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 motivated 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 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 Δ\{V\} and Δ\{|¬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 α ∨ β from a CNF Δ by considering the CNFs Δ \{α\} and Δ \{β\}. 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) if 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 Z of variables is a rooted, full binary tree whose leaves are in one-to-one correspondence with variables in Z. Fig-

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

Consider a vtree with root $v$ and a set of clauses, where each clause is a disjunction of literals (e.g., \{\lor \neg \ell \mid \ell \in Vars(v)\} 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 were 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., \{\lor \neg y \lor \neg z, \lor \neg y \lor z\}). We will often write $\Delta(X)$ to mean that CNF $\Delta$ mentions only variables in $X$. Conditioning a CNF $\Delta$ on a literal $\ell$, denoted $\Delta(\ell)$, amounts to removing literal $\ell$ from all clauses and then dropping all clauses that contain literal $\ell$.

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

### 3 Decomposing CNFs

Consider a vtree with root $v$. Let $X$ be the variables of left child $v'$ and let $Y$ be the variables of right child $v''$. To compile a CNF $\Delta$ into a DNNF that respects this vtree, we will first decompose $\Delta$ into CNFs (called components) that only mention variables $X$ or only mention variables $Y$. These components are then decomposed with respect to the vtrees rooted at $v'$ and $v''$. 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 $\Delta(X, Y)$ where variables $X$ and $Y$ are disjoint. An $(X, Y)$-decomposition of $\Delta$ is a set

\[
\left\{ \left( L_1(X), R_1(Y) \right), \ldots, \left( L_n(X), R_n(Y) \right) \right\}
\]

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

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

\[
\left\{ \{a \lor \neg b, \neg a \lor b\}, \{a \lor b\}, \{\neg a \lor b, \neg c\} \right\}
\]

### 4 Constructing Decompositions

We will now review two systematic methods for constructing $(X, Y)$-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 $V$ is to consider all possible instantiations $v$ of these variables. Here, each instantiation $v$ corresponds to a set of literals, exactly one literal for each variable in $V$. Hence, if $V$ contains $n$ variables, then splitting on variables $V$ leads to $2^n$ cases.

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

\[
\left\{ \left( \{v \lor \Delta(X)/v\}, \{\Delta(Y) \cup \Delta(X, Y)/v\} \right) \mid v \text{ an instantiation of } V \right\}
\]

This implies that

\[
\Delta = \bigvee \{ \{v \lor \Delta(v)\} \land (\Delta/\neg v) \mid v \text{ an instantiation of } V \}
\]

since $\Delta/\neg v = \Delta(X)/v \lor \Delta(Y) \lor \Delta(X, Y)/v$. The $X$-components and the $Y$-components of the above decomposition are all CNFs. Moreover, when the set $V$ contains a single variable $v$, the above decomposition corresponds to the Shannon decomposition of $\Delta$, which is defined as $\Delta = (v \lor \Delta(v)) \lor (\neg v \lor \Delta(\neg v))$.

#### 4.2 Decomposition by Splitting on Clauses

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

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

\[
CD(\Delta, X, Y) = \left\{ \left( \bigcup_{i \in S} \alpha_i, \bigcup_{j \notin S} \beta_j \right) \mid S \subseteq \{1, \ldots, n\} \right\}
\]

This clausal decomposition allows us to write CNF $\Delta$ as follows

\[
\Delta = \bigvee_{S \subseteq \{1, \ldots, k\}} \left( \bigwedge_{i \in S} \alpha_i \right) \land \left( \bigwedge_{j \notin S} \beta_j \right)
\]

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

\[
\left\{ \left( \Delta(X) \cup L_1, \Delta(Y) \cup R_1 \right), \ldots, \left( \Delta(X) \cup L_n, \Delta(Y) \cup R_n \right) \right\}
\]

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

\[
\begin{array}{c}
X \quad \{ z \lor q \} \\
\{ y \lor \neg z \} \quad Q \\
Y \quad Z
\end{array}
\]

Figure 2. Distributing the clauses of \( \{ y \lor \neg z, z \lor q, \neg x \lor z, x \lor \neg y \lor q \} \) 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 is called a Shannon node if 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 \( \pi \) be a variable ordering. The right-linear vtree induced by \( \pi \) is the one whose in-order traversal visits leaves in the same order as \( \pi \). Figure 4 shows the right linear vtree induced by order \( X, Y, \ldots, Y_n \).

We will find it useful to distribute the clauses of a CNF \( \Delta \) on a vtree as follows. Each clause of \( \Delta \) 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 \( \text{Clauses}(v) \) to denote the clauses assigned to vtree node \( v \). We also use \( \text{CNF}(v) \) 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 \( \Delta \) 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 \( c2s(v, \{\}) \) to Algorithm 1 returns a DNNF that respects vtree \( v \) and that is equivalent to \( \text{CNF}(v) \).

More generally, a recursive call \( c2s(v, S) \) will return a DNNF for \( \text{CNF}(v) \cup S \) 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].

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 \( \Delta \) and variables \( X \). Let \( \gamma_1, \ldots, \gamma_n \) be the clauses in \( \Delta \) which mention variables inside and outside \( X \), and let \( \alpha_i \) be the sub-clause of \( \gamma_i \) with variables in \( X \). The \( X \)-components of \( \Delta \) are defined as the following CNFs

\[ \text{CNF}_X(\Delta, X) = \{ \Delta(X) \cup \Gamma \mid \Gamma \subseteq \{ \alpha_1, \ldots, \alpha_n \} \} \]

where \( \Delta(X) \) is the set of clauses of \( \Delta \) that only mention variables \( X \).

