CV-width: A New Complexity Parameter for CNFs

Umut Oztok and **Adnan Darwiche** [1]

Abstract. We present new complexity results on the compilation of CNFs into DNNFs and OBDDs. In particular, we introduce a new notion of width, called *CV-width,* which is specific to CNFs and that dominates the treewidth of the CNF incidence graph. We then show that CNFs can be compiled into *structured* DNNFs in time and space that are exponential only in CV-width. Not only does CV-width dominate the incidence graph treewidth, but the former width can be bounded when the latter is unbounded. We also introduce a restricted version of CV-width, called *linear CV-width,* and show that it dominates both pathwidth and cutwidth, which have been used to bound the complexity of OBDDs. We show that CNFs can be compiled into OBDDs in time and space that are exponential only in linear CV-width. We also show that linear CV-width can be bounded when pathwidth and cutwidth are unbounded. The new notion of width significantly improves existing upper bounds on both structured DNNFs and OBDDs, and is motived by a new decomposition technique that combines variable splitting with clause splitting.

1 Introduction

Decomposability is a fundamental property that underlies many well-known tractable languages in propositional logic. It is a property of conjunctions, requiring that conjuncts share no variables, and is sufficient to ensure the tractability of certain queries, such as clausal entailment and the existential quantification of multiple variables [4]. Decomposability is the characteristic property of decomposable negation normal form (DNNF) [2], which includes many other languages such as structured DNNF [8], sentential decision diagrams (SDD) [3], and ordered binary decision diagrams (OBDD) [1].

Compiling CNFs into decomposable languages has been at the center of attention in the area of knowledge compilation. A key interest here is in providing upper bounds on the complexity of compilation algorithms, based on structural parameters of the input CNF (e.g., [2, 5, 10, 3, 11]). These bounds are based on the treewidth of various graph abstractions of the input CNF (e.g., primal, dual and incidence graphs) [12], in addition to the cutwidth and pathwidth of the CNF [5]. For example, the best known upper bound on compiling DNNFs is based on the treewidth of the CNF incidence graph [11]. Moreover, the best known upper bounds on compiling OBDDs are based on the CNF pathwidth and cutwidth [5].

We significantly improve on these bounds in this paper. In particular, we introduce a new notion of width for CNFs, called clause-variable width (CV-width), which dominates the treewidth of the incidence graph and can be bounded when the mentioned treewidth is unbounded. We then show that CNFs can be compiled into *structured* DNNFs in time and space that are exponential only in CV-width. Not only does this improve on the best known bound for compiling DNNFs [11], but it also extends the bound to *structured* DNNF [10]. The significance here is that structured DNNF supports a polytime conjoin operation [8], while (unstructured) DNNF does not support this (unless P=NP) [4]. We also improve on the best known bounds for compiling OBDDs by introducing the notion of *linear CV-width,* which is a restricted version of CV-width. We show that linear CV-width dominates both the pathwidth and cutwidth of a CNF, and can be bounded when these widths are unbounded. We also show that OBDDs can be compiled in time and space that are exponential only in linear CV-width.

Our complexity results are constructive as they are based on a specific algorithm for compiling CNFs into structured DNNFs (and OBDDs). This algorithm is driven by a tree over CNF variables, known as a *vtree* [8]. Each vtree has its own CV-width. Moreover, the CV-width of a given CNF is the smallest width attained by any of its vtrees. The major characteristic of this algorithm is its employment of both variable and clause splitting. Variable splitting is a well-known technique in both SAT and knowledge compilation and calls for eliminating a variable V from a CNF Δ by considering the CNFs $\Delta|v$ and $\Delta|\neg v$ (i.e., conditioning Δ on both phases of the variable). Clause splitting, however, is a less common technique and calls for eliminating a clause $\alpha \vee \beta$ from a CNF Δ by considering the CNFs $\Delta \cup \{\alpha\}$ and $\Delta \cup \{\beta\}$. Our proposed algorithm combines both techniques. This combination is essential for the complexity of our compilation algorithm and provides the major insight underlying the new notion of CV-width. Moreover, the combination allows us to bound the complexity of compilation in situations where this complexity could not be bounded using either technique alone.

This paper is structured as follows. We start by providing some technical preliminaries, and formal definitions of variable and clause splitting (Sections 2–5). This is followed by presenting our compilation algorithm (Section 6). Then, we introduce CV-width and compare it to well-known graph abstractions of CNFs and their corresponding parameters (Sections 7–8). We close with a discussion of related work and some concluding remarks. Due to space limitations, some proofs are delegated to the full version of the paper. [2]

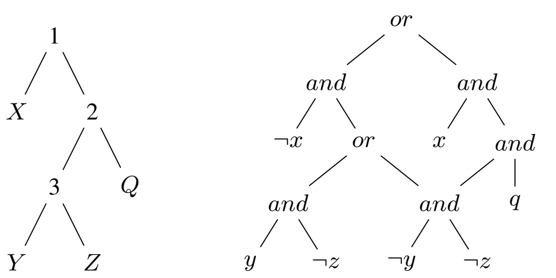
2 Technical Preliminaries

A conjunction is *decomposable* if each pair of its conjuncts share no variables. A *negation normal form* (NNF) is a DAG whose internal nodes are labeled with disjunctions and conjunctions, and whose leaf nodes are labeled with literals or the constants *true* and *false*. An NNF is decomposable (called a DNNF) iff each of its conjunctions is decomposable; see Figure 1(b). We use $Vars(N)$ to denote the set of variables mentioned by an NNF node N.

A *vtree* for a set $\mathbf{Z}$ of variables is a rooted, full binary tree whose leaves are in one-to-one correspondence with variables in $\mathbf{Z}$. Fig-

[1] Computer Science Department, University of California, Los Angeles, email: {umut,darwiche@cs.ucla.edu}

[2] Available at http://reasoning.cs.ucla.edu.



