Automated Reencoding of Boolean Formulas *

Norbert Manthey[1], Marijn J. H. Heule[2,3], and Armin Biere[3]

[1] Institute of Artificial Intelligence, Technische Universität Dresden, Germany
[2] Department of Computer Science, University of Texas, Austin, United States
[3] Institute for Formal Models and Verification, Johannes Kepler University, Austria

Abstract. We introduce a novel preprocessing technique that automatically reduces the size of a Boolean formula. This technique, called Bounded Variable Addition (BVA), exchanges clauses for variables. Similar to other preprocessing techniques, BVA greedily lowers the sum of variables and clauses, a rough measure for the hardness to solve a formula. We show that cardinality constraints (CCs) can efficiently be reencoded: Given a naive CC encoding, BVA automatically constructs a compact translation, which is smaller than sophisticated encodings for several interesting CCs. Experimental results show that applying BVA on bio-informatics problems, circuit designs and benchmarks from recent satisfiability competitions that also contain other patterns than CCs improves performance.

1 Introduction

SAT solvers are used in many applications in electronic design automation (EDA), including combinational [1,2] and sequential equivalence checking [3,4], bounded [5] and unbounded model checking [6,7,8], but also in synthesis [9] and debugging [10]. State-of-the-art solvers commonly expect their input to be a Boolean formula in conjunctive normal form (CNF), which also serves as data structure for storing the formula internally and maintaining a cache of learned facts in form of clauses [11]. This restriction is on one hand a strength: it allows fast algorithms and compact data structures [12]. On the other hand being forced to use CNF instead of high-level constraints is also a weakness of current SAT solvers: it requires complex synthesis [13] and encoding algorithms [14,15] in order to take full advantage of the raw speed of CNF level solving. There have been several attempts to produce hybrid solvers [16,17], which combine CNF and circuit reasoning. These approaches typically involve a considerable overhead at least from the software engineering perspective. An alternative is to use CNF level preprocessing techniques [18,19,20] to efficiently and effectively simulate certain constraint encoding and reasoning techniques. As example, consider the combination of variable elimination [19] and blocked clause elimination [20], which is able to achieve the same effect as sophisticated encoding algorithms [15].

* The second and the third author are supported by the Austrian Science Foundation (FWF) NFN Grant S11408-N23 (RiSE). The second author is supported by DARPA contract number N66001-10-2-4087.

Starting from a problem to solve, the first step is to encode it into CNF. Next, *preprocessing* techniques are used to simplify the formula, before search is started. Recently, *inprocessing* was introduced [21] that applies preprocessing on a partially solved formula (i.e., during search), linking back in the tool chain. Here, we investigate another link back by *reencoding* clauses. This technique can be applied on the original set of clauses, but also on partially solved and inprocessed formulas. Thus, this paper adds to this discussion of which way to go another argument in favor of CNF level preprocessing. We show that it is possible to simulate sophisticated constraint *encoding* techniques with a rather simple CNF level technique, and thus create the missing link in the picture.

The basic idea is to reencode parts of the CNF by introducing new variables, if the size of the CNF *decreases*. This is in essence a reverse application of variable elimination. Bounded variable elimination (BVE), as proposed in [19,22,23], essentially eliminates a variable in a CNF by clause distribution, if the size of the CNF *does not increase*. In many applications, BVE is currently one of the most effective CNF level preprocessing techniques.

We show improvements in SAT solving time after using our preprocessing on various application benchmarks and recent SAT competitions. We also show, that our technique theoretically and empirically simulates optimized encodings of cardinality constraints starting from a naive standard encoding. These constraints occur frequently in many applications [9,24,25] and have been studied by the CP and SAT communities [26,27,28,29,30,19,31,32]. Furthermore, our preprocessing technique is not restricted to cardinality constraints, but it is also able to factor out common logic in arbitrary formulas without cardinality constraints.

The closest related work is an attempt [33] to speed-up SAT solving by allowing extension steps of extended resolution. The idea is to factor out a common prefix of (learned) clauses by replacing it with a new variable. These extension steps never decrease the number of clauses. If applied to original clauses, BVE would eliminate the extensions again, which renders this technique [33] useless in combination with BVE. In contrast, BVE cannot undo our new method.

We do not claim that high-level reasoning is useless in general. Clearly, there are situations where such reasoning should be combined with CNF level reasoning. This paper adds to the arsenal of preprocessing techniques a new algorithm, which allows to simulate additional sophisticated encoding and reasoning techniques on the CNF level. This is particularly useful for inprocessing, as used in PrecoSAT and Lingeling [21], so new learned facts can be taken into account. As future work, we want to extend these ideas to capture even more high-level techniques such as AIG rewriting [13], compact encoding techniques based on technology mapping [14], and Gaussian elimination of XOR constraints [34].

The remainder of this paper is structured as follows: the next section provides background information on satisfiability solving and variable elimination. In Section 3 we present our novel technique Bounded Variable Addition (BVA). Automated reencoding of cardinality constraints is one of the possible applications of BVA, which is discussed in Section 4. Experimental results are described in Section 5. Finally, we draw conclusions in Section 6.

2 Preliminaries

In this section we review necessary background concepts: conjunctive normal form level Boolean satisfiability (SAT), resolution and variable elimination.

2.1 Conjunctive Normal Form

For a Boolean variable x, there are two *literals*, the positive literal, denoted by x, and the negative literal, denoted by $\bar{x}$. A *clause* is a disjunction of literals and a CNF formula a conjunction of clauses. A clause can be seen as a finite set of literals and a CNF formula as a finite set of clauses. A clause is a *tautology* if it contains both x and $\bar{x}$ for some x. The set of literals occurring in a CNF formula F is denoted by $\mathrm{LIT}(F)$. Formulas are *logically equivalent* if they have the same set of satisfying assignments over the common variables.

