Automated Reencoding Meets Graph Theory

Benjamin Przybocki
Carnegie Mellon University, Pittsburgh, PA, USA

Bernardo Subercaseaux
Carnegie Mellon University, Pittsburgh, PA, USA

Marijn J. H. Heule
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

Bounded Variable Addition (BVA) is a central preprocessing method in modern state-of-the-art SAT solvers. We provide a graph-theoretic characterization of which 2-CNF encodings can be constructed by an idealized BVA algorithm. Based on this insight, we prove new results about the behavior and limitations of BVA and its interaction with other preprocessing techniques. We show that idealized BVA, plus some minor additional preprocessing (e.g., equivalent literal substitution), can reencode any 2-CNF formula with n variables into an equivalent 2-CNF formula with $(\frac{\lg(3)}{4} + o(1))\frac{n^2}{\lg n}$ clauses. Furthermore, we show that without the additional preprocessing the constant factor worsens from $\frac{\lg(3)}{4} \approx 0.396$ to 1, and that no reencoding method can achieve a constant below 0.25. On the other hand, for the at-most-one constraint on n variables, we prove that idealized BVA cannot reencode this constraint using fewer than $3n - 6$ clauses, a bound that we prove is achieved by actual implementations. In particular, this shows that the product encoding for at-most-one, which uses $2n + o(n)$ clauses, cannot be constructed by BVA regardless of the heuristics used. Finally, our graph-theoretic characterization of BVA allows us to leverage recent work in algorithmic graph theory to develop a drastically more efficient implementation of BVA that achieves a comparable clause reduction on random monotone 2-CNF formulas.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Theory of computation → Constraint and logic programming; Mathematics of computing → Graph theory

Keywords and phrases SAT solving, CNF encodings, BVA, Rectifier networks

Digital Object Identifier 10.4230/LIPIcs.SAT.2026.29

Related Version *Full Version*: `https://arxiv.org/abs/2603.27774` [22]

Funding All authors were supported by NSF grant DMS-2434625.
Benjamin Przybocki: Przybocki was supported by the NSF Graduate Research Fellowship Program under Grant No. DGE-2140739.

1 Introduction

Providing a satisfactory explanation for the remarkable performance of SAT solvers in practice is a major theoretical challenge. Part of the difficulty of explaining this phenomenon, further detailed in the appropriately titled article "On the Unreasonable Effectiveness of SAT Solvers" by Vardi and Ganesh [9], is that modern SAT solvers combine a variety of heuristics, preprocessing and inprocessing techniques, data structures, and so on. We posit that, to make progress on the overall question, we ought to better understand the power and limitations of each of these ingredients. In this paper, we home in on one of these components, namely, *Bounded Variable Addition* (BVA) [20], a preprocessing technique that introduces auxiliary variables to reencode an input formula into an equisatisfiable formula with fewer clauses. In 2023, a successor of BVA named *Structured BVA* (SBVA) [10] led the solver SBVA-CaDiCaL, whose primary innovation was to incorporate SBVA as preprocessing to the state-of-the-art solver CaDiCaL, to win the 1st place in the 2023 SAT Competition [4].

At the 2024 SAT Competition, many other solvers followed suit by including some variant of BVA as a preprocessing step, including the competition winner Kissat-sc2024 [11]. BVA is now incorporated into the stable versions of CaDiCaL and Kissat under the name factor [6].

However, despite its empirically demonstrated utility, little is known about BVA and its variants from a theoretical point of view. For example, Biere, one of the authors of the original BVA paper, commented [5]:

> *In [20], we already realized that BVA reencodes [the direct encoding of the at-most-one constraint] into something better. Results were only empirical though and we did not analyze whether in principle BVA can turn the direct encoding into for instance the product or sequential counter encoding.*

In this paper, we develop a graph-theoretic framework for analyzing BVA, by means of which we can answer these and similar questions about the power and limitations of BVA. Furthermore, our framework allows us to draw on recent work in algorithmic graph theory to develop a drastically more efficient implementation of BVA.

Our Results

We focus on the behavior of BVA on 2-CNF subformulas. Many formulas of interest, including, e.g., most SAT competition formulas, contain a large 2-CNF subformula that can be reencoded more compactly, leading to faster runtimes. In Section 6, we justify in more detail why the 2-CNF fragment not only captures the most important aspects of the theoretical analysis but also explains most of BVA's empirical success.

Our primary result is a characterization of which reencodings of a 2-CNF formula are achievable by BVA. Our characterization is in terms of *rectifier networks* [8, 14, 19], a graph-theoretic notion that has been studied in the context of circuit complexity. Specifically, we show in Theorem 18 that there is a correspondence between reencodings achievable by BVA and what we call *strict polarized rectifier networks*. Our characterization holds for an *idealized* version of BVA that abstracts away implementation-specific heuristics.

With this characterization in hand, we first study the class of 2-CNF formulas in general. By generalizing an old result of Nechiporuk [21] about rectifier networks, we prove the following:[1]

▶ **Theorem 1** (Informal). *Idealized BVA can reencode every 2-CNF formula with n variables to one with at most $(1 + o(1))\frac{n^2}{\lg n}$ clauses.*

Note that, without reencoding, 2-CNF formulas can have $\Theta(n^2)$ clauses. We also show that the bound from Theorem 1 is sharp, in the sense that there is some 2-CNF formula that idealized BVA reencodes with at least $(1 - o(1))\frac{n^2}{\lg n}$ clauses. On the other hand, we describe a simplification routine that one can run before BVA ("pre-preprocessing"), which consists of equivalent literal substitution and a weaker form of failed literal elimination [12], and this simplification step improves the worst-case performance of idealized BVA:

▶ **Theorem 2** (Informal). *Idealized BVA with simplification can reencode every 2-CNF formula with n variables to one with at most $(\frac{\lg(3)}{4} + o(1))\frac{n^2}{\lg n}$ clauses.*

This bound is also sharp. Furthermore, we show that for every reencoding method whose output is 2-CNF, there is some 2-CNF formula that it reencodes with at least $(\frac{1}{4} - o(1))\frac{n^2}{\lg n}$ clauses. Thus, with respect to worst-case analysis, idealized BVA is optimal up to a constant factor.

[1] All logarithms in this paper are base 2 unless otherwise specified.

An important subset of 2-CNF formulas is the monotone (or antitone) 2-CNF formulas. For example, a direct encoding for finding an independent set in a graph G would include, for every edge $\{u, v\} \in E(G)$, a clause $(\overline{x_u} \vee \overline{x_v})$. We can actually improve the bound from Theorem 1 for monotone 2-CNF formulas:

▶ **Theorem 3** (Informal). *Idealized BVA can reencode every monotone 2-CNF formula with n variables to one with at most $(\frac{1}{4} + o(1))\frac{n^2}{\lg n}$ clauses.*

This bound is sharp, not merely for idealized BVA but for arbitrary reencoding methods whose output is 2-CNF.

Next, we study BVA's performance on a particular class of 2-CNF formulas, namely the direct encoding of the at-most-one constraint on n variables:

$$\mathsf{AtMostOne}(x_1, \ldots, x_n) \coloneqq \bigwedge_{1 \le i < j \le n} (\overline{x_i} \vee \overline{x_j}).$$

Manthey, Heule, and Biere [20] observed experimentally that their implementation of BVA reencodes $\mathsf{AtMostOne}(x_1, \ldots, x_n)$ into $3n - 6$ clauses. We prove in the full version of this paper that, given their heuristics, BVA always achieves this bound [22, Theorem 11]. On the other hand, via a combinatorial analysis of strict rectifier networks, we show a matching lower bound, regardless of the heuristics used:

▶ **Theorem 4** (Informal). *Idealized BVA cannot encode $\mathsf{AtMostOne}(x_1, \ldots, x_n)$ using fewer than $3n - 6$ clauses.*

In particular, this implies that the product encoding [7] for $\mathsf{AtMostOne}(x_1, \ldots, x_n)$, which uses $2n + o(n)$ clauses, cannot be constructed by idealized BVA.

Due to space constraints, most proofs are omitted and can be found in the full version of this paper [22].

