the first case, Proposition 3 allows us to consider an extended branching sequence that leads the SDCL solver to add clauses so that the final database absorbs C and produces a derivation of C from F that has size $O(n^4)$.

In the second case, let $C = l_1 \vee \cdots \vee l_k$ the $\mathcal{R}$-clause that is not 1-provable w.r.t $\mathcal{S}$. We build the SDCL-extended branching sequence $(\neg l_1, \neg l_2, \cdots, \neg l_k, \mathbf{P})$. Note that it cannot happen that, after deciding on a prefix of the literals $\neg l_1, \cdots, \neg l_p$ the SDCL solver unit propagates l_{p+1} because that would imply that C is 1-provable. Note, however, that it could happen that $\neg l_{p+1}$ is propagated. In this case, we remove it from the branching sequence.

After assigning all literals in the sequence and performing unit propagation, the SDCL solver cannot find a conflict. Otherwise, C would be 1-provable in $\mathcal{S}$. After that, as the **P** in the SDCL-extended branching indicates, the solver checks the satisfiability of $p_{\alpha,\mathcal{R}}(\mathcal{S})$, where α is the current assignment in the SDCL solver.

Since C is an $\mathcal{R}$-clause w.r.t $\mathcal{S}$, we can use Lemma 1 to prove that the clause $\neg\alpha$ is an $\mathcal{R}$-clause w.r.t $\mathcal{S}$, because it is of the form $C \vee D$ for some clause D. Hence, the reduct $p_{\alpha,\mathcal{R}}(\mathcal{S})$ is satisfiable and the solver can learn the $\mathcal{R}$-clause $\neg\alpha$. However, if a minimization procedure like the one presented in Section 4 is used, the solver could instead learn a possibly smaller $C' \subseteq \neg\alpha$. The solver could even additionally perform at most $O(n)$ resolution steps from C' in order to derive an asserting clause C'' that is the negation of a subset of trail, and add C'' to its clause database. Note that if no minimization routine is used, and no steps to produce an asserting clause are used, C'' is simply $\neg\alpha$.

In any case, independently of whether minimization is used or an asserting clause is produced, C is now 1-provable w.r.t $\mathcal{S}$: if we assign the negation of all literals of C, the addition of the new clause C'' either causes the solver to obtain a conflict or to produce an assignment that is a superset of α. In the latter case, clause C'', which is the negation of a subset of α, is conflicting.

If clause C is absorbed, we are done. Otherwise, clause C is 1-provable and 1-empowering w.r.t $\mathcal{S}$. Hence Proposition 3 allows us to consider an extended branching sequence that induces the SDCL solver to add clauses so that its final clause database absorbs C and to produce a derivation of C from F that has size $O(n^4)$.

Hence, we have enlarged the SDCL-extended branching sequence so that the clause database $\mathcal{S}$ of a solver that follows the sequence absorbs C. We now add a restart **R** to the SDCL-extended branching sequence.

The process is iterated. Note that at each iteration a clause from Π is absorbed and once a clause is absorbed it remains so because there are no deletions. Hence, we eventually absorb the empty clause, causing $\square$ to be 1-provable and stop the iterative process. Hence, there are in total at most $|\Pi|$ iterations. Since each iteration produces a proof of size $O(n^4)$, the final proof size is at most $O(n^4 \cdot |\Pi|)$. $\square$

One final remark is that, if the satisfiability of the reduct $p_{\alpha,\mathcal{R}}(F)$ can be determined in polynomial time, the SDCL solver can produce the desired polynomial refutation in time polynomial in $|\Pi|$.

7 Conclusions and Future Work

We have provided several novel theoretical results about SDCL. First of all, we have proved that it simulates a proof system consisting of resolution and redundant-clause addition. Secondly, we have introduced a novel pruning predicate that allows one to derive blocked clauses, for which the satisfiability problem can be solved in polynomial time. Finally, we have also shown that finding small assignments for which the corresponding reduct is satisfiable is a hard problem, at least for the positive and the purely positive reduct.

On the practical side we have shown how redundant clauses learned within the SDCL approach can be shortened by encoding the problem as a partial MaxSAT formula. Via extensive empirical evaluation we show that our technique greatly improves the performance of SDCL over families of formulas for which it was theoretically known that SDCL had a competitive advantage with respect to CDCL.