2.2 Resolution and Variable Elimination

The resolution rule states that, given two clauses $C_1 = \{x, a_1, \ldots, a_n\}$ and $C_2 = \{\bar{x}, b_2, \ldots, b_m\}$, 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 $S_{\bar{x}}$ of clauses which all contain x and $\bar{x}$, respectively, we define

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

Following the Davis-Putnam procedure [35] (DP), a basic simplification technique, referred to as *variable elimination by clause distribution* in [22,23,36], 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 $S_{\bar{x}}$. Replacing the original clauses in $S_x \cup S_{\bar{x}}$ with the set of *non-tautological* resolvents $S = S_x \otimes S_{\bar{x}}$ gives the formula $(F \setminus (S_x \cup S_{\bar{x}})) \cup S$ that is logically 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.

3 Bounded Variable Addition

Closely following the heuristics applied in the SatElite preprocessor [36] for applying variable elimination, in this paper we study the bounded variant of variable elimination (VE) by clause distribution (BVE) as a simplification technique. In BVE, a variable x can be eliminated only if $|S| \leq |S_x \cup S_{\bar{x}}|$, i.e., when the resulting CNF formula $(F \setminus (S_x \cup S_{\bar{x}})) \cup S$ will not contain more than $|F|$ clauses, where F is the formula before the elimination step.

Example 1. Consider a CNF formula F with

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

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

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

Since $|S_x| + |S_{\bar{x}}| = 6$ and $|S| = 7$, BVE cannot eliminate the variable x. Notice that the clauses $(x \vee \bar{a} \vee \bar{b})$, $(\bar{x} \vee a)$, and $(\bar{x} \vee b)$ in F are equivalent to the Tseitin encoding of the gate $x = \text{AND}(a, b)$. This is why resolving $(x \vee \bar{a} \vee \bar{b})$ with $(\bar{x} \vee a)$ and $(\bar{x} \vee b)$ on x produces only tautological clauses that are not included in S [36].

The global heuristic used for bounding VE – substitute only if the sum of variables and clauses decreases – appears to be a powerful metric to simplify a Boolean formula. This heuristic inspired us to develop the technique *Bounded Variable Addition* (BVA). As the name suggests, BVA is complementary to BVE: instead of exchanging variables for clauses BVA exchanges clauses for variables. Yet the same bounding heuristic is used: substitute to decrease the size of the CNF.

Example 2. The smallest formula for which adding a variable can decrease the size of the CNF consists of six clauses. Such a formula contains the following pattern

$$E = (a \vee c) \wedge (a \vee d) \wedge (a \vee e) \wedge (b \vee c) \wedge (b \vee d) \wedge (b \vee e)$$

By adding a new variable x, E can be reencoded to the logically equivalent formula E' which has one clause less:

$$E' = (a \vee x) \wedge (b \vee x) \wedge (c \vee \bar{x}) \wedge (d \vee \bar{x}) \wedge (e \vee \bar{x})$$

However, it is not always easy to find patterns that reduce the number of clauses. Consider for instance the resulting S consisting of seven clauses in Example 1. Based on the global heuristic, one would like to replace S by $S_x \cup S_{\bar{x}}$ because the size of the latter is smaller. Given S, however, how can we compute that there exists a $S_x \cup S_{\bar{x}}$ such that $S = S_x \otimes S_{\bar{x}}$ and $|S_x| + |S_{\bar{x}}| < |S|$? Even for this small set of clauses, this question is far from trivial. Since practical SAT instances are huge, say $100,000$ clauses, the number of possibilities for S_x and $S_{\bar{x}}$ are enormous. Hence, general BVA until fixpoint will be very costly.

3.1 The *SimpleBoundedVariableAddition* Algorithm

The number of patterns to add a Boolean variable in order to decrease the size of the CNF is very large. To reduce the computational cost, we limited the search to detect only some specific patterns. We focus on those patterns for which the new variable x occurs positively in binary clauses only, while the occurrences of the complement are unrestricted.

Two sets will be used during the detection: a set of literals M_{lit} and a set of clauses M_{cls}. A pair $\langle M_{\text{lit}}, M_{\text{cls}} \rangle$ is called a *replaceable matching* w.r.t. F if for all $l \in M_{\text{lit}}$ and $C \in M_{\text{cls}}$ the clauses $(C \setminus \{M_{\text{lit}}\}) \cup \{l\}$ are in F. Given a replaceable matching $\langle M_{\text{lit}}, M_{\text{cls}} \rangle$, we can apply the *matching-to-clauses* construction method which creates the sets S_x and $S_{\bar{x}}$ as follows: $S_x = \{(l \vee x) \mid l \in M_{\text{lit}}\}$ and $S_{\bar{x}} = \{(C \setminus M_{\text{lit}}) \cup \{\bar{x}\} \mid C \in M_{\text{cls}}\}$. The final step is to remove all clauses $(C \setminus \{M_{\text{lit}}\}) \cup \{l\}$ with $l \in M_{\text{lit}}$ and $C \in M_{\text{cls}}$ and replace them with $S_x \cup S_{\bar{x}}$.

Consider Example 2 again: For the formula E there exist a replaceable matching: $M_{\text{lit}} = \{a, b\}$ and $M_{\text{cls}} = \{(a \vee c), (a \vee d), (a \vee e)\}$. Applying the matching-to-clauses construction method of S_x and $S_{\bar{x}}$ gives $E' = S_x \cup S_{\bar{x}}$.

