On the Tractable Counting of Theory Models and its Application to Truth Maintenance and Belief Revision

Adnan Darwiche
Computer Science Department
University of California
Los Angeles, Ca 90095
darwiche@cs.ucla.edu

ABSTRACT. We address the problem of counting the models of a propositional theory, under incremental changes to the theory. Specifically, we show that if a propositional theory Δ is in a special form that we call smooth, deterministic, decomposable negation normal form (sd-DNNF), then for any consistent set of literals $\mathbf{S}$, *we can simultaneously count, in time linear in the size of* Δ, *the models of:*

- $\Delta \cup \mathbf{S}$;
- $\Delta \cup \mathbf{S} \cup \{l\}$: *for every literal* $l \notin \mathbf{S}$;
- $\Delta \cup \mathbf{S} \setminus \{l\}$: *for every literal* $l \in \mathbf{S}$;
- $\Delta \cup \mathbf{S} \setminus \{l\} \cup \{\neg l\}$: *for every literal* $l \in \mathbf{S}$.

We present two results relating to the time and space complexity of compiling propositional theories into sd-DNNF. First, we show that if a conjunctive normal form (CNF) has a bounded treewidth, then it can be compiled into an sd-DNNF in time and space which are linear in its size. Second, we show that propositional theories which can be polynomially represented using sd-DNNF form a strict superset of the theories which can be polynomially represented using FBDDs (a generalization of OBDDs that result from relaxing the variable–ordering restriction). Finally, we discuss some applications of the counting results to truth maintenance systems, belief revision, and model-based diagnosis (predicting the behavior of failed devices in particular).

KEYWORDS: Counting models, belief revision, truth maintenance, diagnosis, knowledge compilation.

1 Introduction

A propositional sentence is in negation normal form (NNF) if it is constructed from literals using only the conjoin and disjoin operators [1]. A practical representation of NNF sentences is in terms of rooted, directed acyclic graphs (DAGs), where each leaf node in the DAG is labeled with a literal, true or false; and each non-leaf (internal) node is labeled with a conjunction $\wedge$ or a disjunction $\vee$. Figure 1 depicts a rooted DAG representation of an NNF, where the children of each node are shown below it in the graph.[1]

We have recently proposed a *decomposability* property, which turns NNF into a highly tractable form, known as *decomposable negation normal form (DNNF)* [6, 7]. DNNF permits polynomial-time implementations of the following operations: deciding satisfiability, deciding clausal entailment, projection[2], computing minimum–cardinality[3], minimization according to minimum–cardinality[4], and model enumeration [6, 7].

In this paper, we identify two additional properties for DNNF, called *determinism* and *smoothness* and show a number of results about these two properties. First, given a propositional theory Δ in smooth, deterministic, DNNF (denoted sd-DNNF), we show that for any consistent set of literals $\mathbf{S}$, we can simultaneously count (in time linear in the size of Δ) the models of:

- $\Delta \cup \mathbf{S}$;
- $\Delta \cup \mathbf{S} \cup \{l\}$: for every literal $l \notin \mathbf{S}$;
- $\Delta \cup \mathbf{S} \setminus \{l\}$: for every literal $l \in \mathbf{S}$;
- $\Delta \cup \mathbf{S} \setminus \{l\} \cup \{\neg l\}$: for every literal $l \in \mathbf{S}$.

These counting operations allow one to develop linear-time, complete truth maintenance systems once a theory is put in sd-DNNF. Moreover, we show that sd-DNNF can be minimized in linear time, which allows one to develop linear-time, belief revision systems, and diagnostic systems that can predict the complete behavior of a failed device in time linear in its (sd-DNNF) model size. Our final set of results is related to the time and space complexity of compiling propositional theories into sd-DNNF. In particular, we show that if a conjunctive normal form (CNF) has a bounded treewidth, then it can be compiled into an sd-DNNF in time and space which are linear in its size (which is a strengthening of our results in [7]). Second, we show that propositional theories which can be polynomially represented using sd-DNNF form a strict

[1] We adopt this convention throughout the paper, therefore, eliminating the need to show the directionality of edges in DAGs.

[2] To project a theory on a set of atoms is to compute the strongest sentence entailed by the theory on these atoms.