(a) A vtree (b) A DNNF respecting vtree in (a)

Figure 1. A vtree and a respecting DNNF.

ure 1(a) depicts an example vtree. We will use v^l and v^r to refer to the left and right children of internal vtree node v. We will also use $Vars(v)$ to denote the set of variables at or below a vtree node v.

A DNNF respects a vtree iff every and-node N has exactly two children N^l and N^r, and we have $Vars(N^l) \subseteq Vars(v^l)$ and $Vars(N^r) \subseteq Vars(v^r)$ for some vtree node v. In this case, the DNNF is said to be *structured.* The DNNF in Figure 1(b) respects the vtree in Figure 1(a) and is therefore a structured DNNF. OBDDs are a subset of structured DNNFs with stronger properties [7].

The literals of variable X are denoted by x and $\neg x$. A CNF is a set of clauses, where each clause is a disjunction of literals (e.g., $\{x \vee \neg y \vee \neg z,\ \neg x,\ y \vee z\}$). We will often write $\Delta(\mathbf{X})$ to mean that CNF Δ mentions only variables in $\mathbf{X}$. Conditioning a CNF Δ on a literal ℓ, denoted $\Delta|\ell$, amounts to removing literal $\neg\ell$ from all clauses and then dropping all clauses that contain literal ℓ.

Upper case letters (e.g., X) will denote variables and lower case letters (e.g., x) will denote their instantiations. Bold upper case letters (e.g., $\mathbf{X}$) will denote sets of variables and bold lower case letters (e.g., $\mathbf{x}$) will denote their instantiations. An instantiation $\mathbf{x}$ of variables $\mathbf{X}$ will be interpreted as a *term* (conjunction of literals), or as a CNF (set of clauses, where each clause corresponds to a literal of $\mathbf{x}$).

3 Decomposing CNFs

Consider a vtree with root v. Let $\mathbf{X}$ be the variables of left child v^l and let $\mathbf{Y}$ be the variables of right child v^r. To compile a CNF Δ into a DNNF that respects this vtree, we will first decompose Δ into CNFs (called components) that only mention variables $\mathbf{X}$ or only mention variables $\mathbf{Y}$. These components are then decomposed with respect to the vtrees rooted at v^l and v^r. The process continues recursively until we reach literals or constants. The following definition provides the basis for this recursive decomposition process.

Definition 1 ([9]). *Consider a CNF $\Delta(\mathbf{X}, \mathbf{Y})$ where variables $\mathbf{X}$ and $\mathbf{Y}$ are disjoint. An $\underline{(\mathbf{X}, \mathbf{Y})\text{-decomposition}}$ of Δ is a set*

$$\Big\{\Big(L_1(\mathbf{X}), R_1(\mathbf{Y})\Big), \ldots, \Big(L_n(\mathbf{X}), R_n(\mathbf{Y})\Big)\Big\}$$

such that L_i and R_i are CNFs and Δ is equivalent to $(L_1 \wedge R_1) \vee \ldots \vee (L_n \wedge R_n)$. Each pair (L_i, R_i) is called an element, where L_i is called an $\mathbf{X}$-component and R_i is called a $\mathbf{Y}$-component.

Consider the CNF $\Delta = \{a \vee \neg b \vee \neg c,\ \neg a \vee b \vee c\}$ and let $\mathbf{X} = \{A, B\}$ and $\mathbf{Y} = \{C\}$. The following is then an $(\mathbf{X}, \mathbf{Y})$-decomposition of Δ, which has three elements:

$$\{(\{a \vee \neg b, \neg a \vee b\}, \{\}),\ (\{a \vee \neg b\}, \{c\}),\ (\{\neg a \vee b\}, \{\neg c\})\}.$$

4 Constructing Decompositions

We will now review two systematic methods for constructing $(\mathbf{X}, \mathbf{Y})$-decompositions. The first method is based on *variable splitting* [2] and the second one is based on *clause splitting* [10].

4.1 Decomposition by Splitting on Variables

To split on variables $\mathbf{V}$ is to consider all possible instantiations $\mathbf{v}$ of these variables. Here, each instantiation $\mathbf{v}$ corresponds to a set of literals, exactly one literal for each variable in $\mathbf{V}$. Hence, if $\mathbf{V}$ contains n variables, then splitting on variables $\mathbf{V}$ leads to 2^n cases.

Consider now a CNF Δ over disjoint variables $\mathbf{X}$ and $\mathbf{Y}$. Suppose further that the CNF is partitioned into $\Delta(\mathbf{X})$, $\Delta(\mathbf{Y})$ and $\Delta(\mathbf{X}, \mathbf{Y})$, where $\Delta(\mathbf{X})$ contains all clauses of Δ that only mention variables $\mathbf{X}$ and $\Delta(\mathbf{Y})$ contains all clauses of Δ that mention only variables $\mathbf{Y}$. Let $\mathbf{V}$ be all variables in $\mathbf{X}$ that are mentioned in $\Delta(\mathbf{X}, \mathbf{Y})$. The following is then an $(\mathbf{X}, \mathbf{Y})$-decomposition of CNF Δ [2]:

$$\Big\{\Big(\mathbf{v} \cup \Delta(\mathbf{X})|\mathbf{v},\ \Delta(\mathbf{Y}) \cup \Delta(\mathbf{X}, \mathbf{Y})|\mathbf{v}\Big)\ \Big|\ \mathbf{v} \text{ an instantiation of } \mathbf{V}\Big\}.$$

This implies that

$$\Delta = \bigvee_{\mathbf{v}} \Big(\mathbf{v}\Big) \wedge \Big(\Delta|\mathbf{v}\Big)$$

since $\Delta|\mathbf{v} = \Delta(\mathbf{X})|\mathbf{v} \cup \Delta(\mathbf{Y}) \cup \Delta(\mathbf{X}, \mathbf{Y})|\mathbf{v}$. The $\mathbf{X}$-components and the $\mathbf{Y}$-components of the above decomposition are all CNFs. Moreover, when the set $\mathbf{V}$ contains a single variable V, the above decomposition corresponds to the Shannon decomposition of Δ, which is defined as $\Delta = (v \wedge \Delta|v) \vee (\neg v \wedge \Delta|\neg v)$.