Theorem 1. *Given a replaceable matching $\langle M_{\text{lit}}, M_{\text{cls}} \rangle$ w.r.t. a CNF formula F, a formula F' can be constructed by adding a Boolean variable such that (1) F' is logically equivalent to F and (2) F' contains $|F| + |M_{\text{lit}}| + |M_{\text{cls}}| - |M_{\text{lit}}| \cdot |M_{\text{cls}}|$ clauses.*

Proof. Given a replaceable matching $\langle M_{\text{lit}}, M_{\text{cls}} \rangle$, we can construct F' as follows: remove from F all clauses $(C \setminus \{M_{\text{lit}}\}) \cup \{l\}$ with $l \in M_{\text{lit}}$ and $C \in M_{\text{cls}}$ and replace them with $S_x \cup S_{\bar{x}}$ which are obtained using the matching-to-clauses construction method. The number of removed clauses is $|M_{\text{lit}}| \cdot |M_{\text{cls}}|$, while the number of added clauses is $|M_{\text{lit}}| + |M_{\text{cls}}|$ showing (2). Applying VE on x in F' produces F. (1) holds because VE preserves logical equivalence.

We refer to the *reduction* of a replaceable matching $\langle M_{\text{lit}}, M_{\text{cls}} \rangle$ with respect to the number of clauses as $|M_{\text{lit}}| \cdot |M_{\text{cls}}| - |M_{\text{lit}}| - |M_{\text{cls}}|$. Notice that for each $l \in \text{LIT}(F)$ holds that $M_{\text{lit}} := \{l\}$ and $M_{\text{cls}} := F_l$ is a replaceable matching. However, it is not useful because the reduction is -1. Heuristically the most interesting replaceable matching is the one with the largest reduction.

We developed the *SimpleBoundedVariableAddition* algorithm, see Fig. 1, to find and replace matchings with a positive reduction. In order to find matchings with large reductions first, a priority queue Q is used that sorts literals $l \in \text{LIT}(F)$ in descending order of the number of occurrences of l in F (line 1). While Q is not empty (line 2), the top element l is used to initialize $M_{\text{lit}} := \{l\}$ and $M_{\text{cls}} := F_l$ (line 3).

In the next seven lines a sequence P of literal-clause pairs $\langle l', C \rangle$ is created such that $C \in M_{\text{cls}}$ and $C \setminus \{l\} \cup \{l'\} \in F$. After initialization (line 4), we loop through the clauses $C \in M_{\text{cls}}$ and select in each of them the literal $l_{\min}$ that occurs least frequently in F to reduce the computational cost (line 5). Now we try to extend P by looping through the clauses $D \in F_{l_{\min}}$ (line 7) and check

```
    SimpleBoundedVariableAddition (CNF formula F)
1   let Q be a priority queue of l ∈ LIT(F) sorted by |F_l|
2   while Q ≠ ∅ do
3     l := Q.top(), Q.pop(), M_lit := {l}, M_cls := F_l
4     P := ∅
5     foreach C ∈ M_cls do
6       let l_min ∈ C \ {l} be least occurring in F
7       foreach D ∈ F_{l_min} do
8         if |C| = |D| and C \ D = l then
9           l' := D \ C
10          P := P ∪ ⟨l', C⟩
11    let l_max be occurring most frequently in P
12    if adding l_max to M_lit further reduces |F| then
13      M_lit := M_lit ∪ {l_max}, M_cls := ∅
14      foreach ⟨l_max, C⟩ ∈ P do
15        M_cls := M_cls ∪ {C}
16      goto 4
17    if |M_lit| = 1 then continue
18    let x be a new variable not occurring in F
19    foreach l' ∈ M_lit do
20      F := F ∪ {l', x}
21      foreach C ∈ M_cls do
22        F := F \ {(C \ {l}) ∪ {l'}}
23    foreach C ∈ M_cls do
24      F := F ∪ {(C \ {l}) ∪ {x̄}}
25    Q.push(l), Q.push(x), Q.push(x̄)
26  return F
```

Fig. 1. Pseudo code of the *SimpleBoundedVariableAddition* algorithm.

whether C and D differ in exactly one literal (line 8). Let the different literal be l' (line 9) and extend P with $\langle l', C\rangle$ (line 10).

Now, we try to add a literal to the matching such that the reduction would increase. The best candidate for this addition is $l_{\max}$ the literal occurring most frequently in P (line 11). If adding $l_{\max}$ increases the reduction (line 12), then $l_{\max}$ is added to M_{lit} (line 13) and M_{cls} is updated s.t. M_{lit} and M_{cls} is a replaceable matching (line 14–15). Afterwards, we try to further increase the matching by rebuilding P (line 16).

The last part of the algorithm implements the replacement, if M_{lit} contains multiple literals (line 17). Variable x is added (line 18) and all clauses $(C \setminus \{M_{\text{lit}}\}) \cup \{l\}$ with $l \in M_{\text{lit}}$ and $C \in M_{\text{cls}}$ are removed from F and replaced by $(l' \vee x)$ with $l' \in M_{\text{lit}}$ and $(C \setminus \{l\}) \cup \{\bar{x}\}$ with $C \in M_{\text{cls}}$ (lines 19–24). Last, but not least, l, x and $\bar{x}$ are inserted in Q for possible future replacements.

3.2 Extensions

Several extensions of the BVA algorithm as shown in Fig. 1 are possible. In this subsection we discuss three of them.

First, we observed that for some problems it occurs that $l = \bar{l}_{\max}$. In this special case, the resolvent between the clauses $C \in F_l$ and $D \in F_{l_{\max}}$ such that $|C| = |D|$ and $C \setminus D = l$ subsume the antecedents. This is also known as self-subsumption [36]. We can simply remove l from the corresponding clause in $C \in F_l$, and remove the clause $D \in F_{l_{\max}}$. So even if $\bar{l}$ occurs only once in P, it can be selected as $l_{\max}$ to reduce the number of clauses without adding a new variable. Since this check is straight forward, it has been added to the algorithm for the experimental evaluation.

