Journal of Automated Reasoning manuscript No.
(will be inserted by the editor)

Simulating Circuit-Level Simplifications on CNF [⋆]

Matti Järvisalo · **Armin Biere** · **Marijn Heule**

Received: date / Accepted: date

Abstract Boolean satisfiability (SAT) and its extensions have become a core technology in many application domains, such as planning and formal verification, and continue finding various new application domains today. The SAT-based approach divides into three steps: encoding, preprocessing, and search. It is often argued that by encoding arbitrary Boolean formulas in conjunctive normal form (CNF), structural properties of the original problem are not reflected in the CNF. This should result in the fact that CNF-level preprocessing and SAT solver techniques have an inherent disadvantage compared to related techniques applicable on the level of more structural SAT instance representations such as Boolean circuits. Motivated by this, various simplification techniques and intricate CNF encodings for circuit-level SAT instance representations have been proposed. On the other hand, based on the highly efficient CNF-level clause learning SAT solvers, there is also strong support for the claim that CNF is sufficient as an input format for SAT solvers.

In this work we study the effect of CNF-level simplification techniques, focusing on SatElite-style variable elimination (VE) and what we call blocked clause elimination (BCE). We show that BCE is surprisingly effective both in theory and in practice on CNF formulas resulting from a standard CNF encoding for circuits: without explicit knowledge of the underlying circuit structure, it achieves the same level of simplification as a combination of circuit-level simplifications and previously suggested polarity-based CNF encodings. We also show that VE can achieve many of the same effects as BCE, but not all. On the other hand, it turns out that VE and BCE are indeed partially orthogonal techniques. We also study

[⋆] Parts of this article have been preliminarily presented at the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2010) [30] and at the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT 2010) [29].

The first author is financially supported by Academy of Finland under grant 132812. The second and the third author are supported by the Austrian Science Foundation (FWF) NFN Grant S11408-N23 (RiSE). The third author is supported by the Dutch Organization for Scientific Research under grant 617.023.611.

M. Järvisalo
Department of Computer Science, University of Helsinki, Finland. E-mail: matti.jarvisalo@cs.helsinki.fi

A. Biere
Institute for Formal Models and Verification, Johannes Kepler University, Linz, Austria.

M.J.H. Heule
Department of Software Technology, Delft University of Technology, The Netherlands.

the practical effects of combining BCE and VE for reducing the size of formulas and on the running times of state-of-the-art SAT solvers. Furthermore, we address the problem of how to construct original witnesses to satisfiable CNF formulas when applying a combination of BCE and VE.

Keywords Boolean satisfiability · preprocessing · problem structure · blocked clauses · variable elimination · Boolean circuits

1 Introduction

Boolean satisfiability (SAT) [6] solvers and their extensions, especially satisfiability modulo theories (SMT) [4] solvers, have become core technologies in many application domains, including planning and formal verification. Furthermore, the SAT-based solving approach continues to find various new application domains today. Conflict-driven clause learning (CDCL) SAT solvers are at the heart of SMT solvers, and in some cases such as the theory of bit-vectors, most state-of-the-art SMT solvers are based on bit-blasting and use pure SAT solvers for actual solving (including [21,15,11,10,31,12]). This gives motivation for developing even more efficient solving techniques for SAT.

SAT-based approaches consist of three main steps: *encoding*, *preprocessing*, and *search* (solving). Encoding refers to the task of declaratively expressing (modelling) the problem to be solved in the language of propositional logic. Most state-of-the-art CDCL SAT solvers require the input formulas to be in conjunctive normal form (CNF). However, it is often the case that less restrictive forms are used during the encoding phase. Among such more structural formula representations are *Boolean circuits* that allow compact representation of propositional formulas as directed acyclic graphs (DAGs) which enables e.g. structural hashing (sharing of sub-formulas). Such circuit representations are typically afterwards automatically translated (encoded) into CNF for applying a CNF-level SAT solver to determine satisfiability of the formula. Various simplification techniques and intricate CNF encoders for general propositional formulas and especially circuit-level SAT instance representations have been proposed; see [43,8,28,41,20,40,13] for examples.

Simplification techniques applied in the preprocessing phase have the objective of automatically applying transforms to the input formula in order to make the formula (presumably) easier to solve; typically, the objective is to reduce the number of variables and subformulas appearing in the formula. Preprocessing techniques have been developed both for circuits and for CNF (for examples of CNF-level simplification techniques, see [37,39,2, 42,9,45,19,22,48,32,23]). The way the encoding and simplification/preprocessing is done before the actual solving can have a notable effect on the running times of SAT solvers. Indeed, the steps of encoding, preprocessing, and search are tightly intertwined.

It is often argued that by encoding arbitrary Boolean formulas in CNF, structural properties of the original problem are not reflected in the resulting CNF formula. This should result in the fact that CNF-level preprocessing and SAT solver techniques have an inherent disadvantage compared to related techniques that can be applied on the level of more structural SAT instance representations such as Boolean circuits. On the other hand, based on the success of the current highly efficient CNF-level CDCL SAT solvers and CNF simplification techniques, there is also strong support for the claim that CNF is sufficient as an input format for SAT solvers.

We believe that this controversy highlights that the current understanding of the limitations of CNF as the de facto input form for SAT solvers is somewhat limited, which motivates further studies on the topic. This work contributes to this understanding by analyzing CNF-level simplification techniques from a structural point of view. As main results, we show that, rather surprisingly, simple CNF-level simplification techniques can implicitly achieve many types of circuit-level simplifications which are believed to require specialized circuit-level simplifiers.

1.1 Contributions

In this work we study the effect of two main CNF-level simplification techniques: SatElite-style bounded variable elimination (VE) [19] and what we call *blocked clause elimination* (BCE). As demonstrated by the SatElite preprocessor [19], variable elimination (originating from [14, 18]) provides an effective CNF-level simplication technique via bounded application. Blocked clause elimination, on the other hand, is a clause elimination technique that removes so called *blocked clauses* [36] from CNF formulas. The concept of BCE traces back to [18, Section 2.4] where it was first noted (in dual form, considering formulas in disjunctive normal form) that such clauses can be eliminated without affecting satisfiability.[1] While (partial) elimination of blocked clauses has been proposed before [42], we present in this paper the first systematic analysis of the effectiveness of BCE.

We focus on analyzing the effect of the CNF-level preprocessing techniques of BCE and VE on CNF formulas originating from circuit-level formula representations.

We show that BCE is surprisingly effective both in theory and in practice on CNF formulas resulting from the standard “Tseitin” CNF encoding [47] for circuits: without explicit knowledge of the underlying circuit structure, BCE achieves the same level of simplification as a combination of circuit-level simplifications, such as *cone of influence*, *non-shared input elimination*, and *monotone input reduction*, and previously suggested polarity-based CNF encodings, especially the Plaisted-Greenbaum encoding [43]. This implies that, without losing simplification achieved by such specialized circuit-level techniques, one can resort to applying BCE after the straightforward Tseitin CNF encoding, and hence implementing these circuit-level techniques is somewhat redundant. Moreover, since other related circuit level optimizations for *sequential* problems—in particular, the *bounded cone of influence reduction* [5] and using functional instead of relational representations of circuits [33]—can be mapped to cone of influence, these can also be achieved by BCE purely on the CNF-level. As regards CNF-level simplification techniques, BCE achieves the simplification resulting from, e.g., *pure literal elimination* [17, 14], as also observed in [36, 42].

As regards the effect of variable elimination based preprocessing, we show that VE can achieve many of the same effects as BCE, but not all. Especially, for achieving the same simplifications as combining all the considered circuit-level simplification techniques, it is insufficient to use the Tseitin encoding before applying VE on the resulting CNF formula; instead, the Plaisted-Greenbaum encoding is required. It also turns out that VE and BCE are indeed partially orthogonal techniques, which motivates combining these two techniques for achieving even better simplification.

We also address the problem of reconstructing original solutions to CNF formulas after applying a preprocessing. For many real application scenarios of SAT it is important to be able to extract a full satisfying assignment for original SAT instances from a satisfying assignment for the instances after preprocessing. For instance, when applying BCE, a

[1] We thank one of the anonymous reviewers for pointing out this reference.

solution to the original CNF is not directly available in general. We show how such full solutions can be efficiently reconstructed from solutions to the conjunctive normal form (CNF) formulas resulting from applying a combination of various CNF preprocessing techniques, especially, blocked clause elimination combined with SatElite-style variable elimination and equivalence reasoning [2,3,22].

To accompany the more theoretical analysis in this paper, we present an experimental evaluation of the effectiveness of BCE combined with SatElite-style variable eliminating CNF preprocessing comparing our implementation with the standard Tseitin and Plaisted-Greenbaum encodings and the more recent NiceDAG [40,13] and Minicirc [20] CNF encoders. It turns out that the combination of these CNF-level techniques is in many cases competitive with the circuit-level encoders. However, it turns out that the additional benefit of applying BCE for achieving faster SAT solving is often modest: using BCE as a preprocessor appears in many cases to have only a slight positive effect on the running times of state-of-the-art SAT solvers, especially when considering CDCL. This leads to the conjecture that the often applied circuit simplifications achieved by BCE may in many cases be of limited value from the practical perspective. However, in cases some non-negligible improvements can be observed, especially for stochastic local search.

1.2 Related Work

This work is not the first time removing blocked clauses is proposed for simplifying CNF formulas; see [42] for example. However, in contrast to this paper, the work of [42] does not make the connection between blocked clauses and circuit-level simplifications and CNF encodings and, most importantly, [42] concentrates on extracting underlying circuit gate definitions for applying this knowledge in CNF-level simplification; blocked clause removal in [42] is actually *not* applied in the case any underlying gate definitions can be extracted, but rather as an auxiliary simplification over those clauses which cannot be associated with gate definitions.

Blocked clauses have played a role in studies focusing on improving the worst-case upper bounds on the running time of SAT algorithms [35] whose predecessor dates back to the concept of *complementary search* by Purdom [44].

It should be noted that, from a proof complexity theoretic point of view, there are CNF formulas which can be made easier to prove unsatisfiable with resolution (and hence also with clause learning SAT solvers) by *adding* blocked clauses [36]. In more detail, there are CNF formulas for which minimum-length resolution proofs are guaranteed to be exponential originally, but by adding instance-specific blocked clauses to the formulas, the resulting formulas yield short resolution proofs. The effect of adding (instance-specific) blocked clauses has also been studied in some domain-specifix contexts, e.g. [26]. This duality is discussed further in Section 12. This same observation on duality can also be made about VE. On one hand, VE has been shown to often notably speed-up SAT solving. On the other hand, as shown in Section 8 of this article, VE removes some clauses which are also removed by BCE, and can hence increase the length of resolution proofs dramatically. Furthermore, depending on the variable elimination ordering, VE may eliminate “wrong” variables, in analogy with making bad decisions during search, and hence lead to notable increase in proof lengths, as well.

As a final remark, we note that after the first versions of this work, the technique of blocked clause elimination has been lifted to quantified Boolean formulas (QBFs) in [7].

Quantified blocked clause elimination [7] is reported to give a substantial reduction in QBF solving time.

1.3 Organization

The rest of this paper is organized as follows. After background on Boolean circuits and CNF encodings of circuits (Section 2) and on resolution-based CNF preprocessing (Section 3), we introduce blocked clause elimination (Section 4), and review the circuit-level simplification techniques considered in this work (Section 5). An overview of the main results of this work is presented in Section 6. In-depth analysis of the effectiveness of BCE and VE with respect to known circuit-level simplification techniques and CNF encodings is then presented (Sections 7 and 8), followed by an analysis of the effectiveness of combined BCE and VE (Section 9). After the more theoretical analysis, our implementation of BCE is described in detail, followed by an in-depth description of how full solutions to CNF formulas can be reconstructed from solutions to the CNF after applying both individual and combinations of BCE and VE (Section 10). After this, experimental results are reported on the practical effectiveness of BCE and VE (Section 11), followed by conclusory remarks (Section 12).

2 Boolean Circuits and CNF Satisfiability

This section reviews the needed background related to Boolean circuits and CNF-level satisfiability, and well-known CNF encodings of circuits.

2.1 CNF Satisfiability

Given a Boolean variable x, there are two *literals*, the positive literal, denoted by x, and the negative literal, denoted by $\neg x$, the *negation of* x. As usual, we identify $\neg\neg x$ with x. A *clause* is a disjunction ($\vee$, or) of distinct literals and a CNF formula is a conjunction ($\wedge$, and) of clauses. When convenient, we view a clause as a finite set of literals and a CNF formula as a finite set of clauses; e.g. the formula $(a \vee \neg b) \wedge (\neg c)$ can be written as $\{\{a, \neg b\}, \{\neg c\}\}$. This allows us to write, for examples, $F \setminus G$ to denote the CNF formula consisting of those clauses in the CNF formula F but not in the CNF formula G, and $l \in C$ to denote that a literal l occurs in a clause C.

A truth assignment for a CNF formula F is a function τ that maps variables in F to $\{\mathbf{t}, \mathbf{f}\}$. A truth assignment is extended to literals by defining $\tau(\neg x) = \neg\tau(x)$, where $\neg\mathbf{t} = \mathbf{f}$ and $\neg\mathbf{f} = \mathbf{t}$. A clause is satisfied by τ if it contains at least one literal l such that $\tau(l) = \mathbf{t}$. An assignment τ *satisfies* F if it satisfies every clause in F. A CNF formula is *satisfiable* if there is an assignment that satisfies it, and *unsatisfiable* otherwise.

A clause is a *tautology* if it contains both x and $\neg x$ for some variable x. Finally, given an assignment τ, let τ_x (respectively, $\tau_{\neg x}$) denote the assignment such that $\tau_x(x) = \mathbf{t}$ (respectively, $\tau_{\neg x}(x) = \mathbf{f}$) and which is otherwise identical to τ.

2.2 Boolean Circuits

A Boolean circuit over a finite set G of *gates* is a set $\mathscr{C}$ of equations of form $g := f(g_1, \ldots, g_n)$, where $g, g_1, \ldots, g_n \in G$ and $f : \{\mathbf{t}, \mathbf{f}\}^n \to \{\mathbf{t}, \mathbf{f}\}$ is a Boolean function, with the additional requirements that (i) each $g \in G$ appears at most once as the left hand side in the equations in $\mathscr{C}$, and (ii) the underlying directed graph

$$\langle G, E(\mathscr{C}) = \left\{ \langle g', g \rangle \in G \times G \mid g := f(\ldots, g', \ldots) \in \mathscr{C} \right\} \rangle$$

is acyclic. If $\langle g', g \rangle \in E(\mathscr{C})$, then g' is a *child* of g and g is a *parent* of g'. If $g := f(g_1, \ldots, g_n)$ is in $\mathscr{C}$, then g is an f-gate (or of type f), otherwise it is an *input gate*. A gate with no parents is an *output gate*. The fanout (fanin, respectively) of a gate is the number of parents (children, respectively) the gate has.

A (partial) assignment for $\mathscr{C}$ is a (partial) function $\tau : G \to \{\mathbf{t}, \mathbf{f}\}$. An assignment τ is *consistent* with $\mathscr{C}$ if $\tau(g) = f(\tau(g_1), \ldots, \tau(g_n))$ for each $g := f(g_1, \ldots, g_n)$ in $\mathscr{C}$.

