Lower Bounds on Intermediate Results in Bottom-Up Knowledge Compilation

Alexis de Colnet and **Stefan Mengel**

Univ. Artois, CNRS, Centre de Recherche en Informatique de Lens (CRIL), F-62300 Lens, France
{decolnet, mengel}@cril.fr

Abstract

Bottom-up knowledge compilation is a paradigm for generating representations of functions by iteratively conjoining constraints using a so-called apply function. When the input is not efficiently compilable into a language – generally a class of circuits – because optimal compiled representations are *provably* large, the problem is not the compilation algorithm as much as the choice of a language too restrictive for the input. In contrast, in this paper, we look at CNF formulas for which very small circuits exists and look at the efficiency of their bottom-up compilation in one of the most general languages, namely that of structured decomposable negation normal forms (str-DNNF). We prove that, while the inputs have constant size representations as str-DNNF, any bottom-up compilation in the general setting where conjunction and structure modification are allowed takes exponential time and space, since large intermediate results have to be produced. This unconditionally proves that the inefficiency of bottom-up compilation resides in the bottom-up paradigm itself.

Introduction

One of the main objectives of knowledge compilation is transforming, or *compiling*, knowledge given as a CNF formula into other representations, generally subclasses of circuits in decomposable negation normal form (DNNF), which allow for efficient reasoning (Darwiche 2001). There are mainly two approaches to this: *top-down compilation* roughly consists of remembering the trace of an exhaustive backtracking algorithm exploring the whole solution space (Huang and Darwiche 2005), while *bottom-up compilation* iteratively conjoins representations of the clauses of the input in DNNF. For the latter approach to work, one needs an efficient so-called *apply function* which, given two DNNF and a binary Boolean operation, computes a representation of the function we get by applying the operation on the two DNNF. The only known fragments of DNNF that have such an efficient apply function for conjunctions are so-called structured DNNF (str-DNNF) in which intuitively the variable occurrences in the DNNF must follow a common tree structure called a vtree (Pipatsrisawat and Darwiche 2008). As a consequence, in practice, bottom-up knowledge compilation targets fragments of str-DNNF such as SDD (Darwiche 2011; Choi and Darwiche 2013) or OBDD (Bryant 1986; Somenzi 2009).

One inconvenience of bottom-up compilation that top-down compilation does not have is that it may create intermediate results that are far bigger in size than the final compiled form of the complete input. This was mentioned for OBDD e.g. in (Narodytska and Walsh 2007; Huang and Darwiche 2004), and proved for specific bottom-up algorithms compiling unsatisfiable CNF formulas into OBDD in (Krajícek 2008; Tveretina, Sinz, and Zantema 2010; Friedman and Xu 2013). As remarked by these works, large intermediate results are problematic because they may lead to failed compilation due to memory outs or very long runtime even for instances that have small representations. The same problem occurs also for the state of the art SDD-compiler of (Choi and Darwiche 2013), as can be verified experimentally. To mitigate the problem of large intermediate results, Narodytska and Walsh (2007) introduce heuristics for choosing an order in which to conjoin the clauses to try to decrease the size of these intermediate OBDD and show experimentally that these work well when compiling certain configuration problems bottom-up.

In this paper, we show that having large intermediate results is unavoidable for certain formulas when compiling them bottom-up, even when the final compiled form is of constant size. This is true regardless of the order in which the clauses are conjoined during compilation. We do this by formalizing the bottom-up compilation process into str-DNNF as a deduction process which only uses conjunctive apply and changing of the vtree, also called *restructuring*, a common operation in bottom-up compilation. We then show that in this framework large intermediate results must occur, even when compiling unsatisfiable formulas.

Informally stated, our main result is the following.

Theorem 1 (informal)**.** *There is a class of CNF formulas that have constant size str-DNNF representations such that any bottom-up compilation must produce intermediate str-DNNF of exponential size.*

Note that the result of Theorem 1 is unconditional and does not depend on any unproven complexity assumptions. Moreover, since str-DNNF encompass OBDD and SDD, it is true for bottom-up compilation into these formats.

The formulas that we use to show Theorem 1 are so-called *Tseitin formulas* which encode certain systems of equations over $\{0, 1\}$ whose structure is given by an underlying graph. Tseitin formulas have played a major role in the field of proof complexity, a subfield of theoretical computer science that studies the complexity of refuting unsatisfiable formulas in different proof systems which are often closely linked to practical SAT solvers, see e.g. (Buss and Nordström 2021). In particular, Tseitin formulas have also been studied when analyzing refutations by proof systems based on different forms of branching programs which are conceptually close to bottom-up compilation, see e.g. (Atserias, Kolaitis, and Vardi 2004; Glinskih and Itsykson 2021; Itsykson et al. 2020) for a small sample. Concretely, we here use a recent result from (de Colnet and Mengel 2021) that shows lower bounds on DNNF representations of satisfiable Tseitin formulas. Our basic idea is to show that any bottom-up compilation must essentially construct a DNNF representation of certain sub-formulas of the input that by (de Colnet and Mengel 2021) must be large. In fact, the result of (de Colnet and Mengel 2021) is parameterized by the so-called *treewidth* of the graph underlying the formula, where treewidth is a well-known graph parameter measuring intuitively the treelikeness of a graph. Here, our lower bound is parameterized in the same way, which requires the use of some rather heavy machinery from structural graph theory on the preservation of treewidth under graph partitions.

Preliminaries

A Boolean variable x is a variable taking its value in $\{0, 1\}$. A *literal* is a variable x or its negation $\overline{x}$. An *assignment* to a set of variables X is a mapping from X to $\{0, 1\}$. A Boolean function f on X is a mapping of the assignments to X to $\{0, 1\}$. The *satisfying assignments* of f are the assignments mapped to 1 by f. Two functions on X are equivalent, written $f \equiv g$, when their satisfying assignments are the same. When X is not specified, $\mathit{var}(f)$ denotes the set of variables of f. Given an assignment a to $Y \subseteq X$, the function f *conditioned on* a, written $f|a$, is the function on $X \setminus Y$ obtained from f after fixing all variables in Y to their values given by a. As usual, the symbols $\vee$ and $\wedge$ denote disjunction and conjunction, respectively. A *clause* is a disjunction of literals and a CNF formula (Conjunctive Normal Form) is a conjunction of clauses. The set of clauses of a CNF formula F is denoted by $\mathit{clause}(F)$. We say that F' is a subformula of F when $\mathit{clause}(F') \subseteq \mathit{clause}(F)$. The formula F' is called a *proper* subformula when the inclusion is strict.