4.2 Decomposition by Splitting on Clauses

Another method for constructing $(\mathbf{X}, \mathbf{Y})$-decompositions is by splitting on clauses. That is, each clause γ is split into two sub-clauses α and β, where α mentions only variables in $\mathbf{X}$ and β mentions only variables in $\mathbf{Y}$. We then take the Cartesian product of these sub-clauses. This is formalized next.

Definition 2 (Clausal Decomposition [10]). *Consider a CNF $\Delta = \{\gamma_1, \ldots, \gamma_k\}$ over disjoint variables $\mathbf{X}$ and $\mathbf{Y}$, where each clause has variables in $\mathbf{X}$ and in $\mathbf{Y}$. Let $\gamma_i = \alpha_i \vee \beta_i$, where α_i and β_i are the sub-clauses of γ_i mentioning variables $\mathbf{X}$ and $\mathbf{Y}$, respectively. The $\underline{\text{clausal } (\mathbf{X}, \mathbf{Y})\text{-decomposition}}$ of CNF Δ is defined as*

$$CD(\Delta, \mathbf{X}, \mathbf{Y}) = \Big\{\Big(\bigcup_{i \in S} \alpha_i, \bigcup_{j \notin S} \beta_j\Big)\ \Big|\ S \subseteq \{1, \ldots, k\}\Big\}.$$

This clausal decomposition allows us to write CNF Δ as follows

$$\Delta = \bigvee_{S \subseteq \{1, \ldots, k\}} \Big(\bigwedge_{i \in S} \alpha_i\Big) \wedge \Big(\bigwedge_{j \notin S} \beta_j\Big).$$

More generally, consider a CNF Δ over disjoint variables $\mathbf{X}$ and $\mathbf{Y}$, and suppose that the CNF is partitioned into $\Delta(\mathbf{X})$, $\Delta(\mathbf{Y})$ and $\Delta(\mathbf{X}, \mathbf{Y})$. Suppose further that $\{(L_1, R_1), \ldots, (L_n, R_n)\}$ is the clausal $(\mathbf{X}, \mathbf{Y})$-decomposition of CNF $\Delta(\mathbf{X}, \mathbf{Y})$. The following is then guaranteed to be an $(\mathbf{X}, \mathbf{Y})$-decomposition of CNF Δ:

$$\Big\{\Big(\Delta(\mathbf{X}) \cup L_1,\ \Delta(\mathbf{Y}) \cup R_1\Big), \ldots, \Big(\Delta(\mathbf{X}) \cup L_n,\ \Delta(\mathbf{Y}) \cup R_n\Big)\Big\}.$$

The $\mathbf{X}$-components of this decomposition have the form $\Delta(\mathbf{X}) \cup L_i$, where L_i is an $\mathbf{X}$-component of the clausal decomposition for $\Delta(\mathbf{X}, \mathbf{Y})$. As we shall see later, the number of these components will play a major role in defining our new notion of width.

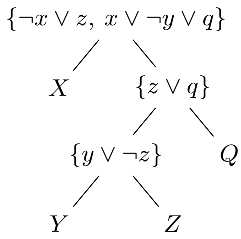


Figure 2. Distributing the clauses of $\{y \vee \neg z,\ z \vee q,\ \neg x \vee z,\ x \vee \neg y \vee q\}$ on a vtree. Internal nodes show assigned clauses.

5 More on Vtrees

Before discussing our compilation algorithm, we will introduce some definitions about vtrees that will be used later.

A vtree node v is called a *Shannon node* iff its left child is a leaf. In this case, the variable labeling the left child is called the *Shannon variable* of node v. In Figure 1(a), vtree nodes 1 and 3 are Shannon nodes, with X and Y as their Shannon variables. A vtree is said to be *right-linear* if every internal node is a Shannon node. Figure 4 shows a right-linear vtree.

Let π be a variable ordering. The right-linear vtree induced by π is the one whose in-order traversal visits leaves in the same order as π. Figure 4 shows the right linear vtree induced by order $X, Y_1, \ldots, Y_n$.

We will find it useful to distribute the clauses of a CNF Δ on a vtree as follows. Each clause of Δ is assigned to the lowest vtree node that contains the clause variables. Figure 2 depicts an example of how clauses are assigned to vtree nodes. We use $Clauses(v)$ to denote the clauses assigned to vtree node v. We also use $CNF(v)$ to denote the clauses assigned to all nodes in the vtree rooted at v.

6 Compiling CNFs into Structured DNNF

We will now present an algorithm that compiles a CNF into a DNNF that respects a given vtree. Our compilation method is given by Algorithm 1, which takes a vtree v and an auxiliary CNF S over the variables of vtree v (S is initially empty). The CNF Δ to be compiled is passed with the vtree as explained earlier.

The following theorem establishes the soundness of the algorithm. Its proof is inductive and follows from the soundness of the decomposition techniques based on variable and clause splitting.

Theorem 1. *The call* $\mathtt{c2s}(v, \{\})$ *to Algorithm 1 returns a DNNF that respects vtree* v *and that is equivalent to* $CNF(v)$.

