On Compiling CNFs into Structured Deterministic DNNFs

Simone Bova[1], Florent Capelli[2], Stefan Mengel[3], and Friedrich Slivovsky[1]

[1] Institute of Computer Graphics and Algorithms, TU Wien
[2] IMJ UMR 7586 - Logique, Université Paris Diderot
[3] LIX UMR 7161, Ecole Polytechnique, Université Paris-Saclay

Abstract. We show that the traces of recently introduced dynamic programming algorithms for #SAT can be used to construct structured deterministic DNNF (decomposable negation normal form) representations of propositional formulas in CNF (conjunctive normal form). This allows us prove new upper bounds on the complexity of compiling CNF formulas into structured deterministic DNNFs in terms of parameters such as the treewidth and the clique-width of the incidence graph.

1 Introduction

The aim of knowledge compilation is to succinctly represent propositional knowledge bases in a format that allows for answering a number of queries in polynomial time [6]. Choosing a representation language generally involves a trade-off between succinctness and the range of queries that can be efficiently answered. Constraints arising in various domains can often be conveniently modeled by propositional formulas in conjunctive normal form (CNFs), but most queries of interest, such as model counting, are intractable for CNF formulas.

Decomposable Negation Normal Forms (DNNFs) are a restricted form of Boolean circuits in negation normal form (NNF) such that the subcircuits leading into an AND gate are defined on disjoint sets of variables [4]. DNNFs—which generalize variants of binary decision diagrams such as ordered binary decision diagrams (OBDDs)—are among the most succinct representation languages considered in knowledge compilation. Although CNFs do not have DNNF representations of polynomial size in general [6, 1] they can be efficiently compiled into DNNFs when certain structural parameters are small, see [4, 5, 13–15, 11].

Among the key properties of DNNFs is that they allow for clause entailment queries in polynomial time. By imposing further restrictions, one obtains languages that efficiently support a wider range of queries and operations. A DNNF is *deterministic* (a *d-DNNF*, for short) if the subcircuits leading into an OR gate do not have satisfying assignments in common, and *structured* if its variables can be associated with the leaves of a binary tree so that, for each AND gate, one can find a tree node whose principal subtrees contain the variables occurring in the subcircuits leading into that gate. Deterministic DNNFs support model counting in linear time [5], and structured DNNFs allow for an efficient conjoin operation [13].

In this paper, we prove the following result (Theorem 1):

Theorem. A CNF formula with n variables, m clauses, and PS-width k can be compiled into a structured d-DNNF of size $O(k^3(n+m))$.

PS-width is a parameter that was introduced to characterize CNF formulas for which the model counting problem (#SAT) can be solved efficiently by means of recently developed dynamic programming algorithms [16, 17]. We prove Theorem 1 by showing that the traces of these algorithms can be used to construct structured d-DNNF representations of CNF formulas.

Our rationale for stating and proving the above theorem in terms of PS-width is that this parameter generalizes most width measures of formulas commonly considered in the literature [16]. Accordingly, we are able to immediately derive a number of corollaries. For instance, a CNF formula with m clauses and an incidence graph[4] of clique-width k has PS-width at most m^k [16]. This allows us to state an upper bound in terms of incidence clique-width as follows (Corollary 2):

Corollary. A CNF formula with n variables, m clauses, and incidence clique-width k can be compiled into a structured d-DNNF of size $O(m^{3k}(n+m))$.

In particular, any class of formulas of bounded incidence clique-width admits compilation into structured d-DNNFs of polynomial size. Such classes can have unbounded incidence treewidth, effectively putting them out of reach of known compilation algorithms generating DNNFs of size exponential in the incidence treewidth [15].

One can further show that a formula with incidence treewidth k has PS-width at most 2^{k+1} (see Proposition 1). Accordingly, the upper bound of Theorem 1 translates into the following bound in terms of incidence treewidth (Corollary 1):

Corollary. A CNF formula with n variables, m clauses, and incidence treewidth k can be compiled into a structured d-DNNF of size $O(8^k(n+m))$.

This comes close to the best known upper bound of $O(3^k n)$ on the complexity of compiling CNFs with incidence treewidth k into structured DNNFs [11], while allowing us to compile into the more restrictive language of structured *deterministic* DNNFs.

As far as compilation of CNFs into d-DNNFs is concerned, the best known result using a structural parameter is an upper bound of $O(2^k n)$ for CNF formulas with n variables and *decision-width* k [12]. As the decision-width of a formula is no greater than the treewidth of its primal graph, this bound translates into an upper bound of $O(2^k n)$ for formulas with n variables and primal treewidth k. The incidence treewidth of a formula is at most its primal treewidth plus one, but there are classes of formulas with bounded incidence treewidth and unbounded primal treewidth, so Corollary 1 yields an improvement whenever the difference between primal treewidth and incidence treewidth is sufficiently large.

[4] The incidence graph of a formula is the bipartite graph whose vertex classes consist of variables and clauses, and a variable is adjacent to the clauses it occurs in.

The degree of the polynomial in the upper bound of Corollary 2 depends on the incidence clique-width k, and one may wonder whether this can be improved to a bound of the form, say, $2^{O(k)}(n+m)^c$ for some constant c. We show that such an improvement is impossible, subject to a complexity-theoretic assumption (Theorem 2).

The remainder of the paper is structured as follows. In Section 2 we introduce basic notation and terminology. Section 3 proves Theorem 1 by showing how ideas implemented in recently introduced dynamic programming algorithms for #SAT can be used for compilation into structured d-DNNFs. We present corollaries of this result in Section 4. Section 5 provides evidence that our upper bound on the DNNF size of formulas in terms of incidence clique-width (Corollary 2) cannot be substantially improved. We conclude in Section 6.

2 Preliminaries

Formulas. A *literal* is a variable x or a negated variable $\neg x$. A *clause* is a finite set of literals. A clause is *tautological* if it contains the same variable negated as well as unnegated. A *(CNF) formula* (or *CNF*, for short) is a finite set of non-tautological clauses. If x is a variable, we let $var(x) = var(\neg x) = x$. The set of variables occurring in a clause C is $var(C) = \{\, var(\ell) \mid \ell \in C \,\}$, and the set of variables occurring in a formula F is $var(F) = \bigcup_{C \in F} var(C)$. The *length* of a formula F is $\sum_{C \in F} |C|$. The *incidence graph* of a formula F is the bipartite graph $I(F) = (F, var(F), E)$ such that there is an edge $xC \in E$ joining a variable $x \in var(F)$ and a clause $C \in F$ if and only if $x \in var(C)$.

A *truth assignment* (*assignment*, for short) is a mapping $\tau : X \to \{0,1\}$, where X is a set of variables. Extending assignments to literals in the usual way, we say that an assignment τ *satisfies* a clause C if there is a literal $\ell \in C$ such that $\tau(\ell) = 1$. An assignment *satisfies* a formula F if it satisfies every clause $C \in F$.