The most natural extension is to search for more (less limited) patterns. For instance consider the following formula:

$$H = (a \vee d) \wedge (a \vee e) \wedge (a \vee f) \wedge \\ (b \vee c \vee d) \wedge (b \vee c \vee e) \wedge (b \vee c \vee f)$$

The BVA algorithm as presented in Fig. 1 cannot reduce the number of clauses. However, if one would allow to have pairs of literals (or even more) in M_{lit}, then substitution is possible. Now consider $M_{\text{lit}} = \{\{a\}, \{b, c\}\}$ and $M_{\text{cls}} = \{(a \vee d), (a \vee e), (a \vee f)\}$, applying the replacement code (lines 19–24) results in the following formula:

$$H' = (a \vee x) \wedge (b \vee c \vee x) \wedge (\bar{x} \vee d) \wedge (\bar{x} \vee e) \wedge (\bar{x} \vee f)$$

Enhancing *SimpleBoundedVariableAddition* with these and other patterns will be part of future research.

The third extension is exploring how to reduce the cost to detect patterns. For instance, all literals $l \in Q$ which occur less than three times in F can be removed because the check on line 12 would fail for those literals. Also, all clauses in M_{cls} must have at least one literal occurring in Q. These observations can be used to speed-up detection which would be important for more complex patterns in particular. The first part of this extension is also used in the evaluated implementation, because of its simplicity. We simply do not add variables back into Q if they occur less than three times.

4 Cardinality Constraints

For encoding applications, e.g. routing, scheduling, verification or code-generation [9,24], as well as for encoding instances from product configuration or radio frequency assignment or the domain of a CSP variable [37,38], it is necessary to encode numerical bounds. These numerical bounds can be notated as follows: $\leq k(x_1, \ldots, x_n)$ where n is the number of variables and k is the number of variables that are allowed to be assigned true. A naive encoding into propositional logic of this constraint is

$$\bigwedge_{\substack{M \subseteq \{1,\ldots,n\} \\ |M|=k+1}} (\bigvee_{i \in M} \bar{x}_i).$$

Many encodings for cardinality constraints have been proposed [30,32,19]. In the following two subsections we will show that BVA can be used to reencode cardinality constraints that are encoded naively efficiently. The comparison to sophisticated encodings is based on applying BVA to the naive encoding of cardinality constraints. To the best of our knowledge we name all proposed encodings for this constraint and then focus on the most promising encodings that maintain arc consistency, since reencoding with BVA also preserves arc consistency. Arc consistency means that if k variables are already assigned to true, than all the other variables will be mapped to false by Boolean constraint propagation.

There exist SAT solvers that handle cardinality constraints within the solver, for example Sat4J [39] or clasp [40]. This feature is used for solving MaxSAT and PB problems. However, these solvers do not extract cardinality constraints from the formula and exploit their special mechanisms. In general it is hard to judge whether handling cardinality constraints natively or encoding them to SAT results in the higher performance. Yet the strongest SAT solvers tend to not support native cardinality constraints. MiniSAT [41] for instance supported native cardinality constraints up to version 1.12, but dropped support in all later versions. Recent approaches to incorporate cardinality constraint reasoning into the solver again are in an early stage [42]. For example, this solver cannot compete with a SAT solver that performs preprocessing and inprocessing.

The advantage of encoding cardinality constraints to SAT and applying BVA is that any SAT solver with its full abilities can be applied. Due to recent portfolio systems [43] the most promising solver can be picked, whereas the set of candidate solvers is much smaller for solvers that handle these constraints natively.

4.1 The At-Most-1 Constraint

A special case of cardinality constraints is $k = 1$ that is applied whenever a finite domain is encoded, for example when CSP is translated into SAT. Several encodings have been proposed with lower number of clauses, for example the *log encoding* (LE) [44] or the *2-product encoding* (PE) [45]. Furthermore, for $k = 1$ the *sequential counter encoding* (SE) [30] can be adopted. The naive encoding for $k = 1$ is referred to as the direct encoding (DE). For each encoding the lower bound on the number of clauses and variables for a given value for n are given in Table 1. The values have been taken from the corresponding publications.

Neither of the encodings PE and SE can be processed by BVA. However, applying BVA to the naive encoding yields major benefit with respect to the number of clauses and variables. Although PE has the best asymptotic number of clauses, DE + BVA produce less clauses until the value for n reaches 47. The same effect can be seen for the variables as long as $n < 45$. Note, that for

Table 1. Encoding the *at-most-one* constraint.

Encoding	Clauses	Variables
DE	$\frac{n \cdot (n-1)}{2}$	n
LE [44]	$n \cdot \lceil \log n \rceil$	$n + \log n$
PE [45]	$2n + 4 \cdot \sqrt{n} + O(\sqrt[4]{n})$	$n + \sqrt{n} + O(\sqrt[4]{n})$
SE [30]	$3n - 4$	$2n - 1$
DE + BVA	$3n - 6$	$\sim 2n$
LE + BVA	$\sim 3n$	$\sim 1.5n$

cardinality constraints in real instances the value of n usually is smaller than 45. Applying BVA to LE does not give better results than using PE. The table shows that by using a naive encoding and applying BVA results in a very good encoding for the *at-most-1* constraint.