Structured Decomposable Negation Normal Forms

A Boolean circuit Σ is a directed acyclic computation graph without parallel edges, whose leaves are labeled by literals or Boolean constants 0 or 1, and whose internal nodes are labeled by Boolean operations. The size of Σ, denoted by $|\Sigma|$, is its number of edges. The set of variables whose literals label the leaves under a node s is written $\mathit{var}(s)$. Each node s computes a Boolean function on $\mathit{var}(s)$ defined in the obvious inductive way. The function computed by Σ is defined as that computed by its roots.

A node s with successors $s_1, \ldots, s_k$ is called *decomposable* when $\mathit{var}(s_i) \cap \mathit{var}(s_j) = \emptyset$ holds for all $i \neq j$. A *Decomposable Negation Normal Form* (short DNNF) for a function f is a Boolean circuit computing f, whose internal nodes are labeled with $\vee$ or $\wedge$ and such that all $\wedge$-nodes are decomposable. The *DNNF language* is the class of DNNF circuits. One can modify a DNNF in linear time without altering the function it computes so that every internal node has fan-in 2. So we assume that all DNNF in this paper have only internal nodes with fan-in 2.

Let X be a finite set of Boolean variables. A *vtree* T for X is a binary tree whose leaves are in bijection with X. For $t \in T$, we denote by $\mathit{var}(t)$ the set of variables corresponding to the leaves under t. A *structured DNNF* (str-DNNF) is a DNNF Σ equipped with a vtree T on its variables and a mapping λ from the nodes of Σ to that of T such that:

1. for every $\wedge$-node s with successors s_0 and s_1, if $\lambda(s) = t$, then t is an internal node of T and there are t_l and t_r rooted under the two children of t such that $\lambda(s_0) = t_l$ and $\lambda(s_1) = t_r$
2. for every $\vee$-node s with successors s_0 and s_1, there is $\lambda(s) = \lambda(s_0) = \lambda(s_1)$
3. for every s, $\mathit{var}(s) \subseteq \mathit{var}(\lambda(s))$ holds

Σ is said to be *structured* by T, or to *respect* the vtree T. Given any vtree T on variables X, all Boolean functions on X are computed by some str-DNNF respecting T: just write the function in DNF (Disjunctive Normal Form) and see that every term can be turned into a str-DNNF respecting T. We remark that both SDD and OBDD are restricted forms of str-DNNF (Darwiche and Marquis 2002; Darwiche 2011).

Enforcing structuredness for DNNF can in theory result in a size blow up (Pipatsrisawat and Darwiche 2010), but it has some very useful benefits. On the one hand, in certain fragments it allows for canonicity which is often desirable (Van den Broeck and Darwiche 2015). On the other hand, structuredness is the only known property that yields efficient algorithms for conjoining DNNF (Pipatsrisawat and Darwiche 2008): there is an algorithm $\text{Apply}(\Sigma_1, \Sigma_2, \wedge)$ that, given two str-DNNF Σ_1 and Σ_2 respecting the same vtree, returns a str-DNNF equivalent to $\Sigma_1 \wedge \Sigma_2$ with the same vtree as Σ_1 and Σ_2, and runs in time $O(|\Sigma_1| \times |\Sigma_2|)$. So, consider a situation in which the clauses of a CNF formula are split into F_1 and F_2, and assume the str-DNNF Σ_1 and Σ_2 compute F_1 and F_2, respectively, and respect the same vtree. Then finding a str-DNNF that computes F is feasible in quadratic-time as it boils down to running $\text{Apply}(\Sigma_1, \Sigma_2, \wedge)$. This is the key principle behind bottom-up compilation.

Bottom-Up Compilation

Let L be a compilation language like str-DNNF. We formalize a bottom-up compilation of CNF formula $F = C_1 \wedge \cdots \wedge C_m$ as a sequence of circuits in L, $\Sigma_1, \ldots, \Sigma_N$, culminating in $\Sigma_N \equiv F$ and such that, for all $i \in [N]$

- $\Sigma_i \equiv C_j$ for some clause C_j in F, $j \in [m]$, or
- $\Sigma_i = \text{Apply}(\Sigma_j, \Sigma_k, \wedge)$ with $j, k < i$ and Σ_j and Σ_k have the same vtree, or
- $\Sigma_i \equiv \Sigma_j$ with $j < i$ and the vtrees for Σ_i and Σ_j differ.

Note that $\Sigma_i \equiv \Sigma_j$ is not necessarily easily verifiable in our framework. We say that we have an $L(\wedge, r)$ compilation of

F, where r indicates that vtree modification (*r*estructuring) is allowed. We call an $L(\wedge, r)$ *refutation* any $L(\wedge, r)$ compilation of an unsatisfiable formula. In this paper we will focus on str-DNNF$(\wedge, r)$ compilations and refutations.

We are interested in the amount of memory used when compiling CNF formulas bottom-up. To abstract away implementation details, we note that in any case a bottom-up compiler must keep every Σ_i in memory at some point[1]. Thus, the size of the biggest Σ_i is a lower bound on the space needed, and thus also on the time taken, by the compilation. One can then envision a compilation whose final circuit is way smaller than the biggest intermediate circuit, i.e., $|\Sigma_N| \ll \max_{i\in[N]} |\Sigma_i|$. Then, the run of the bottom-up compiler leading to the sequence appears intuitively wasteful. This is most visible when compiling unsatisfiable CNF formulas: the smallest compiled form is a single node 0, and since satisfiability testing is tractable in L, we can assume that $\Sigma_N = 0$, and yet, its bottom-up compilation may have large memory cost.

Note that the size of the Σ_i can differ dramatically depending on the sequence of apply operations, i.e., the order in which the clauses are conjoined. However, we will see that there are formulas that have constant-size str-DNNF representation but for which every possible str-DNNF$(\wedge, r)$ compilation must produce big intermediate results.

Graphs

We assume that the reader is familiar with basics and notation from graph theory as e.g. found in (Diestel 2012). In this section, we will remind the reader of some concepts that will be used in the remainder of this paper.

Graphs in this paper are undirected, do not contain self-loops, but may have parallel edges. Given a graph $G = (V, E)$ and a set $A \subseteq V$, we denote by $G - A$ the graph we get from G by deleting all vertices in A and all edges that contain a vertex in A. If A consists of a single node u, we also write $G - u$ instead of $G - \{u\}$. By $G[A]$ we denote the graph induced by A in G, i.e., the graph $G - (V \setminus A)$. Given another set $B \subseteq V$, we denote by $E(A, B)$ the set of edges of G that have one endpoint in A and the other in B.

A graph is called connected if there is a path from every vertex to every other vertex. A connected component is defined as a maximal connected subgraph. A 1-separator of a connected graph G is defined to be a vertex u such that $G - u$ is not connected. A graph is called 2-connected if it is connected, has at least two vertices and contains no 1-separator.