A *constrained Boolean circuit* $\mathscr{C}^\tau$ is a pair $\langle \mathscr{C}, \tau \rangle$, where $\mathscr{C}$ is a Boolean circuit and τ is a partial assignment for $\mathscr{C}$. With respect to a $\mathscr{C}^\tau$, each $\langle g, v \rangle \in \tau$ is a *constraint*, and g is *constrained* to v if $\langle g, v \rangle \in \tau$.

An assignment τ' *satisfies* $\mathscr{C}^\tau$ if (i) it is consistent with $\mathscr{C}$, and (ii) it respects the constraints in τ, meaning that for each gate $g \in G$, if $\tau(g)$ is defined, then $\tau'(g) = \tau(g)$. If some assignment satisfies $\mathscr{C}^\tau$, then $\mathscr{C}^\tau$ is *satisfiable* and otherwise *unsatisfiable*.

The following Boolean functions are some which often occur as gate types.

- $\textsc{not}(v)$ is $\mathbf{t}$ if and only if v is $\mathbf{f}$.
- $\textsc{or}(v_1, \ldots, v_n)$ is $\mathbf{t}$ if and only if at least one of $v_1, \ldots, v_n$ is $\mathbf{t}$.
- $\textsc{and}(v_1, \ldots, v_n)$ is $\mathbf{t}$ if and only if all $v_1, \ldots, v_n$ are $\mathbf{t}$.
- $\textsc{xor}(v_1, \ldots, v_n)$ is $\mathbf{t}$ if and only if an odd number of v_i's are $\mathbf{t}$.
- $\textsc{ite}(v_1, v_2, v_3)$ is $\mathbf{t}$ if and only if (i) v_1 and v_2 are $\mathbf{t}$, or (ii) v_1 is $\mathbf{f}$ and v_3 is $\mathbf{t}$.

As typical, we inline gate definitions of type $g := \textsc{not}(g')$. In other words, each occurrence of g as $\hat{g} := f(\ldots, g, \ldots)$ is expected to be rewritten as $\hat{g} := f(\ldots, \textsc{not}(g'), \ldots)$.

Example 1 A Boolean circuit $\mathscr{C}^\tau$ and its graphical representation are shown in Figure 1. A satisfying truth assignment for the circuit is

$$\tau' = \{\langle c_1, \mathbf{t} \rangle, \langle t_1, \mathbf{t} \rangle, \langle o_0, \mathbf{f} \rangle, \langle t_2, \mathbf{f} \rangle, \langle t_3, \mathbf{t} \rangle, \langle a_0, \mathbf{t} \rangle, \langle b_0, \mathbf{f} \rangle, \langle c_0, \mathbf{t} \rangle\}.$$

$$\begin{aligned} \mathscr{C} = \{ & c_1 := \textsc{or}(t_1, \textsc{not}(t_2)) \\ & t_1 := \textsc{and}(t_3, c_0) \\ & o_0 := \textsc{xor}(t_3, c_0) \\ & t_2 := \textsc{and}(a_0, b_0) \\ & t_3 := \textsc{xor}(a_0, b_0) \} \end{aligned}$$

$$\tau = \{\langle c_1, \mathbf{t} \rangle\}$$

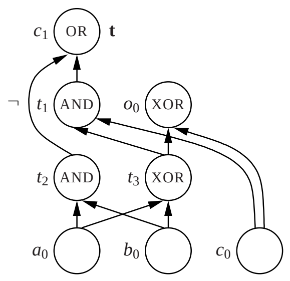

Fig. 1 A constrained Boolean circuit $\mathscr{C}^\tau$ and its graphical representation.

2.3 Well-Known CNF Encodings

The standard satisfiability-preserving “Tseitin” encoding [47] of a constrained Boolean circuit $\mathscr{C}^\tau$ into a CNF formula $\mathrm{TST}(\mathscr{C}^\tau)$ works by introducing a Boolean variable for each gate in $\mathscr{C}^\tau$, and representing for each gate $g := f(g_1, \ldots g_n)$ in $\mathscr{C}^\tau$ the equivalence $g \Leftrightarrow f(g_1, \ldots g_n)$ with clauses. Additionally, the constraints in τ are represented as unit clauses: if $\tau(g) = \mathbf{t}$ ($\tau(g) = \mathbf{f}$, respectively), introduce the clause (g) ($(\neg g)$, respectively). A well-known fact is that unit propagation[2] on $\mathrm{TST}(\mathscr{C}^\tau)$ behaves equivalently to standard Boolean constraint propagation on the original circuit $\mathscr{C}^\tau$ (see, e.g., [16] for details).

A well-known variant of the Tseitin encoding is the Plaisted-Greenbaum encoding [43] which is based on *gate polarities*. Given a constrained Boolean circuit $\mathscr{C}^\tau$, a *polarity function* $\mathsf{pol}^\tau_\mathscr{C} : G \to 2^{\{\mathbf{t},\mathbf{f}\}}$ assigns polarities to each gate in the circuit. Here $\mathbf{t}$ and $\mathbf{f}$ stand for the *positive* and *negative* polarities, respectively. Any polarity function must satisfy the following requirements.

– If $\langle g, v\rangle \in \tau$, then $v \in \mathsf{pol}^\tau_\mathscr{C}(g)$.
– If $g := f(g_1, \ldots, g_n)$, then:
  – If $f = \textsc{not}$, then $v \in \mathsf{pol}^\tau_\mathscr{C}(g)$ implies $\neg v \in \mathsf{pol}^\tau_\mathscr{C}(g_1)$.
  – If $f \in \{\textsc{and}, \textsc{or}\}$, then $v \in \mathsf{pol}^\tau_\mathscr{C}(g)$ implies $v \in \mathsf{pol}^\tau_\mathscr{C}(g_i)$ for each i.
  – If $f = \textsc{xor}$, then $\mathsf{pol}^\tau_\mathscr{C}(g) \neq \emptyset$ implies $\mathsf{pol}^\tau_\mathscr{C}(g_i) = \{\mathbf{t}, \mathbf{f}\}$.
  – If $f = \textsc{ite}$, then $v \in \mathsf{pol}^\tau_\mathscr{C}(g)$ implies
    $\mathsf{pol}^\tau_\mathscr{C}(g_1) = \{\mathbf{t}, \mathbf{f}\}$ and $v \in \mathsf{pol}^\tau_\mathscr{C}(g_i)$ for $i = 2, 3$.

The Plaisted-Greenbaum encoding [43] uses the polarity function $\mathsf{minpol}^\tau_\mathscr{C}$ that assigns for each gate the subset-minimal polarities from $2^{\{\mathbf{t},\mathbf{f}\}}$ respecting the requirements above. In other words, for each gate g,

$$\mathsf{minpol}^\tau_\mathscr{C}(g) := \{v \mid \tau(g) = v \text{ or } v \in \mathsf{minpol}^\tau_\mathscr{C}(g') \text{ for some parent } g' \text{ of } g\}.$$

The Tseitin encoding, on the other hand, can be seen as using the subset-maximal polarity assigning polarity function $\mathsf{maxpol}^\tau_\mathscr{C}(g) := \{\mathbf{t}, \mathbf{f}\}$ for each gate g. For the gate types considered in this paper, the clauses introduced based on gates polarities are listed in Table 1.

Table 1 CNF encoding for constrained Boolean circuits based on gate polarities. In the table, $\mathbf{g}_i$ is $\neg g'_i$ if $g_i := \textsc{not}(g'_i)$, and g_i otherwise.

gate g	$\mathbf{t} \in \mathsf{pol}^\tau_\mathscr{C}(g)$	$\mathbf{f} \in \mathsf{pol}^\tau_\mathscr{C}(g)$
$g := \textsc{or}(g_1, \ldots, g_n)$	$(\neg g \vee \mathbf{g}_1 \vee \cdots \vee \mathbf{g}_n)$	$(g \vee \neg\mathbf{g}_1) \wedge \cdots \wedge (g \vee \neg\mathbf{g}_n)$
$g := \textsc{and}(g_1, \ldots, g_n)$	$(\neg g \vee \mathbf{g}_1) \wedge \cdots \wedge (\neg g \vee \mathbf{g}_n)$	$(g \vee \neg\mathbf{g}_1 \vee \cdots \vee \neg\mathbf{g}_n)$
$g := \textsc{xor}(g_1, g_2)$	$(\neg g \vee \neg\mathbf{g}_1 \vee \neg\mathbf{g}_2) \wedge (\neg g \vee \mathbf{g}_1 \vee \mathbf{g}_2)$	$(g \vee \neg\mathbf{g}_1 \vee \mathbf{g}_2) \wedge (g \vee \mathbf{g}_1 \vee \neg\mathbf{g}_2)$
$g := \textsc{ite}(g_1, g_2, g_3)$	$(\neg g \vee \neg\mathbf{g}_1 \vee \mathbf{g}_2) \wedge (\neg g \vee \mathbf{g}_1 \vee \mathbf{g}_3)$	$(g \vee \neg\mathbf{g}_1 \vee \neg\mathbf{g}_2) \wedge (g \vee \mathbf{g}_1 \vee \neg\mathbf{g}_3)$
$\langle g, \mathbf{t}\rangle \in \tau$	(g)	
$\langle g, \mathbf{f}\rangle \in \tau$	$(\neg g)$	

Example 2 The polarities assigned by the subset-minimal polarity function $\mathsf{minpol}^\tau_\mathscr{C}$ to the gates of the circuit in Example 1 are shown in Figure 2 next to each gate. The clauses in the Tseitin encoding of the circuit are shown on the left. The Plaisted-Greenbaum encoding produces only the underlined clauses. In the figure, the clauses and literals removed by unit propagation are crossed over with lines.

[2] Given a CNF formula F, while there is a unit clause (l) in F, unit propagation removes from F (i) all clauses in F in which l occurs, and (ii) the literal $\neg l$ from each clause in F.

$\cancel{\underline{(c_1)}}$

$\cancel{(c_1 \vee \neg t_1)}$
$\cancel{(c_1 \vee t_2)}$
$\underline{(\cancel{\neg c_1} \vee t_1 \vee \neg t_2)}$

$(t_1 \vee \neg t_3 \vee \neg c_0)$
$\underline{(\neg t_1 \vee t_3)}$
$\underline{(\neg t_1 \vee c_0)}$

$(\neg o_0 \vee t_3 \vee c_0)$
$(\neg o_0 \vee \neg t_3 \vee \neg c_0)$
$(o_0 \vee t_3 \vee \neg c_0)$
$(o_0 \vee \neg t_3 \vee c_0)$

$\underline{(t_2 \vee \neg a_0 \vee \neg b_0)}$
$(\neg t_2 \vee a_0)$
$(\neg t_2 \vee b_0)$

$\underline{(\neg t_3 \vee a_0 \vee b_0)}$
$\underline{(\neg t_3 \vee \neg a_0 \vee \neg b_0)}$
$(t_3 \vee a_0 \vee \neg b_0)$
$(t_3 \vee \neg a_0 \vee b_0)$

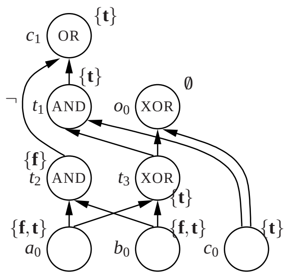

Fig. 2 The polarities assigned by the subset-minimal polarity function $\mathsf{minpol}^{\tau}_{\mathscr{C}}$ to the gate of the circuit in Example 1 (right) and the Tseitin CNF encoding of the circuit (left). The Plaisted-Greenbaum encoding of the circuit consists of the underlined clauses. The clauses and literals removed by unit propagation are crossed over with lines.

Given a constrained Boolean circuit $\mathscr{C}^{\tau}$, we denote the CNF formula resulting from the Plaisted-Greenbaum encoding of $\mathscr{C}^{\tau}$ by $\text{PG}(\mathscr{C}^{\tau})$.

Relevant additional concepts related to polarities are

- *monotone gates*: gate g is monotone if $|\mathsf{minpol}^{\tau}_{\mathscr{C}}(g)| = 1$; and
- *redundant gates*: gate g is redundant if $\mathsf{minpol}^{\tau}_{\mathscr{C}}(g) = \emptyset$.

Example 3 Recall the circuit in Example 2. The gates c_1, t_1, t_2, t_3, and c_0 are monotone, while the gate o_0 is redundant.

3 Resolution and Simplification based on Variable Elimination

The resolution rule states that, given two clauses $C_1 = \{x, a_1, \ldots, a_n\}$ and $C_2 = \{\neg x, b_2, \ldots, b_m\}$, the implied clause $C = \{a_1, \ldots, a_n, b_1, \ldots, b_m\}$, called the *resolvent* of C_1 and C_2, can be inferred by *resolving* on the variable x. We write $C = C_1 \otimes C_2$. This notion can be lifted to sets of clauses: for two sets S_x and $S_{\neg x}$ of clauses which all contain x and $\neg x$, respectively, we define

$$S_x \otimes S_{\neg x} = \{C_1 \otimes C_2 \mid C_1 \in S_x, C_2 \in S_{\neg x}, \text{ and } C_1 \otimes C_2 \text{ is not a tautology}\}.$$

Following the Davis-Putnam procedure [14] (DP), a basic simplification technique, referred to as *variable elimination by clause distribution* in [19], can be defined. The elimination of a variable x in the whole CNF formula can be computed by pair-wise resolving each clause in S_x with every clause in $S_{\neg x}$. Replacing the original clauses in $S_x \cup S_{\neg x}$ with the set of *non-tautological* resolvents $S = S_x \otimes S_{\neg x}$ gives the CNF formula $(F \setminus (S_x \cup S_{\neg x})) \cup S$ which is satisfiability-equivalent to F.

Notice that DP is a complete proof procedure for CNF formulas, with exponential worst-case space complexity. Hence for practical applications of variable elimination by clause distribution as a simplification technique for CNF formulas, variable elimination needs to be bounded. Closely following the heuristics applied in the SatElite preprocessor [19] for applying variable elimination[3], in this paper we study as a simplification technique

[3] More precisely, the SatElite preprocessor [19] applies a variant of VE called *variable elimination by substitution*. The analysis on VE in this paper applies to this variant as well.

the bounded variant of variable elimination by clause distribution, VE$_k$. In VE$_k$, a variable x can be eliminated only if $|S| \leq |S_x \cup S_{\neg x}| + k$, i.e., when the resulting CNF formula $(F \setminus (S_x \cup S_{\neg x})) \cup S$ will not contain more than $|F| + k$ clauses, where F is the formula before the elimination step and k an integer. Notice that, given that VE$_k$ may eliminate a particular variable, then VE$_{k'}$ for any $k' > k$ may also eliminate the same variable. However, the opposite does not hold in general. In the following, we let VE stand for VE$_0$, resembling closely the threshold values typically applied in practice.