DNNFs. A *(Boolean) circuit in negation normal form* (or *NNF*) is a directed acyclic graph (DAG) with a single sink node (outdegree 0) where each source node (indegree 0) is labelled by a constant (0 or 1) or by a literal, and each other node is labelled by $\wedge$ (AND) or $\vee$ (OR). If φ is an NNF and v is a vertex of φ, the *sub-NNF* of φ rooted at v is the NNF obtained from φ by deleting every vertex from which v cannot be reached along a directed path. We write $var(\varphi)$ for the set of variables occurring in an NNF φ. Let φ be an NNF and let τ be an assignment to $X \supseteq var(\varphi)$. Relative to τ, we associate each vertex v of φ with a value $val_\varphi(v,\tau) \in \{0,1\}$ as follows. If v is labelled with a constant $c \in \{0,1\}$ then $val_\varphi(v,\tau) = c$, and if v is labelled with a literal ℓ then $val_\varphi(v,\tau) = \tau(\ell)$. If v is an AND node then we let $val_\varphi(v,\tau) = \min\{\, val_\varphi(w,\tau) \mid w \text{ is a child of } v \,\}$, and if v is an OR node we define $val_\varphi(v,\tau) = \max\{\, val_\varphi(w,\tau) \mid w \text{ is a child of } v \,\}$. We say that τ *satisfies* φ if $val_\varphi(s,\tau) = 1$, where s denotes the (unique) sink of φ. An NNF φ is said to *compute* a CNF formula F if the satisfying assignments of φ and F coincide. Similarly, we say that two NNFs φ

and ψ are *equivalent*, in symbols $\varphi \equiv \psi$, if they have the same set of satisfying assignments. For convenience, we interpret propositional expressions over literals and $\{0, 1, \wedge, \vee\}$ as NNFs. We also use the names of NNFs in expressions involving logical connectives, writing, for instance, $\varphi \wedge \psi$ to denote the NNF constructed from φ and ψ by adding a new AND node as a sink that has incoming edges from the sinks of φ and ψ.

An NNF φ is *decomposable* (in short, a *DNNF*) if every AND node v of φ satisfies the following property: if v has incoming edges from v_1 and v_2, and φ_1 and φ_2 denote the sub-NNFs of φ rooted at v_1 and v_2, respectively, then $var(\varphi_1)$ and $var(\varphi_2)$ are disjoint. A DNNF φ is *deterministic* (a *d-DNNF*) if, for every pair of distinct children v_1 and v_2 of an OR node, the sub-NNFs rooted at v_1 and v_2 do not have satisfying assignments in common.

3 From Dynamic Programming to Structured d-DNNFs

In this section, we show how ideas implemented in #SAT algorithms by Slivovsky and Szeider [17] and Saether et. al. [16] can be used for compiling CNF formulas into structured d-DNNFs.

3.1 Branch Decompositions, Projections, and PS-width

Given a formula F, the algorithms of [17, 16] perform dynamic programming on a *branch decomposition* of $F \cup var(F)$. Here, a branch decomposition of a finite set S is a binary tree whose leaves are in one-to-one correspondence with S (see the left-hand side of Figure 1 for an illustration). Formally, we will think of a branch decomposition as a pair (T, δ) consisting of a rooted binary tree T and a bijection δ from the set of leaves of T to the set S. Accordingly, if (T, δ) is a branch decomposition of the set $F \cup var(F)$ for some formula F, then δ bijectively maps each leaf of T to a variable or a clause of F.[5]

Partial solution counts computed by dynamic programming in [17, 16] are stored in tables indexed by pairs of *projections*. Here, the projection of a truth assignment $\tau : X \to \{0, 1\}$ onto a formula F is the set $F(\tau)$ of clauses of F satisfied by τ. Observe that the projection of the union of two assignments σ and τ (that agree on the intersection of their domains) onto F satisfies $F(\sigma \cup \tau) = F(\sigma) \cup F(\tau)$, and that τ is a satisfying assignment of F if, and only if, $F(\tau) = F$. For a formula F and a set X of variables we write $proj(F, X)$ for the set of projections of truth assignments $\tau : X \to \{0, 1\}$ onto F, formally

$$proj(F, X) = \{\, F(\tau) \mid \tau : X \to \{0, 1\} \,\}.$$

Let F be a CNF formula and let $\boldsymbol{T} = (T, \delta)$ be a branch decomposition of $F \cup var(F)$. For a node v of T, let T_v denote the subtree of T rooted at v, and

[5] Such decompositions can be thought of as generalizations of *vtrees*, which are binary trees whose leaves are in one-to-one correspondence with a set of variables and that have been studied before in knowledge compilation [13].

let $L(T_v)$ denote the set of leaves of T_v. We write $X_v^{\boldsymbol{T}}$ for the set of variables in the image of $L(T_v)$ under δ, and $F_v^{\boldsymbol{T}}$ for the set of clauses in the image of $L(T_v)$ under δ. We write $\overline{X_v^{\boldsymbol{T}}} = var(F) \setminus X_v^{\boldsymbol{T}}$ for the set of variables and $\overline{F_v^{\boldsymbol{T}}} = F \setminus F_v^{\boldsymbol{T}}$ for the set of clauses outside the subtree rooted at v. When $\boldsymbol{T}$ is clear from the context (as will be the case) we will omit $\boldsymbol{T}$ from the superscript.

Our main result states that a CNF formula can be represented by a structured d-DNNF of size polynomial in the number of clauses and a parameter called PS-width, which is defined as follows [16]: let F be a formula and let $\boldsymbol{T} = (T, \delta)$ be a branch decomposition of $F \cup var(F)$. The *PS-width* of $\boldsymbol{T}$ is defined

$$psw(\boldsymbol{T}) = \max_{v \in V(T)} \max(|proj(\overline{F_v}, X_v)|, |proj(F_v, \overline{X_v})|).$$

That is, the PS-width of $\boldsymbol{T}$ is the maximum number of projections "across" one of the bipartitions of F and $var(F)$ induced by a node of T. The *PS-width* of a formula F is the minimum PS-width of a branch decomposition of $F \cup var(F)$.

3.2 Records and Dynamic Programming

We now describe the "records" used by the dynamic programming algorithms for #SAT [17, 16].

Let F be a formula, let $\boldsymbol{T} = (T, \delta)$ be a branch decomposition of $F \cup var(F)$, and let v be a node of T. A *shape* (for v, with respect to $\boldsymbol{T}$) is a pair $\boldsymbol{S} = (S, S')$ of subsets of F such that $S \in proj(\overline{F_v}, X_v)$ and $S' \in proj(F_v, \overline{X_v})$. We say that an assignment $\tau : X_v \to \{0, 1\}$ has shape $\boldsymbol{S}$ if

(A) $\overline{F_v}(\tau) = S$, and
(B) $F_v(\tau) \cup S' = F_v$.

We write $N_v^{\boldsymbol{T}}(\boldsymbol{S})$ for the set of assignments of shape $\boldsymbol{S}$ (again, we drop $\boldsymbol{T}$ from the superscript if it is clear which branch decomposition we are using). The intermediate values for dynamic programming computed at node v are the cardinalities $|N_v(\boldsymbol{S})|$ for each shape $\boldsymbol{S}$ for v.

The reason for using shapes rather than just computing the number of assignments $\tau : X_v \to \{0, 1\}$ with projection $F(\tau) = S$ for each $S \in proj(F, X_v)$ is that, in some cases of interest (such as formulas of bounded clique-width [17]), the cardinality $|proj(F, X_v)|$ can be exponential in the number m of clauses while the number of shapes is bounded by a polynomial in m. This reduction in the amount of information required to represent partial solution counts is achieved by the use of an "expectation from the outside": by Condition (B), an assignment τ of shape (S, S') satisfies F_v when combined with an assignment $\sigma : \overline{X_v} \to \{0, 1\}$ such that $F_v(\sigma) = S'$. Since we are interested in satisfying assignments of F we expect τ to be paired with such an assignment σ and do not have to keep track of the projection $F_v(\tau)$.