For example, if \( \Delta = \{ x_1, x_2 \lor z, x_3 \lor \neg z \} \) and \( X = \{ X_1, X_2, X_3 \} \), then \( \text{CNF}_X(\Delta, X) = \{ \{ x_1 \}, \{ x_1, x_2 \}, \{ x_1, x_3 \}, \{ x_1, x_2, x_3 \} \} \).

Suppose that we split on variables \( V \), leading to CNFs \( \Delta|v \) one CNF for each instantiation \( v \) of variables \( V \). Suppose that we further construct a clausal decomposition for each CNF \( \Delta|v \). We will find it quite useful to count the number of distinct components which are obtained from this process.
Definition 4. Consider a CNF $\Delta$ and disjoint variables $X$ and $V$. The $X|V$-components of $\Delta$ are defined as the following CNFs:

$$\text{CNF}\{\Delta, X|V\} = \bigcup_{v} \text{CNF}\{\Delta|v, X\}.$$ 

Consider the CNF

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

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

$$\Delta|v = \{x_2 \lor v \lor z, x_3 \lor v\},$$

$$\text{CNF}\{\Delta|v, X\} = \{\{x_2\}, \{x_3\}, \{x_2, x_3\}\}.$$ 

Hence,

$$\text{CNF}\{\Delta, X|V\} = \{\{\}, \{x_2\}, \{x_3\}, \{x_2, x_3\}, \{x_2 \lor \neg x_3\}, \{x_1, x_2 \lor \neg x_3\}\}.$$ 

These are all the distinct $X$-components obtained by first splitting on variables $V$, then constructing clausal decompositions. We will use $\#\text{CNF}\{\Delta, X|V\}$ to denote the ceiling of $\log(|\text{CNF}\{\Delta, X|V\}|)$, where $\log 0$ is defined as $0$. Hence, in the above example $\#\text{CNF}\{\Delta, X|V\} = 3$.

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 $\Delta$ 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 $v$ and inside $v'$.
- **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 $X$, context variables $V$, cutset clauses $\Delta$ and context clauses $\Gamma$. The width of node $v$, $\text{width}(v)$, is $|\Delta| + \#\text{CNF}\{\Delta, X|V\}$. 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 $\{y \lor \neg z, z \lor q, \neg x \lor v, x \lor \neg y \lor q\}$ and the vtree in Figure 1(a). The CV-width of this vtree is 2; see Figure 3.

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 $\text{v2OBDD}(v, \{\})$ to Algorithm 1 takes time in $O(nm^3w)$ and returns a DNNF whose size is in $O(nm^3w)$.

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 $O(n3^n)$. In fact, a simple argument can show that the size is actually in $O(n2^n)$.

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 if 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 if 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 $\text{twp}$, $\text{twd}$ and $\text{twi}$ to denote the treewidth of primal, dual and incidence graphs, respectively. It is known that $\text{twp}$ and $\text{twd}$ 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 $\text{twi} \leq \text{twp} + 1$ and $\text{twi} \leq \text{twd} + 1$ [6]. We will next show that CV-width dominates $\text{twi}$, which immediately implies that it also dominates $\text{twp}$ and $\text{twd}$.

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

![Figure 3](image-url)
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 $\Delta_n$, 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 $C_i = x_i \lor \cdots \lor x_i$. The incidence graph of $\Delta_n$ 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 $\Gamma$ be the context clauses of $v$. If $i = 1$, then $\Gamma$ is empty and the width of $v$ is 0. Otherwise, $\Gamma = \{C_{i-1}, \ldots, C_n\}$. Let $X$ be the variables inside $v$, and let $V$ be the context variables of $v$. Then, $CNFs(\Gamma, X|V) = \{\{x_i, x_i \lor x_{i+1}, \ldots, x_i \lor \cdots \lor x_n\}\}$. The width of $v$ is then 1. The CV-width of the vtree is then $n$.

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 $\Delta$. The $i^{th}$ cutset of order $\pi$ is the set of clauses in $\Delta$ that mention a variable $V_j$, $j \leq i$, and a variable $V_k$, $k > i$. The cutwidth of order $\pi$ is the size of its largest cutset. The cutwidth of CNF $\Delta$ is the smallest cutwidth attained by any variable ordering $\pi$.

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

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

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

**Proof.** Consider the right-linear vtree induced by $\pi$. Let $v$ be an internal vtree node with variables $X$, context clauses $\Gamma$, and context variables $V$. It suffices to show that $\text{width}(v) \leq cw$ and $\text{width}(v) \leq pw$. Node $v$ must be a Shannon node. Thus, its cutset is empty and $\text{width}(v) = \#CNFs(\Gamma, X|V)$. Assume that $\pi = V_1, \ldots, V_n$ and that $v'$ 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, $\Gamma$ is the $i^{th}$ cutset of order $\pi$. Since $\Gamma$ only mentions variables $X$ and $V$, $CNFs(\Gamma, X|V)$ is the distinct CNFs $\Gamma|v$. Hence, $\#CNFs(\Gamma, X|V) \leq 2^{\#\Gamma|v}$, leading to $\#CNFs(\Gamma, X|V) \leq |\Gamma|$ and so $\text{width}(v) \leq cw$.

Moreover, $\text{Vars}(\Gamma) \cap V$ is the $i^{th}$ separator of order $\pi$. Since $\#CNFs(\Gamma, X|V) \leq 2^{\#\text{Vars}(\Gamma) \cap V}$, we have $\#CNFs(\Gamma, X|V) \leq |\text{Vars}(\Gamma) \cap V|$ and $\text{width}(v) \leq pw$. 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.