The *treewidth* $tw(G)$ of a graph G is a well-known graph parameter with broad applicability in artificial intelligence that measures roughly how close G is to being a tree. Since we will not need its technical definition in this paper but use several results on it as black boxes, we will not formally introduce it here and refer the reader to (Diestel 2012; Harvey and Wood 2017). We will use the following result from (Bodlaender and Koster 2006) which we reformulate to simplify notation.

[1]Note that the whole sequence $\Sigma_1, \ldots, \Sigma_N$ never has to be kept in memory entirely since earlier Σ_i can be deleted from memory when they are not needed anymore (Buss and Nordström 2021).

Theorem 2. *Let G be a graph with a 1-separator u. Then $G - u$ contains a connected component $G' = (V', E')$ such that $tw(G) = tw(G[V' \cup \{u\}])$.*

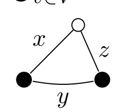
Tseitin Formulas

We study Tseitin formulas which are CNF formulas representing systems of parity constraints structured by a graph $G = (V, E)$. The graph is equipped with a function $c : V \to \{0, 1\}$ which assigns *charges* 0 or 1 to its vertices. Each edge e of G is associated to a Boolean variable x_e. Given a set $E' \subseteq E$, we write $X_{E'} = \{x_e \mid e \in E'\}$. The Tseitin formula encodes the fact that, if we only keep in G the edges whose variables are given value 1, then all vertices with charge 1 have an odd degree and all vertices with charge 0 have an even degree. More formally let $E(v)$ denote the set of edges of which v is an endpoint and define the constraint

$$\chi_{v,c} : \sum_{e\in E(v)} x_e = c(v) \mod 2$$

then the Tseitin formula $T(G, c)$ computes $\bigwedge_{v\in V} \chi_{v,c}$. Each $\chi_{v,c}$ can be encoded in a CNF formula $F_{v,c}$ on variables $X_{E(v)}$ composed of $2^{|E(v)|-1} = 2^{\deg(v)-1}$ clauses of size $\deg(v)$. The Tseitin formula over G for the charge function c is the CNF formula $T(G, c) = \bigwedge_{v\in V} F_{v,c}$. For convenience we often drop c from the notations writing only $T(G)$, χ_v, or F_v. For $v \in V$ let $1_v : V \to \{0, 1\}$ be the function mapping v to 1 and all other vertices to 0. The complement parity constraint to $\chi_{v,c}$ is $\chi_{v,(c+1_v \bmod 2)}$, which we write $\overline{\chi}_{v,c}$ for convenience.

We use the notation *clause* $(\chi_{v,c})$ to denote the set of clauses of $F_{v,c}$. We extend this notation to Tseitin formulas by defining *clause* $(T(G, c)) = \bigcup_{v\in V}$ *clause* $(\chi_{v,c})$.

Example 1. Let G be the graph 

where white vertices have charge 1 and black vertices have charge 0. The corresponding Tseitin formula is $T(G) = (x \vee z) \wedge (\overline{x} \vee \overline{z}) \wedge (x \vee \overline{y}) \wedge (\overline{x} \vee y) \wedge (y \vee \overline{z}) \wedge (\overline{y} \vee z)$.

There is a simple criterion for the satisfiability of Tseitin formulas.

Lemma 1 (Urquhart 1987)**.** *$T(G, c)$ is satisfiable if and only if $\sum_{v\in U} c(v) = 0 \mod 2$ holds for all connected components $G' = (U, E')$ of G.*

In this paper we study the space complexity of str-DNNF$(\wedge, r)$-compilation of unsatisfiable Tseitin formulas whose underlying graph is connected. We parameterize our bounds by the treewidth of the graph. For exponential lower bounds to be relevant, we need an input CNF formula whose length is polynomial in the number of variables. We achieve this by restricting our study to graphs of maximum degree bounded by some constant Δ. This very common restriction leads to an upper bound of $|V| \times 2^{\Delta-1}$ on the number of clauses in $T(G)$.

Note that there is always a small str-DNNF for a single parity constraint.

Lemma 2 (Pipatsrisawat and Darwiche 2010)**.** *Let χ be a parity constraint and let T be a vtree on var (χ). There is a str-DNNF of size $O(|var(\chi)|)$ respecting T that computes χ.*

However representing a *satisfiable* Tseitin formula in str-DNNF, so a system of parity constraints, is expensive.

Theorem 3 (de Colnet and Mengel 2021)**.** *The smallest DNNF representing $T(G)$* satisfiable *with G a graph of maximum degree Δ has size at least $2^{\Omega(k)/\Delta}/n$ with $k = tw(G)$ and $n = |var(T(G))|$.*

Refuting Tseitin Formulas in Str-DNNF($\wedge$, r)

In this section, we will give the formal version of our main result Theorem 1 and prove it, building on several lemmas whose proof we defer to the following sections. We start with a simple observation that essentially says that, given a bottom-up compilation of a function f, one can easily infer a bottom-up compilation of $f|a$, for any partial assignment a. This will be useful in several upcoming proofs.

Lemma 3. *Let F be a CNF formula and $\Sigma_1, \ldots, \Sigma_N$ be a str-DNNF($\wedge$, r) compilation of F. Let a be a partial assignment to var (F), then $\Sigma_1|a, \ldots, \Sigma_N|a$ is a str-DNNF($\wedge$, r) compilation of $F|a$.*

Proof. For every i between 1 and N let Σ'_i be $\Sigma_i|a$. str-DNNF allow linear-time conditioning without size increase nor vtree modification, so $|\Sigma'_i| \leq |\Sigma_i|$ and Σ'_i and Σ_i share a common vtree. We have $\Sigma_N \equiv F$, so $\Sigma'_N \equiv F|a$ follows. We will prove that, for every i, either Σ'_i is the str-DNNF representation of a clause of $F|a$, or there are $j, k < i$ such that $\Sigma'_i = \text{Apply}(\Sigma'_j, \Sigma'_k, \wedge)$ where all three str-DNNF share a common vtree, or there is $j < i$ such that $\Sigma'_i \equiv \Sigma'_j$ and the vtree of Σ'_i and Σ'_j may differ.

Take an arbitrary i between 1 and N. If Σ_i is the str-DNNF representation of a clause $C \in F$, that is, $\Sigma_i \equiv C$, then clearly $\Sigma'_i = \Sigma_i|a \equiv C|a$ and $C|a$ is indeed a clause of $F|a$. Otherwise if Σ_i is the str-DNNF returned by $\text{Apply}(\Sigma_j, \Sigma_k, \wedge)$, then $\Sigma_i \equiv \Sigma_j \wedge \Sigma_k$ and all three str-DNNF share a common vtree. Then $\Sigma'_i = \Sigma_i|a \equiv (\Sigma_j \wedge \Sigma_k)|a \equiv \Sigma_j|a \wedge \Sigma_k|a = \Sigma'_j \wedge \Sigma'_k$. Since the vtree is not modified by conditioning we can feed Σ'_j and Σ'_k to an Apply to obtain $\Sigma'_i = \text{Apply}(\Sigma'_j, \Sigma'_k, \wedge)$. Finally in the case where Σ_i is equivalent to Σ_j with potentially a vtree modification, it is clear that $\Sigma'_i = \Sigma_i|a \equiv \Sigma_j|a = \Sigma'_j$. ☐

Our main result is the following theorem on the refutation of unsatisfiable formulas by bottom-up compilation.