More generally, a recursive call $\mathtt{c2s}(v, S)$ will return a DNNF for $CNF(v) \cup S$ that respects vtree v. Moreover, depending on the type of vtree node, the algorithm will either split on a single variable to compute a Shannon decomposition (Lines 4–13), or will split on clauses to compute a clausal decomposition (Lines 14–20). The algorithm keeps a cache at every vtree node, which is indexed by the auxiliary CNF S.

Algorithm 1 returns an OBDD when the input vtree is right-linear. Since every internal vtree node is a Shannon node, Lines 4–13 will always be invoked to construct a Shannon decomposition. This essentially creates an OBDD which respects the variable order underlying the right-linear vtree. The resulting OBDD is not reduced, however, but this can be addressed by incorporating a *unique-node table* into Algorithm 1, which does not change its complexity [7].

Algorithm 1: $\mathtt{c2s}(v, S)$
$cache(v, \Delta)$ is a hash table that maps v and Δ into a DNNF.
$terminal(\Delta)$ returns the literal or constant equivalent to Δ.

Input: v : a vtree node, S : a CNF over $Vars(v)$.
Output: A DNNF for $CNF(v) \cup S$ that respects vtree v.
1 **if** $cache(v, S) \neq nil$ **then return** $cache(v, S)$
2 $C \leftarrow Clauses(v)$
3 **if** *v is a leaf* **then return** $terminal(C \cup S)$
4 **if** *v is a Shannon node* **then**
5     $X \leftarrow$ Shannon variable of v
6   **if** $\{x\}$ *and* $\{\neg x\}$ *assigned to* v^l **then** $\alpha \leftarrow \bot$
7   **else if** $\{x\}$ *assigned to* v^l **then**
8         $\alpha \leftarrow x \wedge \mathtt{c2s}(v^r, (C \cup S)|x)$
9   **else if** $\{\neg x\}$ *assigned to* v^l **then**
10         $\alpha \leftarrow \neg x \wedge \mathtt{c2s}(v^r, (C \cup S)|\neg x)$
11   **else**
12         $\alpha \leftarrow \Big(x \wedge \mathtt{c2s}(v^r, (C \cup S)|x)\Big) \vee$
13             $\Big(\neg x \wedge \mathtt{c2s}(v^r, (C \cup S)|\neg x)\Big)$
14 **else**
15     $\mathbf{X} \leftarrow$ variables in the vtree rooted at v^l
16     $\mathbf{Y} \leftarrow$ variables in the vtree rooted at v^r
17   Partition S into $S_1(\mathbf{X})$, $S_2(\mathbf{Y})$, and $S_3(\mathbf{X}, \mathbf{Y})$
18     $\alpha \leftarrow \bot$
19   **foreach** $(L, R) \in CD(C \cup S_3, \mathbf{X}, \mathbf{Y})$ **do**
20         $\alpha \leftarrow \alpha \vee \Big(\mathtt{c2s}(v^l, S_1 \cup L) \wedge \mathtt{c2s}(v^r, S_2 \cup R)\Big)$
21 $cache(v, S) \leftarrow \alpha$
22 **return** α

7 A New Complexity Parameter for CNFs

In this section, we will introduce CV-width, and show that the time and space complexity of Algorithm 1 is exponential only in CV-width. First, we will study a concept that will be quite useful in defining CV-width.

7.1 Counting Components

Our new notion of width and the corresponding complexity analysis of our compilation algorithm depend crucially on counting the number of distinct components of clausal decompositions. The following direct definition of these components facilitates this process.

Definition 3. *Consider a CNF* Δ *and variables* $\mathbf{X}$. *Let* $\gamma_1, \ldots, \gamma_n$ *be the clauses in* Δ *which mention variables inside and outside* $\mathbf{X}$, *and let* α_i *be the sub-clause of* γ_i *with variables in* $\mathbf{X}$. *The* $\mathbf{X}$*-components of* Δ *are defined as the following CNFs*

$$CNFs(\Delta, \mathbf{X}) = \{\Delta(\mathbf{X}) \cup \Gamma \mid \Gamma \subseteq \{\alpha_1, \ldots, \alpha_n\}\}$$

where $\Delta(\mathbf{X})$ *is the set of clauses of* Δ *that only mention variables* $\mathbf{X}$.

For example, if $\Delta = \{x_1,\, x_2 \vee z,\, x_3 \vee \neg z\}$ and $\mathbf{X} = \{X_1, X_2, X_3\}$, then $CNFs(\Delta, \mathbf{X}) = \{\{x_1\}, \{x_1, x_2\}, \{x_1, x_3\}, \{x_1, x_2, x_3\}\}$.

Suppose that we split on variables $\mathbf{V}$, leading to CNFs $\Delta|\mathbf{v}$: one CNF for each instantiation $\mathbf{v}$ of variables $\mathbf{V}$. Suppose that we further construct a clausal decomposition for each CNF $\Delta|\mathbf{v}$. We will find it quite useful to count the number of distinct components which are obtained from this process.