We now explain how shapes for an inner node can be related to shapes for its child nodes in order to perform dynamic programming. Let F be a formula and let (T, δ) be a branch decomposition of $F \cup var(F)$. Let $\boldsymbol{S} = (S, S')$ be a shape for an inner node v of T, and let $\boldsymbol{S}_1 = (S_1, S_1')$, $\boldsymbol{S}_2 = (S_2, S_2')$ be shapes for its children v_1 and v_2, respectively. We say that $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ *generate* $\boldsymbol{S}$ if

(a) $S = (S_1 \cup S_2) \cap \overline{F_v}$,
(b) $S'_1 = (S' \cup S_2) \cap F_{v_1}$, and
(c) $S'_2 = (S' \cup S_1) \cap F_{v_2}$.

The following result relates the shapes for an inner node to the generating shapes for its children (here, $\sqcup$ denotes the disjoint union).

Lemma 1. *Let F be a formula, let $\boldsymbol{T} = (T, \delta)$ be a branch decomposition of $F \cup var(F)$, and let v be an inner node of T with children v_1 and v_2. Let $\boldsymbol{S}$ be a shape for v, and let G denote the set of pairs of shapes $\boldsymbol{S}_1$ for v_1 and $\boldsymbol{S}_2$ for v_2 such that $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ generate $\boldsymbol{S}$. Then*

$$N_v(\boldsymbol{S}) = \bigsqcup_{(\boldsymbol{S}_1, \boldsymbol{S}_2) \in G} \{\, \tau_1 \cup \tau_2 \mid \tau_1 \in N_{v_1}(\boldsymbol{S}_1), \tau_2 \in N_{v_2}(\boldsymbol{S}_2) \,\}.$$

Lemma 1 is an easy consequence of the following two lemmas (cf. [17]).

Lemma 2. *Let v be a node of T with children v_1 and v_2. Let $\boldsymbol{S}_1 = (S_1, S'_1)$ be a shape for v_1, let $\boldsymbol{S}_2 = (S_2, S'_2)$ be a shape for v_2, and let $\boldsymbol{S} = (S, S')$ be a shape for v generated by $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$. If $\tau_1 \in N_{v_1}(\boldsymbol{S}_1)$ and $\tau_2 \in N_{v_2}(\boldsymbol{S}_2)$ then $\tau_1 \cup \tau_2 \in N_v(\boldsymbol{S})$.*

Proof. Let $\tau_1 \in N_{v_1}(\boldsymbol{S}_1)$ and $\tau_2 \in N_{v_2}(\boldsymbol{S}_2)$. As $\overline{F_{v_1}}(\tau_1) = S_1$ and $\overline{F_{v_2}}(\tau_2) = S_2$, we get $\overline{F_v}(\tau_1 \cup \tau_2) = (S_1 \cup S_2) \cap \overline{F_v}$. This shows that Condition (A) is satisfied. Consider a clause $C \in F_v$ and assume without loss of generality that $C \in F_{v_1}$. Suppose $C \notin F_v(\tau_1 \cup \tau_2)$. Then τ_1 does not satisfy C and thus $C \in S'_1$ by Condition (B). But τ_2 does not satisfy C either, so $C \notin S_2$. The shapes $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ generate $\boldsymbol{S}$, so $S'_1 \subseteq S' \cup S_2$ and thus $C \in S'$ by Condition (b). This proves that Condition (B) is satisfied. We conclude that $\tau_1 \cup \tau_2$ has shape $\boldsymbol{S}$ as claimed. □

Lemma 3. *Let v be a node of T with children v_1 and v_2, let $\boldsymbol{S} = (S, S')$ be a shape for v, and let $\tau \in N_v(\boldsymbol{S})$. Let τ_1 and τ_2 denote the restrictions of τ to X_{v_1} and X_{v_2}, respectively. There is a unique pair of shapes $\boldsymbol{S}_1$ for v_1 and $\boldsymbol{S}_2$ for v_2 generating $\boldsymbol{S}$ such that $\tau_1 \in N_{v_1}(\boldsymbol{S}_1)$ and $\tau_2 \in N_{v_2}(\boldsymbol{S}_2)$.*

Proof. Let $S_1 = \overline{F_{v_1}}(\tau_1)$ and $S_2 = \overline{F_{v_2}}(\tau_2)$. Let $\tau' : \overline{X_t} \to \{0, 1\}$ be the assignment such that $S' = F_v(\tau)$. Then the sets $S'_1 = (S' \cup S_2) \cap F_{v_1}$ and $S'_2 = (S' \cup S_1) \cap F_{v_2}$ are the projections of the assignments $\tau' \cup \tau_2$ and $\tau' \cup \tau_1$ onto F_{v_1} and F_{v_2}, respectively. It follows that $\boldsymbol{S}_1 = (S_1, S'_1)$ is a shape for v_1 and that $\boldsymbol{S}_2 = (S_2, S'_2)$ is a shape for v_2. We verify that τ_1 has shape $\boldsymbol{S}_1$. Condition (A) is satisfied by construction. To see that Condition (B) is satisfied as well, let $C \in F_{v_1}$ and suppose C is not satisfied by τ_1. There are two cases. If τ_2 does not satisfy C either then $\tau = \tau_1 \cup \tau_2$ does not satisfy C and $C \in S'$ since τ has shape $\boldsymbol{S}$. Otherwise we have $C \in \overline{F_{v_2}}(\tau_2)$, that is, $C \in S_2$. In either case we have $C \in S'_1$ by choice of S'_1. The proof that τ_2 has shape $\boldsymbol{S}_2$ is symmetric. Let $\boldsymbol{R}_1 = (R_1, R'_1)$ and $\boldsymbol{R}_2 = (R_2, R'_2)$ be shapes for v_1 and v_2 such that $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ generate $\boldsymbol{S}$ and such that $\tau_1 \in N_{v_1}(\boldsymbol{R}_1)$ and $\tau_2 \in N_{v_2}(\boldsymbol{R}_2)$. We have $R_1 = \overline{F_{v_1}}(\tau_1) = S_1$ and $R_2 = \overline{F_{v_2}}(\tau_2) = S_2$ by Condition (A). As $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ generate $\boldsymbol{S}$, we further have $R'_1 = (S' \cup R_2) \cap F_{v_1}$ and $R'_2 = (S' \cup R_1) \cap F_{v_2}$. That is, $R'_1 = S'_1$ and $R'_2 = S'_2$, so $\boldsymbol{R}_1 = \boldsymbol{S}_1$ and $\boldsymbol{R}_2 = \boldsymbol{S}_2$. □

3.3 Constructing a Structured d-DNNF

Lemma 1 can be turned into a recurrence for determining the model count of F by dynamic programming [17, 16]. It can also be used to construct a structured d-DNNF for F.