Theorem 1. *Let $\mathcal{G}$ be a class of graphs whose maximum degree is bounded by a constant. All str-DNNF($\wedge$, r) refutation of an unsatisfiable $T(G)$ with $G \in \mathcal{G}$ have size at least $2^{\Omega(k)} poly(1/n)$ with $k = tw(G)$ and $n = |var(T(G))|$.*

We will prove Theorem 1 later in this section after some discussion and preparations. First, note that there are graphs of bounded degree with treewidth linear in the number of vertices, see e.g. (Grohe and Marx 2009). It follows that there are formulas where the intermediate results have exponential size.

Corollary 1. *There is a family of unsatisfiable CNF formulas such that every formula on n variables has $O(n)$ clauses and all its str-DNNF($\wedge$, r) refutations have an intermediate result of size $2^{\Omega(n)}$.*

Let us compare Corollary 1 with known exponential lower bounds on the size of intermediate results for similar refutation systems, see for instance (Krajícek 2008; Segerlind 2008; Tveretina, Sinz, and Zantema 2010; Friedman and Xu 2013). First, we are not aware of refutation systems using str-DNNF circuits that are not OBDD or branching programs. Since OBDD are generally exponentially bigger than str-DNNF, our result is stronger in that respect. Moreover, restructuring is rarely allowed in the OBDD-based proof system while it is in ours. Most known bounds are stated for OBDD-based refutations in which the variable order can be arbitrary but cannot be changed in the refutation. Also we do not require any specific order in which the clauses are conjoined, which is a restruction used for some bounds in, e.g., (Friedman and Xu 2013).

Our results might look somewhat unconvincing since they only talk about the compilation of unsatisfiable formulas, a setting in which costly compilation can be substituted by a usually much less expensive single call of a SAT solver[2]. However, equipped with Lemma 3, we can lift them to satisfiable formulas that have constant size str-DNNF representation with a simple trick.

Corollary 2. *There are satisfiable CNF formulas that have constant size str-DNNF representations such that any str-DNNF($\wedge$, r) compilation must have an intermediate result of size $2^{\Omega(n)}$ where n is the number of variables in the input.*

Proof. Consider a class of unsatisfiable Tseitin formulas $\mathcal{T} := \{T(G) \mid G \in \mathcal{G}\}$ for a class of graphs $\mathcal{G}$ of treewidth linear in the number of vertices and let x be a fresh variable not used in any of these formulas. For each $T(G)$ let $F(G)$ be the formula $T(G)$ with the additional literal x added to all clauses. Clearly, $F(G) \equiv x \vee T(G) \equiv x$, so the smallest str-DNNF representing $F(G)$ has size 1. By Lemma 3, given a str-DNNF($\wedge$,r) compilation of $F(G)$, we can condition all intermediate str-DNNF on $x = 0$ to obtain a str-DNNF($\wedge$,r) refutation of $T(G)$. Since conditioning does not increase the size of str-DNNF the corollary follows from Theorem 1. ☐

Note that we could prove a version of Corollary 2 parameterized by the so-called primal treewidth of the formulas. Since we do not want to introduce even more notions, we abstain from doing so here.

As a first step towards the proof of Theorem 1, let $T(G)$ be unsatisfiable with $G = (V, E)$ connected. We look at the very last Apply in the refutation of $T(G)$ in str-DNNF($\wedge, r$):

$$\Sigma_N = \text{Apply}(\Sigma^\ell, \Sigma^r, \wedge)$$

where $\Sigma_N \equiv 0$ and Σ^ℓ and Σ^r are two satisfiable str-DNNF structured by the same vtree. Roughly put, the proof of Theorem 1 is as follows:

[2]In fact, some knowledge compilers, e.g. the top-down knowledge compiler D4 (Lagniez and Marquis 2017), make a call to a SAT solver before trying to compile the input to avoid wasting time when compiling unsatisfiable instances.

1. We prove that there is a partition (A, B) of V such that *both* $G[A]$ *and* $G[B]$ have treewidth $\Omega(tw(G))$.
2. For that partition we show how to construct from Σ^ℓ and Σ^r in polynomial time a str-DNNF Σ^* computing a *satisfiable* Tseitin formula $T(G[A])$ or $T(G[B])$
3. From Theorem 3 we derive that $|\Sigma^*| = 2^{\Omega(tw(G))}$ and use $|\Sigma^*| = O(|\Sigma^\ell| \times |\Sigma^r|)$ to conclude.

For convenience we denote $G_A := G[A]$ and $G_B := G[B]$. In the second step, we can not really control which of $T(G_A)$ or $T(G_B)$ is satisfiable. But the first step frees us from worrying about this: since both G_A and G_B have large treewidth, Σ^* have size exponential in the treewidth of G regardless of whether it represents $T(G_A)$ or $T(G_B)$.

The following lemmas will be proved in the next sections.

Lemma 4. *Let $G = (V, E)$ be a 2-connected graph with maximum degree Δ. There is a partition (A, B) of V such that G_A is connected, G_B is 2-connected, and $\min(tw(G_A)), tw(G_B)) \geq \lfloor \frac{\alpha tw(G)}{\Delta^2} \rfloor$ where $\alpha > 0$ is a fixed universal constant.*

Lemma 5. *Let $T(G, c)$ be a Tseitin formula with G connected and a partition (A, B) of V such that both G_A and G_B are connected. Then for every assignment a to $X_{E(A,B)}$ there are $c_A^a : A \to \{0, 1\}$ and $c_B^a : B \to \{0, 1\}$ such that*

$$T(G, c)|a = T(G_A, c_A^a) \wedge T(G_B, c_B^a).$$

Moreover, if $T(G, c)$ is unsatisfiable then either $T(G_A, c_A^a)$ or $T(G_B, c_B^a)$ is unsatisfiable, but not both. Which of the two formulas is satisfiable depends on whether the number of variables that a maps to 1 is odd or even.