**Theorem 6.** There is a class of CNFs $\Delta_n$, 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 $\leq 1$.

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

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 conjunctor operation (not supported by unstructured DNNF). Second, our bound is based on CW\-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 CW\-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 CW\-width. Not only does CW\-width dominate 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 CW\-width. We finally showed that linear CW\-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 motivated 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 \(\Sigma\), we use \(\Sigma \downarrow X\) to denote the CNF which results from replacing every clause in \(\Sigma\) by its sub-clause that mentions variables in \(X\). For example, if \(\Sigma = \{a \lor \neg b \lor c, \neg a \lor c \lor \neg d\}\) and \(X = \{A, B\}\), then \(\Sigma \downarrow X = \{a \lor \neg b, \neg a\}\).

**Lemma 1.** Let \(v\) be an internal vtree node with variables \(X\), cutset clauses \(\Delta\), context clauses \(\Gamma\) and context variables \(V\). The following hold when Algorithm 1 starts executing a call \(\text{call} \leq 2\Delta(v, S)\):

- If \(v\) is a Shannon node, then
  - (a) \(S \in CNFS(\Gamma, X | V)\)
- If \(v\) is not a Shannon node, then
  - (b) \(C \subseteq \Delta\)
  - (c) \(S_1 \cup S_2 \in CNFS(\Gamma, X | V)\)
  - (d) \(S_1 \subseteq \Sigma \downarrow X\) where \(\Sigma = \Delta \setminus C\)

We next prove Theorem 2.

**Proof (Theorem 2).** Let \(v\) be an internal vtree node with variables \(X\), cutset clauses \(\Delta\), context clauses \(\Gamma\), and context variables \(V\). 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), \(S \in CNFS(\Gamma, X | V)\). Hence, the number of uncalled to \(v\) is \(\leq 2^{\Delta + |CNFS(\Gamma, X | V)|}\) since \(\Delta = \emptyset\) for a Shannon node. Moreover, each uncalled 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 \(O(m2^{\text{width}(v)})\). Moreover, the total contribution it makes to the DNNF size is \(O(2^{\text{width}(v)})\).

Assume now that \(v\) is not a Shannon node. The following observations all follow from Lemma 1. First, by Lemma 1(d), if \(|S_1| = k\) and \(|\Sigma| = k\), then \(0 \leq k \leq \frac{m}{27}\). Moreover, there are at most \(2^k\) distinct CNFs \(S_i\) of size \(i\). Second, by Lemma 1(c), there are at most \(2^{\#CNFS(\Gamma, X | V)}\) \((k)\) uncalled calls to node \(v\) for which \(|S_1| = i\). Moreover, each of these calls will construct a clausal decomposition of size \(2^{3i+3i}3\) on Line 20. Hence, the decompositions constructed at Line 20 will have a total size of

\[
\sum_{i=0}^{k} 2^{\#CNFS(\Gamma, X | V)} \binom{k}{i} 2^{3i+3i}
\]

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

REFERENCES

B Proof of Theorem 3

In this section, we will prove Theorem 3. For that, we need the following definitions and results.

Definition 10. Let G be the incidence graph of a CNF. A dtree for G is a full binary tree, whose leaves have a one-to-one correspondence with the edges of G.

Note that a dtree node contains both variables and clauses under the subtree rooted at itself. Being able to refer those variables and clauses separately will be useful:

\[ Vars(d) = \begin{cases} \text{Variable of the node,} & \text{if } d \text{ is a leaf node,} \\ Vars(d') \cup Vars(d''), & \text{otherwise.} \end{cases} \]

\[ CNF(d) = \begin{cases} \text{Clause of the node,} & \text{if } d \text{ is a leaf node,} \\ CNF(d') \cup CNF(d''), & \text{otherwise.} \end{cases} \]

We will also need to represent variables and clauses of a dtree node together. For such cases, we will use the following notation:

\[ Labels(d) = Vars(d) \cup CNF(d). \]

We will now provide some definitions to define the width of a dtree.

Definition 11. The cutset of an internal dtree node d is

\[ \text{Cutset}(d) = (Labels(d') \cap Labels(d'')) \setminus \text{Acutset}(d), \]

where \( \text{Acutset}(d) \) is the union of cutsets of ancestors of d.

Definition 12. The context of a dtree node d is

\[ \text{Context}(d) = Labels(d) \cap \text{Acutset}(d), \]

where \( \text{Acutset}(d) \) is the union of cutsets of ancestors of d.

Definition 13. The cluster of a dtree node d is

\[ \text{Cluster}(d) = \begin{cases} \text{Labels}(d), & \text{if } d \text{ is a leaf node,} \\ \text{Cutset}(d) \cup \text{Context}(d), & \text{otherwise.} \end{cases} \]

We can now define the width of a dtree:

Definition 14. The width of a dtree is the size of its maximal cluster minus one.

Theorem 7. Given a CNF whose incidence graph has treewidth \( w \), we can construct a dtree of width \( w \).

Proof. Consider a CNF \( \Delta \) whose incidence graph has treewidth \( w \). Then, we can create an auxiliary CNF \( \Gamma \) from \( \Delta \) as follows:

- for each variable \( V \) in \( \Delta \), add a variable in \( \Gamma \),
- for each clause \( C \) in \( \Delta \), add a variable in \( \Gamma \),
- add a binary clause in \( \Gamma \) for each variable \( V \) and clause-variable \( C \) when \( V \) appears in \( C \).