Example 3. Consider the DE of $\leq 1(a, b, c, d, e, f)$:

$$\begin{aligned} D = &(\bar{a} \vee \bar{b}) \wedge (\bar{a} \vee \bar{c}) \wedge (\bar{a} \vee \bar{d}) \wedge (\bar{a} \vee \bar{e}) \wedge (\bar{a} \vee \bar{f}) \wedge \\ &(\bar{b} \vee \bar{c}) \wedge (\bar{b} \vee \bar{d}) \wedge (\bar{b} \vee \bar{e}) \wedge (\bar{b} \vee \bar{f}) \wedge (\bar{c} \vee \bar{d}) \wedge \\ &(\bar{c} \vee \bar{e}) \wedge (\bar{c} \vee \bar{f}) \wedge (\bar{d} \vee \bar{e}) \wedge (\bar{d} \vee \bar{f}) \wedge (\bar{e} \vee \bar{f}) \end{aligned}$$

Applying BVA on D replaces nine clauses by six using $M_{\text{lit}} = \{\bar{a}, \bar{b}, \bar{c}\}$ and $M_{\text{cls}} = \{(\bar{a} \vee \bar{d}), (\bar{a} \vee \bar{e}), (\bar{a} \vee \bar{f})\}$:

$$\begin{aligned} &(\bar{a} \vee \bar{b}) \wedge (\bar{a} \vee \bar{c}) \wedge (\bar{b} \vee \bar{c}) \wedge (\bar{d} \vee \bar{e}) \wedge (\bar{d} \vee \bar{f}) \wedge (\bar{e} \vee \bar{f}) \wedge \\ &(\bar{a} \vee x) \wedge (\bar{b} \vee x) \wedge (\bar{c} \vee x) \wedge (\bar{d} \vee \bar{x}) \wedge (\bar{e} \vee \bar{x}) \wedge (\bar{f} \vee \bar{x}) \end{aligned}$$

Fig. 2 shows the number of clauses that are needed to encode the at-most-1 constraint with the mentioned encodings. The value on the x-axis gives the number of Boolean variables where a single one has to be set to true. It can be seen clearly that both DE and LE use more clauses than any of the special encodings. Applying BVA to the naive encoding results in almost the same number of clauses as if a special encoding is used. Until the number of elements reaches 47, using DE + BVA results in the smallest number of clauses for the at-most-1 constraint.

4.2 The At-Most-K Constraint

The more generic case of the cardinality constraint does not bind k to a specific value. Thus, it is not possible to easily adopt a special encoding as for the case $k = 1$. In general, for the mentioned applications a value of k that is larger than 1 is required. Still, special encodings have been proposed to encode general cardinality constraints efficiently. Again, we consider only encodings that preserve arc consistency.

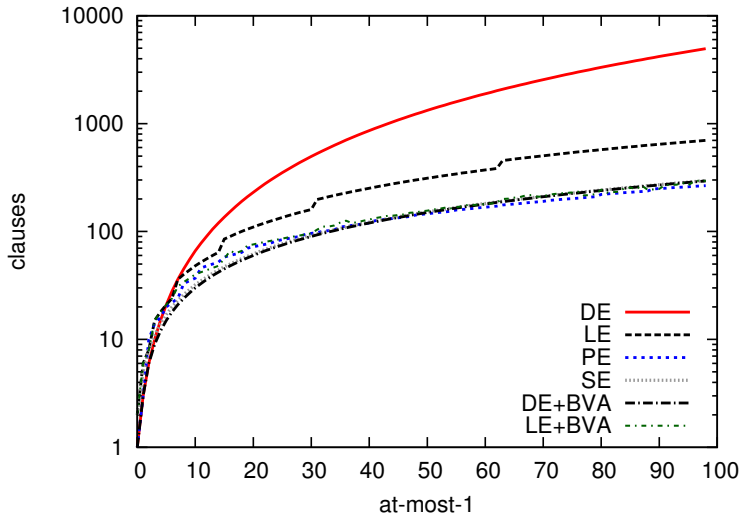

Fig. 2. Clauses needed to encode the *at-most-1* constraint.

Encoding cardinality constraints based on a unary number representation and a binary tree with comparators has been proposed [31] which we refer to as TREE. Sinz introduced a sequential counter encoding and a parallel counter encoding, where the latter one does not preserve arc consistency. Eén and Sörrensen [19] introduced three possibilities to encode a cardinality constraint, namely by using (i) binary decision diagrams, (ii) networks of sorters or (iii) networks of adders [19], where only the first two encoding preserve arc consistency. Encoding the BDD into CNF has been done by the Tseitin transformation. Another encoding, which cannot provide arc consistency but is small in size is the *parallel counter* [30]. There are two small sized encodings for cardinality constraints that do not have this property: the *parallel counter* [30] and the *hybrid perfect hashing function* based encoding [46]. Although the properties of the latter are very nice, it cannot guarantee arc consistency for all possible cardinality constraints. The arc consistent variant of *perfect hashing function* based encoding [46] uses slightly more clauses than the sequential counter, but needs less auxiliary variables. Since we focus on the number of clauses, we do not consider this encoding. Table 2 shows the asymptotic number of clauses and variables that are needed by using the different encodings. Notice that the number of clauses that are required by the naive encoding is significantly higher than for the other encodings.

Discussing the effect of BVA on the naive encoding of *at-most-k* constraints is not as simple as for the special case $k = 1$, because non-binary clauses are involved in these encodings.

Due to the limit of BVA to detect only matchings where M_{lit} is restricted to a set of single literals, many potential matchings cannot be recognized and replaced. The following example illustrates this statement. Although matchings

with a reduction of 3 are part of formula K, only reductions of size 2 can be recognized. By increasing the number of matching literals in BVA, this limit can be overcome. Still, applying BVA to the naive at-most-k encoding reduces the number of clauses significantly. The smaller the value k, the closer the number of clauses after BVA gets to the number of clauses of the special encodings. We present some exemplary values for the number of clauses and variables after applying BVA to the naive encoding in Table 3 to support this statement. The formulas for SE and BDD have been generated by using the tools that have been provided with the corresponding publications. For $n = 10$ using BVA results in the smallest formula. For $n = 20$ the special encodings are almost always more effective than BVA.