Implementation. Leveraging recent work by Krapivin et al. [18] on efficient algorithms for constructing biclique partitions of graphs, which are a special case of rectifier networks, we present an implementation of BVA for 2-CNF formulas that runs in time $O(n^2)$. While the time complexity of the algorithm used in factor, the previous state of the art, has not been rigorously analyzed, it is at least $\Omega(n^3)$. This algorithmic improvement leads to an order-of-magnitude speedup over factor on large random monotone 2-CNF formulas.

2 Preliminaries

2.1 Notation and Terminology

We will use $\top$ and $\bot$ to denote *true* and *false*, respectively. The negation of a variable x is denoted $\overline{x}$, and a *literal* is either a variable or the negation of a variable. Literals x and $\overline{x}$ are said to be complementary, for any variable x. A *clause* is a set of non-complementary literals, and a *formula* is a set of clauses. The *size* of a formula is its number of clauses. We say a formula is *2-CNF* if every clause has size exactly 2. We denote the set of variables appearing in a formula F as $\mathsf{Var}(F)$, and we define $\mathsf{Var}(C)$ analogously for a clause C. Given a set $\mathcal{V}$ of variables, an *assignment* is a function $\tau\colon \mathcal{V} \to \{\bot, \top\}$. For an assignment $\tau : \mathcal{V} \to \{\bot, \top\}$ and a formula F, now with $\mathcal{V} \subseteq \mathsf{Var}(F)$, we denote by $F|_\tau$ the formula obtained by eliminating from F each clause satisfied by τ, and then from each remaining clause eliminating every literal ℓ such that $\tau \models \overline{\ell}$. Note that $\mathsf{Var}(F|_\tau) = \mathsf{Var}(F) \setminus \mathcal{V}$. We will write $\mathsf{SAT}(F)$ to say that $\tau \models F$ for some assignment τ.

2.2 CNF Encodings and BVA

Boolean satisfiability problems, in general, are based on determining whether a boolean function $f\colon \{\bot,\top\}^n \to \{\bot,\top\}$ outputs $\top$ for some input. SAT solvers, however, take CNF formulas as input, and thus require an *encoding* (formally defined next) of f as a CNF formula.

▶ **Definition 5.** *Given a boolean function $f\colon \{\bot,\top\}^n \to \{\bot,\top\}$, and a sequence of propositional variables $X := (x_1,\ldots,x_n)$, we say that a CNF formula φ over variables $X \sqcup Y$* encodes *f if, for every assignment τ of X,*

$$f(\tau(x_1),\ldots,\tau(x_n)) = \top \iff \mathit{SAT}(\varphi|_\tau).$$

The variables in Y are called auxiliary *variables, whereas those in X are called* base *variables.*

For example, the cardinality constraint *"exactly k variables from $x_1,\ldots,x_n$ must be true"* is directly well defined as a boolean function, but there is significant freedom regarding how to encode it in CNF. An important component of this freedom lies in the auxiliary variables, since neither their number nor purpose is predetermined.

Designing effective CNF encodings, and in particular choosing the right auxiliary variables to introduce, has been key to successfully tackling hard instances [13, 17, 23, 25]. However, the design of effective encodings can require significant technical expertise and domain-specific knowledge. This makes *automated reencoding* techniques, such as BVA, particularly appealing. Such techniques work by optimizing some quantitative proxy for the effectiveness of an encoding; a common choice of proxy is to consider some function of the number of clauses and the number of variables (e.g., their sum [20] or weighted sum [1]), although more complicated metrics have also been proposed [10]. Automated reencoding techniques operate by identifying patterned subformulas of a given CNF formula that can be rewritten using fewer clauses by introducing auxiliary variables. BVA, in particular, works exclusively by detecting and rewriting a single concrete pattern, which is illustrated by the following example.

▶ **Example 6.** Consider a formula F that includes the 9 clauses of the form $(x_i \to x_j)$ for $i \in \{1,2,3\}$ and $j \in \{4,5,6\}$. These 9 clauses can be expressed more compactly by introducing an auxiliary variable y and then enforcing:

$$(x_1 \to y) \wedge (x_2 \to y) \wedge (x_3 \to y) \wedge (y \to x_4) \wedge (y \to x_5) \wedge (y \to x_6).$$

This reduces the number of clauses from 9 to 6, at the cost of adding 1 auxiliary variable. Figure 1 illustrates that, under the graph-theoretic perspective of this paper, the reencoded pattern is a *biclique* (complete bipartite subgraph).

(a) Original encoding. **(b)** Reduced encoding.

Figure 1 Illustration of Example 6.

More generally, let $\mathcal{C}$ and $\mathcal{D}$ be sets of clauses. Then, using notation $\mathcal{C} \bowtie \mathcal{D}$ to denote the set of clauses of the form $(C_i \vee D_j)$ for $C_i \in \mathcal{C}$ and $D_j \in \mathcal{D}$, we define a *BVA step* as follows:

▶ **Definition 7** (BVA Step). *Let φ be a formula, and let $\mathcal{C}$ and $\mathcal{D}$ be sets of nonempty clauses such that $\mathcal{C} \bowtie \mathcal{D} \subseteq \varphi$, and $y \notin \mathsf{Var}(\varphi)$ be a fresh variable. Then, we say that the formula*

$$\varphi' := (\varphi \setminus (\mathcal{C} \bowtie \mathcal{D})) \cup \{(C_i \vee y) \mid C_i \in \mathcal{C}\} \cup \{(\overline{y} \vee D_j) \mid D_j \in \mathcal{D}\}$$

can be derived from φ in one BVA step. *We summarize this by writing $\varphi \overset{\text{BVA}}{\rightsquigarrow\!\triangleright} \varphi'$.*

Note that if $\varphi \overset{\text{BVA}}{\rightsquigarrow\!\triangleright} \varphi'$, then fully resolving on the introduced fresh variable y on φ' recovers φ; in other words, BVA is the opposite of variable elimination in the Davis–Putnam sense. BVA operates by the repeated application of BVA steps, so we define *idealized BVA* to be the reflexive transitive closure of $\overset{\text{BVA}}{\rightsquigarrow\!\triangleright}$, denoted $\overset{\text{BVA}}{\rightsquigarrow\!\triangleright}{}^*$.

▶ **Lemma 8** ([20, Thm. 1]). *Let f be a boolean function and φ a CNF encoding of f over the base variables X. If $\varphi \overset{\text{BVA}}{\rightsquigarrow\!\triangleright}{}^* \varphi'$, then φ' is a CNF encoding of f over the base variables X.*

Detecting patterns of the form $\mathcal{C} \bowtie \mathcal{D}$ for arbitrary sets $\mathcal{C}$ and $\mathcal{D}$ can be computationally expensive, and thus actual implementations of BVA restrict themselves to identifying subformulas $\mathcal{C} \bowtie \mathcal{D}$ in which $\mathcal{C}$ is a set of unit clauses [10, 20]. Note that each clause in $(C_i \vee D_j) \in \mathcal{C} \bowtie \mathcal{D}$ has size at most $|C_i| + |D_j|$, and thus in the case we consider in this paper, where φ is a 2-CNF formula, we may assume that *both* $\mathcal{C}$ and $\mathcal{D}$ are sets of unit clauses. Thus, for ease of notation we will write, e.g., $\mathcal{C} = \{x_1, x_2\}$ instead of $\mathcal{C} = \{(x_1), (x_2)\}$.

3 BVA in Terms of Strict Polarized Rectifier Networks

Rectifier networks date back to the 1950s [19], and arise in several different contexts, including circuit complexity, automata theory, and graph theory [8, 14, 15]. In general terms, they provide a way of representing the reachability relationship between vertices of a directed graph by introducing auxiliary vertices with the goal of reducing the total number of edges. This reduction in edge count is the graph-theoretic interpretation of the clause count reduction in Figure 1, and throughout this section, we will develop a precise correspondence between a graph-theoretic and clausal representation for 2-CNF encodings.