Note that the primal graph of \( \Gamma \) is identical to the incidence graph of \( \Delta \). So, the primal graph of \( \Gamma \) has treewidth \( w \). We also know that we can create a “dtree” for \( \Gamma \), a full binary tree whose leaves are the clauses of \( \Gamma \), which has width same as treewidth of the primal graph of \( \Gamma \) [2]. Finally, such a “dtree” of \( \Gamma \) is actually a dtree for \( \Delta \), and their widths are the same, which is \( w \).

Now we know that we can create a dtree for a CNF whose width is the same as treewidth of CNF incidence graph, we will show a width-preserving algorithm that can construct a vtree from a dtree. But first, we need some more definitions.

Definition 15. Consider a CNF, a corresponding vtree and an internal vtree node \( \alpha \). Let \( \alpha \) be a context clause of \( \alpha \). Then, \( \alpha \) is called Type I context clause if all variables of \( \alpha \) that are outside \( \alpha \) are context variables of \( \alpha \). Otherwise, \( \alpha \) is a Type II context clause.

Definition 16. Let \( \Delta \) be a CNF and \( \Gamma \) be a set of variables. The cardinality of \( (\Delta, \Gamma) \) is defined as

\[ \text{Card}(\Delta, \Gamma) = \arg \min_{\Gamma \subseteq \Delta} |\Gamma| + |Vars(\Delta \setminus \Gamma) \cap V|. \]

The notion of cardinality provides us an upper bound on the number of distinct CNFs that can be obtained from a given set of clauses after conditioning those clauses on given variables. For instance, assume that \( \Delta = \{x \lor y, \neg x \lor \neg y, x \lor \neg q, \neg x \lor q, x \lor z \lor w\} \) and \( V = \{Q, W, Y, Z\} \). Suppose we want to bound the number of distinct CNFs we can obtain by conditioning \( \Delta \) on all complete variable assignments of \( V \). Note here that, to compute \( \text{Card}(\Delta, V) \), we should go over subsets \( \Gamma \) of \( \Delta \), and variables \( Vars(\Delta \setminus \Gamma) \cap V \). The table below shows an incomplete list of such pairs of clauses and variables.

<table>
<thead>
<tr>
<th>( \Gamma )</th>
<th>( Vars(\Delta \setminus \Gamma) \cap V )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>( {Q, W, Y, Z} )</td>
</tr>
<tr>
<td>( {Q} )</td>
<td>( {Q, W, Y, Z} )</td>
</tr>
<tr>
<td>( {Q, W} )</td>
<td>( {Q, W, Z} )</td>
</tr>
<tr>
<td>( {W} )</td>
<td>( {W, Z} )</td>
</tr>
</tbody>
</table>

Back to bounding the distinct CNFs, given a variable assignment on \( \{Q, W, Y, Z\} \), observe that any clause in \( \Delta \) would be either subsumed or shrunken to a clause over \( \{X\} \). Then, one (loose) upper bound is \( 2^5 \), as there are 5 clauses. In fact, this is exactly what the 1st row in the table is meant to tells us. Another way to get an (still loose) upper bound is to count the number of variable assignments, as each assignment may create a different CNF. So, the bound is \( 2^3 \), as there are 4 variables to be conditioned on. In this case, this is what the 2nd row in the table is meant to tells us. So, we have a better upper bound. However, we may get even better bounds by considering a subset of clauses and a subset of variables, that is other rows in the table. Consider the 3rd row in the table. It essentially tells us that assignments over variables \( \{Q, Y\} \) can bound the number of distinct CNFs obtained from all clauses but \( x \lor z \lor w \), by conditioning on \( \{Q, W, Y, Z\} \). So, there are at most \( 2^2 \) such distinct CNFs. As we can get two distinct CNFs from the clause \( x \lor z \lor w \) by conditioning on \( \{Q, W, Y, Z\} \), which are \( \emptyset \) and \( \{x\} \), the upper bound is \( 2^2 \). Note that the 3rd row contains the minimum \( \Gamma \) in the definition of \( \text{Card}(\Delta, \Gamma) \). Thus, we have the following result:

Theorem 8. Let \#CNF(\( \Delta, V \)) denote the number of distinct CNFs \( \Delta \setminus \emptyset \) such that \( V \) is an instantiation of variables \( \emptyset \). If \( k \) is the cardinality of \( (\Delta, V) \), then \#CNF(\( \Delta, V \)) \leq 2^k.

We will now show the relationship between \#CNF(\( \Gamma, X \setminus V \)) and the cardinality.

Lemma 2. Let \( \Delta(X, V) \) be a CNF over two disjoint sets of variables \( X \) and \( V \). Then, \( |\text{CNF}(\Delta, X \setminus V)| \) is equal to the number of distinct CNFs \( \Delta \setminus V \).

Proof. Since clauses in $\Delta$ are over variables $X$ and $V$, $\Delta \mid v$ is a CNF over variables $X$. This implies that the set of clauses in $\Delta \mid v$ that mention variables inside and outside $X$, is empty. Thus, we have the following equations, which shows that $|CNFs(\Delta, X \mid V)|$ is equal to the number of distinct CNFs $\Delta \mid v$.

$$CNFs(\Delta, X \mid V) = \bigcup_{v} CNFs(\Delta \mid v, X)$$

$$= \bigcup_{v} (\Delta \mid v \cup \Gamma \mid \Gamma \subseteq \{\})$$

$$= \bigcup_{v} (\Delta \mid v).$$

Lemma 3. Let $\Delta$ be a CNF, and $X$ and $V$ be two disjoint sets of variables. Then, $|CNFs(\Delta, X \mid V)| \leq 2^{|\Delta|}$.