Example 4. Consider the encoding of $\leq 3(a, b, c, d, e, f)$:

$$\begin{aligned} K = &(\bar{a} \vee \bar{b} \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee \bar{b} \vee \bar{c} \vee \bar{e}) \wedge (\bar{a} \vee \bar{b} \vee \bar{c} \vee \bar{f}) \wedge \\ &(\bar{a} \vee \bar{b} \vee \bar{d} \vee \bar{e}) \wedge (\bar{a} \vee \bar{b} \vee \bar{d} \vee \bar{f}) \wedge (\bar{a} \vee \bar{b} \vee \bar{e} \vee \bar{f}) \wedge \\ &(\bar{a} \vee \bar{c} \vee \bar{d} \vee \bar{e}) \wedge (\bar{a} \vee \bar{c} \vee \bar{d} \vee \bar{f}) \wedge (\bar{a} \vee \bar{c} \vee \bar{e} \vee \bar{f}) \wedge \\ &(\bar{a} \vee \bar{d} \vee \bar{e} \vee \bar{f}) \wedge (\bar{b} \vee \bar{c} \vee \bar{d} \vee \bar{e}) \wedge (\bar{b} \vee \bar{c} \vee \bar{d} \vee \bar{f}) \wedge \\ &(\bar{b} \vee \bar{c} \vee \bar{e} \vee \bar{f}) \wedge (\bar{b} \vee \bar{d} \vee \bar{e} \vee \bar{f}) \wedge (\bar{c} \vee \bar{d} \vee \bar{e} \vee \bar{f}) \end{aligned}$$

Applying BVA on the formula K will find the matching $M_{\text{lit}} = \{\bar{a}, \bar{b}\}$ and $M_{\text{cls}} = \{(\bar{a} \vee \bar{d} \vee \bar{e} \vee \bar{f}), (\bar{a} \vee \bar{c} \vee \bar{d} \vee \bar{e}), (\bar{a} \vee \bar{c} \vee \bar{e} \vee \bar{f}), (\bar{a} \vee \bar{c} \vee \bar{d} \vee \bar{f})\}$ with a reduction of 2 clauses. Yet the more interesting case is to use $M_{\text{lit}} = \{\{\bar{a}, \bar{b}\}, \{\bar{a}, \bar{c}\}, \{\bar{b}, \bar{c}\}\}$ and $M_{\text{cls}} = \{(\bar{a} \vee \bar{b} \vee \bar{d} \vee \bar{e}), (\bar{a} \vee \bar{b} \vee \bar{d} \vee \bar{f}), (\bar{a} \vee \bar{b} \vee \bar{e} \vee \bar{f})\}$ which has reduction 3.

5 Experiments

We implemented the algorithm of Fig. 1 in a new tool[4]. Although applying *SimpleBoundedVariableAddition* until fixpoint requires less than a second on most benchmarks, we observed that BVA was sometimes very expensive – even in case no replaceable matching can be found. Therefore, we limited the execution of BVA as follows: when the check on line 8 of Fig. 1 is executed 10,000,000 times, the algorithm is aborted. Then, the formula is returned with all substitutions until that point. This limit ensures that the preprocessing runtime is only a few

[4] The sources of the tool will be made available after acceptance.

Table 2. Encoding the *at-most-k* constraint.

Encoding	Clauses	Variables
naive	$\binom{n}{k+1}$	n
TREE [31]	$O(n^2)$	$\Theta(n \log n + 1)$
SE [30]	$2nk + n - 3k - 1$	$(n-1) \cdot k$
BDD [19]	$2nk + n - k^2$	$(n-k+1) \cdot k + n$

Table 3. Encoding the *at-most-k* constraint.

		naive		naive + BVA		SE [30]		BDD [19]	
k	n	#var	#cls	#var	#cls	#var	#cls	#var	#cls
2	10	10	120	18	32	28	43	33	59
3	10	10	210	18	47	37	60	37	70
4	10	10	252	19	51	46	77	39	75
5	10	10	210	17	53	55	94	39	74
2	20	20	1140	40	80	58	93	73	139
3	20	20	4845	44	209	77	130	87	180
4	20	20	15504	66	326	96	167	99	215
5	20	20	38760	60	768	115	204	109	244
6	20	20	77520	130	1104	134	241	117	267
7	20	20	125970	113	2051	153	278	123	284
8	20	20	167960	227	2247	172	315	127	295
9	20	20	184756	104	3175	191	352	129	300
10	20	20	167960	191	2892	210	389	129	299

seconds for the more costly formulas. Note, that all the experiments use the first and third extension that have been mentioned in Section 3.2.

For the experiments we selected the SAT solver Lingeling (version SAT11 Competition[5]) because of the strong performance of this solver during SAT10 Race and SAT11 Competition.

5.1 Bio-informatics

size on the instances originates from bio-informatics. These formulas encode computing evolutionary tree measures into SAT [47]. The results of these instances are shown in Table 4. The selected benchmarks are very hard and no solver was able to tackle any of the _09 or _10 instances (within the CPU timeout of 40,000 seconds). After applying our BVA tool –which on average reduces the size of a factor ten– Lingeling could solve all instances. Of the original instances only rpoc_08 could be solved, yet 36 times slower.