There are several ways of representing a 2-CNF formula as a graph, with the most common being the *implication graph*, in which a clause $(\overline{p} \vee q)$ corresponds to directed edges $p \to q$ and $\overline{q} \to \overline{p}$. For our analysis of idealized BVA, having two vertices per variable and two edges per clause will be inconvenient, which motivates us to aim for a graph representation (a *diagram*) in which every variable corresponds to a unique vertex, and every binary clause a unique edge. Binary clauses of mixed polarity, such as $(\overline{p} \vee q)$, are mapped to a directed edge (p, q), while clauses $(\overline{p} \vee \overline{q})$ are mapped to an undirected edge $\{p, q\}$. But clauses of the form $(p \vee q)$ cannot be neatly represented in this framework. To that end, we define the following class of 2-CNF formulas:

▶ **Definition 9** (Simple 2-CNF). *We say a 2-CNF formula φ is* simple *if:*

(a) *φ does not imply that any two literals are equivalent,*
(b) *every clause has at least one negative literal (i.e., φ is Horn), and*
(c) *for every pair of distinct clauses $C_1, C_2 \in \varphi$, $\mathsf{Var}(C_1) \neq \mathsf{Var}(C_2)$.*

The goal of this section is to characterize how idealized BVA operates on simple 2-CNF formulas. There is no real loss of generality in restricting our attention to simple 2-CNF formulas, because any satisfiable 2-CNF formula can be converted to a simple one that is equivalent up to replacing some variables by their negations. This simplification is furthermore efficient, as formalized by the following proposition:

▶ **Proposition 10.** *If every simple 2-CNF formula on n variables has a 2-CNF encoding with at most $f(n)$ clauses, computable in time $g(n)$, then every 2-CNF formula on n variables has a 2-CNF encoding with at most $f(n) + O(n)$ clauses, also computable in time $O(g(n))$.*

Proof sketch. Unsatisfiable formulas can be trivially reencoded into the empty clause, so suppose that φ is a satisfiable 2-CNF formula we wish to reencode. Using equivalent literal substitution, we can reduce to a formula φ' containing no equivalent literals; it suffices to reencode φ', since the equivalences of literals can be appended afterward. Next, let L be the set of literals ℓ such that $(\ell \vee x_j)$ and $(\ell \vee \overline{x_j})$ are both present in φ' for some x_j, and note that $\varphi' \models \ell$ for $\ell \in L$, so $\varphi' \equiv \varphi' \wedge \bigwedge_{\ell \in L} \ell$. Then, applying unit propagation on $\varphi' \wedge \bigwedge_{\ell \in L} \ell$ yields a formula φ'' containing no pairs of clauses over the same two variables; it suffices to reencode φ'', since the unit clauses ℓ can be dealt with afterward. Finally, since φ'' is satisfiable and 2-CNF, it is Horn-renamable, so we can reduce to a formula φ''' obtained by replacing some variables by their negations throughout φ'', and an encoding for φ''' can be transformed into an encoding for φ'' via the same replacement. ◀

Now, we introduce the graph-theoretic notions at the heart of our framework and explain their correspondence to the logical notions we study (see Table 1). A virtue of working with simple 2-CNF formulas is that they have a convenient graph-theoretic representation using what we call *diagrams* (a similar definition appears in [2]):

▶ **Definition 11** (Diagram). *A* diagram $G = (V, E)$ *is a graph with both undirected and directed edges allowed (but no loops) and such that* $|\{\{u, v\}, (u, v), (v, u)\} \cap E| \leq 1$ *for all* $u, v \in V$. *Given a simple 2-CNF formula* φ, *its* associated diagram G_φ *is defined by* $V(G_\varphi) = \mathsf{Var}(\varphi)$ *and* $E(G_\varphi) = \{\{x_i, x_j\} \mid (\overline{x_i} \vee \overline{x_j}) \in \varphi\} \cup \{(x_i, x_j) \mid (\overline{x_i} \vee x_j) \in \varphi\}$.

When we define polarized rectifier networks and their relation to diagrams, it will be convenient to work with *polarized diagrams*, in which each undirected edge of a diagram is oriented arbitrarily (see Figure 2):

▶ **Definition 12** (Polarized Diagram). *Given a diagram G with undirected edges E_1 and directed edges E_2, a* polarized diagram *of G is a pair (G', p) consisting of a directed graph G' obtained from G by orienting every edge in E_1 together with a function $p\colon E(G') \to \{-, +\}$ given by* $p(u, v) = \begin{cases} - & \text{if } \{u, v\} \in E_1 \\ + & \text{if } (u, v) \in E_2. \end{cases}$

When BVA reencodes a formula containing a clause of the form $(\overline{x_i} \vee \overline{x_j})$, it can do so using a chain of implications either of the form $x_i \to y_1 \to \cdots \to y_k \to \overline{x_j}$ or of the form $x_j \to y_1 \to \cdots \to y_k \to \overline{x_i}$, where $y_1, \ldots, y_k$ are auxiliary variables. In a polarized diagram, the orientation of the edge $\{x_i, x_j\}$ represents a choice of which of these alternatives idealized

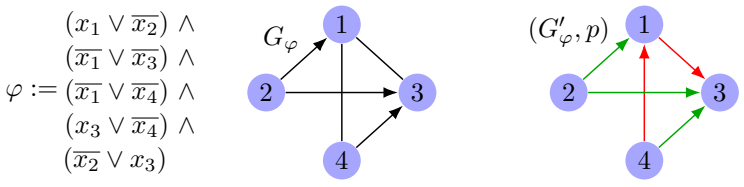


Figure 2 Illustration of a diagram and a polarized diagram for a simple 2-CNF. Edges with *positive polarity* (i.e., $p(u, v) = +$) are colored green, whereas *negative polarity* edges are colored red.

Table 1 Translation between formulas/encodings and graph-theoretic notions.

Formulas and encodings	Graph-theoretic notions
Simple 2-CNF formula	Diagram
Simple 2-CNF formula with auxiliary variables	Polarized rectifier network
Implication chain $x_i \to y_1 \to \ldots \to y_{k-1} \to \pm x_j$	Valid walk $(x_i, y_1, \ldots, y_{k-1}, x_j)$
A simple 2-CNF formula with auxiliary variables *encodes* a simple 2-CNF formula	A polarized rectifier network *realizes* a diagram
Encoding constructible by idealized BVA	Strict polarized rectifier network

BVA may take. Concretely, if a clause $(\overline{x_i} \vee \overline{x_j})$ ever forms part of a BVA step $\mathcal{C} \bowtie \mathcal{D}$, we will orient it as (x_i, x_j) if $\overline{x_i} \in \mathcal{C}$ and $\overline{x_j} \in \mathcal{D}$, and as (x_j, x_i) if $\overline{x_j} \in \mathcal{C}$ and $\overline{x_i} \in \mathcal{D}$. If the clause is never part of a BVA step, then its edge orientation can be chosen arbitrarily.

Next, we extend polarized diagrams into *polarized rectifier networks*, which are polarized diagrams with a distinguished set of *auxiliary vertices*:

▶ **Definition 13** (Polarized Rectifier Network). *A* polarized rectifier network *(PRN) $\mathcal{R}$ is a quadruple (B, A, E, p), where B and A are disjoint sets of* vertices*, $(B \sqcup A, E)$ is a directed graph without loops, and p is a function $\{(u, v) \in E \mid v \in B\} \to \{-, +\}$. We say that B and A are the* base *and* auxiliary *vertices of $\mathcal{R}$ respectively. Given $u, v \in B$ (potentially $u = v$), a* valid walk *from u to v in $\mathcal{R}$ is a sequence of vertices $\pi := (\pi_1, \ldots, \pi_k)$ for some $k \geq 2$ such that $(\pi_1, \pi_k) = (u, v)$, $(\pi_i, \pi_{i+1}) \in E$ for each $i \in [k-1]$, and $\pi_i \in A$ for each $i \in [2, k-1]$. We say that π is* positive *or* negative *according to the value of $p(\pi_{k-1}, \pi_k)$. When a valid walk π is positive, we write $p(\pi) = +$, and $p(\pi) = -$ otherwise.*