Proof. Let $\Gamma$ be a CNF belonging to $CNFs(\Delta, X \mid V)$, and $\alpha$ be a clause of $\Gamma$. Then, $\alpha$ must be over variables $X$. In fact, $\alpha$ is the sub-clause of some clause $\beta$ in $\Delta$, which is obtained by replacing $\beta$ by its sub-clause that mentions variables $X$. To see this, note that $\alpha$ can be obtained after performing a (possible) variable splitting and a (possible) clause splitting on $\Delta$. This implies that $\Gamma$ can be constructed using clauses of $\Delta$. Since the number of different CNFs one can construct using clauses of $\Delta$ is not greater than $2^{|\Delta|}$, the claim of the lemma holds.

Lemma 4. Let $\Delta = \Delta_1 \cup \Delta_2$ be a CNF, and $X$ and $V$ be two disjoint sets of variables. Then, $|CNFs(\Delta, X \mid V)| \leq |CNFs(\Delta_1, X \mid V)| \times |CNFs(\Delta_2, X \mid V)|$.

Proof. In the following, for any CNF $\Sigma$, we denote the clauses in $\Sigma$ that only mention variables $X$ by $\Sigma(X)$. Further, we assume $\delta_1,\ldots,\delta_n \mid v$ contains the clauses in $\Delta \mid v$ that mention variables inside and outside $X$, where $v$ is an instantiation of $V$. So, we denote by $\delta_1,\ldots,\delta_n \mid v$ the sub-clauses of $\delta_i$ with variables in $X$. Finally, $\gamma_1,\ldots,\gamma_k \mid v$ denotes the maximal subset of $\delta_1,\ldots,\delta_n \mid v$ such that each $\gamma_i$ appears in $\Delta \mid v$, and likewise $\gamma_1,\ldots,\gamma_k \mid v$ is the maximal subset of $\delta_1,\ldots,\delta_n \mid v$ such that each $\gamma_i$ appears in $\Delta \mid v$.

$$CNFs(\Delta, X \mid V) = \bigcup_{v} CNFs(\Delta \mid v, X)$$

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

$$= \bigcup_{v} (\Delta \mid v(X) \cup \Delta_2 \mid v(X) \cup \Gamma \mid$$

$$\Gamma \subseteq \{\alpha_1,\ldots,\alpha_n\})$$

$$= \bigcup_{v} (\Delta \mid v(X) \cup \Gamma_1 \cup \Delta_2 \mid v(X) \cup \Gamma_2 \mid$$

$$\Gamma_1 \subseteq \{\gamma_1,\ldots,\gamma_k\},$$

$$\Gamma_2 \subseteq \{\gamma_1,\ldots,\gamma_k\})$$

$$= \bigcup_{v} (\Sigma_1 \cup \Sigma_2 \mid \Sigma_1 \in CNFs(\Delta_1 \mid v, X),$$

$$\Sigma_2 \in CNFs(\Delta_2 \mid v, X))$$

Note that $\Sigma_1$ belongs to $CNFs(\Delta_1 \mid v, X)$ implies that $\Sigma_1$ also belongs to $CNFs(\Delta_1, X \mid V)$ (analogously, $\Sigma_2$ belongs to $CNFs(\Delta_2, X \mid V)$). Therefore, $|CNFs(\Delta, X \mid V)| \leq |CNFs(\Delta_1, X \mid V)| \times |CNFs(\Delta_2, X \mid V)|$.

Algorithm 2: $d2v(d)$

Input: $d$: a dtree node.
Output: A vtree or nil.
1 if $d$ is a leaf then
2 if $Vars(d)$ appears only in $CNFs(d)$ then
3 _ return Leaf node labeled with $Vars(d)$
4 else return nil
5 $C \leftarrow cutset(d)$
6 $T \leftarrow$ right-linear vtree obtained from $Vars(C)$
7 $T_r \leftarrow d2v(d')$
8 $T_r \leftarrow d2v(d')$
9 if $T_l$ and $T_r$ are nil then return $T$
10 else if $T_l$ and $T_r$ are not nil then
11 $T' \leftarrow$ vtree node whose left child is $T_l$ and right child is $T_r$
12 _ return $T$ by making $T'$ its rightmost child
13 else if $T_l$ is nil and $T_r$ is not nil then
14 _ return $T$ by making $T_r$ its rightmost child
15 else if $T_l$ is not nil and $T_r$ is then nil then
16 _ return $T$ by making $T_l$ its rightmost child

Theorem 9. Let $v$ be an internal vtree node, with variables $X$, context clauses $\Gamma$, Type I context clauses $\Gamma_1$, Type II context clauses $\Gamma_2$, and context variables $V$. Then, $|CNFs(\Gamma, X \mid V)| \leq 2^{Card(\Gamma_1 \mid V)}$.

Proof. Clauses in $\Gamma_1$ are over variables $X$ and $V$. So, by Lemma 2 and Theorem 8, $|CNFs(\Gamma_1, X \mid V)| \leq 2^{Card(\Gamma_1 \mid V)}$. Also, by Lemma 3, $|CNFs(\Gamma_2, X \mid V)| \leq 2^{Card(\Gamma_2 \mid V)}$. Finally, by Lemma 4, $|CNFs(\Gamma, X \mid V)| \leq |CNFs(\Gamma_1, X \mid V)| \times |CNFs(\Gamma_2, X \mid V)|$. Therefore, $|CNFs(\Gamma, X \mid V)| \leq 2^{Card(\Gamma_1 \mid V) \times Card(\Gamma_2 \mid V)}$. By taking logs of both sides, we conclude $|CNFs(\Gamma, X \mid V)| \leq 2^{Card(\Gamma_1 \mid V)}$.