Lemma 6. *Let* Apply$(\Sigma^\ell, \Sigma^r, \wedge)$ *be the last step of a str-DNNF($\wedge$,r) refutation of $T(G)$ where G is 2-connected. Assume that there is a partition (A, B) of V such that G_A is connected, G_B is 2-connected, and both have treewidth at least 2. Then there is a str-DNNF of size $O(|\Sigma^\ell| \times |\Sigma^r|)$ computing a satisfiable Tseitin formula whose graph is G_A or G_B.*

Proof of Theorem 1. First, using Lemmas 4 and 6 and Theorem 3, we prove the result when G is 2-connected. Let Δ be an upper bound on the maximum degree of all our graphs. Fix a graph $G = (V, E)$ and consider the partition (A, B) of V given by Lemma 4. Let $k = tw(G)$ and $n = |E(G)|$. We can choose the constant hidden in $2^{\Omega(k)}$ of the statement so that the theorem becomes trivial whenever $\lfloor \alpha k/\Delta^2 \rfloor < 2$, so we assume $\lfloor \alpha k/\Delta^2 \rfloor \geq 2$ in the remainder.

The conditions on (A, B) described in Lemma 6 are met so we obtain a str-DNNF Σ^* computing a *satisfiable* Tseitin formula $T(G_U)$ for some $U \in \{A, B\}$ with $|\Sigma^*| \leq \gamma \times |\Sigma^\ell| \times |\Sigma^r|$ for some $\gamma > 0$. Now Theorem 3 says that there is a constant $\beta > 0$ such that $|\Sigma^*| \geq 2^{\beta k/\Delta^3}/n$. So we have $\min(|\Sigma^\ell|, |\Sigma^r|) \geq 2^{\beta k/2\Delta^3}/(\gamma n)$. This completes the proof in the case where G is 2-connected.

Now we show how to go from the general case to the case where G is 2-connected. Assume G has a 1-separator $\{u\}$ and let $U_1, \dots, U_s$ be the vertex sets of the connected components of G after removal of u. We know from Theorem 2 that there is some $i \in [s]$ such that $tw(G[U_i \cup \{u\}]) = tw(G)$, say $i = 1$. Now there is a proper subset $E' \subset E(u)$ such that removing E' from G yields two connected components G_A and G_B, with $U_i \cup \{u\} \subseteq A$. So $E' = E(A, B)$ and, by Lemma 5, we can choose an assignment a to E' such that $T(G)|a = T(G_A) \wedge T(G_B)$ where $T(G_B)$ is satisfiable and $T(G_A)$ unsatisfiable.

Let a_B be a satisfying assignment of $T(G_B)$. Using Lemma 3 we can condition any str-DNNF($\wedge$,r) refutation of $T(G)$ on the assignment $a \cup a_B$ to obtain a str-DNNF($\wedge$,r) refutation of $T(G_A)$ without size increase. G_A has fewer 1-separators than G and $tw(G_A) = tw(G)$. We repeat the procedure until obtaining a str-DNNF($\wedge$,r) refutation of $T(G')$, with G' a subgraph of G that has the same treewidth of G and has no 1-separator. So G' is 2-connected, and the refutation of $T(G')$ obtained is at most as large as that of $T(G)$ we have started from. $\square$

Graph Bi-Partition with Large Treewidth on Both Sides (Lemma 4)

Lemma 4 is shown with the help of Theorem 4 below combined with Theorem 2. For space reasons we defer the proof to the full version of the paper. We here discuss some of the underlying graph theory, in particular the following result.

Theorem 4. *There exists a constant $0 < \alpha \leq 1$ such that, for all graphs $G = (V, E)$ with maximum degree at most Δ, there is a partition (A, B) of V such that $tw(G[A]) \geq \lfloor \frac{\alpha tw(G)}{\Delta^2} \rfloor$ and $tw(G[B]) \geq \lfloor \frac{\alpha tw(G)}{\Delta^2} \rfloor$.*

To illustrate Theorem 4, we look at the particular case of grid graphs. The $n \times n$ grid has treewidth $n - 1$ and maximum degree 4. It is straightforward to partition its vertices to obtain an $n \times \lfloor n/2 \rfloor$ grid on one side, and an $n \times \lceil n/2 \rceil$ on the other. Using this partition for (A, B) we see that $G[A]$ and $G[B]$ both have an $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$ induced grid and therefore both have treewidth at least $\lfloor n/2 \rfloor - 1 \geq (n-1)/4$. Of course the constant α in the theorem is way smaller than 4.

The proof of Theorem 4 is technical and is deferred to the full version due to space constraint, here we just provide some arguments to justify its veracity. Theorem 4 is an adaptation of the following result of Chekuri of Chuzhoy (2013).

Theorem 5. *Let h and r be integers and let $G = (V, E)$. There are positive constants β and c such that, if $h^3 r \leq \beta \frac{tw(G)}{\log^c(tw(G))}$, then there is an* efficient *algorithm to partition V into $(V_1, \dots, V_h)$, with $tw(G[V_i]) \geq r$ true for all $i \in [h]$.*

Theorem 4 is almost a subcase of Theorem 5 with $h = 2$. The only problem is that in Theorem 4, r would be roughly α/Δ^2 and thus independent of the treewidth, which is not the case in Theorem 5 because of the divisor $\log^c(tw(G))$. A careful examination of Chekuri and Chuzhoy's proof shows that the log-divisor has two reasons: (1) a preprocessing of G to decrease its degree and (2) the use of an approximation algorithm to make their partition efficiently computable. Since we work with graphs of bounded degree and only care about the existence of a partition and not its computation, we can adapt the proof for $h = 2$ and make some other adjustments to get rid of the $\log^c(tw(G))$ to obtain Theorem 4, see the full version of this paper for details.

Graph Partitions for Tseitin Formulas and Subformulas (Lemma 5)

In this section we prepare for the proof of Lemma 6 by recalling some results on how Tseitin formulas behave when we disconnect the underlying graph. The variables of a Tseitin formula $T(G, c)$ uniquely identify the edges of its underlying graph G. After assigning the variable x corresponding to the edge $e := uv$ in $T(G, c)$ and removing the negated literals and the satisfied clauses, the new formula is a Tseitin formula $T(G', c')$, but this time for the graph $G' = (V, E \setminus \{e\})$. If x is assigned 0, then the new charge function is the same as the old one, that is, $c' = c$. Otherwise if x is assigned 1, then c' coincides with c on all vertices except u and v, that is, $c' = c + 1_u + 1_v \mod 2$. By induction, conditioning $T(G, c)$ on a partial assignment of its variables yields a new Tseitin formula whose underlying graph is G without the corresponding edges. We focus on variable conditionings that disconnect G.

Proof of Lemma 5. Let $G = (V, E)$. $T(G, c)|a$ be the CNF obtained by removing from $T(G, c)$ all clauses containing a literal set to 1 by a, and removing all literals set to 0 by a from the remaining clauses. $T(G, c)|a$ is exactly $T(G', c^a)$ with $G' = (V, E \setminus E(A, B))$ and $c^a = c + \sum_{x_{uv}: a(x_{uv})=1} 1_u + 1_v \mod 2$.

