Compilation and Fast Model Counting beyond CNF

Alexis de Colnet, **Stefan Szeider** and **Tianwei Zhang**
Algorithms and Complexity Group, TU Wien, Vienna, Austria
{decolnet,sz,zhangtw}@ac.tuwien.ac.at

Abstract

Circuits in deterministic decomposable negation normal form (d-DNNF) are representations of Boolean functions that enable linear-time model counting. This paper strengthens our theoretical knowledge of what classes of functions can be efficiently transformed, or *compiled*, into d-DNNF. Our main contribution is the fixed-parameter tractable (FPT) compilation of conjunctions of specific constraints parameterized by incidence treewidth. This subsumes the known result for CNF. The constraints in question are all functions representable by constant-width ordered binary decision diagrams (OBDDs) for *all* variable orderings. For instance, this includes parity constraints and cardinality constraints with constant threshold. The running time of the FPT compilation is singly exponential in the incidence treewidth but hides large constants in the exponent. To balance that, we give a more efficient FPT algorithm for model counting that applies to a sub-family of the constraints and does not require compilation.

1 Introduction

Knowledge compilation is a domain of computer sciences that studies the different ways to represent functions. Classes of representations, or *languages*, have been invented where specific problems become tractable. In particular, many Boolean languages have been created that support polynomial-time model counting, that is, determining the number of truth assignments on which a function $f : \{0,1\}^n \to \{0,1\}$ evaluates to 1. In practice, several model counters for CNF formulas are transforming, or *compiling*, their inputs into the language d-DNNF of circuits in deterministic decomposable negation normal form, where model counting is feasible in linear time (in the size of the circuits). For instance, `C2D` [Darwiche, 2004], `DSharp` [Muise *et al.*, 2012], `miniC2D` [Oztok and Darwiche, 2015], and `D4` [Lagniez and Marquis, 2017] all compile CNF formulas into (sublanguages) of d-DNNF. Other model counters do not explicitly compile the CNF but can be seen as compilers in disguise [Kiesel and Eiter, 2023].

Compilation to d-DNNF is often hard in the sense that the d-DNNF circuit representations of many functions provably have exponential size, even for functions that belong to fragments where model counting is tractable such as systems of Boolean linear equations [de Colnet and Mengel, 2023]. On a more positive note, compiling from CNF to d-DNNF is fixed-parameter tractable (FPT) when the parameter is the treewidth of the CNF's primal graph [Huang and Darwiche, 2007] and when it is the treewidth of its incidence graph [Bova *et al.*, 2015]. To further understand when the input is easy to compile, one possibility is to find new CNF parameters that dominate treewidth and yet enable FPT compilation to d-DNNF, for instance, decision-width [Oztok and Darwiche, 2014b], CV-width [Oztok and Darwiche, 2014a] and PS-width [Bova *et al.*, 2015]. Another direction is to part from the CNF input and study the parameterized compilability of functions given in a different format such as DNF or a general Boolean circuit [Amarilli *et al.*, 2020]. In this paper, we follow the latter direction: we investigate the FPT compilation of systems (i.e., conjunctions) of Boolean constraints that are not just CNF. We see each CNF formula as a system of disjunctive clauses and identify other types of constraints that can be added to the system while ensuring an FPT compilation to d-DNNF parameterized by its incidence treewidth k. While compiling general systems of constraints to d-DNNF cannot be FPT parameterized k unless FPT $=$ W[1] (since even satisfiability is W[1]-hard in this setting [Samer and Szeider, 2010]), we show that there is an FPT compilation algorithm for specific constraints. In a first approximation, these are the constraints that can only be in a constant number of states when assigning any subset of the variables in any way. Parity constraints (i.e., constraints of the form "the sum of the variables is odd/even") are a good example where there are only two possible states: given a parity constraint and a subset of its variables, assigning these variables in any way either results in the odd parity constraint or the even parity constraint on the remaining variables. Our main result is the following.

Theorem 1. *Let w be a constant integer and $\mathcal{C}$ be a class of constraints such that, for every $c \in \mathcal{C}$, and every subset Y of c's variables, at most w different constraints can be obtained from c by assigning variables in Y in any way. There is an algorithm that, given a system F of constraints from $\mathcal{C}$, returns in time $2^{O(k)}poly(|F| + |var(F)|))$ a d-DNNF circuit for F, where k is the treewidth of the incidence graph of F.*

The neatest proof we could find for this result on the compilation of non-CNF systems—which we present in this paper—goes through constructing CNF encodings of the systems and then compiling these encodings to d-DNNF.

Theorem 1 includes a rather lengthy characterization of our constraints. For the proof, we use a "more advanced" characterization in knowledge compilation terms. The class $\mathcal{C}$ will be a class of constraints that are w-*slim* for the language OBDD or the language SDNNF. This notion is inspired by Wegener's book [2000, Section 5.3] and introduced in detail in Section 3. We are not the first to consider OBDD constraints to prove FPT results, for instance Chen and Grohe [2010] use similar constraints to prove FPT results for the constraint satisfaction problem. We prove our theorem in Section 4 with a more accurate running time where the exponential part also depends on w. Finally, in Section 5, we show that we can count models faster than by compilation if we put additional restrictions on the constraints. One of our results is based on bounded state-size (introduced in Section 3), which captures important constraints like XORs, modulo, and cardinality constraints.

Theorem 2. *Let F be a system of constraints whose maximum state-size is w. There is an algorithm that, given F and a nice width-k tree decomposition of the incidence graph of F, counts the models of F in time $O(w^{2k}(|F|+|var(F)|))$ in the unit-cost model.*

2 Preliminaries

A Boolean variable x takes a value in $\{0,1\}$. *Literals* are denoted by x and $\bar{x}$. An *assignment* to a set X of Boolean variables is a mapping from X to $\{0,1\}$, and a *Boolean function* f over X maps the assignments to X to $\{0,1\}$. For us, *constraints* are just Boolean functions that appear in systems (conjunctions of constraints). *CNF formulas* are systems of constraints whose constraints are *clauses* (disjunctions of literals). We write $|F|$ for the number of constraints in the system F. Boolean circuits are another way to represent functions. The size of a circuit D, denoted $|D|$, is its number of gates and connectors. We denote by $gates(D)$ the set of gates of D. For $g \in gates(D)$, D_g is the subcircuit of D whose output gate is g. We write $f(X)$, $D(X)$, and $F(X)$ to indicate that the variable set of a function, circuit, or system is X, respectively. If this set is not explicit, we use $var(f)$, $var(D)$, $var(F)$.