To simplify matters, for the remainder of this subsection let F be an arbitrary, but fixed, formula, and let $\boldsymbol{T} = (T, \delta)$ be an arbitrary, but fixed, branch decomposition of $F \cup \mathit{var}(F)$. Starting at the leaves of T, we are going to construct a DNNF $\varphi_v(\boldsymbol{S})$ for each node v and each shape $\boldsymbol{S}$ for v. For a leaf node v of T, we have to consider two cases:

1. Suppose $\delta(v) = x$ for a variable x of F. For $\ell \in \{x, \neg x\}$, let τ_ℓ denote the assignment $\tau_\ell : \{x\} \to \{0,1\}$ such that $\tau(\ell) = 1$. The pairs $\boldsymbol{S}_x = (F(\tau_x), \emptyset)$ and $\boldsymbol{S}_{\neg x} = (F(\tau_{\neg x}), \emptyset)$ are the only shapes for v, and $N_v(\boldsymbol{S}_x) = \{\tau_x\}$ as well as $N_v(\boldsymbol{S}_{\neg x}) = \{\tau_{\neg x}\}$. Accordingly, we let $\varphi_v(\boldsymbol{S}_x) \equiv x$ and $\varphi_v(\boldsymbol{S}_{\neg x}) \equiv \neg x$.
2. Let $\delta(v) = C$ for a clause $C \in F$. The pairs $\boldsymbol{S}_\bot = (\emptyset, \emptyset)$ and $\boldsymbol{S}_\top = (\emptyset, \{C\})$ are the only shapes for v. Since $X_v = \emptyset$ it suffices to determine whether the empty assignment $\varepsilon : \emptyset \to \{0,1\}$ has one of these shapes. Because the empty assignment does not satisfy any clause we get $N_v(\boldsymbol{S}_\top) = \{\varepsilon\}$ and $N_v(\boldsymbol{S}_\bot) = \emptyset$, so we define $\varphi_v(\boldsymbol{S}_\bot) \equiv 0$ and $\varphi_v(\boldsymbol{S}_\top) \equiv 1$.

Let v be an inner node of T with children v_1 and v_2, and assume we have constructed $\varphi_{v_1}(\boldsymbol{S}_1)$ for each shape $\boldsymbol{S}_1$ for v_1 and $\varphi_{v_2}(\boldsymbol{S}_2)$ for each shape $\boldsymbol{S}_2$ for v_2. Let $\boldsymbol{S}$ be a shape for v and let G denote the set of pairs of shapes $\boldsymbol{S}_1$ for v_1 and $\boldsymbol{S}_2$ for v_2 that generate $\boldsymbol{S}$. We construct $\varphi_v(\boldsymbol{S})$ as

$$\varphi_v(\boldsymbol{S}) \equiv \bigvee_{(\boldsymbol{S}_1, \boldsymbol{S}_2) \in G} \varphi_{v_1}(\boldsymbol{S}_1) \wedge \varphi_{v_2}(\boldsymbol{S}_2). \tag{1}$$

That is, we create an AND node conjoining every pair $\varphi_{v_1}(\boldsymbol{S}_1)$ and $\varphi_{v_2}(\boldsymbol{S}_2)$ such that $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ generate $\boldsymbol{S}$, and then add an OR node that has an incoming edge from each AND node thus created. We assume that the resulting DNNF has been simplified by propagating constants.

Lemma 4. *For each node v of T and shape $\boldsymbol{S}$ for v, $\varphi_v(\boldsymbol{S})$ is a d-DNNF such that $\mathit{var}(\varphi_v(\boldsymbol{S})) \subseteq X_v$ and such that an assignment $\tau : X_v \to \{0,1\}$ satisfies $\varphi_v(\boldsymbol{S})$ if, and only if, $\tau \in N_v(\boldsymbol{S})$.*

Proof. It is easy to check that the statement holds for each leaf node v of T. Let v be an inner node and suppose the statement holds for its children v_1 and v_2. Let $\boldsymbol{S}$ be a shape for v. By assumption, $\mathit{var}(\varphi_{v_1}(\boldsymbol{S}_1)) \subseteq X_{v_1}$ and $\mathit{var}(\varphi_{v_2}(\boldsymbol{S}_2)) \subseteq X_{v_2}$ for every shape $\boldsymbol{S}_1$ for v_1 and every shape $\boldsymbol{S}_2$ for v_2. We have $X_v = X_{v_1} \cup X_{v_2}$ and since X_{v_1} and X_{v_2} are disjoint it follows that $\varphi_v(\boldsymbol{S})$ is a DNNF satisfying $\mathit{var}(\varphi_v(\boldsymbol{S})) \subseteq X_v$. Let $\tau : X_v \to \{0,1\}$ be a satisfying assignment of $\varphi_v(\boldsymbol{S})$, and let τ_1 and τ_2 denote the restrictions of τ to X_{v_1} and X_{v_2}, respectively. There is a pair of shapes $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ generating $\boldsymbol{S}$ such that τ satisfies the disjunct $\varphi_{v_1}(\boldsymbol{S}_1) \wedge \varphi_{v_2}(\boldsymbol{S}_2)$. By assumption, the lemma holds for v_1 and v_2. In particular, $\mathit{var}(\varphi_{v_1}(\boldsymbol{S}_1)) \subseteq X_{v_1}$ and $\mathit{var}(\varphi_{v_2}(\boldsymbol{S}_2)) \subseteq X_{v_2}$, so τ_1 satisfies $\varphi_{v_1}(\boldsymbol{S}_1)$ and τ_2

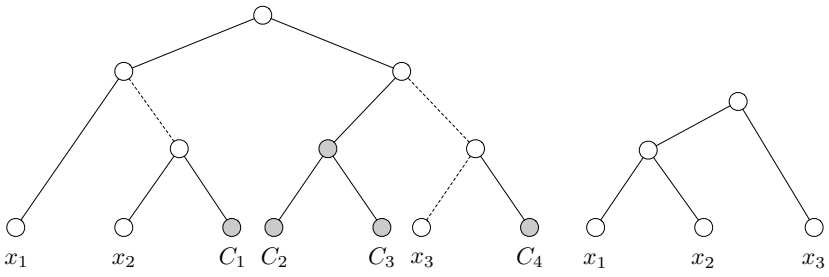

Fig. 1. The tree on the left is a branch decomposition of a formula $F = \{C_1, C_2, C_3, C_4\}$ with $var(F) = \{x_1, x_2, x_3\}$. To obtain the vtree on the right, we first delete each leaf node associated with a clause, as well inner nodes turned into leaf nodes by these deletions (the corresponding vertices are shown in grey). The resulting tree is turned into a binary tree by contracting edges incident to nodes of degree two (these edges are represented by dashed lines).