G_A and G_B are the only two connected components of G' so $T(G', c^a) = T(G_A, c^a_A) \wedge T(G_B, c^a_B)$ where c^a_A and c^a_B are the restrictions of c^a to A and B respectively.

Now if $T(G, c)$ is unsatisfiable, then so is $T(G_A, c^a_A) \wedge T(G_B, c^a_B)$. Let S_A and S_B be the sums of all $c^a(u)$ for u in A and B, respectively. By Lemma 1 we have $S_A = 1 \mod 2$ or $S_B = 1 \mod 2$. Observe that $\sum_{u \in V} c^a(u) = S_A + S_B$ and that $\sum_{u \in V} c^a(u) = \sum_{u \in V} c(u) = 1 \mod 2$. So either $S_A = 0 \mod 2$ or $S_B = 0 \mod 2$ holds. Again by Lemma 1, it follows that either $T(G_A, c^a_A)$ or $T(G_B, c^a_B)$ is satisfiable.

All edges whose variables are assigned values by a have one endpoint in A and the other in B. Let $\text{card}(a) := |\{x_{uv} : a(x_{uv}) = 1\}|$. Then looking at the expression of c^a we see that $S_A = \sum_{u \in A} c^a(u) = \text{card}(a) + \sum_{u \in A} c(u)$ and $S_B = \text{card}(a) + \sum_{u \in B} c(u)$. Since the parity of S_A and S_B decides the satisfiability of $T(G_A, c^a_A)$ and $T(G_B, c^a_B)$, and since $S_A \neq S_B \mod 2$, we get that the parity of $\text{card}(a)$ decides which Tseitin formula is satisfiable. □

str-DNNF in the refutation of $T(G, c)$ represent subformulas of $T(G, c)$. For F such a subformula, given a partition (A, B) of V and an assignment a to $X_{E(A,B)}$, the CNF formula $F|a$ is of the form $F^a_A \wedge F^a_B$ where F^a_A is a subformula of $T(G_A, c^a_A)$ and F^a_B is a subformula of $T(G_B, c^a_B)$.

Lemma 7. *Let Σ be a str-DNNF representing a subformula F of $T(G)$. Let (A, B) be a partition of V and a be an assignment to $X_{E(A,B)}$ such that $F|a$ is satisfiable, then there are str-DNNF Σ_A and Σ_B of size at most $|\Sigma|$ and with the same vtree as Σ, that represent F^a_A and F^a_B respectively.*

Proof. This follows from conditioning being feasible without size increase nor vtree modification on str-DNNF (Pipatsrisawat and Darwiche 2008). Let T be the vtree of Σ. First we can obtain a str-DNNF Σ' equivalent to $\Sigma|a \equiv F|a = F^a_A \wedge F^a_B$ of size at most $|\Sigma|$ and that respects T. Since $F|a$ is satisfiable and since the variables of F^a_A and F^a_B are disjoint, we have an assignment a' to the variables of F^a_B that satisfies F^a_B and such that $(F|a)|a' = F^a_A$. So from Σ' we can obtain a str-DNNF equivalent to $\Sigma'|a \equiv F^a_A$, of size at most $|\Sigma'|$, and whose vtree is T. The argument works analogously for F^a_B. □

From Unsatisfiable to Satisfiable Tseitin Formulas (Lemma 6)

We call a constraint χ *incomplete* in a CNF formula F when *clause* $(\chi) \cap$ *clause* $(F) \neq$ *clause* (χ). Clearly a subformula of $T(G)$ has incomplete constraints if and only if it is a proper subformula of $T(G)$. All str-DNNF in a str-DNNF($\wedge$,r) compilation of $T(G)$, except the last one, have incomplete constraints.

The proof of Lemma 6 intuitively works by considering two cases as follows. In the first case, we assume that one of Σ^r and Σ^ℓ contains the constraints for B almost completely, say this is the case for Σ^ℓ. We choose an assignment a to $X_{E(A,B)}$ such that the resulting Tseitin formula $T(G_B, c^a_B)$ is satisfiable. Then we can extract from Σ^ℓ a str-DNNF and conjoin to this str-DNNF the few missing constraints with increasing is size too much, so that it computes $T(G_B, c^a_B)$.

In the second case, several constraints for B are incomplete in both Σ^ℓ and Σ^r. In that case, we can choose an assignment a to $X_{E(A,B)}$ such that the subformulas made of constraints for A used in the construction of Σ^r are satisfiable, and the same is true for Σ^ℓ. Then we can conjoin suitably processed versions of Σ^r and Σ^ℓ to get a str-DNNF representation of $T(G_A, c^a_A)$ without increasing the size too much. More formally, we consider the following two cases:

1. For *some* $\Sigma \in \{\Sigma^\ell, \Sigma^r\}$, at most two constraints for vertices in B are incomplete in Σ.
2. For *every* $\Sigma \in \{\Sigma^\ell, \Sigma^r\}$, at least three constraints for vertices in B are incomplete in Σ.

Lemma 6 (Case 1). *Use the notation of Lemma 6. If for some $\Sigma \in \{\Sigma^\ell, \Sigma^r\}$ at most two constraints of $T(G)$ for vertices of B are incomplete in Σ, then there is a str-DNNF of size $O(|\Sigma|)$ computing a satisfiable Tseitin formula whose underlying graph is G_B.*

Proof. Σ is satisfiable, so there is an assignment a to $X_{E(A,B)}$ such that $\Sigma|a$ is satisfiable. Let F be the CNF whose clauses are used to construct Σ in the refutation. With Lemma 7 we obtain a str-DNNF Σ_B equivalent to F^a_B and such that $|\Sigma_B| = O(|\Sigma|)$. Let B' be the set of vertices in B whose constraints are incomplete in Σ. By assumption $|B'| \leq 2$. For all $v \in B \setminus B'$, the constraint of $T(G_B, c^a_B)$ for v is complete in Σ_B because the constraint of $T(G)$ for v is complete in Σ. If $B' = \emptyset$ then all constraints of $T(G_B, c^a_B)$ are complete in Σ_B, so $\Sigma_B \equiv T(G_B, c^a_B)$ and we are done.