2.1 Treewidth and Tree Decompositions

A *tree decomposition* $\mathcal{T}$ of a graph G is a pair (T,b) with T a tree and $b : V(T) \to \mathcal{P}(V(G))$ a *bag* function such that (1) $\bigcup_{t\in V(T)} b(t) = V(G)$, (2) for all $uv \in E(G)$, there is $t \in T$ such that $\{u,v\} \subseteq b(t)$, and (3) for all $v \in V(G), T[t \mid v \in b(t)]$ is connected. The *width* of $\mathcal{T}$ is $\max_{t\in T} |b(t)|$. The *treewidth* of G is $tw(G) := \min_{\mathcal{T}} \max_{t\in T} |b(t)| - 1$ where $\mathcal{T}$ ranges over all tree decompositions of G. A tree decomposition is *nice* when it is rooted and when each node $t \in V(T)$ is of one of the following three types: a join node, an introduce node, or a forget node. t is a *join node* if it has two children t_1 and t_2 and $b(t) = b(t_1) = b(t_2)$. t is an *introduce node* for $v \in V(G)$ if it has a single child t' and $v \notin b(t')$ and $b(t) = b(t') \cup \{v\}$. t is a *forget node* for $v \in V(G)$ if it has a single child t' and $v \notin b(t)$ and $b(t') = b(t) \cup \{v\}$. In addition, the bag of the root node is empty. Every tree decomposition can be made nice without increasing its width.

The *incidence graph* of a system F of constraints, denoted by G_F, is the graph whose vertices are the constraints and the variables of F and such that there is an edge between a constraint c and a variable x if and only if $x \in var(c)$. The *incidence treewidth* of F is $tw_i(F) := tw(G_F)$.

2.2 OBDDs and SDNNF Circuits

(d-)SDNNF Circuits. A variable tree (vtree) τ over a set X of variables is a rooted binary tree whose leaves are in bijection with X. For every $t \in V(\tau)$, $var(t)$ is the set of variables on the leaves below t. A circuit D in structured-decomposable negation normal form (SDNNF) [Pipatsrisawat and Darwiche, 2008] is a Boolean circuit with literal inputs, whose gates are binary $\vee$-gates and binary $\wedge$-gates, and such that there exist a vtree τ over $var(D)$ and a mapping $\lambda : gates(D) \to V(\tau)$ verifying the following:

- For every $g \in gates(D)$, $var(D_g) \subseteq var(\lambda(g))$.
- For every $\vee$-gate $g = g_1 \vee g_2$, $\lambda(g) = \lambda(g_1) = \lambda(g_2)$.
- For every $\wedge$-gate $g = g_1 \wedge g_2$, $\lambda(g)$ has two children t_1 and t_2 and $\lambda(g_1)$ is below t_1 and $\lambda(g_2)$ is below t_2.

We say that D is *structured by* (τ,λ) and sometimes omit λ. An example of an SDNNF circuit is shown in Figure 2b with the mapping λ given by the color code. An SDNNF circuit D is deterministic (d-SDNNF) when for all $\vee$-gates $g = g_1 \vee g_2$, we have $D_{g_1}^{-1}(1) \cap D_{g_2}^{-1} = \emptyset$. Counting the models of D, i.e., finding $|D^{-1}(1)|$, is tractable when D is a d-SDNNF circuit.

OBDDs. A binary decision diagram (BDD) is a directed acyclic graph with a single source, two sinks labeled 0 and 1, whose internal nodes are decision nodes with two distinct children and labeled by variables. A node v labeled by variable x and with children v_0 and v_1 is recursively interpreted as a Boolean function $v = (\bar{x} \wedge v_0) \vee (x \wedge v_1)$. Every assignment α corresponds to a path in the DAG. Starting from the root, assuming α's path reaches v follows v_0 if $\alpha(x) = 0$ and v_1 otherwise. The assignments satisfying the BDD are exactly those whose paths reach the sink 1. For a total order π on the variables, a π-Ordered BDD (π-OBDD) is a BDD whose variables appear at most once along every path from the root to a sink and always in an order consistent with π. A π-OBDD is *complete* if every path contains all the variables. d-SDNNF circuits generalize OBDDs in the sense that there is a simple linear-time rewriting that transforms OBDDs into d-SDNNF circuits structured by *linear vtrees* (i.e., vtrees whose internal nodes all have a leaf child).

Width measures. The *width* of an OBDD is the maximum number of nodes labeled by the same variable. Note that making an OBDD complete, while feasible in linear time, can also increase the width by a linear factor [Bollig and Wegener, 2000]. The *width* of an SDNNF circuit structured by (T,λ) is defined as $\max_{t\in V(T)} |\lambda^{-1}(t)|$. This definition differs from that of Capelli and Mengel [2019] but is more convenient for stating our results.

3 Slim Functions and STS

In this paper, constraints are *global constraints* [van Hoeve and Katriel, 2006]. So we know the type (e.g., clauses, parity constraints, cardinality constraints) of every constraint we manipulate.

CNF formulas are known to be FPT compilable to d-SDNNF parameterized by their incidence treewidth.

Theorem 3 (Bova *et al.*, 2015)**.** *There is an algorithm that transforms any CNF formula F into an equivalent d-SDNNF circuit in time $2^{O(tw_i(F))}poly(|F| + |var(F)|)$.*

The result leans on the fact that for every clause c, every subset $Y \subseteq var(c)$, and every assignment α to Y, the clause $c|\alpha$ where variables are assigned as in α can be in only two states: either it is satisfied, or it is the projection of c onto the complement of Y. Note that α falsifying c falls into the second case ($Y = var(c)$ and $c|\alpha$ is the empty clause). Our intuition is that FPT compilation to d-SDNNF should be possible for systems of constraints with a similar property: when subject to any assignment to any fixed set Y, every constraint of the system can only be in a few states. For example, consider an XOR constraint $c : x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 = 0$ and $Y = \{x_1, x_2, x_3\}$. Then $c|\alpha$ is either the odd parity constraint $x_4 \oplus x_5 = 1$, or the even parity constraint $x_4 \oplus x_5 = 0$.