Example 4 Consider a CNF formula F with

$$S_x = (x \vee c) \wedge (x \vee \neg d) \wedge (x \vee \neg a \vee \neg b) \quad \text{and} \quad S_{\neg x} = (\neg x \vee a) \wedge (\neg x \vee b) \wedge (\neg x \vee \neg e \vee f)$$

for the variable x. Applying variable elimination to eliminate x, we have

$$\begin{aligned} S &= S_x \otimes S_{\neg x} \\ &= (a \vee c) \wedge (b \vee c) \wedge (a \vee \neg d) \wedge (b \vee \neg d) \wedge \\ &\quad\ (\neg a \vee \neg b \vee \neg e \vee f) \wedge (c \vee \neg e \vee f) \wedge (\neg d \vee \neg e \vee f). \end{aligned}$$

Since $|S_x| + |S_{\neg x}| = 6$ and $|S| = 7$, VE$_0$ can not eliminate the variable x, in contrast to VE$_k$ for any $k > 0$. Notice that the clauses $(x \vee \neg a \vee \neg b)$, $(\neg x \vee a)$, and $(\neg x \vee b)$ in F are equivalent to the Tseitin encoding of the gate $x = \text{AND}(a, b)$. This is why resolving $(x \vee \neg a \vee \neg b)$ with $(\neg x \vee a)$ and $(\neg x \vee b)$ on x produces only tautological clauses that are not included in S [19]. ■

The observation made in this example (and applied e.g. in [19]) can be formulated as follows.

Proposition 1 *For any $f \in \{\text{AND}, \text{OR}, \text{XOR}, \text{ITE}\}$ and gate definition of the form*

$$g := f(g_1, \ldots, g_k),$$

applying variable elimination to eliminate the variable g in the Tseitin encoding of $g := f(g_1, \ldots, g_k)$ produces only tautological clauses.

Proof Easy to check for each $f \in \{\text{AND}, \text{OR}, \text{XOR}, \text{ITE}\}$ (recall Table 1). □

It should be noted that the result of VE can vary noticeably depending on the order in which variables are eliminated. In more detail, VE does not have a unique fixpoint for all CNF formulas, and the fixpoint reached in practice is dependent on variable elimination ordering heuristics. Hence VE is not *confluent*.

Theorem 1 VE$_k$ *is not confluent for any $k \geq 0$.*

Proof Consider the following CNF formula

$$\begin{aligned} F^n_{\text{VE}} = &\bigwedge_{1 \leq i \leq n, 1 \leq j \leq n} (x_i \vee y_j) \wedge && (1) \\ &(\neg x_1 \vee z_1) \wedge (\neg x_2 \vee z_1) \wedge \bigwedge_{3 \leq i \leq n} (\neg x_i \vee \neg z_1) \wedge && (2) \\ &(\neg y_1 \vee z_2) \wedge (\neg y_2 \vee z_2) \wedge \bigwedge_{3 \leq j \leq n} (\neg y_j \vee \neg z_2). && (3) \end{aligned}$$

Notice that each variable x_i and y_i, where $1 \leq i \leq n$, occurs exactly once negatively in F^n_{VE}. Furthermore, eliminating any x_i from F^n_{VE} will replace all positive occurrences of x_i by z_1 in the clauses of type 1 (if $i \leq 2$) or respectively by $\neg z_1$ in the clauses of type 2 (if $i \geq 3$). Additionally the single clause of type 2 containing the negative occurrence of x_i is removed. For example, eliminating the variable x_1 from F^n_{VE} results in the formula

$$\begin{aligned}&\bigwedge_{1\leq j\leq n}(z_1 \vee y_j) \wedge \bigwedge_{2\leq i\leq n, 1\leq j\leq n}(x_i \vee y_j) \wedge \\ &(\neg x_2 \vee z_1) \wedge \bigwedge_{3\leq i\leq n}(\neg x_i \vee \neg z_1) \wedge \\ &(\neg y_1 \vee z_2) \wedge (\neg y_2 \vee z_2) \wedge \bigwedge_{3\leq j\leq n}(\neg y_j \vee \neg z_2).\end{aligned}$$

Similarly, eliminating any y_i from F^n_{VE} replaces the positive occurrences of y_i by z_2 (or $\neg z_2$).

Now, for any n and $k \geq 0$, let us consider any total order $\prec$ on the variables in F^n_{VE} such that $x_i \prec z_1, z_2$ and $y_i \prec z_1, z_2$ for all $1 \leq i \leq n$. It is not difficult to see that applying VE_k on each x_i and y_i in the order given by $\prec$ will result in the formula

$$(z_1 \vee z_2) \wedge (z_1 \vee \neg z_2) \wedge (\neg z_1 \vee z_2) \wedge (\neg z_1 \vee \neg z_2).$$

Applying VE_k on this reduced formula will result in the empty clause, no matter which one of z_1 and z_2 is eliminated first.

On the other hand, fix $k \geq 0$ to an arbitrary value and consider the formula F^{k+4}_{VE}. Take any total order $\prec$ on the variables in F^n_{VE} such that $z_1, z_2 \prec' x_i$ and $z_1, z_2 \prec' y_i$ for all $1 \leq i \leq n$. Using any such variable order $\prec'$, VE_k can eliminate the variables z_1 and z_2: both variables occur positively in exactly two clauses and negatively in $k+2$ clauses. Hence, eliminating a single z_i, where $i \in \{1,2\}$, will produce $2(k+2)-2-(k+2) = k$ new clauses. For example, eliminating the variable z_1 from F^{k+4}_{VE} will result in the formula

$$\begin{aligned}&\bigwedge_{1\leq i\leq k+4, 1\leq j\leq k+4}(x_i \vee y_j) \wedge \\ &\bigwedge_{3\leq i\leq k+4}(\neg x_i \vee \neg x_1) \wedge \bigwedge_{3\leq i\leq k+4}(\neg x_i \vee \neg x_2) \wedge \\ &(\neg y_1 \vee z_2) \wedge (\neg y_2 \vee z_2) \wedge \bigwedge_{3\leq j\leq k+4}(\neg y_j \vee \neg z_2),\end{aligned}$$

after which, following the order $\prec'$, the variable z_2 can still be eliminated in a similar fashion, resulting in the formula

$$\begin{aligned}&\bigwedge_{1\leq i\leq k+4, 1\leq j\leq k+4}(x_i \vee y_j) \wedge \\ &\bigwedge_{3\leq i\leq k+4}(\neg x_i \vee \neg x_1) \wedge \bigwedge_{3\leq i\leq k+4}(\neg x_i \vee \neg x_2) \wedge \\ &\bigwedge_{3\leq i\leq k+4}(\neg x_i \vee \neg y_1) \wedge \bigwedge_{3\leq i\leq k+4}(\neg x_i \vee \neg y_2).\end{aligned}$$

However, we notice that after eliminating z_1 and z_2, all remaining variables occur at least twice negatively and $k+4$ times positively. In fact, VE_k cannot remove any of these variables because they will introduce at least $2(k+4)-2-(k+4) = k+2$ clauses. Therefore, by eliminating the z_i variables first according to $\prec'$, VE_k does not result in the empty clause, in contrast to the case in which the order $\prec$ was used. $\square$

4 Blocked Clause Elimination

The main analysis presented in this paper involves what we call *blocked clause elimination* (BCE), a CNF-level simplification technique removes so called *blocked clauses* [36] from CNF formulas.

Definition 1 (Blocking literal) A literal l in a clause C of a CNF formula F blocks C (with respect to F) if for every clause $C' \in F$ with $\neg l \in C'$, the resolvent $(C \setminus \{l\}) \cup (C' \setminus \{\neg l\})$ obtained from resolving C and C' on l is a tautology.

With respect to a fixed CNF formula and its clauses we have:

Definition 2 (Blocked clause) A clause is blocked if it has a literal that blocks it.

Example 5 Consider the formula

$$F_{\text{blocked}} = (a \vee b) \wedge (a \vee \neg b \vee \neg c) \wedge (\neg a \vee c).$$

Only the first clause of F_{blocked} is not blocked. Both of the literals a and $\neg c$ block the second clause. The literal c blocks the last clause. Notice that after removing either $(a \vee \neg b \vee \neg c)$ or $(\neg a \vee c)$, the clause $(a \vee b)$ becomes blocked. This is actually an extreme case in which BCE can remove all clauses of a formula, resulting in a trivially satisfiable formula. ■

As a side-remark, notice that a literal l cannot block any clause in a CNF formula F if F contains the unit clause $\{\neg l\}$, and hence in this case no clause containing l can be blocked with respect to F. An important fact is that BCE preserves satisfiability.

Proposition 2 ([36]) *Removal of an arbitrary blocked clause preserves satisfiability.*

which follows immediately from the following proposition

Proposition 3 ([36]) *Assume that literal l blocks C w.r.t. F. Let τ be a satisfying assignment for $F \setminus \{C\}$. If τ does not satisfy C, then τ_l satisfies both $F \setminus \{C\}$ and C and thus F.*

Additionally, we have the following.

Lemma 1 *Given a CNF formula F, let clause $C \in F$ be blocked with respect to F. Any clause $C' \in F$, where $C' \neq C$, that is blocked with respect to F is also blocked with respect to $F \setminus \{C\}$.*

Proof If the clause C where blocked with respect to F but not with respect to $F \setminus \{C\}$, then there should be a $C' \in F \setminus \{C\}$ which causes C not to be blocked with respect to $F \setminus \{C\}$. However, since we know that $C' \in F$, the clause C can not be blocked with respect to F. □

Therefore the result of blocked clause elimination is independent of the order in which blocked clauses are removed, and hence blocked clause elimination has a unique fixpoint for any CNF formula, i.e., BCE is confluent.

Theorem 2 BCE *is confluent.*

5 Circuit-Level Simplifications

In this section we review the circuit-level simplification techniques—*non-shared inputs elimination*, *monotone input reduction*, and *cone of influence reduction* [16]—considered in this work.

For the following, we consider an arbitrary constrained Boolean circuit $\mathscr{C}^{\tau}$.

Non-shared inputs elimination (NSI): While there is a (non-constant) gate g with the definition $g := f(g_1,\ldots,g_n)$ such that each g_i is an input gate with fanout one (non-shared) in $\mathscr{C}^{\tau}$, remove the gate definition $g := f(g_1,\ldots,g_n)$ from $\mathscr{C}$.

Monotone input reduction (MIR): While there is a monotone input gate g in $\mathscr{C}^{\tau}$, extend τ by assigning g to $\mathsf{minpol}^{\tau}_{\mathscr{C}}(g)$.

Cone of influence reduction (COI): While there is a redundant gate g in $\mathscr{C}^{\tau}$, remove the gate definition $g := f(g_1,\ldots,g_n)$ from $\mathscr{C}$.

These circuit-level simplifications, along with the Plaisted-Greenbaum encoding, are implemented e.g. in the `bc2cnf` circuit simplifier and CNF encoder that is part of the BC package implemented by Tommi Junttila.[4]

Example 6 Recall the circuit in Example 2. The definition of the gate o_0 is removed by COI since o_0 is redundant. The gate c_0 is assigned to **t** by MIR since c_0 is a monotone input. As an interesting example of the behavior of NSI, consider a chain of XOR-gates, $g_3 := \text{XOR}(g_2,g_1), g_5 := \text{XOR}(g_4,g_3),\ldots$ in which the gate g_1 and each g_{2i}, where $i > 0$, are non-shared input gates (as illustrated on the left in Figure 3). As long as each g_{2i+1}, where $i > 0$, is non-shared, NSI will remove the whole chain, first removing the definition $g_3 := \text{XOR}(g_2,g_1)$ and so forth (as illustrated on the right in Figure 3). ■

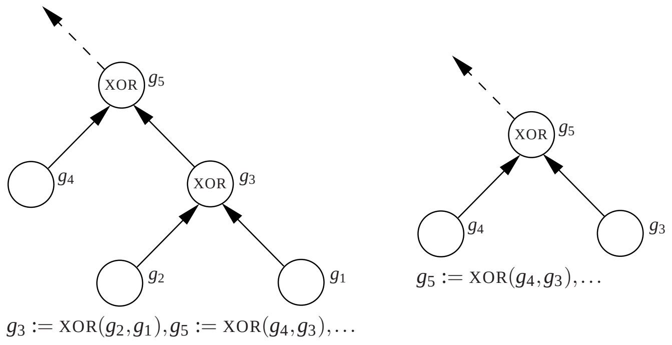
XOR
g_5
g_4
XOR
g_3
g_2
g_1
$g_3 := \text{XOR}(g_2,g_1), g_5 := \text{XOR}(g_4,g_3),\ldots$
XOR
g_5
g_4
g_3
$g_5 := \text{XOR}(g_4,g_3),\ldots$

Fig. 3 An XOR chain in which the gates g_1 and each g_{2i}, where $i > 0$, are input gates (left); the chain after the gate g_3 has been removed by NSI (right).

6 Overview of Main Results

The main results of this section show the surprising effectiveness of the CNF-level simplification of blocked clause elimination and variable elimination (when applied until fixpoint).

[4] See `http://users.ics.tkk.fi/tjunttil/circuits/`.

For the analysis, we will apply the following definition of the relative effectiveness of CNF encodings and both circuit and CNF-level simplification techniques.

Definition 3 Assume two methods T_1 and T_2 that take as input an arbitrary constrained Boolean circuit $\mathscr{C}^\tau$ and output CNF formulas $T_1(\mathscr{C}^\tau)$ and $T_2(\mathscr{C}^\tau)$, respectively, that are satisfiability-equivalent to $\mathscr{C}^\tau$. We say that T_1 is *at least as effective as* T_2 if, for any $\mathscr{C}^\tau$, $T_1(\mathscr{C}^\tau)$ contains at most as many clauses and variables as $T_2(\mathscr{C}^\tau)$ does. If T_1 is at least as effective as T_2 and vice versa, then T_1 and T_2 are *equally effective*. If there is a $\mathscr{C}^\tau$ for which $T_2(\mathscr{C}^\tau)$ contains more clauses or variables than $T_1(\mathscr{C}^\tau)$ does, then T_2 is *not as effective as* T_1. Finally, if T_1 is at least as effective as T_2, and T_2 is not as effective as T_1, then T_1 is *strictly more effective than* T_2.

Notice that, considering BCE, a stricter variant of this definition, based on clause elimination, could be applied: T_1 is at least as effective as T_2 , if for every circuit $\mathscr{C}^\tau$ we have $T_1(\mathscr{C}^\tau) \subseteq T_2(\mathscr{C}^\tau)$. However, for VE this stricter definition cannot be naturally applied, since in general VE produces non-tautological resolvents which are not subsumed by the original clauses. Because of this inherent property of VE, we will for simplicity in the following use the “weaker” version, as in Definition 3. All the results presented not concerning VE also hold under the stricter version of the definition. Also notice that the “at least as effective” relation is analogously defined for two CNF-level simplification methods which, instead of Boolean circuits, take CNF formulas as input.

When considering the effectiveness of VE in this paper, we apply a non-deterministic interpretation which allows for *any* variable elimination order, i.e., we say that VE can achieve the effectiveness of another simplification technique, if there is some elimination order for which VE achieves the same effectiveness. Finally, note that in the following *we always assume that formulas are closed under standard unit propagation.*

An overview of the main results of this section is presented in Fig. 4. An edge from X to Y implies that X is as least as effective as Y. Notice that transitive edges are omitted: for example, BCE is at least as effective as the combination of PG and the circuit-level simplification techniques *cone of influence reduction* (COI), *non-shared inputs elimination* (NSI), and *monotone input reduction* (MIR). On the left side, X_{PG} means the combination of first applying the Plaisted-Greenbaum and then the CNF-level simplification technique X on the resulting CNF formula. Analogously, PG_X means the combination of first applying the circuit-level simplification X and then the Plaisted-Greenbaum encoding. On the right side the standard Tseitin encoding is always applied. The pointed circles around COI, MIR, and NSI on the left and right represent applying the combination of these three simplifications and then the Plaisted-Greenbaum (left) or Tseitin encoding (right). Additionally, BCE + VE refers to all possible ways of alternating BCE and VE until fixpoint.