satisfies $\varphi_{v_2}(\boldsymbol{S}_2)$, which in turn implies that $\tau_1 \in N_{v_1}(\boldsymbol{S}_1)$ and $\tau_2 \in N_{v_2}(\boldsymbol{S}_2)$. It now follows from Lemma 1 that τ has shape $\boldsymbol{S}$. In addition to that, Lemma 1 tells us that $(\boldsymbol{S}_1, \boldsymbol{S}_2)$ is the unique pair of shapes generating $\boldsymbol{S}$ such that τ_1 has shape $\boldsymbol{S}_1$ and τ_2 has shape $\boldsymbol{S}_2$. Thus $\varphi_{v_1}(\boldsymbol{S}_1) \wedge \varphi_{v_2}(\boldsymbol{S}_2)$ is the unique disjunct satisfied by τ. By assumption, $\varphi_{v_1}(\boldsymbol{S}'_1)$ and $\varphi_{v_2}(\boldsymbol{S}'_2)$ are deterministic DNNFs for each shape $\boldsymbol{S}'_1$ for v_1 and $\boldsymbol{S}'_2$ for v_2, so $\varphi_v(\boldsymbol{S})$ is deterministic as well. Now let $\tau : X_v \to \{0,1\}$ be an assignment of shape $\boldsymbol{S}$, and let τ_1 and τ_2 denote its restrictions to X_{v_1} and X_{v_2}, respectively. By Lemma 1, there has to be a pair $(\boldsymbol{S}_1, \boldsymbol{S}_2)$ of shapes $\boldsymbol{S}_1$ for v_1 and $\boldsymbol{S}_2$ for v_2 generating $\boldsymbol{S}$ such that $\tau_1 \in N_{v_1}(\boldsymbol{S}_1)$ and $\tau_2 \in N_{v_2}(\boldsymbol{S}_2)$. It follows from our assumption that the lemma holds for v_1 and v_2 that τ_1 satisfies $\varphi_{v_1}(\boldsymbol{S}_1)$ and that τ_2 satisfies $\varphi_{v_2}(\boldsymbol{S}_2)$. Thus τ satisfies $\varphi_{v_1}(\boldsymbol{S}_1) \wedge \varphi_{v_2}(\boldsymbol{S}_2)$ and $\varphi_v(\boldsymbol{S})$. □

To show that $\varphi_v(\boldsymbol{S})$ is a *structured* DNNF, we have to provide a vtree respected by $\varphi_v(\boldsymbol{S})$ [13]. A vtree is a binary tree whose leaves are in one-to-one correspondence with a set of variables. We will think of a vtree simply as a branch decomposition of a set X of variables. A DNNF φ *respects* a vtree (T, δ) if each AND node v of φ has exactly two children and furthermore satisfies the following property: let v_1 and v_2 be the children of v in T, and let φ_1 and φ_2 denote the sub-DNNFs of φ rooted at v_1 and v_2, respectively; then there is a node t of T with children t_1 and t_2 such that the sub-DNNFs satisfy $var(\varphi_1) \subseteq \delta(L(T_{t_1}))$ and $var(\varphi_2) \subseteq \delta(L(T_{t_2}))$. Here, $L(T_{t_i})$ denotes the set of leaves in the subtree T_{t_i}, for $i \in \{1,2\}$.

For a node v of T, let $vtree(\boldsymbol{T}, v) = (T', \delta')$, where T' is the tree obtained from the subtree T_v by deleting all leaves w such that $\delta(w) \in F$, followed—if necessary—by a sequence of operations to make the resulting tree binary, and δ' is the restriction of δ to leaves of T'. Verify that $vtree(\boldsymbol{T}, v)$ is a branch decomposition of X_v and hence a vtree. We illustrate this construction in Figure 1.

Lemma 5. *For each node v of T and shape $\boldsymbol{S}$ for v, the DNNF $\varphi_v(\boldsymbol{S})$ respects $vtree(\boldsymbol{T}, v)$.*

Proof. The lemma trivially holds for each leaf node v of T and shape $\boldsymbol{S}$ for v, as $\varphi_v(\boldsymbol{S})$ does not contain any AND nodes. Let v be an inner node of T with children v_1 and v_2, and assume the lemma holds for v_1 and v_2 and their respective shapes. Let $\boldsymbol{S}$ be a shape for v. By construction, each AND node introduced in $\varphi_v(\boldsymbol{S})$ computes a conjunction $\varphi_{v_1}(\boldsymbol{S}_1) \wedge \varphi_{v_2}(\boldsymbol{S}_2)$, where $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ are shapes for v_1 and v_2, respectively, that generate $\boldsymbol{S}$. Since we assume $\varphi_v(\boldsymbol{S})$ to be simplified, both X_{v_1} and X_{v_2} have to be nonempty: otherwise, one of the conjuncts $\varphi_{v_i}(\boldsymbol{S}_i)$ for $i \in \{1, 2\}$ would satisfy $var(\varphi_{v_i}(\boldsymbol{S}_i)) = \emptyset$ by Lemma 4 and would have been simplified to a constant, which in turn would have been propagated through the AND node. Let $vtree(\boldsymbol{T}, v) = (T', \delta')$, let $vtree(\boldsymbol{T}, v_1) = (T_1, \delta_1)$, and let $vtree(\boldsymbol{T}, v_2) = (T_2, \delta_2)$. As both X_{v_1} and X_{v_2} are nonempty, T' is a binary tree whose principal subtrees are T_1 and T_2. By Lemma 4, the conjuncts satisfy $var(\varphi_{v_1}(\boldsymbol{S}_1)) \subseteq X_{v_1}$ and $var(\varphi_{v_2}(\boldsymbol{S}_2)) \subseteq X_{v_2}$. In combination with the assumption that the DNNF $\varphi_{v_i}(\boldsymbol{S}'_i)$ respects $vtree(\boldsymbol{T}, v_i)$ for each $i \in \{1, 2\}$ and shape $\boldsymbol{S}'_i$ for v_i, this implies that $\varphi_v(\boldsymbol{S})$ respects $vtree(\boldsymbol{T}, v)$. □

Let r denote the root of T and let $\emptyset = (\emptyset, \emptyset)$. We now prove that our construction yields a structured d-DNNF representation of F.

Lemma 6. *The pair $\emptyset$ is the only shape for r and $\varphi_r(\emptyset)$ is a structured d-DNNF computing F.*

Proof. The first part follows from the fact that $X_r = var(F)$ and $F_r = F$, so that $\overline{X_r} = \emptyset$ and $\overline{F_r} = \emptyset$. By Lemma 4 and Lemma 5, $\varphi_r(\emptyset)$ is a structured d-DNNF such that an assignment $\tau : var(F) \to \{0, 1\}$ satisfies $\varphi_r(\emptyset)$ if, and only if, $\tau \in N_r(\emptyset)$. By Condition (B), an assignment $\tau : var(F) \to \{0, 1\}$ has shape $\emptyset$ if, and only if, $F(\tau) \cup \emptyset = F$. That is, $N_r(\emptyset)$ is the set of satisfying assignments of F. □

Let n be the number of variables of F, let m be the number of clauses in F, and let k denote the PS-width of $\boldsymbol{T}$. The size of the structured d-DNNF constructed for F can be bounded as follows.

Lemma 7. *The DNNF $\varphi_r(\emptyset)$ has size at most $7k^3(n + m)$.*

Proof. We can assume without loss of generality that T contains at least one inner node. Let v be an inner node of T with children v_1 and v_2. Consider the DNNFs $\varphi_{v_1}(\boldsymbol{S}_1)$ for shapes $\boldsymbol{S}_1$ for v_1 and $\varphi_{v_2}(\boldsymbol{S}_2)$ for shapes $\boldsymbol{S}_2$ for v_2. We claim that all DNNFs $\varphi_v(\boldsymbol{S})$ for shapes $\boldsymbol{S}$ of v can be constructed from these DNNFs by introducing at most $5k^3$ new nodes and edges. If $\boldsymbol{S}$ is a shape for v and $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ are shapes for v_1 and v_2 that generate $\boldsymbol{S}$, we have to introduce an AND node and two edges to construct the DNNF computing $\varphi_{v_1}(\boldsymbol{S}_1) \wedge \varphi_{v_2}(\boldsymbol{S}_2)$, as well an edge from this AND node to the OR node that will eventually compute $\varphi_v(\boldsymbol{S})$. In the worst case, we have to create this OR node first. In total, we have to introduce at most 5 nodes and edges for each triple $(\boldsymbol{S}, \boldsymbol{S}_1, \boldsymbol{S}_2)$ of shapes such