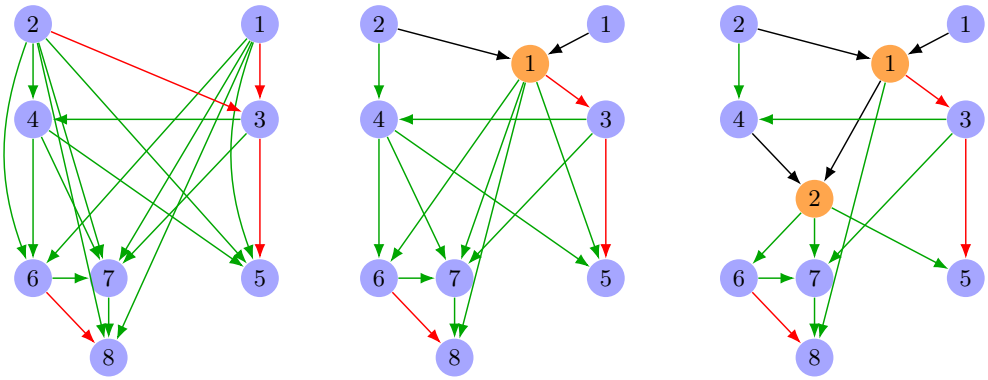
Recall from Table 1 that PRNs correspond to simple 2-CNF formulas with auxiliary variables.[2] If $\mathcal{R} = (B, A, E, p)$ is a PRN, then B and A correspond to base and auxiliary variables respectively. An edge $(u, v) \in E$ corresponds to an implication $u \to v$, unless v is a base vertex and $p(u, v) = -$, in which case it corresponds to the implication $u \to \overline{v}$. A positive (respectively, negative) valid walk from u to v in $\mathcal{R}$ corresponds to a chain of implications $u \to y_2 \to \cdots \to y_{k-1} \to v$ (respectively, $u \to y_2 \to \cdots \to y_{k-1} \to \overline{v}$), where $y_2, \ldots, y_{k-1}$ are auxiliary variables. Next, we define the translation from a PRN to a 2-CNF formula that makes this correspondence precise:

▶ **Definition 14** (PRN to Encoding). *Given a PRN $\mathcal{R} = (B, A, E, p)$, we define a 2-CNF formula $F_{\mathcal{R}}$ over the variables $B \sqcup A$ by mapping each edge $(u, v) \in E$ into a clause $C_{(u,v)}$ as follows: if $v \in B$ and $p(u, v) = -$, then $C_{(u,v)} := (\overline{u} \vee \overline{v})$; otherwise, $C_{(u,v)} := (\overline{u} \vee v)$. Finally, $F_{\mathcal{R}} := \bigwedge_{e \in E} C_e$.*

Next, we define the relation between a PRN $\mathcal{R}$ and a diagram G_φ that holds when $F_{\mathcal{R}}$ is an encoding of φ:

▶ **Definition 15** (PRN Realizing a Diagram). *Given a polarized diagram (G, p), a PRN $\mathcal{R} = (V(G), A, E, p')$* realizes *$(G, p)$ if for every pair $u, v \in V(G)$ (potentially $u = v$), we have $(u, v) \in E(G)$ if and only if there is a valid walk π from u to v in $\mathcal{R}$ such that $p'(\pi) = p(u, v)$. If (G, p) is a polarized diagram of G' and $\mathcal{R}$ realizes (G, p), we also say that $\mathcal{R}$* realizes *G'.*

[2] The correspondence here is not exact. Some simple 2-CNF formulas with auxiliary variables (namely, those containing clauses of the form $(\overline{y_i} \vee \overline{y_j})$, where y_i and y_j are auxiliary variables) do not correspond to polarized rectifier networks. However, our goal is to characterize encodings constructible by idealized BVA (Theorem 18), which never constructs clauses of that form.



(a) (G, p). **(b)** $\mathcal{R}_1$. **(c)** $\mathcal{R}_2$.

Figure 3 Illustration of the BVA steps in Example 19.

The previous definition is motivated by the fact that direct implications in a formula, which correspond to edges of a polarized diagram, must be captured by implication chains in an encoding, which correspond to valid walks in a PRN. Note that given a polarized diagram (G, p), we can naturally define a PRN $\mathcal{R}_{(G,p)} = (V(G), \emptyset, E(G), p)$ that realizes (G, p).

▶ Remark 16. Let φ be a simple 2-CNF formula, and let (G, p) be a polarized diagram of G_φ. Then, $\varphi = F_{\mathcal{R}_{(G,p)}}$.

It turns out that polarized rectifier networks represent a more general class of simple 2-CNF encodings than what can be achieved by idealized BVA. To that end, we define the following:

▶ **Definition 17** (Strict Polarized Rectifier Network). *Let $\mathcal{R} = (B, A, E, p)$ be a PRN. We say that $\mathcal{R}$ is a* strict polarized rectifier network *(SPRN) if the following conditions are satisfied:*

1. *for every $\{u, v\} \in \binom{B}{2}$, $\mathcal{R}$ contains at most one valid walk from u to v or from v to u;*
2. *every vertex $u \in A$ belongs to some valid walk.*

The second condition of this definition rules out "dangling" auxiliary vertices, which serve no purpose and are never constructed by idealized BVA. The first condition is more interesting. It forbids us from realizing an implication between base vertices via multiple valid walks, something idealized BVA will never do. There is, however, no principled justification for this restriction; for some diagrams, the smallest SPRNs realizing them may be asymptotically larger than the smallest PRNs realizing them [16, Chapter 5]. This suggests a possible avenue for improving BVA by allowing it to construct multiple implication chains capturing an implication between base variables.

Our characterization theorem identifies which formulas can be obtained by idealized BVA from a simple 2-CNF formula. To state it, we define the following equivalence relation on formulas: Given formulas φ and φ' and a set of variables W, we write $\varphi \cong_W \varphi'$ if φ' can be obtained from φ by replacing some variables from W by their negations wherever they appear. Naturally, replacing auxiliary variables by their negations is irrelevant for encodings: if φ is an encoding of some boolean function f with auxiliary variables W, and $\varphi \cong_W \varphi'$, then φ' is also an encoding of f. We can now state our characterization theorem:

▶ **Theorem 18** (BVA Characterization). *Let φ and φ' be 2-CNF formulas, where φ is simple. We have $\varphi \overset{BVA}{\rightsquigarrow}{}^* \varphi'$ if and only if there is an SPRN $\mathcal{R}$ realizing the diagram G_φ with $F_\mathcal{R} \cong_W \varphi'$, where $W = \mathsf{Var}(\varphi') \setminus \mathsf{Var}(\varphi)$.*

Since the proof of Theorem 18 is quite technical, let us give some intuition for the result through the following example:

▶ **Example 19.** Consider the following simple 2-CNF formula:

$$\varphi := (\overline{x_1} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_5) \wedge (\overline{x_1} \vee x_6) \wedge (\overline{x_1} \vee x_7) \wedge (\overline{x_1} \vee x_8) \wedge (\overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_5) \\ \wedge (\overline{x_2} \vee x_4) \wedge (\overline{x_2} \vee x_6) \wedge (\overline{x_2} \vee x_7) \wedge (\overline{x_2} \vee x_8) \wedge (\overline{x_3} \vee \overline{x_5}) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_3} \vee x_7) \\ \wedge (\overline{x_4} \vee x_5) \wedge (\overline{x_4} \vee x_6) \wedge (\overline{x_4} \vee x_7) \wedge (\overline{x_6} \vee x_7) \wedge (\overline{x_6} \vee \overline{x_8}) \wedge (\overline{x_7} \vee x_8).$$

A polarized diagram for φ is given in Figure 3a. As our first BVA step, we take $\mathcal{C} = \{\overline{x_1}, \overline{x_2}\}$ and $\mathcal{D} = \{\overline{x_3}, x_5, x_6, x_7, x_8\}$, and replace $\mathcal{C} \bowtie \mathcal{D}$ by $\{(C_i \vee y_1) \mid C_i \in \mathcal{C}\} \cup \{(\overline{y_1} \vee D_j) \mid D_j \in \mathcal{D}\}$, where y_1 is a new auxiliary variable. After this BVA step, the formula corresponds to the rectifier network in Figure 3b. For the second BVA step, we take $\mathcal{C} = \{\overline{x_4}, \overline{y_1}\}$ and $\mathcal{D} = \{x_5, x_6, x_7\}$, and replace $\mathcal{C} \bowtie \mathcal{D}$ by $\{(C_i \vee y_2) \mid C_i \in \mathcal{C}\} \cup \{(\overline{y_2} \vee D_j) \mid D_j \in \mathcal{D}\}$, where y_2 is a new auxiliary variable. The resulting formula corresponds to the rectifier network in Figure 3c.