6.1 Pure Literal Elimination by BCE and VE

Before turning to the main results, relating BCE with circuit-level simplification techniques, we begin by first arguing that both BCE and VE actually achieve the same simplifications as the well-known *pure literal elimination*, first introduced in [17, 14]. Given a CNF formula F, a literal l occurring in F is *pure* if $\neg l$ does not occur in F.

Pure Literal Elimination (PL): While there is a pure literal l in F, remove all clauses containing l from F.

Notice that the following two lemmas apply for all CNF formulas, and are not restricted to CNF formulas produced by the TST or PG encodings.

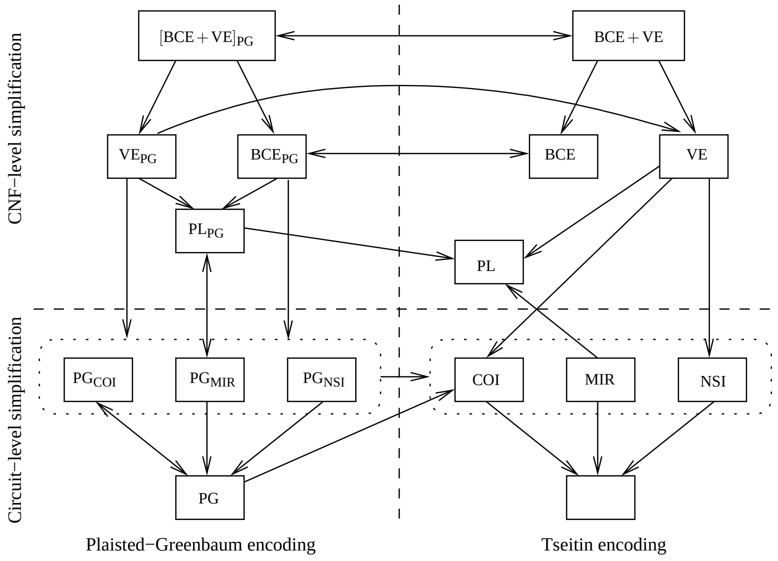

Fig. 4 Relative effectiveness of combinations of CNF encodings with both circuit and CNF-level simplification techniques. An edge from X to Y implies that X is as least as effective as Y. Transitive edges are omitted. Notice that these results are dependent on the assumption that formulas are closed under standard unit propagation.

Lemma 2 BCE *is at least as effective as* PL.

Proof A pure literal blocks all clauses which contain it by definition, and hence clauses containing a pure literal are blocked. □

Notice that a similar observation to Lemma 2 (that clauses that contain pure literals are blocked) was already made in [36, 42].

Lemma 3 VE *is at least as effective as* PL.

Proof Let l be a pure literal. By definition, $S_{\neg l}$ (the set of clauses containing $\neg l$) is empty. Hence $S_l \otimes S_{\neg l} = \emptyset$, and therefore VE removes the clauses in S_l. □

It is also evident—as will become clear in the following—that there are examples on which both BCE and VE can remove clauses not removed by applying PL. Hence both BCE and VE are in fact strictly more effective than PL.

7 Effectiveness of BCE on Circuit-Based CNF Formulas

In this section we show that BCE, starting from the Tseitin encoding of any Boolean circuit, achieves all of the simplifications achieved by the circuit-level techniques NSI, COI, and MIR, and also removes those clauses that do not appear in the Plaisted-Greenbaum encoding of the simplified circuit. Before proceeding, let us remind the reader again that *these results are dependent on the assumption that formulas are closed under standard unit propagation.*

First, we observe that the Plaisted-Greenbaum encoding actually achieves the effectiveness of COI.

Lemma 4 PG *is at least as effective as* $\mathrm{PG}_{\mathrm{COI}}$

Proof For any redundant gate g, $\mathsf{minpol}^{\tau}_{\mathscr{C}}(g) = \emptyset$ by definition. Hence the Plaisted-Greenbaum encoding does not introduce any clauses for such a gate. □

On the other hand, blocked clause elimination can achieve the Plaisted-Greenbaum encoding starting with the result of the Tseitin encoding. For the result, the following small lemma is useful.

Lemma 5 *For any* $f \in \{\mathrm{AND}, \mathrm{OR}, \mathrm{XOR}, \mathrm{ITE}\}$ *and gate definition of the form* $g := f(g_1,\dots,g_k)$, *applying* BCE *on the Tseitin encoding of* $g := f(g_1,\dots,g_k)$ *removes all clauses.*

Proof For any $f \in \{\mathrm{AND}, \mathrm{OR}, \mathrm{XOR}, \mathrm{ITE}\}$ and $g := f(g_1,\dots,g_k)$, it is easy to check that the literals associated with g (recall Table 1) block each of the clauses in the Tseitin encoding of $g := f(g_1,\dots,g_k)$, and hence all of the clauses are blocked. □

Lemma 6 $\mathrm{BCE}_{\mathrm{TST}}$ *is at least as effective as* PG.

Proof We claim that BCE removes all clauses in $\mathrm{TST}(\mathscr{C}^{\tau}) \setminus \mathrm{PG}(\mathscr{C}^{\tau})$ from $\mathrm{TST}(\mathscr{C}^{\tau})$. There are two cases to consider: redundant and monotone gates. For both cases, BCE works implicitly in a top-down manner, starting from the output gates. Notice that BCE has no and does not need explicit knowledge of the circuit $\mathscr{C}^{\tau}$ underlying $\mathrm{TST}(\mathscr{C}^{\tau})$. Since BCE is confluent it will remove *all* blocked clauses independent of the elimination order.

Consider an arbitrary redundant output gate definition $g := f(g_1,\dots,g_n)$. Since g is not constrained under τ, all clauses in $\mathrm{TST}(\mathscr{C}^{\tau})$ in which g occurs are related to this definition. By Lemma 5, BCE removes all clauses in which g occurs. On the circuit level, this is equivalent to removing the definition $g := f(g_1,\dots,g_n)$.

Now consider an arbitrary monotone output gate definition $g := f(g_1,\dots,g_n)$ with polarity $\mathsf{minpol}^{\tau}_{\mathscr{C}}(g) = \{v\}$, where $v \in \{\mathbf{t},\mathbf{f}\}$. Then g must be constrained: $\tau(g) = v$. Hence unit propagation on g removes all clauses produced by TST for the case "if $\neg v \in \mathsf{pol}^{\tau}_{\mathscr{C}}(g)$" in Table 1 and removes the occurrences of g from the clauses produced for the case "if $v \in \mathsf{pol}^{\tau}_{\mathscr{C}}(g)$". To see how BCE removes in a top-down manner those clauses related to monotone gate definitions which are not produced by PG, consider the gate definition $g_i := f'(g'_1,\dots,g'_{n'})$. Assume that unit propagation on g has no effect on the clauses produced by TST for this definition, that $\mathsf{minpol}^{\tau}_{\mathscr{C}}(g_i) = \{v\}$, and that BCE has removed all clauses related to the parents of g_i in $\mathrm{TST}(\mathscr{C}^{\tau}) \setminus \mathrm{PG}(\mathscr{C}^{\tau})$. Now one can check that the literals associated with g_i block each of the clauses produced by TST for the case "if $\neg v \in \mathsf{pol}^{\tau}_{\mathscr{C}}(g_i)$". This is because all the clauses produced by TST for the definitions of g_i's parents and in which g_i occurs have been already removed by BCE (or by unit propagation). Hence all the clauses produced by TST for the case "if $\neg v \in \mathsf{pol}^{\tau}_{\mathscr{C}}(g_i)$" in Table 1 are blocked. □

Example 7 Recall the circuit and the CNF formula resulting from applying the Tseitin encoding on this circuit in Example 2. BCE can remove the clauses that are not produced by applying the Plaisted-Greenbaum on this circuit (that is, the clauses that are not underlined) for instance in the following order:

1. $(\neg o_0 \vee t_3 \vee c_0)$, $\neg o_0$ blocking
2. $(\neg o_0 \vee \neg t_3 \vee \neg c_0)$, $\neg o_0$ blocking
3. $(o_0 \vee t_3 \vee \neg c_0)$, o_0 blocking
4. $(o_0 \vee \neg t_3 \vee c_0)$, o_0 blocking
5. $(t_1 \vee \neg t_3 \vee \neg c_0)$, t_1 blocking

6. $(\neg t_2 \vee a_0)$, $\neg t_2$ blocking
7. $(\neg t_2 \vee b_0)$, $\neg t_2$ blocking
8. $(t_3 \vee a_0 \vee \neg b_0)$, t_3 blocking
9. $(t_3 \vee \neg a_0 \vee b_0)$, t_3 blocking

■

Combining Lemmas 4 and 6, we have

Lemma 7 BCE$_{\text{TST}}$ *is at least as effective as* PG$_{\text{COI}}$.

Next, we consider non-shared inputs elimination.

Lemma 8 BCE$_{\text{TST}}$ *is at least as effective as* PG$_{\text{NSI}}$.

Proof Assume a gate definition $g := f(g_1,\ldots,g_n)$ such that each g_i is a non-shared input gate. It is easy to check from Table 1 that for each g_i, each clause produced by TST for $g := f(g_1,\ldots,g_n)$ is blocked by $\mathbf{g}_i$. The result now follows from Lemma 6 and Proposition 1 (notice that PG($\mathscr{C}^\tau$) is always a subset of TST($\mathscr{C}^\tau$)). □

Example 8 Consider a chain of XOR-gates, $g_3 := \text{XOR}(g_2,g_1), g_5 := \text{XOR}(g_4,g_3),\ldots$ in which the gate g_1 and each g_{2i}, where $i > 0$, are non-shared input gates (recall Figure 3, illustrated on the left). The clauses in the Tseitin encoding of $g_3 := \text{XOR}(g_2,g_1)$ are

$$(\neg g_3 \vee \neg g_1 \vee \neg g_2), (\neg g_3 \vee g_1 \vee g_2), (g_3 \vee \neg g_1 \vee g_2), (g_3 \vee g_1 \vee \neg g_2).$$

Since the gates g_1 and g_2 are non-shared, they do not appear in any other clauses of the Tseitin encoding. Now both g_1 and g_2 block each of these clauses. Seeing the Tseitin encoding on the circuit-level after removing these four clauses clauses, the result is the circuit on the right in Figure 3, which corresponds exactly to NSI on the gate g_3.

Notice that, in fact, BCE can remove all the four aforementioned clauses even in the case that only one of the gates g_1 and g_2 is non-shared. In this case, the non-shared one blocks all the four clauses.

Related to this example, we note that in [25] XOR chains are explicitly detected on the CNF-level. Detected chains are removed if they contain a non-shared input gate (i.e., a variable that does not occur in the other clauses). Moreover, if a variable occurs only in XOR chains, the variable is substituted in all but one chain using XOR reasoning. The remaining XOR chain in which it occurs is removed. ■

On the other hand, notice that PL cannot achieve the effectiveness of NSI when applying PG: since PG produces the same set of clauses as TST for any gate g with $\mathsf{minpol}^\tau_{\mathscr{C}}(g) = \{\mathbf{t},\mathbf{f}\}$, no literal occurring in these clauses can be pure.

We now turn to the monotone input reduction. Notice that MIR is a proper generalization of PL: given a CNF formula F, any pure literal in F is monotone in the straight-forward circuit representation of F where each clause $C \in F$ is represented as an output OR-gate the children of which are the literals in C. On the other hand, a monotone input gate in a circuit $\mathscr{C}^\tau$ is not necessarily a pure literal in TST($\mathscr{C}^\tau$): TST introduces clauses which together contain both positive and negative occurrences of all gates, including monotone ones. Hence we have the following.

Proposition 4 TST$_{\text{MIR}}$ *is strictly more effective than* PL$_{\text{TST}}$.

However, it actually turns out that, when applying the Plaisted-Greenbaum encoding, PL and MIR are equally effective (and hence PL_{PG} is also strictly more effective than PL_{TST}).

Lemma 9 PL_{PG} *and* PG_{MIR} *are equally effective.*

Proof Assume a gate definition $g := f(g_1,\dots,g_n)$, where some g_i is a monotone input gate. To see that $\text{PL}(\text{PG}(\mathscr{C}^\tau))$ is at least as effective as $\text{PG}(\text{MIR}(\mathscr{C}^\tau))$, first notice that since g_i is monotone, g is monotone. Now, it is easy to check (recall Table 1) that g_i occurs only either negatively or positively in the clauses introduced by PG for $g := f(g_1,\dots,g_n)$, and hence $\mathbf{g}_i$ is pure.

To see that $\text{PG}(\text{MIR}(\mathscr{C}^\tau))$ is at least as effective as $\text{PL}(\text{PG}(\mathscr{C}^\tau))$, notice that in order to be a pure literal in $\text{PG}(\mathscr{C}^\tau)$, a gate has to be both monotone and an input. □

Using this lemma, we arrive at the fact that BCE on TST can achieve the combined effectiveness of MIR and PG.

Lemma 10 BCE_{TST} *is at least as effective as* PG_{MIR}.

Proof Since BCE can remove all clauses in $\text{TST}(\mathscr{C}^\tau) \setminus \text{PG}(\mathscr{C}^\tau)$ by Lemma 6, after this BCE can remove all clauses containing some monotone input gate g_i since BCE is at least as effective as PL (Lemma 2). The result then follows by Lemma 9. □

Combining Lemmas 6, 7, 8, and 10, we arrive at the following.

Lemma 11 BCE_{TST} *is at least as effective as first applying the combination of* COI, MIR, *and* NSI *on the circuit-level until fixpoint, and then applying* PG *on the resulting circuit.*

As an interesting side-remark, we also have the following.

Proposition 5 *The combination of* NSI, MIR, *and* COI *is confluent.*

Moreover, BCE is more effective than applying the combination of COI, MIR, and NSI on $\mathscr{C}^\tau$ until fixpoint, and then applying PG on the resulting circuit.

Lemma 12 *First applying the combination of* COI, MIR, *and* NSI *on the circuit-level until fixpoint, and then applying* PG *on the resulting circuit, is not as effective as* BCE_{TST}.

Proof Consider a gate definition $g := \text{XOR}(g_1,\dots,g_n)$, where g has $\mathsf{minpol}^\tau_{\mathscr{C}}(g) = \{\mathbf{t},\mathbf{f}\}$ (hence COI and MIR cannot remove/assign this gate definition, and the TST and PG CNF encodings produce exactly the same clauses for this definition) and only a *single* g_i is a non-shared input gate (hence NSI cannot remove the definition), i.e. it occurs only in the definition of g. However, in this case the clauses in $\text{TST}(\mathscr{C}^\tau)$ in which g_i occurs are blocked. □

Combining Lemmas 11 and 12, we finally arrive at our main theorem.