that $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ generate $\boldsymbol{S}$. How many such triples are there? For any three projections $S_1 \in \mathit{proj}(\overline{F_{v_1}}, X_{v_1})$, $S_2 \in \mathit{proj}(\overline{F_{v_2}}, X_{v_2})$, and $S' \in \mathit{proj}(F_v, \overline{X_v})$, the projections $S'_1 \in \mathit{proj}(F_{v_1}, \overline{X_{v_1}})$, $S'_2 \in \mathit{proj}(F_{v_2}, \overline{X_{v_2}})$, and $S \in \mathit{proj}(\overline{F_v}, X_v)$ such that $\boldsymbol{S}_1 = (S_1, S'_1)$ and $\boldsymbol{S}_2 = (S_2, S'_2)$ generate $\boldsymbol{S} = (S, S')$ is uniquely determined. As there are at most k^3 such projections, we have to introduce at most $5k^3$ nodes and edges. The tree T has exactly $n + m - 1$ inner nodes, so we need at most $5k^3(n + m)$ nodes and edges to construct the DNNF $\varphi_r(\emptyset)$ from the DNNFs constructed for leaves of T. For each leaf node there at most two DNNFs consisting of a single node and there are $n + m$ leaves, so we require at most $7k^3(n + m)$ nodes and edges in total. □

Since we did not make any assumptions about the formula F and the branch decomposition $\boldsymbol{T}$, Lemma 6 and Lemma 7 yield the following result.

Theorem 1. *A CNF formula with n variables, m clauses, and PS-width k can be compiled into a structured d-DNNF of size $O(k^3(n + m))$.*

The above construction leads to an algorithm which, given a formula F and a branch decomposition $\boldsymbol{T}$ of $F \cup \mathit{var}(F)$, computes a structured d-DNNF representation of F. The pseudocode listed as Algorithm 1 provides the outlines of this procedure.[6] Using an efficient method for computing the set of shapes for each node during the initialization phase (for details, see Saether et. al. [16]), this algorithm can be made to run in time $O(k^3 m(n + m))$, where n is the number of variables of F, m is the number of clauses of F, and k is the PS-width of $\boldsymbol{T}$.

4 Corollaries

Theorem 1 allows us to derive compilation results for CNF formulas based on structural properties of their incidence graphs, namely *treewidth*, *directed clique-width*, and *clique-width* [8].

We first consider treewidth. A *tree decomposition* of a graph $G = (V, E)$ is a pair $(T, (B_t)_{t \in V(T)})$ where T is a tree and $(B_t)_{t \in V(T)}$ is a family of subsets of V (called "bags") such that:

1. For every vertex $v \in V$, the set $\{t \in V(T) \mid v \in B_t\}$ is non-empty and connected in T.
2. For every edge $uv \in E$, there is a $t \in V(T)$ such that $u, v \in B_t$.

The *width* of a tree decomposition $(T, (B_t)_{t \in V(T)})$ is the maximum size of a bag minus one, and the treewidth of G is the minimum of width attained over all tree decompositions of G. The incidence treewidth of a formula F defined as the treewidth of its incidence graph $I(F)$.

Proposition 1. *A formula of incidence treewidth k has PS-width at most 2^{k+1}.*

[6] To enhance readability, we suppress double brackets around shapes, writing, for instance, $\varphi_v(S, S')$ instead of $\varphi_v((S, S'))$.

Algorithm 1: Compiling CNFs into structured d-DNNFs.

Input: a CNF F and a branch decomposition (T, δ) of $F \cup var(F)$
Output: a structured d-DNNF computing F

// initialization, precomputing shapes
for v in T
    compute $proj(\overline{F_v}, X_v)$ and $proj(F_v, \overline{X_v})$
// compilation, leaf nodes
for v in $L(T)$
    if $\delta(v)$ in $var(F)$
        $x = \delta(v)$
        $S_x = \{ C \in F \mid x \in C \}$
        $S_{\neg x} = \{ C \in F \mid \neg x \in C \}$
        $\varphi_v(S_x, \emptyset) = x$
        $\varphi_v(S_{\neg x}, \emptyset) = \neg x$
    else
        $C = \delta(v)$
        $\varphi_v(\emptyset, \{C\}) = 1$
        $\varphi_v(\emptyset, \emptyset) = 0$
    mark v as processed
// compilation, inner nodes
while T contains an unprocessed node
    let v be an unprocessed node whose children v_1 and v_2 have been processed
    for (S_1, S_2, S') in $proj(\overline{F_{v_1}}, X_{v_1}) \times proj(\overline{F_{v_2}}, X_{v_2}) \times proj(F_v, \overline{X_v})$
        $S = S_1 \cup S_2$
        $S'_1 = S' \cup S_2$
        $S'_2 = S' \cup S_1$
        if $\varphi_v(S, S')$ has not been created
            // initialize $\varphi_v(S, S')$
            $\varphi_v(S, S') = 0$
        $\varphi_v(S, S') = \varphi_v(S, S') \vee (\varphi_{v_1}(S_1, S'_1) \wedge \varphi_{v_2}(S_2, S'_2))$
        propagate constants in $\varphi_v(S, S')$
    mark v as processed
return $\varphi_r(\emptyset, \emptyset)$

Proof (Sketch). Let F be a formula and let $\boldsymbol{T} = (T, (B_t)_{t \in V(T)})$ be a tree decomposition of its incidence graph such that $\boldsymbol{T}$ has width k. We can assume without loss of generality that T is binary (this can be achieved by copying nodes and bags of T). We construct a branch decomposition $\boldsymbol{T'} = (T', \delta)$ of $F \cup \mathit{var}(F)$ as follows: for every variable $x \in \mathit{var}(F)$ we introduce a vertex v_x and connect it to the node t of T such that t is closest to the root among nodes whose associated bags contain the variable x. For each clause $C \in F$ we add a vertex v_C in an analogous way. The result is a tree where every vertex has at most three neighbors. We obtain the desired branch decomposition $\boldsymbol{T'}$ by iteratively deleting all leaves not among the nodes v_x and v_C introduced in the first step and contracting paths to edges. We now claim the following.

- For every $v \in T'$, there are at most $k+1$ clauses in $\overline{F_v}$ that contain a variable from X_v. Each projection $\overline{F_v}(\tau)$ of an assignment $\tau : X_v \to \{0,1\}$ onto $\overline{F_v}$ is a subset of these clauses, so $|\mathit{proj}(\overline{F_v}, X_v)| \leq 2^{k+1}$.
- Symmetrically, for every $v \in T'$, there are at most $k+1$ variables in $\overline{X_v}$ that occur in a clause $C \in F_v$. It follows that $|\mathit{proj}(F_v, \overline{X_v})| \leq 2^{k+1}$ because there are at most 2^{k+1} assignments $\tau : \overline{X_v} \to \{0,1\}$.

That is, $\boldsymbol{T'}$ has PS-width at most 2^{k+1}. □

Combining Proposition 1 and Theorem 1, we obtain the following result.