We outline several directions for future work. First of all, we could adapt the technique to also work for the filtered positive reduct. Secondly, there is a very interesting research opportunity in developing sophisticated

adaptive strategies aimed at deciding as to when the SDCL solver should attempt to learn a redundant clause. Finally, parallelization of the MaxSAT calls would greatly improve the runtime of SDCL-based systems.

Acknowledgments

Albert Oliveras is supported by grant PID2021-122830OB-C43, funded by MCIN/AEI/ 10.13039/501100011033 and by "ERDF: A way of making Europe".

References

[1] A. Atserias, J. K. Fichte, and M. Thurley. 2011. Clause-learning algorithms with many restarts and bounded-width resolution. *J. Artif. Intell. Res.*, 40, 353–373. DOI: 10.1613/jair.3152.

[2] F. Avellaneda. 2020. A short description of the solver EvalMaxSAT. In *MaxSAT Evaluation 2020*. F. Bacchus, J. Berg, M. Järvisalo, and R. Martins, (Eds.), 8–9.

[3] A. Biere, K. Fazekas, M. Fleury, and M. Heisinger. 2020. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions* (Department of Computer Science Report Series B). T. Balyo, N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda, (Eds.) Vol. B-2020-1. University of Helsinki, 51–53.

[4] A. Biere and A. Fröhlich. 2015. Evaluating CDCL variable scoring schemes. In *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings* (Lecture Notes in Computer Science). M. Heule and S. A. Weaver, (Eds.) Vol. 9340. Springer, 405–422. DOI: 10.1007/978-3-319-24318-4_29.

[5] A. Biere and D. Kröning. 2018. Sat-based model checking. In *Handbook of Model Checking*. E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, (Eds.) Springer, 277–303. DOI: 10.1007/978-3-319-10575-8_10.

[6] A. L. Blum and M. L. Furst. 1997. Fast planning through planning graph analysis. *Artificial intelligence*, 90, 1-2, 281–300.

[7] C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler. 2008. EXE: Automatically Generating Inputs of Death. *ACM Transactions on Information and System Security (TISSEC)*, 12, 2, 1–38.

[8] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith. 2018. *Model Checking*. MIT press.

[9] C. Codel, J. Reeves, M. Heule, and R. Bryant. 2021. Bipartite perfect matching benchmarks. In *Proceedings of Pragmatics of SAT*.

[10] C. Codel, J. Reeves, M. Heule, and R. Bryant. 2023. Pigeon Hole and Mutilated Chessboard with Mixed Constraint Encodings and Symmetry-Breaking. In *Proceedings of SAT Competition 2023: Solver, Benchmark and Proof Checker Descriptions* (Department of Computer Science Series of Publications B). T. Balyo, M. Heule, M. Iser, M. Järvisalo, and M. Suda, (Eds.) Department of Computer Science, University of Helsinki.

[11] S. A. Cook. 1976. A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, 8, 4, 28–32. DOI: 10.1145/1008335.1008338.

[12] S. A. Cook. 1971. The Complexity of Theorem-Proving Procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, 151–158. DOI: 10.1145/800157.805047.

[13] J. Dolby, M. Vaziri, and F. Tip. 2007. Finding Bugs Efficiently With a SAT Solver. In *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 195–204. DOI: 10.1145/1287624.1287653.

[14] M. J. H. Heule, B. Kiesl, and A. Biere. 2019. Encoding redundancy for satisfaction-driven clause learning. In *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I* (Lecture Notes in Computer Science). T. Vojnar and L. Zhang, (Eds.) Vol. 11427. Springer, 41–58. DOI: 10.1007/978-3-030-17462-0_3.

[15] M. J. H. Heule, B. Kiesl, and A. Biere. 2017. Short proofs without new variables. In *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings* (Lecture Notes in Computer Science). L. de Moura, (Ed.) Vol. 10395. Springer, 130–147. DOI: 10.1007/978-3-319-63046-5_9.

[16] M. J. H. Heule, B. Kiesl, and A. Biere. 2020. Strong extension-free proof systems. *J. Autom. Reason.*, 64, 3, 533–554. DOI: 10.1007/s10817-019-09516-0.