Theorem 3 BCE_{TST} *is strictly more effective than first applying the combination of* COI, MIR, *and* NSI *on the circuit-level until fixpoint, and then applying* PG *on the resulting circuit.*

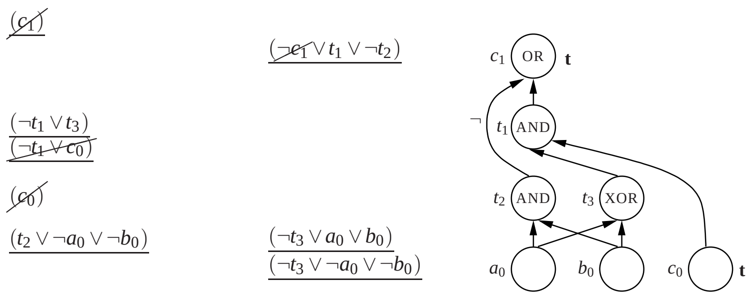

Fig. 5 The circuit of Example 2 after applying COI, NSI, and MIR, and its Plaisted-Greenbaum encoding after unit propagation. the clauses and literals removed by unit propagation are crossed over with lines.

Example 9 Recall the circuit and the CNF formula resulting from applying the Tseitin encoding on this circuit in Example 2. The circuit after applying COI, NSI, and MIR, along with its Plaisted-Greenbaum encoding after unit propagation, is shown in Figure 5 (notice that the unit clauses are shown for clarity only).

In contrast, after removing those clauses in the Tseitin encoding of the original circuit that are not produced by applying the Plaisted-Greenbaum on the original circuit (recall Example 7), BCE can remove *all* the remaining clauses in the following order by simulating PL.

1. $(t_1 \vee t_2)$ and $(t_2 \vee \neg a_0 \vee \neg b_0)$, t_2 is pure
2. $(\neg t_1 \vee t_3)$, $\neg t_1$ is pure
3. $(\neg t_3 \vee a_0 \vee b_0)$ and $(\neg t_3 \vee \neg a_0 \vee \neg b_0)$, $\neg t_3$ is pure

■

8 VE and Circuit-Level Simplifications

We will now show that VE, using an optimal elimination ordering, can also achieve the effectiveness of many of the considered circuit-level simplifications.

Proposition 6 VE_{TST} *is at least as effective as (i)* TST_{COI}*; (ii)* TST_{NSI}.

Proof

(i) Assume a redundant output gate definition $g := f(g_1, \ldots, g_n)$. Now $S_g \otimes S_{\neg g} = \emptyset$ since all resolvents are tautologies when resolving on g (recall Table 1).

(ii) Assume a gate definition $g := f(g_1, \ldots, g_n)$ such that each g_i is a non-shared input gate. For OR (similarly for AND), $S_{g_1} \otimes S_{\neg g_1} = \emptyset$. After resolving on g_1 we are left with the clauses $\cup_{i=2}^{k}(g \vee \neg \mathbf{g}_i)$, where each $\neg \mathbf{g}_i$ is then a pure literal. For XOR, simply notice that $S_{g_1} \otimes S_{\neg g_1} = \emptyset$. For ITE, notice that $S_{g_1} \otimes S_{\neg g_1} = (\neg g \vee \mathbf{g}_2 \vee \mathbf{g}_3)$, and then $\mathbf{g}_2$ and $\mathbf{g}_3$ are both pure literals.

□

Proposition 7 VE_{PG} *is at least as effective as* VE_{TST}.

Proof Follows from $\text{PG}(\mathscr{C}^\tau) \subseteq \text{TST}(\mathscr{C}^\tau)$. □

Proposition 8 $\mathrm{VE_{PG}}$ *is at least as effective as (i)* $\mathrm{PG_{COI}}$*; (ii)* $\mathrm{PG_{NSI}}$*; and (iii)* $\mathrm{PG_{MIR}}$.

Proof (i) Follows directly from Lemma 4.
(ii) By a similar argument as in the proof of Proposition 6, part (ii).
(iii) Follows directly from Lemmas 3 and 9.

□

Evidently, there are also examples on which $\mathrm{VE_{PG}}$ (or $\mathrm{VE_{TST}}$, when mentioned) can remove more clauses than the techniques considered in the three propositions above. Hence $\mathrm{VE_{PG}}$ is in fact strictly more effective than any of these techniques. However, there are cases in which VE is not as effective as BCE. In fact, as we will show next, compared to applying only BCE or VE, one can benefit from applying the *combination* of BCE and VE.

9 Benefits of Combining BCE and VE

We will now consider aspects of applying BCE in combination with VE.

There are cases in which VE is not as effective as BCE. Namely, VE cannot achieve the effectiveness of MIR when applying TST, in contrast to BCE.

Proposition 9 $\mathrm{VE_{TST}}$ *is not as effective as* $\mathrm{BCE_{TST}}$.

Proof To see this, notice that an input gate can have arbitrarily large finite fanout and still be monotone. On the other hand, VE_k cannot be applied on gates which have arbitrarily large fanout and fanin, since the elimination bound k can then be exceeded (number of clauses produced would be greater than the number of clauses removed).

For a concrete example, consider the following circuit:

$$\begin{aligned}\mathscr{C}_{PG,n} = \{&x_1 = \mathrm{AND}(g_2,\ldots,g_n),\ldots,x_i = \mathrm{AND}(g_1,\ldots,g_{i-1},g_{i+1},\ldots,g_n),\\ &\ldots,x_n = \mathrm{AND}(g_1,\ldots,g_{n-1}),\\ &y_1 = \mathrm{OR}(x_1,x_2),\ldots,y_{n-1} = \mathrm{OR}(x_{n-1},x_n),y_n = \mathrm{OR}(x_n,x_1)\}\end{aligned}$$

$$\tau = \{\langle y_1,\mathbf{t}\rangle,\ldots,\langle y_n,\mathbf{t}\rangle\}.$$

Now, the Tseitin encoding of $\mathscr{C}^{\tau}_{PG,k+5}$ after unit propagation (on the y_i variables) is

$$F^{k+5}_{\mathrm{TST}} = \bigwedge_{1\le i\le k+5}\Big(x_i \vee \bigvee_{1\le j\le k+5,\, j\ne i} \neg g_j\Big) \wedge \bigwedge_{1\le i,j\le k+5, i\ne j}(\neg x_i \vee g_j) \wedge \bigwedge_{1\le i<k+5}(x_i \vee x_{i+1}) \wedge (x_{k+5}\vee x_1).$$

For any $k \ge 0$, VE_k can not eliminate any variables from F^{k+5}_{TST}: each of the x_i variables occurs three times positively and $k+4$ times negatively, so that each x_i occurs in exactly $k+7$ clauses. Eliminating any x_i, the number of distinct new non-tautological resolvents would be $2(k+4)$; we have $2(k+4)-(k+7) = k+1 > k$. Each of the g_i variables occurs $k+4$ times positively and $k+4$ times negatively. By eliminating any g_i, the number of distinct new non-tautological resolvents would be $(k+3)(k+4)$; we have $(k+3)(k+4)-2(k+4) = k^2+5k+4 > k$.

On the other hand, for any n, BCE can remove *all* clauses of F^n_{TST} by eliminating the clauses in chronological order. First, all literals x_i in the non-binary clauses are blocking. (this step is essentially applying Plaisted-Greenbaum). After this, all g_j literals are blocking and pure (this step is essentially applying MIR). After all clauses that contain some g_j literal are removed, all the x_i literal are blocking the remaining clauses. □

In general, a main point to notice is that for VE, in order to achieve the effectiveness of BCE (on the standard Tseitin encoding), one has to apply the Plaisted-Greenbaum encoding before applying VE. In addition, since VE is not confluent in contrast to BCE, in practice the variable elimination ordering heuristics for VE has to be good enough so that it forces the “right” elimination order. In addition, there are cases in which BCE is more effective than $\mathrm{VE}_{\mathrm{PG}}$.

Theorem 4 $\mathrm{VE}_{\mathrm{PG}}$ *is not as effective as* $\mathrm{BCE}_{\mathrm{TST}}$.

Proof For some intuition, consider a clause C with blocking literal l. Notice that the result of performing VE on l is not dependent on whether C is removed. However, for any non-blocking literal $l' \in C$ the number of non-tautological clauses after applying VE on l' would be smaller if BCE would first remove C.

For a concrete example in which BCE can remove more clauses than VE, consider the following formula.

$$\begin{aligned} F = {} & (a \lor b \lor c) \land (\neg a \lor \neg b) \land (\neg a \lor \neg c) \land (\neg b \lor \neg c) \ \land \\ & (a \lor d \lor e) \land (a \lor d \lor \neg e) \land (a \lor \neg d \lor e) \land (a \lor \neg d \lor \neg e) \ \land \\ & (b \lor d) \land (b \lor e) \land (\neg b \lor \neg d) \land (\neg b \lor \neg e) \land (c \lor d) \land (c \lor e) \land (\neg c \lor \neg d) \land (\neg c \lor \neg e). \end{aligned}$$

Notice that F can be seen as a Boolean circuit in which each clause in F is represented by an OR-gate that is constrained to $\mathbf{t}$; for example, the clause $(a \lor b \lor c)$ is represented as $g_{(a \lor b \lor c)} := \textsc{or}(a,b,c)$. In fact, unit propagation on both the Tseitin and the Plaisted-Greenbaum encoding of this circuit gives back exactly F.

Now, it can be checked that VE cannot eliminate any of the variables in F. However, BCE can remove $(a \lor b \lor c)$ because the literal a is blocking the clause. □

On the other hand, there are also cases in which the combination of BCE and VE can be more effective than applying BCE or VE only. For instance, by applying VE on a CNF formula, new blocked clauses may arise.

Theorem 5 $[\mathrm{BCE}+\mathrm{VE}]_{\mathrm{TST}}$ *is strictly more effective than* $\mathrm{BCE}_{\mathrm{TST}}$.

Proof Consider a circuit with an XOR-gate $g := \textsc{xor}(g_1, g_2)$ where g_1 and g_2 are input gates with fanout one (non-shared). Assume that $g := \textsc{xor}(g_1, g_2)$ is rewritten as an AND-OR circuit structure $g := \textsc{and}(a,b)$, $a := \textsc{or}(g_1, g_2)$, $b := \textsc{or}(\textsc{not}(g_1), \textsc{not}(g_2))$, where a and b are newly introduced gates with fanout one (see Fig. 9). Notice that g_1 and g_2 now have fanout two. In the Tseitin encoding of this structure, BCE cannot see that g_1 and g_2 are non-shared in the underlying XOR. However, by first eliminating the OR-gates a and b with VE, BCE can then remove the clauses containing the variables g_1 and g_2 (the gates become implicitly “non-shared” again). □

In other words, there are cases in which variable elimination results in additional clauses to be blocked. On the other hand, since BCE can remove the clauses not produced by the Plaisted-Greenbaum encoding, $[\mathrm{BCE}+\mathrm{VE}]_{\mathrm{PG}}$ can not be more effective than $[\mathrm{BCE}+\mathrm{VE}]_{\mathrm{TST}}$.

Proposition 10 $[\mathrm{BCE}+\mathrm{VE}]_{\mathrm{TST}}$ *and* $[\mathrm{BCE}+\mathrm{VE}]_{\mathrm{PG}}$ *are equally effective.*

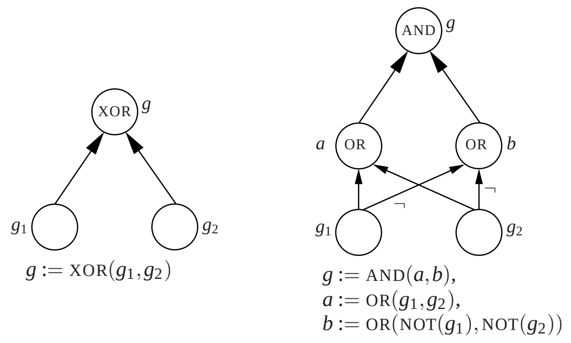
XOR
g
g_1
g_2
$g := \text{XOR}(g_1, g_2)$
AND
g
a
OR
OR
b
¬
¬
g_1
g_2
$g := \text{AND}(a, b),$
$a := \text{OR}(g_1, g_2),$
$b := \text{OR}(\text{NOT}(g_1), \text{NOT}(g_2))$

Fig. 6 An XOR-gate (left) and XOR rewritten as an AND-OR circuit structure (right)

10 Implementing BCE and Reconstructing Solutions

Before proceeding with results of an experimental evaluation on the effectiveness of BCE and VE, we will now explain how we have implemented BCE as part of our PrecoSAT solver (`http://fmv.jku.at/precosat`). Furthermore, we will address the related and practically relevant question of how to reconstruct original solutions to CNF formulas when applying BCE and VE (among other simplification techniques).

10.1 Implementing BCE

As explained in the following, BCE can be implemented in a similar way as VE in the SatElite preprocessor, which is described in [19]. Improved and simplified implementations of SatElite can be found in the source codes of MiniSAT 2.0 (`http://minisat.se/`) and PrecoSAT.

Basically the BCE algorithm is implemented as follows. First "touch" all literals. Then, as long as there is a touched literal l, find clauses that are blocked by l, mark l as not touched any more, remove these blocked clauses, and touch the negation of all literals in these clauses. Touched literals are kept in a priority list that is ordered by the number of occurrences. Literals with few occurrences are to be tried first. This algorithm, pseudo-code of which is given in Fig. 7, is in essence the basis for the implementation of BCE in PrecoSAT starting with version 465.

10.1.1 Some Practical Aspects

In practice, we have noticed that BCE, implemented as just described, takes far less time than the mentioned implementations of VE. In the common case that a literal does not block a clause, on average only a few tautological resolutions are performed before a non-tautological one is found. Thus in most application scenarios BCE can almost always be run until fixpoint (at least once).

Notice also that, similar to on-the-fly self-subsuming resolution [24], BCE can also be applied on-the-fly during VE. If elimination of a candidate variable would add too many resolvents and the variable is not eliminated, some of the antecedents could still have produced only tautologies. Such antecedents are thus blocked and can be removed by BCE.

```
touch(F,Q,l)
    determine number of occurrences of $l$ in $F$
    if there are no occurrences of $l$ in $F$ then return
    if $l \notin Q$ then enqueue$(Q,l)$
    update position of $l$ in $Q$ accordingly

bce(F)
    $Q$ = new empty priority queue of literals      // sorted by number of occurrences
    foreach literal $l$ in $F$ do touch$(F,Q,l)$
    while $Q \neq \emptyset$ do
        $l$ = dequeue$(Q)$                                  // dequeue minimum
        foreach clause $C \in F$ with $l \in C$ do          // start of outer loop
            foreach clause $D \in F$ with $\neg l \in D$ do
                if resolvent of $C$ with $D$ is non-tautological then
                    continue with next $C$ in outer loop
            // all resolvents of $C$ on $l$ in $F$ are tautologies and thus $C$ is blocked
            $F = F \setminus \{C\}$
            save $C$ on stack for solution reconstruction      // see Sect. 10.2
            foreach literal $k \in C$ do touch$(F,Q,\neg k)$
    return $F$
```

Fig. 7 Pseudo-code for implementing BCE for a CNF formula F.

10.2 Reconstructing Original Solutions

For many real application scenarios of SAT it is important to be able to extract a full satisfying assignment for original SAT instances from a satisfying assignment for the instances after preprocessing. For instance, notice that in Example 5 in Section 4, although BCE alone can show that the original formula is satisfiable, a solution to the original CNF formula is not directly available. We will now show how such full solutions can be efficiently reconstructed from solutions to the CNF formulas resulting from applying the combination of BCE and VE. Furthermore, we will show that *equivalence reasoning* [2,3,22], which is a further important simplification technique (and also implemented in e.g. PrecoSAT), does not interfere with the BCE reconstruction. The presented reconstruction techniques are both time and space wise linear, and hence have no real overhead w.r.t. solving.