3.1 Slim Functions for Complete OBDDs and SDNNFs

One can consider a *complete* OBDD is as state diagram where the values of the variables are read in a predefined order. The width of the complete OBDD is then the maximum number of states reachable after having set any number of variables in that order. The property we want for our constraints translates into the requirement that the smallest possible width in one of their π-OBDD representations is bounded for *all* the variable orders π. In the following, $h : \mathbb{N} \to \mathbb{R}$ is a real function.

Definition 4. *A class $\mathcal{F}$ of Boolean functions is* h-slim for complete OBDDs *when, for every n-variables function $f \in \mathcal{F}$ and every total order π of $var(f)$, there is a complete π-OBDD of width at most $h(n)$ computing f.*

The notions of $O(1)$-slim functions, $O(n)$-slim functions, etc. should be self-explanatory. Given a fixed constant w, we talk of w-slim functions when h is the constant w function. When h is an unknown polynomial, the h-slim functions for complete OBDDs coincide with the *nice functions* from Wegener's book [2000, Section 5.3]. Our proofs below rely on SDNNF representations rather than OBDD representations. The concept of h-slim functions generalizes to SDNNF.

Definition 5. *A class $\mathcal{F}$ of Boolean functions is* h-slim for complete SDNNFs *when, for every n-variables function $f \in \mathcal{F}$ and vtree T over $var(f)$, there is a complete SDNNF of width at most $h(n)$ and with vtree T that computes f.*

Given the rewriting of complete OBDDs into complete d-SDNNFs, it is immediate that having a small OBDD-width for every variable ordering implies having a small SDNNF-width for every *linear* vtree. However, generalizing to *all* possible vtrees is not straightforward.

Lemma 6. *Let $\mathcal{F}$ be a class of functions.*

• If $\mathcal{F}$ is h-slim for complete OBDDs, then it is $O(h^3)$-slim for complete SDNNFs.

• If $\mathcal{F}$ is h-slim for complete SDNNFs, then it is $2^{2^{O(h)}}$-slim for complete OBDDs.

The doubly exponential upper bound could be a bit loose but cannot be decreased below exponential due to the following.

Lemma 7. *There are functions computed by complete SDNNFs of width $O(n)$ for every vtree, but that are only computed by OBDDs and d-SDNNFs of width $2^{\Omega(n)}/n$.*

Let us now restate our main result, i.e., Theorem 1, using constraints that belong to families of functions $O(1)$-slim for complete SDNNFs. Recall that our constraints are global. We will need the following assumption.

Assumption 8. *For every type of constraint appearing in our system, we have a polynomial-time algorithm to compile every constraint of this type into a minimal-width complete SDNNF for any given vtree.*

Theorem 9. *Let $w \in \mathbb{N}$ be a fixed constant. Under assumption 8, there is an algorithm that, given a system F of constraints that are all w-slim for complete SDNNFs, constructs in time $2^{O(w \cdot tw_i(F))}poly(|F| + |var(F)| + w)$ a circuit in d-SDNNF that computes F.*

In Table 1, we give several families of constraints along with upper bounds on the smallest h functions for which they are h-slim for complete OBDDs and SDNNFs. The proof of the correctness of these values for h appear in the long version of the paper. Assumption 8 is reasonable for most constraint types of the table 1, in particular for clauses, XORs, sums modulo, cardinality and threshold constraints.

3.2 Commutative State Transitions Systems

Definition 10. *A* state transition system (STS) *consists of a set of states S, two transition functions $f_0, f_1 : S \to S$, a starting state $s_0 \in S$ and a set of accepting states $T \subseteq S$. The* extended transition function $\delta : \{0,1\}^* \to S$ *is defined as:*

$$\delta(\epsilon) := s_0, \quad \delta(l0) := f_0(\delta(l)), \quad \delta(l1) := f_1(\delta(l)).$$

The STS is finite *if S is finite, and* commutative *if $f_0(f_1(s)) = f_1(f_0(s))$ for all $s \in S$. We write* CSTS *for commutative STS.*

Note that the extended transition function δ for a CSTS has the following property: for any $l_1, l_1', l_2, l_2' \in \{0,1\}^*$, if $\delta(l_1) = \delta(l_1')$ and $\delta(l_2) = \delta(l_2')$, then $\delta(l_1 l_2) = \delta(l_1' l_2')$. This is because $\delta(l_1 l_2) = \delta(l_1' l_2) = \delta(l_2 l_1') = \delta(l_2' l_1') = \delta(l_1' l_2')$. If we assume that for every $s \in S$ there is $l \in \{0,1\}^*$ such that $\delta(l) = s$, that is, the CSTS is in a sense connected, we can then define a binary operation $+ : S \times S \to S$ as follows: for any $s_1, s_2, s \in S$, $s_1 + s_2 = s$ if there is $l_1, l_2 \in \{0,1\}^*$ such that $\delta(l_1) = s_1, \delta(l_2) = s_2$, and $\delta(l_1 l_2) = s$. Note that for all $s, s', s'' \in S$, we have $s + s_0 = s$, $s + s' = s' + s$ and $(s + s') + s'' = s + (s' + s'')$. That is, $(S, +, s_0)$ forms a commutative monoid.

Given a function $f : \{0,1\}^n \to \{0,1\}$, we say an STS *describes* f if, for every $l \in \{0,1\}^n$, $\delta(l) \in T$ if and only if $f(l) = 1$. Let $f, g : \{0,1\}^n \to \{0,1\}$ be two functions. We say that a CSTS describes f *modulo literal-flipping* if it

describes g and there exists a literal-flipping function ϕ that sends x_i to either x_i or $\bar{x}_i$ such that $f = g \circ \phi$. We call the minimum number of states of a CSTS that describes f modulo literal-flipping the *state size* of f.

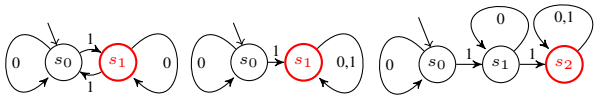


Figure 1: CSTS for XOR, OR and "$x_1 + \cdots + x_n \geq 2$" constraints.