We next present a width-preserving algorithm that constructs vtrees from dtrees. In particular, given a dtree of width $w$, we create a vtree of width at most $w$.

Given a dtree for a CNF, Algorithm 2 computes a vtree. Observe that in the algorithm, an internal vtree node can be constructed on either Line 6 or Line 11. Also, any such node constructed on Line 6 is a Shannon node.

In the following, after presenting some lemmas, we show the result relating the width of a vtree to the width of a dtree that constructs the vtree.

Lemma 5. Let $v$ be an internal vtree node, which is constructed by Algorithm 2 at a call $d2v(d)$, where $d$ is a dtree node. Let $X$ be a variable inside $v$. Let $C$ be a clause in which a literal of $X$ appears. Then, there is a leaf dtree node with label $\{X,C\}$ in the subtree rooted at $d$.

Proof. Since $v$ is constructed at the call $d2v(d)$ and $X$ is inside $v$, the leaf vtree node $v'$ with label $X$ must be constructed at a call $d2v(d')$ where $d'$ is a dtree node in the subtree rooted at $d$. We show that there is a leaf dtree node with label $\{X,C\}$ in the subtree rooted at $d'$. As $v'$ is a leaf, it is constructed either on Line 2 or on Line 6. Assume $v'$ is constructed on Line 2. Then, by the if statement, $C$ is the only clause in which a literal of $X$ appears and also $d'$ is the leaf dtree node labeled with $\{X,C\}$. Now, assume $v'$ is constructed on Line 6. Then, $X$ is in the cutset of $d'$, which implies $X$ is in $Labels(d')$. Moreover, by Definition 10 (Dtree), there is a leaf node $l$ labeled by $\{X,C\}$ in the subtree in which $d'$ appears. Assume $l$ is outside $d'$. Since $X$ is in $Labels(d')$, $X$ must be in the cutset of an
Lemma 6. Let $v$ be an internal vtree node, which is constructed by Algorithm 2 at a call $\Delta d2v(d)$, where $d$ is a dtree node. Let $C$ be a clause mentioning variables inside and outside $v$. If $C$ is not in $Cluster(d)$, then $Vars(C) \cap Vars(v)$ is in $Cluster(d)$.

Proof. Assume $C$ is not in $Cluster(d)$. Let $X$ be a variable inside $v$ such that a literal of $X$ appears in $C$. Let $Y$ be a variable outside $v$ such that a literal of $Y$ appears in $C$. By Lemma 5, there is a leaf dtree node $l_x$ with label $\{X, C\}$ in the subtree rooted at $d$. Also, by Definition 10 (Dtree), there is a leaf dtree node $l_y$ with label $\{Y, C\}$ in the dtree in which $d$ appears. Since $l_x$ is inside $d$, $l_y$ must also be inside $d$. Otherwise, $C$ is in $Cluster(d)$, which is a contradiction. Wlog, assume that $l_y$ is inside $d$. Now assume that $Y$ is not in $Cluster(d)$. It implies that $Y$ only appears in $d'$. Recall that $v$ is constructed at the call $\Delta d2v(d')$, and $Y$ is outside $v$. However, if $Y$ only appears in $d'$, then it can be constructed only at a call $\Delta d2v(d')$, where $d'$ is a dtree node in the subtree rooted at $d'$. This means $Y$ is inside $v$, which is a contradiction. So, $Y$ is in $Cluster(d)$. Thus, when $C$ is not in $Cluster(d)$, we have $Vars(C) \cap Vars(v)$ is in $Cluster(d)$.

Lemma 7. Let $v$ be an internal vtree node, which is constructed by Algorithm 2 at a call $\Delta d2v(d)$, where $d$ is a dtree node. Let $C$ be a clause mentioning variables inside and outside $v$. If $C$ is not in $Cluster(d)$, then $C$ is a Type I context clause of $v$.

Proof. Assume $C$ is not in $Cluster(d)$. We need to show that all variables of $C$ appearing outside $v$ are context variables of $v$, and $C$ is not in the cutset of $v$. Let $Y$ be a variable of $C$ appearing outside $v$. By Lemma 6, $Vars(C) \cap Vars(v)$ is in $Cluster(d)$. So, $Y$ is in $Cluster(d)$. It implies $Y$ is in the cutset of a dtree node $d'$ that is on the path from the root of the dtree to $d$ (including $d$). So, the leaf vtree node labeled as $Y$ must be created at the call $\Delta d2v(d')$ on Line 6. As $d'$ is either $d$ or an ancestor of $d$, and $Y$ is outside $v$, $Y$ is a context variable of $v$. Let $\Delta$ be the cutset of $v$. We now show $C$ is not in $\Delta$. Assume $C$ is in $\Delta$. Then, $v$ cannot be a Shannon node, since $\Delta$ is empty for Shannon nodes, by Definition 5 (Cutset). So, $v$ must be created on Line 11. Also, by Definition 5 (Cutset), there is a variable $X_l$ inside $v$ such that a literal of $X_l$ appears in $C$, and, similarly, there is a variable $X_r$ inside $v$ such that a literal of $X_r$ appears in $C$. Since $v$ is constructed on Line 11, $v$ is constructed in a call $\Delta d2v(d')$ where $d'$ is a dtree node in the subtree rooted at $d'$. Then, by Lemma 5, there is a leaf dtree node $l_1$ with label $\{X, C\}$ inside $d'$. As $d'$ is inside $d'$, $l_1$ is also inside $d'$. Analogously, there is a leaf dtree node $l_2$ with label $\{Y, C\}$ inside $d'$. Then, $C$ is in $Cluster(d)$, which is a contradiction. So, $C$ is not in $\Delta$. Hence, we conclude that $C$ is a Type I context clause of $v$.