4 Encodings for General 2-CNF Formulas

As a first application of our characterization of idealized BVA (Theorem 18), we answer the following natural question: Given an arbitrary 2-CNF formula, how many clauses are in its smallest 2-CNF encoding constructible by idealized BVA? The answer, which depends on whether we use the simplification from Proposition 10, is presented in Table 2.

Table 2 Size of the smallest 2-CNF reencoding of a 2-CNF formula in the worst case.

	Lower bound	**Upper bound**
Idealized BVA	$(1 - o(1))\,\frac{n^2}{\lg n}$	$(1 + o(1))\,\frac{n^2}{\lg n}$
Idealized BVA with simplification	$\left(\frac{\lg(3)}{4} - o(1)\right)\frac{n^2}{\lg n}$	$\left(\frac{\lg(3)}{4} + o(1)\right)\frac{n^2}{\lg n}$ (Corollary 21)
Arbitrary 2-CNF reencoding methods	$\left(\frac{1}{4} - o(1)\right)\frac{n^2}{\lg n}$ (Proposition 25)	$\left(\frac{\lg(3)}{4} + o(1)\right)\frac{n^2}{\lg n}$ (Corollary 21)

Notice that, although a 2-CNF formula can have $\Omega(n^2)$ clauses, it always has an encoding with $O(n^2/\lg n)$ clauses, and such an encoding can be constructed by idealized BVA. Furthermore, there is no reencoding method that can improve on this by more than a factor of 4. On the other hand, the simplification from Proposition 10 yields an improved bound when compared to idealized BVA alone. In fact, Corollary 21 proves that there is an implementation of idealized BVA with simplification that achieves the stated bound whose runtime is $O(n^2)$; note that the time complexity of factor, the BVA implementation used in CaDiCaL and Kissat, is at least $\Omega(n^3)$.

The upper bounds in Table 2 make essential use of our characterization of idealized BVA in terms of SPRNs. Specifically, Corollary 21 employs a technique used by Nechiporuk [21] to construct small rectifier networks for bipartite graphs; we generalize his result to our setting of polarized rectifier networks, following the line of work of Krapivin et al. [18], which used similar techniques to extend results about *biclique partitions* (i.e., strict rectifier networks without edges between auxiliary vertices) from bipartite graphs to arbitrary graphs.

4.1 Upper Bound for BVA With Simplification

To prove Corollary 21, we start by proving a similar result for simple 2-CNF formulas:

▶ **Theorem 20** (Formal version of Theorem 2). *For every simple 2-CNF formula φ on n variables, there is a 2-CNF formula φ' such that $\varphi \overset{BVA}{\rightsquigarrow}{}^{*} \varphi'$ and $|\varphi'| \leq \left(\frac{\lg(3)}{4} + o(1)\right) \frac{n^2}{\lg n}$.*

Proof sketch. Let G_φ be the associated diagram of φ. By Theorem 18, it suffices to build an SPRN $\mathcal{R}$ realizing G_φ with at most $\left(\frac{\lg(3)}{4} + o(1)\right) \frac{n^2}{\lg n}$ edges. Since φ is simple, the directed part of G_φ is acyclic, so we can topologically order the vertices $v_1, \ldots, v_n$ so that every directed edge (v_i, v_j) satisfies $i < j$. We create a polarized diagram (G, p) of G_φ by orienting each $\{v_i, v_j\} \in E(G_\varphi)$ to be (v_i, v_j), where $i < j$. Our SPRN $\mathcal{R} = (V(G), A, E, p')$ will be a realization of (G, p) and thus of G_φ.

Let $\log_3 x = \frac{\lg x}{\lg 3}$. Let $q = \lfloor n / \log_3^2 n \rfloor$, and partition $V(G)$ into blocks $B_1, \ldots, B_{\lceil n/q \rceil}$ of size at most q; specifically, let $B_i = \{v_{(i-1)\cdot q+1}, \ldots, v_{i\cdot q}\}$ or, if $i \cdot q > n$, a truncation thereof. Within each block B_i, for each pair of vertices $\{u, v\} \in \binom{B_i}{2}$ create in $\mathcal{R}$ an auxiliary vertex $z_{\{u,v\}}$, and create edges $(u, z_{\{u,v\}})$ and $(v, z_{\{u,v\}})$. Now, let $r = \lfloor \log_3 n - 3 \log_3 \log_3 n \rfloor$, and partition as well $V(G)$ into parts $P_1, \ldots, P_{\lceil n/r \rceil}$ of size at most r; specifically, let $P_i = \{v_{(i-1)\cdot r+1}, \ldots, v_{i\cdot r}\}$ or, if $i \cdot r > n$, a truncation thereof. For each part P_i and edge $(u, v) \in E(G) \cap (P_i \times P_i)$, create in $\mathcal{R}$ a directed edge (u, v) with $p'(u, v) = p(u, v)$.

Now, for each part P_i and each function $S \colon P_i \to \{-, 0, +\}$, create in $\mathcal{R}$ an auxiliary vertex y_S, and create a directed edge (y_S, v) with $p'(y_S, v) = -$ for every $v \in P_i$ satisfying $S(v) = -$, and create a directed edge (y_S, v) with $p'(y_S, v) = +$ for every $v \in P_i$ satisfying $S(v) = +$. Given a vertex $v \in V(G)$, let $N^{\pm}(v) = \{w \in V(G) \mid (v, w) \in E(G) \text{ and } p(v, w) = \pm\}$. Then, for each $S \colon P_i \to \{-, 0, +\}$, let

$$A(S) := \{v \in P_j \mid j < i,\ N^-(v) \cap P_i = S^{-1}(-),\ \text{and } N^+(v) \cap P_i = S^{-1}(+)\}.$$

Finally, for each $\ell \in [\lceil n/q \rceil]$, let $A_\ell(S) := A(S) \cap B_\ell$, and let $a_1, \ldots, a_k$ be the elements of $A_\ell(S)$ in an arbitrary order. Then, create in $\mathcal{R}$ directed edges $(z_{\{a_{2j-1}, a_{2j}\}}, y_S)$ for $j \in [\lfloor k/2 \rfloor]$, and if k is odd, create the directed edge (a_k, y_S). If $\mathcal{R}$ contains any auxiliary vertices not belonging to a valid walk, then delete them and their incident edges.

This completes the construction of $\mathcal{R}$. The full proof (in the arXiv version) proves that $\mathcal{R}$ is a strict rectifier network for (G, p) and has at most $\left(\frac{\lg(3)}{4} + o(1)\right) \frac{n^2}{\lg n}$ edges. ◀

Together with the simplification from Proposition 10, we obtain the following:

▶ **Corollary 21.** *Every 2-CNF formula φ on n variables has a 2-CNF encoding φ' such that $|\varphi'| \leq (\frac{\lg(3)}{4} + o(1)) \frac{n^2}{\lg n}$, and moreover, such an encoding can be computed in time $O(n^2)$.*

Proof. It suffices to combine Proposition 10 with Theorem 20, and observe from the previous proof that the construction can be carried out in time $O(n^2)$.[3] ◀

4.2 Lower Bound for BVA With Simplification

Using an information-theoretic argument, we now show that the bound from Theorem 20 is sharp. We start by bounding the number of 2-CNF formulas with at most m clauses and the number of simple 2-CNF formulas on n variables.

[3] A similar construction, with pseudocode, appears in [18, Algorithm 1].