Now assume that B' contains two vertices: $B' = \{v, w\}$ and let χ'_v, χ'_w be the constraints of $T(G_B, c^a_B)$ for v and w (for the case of one vertex, just take $v = w$). All constraints but χ'_v and χ'_w are complete in Σ_B, so

$$\Sigma_B \wedge \chi'_v \wedge \chi'_w \equiv T(G_B, c^a_B)$$

Let T be the vtree for Σ_B. Lemma 2 gives us str-DNNF D_v and D_w computing χ'_v and χ'_w respectively, of size $O(\Delta)$, and both respecting T. We get a str-DNNF D structured by T and equivalent to $T(G_B, c^a_B)$ of size $O(|\Sigma_B| \times \Delta^2) = O(|\Sigma|)$ by conjoining D_v and D_w to Σ_B. □

Lemma 6 (**Case 2**). *Use the notation of Lemma 6. If for every $\Sigma \in \{\Sigma^\ell, \Sigma^r\}$ at least three constraints of $T(G)$ for vertices of B are incomplete in Σ, then there is a str-DNNF of size $O(|\Sigma^\ell| \times |\Sigma^r|)$ computing a satisfiable Tseitin formula whose underlying graph is G_A.*

Proof. Let F^ℓ and F^r be the CNF formulas whose clauses were used to construct Σ^ℓ and Σ^r, respectively. Apply(Σ^ℓ, Σ^r, $\wedge$) is the last apply of the refutation so there must be $F^\ell \wedge F^r = T(G)$. Our aim is to find an assignment a to $X_{E(A,B)}$ such that $T(G_A, c^a_A)$, $\Sigma^\ell|a$, and $\Sigma^r|a$ are satisfiable. If such an assignment exists, then using Lemma 7 we could obtain str-DNNF Σ^ℓ_A and Σ^r_A that represent $(F^\ell)^a_A$ and $(F^r)^a_A$ respectively. Since Σ^ℓ and Σ^r have the same vtree, so would Σ^ℓ_A and Σ^r_A. So we could construct a str-DNNF computing $\Sigma^\ell_A \wedge \Sigma^r_A \equiv (F^\ell)^a_A \wedge (F^r)^a_A = T(G_A, c^a_A)$ in time $|\Sigma^\ell_A| \times |\Sigma^r_A| \leq |\Sigma^\ell| \times |\Sigma^r|$, thus finishing the proof.

It remains to find this assignment a. Take $\Sigma \in \{\Sigma^\ell, \Sigma^r\}$ and let $F \in \{F^\ell, F^r\}$ be the corresponding CNF. By Lemma 5, if $T(G_A, c^a_A)$ is satisfiable then $T(G_B, c^a_B)$ is unsatisfiable. Then we have $\Sigma|a \equiv F^a_A \wedge F^a_B$ where F^a_A is satisfiable since it is a subformula of $T(G_A, c^a_A)$, and we want F^a_B to be satisfiable as well. The following claims help us find a such that F^a_B is satisfiable. The proofs are deferred to the full version of the paper. Note that Claim 1 is a folklore result on Tseitin formulas.

Claim 1. *Since G_B is a 2-connected graph, the* proper *subformulas of* any *Tseitin formula $T(G_B)$ are* all *satisfiable.*

Claim 2. *Let F be a proper subformula of $T(G)$. Take $C_v \in$ clause(χ_v) not in F and denote by C'_v its restriction to $X_{E(A,B)}$. If both G_A and G_B have treewidth at least 2, then for every assignment a to $X_{E(A,B)}$ that falsifies C'_v, the constraint $\chi_v|a$ is incomplete in $F|a$.*

If we can find a such that $T(G_A, c^a_A)$ is satisfiable and such that F^a_B is a *proper* subformula of $T(G_B, c^a_B)$, i.e., not all constraints of $T(G_B, c^a_B)$ are complete in F^a_B. Then by the above claims, F^a_B will be satisfiable. Then $\Sigma|a \equiv F^a_A \wedge F^a_B$ will be satisfiable as well since *var* $(F^a_A) \cap$ *var* $(F^a_B) = \emptyset$. Recall that this must hold for both $F = F^\ell$ and $F = F^r$.

By assumption there is $u^r \in B$ whose constraint is incomplete in Σ^r and there are $u^\ell, v^\ell, w^\ell \in B$ whose constraints are incomplete in Σ^ℓ. The latter three vertices are distinct, so at least two of them are different from u^r. Suppose, without loss of generality, that $u^r \neq v^\ell$ and $u^r \neq w^\ell$. For convenience, rename $u = u^r$, $v = v^\ell$ and $w = w^\ell$.

Let C_u be a clause of χ_u missing from *clause* (Σ^r) and let C_v and C_w be clauses of χ_v and χ_w missing from *clause* (Σ^ℓ). We denote $C_u = C'_u \vee C''_u$ where C'_u is the restriction of C_u to *var* $(X_{E(A,B)})$. Note that C'_u may be empty. Define $C_v = C'_v \vee C''_v$ and $C_w = C'_w \vee C''_w$ similarly. Let $E'(u)$, $E'(v)$ and $E'(w)$ be the set of edges corresponding to *var* (C'_u), *var* (C'_v) and *var* (C'_w), respectively. By definition, all three sets are subsets of $E(A, B)$.

Claim 3. *We have $E(A, B) \neq E'(u) \cup E'(v)$ or $E(A, B) \neq E'(u) \cup E'(w)$.*

Proof. If $E'(u) = \emptyset$ or $E'(v) = \emptyset$ or $E'(w) = \emptyset$, then the claim holds because otherwise $E(A, B)$ would be a subset or $E(u)$, or a subset of $E(v)$, or a subset of $E(w)$, which is not possible since G is 2-connected.

Otherwise, if neither $E'(u)$ nor $E'(v)$ nor $E'(w)$ is empty, then the three sets are pairwise disjoint since $u, v, w \in B$. So if $E(A, B) = E'(u) \cup E'(v)$ were to hold, then we would have $E(A, B) \neq E'(u) \cup E'(w)$ because otherwise $E'(v) = E'(w) \neq \emptyset$ would hold, which is impossible. □

Suppose, w.l.o.g., that $E(A, B) \neq E'(u) \cup E'(v)$. Let a'_u and a'_v be the assignments to *var* (C'_u) and *var* (C'_v) that falsify C'_u and C'_v, respectively (if C'_u is empty, then so is a'_u). Conditioning $T(G)$ on $a'_u \cup a'_v$ gives an unsatisfiable Tseitin formula on the graph obtained by removing $E'(u) \cup E'(v)$ from G. Call that graph G'. Since G, G_A, and G_B are connected, and since $E'(u) \cup E'(v)$ is a *proper* subset of $E(A, B)$, we have that G' is connected. So by Lemma 5, we have an assignment a to $X_{E(A,B)}$ that extends $a'_u \cup a'_v$ and such that $T(G)|a = T(G_A, c^a_A) \wedge T(G_B, c^a_B)$ where $T(G_A, c^a_A)$ is satisfiable and $T(G_B, c^a_B)$ is unsatisfiable.