Corollary 1. *A formula with n variables, m clauses, and incidence treewidth k can be compiled into a structured deterministic DNNF of size $O(8^k(n+m))$.*

Clique-width is a generalization of treewidth defined as follows. A *k-graph* is a pair (G, λ) consisting of a graph $G = (V(G), E(G))$ and a mapping $\lambda : V(G) \to \{1, \ldots, k\}$. We call $\lambda(v)$ the *label* of vertex v. We define the following operations for constructing k-graphs:

(i) For $i \in \{1, \ldots, k\}$, we write $\bullet_i$ for the k-graph (G, λ) where G contains a single isolated vertex v and $\lambda(v) = i$.
(ii) Let $i, j \in \{1, \ldots, k\}$ such that $i \neq j$, and let $\boldsymbol{G} = (G, \lambda)$ be a k-graph. Then $\rho_{i \to j}(\boldsymbol{G}) = (G, \lambda')$, where $\lambda'(v) = \lambda(v)$ if $\lambda(v) \neq i$, and $\lambda'(v) = j$ if $\lambda(v) = i$, for each vertex $v \in V(G)$.
(iii) Let $i, j \in \{1, \ldots, k\}$ such that $i \neq j$, and let $\boldsymbol{G} = (G, \lambda)$ be a k-graph. Then $\eta_{i,j}(\boldsymbol{G}) = (G', \lambda)$, where G' is the graph such that $V(G') = V(G)$, and such that $E(G') = E(G) \cup \{\, vw \mid \lambda(v) = i, \lambda(w) = j \,\}$. That is, G' is obtained from G by adding an edge between any two vertices v and w such that v is labelled i and w is labelled j.
(iv) We write $\boldsymbol{G} \sqcup \boldsymbol{G'}$ to denote the disjoint union of two k-graphs $\boldsymbol{G} = (G, \lambda)$ and $\boldsymbol{G'} = (G', \lambda')$, that is, $\boldsymbol{G} \sqcup \boldsymbol{G'} = (G \sqcup G', \lambda \cup \lambda')$.

A *k-expression* is a well-formed expression using the symbols $\bullet_i$ (constant), $\rho_{i \to j}$, $\eta_{i,j}$ (both unary), and $\sqcup$ (binary). The k-graph associated with a k-expression t (and any k-graph isomorphic to it) is called the *value* of t. If a k-expression t has the value (G, λ) we say that t is a *k-expression of G*. The *clique-width* of a graph G is the minimum k such that there is a k-expression of G.

A formula with m clauses and incidence clique-width k has PS-width at most m^k [16]. In combination with Theorem 1, this gives the following result.[7]

Corollary 2. *A formula with n variables, m clauses, and incidence clique-width k can be compiled into a structured d-DNNF of size $O(m^{3k}(n+m))$.*

The *directed clique-width* of a directed graph is defined analogously to clique-width. If F is a CNF formula with a directed incidence graph[8] of directed clique-width k, then F has PS-width at most 4^k [2]. By combining this fact and Theorem 1, we obtain the following.

Corollary 3. *A formula with n variables, m clauses, and directed incidence clique-width k can be compiled into a structured d-DNNF of size $O(64^k(n+m))$.*

5 A Lower Bound for Clique-Width

Note that there is a qualitative difference in the size bounds of Corollary 1 and Corollary 3 on the one hand, and Corollary 2 on the other hand. If k is the value of a structural parameter of a formula with n variables and m clauses, then the former bound has the shape $2^{O(k)}(n+m)$, whereas the latter bound has the shape $m^{O(k)}(n+m)$. For small values of k and large values of n and m, bounds of the form $2^{O(k)}(n+m)$ are preferable to bounds of the form $m^{O(k)}(n+m)$.

In this section we will give evidence that the size bound of Corollary 2 is optimal qualitatively, so that the qualitative difference discussed above is unavoidable. To this end, we introduce the following notions from parameterized complexity.

A *parameterized problem* is a pair (P, κ) where P is a decision problem and $\kappa : \{0,1\} \to \mathbb{N}$ is a computable function associating every instance of P with a *parameter*. A parameterized problem (P, κ) is in the complexity class FPT, or *fixed-parameter tractable*, if there is an algorithm solving P in time $f(\kappa(x))|x|^c$ for every instance x, where $f : \mathbb{N} \to \mathbb{N}$ is a computable function and c is a constant. A parameterized problem (P, κ) is in the complexity class FPT/ppoly if there is an algorithm that, given an instance x of P and $f'(\kappa(x))|x|^{c'}$ advice bits, correctly solves x in time $f(\kappa(x))|x|^c$ where $f : \mathbb{N} \to \mathbb{N}$ and $f' : \mathbb{N} \to \mathbb{N}$ are computable functions and c, c' are constants [3].[9] Clearly, FPT is contained in FPT/ppoly.

Theorem 2. *Assume that* W[1] $\not\subseteq$ FPT/ppoly. *Then there is no computable function $f : \mathbb{N} \to \mathbb{N}$ and constant c such that for every CNF-formula F with n variables, m clauses and clique-width k there is a DNNF D such that F and D compute the same function and the size of D is at most $f(k)(n+m)^c$.*

[7] By the same token, the statement could also be proved for other structural parameters, like Boolean width, rank-width, or MIM-width [18, 16].

[8] The directed incidence graph is an orientation of the incidence graph encoding positive and negative occurrences of variables.

[9] Note that the advice given to the algorithm may only depend on $\kappa(x)$ and $|x|$ but not directly on x. Thus for two instances x and x' with $\kappa(x) = \kappa(x')$ and $|x| = |x'|$ the algorithm is given *the same* advice string.

The assumption W[1] $\not\subseteq$ FPT/ppoly is a parameterized analogue of the assumption NP $\not\subseteq$ P/poly in classical complexity; the latter is known to hold unless the Polynomial Hierarchy collapses to the second level [9]. Although W[1] $\not\subseteq$ FPT/ppoly is a stronger assumption than W[1] $\not\subseteq$ FPT, which is standard in parameterized complexity, we still consider it plausible, since it is not clear how nonuniformity should help in solving W[1]-hard problems.

Proof (of Theorem 2). We use a reduction from partitioned clique to the satisfiability problem presented in [10]. The partitioned clique problem is to decide, given a k-partite graph G whose color classes all have the same size, whether G has a clique of size k, i.e. containing a vertex from every color class. Here, k is the parameter of the problem instance. The partitioned clique problem is W[1]-complete under fixed-parameter tractable many-one reductions; see [7] for more details.

In [10], it is shown (Theorem 4 and Corollary 1) that, given a k-partite graph $G = (V_1, \ldots, V_k, E)$ with the same number of vertices in each color class, one can construct a CNF formula F_G such that the incidence graph of F_G has clique-width at most $k+4$ and the size of F_G is polynomial in the size of G, and such that the formula F_G has a satisfying assignment if and only if G has a clique of size k. If $V_i = \{v_1^i, \ldots, v_n^i\}$, the formula contains the variables V_i for each $1 \leq i \leq k$. For each pair (u, v) such that $v \in V_i$, $u \in V_j$ $(i \neq j)$, and such that $uv \notin E$, the formula F_G contains the clause $C_{u,v} = \{\neg u, \neg v\} \cup \{\, w \mid w \in (V_i \cup V_j) \setminus \{u, v\}\,\}$. The idea is that the variables v_j^i mapped to 1 by a satisfying assignment of F_G correspond to a partitioned clique of G. The clauses $C_{u,v}$ are padded with the remaining variables in order to keep the clique-width of F_G's incidence graph small; to make sure that a clause $C_{u,v}$ cannot be satisfied by these extra variables when u and v are both assigned to 1, a "selection gadget" is attached to each color class V_i. This gadget (which we will not describe here) guarantees that each satisfying assignment of F_G maps exactly one of the variables in each color class V_i to 1.