Lemma 11. *Let f be a symmetric (resp. literal-symmetric) function. If f admits a complete OBDD representation of width w, then f is described (resp. described modulo literal-flipping) by a CSTS with at most $(w+1)^2/4$ states.*

4 FPT Compilation of Systems of $O(1)$-Slim Constraints

In this section, we explain the proof of Theorem 9. The proof is in three steps: (1) find a CNF encoding $H(X,Z)$ of $F(X)$ whose incidence treewidth is at most $O(w \cdot tw_i(F))$, (2) compile $H(X,Z)$ to d-SDNNF using Theorem 3, and (3) existentially forget the auxiliary variables Z from the resulting d-SDNNF circuit. We do not have to take care of (2). For (3), *forgetting* the variables Z from a d-SDNNF circuit $D(X,Z)$ means finding another d-SDNNF circuit computing $\exists Z.D(X,Z)$. Forgetting many variables from a d-SDNNF circuit is generally intractable [Pipatsrisawat and Darwiche, 2008]. However the operation is tractable when the Z-variables are completely defined in terms of the X-variables. We make sure to be in this situation by using only Tseitin CNF encodings to generate $H(X,Z)$.

4.1 The Tseitin CNF Encoding

Let $D(X)$ be a Boolean circuit whose internal gates are binary $\vee$-gates and binary $\wedge$-gates. Let $gates^*(D)$ be the set of its internal gates. Every gate g is associated with a variable z_g. Let $Z = \{z_g \mid g \in gates^*(D)\}$. The Tseitin encoding of D is the CNF

$$H_D^{Tseitin}(X,Z) := \bigwedge_{g \in gates^*(D)} Tseitin(g)$$

where, for every gate g, if $g = g_1 \wedge g_2$ then $Tseitin(g) := (\bar{z}_g \vee z_{g_1}) \wedge (\bar{z}_g \vee z_{g_2}) \wedge (\bar{z}_{g_1} \vee \bar{z}_{g_2} \vee z_g)$, if $g = g_1 \vee g_2$ then $Tseitin(g) := (\bar{z}_g \vee z_{g_1} \vee z_{g_2}) \wedge (\bar{z}_{g_1} \vee z_g) \wedge (\bar{z}_{g_2} \vee g)$, and if g is an input gate, so if $g = x, \bar{x}, 0$ or 1, then $z_g = g$. The formula $H_D^{Tseitin}$ is a CNF encoding of D in the sense that

$$\exists Z.H_D^{Tseitin}(X,Z) \equiv D(X).$$

We often drop the *Tseitin* superscript in the rest of the paper.

Lemma 12. *Let $f(X)$ be a Boolean function and $D_1(X_1), \ldots, D_m(X_m)$ be Boolean circuits. There is a linear-time algorithm that, given a d-SDNNF circuit D computing $f(X) \wedge H_{D_1}^{Tseitin}(X_1,Z_1) \wedge \cdots \wedge H_{D_m}^{Tseitin}(X_1,Z_m)$, where the sets $Z_1, \ldots, Z_m$ are pairwise disjoint, returns a d-SDNNF circuit computing $f(X) \wedge D_1(X_1) \wedge \cdots \wedge D_m(X_m)$.*

4.2 Proof Sketch of Theorem 9

The only real hurdle in the proof of Theorem 9 is (1), that is, encoding the system of constraints $F(X) = c_1(X_1) \wedge \cdots \wedge c_m(X_m)$ into the CNF $H(X,Z)$ while controlling the incidence treewidth. For every $i \leq m$ we do the following. First, using a certain tree decomposition of F's incidence graph, we construct a vtree τ_{c_i} over X_i. Second, we construct an SDNNF circuit $D_i(X_i)$ structured by τ_{c_i} that computes $c_i(X_i)$. Next, we encode D_i into a CNF formula $H_i(X_i,Z_i)$ using the Tseitin encoding and a set Z_i of fresh auxiliary variables. The CNF encoding of F is then $H(X,Z) := H_1(X_1,Z_1) \wedge \cdots \wedge H_m(X_m,Z_m)$. Given the vtrees τ_{c_i}, it should be quite clear that $H(X,Z)$ can be constructed in polynomial time under Assumption 8, and that the Z variables can be forgotten using Lemma 12. So we only need to justify that the τ_{c_i} can be found efficiently, and that the incidence treewidth of $H(X,Z)$ can be controlled.

Constructing the vtrees. To construct the vtree τ_c for $c \in F$, we need a tree decomposition (t.d.) of G_F with specific properties. We do not give the details here but, basically, we use nice t.d. in which a few bags are cloned. Roughly put, τ_c shows how the variables of c appear relative to each other in the t.d. For instance, in the t.d. shown Figure 2c (not of the type used in the proof, but sufficient for the example) for the incidence graph of Figure 2a, we have x_2, x_3 and x_4, x_5 are introduced in a different branches, and this yields the vtree τ_{c_2} shown Figure 2b. τ_c is found using the following lemma.

Lemma 13. *Every tree decomposition (t.d.) of G_F can be transformed in linear time into another t.d. (T,b) of G_F, that has the same width, and that can be used to find in linear time a vtree τ_c over $var(c)$ for every $c \in F$. Each vtree τ_c has a t.d. (T,b_c) of width 2 such that, for all $t \in V(T)$,*

i. if $c \notin b(t)$ then $b_c(t) = \emptyset$,

ii. if $c \in b(t)$ then $b_c(t) \cap b(t) \subseteq b(t) \cap var(c)$.

It may seem bizarre that Lemma 13 looks at t.d. for vtrees. The key point here is that the vtrees have width-2 t.d. with *the same underlying tree* T as the t.d. of G_F. For instance, a t.d. (T,b_{c_2}) for τ_{c_2} from Figure 2b using the same tree T as the t.d. from Figure 2c could use $b_{c_2}(t_1) = \{1\}$, $b_{c_2}(t_2) = \{1,3\}$, $b_{c_2}(t_3) = \{3, x_4\}$, $b_{c_2}(t_4) = \{3, x_5\}$, $b_{c_2}(t_7) = \{1,2\}$, $b_{c_2}(t_8) = \{2, x_3\}$, $b_{c_2}(t_9) = \{2, x_2\}$ and all the other bags empty.