5.2 FPGA routing

As discussed in prior sections, several benchmarks arising from EDA consist of cardinality constraints. A family of this type used in recent SAT competitions encodes FPGA routing problems [9]. This family consists of six routing configurations (chnlXX_YY) in which one tries to route (a) 11, 12 or 13 connections through 10 tracks, and (b) 12, 13 or 20 connections through 11 track. Table 5 shows the results. BVA decreases the size of the CNF by more than a factor two. The preprocessed formulas are easier to solve. FPGA routing can also be solved

[5] `http://www.satcompetition.org`

Table 4. Results on bio-informatics benchmarks. TO is 20,000 seconds.

	original			BVA preprocessed			
instance	#var	#cls	solve	#var	#cls	pre	solve
ndhf_09	1910	167476	TO	3098	14588	1.47	**187**
ndhf_10	2112	191333	TO	3418	16756	1.70	**1272**
rbcl_08	1278	67720	TO	1981	8669	0.29	**16**
rbcl_09	1430	79118	TO	2192	10157	0.39	**101**
rbcl_10	1584	91311	TO	2443	11811	0.43	**604**
rpoc_08	1278	74454	8628	2011	8494	0.39	**237**
rpoc_09	1430	86709	TO	2252	10063	0.47	**3590**
rpoc_10	1584	99781	TO	2474	11667	0.66	**11945**

with special purpose solvers that perform well on these instances. Techniques that are used in these solvers are for example symmetry breaking [9]. Since symmetry breaking and BVA are orthogonal, it is a reasonable choice to measure the effect of BVA also on this instance family. Furthermore, it would be possible to combine symmetry breaking and BVA.

Table 5. Results on FPGA routing problems. TO is 20,000 seconds.

	original			BVA preprocessed			
instance	#var	#cls	solve	#var	#cls	pre	solve
chnl10_11	220	1122	9372	302	562	0.00	**69.3**
chnl10_12	240	1344	7279	340	624	0.00	**15.0**
chnl10_13	260	1586	2682	380	686	0.00	**26.0**
chnl11_12	264	1476	TO	374	684	0.00	**41.6**
chnl11_13	286	1742	TO	418	752	0.00	**17.1**
chnl11_20	440	4220	TO	667	1228	0.00	**12.1**

5.3 Recent SAT Competitions

We observed that applying variable elimination (BVE) creates many patterns for variable addition (BVA). Therefore we preprocessed, using SatElite of MiniSAT 2.2 [36], the formulas of recent SAT competitions with BVE –which is default in the strongest SAT solvers– and applied our *SimpleBoundedVariableAddition* algorithm afterwards.

On the application benchmarks of SAT09, Lingeling solved 196 instances (75 SAT and 121 UNSAT) within 900 seconds (including all preprocessing time), while without BVA 190 instances (74 SAT, 116 UNSAT) were solved. The same experiment on the application benchmarks of SAT11, resulted in a similar pic-

ture: with BVA 169 (79 SAT, 90 UNSAT) were solved, while without BVA, Lingeling solves 162 instances (80 SAT, 82 UNSAT).

On the crafted instances we noticed that BVA works particularly well on benchmarks from the Satisfiable Random High Degree Subgraph Isomorphism (SRHD) family [48]. Using BVA, Lingeling is able to solve several more instances of this family. However, even with the improved performance Lingeling requires minutes to solve these benchmarks, while local search SAT algorithms can find a solution in seconds.

6 Conclusions

We presented the preprocessing technique BVA that reduces the size of CNF formulas by introducing new variables. BVA can shrink formulas containing for instance cardinality constraints. Experiments show that the smaller CNFs are generally solved faster, making BVA a useful tool. Also interestingly, the presented techniques is orthogonal to BVE, which is one of the most powerful preprocessing techniques.

Future work in this direction will focus on enhancing BVA with more replacement patterns. Additionally, BVA will be studied in the context of inprocessing to observe the interaction with other techniques such as BVE and BCE.

References

1. Goldberg, E.I., Prasad, M.R., Brayton, R.K.: Using SAT for combinational equivalence checking. In: DATE. (2001) 114–121
2. Mishchenko, A., Chatterjee, S., Brayton, R.K., Eén, N.: Improvements to combinational equivalence checking. In Hassoun, S., ed.: ICCAD, ACM (2006) 836–843
3. Baumgartner, J., Mony, H., Paruthi, V., Kanzelman, R., Janssen, G.: Scalable sequential equivalence checking across arbitrary design transformations. In: ICCD, IEEE (2006)
4. Kaiss, D., Skaba, M., Hanna, Z., Khasidashvili, Z.: Industrial strength SAT-based alignability algorithm for hardware equivalence verification. In: FMCAD, IEEE Computer Society (2007) 20–26
5. Biere, A., Cimatti, A., Clarke, E.M., Fujita, M., Zhu, Y.: Symbolic model checking using SAT procedures instead of bdds. In: DAC. (1999) 317–320
6. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a SAT-solver. In Jr., W.A.H., Johnson, S.D., eds.: FMCAD. Volume 1954 of LNCS., Springer (2000) 108–125
7. McMillan, K.L.: Interpolation and SAT-based model checking. In: CAV. (2003) 1–13
8. Bradley, A.R.: SAT-based model checking without unrolling. In: VMCAI. (2011) 70–87
9. Aloul, F.A., Ramani, A., Markov, I.L., Sakallah, K.A.: Solving difficult SAT instances in the presence of symmetry. In: DAC '02, New York, NY, USA, ACM (2002) 731–736