We begin by describing how to reconstruct solutions for VE and BCE techniques separately, and then explain reconstruction in the combined case, along with an explanation of why equivalent literals do not interfere with this process.

10.2.1 Reconstruction for Variable Elimination

We start with variable elimination for which reconstruction can be seen as part of the completeness proof of DP. For the following, let $\mathrm{VE}(F,x)$ denote the result of applying variable elimination to F w.r.t. x.

Proposition 11 *Let τ be a satisfying assignment for* $\text{VE}(F,x)$. *Either τ_x or $\tau_{\neg x}$ satisfies $S_x \cup S_{\neg x}$, and, the one that does, also satisfies $F = \text{VE}(F,x) \cup (S_x \cup S_{\neg x})$.*

To reconstruct a solution after VE has been applied repeatedly for the variables $x_1,\ldots,x_m$, it is enough to save (remember) the clauses $(S_{x_1} \cup S_{\neg x_1}),\ldots,(S_{x_m} \cup S_{\neg x_m})$. Assume that τ satisfies $\text{VE}(\cdots\text{VE}(\text{VE}(F,x_1),x_2)\cdots,x_m)$. Let $\tau^{m+1} = \tau$, and, iteratively from $i = m$ to 1, define τ^i as the one of $\tau^{i+1}_{x_i}$ and $\tau^{i+1}_{\neg x_i}$ which satisfies $(S_{x_m} \cup S_{\neg x_m})$. Proposition 11 guarantees that τ^1 is a satisfying assignment for the original formula F.

If the application only requires to reconstruct one solution, then in practice[5] it is enough to only save either S_{x_i} or $S_{\neg x_i}$. W.l.o.g. assume S_{x_i} is saved. Then, if $\tau^{i+1}_{\neg x_i}$ satisfies the saved S_{x_i}, we pick $\tau^i = \tau^{i+1}_{\neg x_i}$, since this truth assignment obviously satisfies $S_{\neg x_i}$ as well. Otherwise x_i is forced to be **t** and we must set $\tau^i = \tau^{i+1}_{x_i}$. This case occurs if and only if there is a clause in S_{x_i} for which τ^{i+1} assigns all literals except x_i to **f**.

In an actual implementation only the smaller of the two sets is saved. Thus this technique is also efficient in the case where VE is used for pure literal elimination as discussed in Section 6.1. In addition to plain VE, it also works for functional substitution [19] as in the SatElite preprocessor. The only difference between VE and functional substitution is that the latter removes some redundant clauses from $S_x \otimes S_{\neg x}$ while maintaining the set of satisfying assignments.

10.2.2 Reconstruction Blocked Clause Elimination

Now consider solution reconstruction for BCE. In analogy to the case of VE, the proof [36] which shows that removal of a blocked clause does not turn an unsatisfiable formula into a satisfiable formula, gives us grounds to reconstruct solutions for BCE (see Proposition 3).

In practice it is enough to save all removed blocked clauses $C_1,\ldots,C_m$ together with their blocking literals $l_1,\ldots,l_m$.[6] Let τ^m be a satisfying assignment for F_m, where $F_i = F \setminus \cup_{j=1}^{i}\{C_j\}$ for $i = 1\ldots m$ and $F_0 = F$. If τ^i satisfies C_i, we pick $\tau^{i-1} = \tau^i$, and otherwise $\tau^{i-1} = \tau^i_{l_i}$. Using Proposition 3, one can show by induction that τ^i satisfies F_i, and thus τ^0 is a satisfying assignment for F.

10.2.3 Combined Solution Reconstruction

First, BCE and VE can be combined by saving clauses for reconstructing solutions after BCE resp. VE on the same reconstruction stack. Reconstruction works in reverse order in which these clauses have been saved. This also works nicely if BCE is applied on-the-fly during VE: while counting the non-tautological resolvents of $S_x \otimes S_{\neg x}$ to determine whether VE is applied to x, it may occur that a clause $C \in (S_x \cup S_{\neg x})$ has only tautological resolvents w.r.t. x, even though the overall number of non-tautological resolvents exceeds $|S_x \cup S_{\neg x}|$, which prevents x from being eliminated. Yet C can be removed as a blocked clause and is saved on the reconstruction stack.

[5] By private communication with Niklas Sörensson.

[6] A space efficient way to save this information is to maintain l_i as the first literal in the saved clause C_i. This also allows to keep track of eliminated variables in VE.

10.2.4 Equivalent Literals and Solution Reconstruction

In addition to BCE and VE, various other simplification techniques can be exploited within the SAT solving process. One well-known and often useful techniques is *equivalent literal reduction*.

For two literals l_1 and l_2, let $l_1 \equiv l_2$ denote the CNF formula $(l_1 \vee \neg l_2) \wedge (\neg l_1 \vee l_2)$. For a given CNF formula F, if $F \vdash l_1 \equiv l_2$ (that is, both of the clauses $(l_1 \vee \neg l_2)$ and $(\neg l_1 \vee l_2)$ can be derived from F), the equivalent literals l_1 and l_2 can be exploited by the equivalence reduction in which all occurrences of l_2 are substituted by l_1 (or vice versa), eliminating the variable of l_2 (or l_1).

In order to combine VE and equivalent literal reduction it is enough to make sure that VE is only attempted after all equivalent literals have been first substituted. Enforcing this order of using equivalent literal reasoning and VE makes sure that variables eliminated with VE are always representatives and the only remaining variables of their equivalence class. Eliminating a representative through VE will eliminate its whole equivalence class, and after this it is not possible that further equivalent literals could be added to the equivalence class of an eliminated variable.

In contrast to the case of combining BCE and VE, when combining BCE with equivalent literal reduction, the question of solution reconstruction appears more intricate: at some point after removing a blocked clause C, a literal l which blocked C may become equivalent to another literal and may even become a representative of its equivalence class. On the other hand, it could be that one would be forced to flip the value of l during solution reconstruction since BCE removed C (recall Section 10.2.2). Hence the values of all the literals in the equivalence class should be flipped, which does not seem to be sound in general, since this could make some other clause unsatisfied. However, we will now show that the value of such a l will *never* have to be flipped in such a situation during the BCE reconstruction step. The following example highlights this fact.

Example 10 Consider the CNF formula

$$F = (x \vee y) \wedge (x \vee \neg z) \wedge (\neg x \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (y \vee z).$$

Notice that the clause $(x \vee y)$ is blocked by y w.r.t. F. Assume that BCE removes $(x \vee y)$ from F. Now, resolving on y, one can derive from the clauses $(\neg x \vee \neg y \vee z)$ and $(y \vee z)$ the clause $(\neg x \vee z)$. Together with $(x \vee \neg z) \in F \setminus \{(x \vee y)\}$, $(\neg x \vee z)$ forms the equivalence $x \equiv z$.

Consider the truth assignment τ which assigns $\tau(x) = \mathbf{f}$, $\tau(y) = \mathbf{t}$, and $\tau(z) = \mathbf{f}$. Notice that τ satisfies $F \setminus \{(x \vee y)\}$. Moreover, it also satisfies $(x \vee y)$, and hence one does not need to flip the assignment on x in the BCE reconstruction step for $(x \vee y)$. However, if one would still flip the assignment on x to $\tau(x) = \mathbf{t}$, one would then need to flip the assignment on z to $\tau(z) = \mathbf{t}$ due to the equivalence $x \equiv z$. Notice that these flips together result in the assignment $\tau(x) = \tau(y) = \tau(z) = \mathbf{t}$ that does not satisfy the clause $(\neg x \vee \neg z) \in F$. ■

Especially, equivalent literals detected and applied in simplifying a CNF formula after removing blocked clauses cannot make the removed blocked clauses to not to be satisfied under a satisfying assignment for the rest of the formula. For proving this result, we recall some well-known concepts related to resolution proofs. A sequence of clauses $(C_0, C_1, \ldots, C_n)$ is a resolution derivation of the clause C from a CNF formula F if (i) $C_n = C$, and (ii) each C_i, where $0 \leq i < n$, is either a clause in F (in this case C_i is called an *input clause*), or C_i is the resolvent of two clauses C_j and C_k, where $j, k < i$. We denote by $F \vdash C$ the fact that there is a resolution derivation of the clause C from the CNF formula F. A well-known refinement of resolution is tree-like resolution, where derivations have to be representable as trees.

Theorem 6 *Assume a CNF formula F, a clause $C \in F$ which is blocked for $l \in C$ w.r.t. F, and a literal l'. If $F \setminus \{C\} \vdash l \equiv l'$, then $(F \setminus \{C\}) \cup (l \equiv l') \models C$.*

In other words, any satisfying assignment for $(F \setminus \{C\}) \cup (l \equiv l')$ also satisfies the blocked clause C. This means that binary equivalences detected during preprocessing can be exploited when applying BCE, at the same time guaranteeing all the blocked clauses removed by BCE will be satisfied by any satisfying assignment for the resulting preprocessed CNF formula. Notice that this lemma is independent of the techniques used for deriving the clauses in $l \equiv l'$.

Proof (of Theorem 6) Assume a CNF formula F, a clause $C = (l \vee l_1 \vee \cdots \vee l_k) \in F$ which is blocked for $l \in C$ w.r.t. F. Denote by $B \subset F$ the set of clauses which contain the literal $\neg l$. Hence each clause in B contains at least one of the literals $\neg l_1, \ldots, \neg l_k$. Assume that $F \setminus \{C\} \vdash l \equiv l'$ for some literal l', and hence there is a resolution derivation of $(l \vee \neg l')$ and $(\neg l \vee l')$ from $F \setminus \{C\}$.

If F is unsatisfiable, $F \setminus \{C\}$ is also unsatisfiable since C is blocked, and hence trivially $(F \setminus \{C\}) \cup (l \equiv l') \models C$. Now consider the case that F and (thus) also $F \setminus \{C\}$ and $(F \setminus \{C\}) \cup (l \equiv l')$ are satisfiable. Take an arbitrary satisfying assignment τ for $(F \setminus \{C\}) \cup (l \equiv l')$. We will show that any such τ also satisfies C.

The case in which $\tau(l) = \mathbf{t}$ (that is, τ satisfies l) is trivial. Now assume $\tau(l) = \mathbf{f}$. Then $\tau(l') = \mathbf{f}$ since τ satisfies $l \equiv l'$. Consider an arbitrary resolution derivation $\pi = (C_1, \ldots, C_m)$ of $C_m = (\neg l \vee l')$ from $F \setminus \{C\}$. Assume w.l.o.g. that π is tree-like. We claim that there is an input clause $C' = (\neg l \vee l'_1 \vee \cdots \vee l'_k) \in B$ in π such that $\tau(l'_i) = \mathbf{f}$ for all i. Since $C' \in B$, it then follows that one of the l'_is is one of the literals $\neg l_1, \ldots, \neg l_k$, and hence τ satisfies C (recall that $C = (l \vee l_1 \vee \cdots \vee l_k)$).

To prove the claim, we show that there is a path $P_1, \ldots, P_n$ of clauses in π (seen as a tree) from the root of the tree ($P_1 = C_m$) to a leaf (P_n is an input clause of π), such that each clause P_i on the path contains $\neg l$ and τ assigns all literals in P_i except $\neg l$ to $\mathbf{f}$.

First notice that for $P_1 = C_m$ we know that $\tau(\neg l) = \mathbf{t}$ and $\tau(l') = \mathbf{f}$. Now assume that $P_i = \{\neg l\} \cup D$, where D is a set of literals such that τ assigns every literal in D to $\mathbf{f}$, was directly derived from clauses C_a and C_b in π resolving on the variable x. Notice that at least one of C_a and C_b must contain $\neg l$. First consider the case that C_a contains $\neg l$ and C_b does not. Since τ assigns all literals in D to $\mathbf{f}$, τ must satisfy the literal for x in C_b. (Otherwise τ does not satisfy C_b which would imply that τ does not satisfy an input clause in π and hence τ cannot be a satisfying truth assignment for $(F \setminus \{C\}) \cup (l \equiv l')$, in contradiction to our assumption.) Hence τ assigns all literals in C_a apart from $\neg l$ to $\mathbf{f}$. In this case let $P_{i+1} = C_a$. The case that C_b contains $\neg l$ and C_a does not is identical.

Now consider the case that both C_a and C_b contain $\neg l$. Since τ assigns a unique truth value to x, τ assigns all literals in either C_a or C_b apart from $\neg l$ to $\mathbf{f}$. In this case let P_{i+1} be this particular clause. $\square$

11 Experiments

In this section we present results of experiments on how much reduction can be achieved using BCE in combination with VE and various circuit encoding techniques. Here reduction is measured in the size of the CNF formula before and after preprocessing, and on the other hand, as gain in the number of instances solved.

11.1 Experiment Setup

We used all formulas of SMT-Lib (`http://smtlib.org`) over the theory of bit-vectors (QF_BV) made available on July 2, 2009, as a practice benchmark set for the SMT competition 2009. From these we removed the large number of mostly trivial SAGE examples. The remaining 3672 SMT problems were bit-blasted to And-Inverter Graphs (AIGs) in the AIGER format (`http://fmv.jku.at/aiger`) using our SMT solver Boolector [10]. Furthermore, we used the AIG instances used in [13], consisting of two types of instances: (i) AIGs representing BMC problems (with step bound $k = 45$) obtained from all the 645 sequential HWMCC'08 (`http://fmv.jku.at/hwmcc08`) model checking problems, and (ii) 62 AIGs from the structural SAT track of the SAT competition. We have made the SMT-Lib instances publicly available at `http://fmv.jku.at/aiger/smtqfbv-aigs.7z` (260MB); the others cannot be distributed due to license restrictions. However, the HWMCC'08 instances can easily be regenerated using publicly available tools[7] and the model checking benchmarks available at `http://fmv.jku.at/hwmcc08`.

11.2 Results on Achieved Simplifications

We encoded these 4379 structural SAT instances with four algorithms: the standard Tseitin encoding [47], the Plaisted-Greenbaum polarity-based encoding [43], the Minicirc encoder based on technology mapping [20] and VE, and the most recent NiceDAG encoder [40,13]. The NiceDAG implementation was obtained from the authors. For Minicirc, we used an improved implementation of Niklas Eén.

In order to additionally experiment with application benchmarks already in CNF, we also included 292 CNF formulas of the application track of the SAT competition 2009 to our benchmark set. All resulting CNF formulas were preprocessed with VE alone (further abbreviated e), and separately first with BCE (b), followed by VE (e), and both repeated again, which altogether gives 6 versions of each CNF formula (no BCE or VE, e, b, be, beb, bebe). We call such an application of one preprocessing algorithm, either BCE or VE, which is run to completion, a *preprocessing phase*.

The results are presented in Table 2. The first column lists the benchmark family: S = SAT'09 competition, A = structural SAT track, H = HWMCC'08, B = bit-blasted bit-vector problems from SMT-Lib. These are all AIGs except for the CNF formulas in S. The next column gives the encoding algorithm used: T = Tseitin, P = Plaisted-Greenbaum, M = Minicirc, N = NiceDAG, and U = unknown for the S family already in CNF. The t columns give the sum of the time in seconds spent in one encoding/preprocessing phase. The columns V and C list in millions the sum of numbers of variables and clauses over all produced CNF formulas in each phase.