Remember that C_u and C_v are missing from *clause* (Σ^r) and *clause* (Σ^ℓ), respectively, and that C'_u and C'_v are their restrictions to $X_{E(A,B)}$. By construction, a falsifies C'_u and C'_v, therefore by Claim 2 the constraint for the vertices u and v are incomplete in $\Sigma^r|a$ and $\Sigma^\ell|a$, respectively. It follows, since u and v belong to B, that $(F^\ell)^a_B$ and $(F^r)^a_B$ are proper subformulas of $T(G_B, c^a_B)$. Then since G_B is 2-connected, Claim 1 entails that both $(F^\ell)^a_B$ and $(F^r)^a_B$ are satisfiable, and therefore $\Sigma^\ell|a$ and $\Sigma^r|a$ are satisfiable. □

Conclusion

In the past, experimental works hinted at the inefficiency of the bottom-up approach for compiling some inputs into specific languages like OBDD or SDD. In this paper, we provide theoretical arguments that support the idea that the inefficiency of bottom-up compilation resides in the bottom-up paradigm itself. We propose a framework for compilation that targets the very general language of str-DNNF, puts no constraint on the order in which clauses are conjoined, and allows on-the-fly restructuring of the str-DNNF. Despite these degrees of freedom, we have found a class of CNF formulas that have constant-size str-DNNF representations and proved that they require exponential time and space to be compiled with the bottom-up approach.

In the future, it would be interesting to better understand how the size of intermediate results in bottom-up compilation is impacted by the order in which clauses are conjoined. For example, can it be shown theoretically when the heuristics from (Narodytska and Walsh 2007) perform well? Can similar heuristics also be used in the construction of SDD?

Acknowledgments

This work has been partly supported by the PING/ACK project of the French National Agency for Research (ANR-18-CE40-0011).

References

Atserias, A.; Kolaitis, P. G.; and Vardi, M. Y. 2004. Constraint Propagation as a Proof System. In Wallace, M., ed., *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*, 77–91. Springer.

Bodlaender, H. L.; and Koster, A. M. C. A. 2006. Safe separators for treewidth. *Discret. Math.*, 306(3): 337–350.

Bryant, R. E. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers*, 35(8): 677–691.

Buss, S.; and Nordström, J. 2021. Proof complexity and SAT solving. *2nd edition of Handbook of Satisfiability, Draft version available at https://www. math. ucsd. edu/ sbuss/ResearchWeb/ProofComplexitySAT*.

Chekuri, C.; and Chuzhoy, J. 2013. Large-treewidth graph decompositions and applications. In Boneh, D.; Roughgarden, T.; and Feigenbaum, J., eds., *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, 291–300. ACM.

Choi, A.; and Darwiche, A. 2013. Dynamic Minimization of Sentential Decision Diagrams. In desJardins, M.; and Littman, M. L., eds., *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. AAAI Press.

Darwiche, A. 2001. Decomposable negation normal form. *J. ACM*, 48(4): 608–647.

Darwiche, A. 2011. SDD: A New Canonical Representation of Propositional Knowledge Bases. In Walsh, T., ed., *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, 819–826. IJCAI/AAAI.

Darwiche, A.; and Marquis, P. 2002. A Knowledge Compilation Map. *J. Artif. Intell. Res.*, 17: 229–264.

de Colnet, A.; and Mengel, S. 2021. Characterizing Tseitin-Formulas with Short Regular Resolution Refutations. In Li, C.; and Manyà, F., eds., *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, volume 12831 of *Lecture Notes in Computer Science*, 116–133. Springer.

Diestel, R. 2012. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer. ISBN 978-3-642-14278-9.

Friedman, L.; and Xu, Y. 2013. Exponential Lower Bounds for Refuting Random Formulas Using Ordered Binary Decision Diagrams. In Bulatov, A. A.; and Shur, A. M., eds., *Computer Science - Theory and Applications - 8th International Computer Science Symposium in Russia, CSR 2013, Ekaterinburg, Russia, June 25-29, 2013. Proceedings*, volume 7913 of *Lecture Notes in Computer Science*, 127–138. Springer.

Glinskih, L.; and Itsykson, D. 2021. On Tseitin Formulas, Read-Once Branching Programs and Treewidth. *Theory Comput. Syst.*, 65(3): 613–633.

Grohe, M.; and Marx, D. 2009. On tree width, bramble size, and expansion. *J. Comb. Theory, Ser. B*, 99(1): 218–228.

Harvey, D. J.; and Wood, D. R. 2017. Parameters Tied to Treewidth. *J. Graph Theory*, 84(4): 364–385.

Huang, J.; and Darwiche, A. 2004. Using DPLL for Efficient OBDD Construction. In *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*.

Huang, J.; and Darwiche, A. 2005. DPLL with a Trace: From SAT to Knowledge Compilation. In Kaelbling, L. P.; and Saffiotti, A., eds., *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, 156–162. Professional Book Center.

Itsykson, D.; Knop, A.; Romashchenko, A. E.; and Sokolov, D. 2020. On OBDD-based Algorithms and Proof Systems that Dynamically Change the order of Variables. *J. Symb. Log.*, 85(2): 632–670.

Krajícek, J. 2008. An exponential lower bound for a constraint propagation proof system based on ordered binary decision diagrams. *J. Symb. Log.*, 73(1): 227–237.

Lagniez, J.; and Marquis, P. 2017. An Improved Decision-DNNF Compiler. In Sierra, C., ed., *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 667–673. ijcai.org.

Narodytska, N.; and Walsh, T. 2007. Constraint and Variable Ordering Heuristics for Compiling Configuration Problems. In Veloso, M. M., ed., *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 149–154.

Pipatsrisawat, K.; and Darwiche, A. 2008. New Compilation Languages Based on Structured Decomposability. In Fox, D.; and Gomes, C. P., eds., *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, 517–522. AAAI Press.

Pipatsrisawat, K.; and Darwiche, A. 2010. Top-Down Algorithms for Constructing Structured DNNF: Theoretical and Practical Implications. In Coelho, H.; Studer, R.; and Wooldridge, M. J., eds., *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, 3–8. IOS Press.

Segerlind, N. 2008. On the Relative Efficiency of Resolution-Like Proofs and Ordered Binary Decision Diagram Proofs. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity, CCC 2008, 23-26 June 2008, College Park, Maryland, USA*, 100–111. IEEE Computer Society.

Somenzi, F. 2009. CUDD: CU decision diagram package-release 2.4. 0. *University of Colorado at Boulder*.

Tveretina, O.; Sinz, C.; and Zantema, H. 2010. Ordered Binary Decision Diagrams, Pigeonhole Formulas and Beyond. *J. Satisf. Boolean Model. Comput.*, 7(1): 35–58.

Urquhart, A. 1987. Hard examples for resolution. *J. ACM*, 34(1): 209–219.

Van den Broeck, G.; and Darwiche, A. 2015. On the Role of Canonicity in Knowledge Compilation. In Bonet, B.; and Koenig, S., eds., *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, 1641–1648. AAAI Press.