[17] M. J. H. Heule, B. Kiesl, M. Seidl, and A. Biere. 2017. Pruning through satisfaction. In *Hardware and Software: Verification and Testing - 13th International Haifa Verification Conference, HVC 2017, Haifa, Israel, November 13-15, 2017, Proceedings* (Lecture Notes in Computer Science). O. Strichman and R. Tzoref-Brill, (Eds.) Vol. 10629. Springer, 179–194. DOI: 10.1007/978-3-319-70389-3_12.

[18] M. J. H. Heule, O. Kullmann, and V. W. Marek. 2016. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings* (Lecture Notes in Computer Science). N. Creignou and D. L. Berre, (Eds.) Vol. 9710. Springer, 228–245. DOI: 10.1007/978-3-319-40970-2_15.

[19] L. M. Kirousis and P. G. Kolaitis. 2003. The complexity of minimal satisfiability problems. *Inf. Comput.*, 187, 1, 20–39. DOI: 10.1016/S0890-5401(03)00037-3.

[20] C. M. Li and F. Manyà. 2021. Maxsat, hard and soft constraints. In *Handbook of Satisfiability - Second Edition*. Frontiers in Artificial Intelligence and Applications. Vol. 336. A. Biere, M. Heule, H. van Maaren, and T. Walsh, (Eds.) IOS Press, 903–927. DOI: 10.3233/FAIA201007.

[21] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki. 2016. Learning rate based branching heuristic for SAT solvers. In *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings* (Lecture Notes in Computer Science). N. Creignou and D. L. Berre, (Eds.) Vol. 9710. Springer, 123–140. DOI: 10.1007/978-3-319-40970-2_9.

[22] J. Marques-Silva, I. Lynce, and S. Malik. 2021. Conflict-driven clause learning SAT solvers. In *Handbook of Satisfiability - Second Edition*. Frontiers in Artificial Intelligence and Applications. Vol. 336. A. Biere, M. Heule, H. van Maaren, and T. Walsh, (Eds.) IOS Press, 133–182. DOI: 10.3233/FAIA200987.

[23] A. Oliveras, C. Li, D. Wu, J. Chung, and V. Ganesh. 2023. Learning shorter redundant clauses in SDCL using maxsat. In *26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy* (LIPIcs). M. Mahajan and F. Slivovsky, (Eds.) Vol. 271. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 18:1–18:17. DOI: 10.4230/LIPICS.SAT.2023.18.

[24] K. Pipatsrisawat and A. Darwiche. 2011. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175, 2, 512–525. DOI: 10.1016/j.artint.2010.10.002.

[25] M. R. Prasad, A. Biere, and A. Gupta. 2005. A survey of recent advances in sat-based formal verification. *Int. J. Softw. Tools Technol. Transf.*, 7, 2, 156–173. DOI: 10.1007/s10009-004-0183-4.

[26] J. E. Reeves, M. J. H. Heule, and R. E. Bryant. 2022. Preprocessing of propagation redundant clauses. In *Automated Reasoning - 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8-10, 2022, Proceedings* (Lecture Notes in Computer Science). J. Blanchette, L. Kovács, and D. Pattinson, (Eds.) Vol. 13385. Springer, 106–124. DOI: 10.1007/978-3-031-10769-6_8.

[27] J. P. M. Silva and K. A. Sakallah. 2000. Invited tutorial: boolean satisfiability algorithms and applications in electronic design automation. In *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings* (Lecture Notes in Computer Science). E. A. Emerson and A. P. Sistla, (Eds.) Vol. 1855. Springer, 3. DOI: 10.1007/10722167_3.

[28] G. S. Tseitin. 1983. On the complexity of derivation in propositional calculus. *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*, 2, 466–483.

[29] A. Urquhart. 1987. Hard examples for resolution. *J. ACM*, 34, 1, (Jan. 1987), 209–219. DOI: 10.1145/7531.8928.

[30] Y. Xie and A. Aiken. 2005. Saturn: A SAT-Based Tool for Bug Detection. In *Proceedings of the 17th International Conference on Computer Aided Verification, CAV 2005*, 139–143. DOI: 10.1007/11513988_13.

[31] L. Zhang and S. Malik. 2002. Conflict driven learning in a quantified boolean satisfiability solver. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, ICCAD 2002, San Jose, California, USA, November 10-14, 2002*. L. T. Pileggi and A. Kuehlmann, (Eds.) ACM / IEEE Computer Society, 442–449. DOI: 10.1145/774572.774637.

Received 6 February 2025; accepted 25 June 2025