Controlling the treewidth of the encoding. Thanks to the properties stated in points $i.$ and $ii.$ of Lemma 13, we can *merge* the t.d. (T,b_c) of τ_c with the t.d. (T,b) of G_F with a simple bag-wise union to get a t.d. of $G_F \cup \tau_c$ (the graph with vertex set $V(G_F) \cup V(\tau_c)$ and with edge set $E(G_F) \cup E(\tau_c)$).

Lemma 14. *Let G and G' be two graphs and let $V_s := V(G) \cap V(G')$. Let (T,b) and (T',b') tree decompositions of G and G', respectively. If $T = T'$ and if for all $t \in V(T)$ we have $b(t) \cap V_s \subseteq b'(t) \cap V_s$ then $(T, b \cup b')$ is a tree decomposition of $G \cup G'$.*

If we merge (T,b) with all the t.d. $(T,b_{c_1}), \ldots, (T,b_{c_m})$ obtained from Lemma 13 then we obtain a t.d. (T,b^*) of $G_F \cup \tau_{c_1} \cup \cdots \cup \tau_{c_m}$ of width at most $2 \cdot width(T,b)$ because, by property $i.$, new elements are added to a bag $b(t)$ only from

functions with constant-size one-sided CSTS modulo literal-flipping $\subset$ functions with constant-size CSTS modulo literal-flipping $=$ literal-symmetric functions $O(1)$-slim for OBDDs $\subset$ functions $O(1)$-slim for OBDDs $=$ functions $O(1)$-slim for SDNNFs

Name	Formal expression	h-slim for OBDDs	h-slim for SDNNFs	
Clauses	$x_1 \vee \cdots \vee \bar{x}_n$	$h = 2$	$h = O(1)$	representable by one-sided STS of constant size when $k = O(1)$
XORs	$x_1 \oplus \cdots \oplus x_n = 0$ or 1	$h = 2$	$h = O(1)$	
Sum modulo	$x_1 + \cdots + x_n = c \mod k$	$h = k$	$h = O(k^2)$	
Cardinality	$x_1 + \cdots + x_n \geq k$	$h = O(\min(n-k, k))$	$h = O(\min(n-k, k)^2)$	
Small scope constraints	$f(x_1, \ldots, x_k)$	$h = 2^k$	$h = O(2^k)$	
Threshold	$w_1 x_1 + \cdots + w_n x_n \geq \theta$ with $k := \lvert w_1 \rvert + \cdots + \lvert w_n \rvert$	$h = 2k + 1$	$h = O(k^2)$	
Symmetric functions	$f(x_1 + \cdots + x_n)$ for $f : \mathbb{N} \to \{0, 1\}$	$h = poly(n)$	$h = poly(n)$	
Literal-sym. functions	$f(\ell_1 + \cdots + \ell_n)$ for $f : \mathbb{N} \to \{0, 1\}$ and $\ell_i \in \{x_i, \bar{x}_i\}$	$h = poly(n)$	$h = poly(n)$	

Table 1: Families of constraints

the vtrees τ_c such that $c \in b(t)$, and each vtree contributes at most 2 elements.

Now suppose we have a complete DNNF circuit D_i computing c_i and respecting the vtree τ_{c_i}. Every gate of D_i and, by extension, every clause and every auxiliary variable of its Tseitin encoding, is mapped to a node of τ_{c_i}. We claim that, if for every bag $b^*(t)$ and every node s of τ_{c_i} in this bag, one replaces s by the clauses $clauses(\textit{Tseitin}(g))$ and the auxiliary variables used in these clauses for all gates g of D_i mapped to s, then the result is a t.d. (T, b^+) of $G_F \cup G_{H_1} \cup \cdots \cup G_{H_m} = G_F \cup G_H$. One then just have to remove the initial constraints of F from every bag to obtain a t.d. of G_H. Going back to our example, if D_2 is the SDNNF circuit shown in Figure 2b, one would replace all occurrences of the vtree node (2) in $b^*(t)$ by the clauses and variables used in the Tseitin encoding of the nodes squared in red in Figure 2b: each such gate g with input g_1 and g_2 will contribute the clauses of $\textit{Tseitin}(g)$ plus the variables z_g, z_{g_1} and z_{g_2} when they are not input variables of D_2.

Since every c_i is w-slim for complete SDNNFs, at most w gates of D_i is mapped to s, and the Tseitin encoding generates a constant number of clauses and auxiliary variables per gate of D_i. So we will have that $|b^+(t)| \in O(w|b^*(t)|) \in O(w|b(t)|)$. So, if we started from a t.d. of G_F of width $O(tw_i(F))$ (computable in time FPT parameterized by $tw_i(F)$ [Bodlaender, 1996]), then we end up with an CNF encoding H whose incidence treewidth is $O(w \cdot tw_i(F))$.

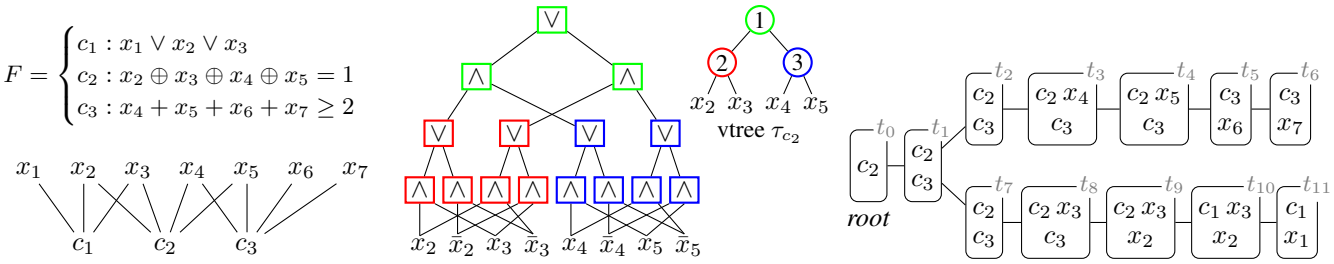
In the detailed proof, available in the long version of the paper, we directly merge the tree decompositions of the CNF encodings H_i, whose incidence treewidth we control using the following lemma for $D = D_i$ and $\tau = \tau_{c_i}$. This is equivalent to what we have just explained.

