CV-width: A New Complexity Parameter for CNFs

Umut Oztok and Adnan Darwiche

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 clause 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 \( \Delta \) by considering the CNFs \( \Delta \mid v \) and \( \Delta \mid \neg v \) (i.e., conditioning \( \Delta \) on both phases of the variable). Clause splitting, however, is a less common technique and calls for eliminating a clause \( \alpha \lor \beta \) from a CNF \( \Delta \) 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 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-
Figure 1. A vtree and a respecting DNNF.

ure 1(a) depicts an example vtree. We will use $v^i$ and $v^r$ to refer to the left and right children of internal vtree node $v$. We will also use $V_{ars}(v)$ 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 $V_{ars}(N^l) \subseteq V_{ars}(v^l)$ and $V_{ars}(N^r) \subseteq V_{ars}(v^r)$ 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., $\{x \lor \neg y \lor \neg z, \neg x, y \lor z\}$). 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 $\neg \ell$ from all clauses and then dropping all clauses that contain that 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 $x$ of variables $X$ will be interpreted as a term (conjunct 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 $v^l$ and let $Y$ be the variables of right child $v^r$. 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 $v^l$ and $v^r$. The process continues recursively until we reach literals or constants.

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:

$$\left\{\left\{a \lor \neg b, \neg a \lor b\right\}, \left\{a \lor \neg b\right\}, \left\{\neg a \lor b, \{\neg c\}\right\}\right\}.$$
Algorithm 1: \( c2s(v, S) \)

\[ \text{cache}(v, \Delta) \] is a hash table that maps \( v \) and \( \Delta \) into a DNNF. \( \text{terminal}(\Delta) \) returns the literal or constant equivalent to \( \Delta \).

**Input:** \( v \) : a vtree node, \( S \) : a CNF over \( \text{Vars}(v) \).

**Output:** A DNNF for \( \text{CNF}(v) \cup S \) that respects vtree \( v \).

1. if \( \text{cache}(v, S) \neq \text{nil} \) then return \( \text{cache}(v, S) \)
2. \( C \leftarrow \text{Clauses}(v) \)
3. if \( v \) is a leaf then return \( \text{terminal}(C \cup S) \)
4. if \( v \) is a Shannon node then
5. \( X \leftarrow \text{Shannon variable of } v \)
6. if \( \{x\} \) and \( \{-x\} \) assigned to \( v' \) then \( \alpha \leftarrow \bot \)
7. else if \( \{x\} \) assigned to \( v' \) then
8. \( \alpha \leftarrow x \land c2s(v', (C \cup S) \setminus x) \)
9. else if \( \{-x\} \) assigned to \( v' \) then
10. \( \alpha \leftarrow \neg x \land c2s(v', (C \cup S) \setminus \neg x) \)
11. else
12. \( \alpha \leftarrow \left( x \land c2s(v', (C \cup S)(x) \right) \lor \left( \neg x \land c2s(v', (C \cup S) \setminus \neg x) \right) \)
13. \( X \leftarrow \text{variables in the vtree rooted at } v' \)
14. \( Y \leftarrow \text{variables in the vtree rooted at } v' \)
15. Partition \( S \) into \( S_1(X) \), \( S_2(Y) \), and \( S_3(X, Y) \)
16. \( \alpha \leftarrow \bot \)
17. foreach \((L, R) \in \text{CD}(C \cup S) \) do
18. \( \alpha \leftarrow \alpha \lor \left( c2s(v', S_1 \cup L) \land c2s(v', S_2 \cup R) \right) \)
19. \( \text{cache}(v, S) \leftarrow \alpha \)
20. return \( \alpha \)

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

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

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 \( \text{CNFs}(\Delta, X) = \{\{x_1\}, \{x_1, x_2\}, \{x_1, x_3\}, \{x_1, x_2, x_3\}\} \).

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

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

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

- $\Delta|w = \{x_2 \lor v \lor z, x_3 \lor v\}$, $CNFs(\Delta|w, X) = \{\{\}, \{x_2\}, \{x_3\}, \{x_2, x_3\}\}$,
- $\Delta|\neg w = \{x_1 \lor v \lor z, x_2 \lor \neg x_3\}$,
- $CNFs(\Delta|\neg w, X) = \{\{x_2 \lor \neg x_3\}, \{x_1, x_2 \lor \neg x_3\}\}$.

Hence,

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

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

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

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 $v$, width$(v)$, is $|\Delta| + \#CNFs(\Gamma, |X,V|)$. 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 $\text{call2v}([\{\}]$) 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 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 \( C_i = x_i \lor \cdots \lor x_1 \). 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, \( \Gamma = \{ C_1, \ldots, C_n \} \). Let \( X \) be the variables inside \( v \), and let \( V \) be the context variables of \( v \). Then, \( CNF(\Gamma, X | V) = \{ \} \), \( \{ x_i, x_i \lor x_{i+1}, \ldots, x_i \lor \cdots \lor x_n \} \). The width of \( v \) is then 1. The CV-width of the vtree is then 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 \( CNF \). The \( i \)th cutset of order \( \pi \) is the set of clauses in \( \Delta \) that mentions 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 \( CNF \) 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 \( CNF \). 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 \( CNF \) 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 \( CNF \), 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 width(\( v \)) \leq cw and width(\( v \)) \leq pw. Node \( v \) must be a Shannon node. Thus, its cutset is empty and width(\( v \)) \leq \#CNF(\( \Gamma, X | V \)). 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 \( V \), \( CNF(\Gamma, X | V) \) is the distinct CNFs \( \Gamma | V \). Hence, \( \#CNF(\Gamma, X | V) \leq 2^{\#\Gamma} \), leading to \( \#CNF(\Gamma, X | V) \leq |\Gamma| \) and so width(\( v \)) \leq cw. Moreover, \( Vars(\Gamma) \cap V \) is the \( i \)th separator of order \( \pi \). Since \( CNF(\Gamma, X | V) \leq \{ Y, Y \}, X \), we have \( \#CNF(\Gamma, X | V) \leq \#Vars(\Gamma) \cap V \) 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.

**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 \( \pi \). 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 primal 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 treewidth, 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 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 will 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 $\Sigma \downarrow X = \{a \lor \neg b, \neg a\}$. 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 $\text{call}(\alpha, S)$:

- If $v$ is a Shannon node, then
  - (a) $S \in \text{CNFS}(\Gamma, X | V)$
  - (b) $|X| \leq \Delta$
  - (c) $S_I \cap S_Z \in \text{CNFS}(\Gamma, X | V)$
  - (d) $S_I \subseteq \Sigma \downarrow X$ 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), $S \in \text{CNFS}(\Gamma, X | V)$. Hence, the number of uncached calls to $v$ is $\leq 2^{|\Delta| + \# \text{CNFS}(\Gamma, X | V)}$ since $|\Sigma| = 0$ 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 $O(m 2^{|\text{cut wid}(v)|})$. Moreover, the total contribution it makes to the DNNF size is $O(2^{|\text{cut wid}(v)|})$.

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

$$\sum_{i=0}^{k} 2^{|\text{CNFS}(\Gamma, X | V)|} \frac{k^i}{i!} 2^{|C|+i}.$$

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

REFERENCES