Lemma 8. Let $v$ be an internal vtree node with cutset $\Delta$, and Type II context clauses $\Gamma_2$, which is constructed by Algorithm 2 at a call $\Delta d2v(d')$, where $d'$ is a dtree node. Then, $(\Delta \cup \Gamma_2) \subseteq Cluster(d)$.

Proof. Let $C$ be a clause in $(\Delta \cup \Gamma_2)$. If $C$ is not in $Cluster(d)$, then $C$ is a Type I context clause of $v$ by Lemma 7. However, by Definition 15 (Type I context clause), $C$ cannot be a Type I context clause of $v$. So, $C$ is in $Cluster(d)$, that is, $(\Delta \cup \Gamma_2) \subseteq Cluster(d)$.

Lemma 9. Let $v$ be an internal vtree node with cutset $\Delta$, Type I context clauses $\Gamma_1$, Type II context clauses $\Gamma_2$, and context variables $V$, which is constructed by Algorithm 2 at a call $\Delta d2v(d')$, where $d'$ is a dtree node. Then, $Card(\Gamma_1, V) \leq |Cluster(d) \setminus (\Delta \cup \Gamma_2)|$.

Proof. Let $S = Cluster(d) \setminus (\Delta \cup \Gamma_2)$. Let $\Gamma$ be the maximal subset of $\Gamma_1$ such that $\Gamma \subseteq S$. We show that $\Gamma \cup (Vars(\Gamma_1 \setminus \Gamma) \cap V) \subseteq S$, which suffices to show $Card(\Gamma_1, V) \leq |S|$. By definition, $\Gamma \subseteq S$. Let $C$ be a clause in $\Gamma_1 \setminus \Gamma$. So, $C$ is not in $S$. Also, as $\Gamma_1$ and $\Delta \cup \Gamma_2$ are disjoint, $C$ is not in $Cluster(d)$. Then, by Lemma 6, $Vars(C) \cap Vars(v)$ is in $Cluster(d)$. In fact, as $Vars(C) \cap Vars(v)$ and $\Delta \cup \Gamma_2$ are disjoint, $Vars(C) \cap Vars(v)$ is in $S$. Also, by Definition 15 (Type I context clause), all variables of $C$ appearing outside $v$ are context variables of $v$. Thus, $Vars(C) \cap V \subseteq S$, which implies $(Vars(\Gamma_1 \setminus \Gamma) \cap V) \subseteq S$. As we have $\Gamma \cup (Vars(\Gamma_1 \setminus \Gamma) \cap V) \subseteq S$, and $\Gamma$ and $Vars(\Gamma_1 \setminus \Gamma) \cap V$ are disjoint, $|\Gamma| + |Vars(\Gamma_1 \setminus \Gamma) \cap V| \leq |S|$. So, by Definition 16, we conclude $Card(\Gamma_1, V) \leq |S|$.
\[ i - 1 \text{ elements of } \pi \text{ from } G. \text{ If none of } v_1, \ldots, v_{i-1} \text{ belongs to } S, \\text{ then each edge between } v_i \text{ and elements of } S \text{ exists in } G_i. \text{ That is, } v_i \text{ has at least } [n/2] \text{ edges in } G_i. \text{ If some of } v_1, \ldots, v_{i-1} \text{ belongs to } S, \text{ then } v_i \text{ has an edge to each element, except itself, of } I. \text{ To see this, note that each element of } I \text{ exists while eliminating } i - 1 \text{ elements, and all elements of } I \text{ must become connected after eliminating an element from } S \text{ in one of } i - 1 \text{ eliminations. So, } v_i \text{ has at least } [n/2] \text{ edges. Thus, treewidth is } \geq [n/2] - 1, \text{ which is } \geq n/2 - 2. \text{ Next, we show CV-width is } 0. \text{ Consider the right-linear vtree induced by the variable ordering } X_1, X_2, \ldots, X_n. \text{ That is, the left child of the vtree root } v = X_1. \text{ The left child of } v' = X_2, \text{ and so on (Figure 4 shows an example right-linear vtree). Consider a vtree node } v \text{ whose left child is } X_1. \text{ Since } v \text{ is a Shannon node, its cutset is empty. Let } \Gamma \text{ be the context clauses of } v. \text{ If } i = 1, \text{ then } \Gamma \text{ is empty and the width of } v \text{ is } 0. \text{ Otherwise, } \Gamma = \{C_1, \ldots, C_n\}. \text{ Let } X \text{ be the variables inside } v, \text{ and let } V \text{ be the context variables of } v. \text{ Then, } CNFs(\Gamma, X|V) = \{\{x_1, x_2 \lor x_{i+1}, \ldots, x_i \lor \ldots \lor x_n\}\}. \text{ The width of } v \text{ is then } 1. \text{ The CV-width of the vtree is then } 0. \\]

D Proof of Theorem 6

Theorem 6. There is a class of CNFs \( \Delta_n \), 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 \( \leq 1 \).