We modify this construction in the following way. Let $G_n^k = (V_1, \ldots, V_k, \emptyset)$ denote the empty k-partite graph with n vertices in each color class. The formula $F_{G_n^k}$ contains a clause $C_{u,v}$ for each pair of variables $u \in V_i$, $v \in V_j$ $(i \neq j)$, as G_n^k does not contain any edges. Starting from $F_{G_n^k}$, we construct a new formula $F_{k,n}$ by adding a distinct relaxation variable $x_{u,v}$ to each clause $C_{u,v}$. These variables allow us to "switch clauses on and off" as needed. Adding the variable $x_{u,v}$ to the clause $C_{u,v}$ corresponds to adding a vertex $x_{u,v}$ and a "dangling edge" $\{x_{u,v}, C_{u,v}\}$ to the incidence graph of $F_{G_n^k}$. This can be done for each clause $C_{u,v}$ while increasing the clique-width of the incidence graph by at most 3, as can be seen from the following argument. Consider a $(k+4)$-expression t of the incidence graph $I(F_{G_n^k})$ of $F_{G_n^k}$. For each clause $C_{u,v}$, the expression t contains a subexpression $\bullet_j$ that introduces the vertex $C_{u,v}$ with some label $j \in \{1, \ldots, k+4\}$. Using fresh labels, we replace each such subexpression with the expression $\rho_{k+6 \to k+7}(\rho_{k+5 \to j}(\eta_{k+5,k+6}(\bullet_{k+5} \sqcup \bullet_{k+6})))$. That is, instead of introducing $C_{u,v}$ with label j, we first introduce it with label $k+5$, along with the vertex $x_{u,v}$, which we label with $k+6$. We then create the edge

$\{C_{u,v}, x_{u,v}\}$ and before relabelling both vertices: the vertex $C_{u,v}$ gets its original label j, while vertex $x_{u,v}$ is assigned an auxiliary label $k+7$ to make sure it does not become the endpoint of further edges. The resulting expression is a $(k+7)$-expression of $I(F_{k,n})$.

Given a k-partite graph G with n vertices in each color class, the formula F_G can be obtained from $F_{k,n}$ by assigning the relaxation variables: simply set $x_{u,v}$ to 1 if uv is an edge of G, and to 0 if uv is not an edge of G.

Now assume, by way of contradiction, that there is a function $f : \mathbb{N} \to \mathbb{N}$ and a constant c such that for every CNF formula F with n variables, m clauses and clique-width k, there is a DNNF D such that F and D compute the same function and the size of D is at most $f(k)(n+m)^c$. Then in particular, there is a constant c' such that for every n and k, there is a DNNF $D_{k,n}$ of size $f(k)n^{c'}$ that computes $F_{k,n}$.

We now describe a non-uniform algorithm for the partitioned clique problem: Given a k-partite graph G with n vertices in each color class, the advice string is a desciption of $D_{k,n}$. The algorithm first sets the relaxation variables so as to get a DNNF D_G computing F_G. This can be done in linear time [4]. The graph G has a k-clique if and only if D_G is satisfiable. Since checking satisfiability of DNNF can be done in linear time [4], this gives the desired algorithm. It follows that the partitioned clique problem, and hence every problem in W[1], is in FPT/ppoly, which is a contradiction to the assumption of the lemma. We conclude that DNNFs of the desired size cannot exist if W[1] $\not\subseteq$ FPT/ppoly. □

6 Conclusion

We demonstrated how dynamic programming algorithms for #SAT [17, 16] can be modified to construct structured d-DNNF representations of CNF formulas. This observation allowed us to prove an upper bound on the size of structured d-DNNF representations of CNFs in terms of a parameter called PS-width [16]. We showed that this bound translates into new upper bounds in terms of parameters such as the treewidth and the clique-width of the incidence graph. We also provided evidence that the upper bound in terms of incidence clique-width cannot be substantially improved, even for general DNNFs.

The d-DNNFs generated by our compilation algorithm do not necessarily fall into the more restricted subclass of *decision DNNFs*. We do not know whether this is an artifact of our methods or due to an inherent limitation of decision DNNFs. In particular, we would like to know if CNF formulas can be compiled into decision DNNFs of size exponential only in their incidence treewidth. Finally, it would be interesting to compare PS-width to other recently proposed width measures of CNF-formulas, such as CV-width [11] and decision-width [12].

Acknowledgments. The first and fourth author were supported by the FWF Austrian Science Fund (P26200). The second author was supported by the ANR Blanc AGGREG reference ANR-14-CE25-0017-01. The third author was supported by the ANR Blanc International ALCOCLAN.

References

1. Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. Expander CNFs have exponential DNNF size. *CoRR*, abs/1411.1995, 2014.
2. Johann Brault-Baron, Florent Capelli, and Stefan Mengel. Understanding model counting for beta-acyclic CNF-formulas. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015*, volume 30 of *LIPIcs*, pages 143–156. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.
3. Hubie Chen. Parameterized compilability. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, pages 412–417, 2005.
4. Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001.
5. Adnan Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *Journal of Applied Non-Classical Logics*, 11(1-2):11–34, 2001.
6. Adnan Darwiche and Pierre Marquis. A Knowledge Compilation Map. *J. Artif. Intell. Res. (JAIR)*, 17:229–264, 2002.
7. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag New York Inc, 2006.
8. Petr Hlinený, Sang-il Oum, Detlef Seese, and Georg Gottlob. Width parameters beyond tree-width and their applications. *Comput. J.*, 51(3):326–362, 2008.
9. Richard M. Karp and Richard J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of STOC 1980*, pages 302–309. ACM, 1980.
10. Sebastian Ordyniak, Daniël Paulusma, and Stefan Szeider. Satisfiability of acyclic and almost acyclic CNF formulas. *Theoretical Computer Science*, 481:85–99, 2013.
11. Umut Oztok and Adnan Darwiche. CV-width: A New Complexity Parameter for CNFs. In *ECAI 2014 - 21st European Conference on Artificial Intelligence*, pages 675–680, 2014.
12. Umut Oztok and Adnan Darwiche. On Compiling CNF into Decision-DNNF. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014*, pages 42–57, 2014.
13. Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 1*, AAAI'08, pages 517–522. AAAI Press, 2008.
14. Knot Pipatsrisawat and Adnan Darwiche. Top-down algorithms for constructing structured DNNF: theoretical and practical implications. In *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 3–8. IOS Press, 2010.
15. Igor Razgon and Justyna Petke. Cliquewidth and knowledge compilation. In *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference*, pages 335–350, 2013.
16. S. Hortemo Sæther, J.A. Telle, and M. Vatshelle. Solving MaxSAT and #SAT on structured CNF formulas. In *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference*, pages 16–31, 2014.
17. Friedrich Slivovsky and Stefan Szeider. Model counting for formulas of bounded clique-width. In *Algorithms and Computation - 24th International Symposium, ISAAC 2013*, pages 677–687, 2013.

18. M. Vatshelle. *New Width Parameters of Graphs*. PhD thesis, University of Bergen, 2012.