▶ **Lemma 22.** *The number of 2-CNF formulas with at most m clauses is at most $2^{(1+o(1))m\cdot\lg m}$.*

▶ **Lemma 23.** *The number of simple 2-CNF formulas on n variables is at least $3^{(1/2-o(1))n^2}$.*

Now, we can combine these two lemmas to prove the lower bound:

▶ **Proposition 24.** *There is a simple 2-CNF formula φ on n variables such that, for every φ' with $\varphi \overset{\text{BVA}}{\rightsquigarrow\!\triangleright}{}^*\varphi'$, we have $|\varphi'| \geq \left(\frac{\lg(3)}{4} - o(1)\right)\frac{n^2}{\lg n}$.*

Proof. Let $g(m)$ be the number of 2-CNF formulas with at most m clauses, and let $h(n)$ be the number of simple 2-CNF formulas on n variables. By Lemmas 22 and 23, $\lg g(m) \leq (1+o(1))m\cdot\lg m$ and $\lg h(n) \geq (\lg(3)/2 - o(1))n^2$.

If $\varphi \overset{\text{BVA}}{\rightsquigarrow\!\triangleright}{}^*\varphi'$, then performing variable elimination (in the Davis–Putnam sense) on the auxiliary variables in φ' yields φ. In particular, φ' has enough information to uniquely reconstruct φ. Thus, if for every simple 2-CNF formula φ on n variables, there is a 2-CNF formula φ' with at most m clauses such that $\varphi \overset{\text{BVA}}{\rightsquigarrow\!\triangleright}{}^*\varphi'$, then $g(m) \geq h(n)$, and consequently $\lg g(m) \geq \lg h(n)$. If $m \leq (r+o(1))n^2/\lg n$ for some constant r, then $\lg m \leq (2+o(1))\lg n$, and thus

$$(\lg(3)/2 - o(1))n^2 \leq \lg h(n) \leq \lg g(m) \leq (1+o(1))m\cdot\lg m \leq (2r+o(1))n^2,$$

from where $r \geq \lg(3)/4$; that is, $m \geq (\frac{\lg(3)}{4} - o(1))n^2/\lg n$. ◀

4.3 Monotone Formulas

The proof of Proposition 24 exploits the fact that idealized BVA reencodes distinct simple 2-CNF formulas into distinct encodings. To get a lower bound for arbitrary reencoding methods, which may not have this property, we focus on monotone 2-CNF formulas:

▶ **Proposition 25.** *There is a monotone 2-CNF formula with n variables whose smallest 2-CNF encoding has at least $(\frac{1}{4} - o(1))\frac{n^2}{\lg n}$ clauses.*

In fact, idealized BVA can match this lower bound:

▶ **Theorem 26** (Formal version of Theorem 3)**.** *For every monotone 2-CNF formula φ on n variables, there is a 2-CNF formula φ' such that $\varphi \overset{\text{BVA}}{\rightsquigarrow\!\triangleright}{}^*\varphi'$ and $|\varphi'| \leq \left(\frac{1}{4} + o(1)\right)\frac{n^2}{\lg n}$.*

This strengthens a result of Subercaseaux [24, Theorem 10], who proved that every monotone 2-CNF formula has a 2-CNF encoding with $O(n^2/\lg n)$ clauses.

Allen [2] proved the remarkable result that almost all 2-CNF functions are monotone after possibly negating some input variables (i.e., *unate*). Together with Theorem 26, this implies that almost every 2-CNF function on n variables has a 2-CNF encoding with at most $(\frac{1}{4} + o(1))n^2/\lg n$ clauses, matching the lower bound from Proposition 25. On this basis, we conjecture that this bound holds for *all* 2-CNF functions, which would imply that there is a reencoding method improving idealized BVA with simplification:

▶ **Conjecture 27.** *Every 2-CNF formula φ on n variables has a 2-CNF encoding φ' such that $|\varphi'| \leq (\frac{1}{4} + o(1))\frac{n^2}{\lg n}$.*

5 Encodings for AtMostOne

The previous section showed that idealized BVA performs well from the perspective of worst-case analysis. But many 2-CNF formulas that arise in practice are highly structured and can therefore be encoded with far fewer than $\Theta(n^2/\lg n)$ clauses. Our framework also allows us to analyze the operation of idealized BVA on specific 2-CNF formulas. We demonstrate this by studying encodings of the most structured 2-CNF formula of all, one which also frequently arises in practice, namely the at-most-one constraint:

$$\mathsf{AtMostOne}(x_1, \ldots, x_n) \coloneqq \bigwedge_{1 \le i < j \le n} (\overline{x_i} \vee \overline{x_j}).$$

Actual implementations of BVA reencode this formula into one with $3n - 6$ clauses [20]; while this has been empirically observed before, we formalize and prove this fact in the full version of this paper. By Theorem 18, a BVA-constructible encoding for this formula corresponds to an SPRN realizing K_n. Figure 4 depicts the corresponding SPRN for the encoding with $3n - 6$ edges.

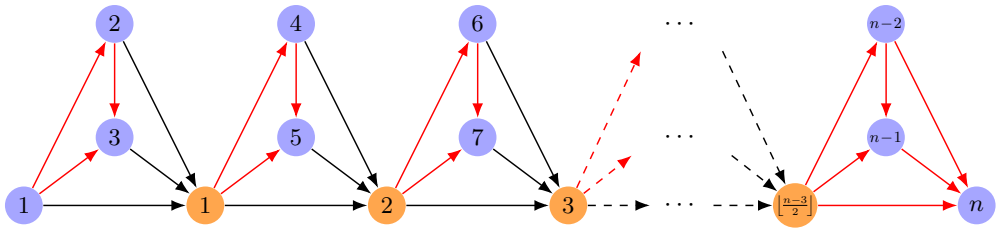


Figure 4 SPRN realizing K_n with $3n - 6$ edges.

Using Theorem 18, we prove that $3n - 6$ clauses is the smallest possible encoding for this constraint achievable by idealized BVA:

▶ **Theorem 28** (Formal version of Theorem 4). *For every φ with $\mathsf{AtMostOne}(x_1, \ldots, x_n) \overset{\mathit{BVA}}{\rightsquigarrow\!\triangleright}{}^{*} \varphi$, we have $|\varphi| \ge 3n - 6$.*

Note that there are 2-CNF encodings for $\mathsf{AtMostOne}(x_1, \ldots, x_n)$ using $2n + o(n)$ clauses [7], and it was previously unknown whether idealized BVA could construct these. Thus, for a reencoding method to encode $\mathsf{AtMostOne}(x_1, \ldots, x_n)$ using fewer than $3n - 6$ clauses, it must be able to create encodings that cannot be expressed as SPRNs.

We start with some lemmas that will help us establish properties of a minimal counterexample to Theorem 28.

▶ **Lemma 29.** *Let $\mathcal{R}$ be an SPRN with m edges realizing a diagram G, and let y be an auxiliary vertex. If the in-degree or out-degree of y is 1, then there is an SPRN realizing G with at most $m - 1$ edges.*

Proof sketch. If (y', y) is the unique incoming edge to y, or (y, y') is the unique outgoing edge from y, then we can contract this edge to get an SPRN with one fewer edge. ◀

▶ **Lemma 30.** *Let $\mathcal{R}$ be an SPRN realizing K_n with m edges for some $n \ge 4$, and let x be a base vertex. If the total degree of x is at least 3, then there is an SPRN realizing K_{n-1} with at most $m - 3$ edges.*

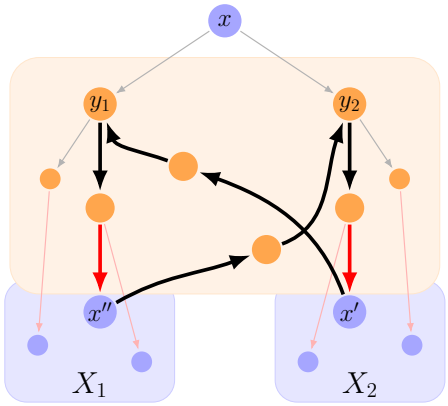


Figure 5 Illustration for the proof of Theorem 28.