Lemma 15. *Let $D(Y)$ be a complete SDNNF circuit whose respecting (τ, λ). Let $\mathcal{T} = (T, b)$ be a tree decomposition of τ and let ϕ be defined as $\phi(x) = \{x\}$ for every variable x and as $\phi(s) = \bigcup_{g:\lambda(g)=s} var(\textit{Tseitin}(g)) \cup clauses(\textit{Tseitin}(g))$ for every internal node $s \in \tau$. Let $b'(t) = \bigcup_{s \in b(t)} \phi(s)$. Then (T, b') is a tree decomposition of $H_D^{\textit{Tseitin}}$'s incidence graph and has width $O(width(D)width(\mathcal{T}))$.*

4.3 Lower Bounds for Systems of Slim Constraints

Theorem 9 requires a compilation time with a $2^{O(w \cdot k)}$ component, with $k = tw_i(F)$ and w an upper bound on the width of complete SDNNF circuits representing the constraints. Can we get rid of w in the exponent? The answer is negative due to Lemma 7. One can use the hard functions from this lemma as constraints and show that w cannot be dropped from the exponent in the compilation time simply by considering systems made of a single constraint (so of incidence treewidth 1). The hard functions of Lemma 7 are specific monotone DNF formulas and the proof uses d-SDNNF lower bounds shown by Amarilli *et al.* [2020]. But DNF formulas can be converted in linear time in SDNNF circuits, so the hard functions for Lemma 7 admit SDNNF circuits of width polynomial in the number n of variables, whereas they only have OBDD representations of width exponential in n. Thus it could be possible that the best running time of an FPT compilation to d-SDNNF is of the form $f(w+k)poly(n+w)$ when w is the SDNNF-width, but also of the form $f(k)poly(n+w)$ when w is the OBDD-width. We can prove that, even for classes of functions that are w-slim for OBDDs, we cannot remove w from f's argument.

Theorem 16. *For every k, there exist systems of constraints*



(a) The incidence graph of F (b) An SDNNF circuit for c_2 (c) A tree decomposition of G_F

Figure 2

over n variables and of incidence treewidth $O(k)$ whose d-SDNNF representations all have size $(n/k)^{\Omega(k)}$, whereas these systems only comprise constraints that are $O(nk)$-slim for OBDDs.

The hard systems are CSP (constraints satisfaction problems) encodings of W[1]-hard problems. It seems unlikely that there exist FPT-reductions parameterized by incidence treewidth of W[1]-hard problems to CSP where all constraints are $O(1)$-slim for complete OBDDs. Indeed, using Theorem 9 plus the fact that deciding the satisfiability of a d-SDNNF circuit is straightforward, the existence of such reductions would imply FPT = W[1]. So, to find hard systems of constraints for the theorem, we started from a W[1]-hard problem with parameter k and reduced it to a CSP of incidence treewidth $O(k)$ and whose constraints are all $O(nk)$-slim for OBDDs (and not $O(1)$-slim). We have used the problem k-CLIQUE. The reduction is inspired by that used by Samer and Szeider [2010, Theorem 6] for k-INDEPENDENTSET.

5 Faster Model Counting

In this section, we show that model counting for three special cases of systems of constraints that are $O(1)$-slim for OBDDs can be done faster than by compiling the system as in Section 4 and counting from the compiled form. To this aim, we provide an FPT algorithm based on dynamic programming over a nice tree-decomposition of the incident graph of the system. We start with describing the algorithm for systems of literal-symmetric functions that are $O(1)$-slim for OBDDs, and then show that the speed of the algorithm can be improved when we consider even more particular cases, namely systems of one-sided constraints and systems of a mixture of CNF clauses and modulo constraints. All results in this section are stated in the *unit-cost model*, where every arithmetic operation is counted as an elementary operation.

5.1 Algorithm

In Section 3.2, we have shown that literal-symmetric functions that are $O(1)$-slim for OBDDs are precisely the functions described by a CSTS of constant size modulo literal-flipping. We use the latter representation and provide an FPT algorithm for model counting for a system of such constraints. The algorithm is based on dynamic programming over a nice tree decomposition of the incident graph of the system and is parameterized by the treewidth k of the incident graph of the system and the maximum state size w of its constraints. Throughout the entire section, we assume $w \geq 2$.

Theorem 17. *Let F be a system of constraints whose maximum state size is w, and let $\mathcal{T} = (T, b)$ be a nice width-k tree decomposition of the incidence graph G_F. Then, given F and $\mathcal{T}$, one can count the number of models of F with $O(w^{2k} \cdot |G_F|)$ elementary operations.*

The idea for the algorithm is to perform dynamic programming over the nice tree decomposition. Let F, $\mathcal{T}$ be as in Theorem 17. For each node t of T, let T_t denote the subtree of T rooted at t, let F_t and V_t be the set constraints and the set of variables that appear in the bags of T_t, respectively. We write $b_F(t) = b(t) \cap F$, and $b_V(t) = b(t) \cap var(F)$ for the set of constraints and variables in $b(t)$, respectively. For all $t \in T$, let $\widetilde{V}_t := V_t \setminus b_V(t)$. For each $c \in b_F(t)$, let $(S^c, f_0^c, f_1^c, s_0^c, T^c)$ be a minimal CSTS that describes c modulo literal-flipping and let ϕ^c be the literal-flipping that witnesses this. We fix this choice of CSTS before the start of the algorithm. Let δ^c be the corresponding extended transition function. For each $F' \subseteq F$, let $\mathcal{S}(F') := \{R \subseteq \bigcup_{c \in F'} \{c\} \times S^c \mid \text{for each } c \in F' \text{ there is a unique } s \text{ s.t. } (c, s) \in R\}$. Given $F' \subseteq F$, $c \in F'$ and $\bar{s} \in \mathcal{S}(F')$, we use s^c to denote the unique s such that $(c, s) \in \bar{s}$. Define a binary operation $+$ on $\mathcal{S}(F')$ by $\bar{s}_1 + \bar{s}_2 := \bar{s}$ where $s^c = s_1^c + s_2^c$. For each $c \in F$, we use c^+ to denote the set of variables x_i that occur in c with $\phi^c(x_i) = x_i$, and c^- the set of variables x_i that occur in c with $\phi^c(x_i) = \bar{x}_i$. For any partial assignment τ to the variables, let $q_0(\tau, c) := |\{v \in c^+ \mid \tau(v) = 0\}| + |\{v \in c^- \mid \tau(v) = 1\}|$ and $q_1(\tau, c) := |\{v \in c^+ \mid \tau(v) = 1\}| + |\{v \in c^- \mid \tau(v) = 0\}|$. Finally let $\delta^c[\tau] := \delta^c(1^{q_1(\tau,c)} 0^{q_0(\tau,c)})$.