We applied a time limit of 900 seconds and a memory limit of 4096 MB for each encoder and each preprocessing phase. Thus 139 out of $106848 = 6 \cdot (4 \cdot 4379 + 292)$ CNF formulas were not generated: HM encoding ran out of memory on 5 very large BMC instances, one large CNF formula in S could not be preprocessed at all, and there was a problem with the parser in NiceDAG, which could not parse 14 actually rather small AIGs in BN. Furthermore, there were 10 timeouts for various preprocessing phases in the A family: 2 in AT/beb,

[7] Notice that COI is performed already in the generation process by these tools. However, we did not implement the non-trivial NSI or MIR for the experiments.

Table 2 Effectiveness of BCE in combination with VE using various encoders. Values in the table are sums over all instances in the specific benchmark family: S = SAT'09 competition, A = structural SAT track, H = HWMCC'08, B = bit-blasted bit-vector problems from SMT-Lib. Circuit encoders: T = Tseitin, P = Plaisted-Greenbaum, M = Minicirc, N = NiceDAG, and U = unknown for the S family already in CNF. Columns: t = sum of running times over all instances in the specific benchmarks family, V = sum of numbers of variables in the resulting CNF formulas in millions, C = sum of numbers of clauses in the resulting CNF formulas in millions. The CNF-level preprocessors run after the circuit encoders: encoding = no CNF-level preprocessing, b = BCE, e = VE, be = BCE followed by VE, beb = be followed by BCE, bebe = beb followed by VE.

		encoding			b			be			beb			bebe			e		
		t	V	C	t	V	C	t	V	C	t	V	C	t	V	C	t	V	C
S	U	0	46	256	2303	29	178	1042	11	145	1188	11	145	569	11	144	2064	11	153
A	T	12	9	27	116	7	18	1735	1	8	1835	1	6	34	1	6	244	1	9
A	P	10	9	20	94	7	18	1900	1	6	36	1	6	34	1	6	1912	1	6
A	M	190	1	8	42	1	7	178	1	7	675	1	7	68	1	7	48	1	8
A	N	9	3	10	50	3	10	1855	1	6	36	1	6	34	1	6	1859	1	6
H	T	147	121	347	1648	117	277	2641	18	118	567	18	118	594	18	116	3240	23	140
H	P	130	121	286	1398	117	277	2630	18	118	567	18	118	595	18	116	2835	19	119
H	M	6961	16	91	473	16	84	621	12	78	374	12	77	403	12	76	553	15	90
H	N	134	34	124	573	34	122	1185	17	102	504	17	101	525	17	100	1246	17	103
B	T	577	442	1253	5799	420	1119	7023	57	321	1410	56	310	1505	52	294	8076	64	363
B	P	542	442	1153	5461	420	1119	7041	57	321	1413	56	310	1506	52	294	7642	57	322
B	M	10024	59	311	1252	58	303	1351	53	287	1135	53	286	1211	52	280	1435	55	303
B	N	13148	196	643	2902	193	635	4845	108	508	2444	107	504	2250	105	500	5076	114	518

2 in AN/be, 2 in AN/e, 2 in AP/be, and 2 in AP/e. However, except for the one large CNF formula, where also VE run out of memory, there is not a single case where BCE did not run until completion within the given time and memory limits.

The results show that the combination “be” of BCE and VE always gives better results than VE (e) alone, with comparable speed. Using a second phase (beb) of BCE gives further improvements, even more if VE is also applied a second time (bebe). The CNF sizes after applying BCE (b) for the P encoder and the T encoder are equal, as expected. Further preprocessing, however, diverges: since clauses and literals are permuted, VE is not confluent, and thus VE phases can produce different results.

Notice that BCE on the Tseitin encoding (row T, column b) removes more clauses and variables than what are removed by the Plaisted-Greenbaum encoding (row P, column “encoding”) for each of the three AIG instance families A, H, and B, which is in line with out analysis on the effectiveness of BCE. A further interesting aspect to notice is that the combination bebe of BCE and VE compares well against the Minicirc and NiceDAG encoders. Especially, on the bit-vector AIG instances (rows B), the bebe combination on the Tseitin encoding (row T, column bebe) removes more clauses and variables than both of Minicirc (row M, column “encoding”) and NiceDAG (row N, column “encoding”. This is true even when VE is run after these circuit encoders (rows M and N, column e). While the remaining numbers of clauses and variables are in the same range as for Minicirc, the difference between bebe and NiceDAG is notable: the numbers of clauses and variables produced by NiceDAG are close to double the numbers for bebe. For the implementations used in this experiment, the total running time on the bit-vector AIG instances for the combination of Tseitin encoding and applying bebe was around $16,000$ seconds, compared to around $10,000$ and $13,000$ for Minicirc and NiceDAG, respectively, without VE, and around $11,500$ and $18,000$ with VE, respectively.

An alternative view to the reduction achieved by BCE, VE, and their combinations is given in Figures 8 and 9. The plot in Figure 8 shows the absolute sizes of the original and preprocessed CNF formulas. The horizontal axis ranges over all CNF formulas sorted for each preprocessing phase individually with respect to the number of clauses in the CNF formulas (similarly as in “cactus” plots used in presenting the results of the SAT competitions). The vertical axis gives the number of clauses in the sorted formulas. The plot in Figure 9 shows the percentage of clauses remaining after preprocessing. As the plots shows blocked clause elimination (“b”) already reduces CNF size. Variable elimination alone (“e”) is considerably more effective, but can be improved by combining it with blocked clause elimination (“be”). This trend continues if these preprocessing techniques are applied repeatedly (twice in “bebe”).[8]

Notice that, in addition to “be”, “beb”, and “bebe”, one could also study the effect of the variants “eb”, “ebe”, and “ebeb”. However, we do not consider these variants here due to the following reasons. First, notice that by eliminating clauses using BCE can only *increase* the number of possible variable eliminations by VE_k for any k, since removing clauses does not increase the number of variable occurrences. Second, notice that the property of a clause being blocked is maintained by VE in the sense that any resolvent of a blocked clause either remains blocked (in case the eliminated variable does not contribute to the fact that the clause is blocked) or is a tautology (otherwise). Hence applying BCE *after* each round of VE can eliminate resolvents of blocked clauses and also increase the benefit of the next round of VE, actually re-enabling VE.

[8] VE and BCE are idempotent: “ee = e” and “bb = b”.

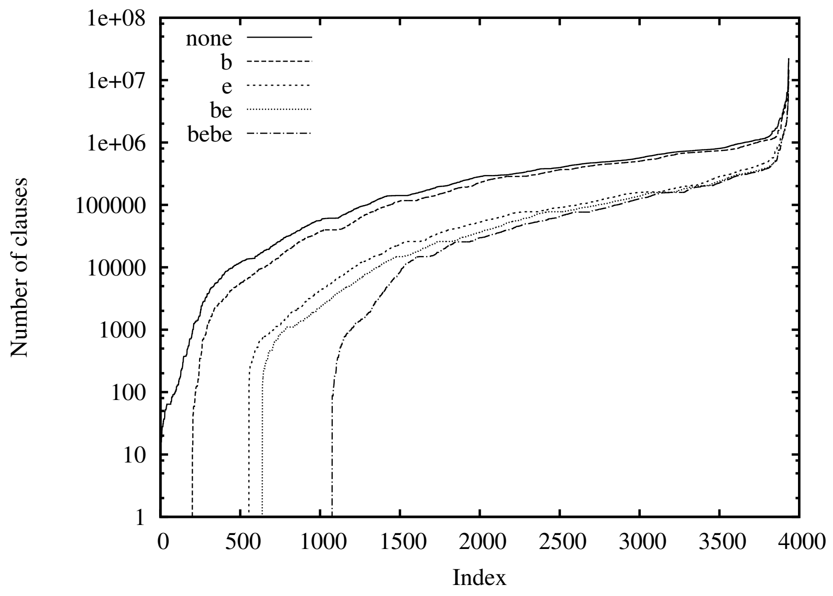

Fig. 8 Number of clauses in preprocessed CNF formulas starting from the Tseitin encoding ("none"), followed by one round of blocked clauses elimination ("b"), one round of variable elimination ("e"), combination blocked clause and variable elimination once ("be") and twice ("bebe"). The horizontal axis ranges over CNF formulas sorted for each preprocessing phase individually with respect to the number of clauses in the CNF formula(similarly as in "cactus" plots used in presenting the results of the SAT competitions). The vertical axis gives the absolute number of remaining clauses in each CNF formula on a logarithmic scale.

11.3 Results on the Effect on Solving Times

Reducing the size of a CNF formula by preprocessing does not necessarily lead to faster running times. In this section we address the question of how applying BCE can affect state-of-the-art CDCL and local search SAT solvers.

11.3.1 Effect of BCE *on CDCL Solvers*

Although it was impossible to run all structural instances with a large time limit, we performed preliminary experiments with a time limit of 90 seconds. We used PrecoSAT v236, the winner of the application track of the SAT competition 2009. The results were inconclusive. Running preprocessing until completion takes a considerable portion of the 90 seconds time limit, even if restricted to VE.

It should be noted that the success of PrecoSAT shows that *inprocessing*, i.e., interleaving preprocessing with CDCL search (and thus not running preprocessing until completion) is a much better strategy than typical run-to-completion preprocessing, particularly if preprocessors are run during inprocessing repeatedly, with enough time spent on search in-between. However, this strategy is difficult to evaluate objectively when many preprocessing techniques are combined.[9] Therefore, for these experiments, we decided to stick

[9] In PrecoSAT we have failed literal preprocessing, various forms of equivalence reasoning, explicit pure literal pruning, BCE, VE, combined with on-the-fly subsumption.

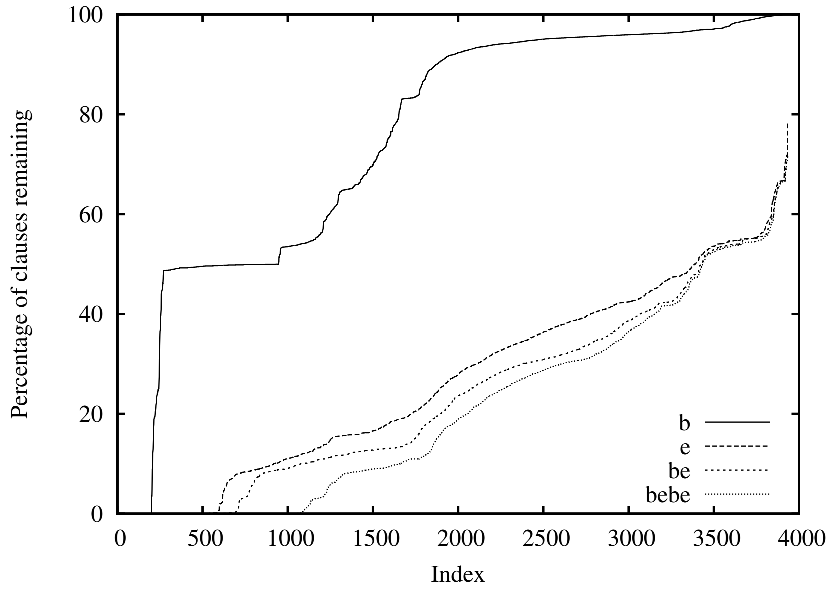

Fig. 9 Size reduction by preprocessing after one round of blocked clauses elimination ("b"), one round of variable elimination ("e"), combination blocked clause and variable elimination once ("be") and twice ("bebe"). The horizontal axis ranges over CNF formulas sorted for each preprocessing phase individually with respect to the size reduction measured in the number of remaining clauses. The vertical axis gives the percentage of clauses remaining in each CNF formula.

with the run-to-completion approach, which also gives some clear indication of how much CNF size reduction can be achieved through BCE.

For the 292 SAT competition instances we were able to run PrecoSAT with a more reasonable timeout of 900 seconds. The cluster machines used for the experiments, with Intel Core 2 Duo Quad Q9550 2.8-GHz processor, 8-GB main memory, running Ubuntu Linux version 9.04, are around two times as fast as the ones used in the first phase of the 2009 SAT competition. In the first phase of the competition, with a similar time limit, PrecoSAT solved many more instances than competitors. The results of this experiment show that using BCE is somewhat beneficial: PrecoSAT solves 176 original instances, 177 preprocessed by BCE and VE alone (b and e, respectively), 179 be instances, 180 beb instances, and 183 bebe instances. If we accumulate the time for all the preprocessing phases and add it to the actual running time, then 181 instances can be solved in the last case. For the other cases the number of solved instances does not change.

These numbers are confirmed by similar experiments with MiniSAT 2.2.0 as the backend solver. Plain MiniSAT without preprocessing solves 154 instances. Enabling the internal preprocessor of MiniSAT, which implements a variant of VE, gives 169 solved instances. Combining blocked clause elimination with this version solves 171 instances. In this last experiment we used PrecoSAT 465 as BCE preprocessor (command line switch "`-k -p`") and the result was fed into MiniSAT. The time is measured by adding up preprocessing time and the time of running MiniSAT.

11.3.2 Effect of BCE *on Local Search*

We now turn to the question of the effect of applying BCE on CNF-level stochastic local search (SLS) for SAT. It is well-known that, at present, CNF-level SLS solvers are most often highly inferior to CDCL solvers on real-world application instance families. Examples of such families of instances include the instances resulting from CNF encodings of AIGs as well as the SAT competition application benchmarks which we used in Sections 11.2 and 11.3.1 for evaluating the simplification power of BCE and VE, and the performance of state-of-the-art CDCL solvers when applying BCE. Thus, for evaluating the effect of BCE on the efficiency of CNF-level SLS solvers, we need to consider alternative sources of instances.

It is well-known that SLS solvers show strong performance on randomly generated SAT problems. However, we have noticed that random *k*-SAT instances contain almost no blocked clauses, especially when considering the most difficult instances taken from the satisfiability threshold.

For a possible alternative source of benchmark instances that satisfy both of our requirements that (i) state-of-the-art SLS solvers perform well on the instances, and (ii) the instances contain a non-negligible number of blocked clauses, we analyzed the results of the crafted satisfiable instance category of the 2011 SAT Competition (`http://satcompetition.org/2011`). Among those instance families, we were able to pinpoint three interesting families of instances: `frb`, `rbsat`, and `srhd`.

We solved these instances with the adaptg2wsat+p SLS algorithm [38] using the implementation available in the UBCSAT SLS version 1.1 solver [46] (`http://www.satlib.org/ubcsat/`). To the best of our knowledge, adaptg2wsat+p is among the best current SLS solvers for such crafted SAT instances. We ran the solver on each instance 1000 times using random seeds without timeout. The results of this experiment are presented in Table 3 for representative instances from each family. In the table, “flips” and “time” represent the median number of flips performed and the median running times in seconds used for each instance, respectively. We note that the time needed to apply BCE on these instances was negligible (typically less than 0.1 seconds). The effect of BCE seems to be consistent for all the instances: removing blocked clauses by BCE does not drastically influence the number of flips required to solve any of the instances. However, interestingly the solving times are improved by a non-negligible amount. We suspect that this is due to the fact the computational cost per flip is reduced after removing blocked clauses with BCE since smaller lookup tables are needed on the implementation level for performing the actual search.