Proof. Let $\mathcal{R}'$ be obtained from $\mathcal{R}$ by deleting x and its incident edges; also if $\mathcal{R}'$ contains any auxiliary vertices not belonging to a valid walk, then delete them and their incident edges. Then, $\mathcal{R}'$ is an SPRN realizing K_{n-1} and has at most $m-3$ edges. ◀

▶ **Lemma 31.** *Let $\mathcal{R}$ be an SPRN realizing K_n with m edges for some $n \geq 4$, and let x be a base vertex. If the total degree of x is exactly 1, then there is an SPRN realizing K_n with at most $m-1$ edges.*

Proof sketch. If the total degree of x is exactly 1, then its unique neighbor must be an auxiliary vertex y. Without loss of generality, (x, y) is the edge in $\mathcal{R}$ connecting these two vertices. Then, y has in-degree exactly 1, so the conclusion follows by Lemma 29. ◀

▶ **Lemma 32.** *Let $\mathcal{R}$ be an SPRN realizing K_n with m edges for some $n \geq 4$, and let x_1 and x_2 be base vertices. If the total degrees of x_1 and x_2 in $\mathcal{R}$ are both 2 and there is an edge between x_1 and x_2, then there is an SPRN realizing K_{n-1} with at most $m-3$ edges.*

Proof. Let $\mathcal{R}'$ be obtained from $\mathcal{R}$ by deleting x_1 and its two incident edges; also if $\mathcal{R}'$ contains any auxiliary vertices not belonging to a valid walk, then delete them and their incident edges. Then $\mathcal{R}'$ is an SPRN realizing K_{n-1} and has at most $m-2$ edges. In $\mathcal{R}'$, the total degree of x_2 is exactly 1, so there is an SPRN realizing K_{n-1} with at most $m-3$ edges by Lemma 31. ◀

We are now ready to prove Theorem 28. Given a minimal counterexample (an SPRN realizing K_n with fewer than $3n-6$ edges), we derive a contradiction by showing that the SPRN is not strict. Specifically, we identify two base vertices x' and x'' such that we have a valid walk from x' to x'' and a valid walk from x'' to x'. See Figure 5 for an illustration of the proof.

Proof of Theorem 28. Let $\mathcal{R}$ be an SPRN realizing K_n with the minimum number of edges. By Theorem 18, it suffices to show that $\mathcal{R}$ has at least $3n-6$ edges. We may assume that $n \geq 3$. The proof is by induction on n. The base case, $n = 3$, is easy to check. If $n \geq 4$, and there is some base vertex whose total degree is at least 3, then we are done by Lemma 30. So assume that every base vertex has total degree at most 2. By Lemma 31, every base vertex has total degree exactly 2. By Lemma 32, we may assume that there are no edges between two base vertices.

From these assumptions, we will arrive at a contradiction. Fix a base vertex x, and let y_1 and y_2 be its two adjacent auxiliary vertices. Suppose that the edges incident to x are (x, y_1) and (x, y_2); if either edge goes in the other direction, the rest of the argument proceeds with only minor modifications. For $i \in \{1, 2\}$, let X_i be the elements of $V(K_n) \setminus \{x\}$ reachable by a valid walk starting with (x, y_i). Then, since $\mathcal{R}$ is strict, $X_1 \cup X_2$ is a partition of $V(K_n) \setminus \{x\}$. By Lemma 29, there is a base vertex $x' \in V(K_n) \setminus \{x\}$ with a valid walk to y_1 and a base vertex $x'' \in V(K_n) \setminus \{x\}$ with a valid walk to y_2. We must have $x' \in X_2$, or else there would be a valid walk $(x', \ldots, y_1, \ldots, x')$, which would contradict our assumption that $\mathcal{R}$ is an SPRN realizing K_n. Similarly, $x'' \in X_1$. But then, we have valid walks $(x', \ldots, y_1, \ldots, x'')$ and $(x'', \ldots, y_2, \ldots, x')$, which contradicts our assumption that $\mathcal{R}$ is strict. ◀

6 Discussion and Experimental Results

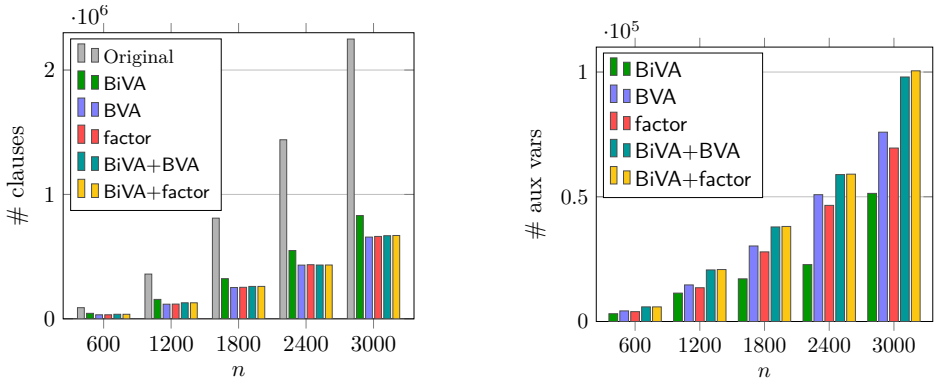
We have developed a theoretical framework to better understand Bounded Variable Addition (BVA) as a preprocessing technique, especially when accompanied by simplification techniques such as equivalent literal substitution and failed literal elimination.

We have focused on characterizing BVA's behavior and reencoding potential on the 2-CNF fragment of formulas. This not only simplifies the theoretical analysis, but is also justified by practical considerations. Namely, structured formulas are heavily biased towards narrow clauses: analyzing the 2025 SAT Competition, over 60% of clauses have width 2 (avg. width is about 2.67, and fewer than 2% have width greater than 4). When factor, the state-of-the-art implementation of BVA, is run on these formulas, over 70% of the clauses saved by the reencoding correspond to binary clauses, and fewer than 4% correspond to width greater than 3. Thus, for size-reducing BVA-style reencoding, width 3 appears to be the next target, although wider clauses may still be important for solver performance.

Generalizing our framework from 2-CNF to k-CNF will require reasoning about hypergraphs instead of graphs. Some initial work in this direction was recently done by Krapivin et al. [18], who studied k-partite decompositions of k-uniform graphs, which can be seen as depth-2 rectifier networks for hypergraphs. In particular, their work implies that every monotone k-CNF formula on n variables can be encoded with at most $(\frac{1}{k!} + o_k(1))n^k/k!$ clauses. We also remark that Allen's result that almost all 2-CNF functions are unate has recently been generalized to k-CNF functions [3].

Within the 2-CNF fragment, we have both established results for very structured formulas, such as encodings of the at-most-one constraint, as well as for general unstructured (e.g., random) formulas. For unstructured formulas, we have established sharp bounds on the reencoding potential of idealized BVA. While our main focus has been theoretical, we leverage an efficient algorithm of Krapivin et al. [18] for computing biclique partitions to implement a tool, BiVA (Biclique Variable Addition), which reencodes monotone 2-CNF fragments of formulas. Since BiVA is optimized for worst-case formulas, we do not aim to compete with other BVA implementations on structured formulas. Thus, we benchmark on formulas encoding the independent set problem on $G(n, \frac{1}{2})$-random graphs.

We benchmark against the BVA implementation from [20] (available at `https://fmv.jku.at/bva/`), factor from Kissat (v4.0.4), and interestingly, we consider combinations BiVA+BVA and BiVA+factor, meaning that the reencoded output of BiVA is fed for a second round of reencoding (in contrast, if BVA or factor are applied to completion, then running BiVA on top has no effect). We do not include SBVA in our experiments since its exploitation of structure was not beneficial on random formulas, and the reencoding time was significantly longer than for the other methods.



(a) clauses. **(b)** auxiliary variables.

Figure 6 Comparison of reencoding methods on random monotone formulas.