Proof. \( \Delta_n = \{x \lor y_1, \ldots, x \lor y_n, y_1 \lor \ldots \lor y_n\} \). Consider the variable ordering \( \pi = X, Y_1, \ldots, Y_n \). Figure 4 shows the right-linear vtree induced by \( \pi \). According to this figure, the linear CV-width of this vtree is 1 and the linear CV-width of CNF \( \Delta_n \), is \( \leq 1 \). Consider now an arbitrary variable ordering \( \pi \) for \( \Delta_n \). 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 \( \pi \) cannot both be \( X \). So, due to clause \( \{y_1 \lor \ldots \lor y_n\} \), the \((n - 1)^{th}\) separator must contain at least \( n - 2 \) variables. Thus, the pathwidth is \( \geq n - 2 \) for any order \( \pi \). To show that cutwidth is \( \geq n/2 - 1 \), we will look at the position of variable \( X \) in the order \( \pi \). Assume \( \pi = V_1, \ldots, V_{n+1} \). Let \( V_i = X \). If \( i \leq [n/2] \), then the set \{\( V_{i+1}, \ldots, V_{n+1} \)\} has at least \([n/2] + 1\) variables. Note that there is a distinct clause \( \{x \lor y_j\} \) for each \( V_j \neq i \). So, the cutwidth has at least \([n/2] + 1\) clauses. If \( i \geq [n/2] + 1 \), then the set \{\( V_1, \ldots, V_{i-1} \)\} has at least \([n/2] - 1\) variables. Then, the \((i - 1)^{th}\) cutset has at least \([n/2] - 1\) clauses. So, the cutwidth is \( \geq [n/2] \), which is \( \geq n/2 - 1 \), for any order \( \pi \).

E Proof of Theorem 1

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

Proof. The proof is by induction on vtree nodes. Base case, which happens when v is a leaf node, is trivially satisfied by Line 3. Now, let v be an internal node. As an induction base, assume that for each vtree node v' below v, c2s(v', S') computes a DNNF that respects v', and that is equivalent to CNF(v') \( \cup S' \). Where S' is a CNF over Vars(v'). During the call to v, we either perform variable splitting (Lines 4–13) or clause splitting (Lines 14–20). In both cases, due to induction hypothesis, recursive calls c2s(v', S') must compute structured DNNFs of CNF(v') \( \cup S' \). As variable and clause splittings are both sound methods, the algorithm returns a DNNF of CNF(v) \( \cup S \) that respects v. So, the call c2s(v, \{\}\}) returns a DNNF that respects vtree v and that is equivalent to CNF(v).
Proof. By Lemma 10, \( S \in CNFs(\Sigma, X|V) \) with \( \Sigma \) being clauses that mention variables inside and outside \( X \). That means, there exists an instantiation \( v \) of \( V \) such that \( S \in CNFs(\Sigma|v, X) \). Note that, for a vtree node that is not Shannon, \( \Sigma = \Gamma \cup (\Delta \setminus C) \), where \( \Gamma \) and \( \Delta \setminus C \) are disjoint. Then, we have the following:

\[
S \in CNFs(\Gamma|v \cup (\Delta \setminus C)|v, X) \quad (1)
\]

\[
S \in \{ \Sigma_1 \cup \Sigma_2 \mid \Sigma_1 \in CNFs(\Gamma|v, X), \Sigma_2 \in CNFs((\Delta \setminus C)|v, X) \} \quad (2)
\]

\[
S_3 \in CNFs((\Delta \setminus C)|v, X) \quad (3)
\]

\[
S_3 \subseteq \Sigma|v \cup (\Delta \setminus C)|v, X \quad (4)
\]

\[
S_1 \cup S_2 \in CNFs(\Gamma|v, X) \quad (5)
\]

Eq. (1) is due to Lemma 11. Eq. (2) is due to \( S_3 \) consists of clauses that mention both variables in \( v_l \) and \( v_r \), but none of the clauses in \( CNFs(\Gamma|v, X) \) can mention both variables in \( v_l \) and \( v_r \). Eq. (3) follows from Definition 3. Eq. (4) is due to \( S_1 \cup S_2 \) does not contain any clause that mentions both variables in \( v_l \) and \( v_r \), but each clause in \( CNFs((\Delta \setminus C)|v, X) \) must mention both variables in \( v_l \) and \( v_r \). Eq. (5) follows from Definition 3.

Lemma 1. Let \( v \) be an internal vtree node with variables \( X \), cutset clauses \( \Delta \), context clauses \( \Gamma \) and context variables \( V \). The following hold when Algorithm 1 starts executing a call \( c2s(v, S) \):

If \( v \) is a Shannon node, then

(a) \( S \in CNFs(\Gamma, X|V) \)

If \( v \) is not a Shannon node, then

(b) \( C \subseteq \Delta \)

(c) \( S_1 \cup S_2 \in CNFs(\Gamma, X|V) \)

(d) \( S_3 \subseteq \Sigma \downarrow X \) where \( \Sigma = \Delta \setminus C \)

Proof. Item (a) is an immediate corollary of Lemma 10, as for a Shannon node clauses that mention variables inside and outside \( X \) are its context clauses \( \Gamma \). Item (b) holds because of the way clauses are distributed over vtree nodes. Item (c) is due to Lemma 12. For Item (d), by Lemma 12, we know that \( S_3 \in CNFs(\Sigma, X|V) \). This simply implies that \( S_3 \) is obtained from some of the clauses in \( \Sigma \) by replacing them with their sub-clauses that mention variables \( X \). That is, \( S_3 \subseteq \Sigma \downarrow X \). ■