Definition 18. *For each assignment $\alpha : b_V(t) \to \{0, 1\}$ and $\bar{s} \in \mathcal{S}(b_F(t))$, we define $N(t, \alpha, \bar{s})$ as the set of assignments $\tau : V_t \to \{0, 1\}$ for which the following conditions hold:* (1.) $\tau(v) = \alpha(v)$ for all variables $v \in b_V(t)$. (2.) For each $c \in b_F(t)$, $\delta^c[\tau|_{\widetilde{V}_t}] = s^c$. (3.) For each $c \in F_t \setminus b_F(t)$, $\delta^c[\tau] \in T^c$.

We represent the values of $n(t, \alpha, \bar{s}) = |N(t, \alpha, \bar{s})|$ for all $\alpha : b_V(t) \to \{0, 1\}$ and $\bar{s} \in \mathcal{S}(b_F(t))$ by a table M_t with $|b(t)| + 1$ columns and $2^{|b_V(t)|} \cdot |\mathcal{S}(b_F(t))| \leq 2^{|b_V(t)|} \cdot w^{|b_F(t)|} \leq w^k$ rows. The first $|b_V(t)|$ columns of M_t contain Boolean values encoding $\alpha(v)$ for variables $v \in b_V(t)$, fol-

lowed by $|b_F(t)|$ columns, one for each $c \in b_F(t)$ with the entry s^c. The last column contains the integer $n(t, \alpha, \bar{s})$.

The following lemmas show how to compute the table M_t for a node $t \in T$ depending on its type, assuming the tables of its children have already been computed.

Lemma 19. *Let t be a join node with children t_1, t_2. For each assignment $\alpha : b_V(t') \to \{0,1\}$, and $\bar{s} \in \mathcal{S}(b_F(t))$, we have*

$$n(t, \alpha, \bar{s}) = \sum_{\bar{s}_1 + \bar{s}_2 = \bar{s}} n(t_1, \alpha, \bar{s}_1) \cdot n(t_2, \alpha, \bar{s}_2).$$

Lemma 20. *Let t be an introduce node with child t'. For each truth assignment $\alpha : b_V(t') \to \{0,1\}$, and $\bar{s} \in \mathcal{S}(b_F(t'))$, we have the following equalities depending on whether it is a variable or a constraint that is introduced at t.*

1. *A variable v is introduced. We have*

$$n(t, \alpha \cup \{(v,0)\}, \bar{s}) = n(t, \alpha \cup \{(v,1)\}, \bar{s}) = n(t', \alpha, \bar{s}).$$

2. *A constraint c is introduced. We have*

$$n(t, \alpha, \bar{s} \cup \{(c,s)\}) = \begin{cases} n(t', \alpha, \bar{s}), & \text{if } s = s_0^c; \\ 0, & otherwise. \end{cases}$$

Lemma 21. *Let t be a forget node with child t'. For each truth assignment $\alpha : b_V(t) \to \{0,1\}$, and $\bar{s} \in \mathcal{S}(b_F(t))$, we have the following two equalities depending on whether it is a variable or a constraint that is forgotten at t.*

1. *A variable v is forgotten. Let α_0 and α_1 denote the extension to α that sets v to 0 and 1, respectively. We have*

$$n(t, \alpha, \bar{s}) = \sum_{\bar{s}' \in \sigma_0(v, \bar{s})} n(t', \alpha_0, \bar{s}') + \sum_{\bar{s}' \in \sigma_1(v, \bar{s})} n(t', \alpha_1, \bar{s}')$$

where $\sigma_1(v, \bar{s}')$ is the set

$$\left\{ \bar{s}' \in \mathcal{S}(b_F(t)) \;\middle|\; s^c = \begin{cases} f_0^c(s'^c) & v \in c^- \\ f_1^c(s'^c) & v \in c^+ \\ s'^c & v \notin var(c) \end{cases} \right\}$$

and σ_0 is defined similarly by exchanging f_0^c and f_1^c.

2. *A constraint c is forgotten. We have*

$$n(t, \alpha, \bar{s}) = \sum_{\delta^{c'}[\alpha] + s \in T^{c'}} n(t', \alpha, \bar{s} \cup \{(c', s)\}).$$

Lemma 22. *Let t be a leaf node. For each truth assignment $\alpha : b_V(t) \to \{0,1\}$ and $\bar{s} \in \mathcal{S}(b_F(t))$, we have*

$$n(t, \alpha, \bar{s}) = \begin{cases} 1, & \text{if } s^c = s_0^c \text{ for all } c \in b_F(t); \\ 0, & otherwise. \end{cases}$$

5.2 One-Sided and Modulo Constraints

A close inspection of the proof of Theorem 17 reveals that the bottleneck lies in the computation of the table M_t for a join node t. In fact, if we restrict ourselves to more particular cases of systems of constraints, namely systems of one-sided constraints and systems of disjunctive clauses and modulo constraints, we can speed-up the algorithm by computing the tables for join nodes faster, using advanced techniques like the Convolution Theorem and fast Fourier transform (see, e.g., Björklund *et al.*, 2007, Slivovsky and Szeider, 2020).

We call a constraint *one-sided* when there exists an STS that describes it where either $f_0(s) = s$ for every state s, or $f_1(s) = s$ for every state s. In the first case, we talk of 1-*only* constraints and STS, and in the second case, we talk of 0-*only* constraints and STS. Throughout this section, we assume that for each one-sided constraint $c \in F$, a minimal one-sided STS that describes it is known before the algorithm starts. Given a one-sided constraint c, if c is a 1-only (resp. 0-only) constraint, then let $c(i)$ denote the state the STS is in after receiving i many 1's (resp. i many 0's).