Definition 4. *Consider a CNF Δ and disjoint variables $\mathbf{X}$ and $\mathbf{V}$. The $\mathbf{X}|\mathbf{V}$-components of Δ are defined as the following CNFs*

$$CNFs(\Delta, \mathbf{X}|\mathbf{V}) = \bigcup_{\mathbf{v}} CNFs(\Delta|\mathbf{v}, \mathbf{X}).$$

Consider the CNF

$$\Delta = \{x_1 \vee v \vee z,\ x_2 \vee \neg x_3 \vee v,\ x_2 \vee \neg v \vee z,\ x_3 \vee \neg v \vee z\}.$$

If $\mathbf{X} = \{X_1, X_2, X_3\}$ and $\mathbf{V} = \{V\}$, then

$$\begin{aligned} \Delta|v &= \{x_2 \vee z,\ x_3 \vee z\}, \\ CNFs(\Delta|v, \mathbf{X}) &= \{\{\},\ \{x_2\},\ \{x_3\},\ \{x_2, x_3\}\}, \\ \Delta|\neg v &= \{x_1 \vee z,\ x_2 \vee \neg x_3\}, \\ CNFs(\Delta|\neg v, \mathbf{X}) &= \{\{x_2 \vee \neg x_3\},\ \{x_1,\ x_2 \vee \neg x_3\}\}. \end{aligned}$$

Hence,

$$CNFs(\Delta, \mathbf{X}|\mathbf{V}) = \{\{\},\ \{x_2\},\ \{x_3\},\ \{x_2, x_3\},\ \{x_2 \vee \neg x_3\},\ \{x_1, x_2 \vee \neg x_3\}\}.$$

These are all the distinct $\mathbf{X}$-components obtained by first splitting on variables $\mathbf{V}$, then constructing clausal decompositions.

We will use $\#CNFs(\Delta, \mathbf{X}|\mathbf{V})$ to denote the ceiling of $\log(|CNFs(\Delta, \mathbf{X}|\mathbf{V})|)$, where $\log 0$ is defined as 0. Hence, in the above example $\#CNFs(\Delta, \mathbf{X}|\mathbf{V}) = 3$.

7.2 Clause-Variable Width

We are now ready to introduce the new notion of width, called CV-width. This new width is based on counting the number of distinct components that arise when decomposing a CNF using a series of splits on variables and clauses.

CV-width is defined for a vtree and a corresponding CNF. The CV-width of a CNF is then defined as the smallest CV-width attained by any of its vtrees. To define CV-width for a given vtree, we need to associate a set of clauses and variables with each internal node in the vtree. These sets are defined next.

Definition 5. *Consider a CNF Δ and a corresponding vtree. Each internal vtree node v is associated with the following sets:*

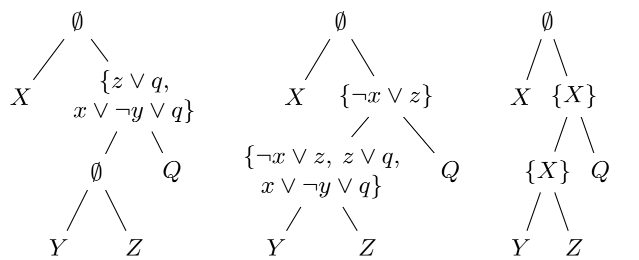
– *Context Variables: Shannon variables of v's ancestors.*
– *Cutset Clauses: empty set if v is a Shannon node; otherwise, clauses with variables inside v^l and inside v^r.*
– *Context Clauses: clauses with variables inside and outside v, and that do not belong to the cutset.*

Figure 3 depicts a CNF, a corresponding vtree and the associated cutset clauses, context clauses, and context variables of vtree nodes.

When Algorithm 1 is decomposing a CNF with respect to a vtree node v, it would have already split on its context variables. At this point, the CNF can be decomposed by splitting on its cutset and context clauses. One will always split on cutset clauses. However, whether one would need to split on a particular context clause depends on the specific splits adopted at ancestors. This motivates the following definition of width.

Definition 6 (CV-width)**.** *Consider a CNF and a corresponding vtree. Let v be an internal vtree node with variables $\mathbf{X}$, context variables $\mathbf{V}$, cutset clauses Δ and context clauses Γ. The width of node v, $width(v)$, is $|\Delta| + \#CNFs(\Gamma, \mathbf{X}|\mathbf{V})$. The CV-width of the vtree is the largest width of any of its internal nodes minus 1. The CV-width of a CNF is the smallest CV-width attained by any of its vtrees.*

Consider the CNF $\{y \vee \neg z,\ z \vee q,\ \neg x \vee z,\ x \vee \neg y \vee q\}$ and the vtree in Figure 1(a). The CV-width of this vtree is 2; see Figure 3.



(a) Cutset Clauses (b) Context Clauses (c) Context Vars.

Figure 3. Cutset clauses, context clauses, and context variables of the vtree in Figure 1(a), defined for the CNF $\{y \vee \neg z,\ z \vee q,\ \neg x \vee z,\ x \vee \neg y \vee q\}$.

7.3 Complexity Analysis

The following theorem reveals the time and space complexity of our compilation algorithm (the proof is delegated to the Appendix).

Theorem 2. *If vtree v is over n variables and has CV-width w, and if $CNF(v)$ has size m, then the call $\texttt{c2s}(v, \{\})$ to Algorithm 1 takes time in $O(nm3^w)$ and returns a DNNF whose size is in $O(n3^w)$.*

We know that Algorithm 1 is guaranteed to return an OBDD when the input vtree is right-linear. In this case, we need to state the complexity of the algorithm by using a restricted version of CV-width, which is defined for right-linear vtrees.

Definition 7. *The linear CV-width of a CNF is the smallest CV-width attained by any right-linear vtree of the CNF.*

Therefore, if a CNF has n variables and has a linear CV-width w, it must have an OBDD whose size is in $O(n3^w)$. In fact, a simple argument can show that the size is actually in $O(n2^w)$.

8 Relationship to Classical CNF Parameters

We now compare CV-width to some classical parameters that characterize the structural properties of CNFs. We consider three parameters: treewidth, cutwidth and pathwidth. The first parameter is a property of some graph abstraction of the CNF, such as primal, dual and incidence graphs, and has been used to bound the size of DNNF compilations. The last two parameters apply directly to a CNF and have been used to bound the size of OBDD compilations.