10. Chen, Y., Safarpour, S., Marques-Silva, J.P., Veneris, A.G.: Automated design debugging with maximum satisfiability. IEEE Trans. on CAD of Integrated Circuits and Systems **29**(11) (2010) 1804–1817
11. Marques Silva, J.P., Sakallah, K.A.: Grasp: A search algorithm for propositional satisfiability. IEEE Trans. Computers **48**(5) (1999) 506–521
12. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: DAC, ACM (2001) 530–535
13. Mishchenko, A., Chatterjee, S., Brayton, R.K.: Dag-aware aig rewriting a fresh look at combinational logic synthesis. In: DAC. (2006) 532–535
14. Eén, N., Mishchenko, A., Sörensson, N.: Applying logic synthesis for speeding up SAT. In: SAT'07. Volume 4501 of LNCS., Springer (2007) 272–286
15. Chambers, B., Manolios, P., Vroon, D.: Faster SAT solving with better CNF generation. In: DATE, IEEE (2009) 1590–1595
16. Guerra e Silva, L., Miguel Silveira, L., Marques Silva, J.P.: Algorithms for solving boolean satisfiability in combinational circuits. In: DATE, IEEE Computer Society (1999) 526–530
17. Ganai, M.K., Ashar, P., Gupta, A., Zhang, L., Malik, S.: Combining strengths of circuit-based and CNF-based algorithms for a high-performance SAT solver. In: DAC, ACM (2002) 747–750
18. Bacchus, F., Winter, J.: Effective preprocessing with hyper-resolution and equality reduction. In: SAT'03. Volume 2919 of LNCS., Springer (2004) 341–355
19. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into SAT. JSAT **2**(1-4) (2006) 1–26
20. Järvisalo, M., Biere, A., Heule, M.J.H.: Blocked clause elimination. In: TACAS'10. Volume 6015 of LNCS., Springer (2010) 129–144
21. Biere, A.: Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. FMV Report Series Technical Report 10/1, Johannes Kepler University, Linz, Austria (2010)
22. Biere, A.: Resolve and expand. In Hoos, H.H., Mitchell, D.G., eds.: SAT (Selected Papers. Volume 3542 of LNCS., Springer (2004) 59–70
23. Subbarayan, S., Pradhan, D.K.: Niver: Non increasing variable elimination resolution for preprocessing SAT instances. In: SAT. (2004)
24. Chai, D., Kuehlmann, A.: A fast pseudo-boolean constraint solver. IEEE Trans. on CAD of Integrated Circuits and Systems **24**(3) (2005) 305–317
25. Marques-Silva, J.P., Planes, J.: Algorithms for maximum satisfiability using unsatisfiable cores. In: DATE, IEEE (2008) 408–413
26. Quimper, C.G., López-Ortiz, A., van Beek, P., Golynski, A.: Improved algorithms for the global cardinality constraint. In Wallace, M., ed.: CP. Volume 3258 of LNCS., Springer (2004) 542–556
27. Quimper, C.G., Walsh, T.: Beyond finite domains: The all different and global cardinality constraints. In van Beek, P., ed.: CP. Volume 3709 of LNCS., Springer (2005) 812–816
28. Zanarini, A., Pesant, G.: Generalizations of the global cardinality constraint for hierarchical resources. In: Proceedings of the 4th international conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. CPAIOR '07, Berlin, Heidelberg, Springer-Verlag (2007) 361–375
29. Régin, J.C.: Combination of among and cardinality constraints. In: Proceedings of the Second international conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. CPAIOR'05, Berlin, Heidelberg, Springer-Verlag (2005) 288–303

30. Sinz, C.: Towards an optimal CNF encoding of boolean cardinality constraints. In: CP. (2005) 827–831
31. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of boolean cardinality constraints. In Rossi, F., ed.: CP. Volume 2833 of LNCS., Springer (2003) 108–122
32. Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality networks and their applications. In Kullmann, O., ed.: SAT. Volume 5584 of LNCS., Springer (2009) 167–180
33. Audemard, G., Katsirelos, G., Simon, L.: A restriction of extended resolution for clause learning SAT solvers. In Fox, M., Poole, D., eds.: AAAI, AAAI Press (2010)
34. Warners, J.P., van Maaren, H.: A two-phase algorithm for solving a class of hard satisfiability problems. Operations Research Letters **23**(35) (1998) 81 – 88
35. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM **7**(3) (1960) 201–215
36. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: SAT'05. Volume 3569 of LNCS., Springer (2005) 61–75
37. Küchlin, W., Sinz, C.: Proving consistency assertions for automotive product data management. J. Autom. Reasoning **24**(1/2) (2000) 145–163
38. Cabon, B., de Givry, S., Lobjois, L., Schiex, T., Warners, J.P.: Radio link frequency assignment. Constraints **4**(1) (1999) 79–89
39. Le Berre, D., Parrain, A.: The sat4j library, release 2.2, system description. Journal on Satisfiability, Boolean Modeling and Computation(JSAT) **7** (2010) 59–64
40. Gebser, M., Kaufmann, B., Schaub, T.: The conflict-driven answer set solver clasp: Progress report. In: Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning. LPNMR '09, Berlin, Heidelberg, Springer-Verlag (2009) 509–514
41. Eén, N., Sörensson, N.: An extensible sat-solver. In Giunchiglia, E., Tacchella, A., eds.: SAT. Volume 2919 of LNCS., Springer (2003) 502–518
42. Liffiton, M.H., Maglalang, J.C.: A cardinality solver: More expressive constraints for free - (poster presentation). In: SAT. Volume 7317 of LNCS., Springer (2012) 485–486
43. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: portfolio-based algorithm selection for sat. J. Artif. Int. Res. **32**(1) (2008) 565–606
44. Prestwich, S.D.: Variable Dependency in Local Search: Prevention Is Better Than Cure. In Marques-Silva, J.P., Sakallah, K.A., eds.: SAT. Volume 4501 of LNCS., Springer (2007) 107–120
45. Chen, J.: A New SAT Encoding of the At-Most-One Constraint. In: Proceedings of ModRef 2011. (2011)
46. Ben-Haim, Y., Ivrii, A., Margalit, O., Matsliah, A.: Perfect hashing and cnf encodings of cardinality constraints. In: SAT. (2012) 397–409
47. Bonet, M.L., John, K.S.: Efficiently calculating evolutionary tree measures using SAT. In: SAT '09, Berlin, Heidelberg, Springer-Verlag (2009) 4–17
48. Anton, C.a.: An improved satisfiable sat generator based on random subgraph isomorphism. In: Proceedings of the 24th Canadian conference on Advances in artificial intelligence. Canadian AI'11, Berlin, Heidelberg, Springer-Verlag (2011) 44–49