Theorem 23. *Let F be a system of one-sided constraints of maximal state size w. Given F and a width-k tree decomposition of G_F, one can compute the number of models of F in $O((2w)^k k \log(w) \cdot |G_F|)$ elementary operations.*

Let $m \in \mathbb{N}$, an m-modulo constraint c is a constraint such that, for all $x_1, \ldots, x_n, x'_1, \ldots, x'_n \in \{0,1\}$ verifying $\sum_{i \in [n]} x_i = \sum_{i \in [n]} x'_i \mod m$, we have $c(x_1, \ldots, x_n) = c(x'_1, \ldots, x'_n)$. Note that every m-modulo constraint is 1-only. We assume that the CSTS chosen for a disjunctive clause is always as shown in Figure 1, where $c(0) = s_0$ and $c(i) = s_1$ for all $i > 0$.

Theorem 24. *Let w be a natural number. Let F be a system of constraints comprising only clauses and m-modulo constraints for possibly different $m \leq w$. Given F and a width-k tree decomposition of G_F, one can compute the model count of F in $O(w^k k \log(w) \cdot |G_F|)$ elementary operations.*

Since XOR constraints are 2-modulo constraints, we obtain the following corollary.

Corollary 25. *Let F be a system of clauses and XOR constraints. Given F and a width-k tree decomposition of G_F, one can compute the model count of F in $O(2^k k \cdot |G_F|)$ elementary operations.*

6 Conclusion

We have shown that the compilation of systems of constraints parameterized by incidence treewidth to d-SDNNF is FPT for specific families of constraints, namely, constraints whose OBDD- and SDNNF-width are bounded by a constant for all variable orders and all vtrees. This generalizes known results for CNF, i.e., systems of disjunctive clauses, to many more constraints, including modulo and small-threshold constraints. Since compilation to d-SDNNF is often used in practice as a first step towards model counting, we have also shown that faster FPT model counting algorithms exist without compilation when we restrict the constraints considered. A natural question here is whether one can push our results further, that is, to constraints that do not belong to the families considered in this paper. It seems that positive compilation results can always be established by reduction to the compilation of CNF formulas (in this paper, CNF encodings of the constraints). We also ask if there are situations where encoding the problem to CNF before compiling is provably a worse strategy than reasoning on the original problem.

Acknowledgments

The research leading to this publication has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 101034440, and was supported by the Vienna Science and Technology Fund (WWTF) within the project ICT19-065 and from the Austrian Science Fund (FWF) within the projects 10.55776/ESP235, 10.55776/P36688, and 10.55776/P36420.

References

[Amarilli *et al.*, 2020] Antoine Amarilli, Florent Capelli, Mikaël Monet, and Pierre Senellart. Connecting knowledge compilation classes and width parameters. *Theory Comput. Syst.*, 64(5):861–914, 2020.

[Björklund *et al.*, 2007] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 67–74. ACM, 2007.

[Bodlaender, 1996] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[Bollig and Wegener, 2000] Beate Bollig and Ingo Wegener. Asymptotically optimal bounds for OBDDs and the solution of some basic OBDD problems. *J. Comput. Syst. Sci.*, 61(3):558–579, 2000.

[Bova *et al.*, 2015] Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. On compiling CNFs into structured deterministic DNNFs. In Marijn Heule and Sean A. Weaver, editors, *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, volume 9340 of *Lecture Notes in Computer Science*, pages 199–214. Springer, 2015.

[Capelli and Mengel, 2019] Florent Capelli and Stefan Mengel. Tractable QBF by knowledge compilation. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPIcs*, pages 18:1–18:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[Chen and Grohe, 2010] Hubie Chen and Martin Grohe. Constraint satisfaction with succinctly specified relations. *J. Comput. Syst. Sci.*, 76(8):847–860, 2010.

[Darwiche, 2004] Adnan Darwiche. New advances in compiling CNF into decomposable negation normal form. In Ramón López de Mántaras and Lorenza Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 328–332. IOS Press, 2004.

[de Colnet and Mengel, 2023] Alexis de Colnet and Stefan Mengel. Characterizing Tseitin-formulas with short regular resolution refutations. *J. Artif. Intell. Res.*, 76:265–286, 2023.

[Huang and Darwiche, 2007] Jinbo Huang and Adnan Darwiche. The language of search. *J. Artif. Intell. Res.*, 29:191–219, 2007.

[Kiesel and Eiter, 2023] Rafael Kiesel and Thomas Eiter. Knowledge compilation and more with SharpSAT-TD. In Pierre Marquis, Tran Cao Son, and Gabriele Kern-Isberner, editors, *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2-8, 2023*, pages 406–416, 2023.

[Lagniez and Marquis, 2017] Jean-Marie Lagniez and Pierre Marquis. An improved decision-DNNF compiler. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 667–673. ijcai.org, 2017.

[Muise *et al.*, 2012] Christian J. Muise, Sheila A. McIlraith, J. Christopher Beck, and Eric I. Hsu. Dsharp: Fast d-DNNF compilation with sharpSAT. In Leila Kosseim and Diana Inkpen, editors, *Advances in Artificial Intelligence - 25th Canadian Conference on Artificial Intelligence, Canadian AI 2012, Toronto, ON, Canada, May 28-30, 2012. Proceedings*, volume 7310 of *Lecture Notes in Computer Science*, pages 356–361. Springer, 2012.

[Oztok and Darwiche, 2014a] Umut Oztok and Adnan Darwiche. CV-width: A new complexity parameter for CNFs. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 675–680. IOS Press, 2014.

[Oztok and Darwiche, 2014b] Umut Oztok and Adnan Darwiche. On compiling CNF into Decision-DNNF. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 42–57. Springer, 2014.

[Oztok and Darwiche, 2015] Umut Oztok and Adnan Darwiche. A top-down compiler for sentential decision diagrams. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 3141–3148. AAAI Press, 2015.

[Pipatsrisawat and Darwiche, 2008] Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 517–522. AAAI Press, 2008.

[Samer and Szeider, 2010] Marko Samer and Stefan Szeider. Constraint satisfaction with bounded treewidth revisited. *J. Comput. Syst. Sci.*, 76(2):103–114, 2010.

[Slivovsky and Szeider, 2020] Friedrich Slivovsky and Stefan Szeider. A faster algorithm for propositional model counting parameterized by incidence treewidth. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing – SAT 2020*, pages 267–276, Cham, 2020. Springer International Publishing.

[van Hoeve and Katriel, 2006] Willem-Jan van Hoeve and Irit Katriel. Global constraints. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 169–208. Elsevier, 2006.

[Wegener, 2000] Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000.