The *primal graph* of a CNF is obtained by treating CNF variables as graph nodes, while adding an edge between two variables iff they appear in the same clause. The *dual graph* is obtained by treating CNF clauses as graph nodes, while adding an edge between two clauses iff they share a common variable. The *incidence graph* is obtained by treating CNF variables and clauses as graph nodes, while adding an edge between a variable and a clause iff the variable appears in the clause.

We will use twp, twd and twi to denote the treewidth of primal, dual and incidence graphs, respectively. It is known that twp and twd are incomparable, in the sense that there are classes of CNFs for which one can be bounded while the other is unbounded. Moreover, it has been shown that $twi \leq twp + 1$ and $twi \leq twd + 1$ [6]. We will next show that CV-width dominates twi, which immediately implies that it also dominates twp and twd.

Theorem 3. *Let Δ be a CNF whose incidence graph has treewidth w. We can construct a vtree for this CNF whose CV-width $\leq w$.*

The following theorem shows that the incidence graph of a CNF may have an unbounded treewidth, yet its CV-width may be bounded.

Theorem 4. *There is a class of CNFs Δ_n, with n variables and n clauses, $n \geq 1$, whose incidence graph has treewidth $\geq n/2 - 2$, yet whose CV-width is 0.*

Proof (Sketch)**.** $\Delta_n = \{C_1, \ldots, C_n\}$, where $C_i = x_1 \vee \ldots \vee x_i$. The incidence graph of Δ_n has treewidth $\geq n/2 - 2$ (proof in full paper). Consider the right-linear vtree induced by the variable ordering $X_1, \ldots, X_n$. Consider a vtree node v whose left child is X_i. Since v is a Shannon node, its cutset is empty. Let Γ be the context clauses of v. If $i = 1$, then Γ is empty and the width of v is 0. Otherwise, $\Gamma = \{C_i, \ldots, C_n\}$. Let $\mathbf{X}$ be the variables inside v, and let $\mathbf{V}$ be the context variables of v. Then, $CNFs(\Gamma, \mathbf{X}|\mathbf{V}) = \{\{\}, \{x_i,\, x_i \vee x_{i+1}, \ldots, x_i \vee \ldots \vee x_n\}\}$. The width of v is then 1. The CV-width of the vtree is then 0. ■

We now turn our attention to cutwidth and pathwidth, which have been used to bound the complexity of OBDDs obtained from CNFs [5]. These parameters will be compared to linear CV-width. We want to remark again that Algorithm 1 constructs an OBDD when the input vtree is right-linear.

Cutwidth and pathwidth are incomparable. We will show next that linear CV-width dominates both and can be bounded when neither cutwidth or pathwidth are bounded. We start, however, by the definitions of cutwidth and pathwidth based on [5].

Definition 8. *Let $\pi = V_1, \ldots, V_n$ be an ordering of the variables in CNF Δ. The i^{th} cutset of order π is the set of clauses in Δ that mentions a variable V_j, $j \leq i$, and a variable V_k, $k > i$. The cutwidth of order π is the size of its largest cutset. The <u>cutwidth</u> of CNF Δ is the smallest cutwidth attained by any variable ordering π.*

Definition 9. *Let $\pi = V_1, \ldots, V_n$ be an ordering of the variables in CNF Δ. The i^{th} separator of order π is the set of variables V_j, $j \leq i$, that appear in the i^{th} cutset of order π. The pathwidth of order π is the size of its largest separator. The <u>pathwidth</u> of CNF Δ is the smallest pathwidth attained by any variable ordering π.*

The following theorem implies that linear CV-width dominates both cutwidth and pathwidth.

Theorem 5. *Let π be an ordering of the variables in CNF Δ, where π has cutwidth cw and pathwidth pw. Let w be the CV-width of the right-linear vtree induced by order π. Then, $w < cw$ and $w < pw$.*

Proof. Consider the right-linear vtree induced by π. Let v be an internal vtree node with variables $\mathbf{X}$, context clauses Γ, and context variables $\mathbf{V}$. It suffices to show that $width(v) \leq cw$ and $width(v) \leq pw$. Node v must be a Shannon node. Thus, its cutset is empty and $width(v)$ is $\#CNFs(\Gamma, \mathbf{X}|\mathbf{V})$. Assume that $\pi = V_1, \ldots, V_n$ and that v^l is labeled with variable V_{i+1}. The variables outside v are then $\{V_1, \ldots, V_i\}$ and the ones inside v are $\{V_{i+1}, \ldots, V_n\}$. Thus, Γ is the i^{th} cutset of order π. Since Γ only mentions variables $\mathbf{X}$ and $\mathbf{V}$, $CNFs(\Gamma, \mathbf{X}|\mathbf{V})$ is the distinct CNFs $\Gamma|\mathbf{v}$. Hence, $|CNFs(\Gamma, \mathbf{X}|\mathbf{V})| \leq 2^{|\Gamma|}$, leading to $\#CNFs(\Gamma, \mathbf{X}|\mathbf{V}) \leq |\Gamma|$ and so $width(v) \leq cw$. Moreover, $Vars(\Gamma) \cap \mathbf{V}$ is the i^{th} separator of order π. Since $|CNFs(\Gamma, \mathbf{X}|\mathbf{V})| \leq 2^{|Vars(\Gamma) \cap \mathbf{V}|}$, we have $\#CNFs(\Gamma, \mathbf{X}|\mathbf{V}) \leq |Vars(\Gamma) \cap \mathbf{V}|$ and $width(v) \leq pw$. So, $w < cw$ and $w < pw$. ■

We now know that linear CV-width dominates both cutwidth and pathwidth. The following theorem shows that these widths can be unbounded when linear CV-width is bounded.