First, as shown in Figure 6a, the reduction in clauses is comparable between all methods, except for a 5–15% margin by which BiVA is worse; however, this is almost entirely compensated by the combination with BVA or factor. In fact, as shown in Figure 7, these combinations run much more efficiently than BVA/factor in isolation. This can be explained by the fact that their runtimes depend on the amount of compression they achieve, and thus they run much faster on formulas that are already partially compressed. Finally, as shown in Figure 6b, BiVA creates significantly fewer auxiliary variables, and unfortunately, its combination with BVA/factor generates the most in aggregate. More experimental details are provided in the full version of this paper [22, Section D].

Future directions. Our characterization of idealized BVA reveals not only its expressive power but also specific limitations that could be overcome by future reencoding methods. For example, one of BVA's limitations arises from its *strictness*, which algorithmically corresponds to the fact that upon reencoding a biclique, BVA removes the original edges immediately, preventing any future reuse of them. This is necessary (but not sufficient) for automatically deriving the product encoding for AtMostOne. Given our promising preliminary experimental results, another direction is to develop a BiVA-inspired implementation that can exploit the structure present in real-world formulas. One step in this direction was accomplished by Krapivin et al. [18], who showed how to efficiently construct biclique partitions that exploit the edge density of the input graph.

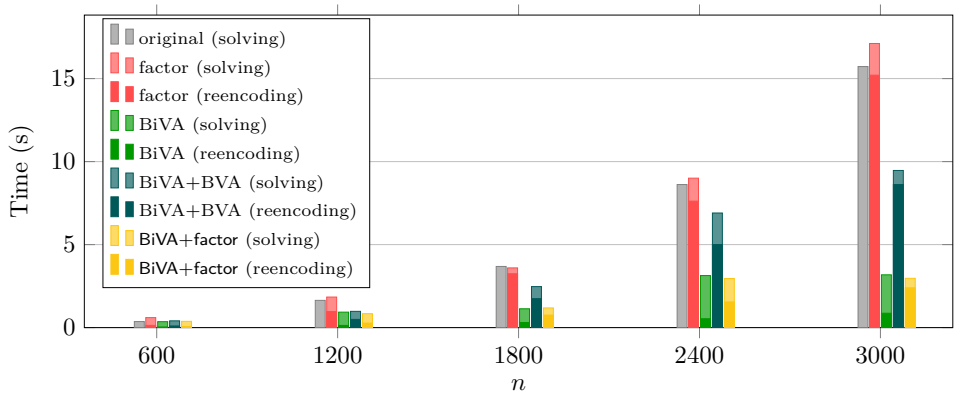


Figure 7 End-to-end runtime comparison of reencoding methods on random monotone formulas.

References

1 Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. A parametric approach for smaller and better encodings of cardinality constraints. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, pages 80–96, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

2 Peter Allen. Almost every 2-SAT function is unate. *Israel J. Math.*, 161:311–346, 2007. doi:10.1007/s11856-007-0081-z.

3 József Balogh, Dingding Dong, Bernard Lidický, Nitya Mani, and Yufei Zhao. Nearly all k-SAT functions are unate. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 958–962, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3564246.3585123.

4 Tomas Balyo, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2023: Solver, Benchmark and Proof Checker Descriptions*. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2023.

5 Armin Biere. exactly one clauses · Issue #39 · arminbiere/kissat – github.com. https://github.com/arminbiere/kissat/issues/39#issuecomment-1686043817, 2023. [Accessed 06-06-2025].

6 Armin Biere. cadical/src/factor.cpp at master · arminbiere/cadical – github.com. https://github.com/arminbiere/cadical/blob/master/src/factor.cpp, 2026. [Accessed 06-02-2026].

7 Jingchao Chen. A new SAT encoding of the at-most-one constraint. *Proc. of the Tenth Int. Workshop of Constraint Modelling and Reformulation*, page 8, 2010.

8 Dmitry Chistikov, Szabolcs Iván, Anna Lubiw, and Jeffrey Shallit. Fractional coverings, greedy coverings, and rectifier networks. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*, volume 66 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:14. Dagstuhl, 2017. doi:10.4230/LIPIcs.STACS.2017.23.

9 Vijay Ganesh and Moshe Y. Vardi. *On the Unreasonable Effectiveness of SAT Solvers*, pages 547–566. Cambridge University Press, 2021. doi:10.1017/9781108637435.032.

10 Andrew Haberlandt, Harrison Green, and Marijn J. H. Heule. Effective auxiliary variables via structured reencoding. In Meena Mahajan and Friedrich Slivovsky, editors, *26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023)*, volume 271 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:19. Dagstuhl, 2023. doi:10.4230/LIPIcs.SAT.2023.11.

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

12 Marijn J. H. Heule, Matti Järvisalo, and Armin Biere. Efficient CNF simplification based on binary implication graphs. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing – SAT 2011*, pages 201–215. Springer, 2011. doi:10.1007/978-3-642-21581-0_17.

13 Marijn J. H. Heule and Manfred Scheucher. Happy ending: An empty hexagon in every set of 30 points. In Bernd Finkbeiner and Laura Kovács, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 61–80, Cham, 2024. Springer Nature Switzerland.

14 Szabolcs Iván, Ádám D. Lelkes, Judit Nagy-György, Balázs Szörényi, and György Turán. Biclique coverings, rectifier networks and the cost of ε-removal. In Helmut Jürgensen, Juhani Karhumäki, and Alexander Okhotin, editors, *Descriptional Complexity of Formal Systems*, pages 174–185, Cham, 2014. Springer International Publishing.

15 Stasys Jukna. Computational Complexity of Graphs. In *Advances in Network Complexity*, chapter 5, pages 99–153. John Wiley & Sons, Ltd, 2013. doi:10.1002/9783527670468.ch05.

16 Stasys Jukna and Igor Sergeev. Complexity of linear boolean operators. *Found. Trends Theor. Comput. Sci.*, 9(1):1–123, 2013. doi:10.1561/0400000063.

17 Markus Kirchweger, Tomáš Peitl, Bernardo Subercaseaux, and Stefan Szeider. From the finite to the infinite: Sharper asymptotic bounds on Norin's conjecture via SAT, 2025. doi:10.48550/arXiv.2511.08386.

18 Andrew Krapivin, Benjamin Przybocki, Nicolás Sanhueza-Matamala, and Bernardo Subercaseaux. Optimal and efficient partite decompositions of hypergraphs, 2025. doi:10.48550/arXiv.2511.11855.

19 Oleg Lupanov. On rectifier and switching-and-rectifier circuits. *Doklady Academii nauk SSSR*, 111, 1956. Available at https://web.vu.lt/mif/s.jukna/boolean/lupanov56.pdf, thanks to Stasys Jukna.

20 Norbert Manthey, Marijn J. H. Heule, and Armin Biere. Automated reencoding of boolean formulas. In *Haifa Verification Conference*, pages 102–117. Springer, 2012. doi:10.1007/978-3-642-39611-3_14.

21 Éduard Ivanovich Nechiporuk. The topological principles of self-correction. *Problemy Kibernet.*, 21:5–102, 1969. Available in Russian at https://web.vu.lt/mif/s.jukna/boolean/Russians/Nechiporuk-1969a.pdf, thanks to Stasys Jukna.

22 Benjamin Przybocki, Bernardo Subercaseaux, and Marijn J. H. Heule. Automated reencoding meets graph theory, 2026. doi:10.48550/arXiv.2603.27774.

23 Long Qian, Eric Wang, Bernardo Subercaseaux, and Marijn J. H. Heule. Unfolding boxes with local constraints. In *Automated Deduction – CADE 30: 30th International Conference on Automated Deduction, Stuttgart, Germany, July 28-31, 2025, Proceedings*, pages 736–754, Berlin, Heidelberg, 2025. Springer-Verlag. doi:10.1007/978-3-031-99984-0_38.

24 Bernardo Subercaseaux. Asymptotically smaller encodings for graph problems and scheduling, 2025. (*ModRef workshop*). doi:10.48550/arXiv.2506.14042.

25 Bernardo Subercaseaux and Marijn J. H. Heule. The packing chromatic number of the infinite square grid is 15. In Sriram Sankaranarayanan and Natasha Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 29th International Conference, TACAS 2023*, volume 13993 of *Lecture Notes in Computer Science*, pages 389–406. Springer, 2023. doi:10.1007/978-3-031-30823-9_20.