The fact that BCE does not appear to have a notable influence on the number of flips performed by SLS leads to the following conjecture: blocked clauses are often either satisfied or, when falsified, easy to satisfy during local search. For some intuition, consider an arbitrary complete assignment τ over the variables of a CNF formula F. Assume that τ characterizes the current configuration of a CNF-level SLS solver. Take any blocked clause $C \in F$, and let $l \in C$ be a blocking literal. Now, if C is falsified by τ, then we know by the definition of blocked clauses that τ satisfies at least two literals of each clause in F in which the literal $\neg l$ occurs. Hence by flipping l to $\mathbf{t}$, C becomes satisfied while all clauses satisfied by τ remain satisfied.

Table 3 The effect of applying BCE on the performance of the adaptg2wsat+p SLS algorithm when solving representative instances of three families from the crafted satisfiable track of the 2011 SAT Competition. The columns have the following explanations: size = the size of the CNF formula (original and after applying BCE, flips = the median number of flips performed by adaptg2wsat+p, time = the median running time of adaptg2wsat+p in seconds.

	Original CNF formula			CNF formula after BCE		
CNF formula	size	flips	time (s)	size	flips	time (s)
`frb45-21-1`	61855	2600763	12.47	49961	2640216	9.70
`frb75-13-2`	34275	181756	0.50	27588	181531	0.39
`frb80-14-2`	43179	1129643	3.50	34948	1131352	2.77
`rbsat-v1150c84314g1`	84314	7737733	43.24	67882	7042883	31.01
`rbsat-v945c61409g3`	61409	3129700	14.47	51959	3231077	12.23
`rbsat-v945c61409gyes1`	61409	4591362	21.17	51959	4273371	15.93
`srhd-m27-q255-n25-p30`	45238	3242589	13.10	36691	3372989	11.77
`srhd-m32-q369-n30-p15`	82294	1357707	7.39	69274	1369229	6.78
`srhd-m37-q446-n35-p15`	129272	2100255	18.65	112111	1999134	12.93

12 Conclusions

This work addresses the important question of interplay between problem structure and practical reasoning techniques for Boolean satisfiability. The focus is on analyzing conjunctive normal form (CNF) level reasoning techniques and relating the behavior of such techniques with the behavior of known techniques that work on more structural representation forms of Boolean satisfiability instances, especially, on the level of Boolean circuits. In more detail, we analyzed the two CNF-level simplification techniques of SatElite-style variable elimination (VE) and what we call blocked clause elimination (BCE). We showed that BCE, although a simple concept, is surprisingly effective: without any explicit knowledge of the underlying circuit structure, BCE achieves the same simplifications as combinations of circuit-level simplifications and the well-known polarity-based Plaisted-Greenbaum CNF encoding. This implies that the effect of such specialized circuit-level techniques can actually be systematically accomplished directly on the CNF-level. Furthermore, in contrast to specialized circuit-level techniques, BCE can be naturally applied on any CNF formula, regardless of its origin. We also showed that VE can achieve many of the same effects as BCE (but not all). It turns out that VE and BCE are indeed partially orthogonal techniques, which motivates combining these two techniques for achieving even better simplification. Further, we showed how witnesses to original CNF formulas can be reconstructed from solutions acquired after applying combinations of BCE, VE, and equivalent literal reduction on the formulas. Experimental results with an implementation of a CNF-level preprocessor combining BCE and VE show that BCE can be applied effectively and efficiently, though the improvement due to using BCE in combination with VE with respect to solving more instances appears to be at most modest.

It is also possible to improve SAT solver running times by *adding* (instead of *removing* by BCE) blocked clauses to CNF formulas in intelligent ways. While motivated by proof complexity theoretical arguments [36] as well as the more practical evidence presented in [26, 34] using manually added domain-specific blocked clauses, this question poses multiple challenges. For example, in contrast to BCE, adding blocked clauses is not confluent, and in general there is an exponential number of possibilities to introduce blocked clauses to CNF formulas. In fact, without restricting the focus on variables that already occur in a formula, allowing one to add arbitrary blocked clauses into a formula covers the extension rule of the extremely powerful Extended Resolution proof system [47, 36]. While the possibilities of applying the extension rule within CDCL SAT solvers have been studied [1, 27], the resulting implementations do not yet reach the full potential on specific families of CNF formulas that are known to be easy for extended resolution but hard for resolution (including pigeon-hole formulas). This hints to the direction that it is difficult to exploit the potential of blocked clause addition even for formulas for which it is known that this technique could—in principle—improve solving times significantly. We note that studying possibilities of improving SAT solver running times by developing novel ways of *automatically adding* blocked clauses is out of the scope of this work. However, it is an interesting question whether one could benefit from developing good heuristics for applying a combination of restricted BCE and addition of blocked clauses. Our first, very preliminary experiments on automatically adding blocked clauses, performed after the acceptance of this article, has led us to conjecture that it is at least non-trivial to enhance the performance of CDCL solvers via adding blocked clauses. However, understanding the duality between eliminating and adding blocked clauses for practical purposes remains as interesting future work.

Acknowledgements The authors thank Niklas Eén and Pete Manolios, respectively, for providing up-to-date versions of the Minicirc and NiceDAG encoders used in the experiments. The first author thanks Tommi Junttila for multiple extended discussions related to the circuit simplifications implemented in the `bc2cnf` tool.

References

1. Audemard, G., Katsirelos, G., Simon, L.: A restriction of extended resolution for clause learning SAT solvers. In: M. Fox, D. Poole (eds.) Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010). AAAI Press (2010)
2. Bacchus, F.: Enhancing Davis Putnam with extended binary clause reasoning. In: Proceedings of the 18th National Conference on Artificial Intelligence (AAAI 2002), pp. 613–619. AAAI Press (2002)
3. Bacchus, F., Winter, J.: Effective preprocessing with hyper-resolution and equality reduction. In: Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003), *Lecture Notes in Computer Science*, vol. 2919, pp. 341–355. Springer (2004)
4. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Biere et al. [6], pp. 825–885
5. Biere, A., Clarke, E.M., Raimi, R., Zhu, Y.: Verifiying safety properties of a power PC microprocessor using symbolic model checking without BDDs. In: N. Halbwachs, D. Peled (eds.) Proceedings of the 11th International Conference on Computer Aided Verification (CAV 1999), *Lecture Notes in Computer Science*, vol. 1633, pp. 60–71. Springer (1999)
6. Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, *Frontiers in Artificial Intelligence and Applications*, vol. 185. IOS Press (2009)
7. Biere, A., Lonsing, F., Seidl, M.: Quantified blocked clause elimination. In: Proceedings of the 23nd International Conference on Automated Deduction (CADE-23), Lecture Notes in Computer Science. Springer (2011)
8. Boy de la Tour, T.: An optimality result for clause form translation. Journal of Symbolic Computation **14**(4), 283–302 (1992)
9. Brafman, R.I.: A simplifier for propositional formulas with many binary clauses. IEEE Transactions on Systems, Man, and Cybernetics, Part B **34**(1), 52–59 (2004)
10. Brummayer, R., Biere, A.: Boolector: An efficient SMT solver for bit-vectors and arrays. In: S. Kowalewski, A. Philippou (eds.) Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009), *Lecture Notes in Computer Science*, vol. 5505, pp. 174–177. Springer (2009)
11. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R.: The MathSAT 4 SMT solver. In: A. Gupta, S. Malik (eds.) Proceedings of the 20th International Conference on Computer Aided Verification (CAV 2008), *Lecture Notes in Computer Science*, vol. 5123, pp. 299–303. Springer (2008)

12. Bruttomesso, R., Pek, E., Sharygina, N., Tsitovich, A.: The OpenSMT solver. In: J. Esparza, R. Majumdar (eds.) Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2010), *Lecture Notes in Computer Science*, vol. 6015, pp. 150–153. Springer (2010)
13. Chambers, B., Manolios, P., Vroon, D.: Faster SAT solving with better CNF generation. In: Proceedings of Design, Automation and Test in Europe (DATE 2009), pp. 1590–1595. IEEE (2009)
14. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM **7**(3), 201–215 (1960)
15. de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: C.R. Ramakrishnan, J. Rehof (eds.) Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008), *Lecture Notes in Computer Science*, vol. 4963, pp. 337–340. Springer (2008)
16. Drechsler, R., Junttila, T., Niemelä, I.: Non-clausal SAT and ATPG. In: A. Biere, M.J.H. Heule, H. van Maaren, T. Walsh (eds.) Handbook of Satisfiability, *Frontiers in Artificial Intelligence and Applications*, vol. 185, chap. 21, pp. 655–694. IOS Press (2009)
17. Dunham, B., Fridshal, R., Sward, G.: A heuristic program for proving elementary logical theorems. In: Proceedings of the International Conference on Information Processing (IFIP 1959), pp. 282–284 (1959)
18. Dunham, B., Wang, H.: Towards feasible solutions of the tautology problem. Annals of Mathematical logic **10**, 117–154 (1976)
19. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: F. Bacchus, T. Walsh (eds.) Proceedings of 8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005), *Lecture Notes in Computer Science*, vol. 3569, pp. 61–75. Springer (2005)
20. Eén, N., Mishchenko, A., Sörensson, N.: Applying logic synthesis for speeding up SAT. In: J. Marques-Silva, K.A. Sakallah (eds.) Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT 2007), *Lecture Notes in Computer Science*, vol. 4501, pp. 272–286. Springer (2007)
21. Ganesh, V., Dill, D.L.: A decision procedure for bit-vectors and arrays. In: W. Damm, H. Hermanns (eds.) Proceedings of the 19th International Conference on Computer Aided Verification (CAV 2007), *Lecture Notes in Computer Science*, vol. 4590, pp. 519–531. Springer (2007)
22. Gershman, R., Strichman, O.: Cost-effective hyper-resolution for preprocessing CNF formulas. In: F. Bacchus, T. Walsh (eds.) Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005), *Lecture Notes in Computer Science*, vol. 3569, pp. 423–429. Springer (2005)
23. Han, H., Somenzi, F.: Alembic: An efficient algorithm for CNF preprocessing. In: Proceedings of the 44rd Design Automation Conference (DAC 2007), pp. 582–587 (2007)
24. Han, H., Somenzi, F.: On-the-fly clause improvement. In: O. Kullmann (ed.) SAT, *Lecture Notes in Computer Science*, vol. 5584, pp. 209–222. Springer (2009)
25. Heule, M.J.H., van Maaren, H.: Aligning CNF- and equivalence-reasoning. In: H.H. Hoos, D.G. Mitchell (eds.) SAT 2005 Selected Revised Papers, *Lecture Notes in Computer Science*, vol. 3542, pp. 145–156. Springer (2005)
26. Heule, M.J.H., Verwer, S.: Exact DFA identification using SAT solvers. In: J.M. Sempere, P. García (eds.) Proceedings of the 10th International Colloquium on Grammatical Inference: Theoretical Results and Applications (ICGI 2010), *Lecture Notes in Computer Science*, vol. 6339, pp. 66–79 (2010)
27. Huang, J.: Extended clause learning. Artificial Intelligence **174**(15), 1277–1284 (2010)
28. Jackson, P., Sheridan, D.: Clause form conversions for Boolean circuits. In: H.H. Hoos, D.G. Mitchell (eds.) SAT 2004 Selected Revised Papers, *Lecture Notes in Computer Science*, vol. 3542, pp. 183–198. Springer (2005)
29. Järvisalo, M., Biere, A.: Reconstructing solutions after blocked clause elimination. In: O. Strichman, S. Szeider (eds.) Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT 2010), *Lecture Notes in Computer Science*, vol. 6175, pp. 340–345. Springer (2010)
30. Järvisalo, M., Biere, A., Heule, M.J.H.: Blocked clause elimination. In: J. Esparza, R. Majumdar (eds.) Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2010), *Lecture Notes in Computer Science*, vol. 6015, pp. 129–144. Springer (2010)
31. Jha, S., Limaye, R., Seshia, S.A.: Beaver: Engineering an efficient SMT solver for bit-vector arithmetic. In: A. Bouajjani, O. Maler (eds.) Proceedings of the 21st International Conference on Computer Aided Verification (CAV 2009), *Lecture Notes in Computer Science*, vol. 5643, pp. 668–674. Springer (2009)
32. Jin, H., Somenzi, F.: An incremental algorithm to check satisfiability for bounded model checking. Electronic Notes in Theoretical Computer Science **119**(2), 51–65 (2005)
33. Jussila, T., Biere, A.: Compressing BMC encodings with QBF. Electronic Notes in Theoretical Computer Science **174**(3), 45–56 (2007)

34. Kautz, H.A., Ruan, Y., Achlioptas, D., Gomes, C.P., Selman, B., Stickel, M.E.: Balance and filtering in structured satisfiable problems. In: B. Nebel (ed.) Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001), pp. 351–358. Morgan Kaufmann (2001)
35. Kullmann, O.: New methods for 3-SAT decision and worst-case analysis. Theoretical Computer Science **223**(1–2), 1–72 (1999)
36. Kullmann, O.: On a generalization of extended resolution. Discrete Applied Mathematics **96–97**, 149–176 (1999)
37. Le Berre, D.: Exploiting the real power of unit propagation lookahead. Electronic Notes in Discrete Mathematics **9**, 59–80 (2001)
38. Li, C.M., Wei, W., Zhang, H.: Combining adaptive noise and look-ahead in local search for SAT. In: J. Marques-Silva, K.A. Sakallah (eds.) Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT 2007), Lecture Notes in Computer Science, pp. 121–133. Springer (2007)
39. Lynce, I., Marques-Silva, J.: The interaction between simplification and search in propositional satisfiability. In: CP'01 Workshop on Modeling and Problem Formulation (2001)
40. Manolios, P., Vroon, D.: Efficient circuit to CNF conversion. In: J. Marques-Silva, K.A. Sakallah (eds.) Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT 2007), *Lecture Notes in Computer Science*, vol. 4501, pp. 4–9. Springer (2007)
41. Mishchenko, A., Chatterjee, S., Brayton, R.K.: DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In: E. Sentovich (ed.) Proceedings of the 43rd Design Automation Conference (DAC 2006), pp. 532–535. ACM (2006)
42. Ostrowski, R., Grégoire, É., Mazure, B., Sais, L.: Recovering and exploiting structural knowledge from CNF formulas. In: P.V. Hentenryck (ed.) Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP 2002), *Lecture Notes in Computer Science*, vol. 2470, pp. 185–199. Springer (2002)
43. Plaisted, D.A., Greenbaum, S.: A structure-preserving clause form translation. Journal of Symbolic Computation **2**(3), 293–304 (1986)
44. Purdom, P.W.: Solving satisfiability with less searching. IEEE Transactions on Pattern Analysis and Machine Intelligence **6**(4), 510–513 (1984)
45. Subbarayan, S., Pradhan, D.K.: NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances. In: H.H. Hoos, D.G. Mitchell (eds.) SAT 2004 Selected Revised Papers, *Lecture Notes in Computer Science*, vol. 3542, pp. 276–291. Springer (2005)
46. Tompkins, D.A., Hoos, H.H.: UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT & MAX-SAT. In: Online Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT 2004) (2004)
47. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: J. Siekmann, G. Wrightson (eds.) Automation of Reasoning 2: Classical Papers on Computational Logic 1967–1970, pp. 466–483. Springer (1983)
48. Van Gelder, A.: Toward leaner binary-clause reasoning in a satisfiability solver. Annals of Mathematics and Artificial Intelligence **43**(1), 239–253 (2005)