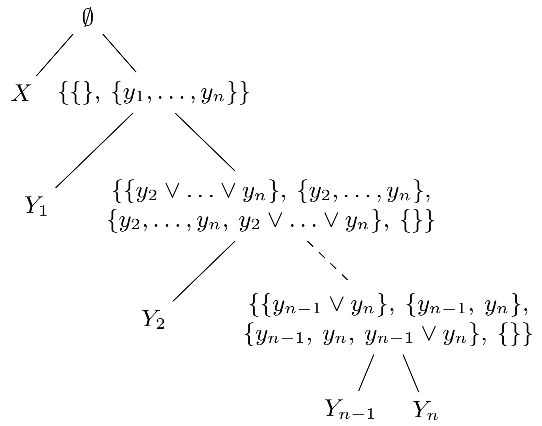


Figure 4. A right-linear vtree induced by order $X, Y_1, \ldots, Y_n$. Nodes v show $CNFs(\Gamma, \mathbf{X}|\mathbf{V})$ wrt $\Delta_n = \{x \vee y_1, \ldots, x \vee y_n,\, y_1 \vee \ldots \vee y_n\}$, where Γ is context clauses, $\mathbf{X}$ is variables, and $\mathbf{V}$ is context variables of v.

Theorem 6. *There is a class of CNFs Δ_n, with $n+1$ variables and $n+1$ clauses, $n \geq 1$, whose cutwidth is $\geq n/2 - 1$, pathwidth is $\geq n - 2$, yet whose linear CV-width is ≤ 1.*

Proof (Sketch)**.** $\Delta_n = \{x \vee y_1, \ldots, x \vee y_n, y_1 \vee \ldots \vee y_n\}$. Consider the variable ordering $\pi = X, Y_1, \ldots, Y_n$. Figure 4 shows the right-linear vtree induced by π. According to this figure, the CV-width of this vtree is 1 and the linear CV-width of CNF Δ_n is ≤ 1. Consider now an arbitrary variable ordering π' for Δ_n. The size of the $(n-1)^{th}$ separator of this order must be $\geq n - 2$. To see this, note that the last two variables in order π' cannot both be X. So, due to clause $\{y_1 \vee \ldots \vee y_n\}$, the $(n-1)^{th}$ separator must contain at least $n - 2$ variables. Thus, the pathwidth of Δ_n is $\geq n - 2$ for any order π'. One can also show that the i^{th} cutset of order π' is $\geq n/2 - 1$ for some i that depends on the position of variable X in the order. Thus, the cutwidth of Δ_n is $\geq n/2 - 1$ for any order π'. ■

9 Related Work

Two algorithms for compiling structured DNNFs were given in [10]. One algorithm splits on variables and the other one splits on clauses. The latter has a time and space complexity that is exponential in the treewidth of the CNF dual graph, and the former has a time and space complexity that is exponential in the treewidth of the CNF primal graph. The compilation algorithm we proposed in this paper splits on both variables and clauses. One would have expected that this combination will lead to a complexity that is a minimum of the two complexities attained by the mentioned algorithms. Interestingly though, the combination leads to a more significant improvement. In particular, our algorithm has a time and space complexity that is exponential in CV-width, which we showed to strictly dominate the treewidth of the CNF incidence graph. Moreover, it is already known that this treewidth dominates the ones for the CNF primal and dual graphs.

An algorithm for compiling OBDDs was also presented in [5]. The complexity of the algorithm is exponential in the cutwidth or the pathwidth of input CNF. Our algorithm is exponential in the linear CV-width of the CNF. Since linear CV-width strictly dominates both cutwidth and pathwidth, our upper bound significantly improves on the ones given in [5].

Another bound was recently shown for DNNFs compiled from CNFs [11]. Given a CNF with n variables, size m, and an incidence graph with treewidth w, this bound shows that the DNNF size is in $O((n+m)3^w)$. Our results improve on this bound in two fundamental ways. First, our bound applies to structured DNNF, which is a subset of DNNF that supports a polytime conjoin operation (not supported by unstructured DNNF). Second, our bound is based on CV-width, which strictly dominates the treewidth of the incidence graph. Hence, our bound significantly improves on the existing bound for DNNFs, even when unstructured. Finally, our size upper bound is linear in the number of variables, whereas the existing upper bound is linear in the number of variables plus the size of the CNF (which can be much larger than the number of variables).

10 Conclusion

We presented new complexity results on the compilation of CNFs into DNNFs and OBDDs. In particular, we introduced a new notion of width, called CV-width, which is specific to CNFs and that dominates the treewidth of the CNF incidence graph. We then showed that CNFs can be compiled into structured DNNFs in time and space that are exponential only in CV-width. Not only does CV-width dominate the incidence graph treewidth, but the former width can be bounded when the latter is unbounded. We also introduced a restricted version of CV-width, called linear CV-width, and showed that it dominates both pathwidth and cutwidth, which have been used to bound the complexity of OBDDs. We also showed that CNFs can be compiled into OBDDs in time and space that are exponential only in linear CV-width. We finally showed that linear CV-width can be bounded when pathwidth and cutwidth are unbounded. Our results significantly improved the previously known best upper bounds for both DNNFs and OBDDs, and are motived by a novel decomposition technique that combines variable and clause splitting.

ACKNOWLEDGEMENTS

This work has been partially supported by ONR grant #N00014-12-1-0423 and NSF grant #IIS-1118122.

A Additional Proofs

We will now prove the complexity of Algorithm 1. This requires the following lemma. For CNF Σ, we will use $\Sigma{\downarrow}\mathbf{X}$ to denote the CNF which results from replacing every clause in Σ by its sub-clause that mentions variables in $\mathbf{X}$. For example, if $\Sigma = \{a \vee \neg b \vee c,\ \neg a \vee c \vee \neg d\}$ and $\mathbf{X} = \{A, B\}$, then $\Sigma{\downarrow}\mathbf{X} = \{a \vee \neg b,\ \neg a\}$.

Lemma 1. *Let v be an internal vtree node with variables $\mathbf{X}$, cutset clauses Δ, context clauses Γ and context variables $\mathbf{V}$. The following hold when Algorithm 1 starts executing a call* $\texttt{c2s}(v, S)$*:*

If v is a Shannon node, then

(a) $S \in CNFs(\Gamma, \mathbf{X}|\mathbf{V})$

If v is not a Shannon node, then

(b) $C \subseteq \Delta$

(c) $S_1 \cup S_2 \in CNFs(\Gamma, \mathbf{X}|\mathbf{V})$

(d) $S_3 \subseteq \Sigma{\downarrow}\mathbf{X}$ where $\Sigma = \Delta \setminus C$

We next prove Theorem 2.

Proof (Theorem 2). Let v be an internal vtree node with variables $\mathbf{X}$, cutset clauses Δ, context clauses Γ, and context variables $\mathbf{V}$. We will next bound the time spent at node v and the contribution it makes to the DNNF size during all calls made to node v. By adding these time and size bounds for all internal vtree nodes, we can bound the time and space complexity of Algorithm 1.

Assume that v is a Shannon node. By Lemma 1(a), $S \in CNFs(\Gamma, \mathbf{X}|\mathbf{V})$. Hence, the number of uncached calls to v is $\leq 2^{|\Delta| + \#CNFs(\Gamma,\mathbf{X}|\mathbf{V})}$ since $\Delta = \emptyset$ for a Shannon node. Moreover, each uncached call to v will construct a decomposition of size at most 2 by doing $O(2m)$ work (Lines 4–13). The total contribution of a Shannon node to time complexity is then $O(m2^{width(v)})$. Moreover, the total contribution it makes to the DNNF size is $O(2^{width(v)})$.

Assume now that v is not a Shannon node. The following observations all follow from Lemma 1. First, by Lemma 1(d), if $|S_3| = i$ and $|\Sigma| = k$, then $0 \leq i \leq k$. Moreover, there are at most $\binom{k}{i}$ distinct CNFs S_3 of size i. Second, by Lemma 1(c), there are at most $2^{\#CNFs(\Gamma,\mathbf{X}|\mathbf{V})}\binom{k}{i}$ uncached calls to node v for which $|S_3| = i$. Moreover, each of these calls will construct a clausal decomposition of size $2^{|C|+i}$ on Line 20. Hence, the decompositions constructed at Line 20 will have a total size of

$$
\begin{aligned}
&\sum_{i=0}^{k} 2^{\#CNFs(\Gamma,\mathbf{X}|\mathbf{V})}\binom{k}{i}2^{|C|+i} \\
&= 2^{\#CNFs(\Gamma,\mathbf{X}|\mathbf{V})+|C|}\sum_{i=0}^{k}\binom{k}{i}2^{i} \\
&= 2^{\#CNFs(\Gamma,\mathbf{X}|\mathbf{V})+|C|}3^{k} \\
&\leq 3^{\#CNFs(\Gamma,\mathbf{X}|\mathbf{V})+|C|+k} \\
&= 3^{\#CNFs(\Gamma,\mathbf{X}|\mathbf{V})+|\Delta|} \qquad \text{by Lemma 1(b)} \\
&= 3^{width(v)}.
\end{aligned}
$$

Computing a clausal decomposition is linear in the CNF size. Hence, the total contribution of node v to time complexity is $O(m3^{width(v)})$. Moreover, the total contribution it makes to the DNNF size is $O(3^{width(v)})$. As there are $O(n)$ vtree nodes, Algorithm 1 has a total time complexity in $O(nm3^w)$. Moreover, the structured DNNF it constructs has size in $O(n3^w)$. ∎

REFERENCES

[1] Randal E. Bryant, 'Graph-Based Algorithms for Boolean Function Manipulation', *IEEE Trans. Computers*, **35**(8), 677–691, (1986).

[2] Adnan Darwiche, 'Decomposable Negation Normal Form', *J. ACM*, **48**(4), 608–647, (2001).

[3] Adnan Darwiche, 'SDD: A New Canonical Representation of Propositional Knowledge Bases', in *IJCAI*, pp. 819–826, (2011).

[4] Adnan Darwiche and Pierre Marquis, 'A Knowledge Compilation Map', *J. Artif. Intell. Res. (JAIR)*, **17**, 229–264, (2002).

[5] Jinbo Huang and Adnan Darwiche, 'Using DPLL for Efficient OBDD Construction', in *SAT*, (2004).

[6] Phokion G. Kolaitis and Moshe Y. Vardi, 'Conjunctive-Query Containment and Constraint Satisfaction', *J. Comput. Syst. Sci.*, **61**(2), 302–332, (2000).

[7] Christoph Meinel and Thorsten Theobald, *Algorithms and Data Structures in VLSI Design: OBDD - Foundations and Applications*, Springer, 1998.

[8] Knot Pipatsrisawat and Adnan Darwiche, 'New Compilation Languages Based on Structured Decomposability', in *AAAI*, (2008).

[9] Knot Pipatsrisawat and Adnan Darwiche, 'A Lower Bound on the Size of Decomposable Negation Normal Form', in *AAAI*, (2010).

[10] Knot Pipatsrisawat and Adnan Darwiche, 'Top-Down Algorithms for Constructing Structured DNNF: Theoretical and Practical Implications', in *ECAI*, pp. 3–8, (2010).

[11] Igor Razgon and Justyna Petke, 'Cliquewidth and Knowledge Compilation', in *SAT*, pp. 335–350, (2013).

[12] Neil Robertson and Paul D. Seymour, 'Graph minors. III. Planar tree-width', *J. Comb. Theory, Ser. B*, **36**(1), 49–64, (1984).