[3] The cardinality of a model is the number of atoms that are set to false in the model. The minimum–cardinality of a theory is the minimum–cardinality of any of its models.

[4] To minimize a theory is to produce another theory whose models are exactly the minimum-cardinality models of the original theory.

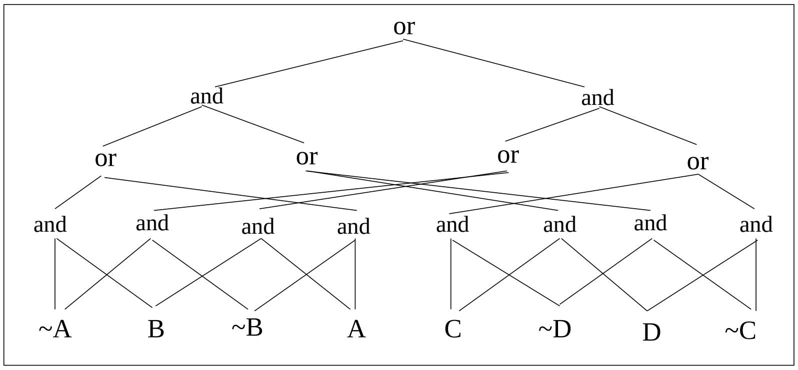

Figure 1: A theory in sd-DNNF. The theory has eight models (the odd-cardinality models): $\{\neg A, B, C, D\}$; $\{A, \neg B, C, D\}$; $\{A, B, \neg C, D\}$; $\{A, B, C, \neg D\}$; $\{\neg A, \neg B, \neg C, D\}$; $\{\neg A, \neg B, C, \neg D\}$; $\{\neg A, B, \neg C, \neg D\}$; and $\{A, \neg B, \neg C, \neg D\}$.

superset of the theories which can be polynomially represented using FBDDs (a generalization of OBDDs that result from relaxing the variable–ordering restriction) [4, 13, 17].

This paper is structured as follows. Section 2 reviews DNNF and introduces the class of smooth, deterministic DNNF. Section 3 presents the new operations for model counting based on this class of propositional theories. Sections 4-5 discuss the application of these counting operations to truth maintenance, belief revision, and model-based diagnosis. Section 6 presents the compilation algorithms from CNF into sd-DNNF and discusses related complexity results. Section 7 closes with some concluding remarks. Proofs of all results are given in Appendix B.

2 Smooth, Deterministic DNNF

We start with the formal definition of NNF, which is the basis of other logical forms we define next.

Definition 1 *Let $\mathbf{X}$ be a finite set of propositional atoms. A sentence in NNF is a rooted, directed acyclic graph (DAG) where each leaf node is labeled with* true*,* false*, X or $\neg X$, $X \in \mathbf{X}$; and each internal node is labeled with $\wedge$ or $\vee$. The size of a sentence in NNF is the number of its DAG edges.*

We will now present three conditions on NNF. The first of which is decomposability and is probably the most influential:

Definition 2 (Decomposability) *[6, 7] A decomposable NNF, written DNNF, is an NNF Δ satisfying the following property: for every conjunction $\alpha =$*

$\alpha_1 \wedge \ldots \wedge \alpha_n$ appearing in Δ, no atoms are shared between the conjuncts of α; that is, $atoms(\alpha_i) \cap atoms(\alpha_j) = \emptyset$ for $i \neq j$.

The NNF $(A \vee B) \wedge (\neg A \vee C)$ is not decomposable since atom A is shared by the two conjuncts. But the NNF in Figure 1 is decomposable. It has ten conjunctions and the conjuncts of each share no atoms.

Decomposability is a very powerful property as it permits a number of interesting logical operations to be implemented in polynomial-time on DNNF, including satisfiability, clausal entailement, projection, minimization and model enumeration [6, 7].

We now introduce another property of NNF, which permits even more operations to be implemented in polynomial-time:

Definition 3 (Determinism) *A <u>deterministic</u> NNF, written d-NNF, is an NNF Δ satisfying the following property: for every disjunction $\alpha = \alpha_1 \vee \ldots \vee \alpha_n$ appearing in Δ, every pair of disjuncts in α is logically inconsistent; that is, $\alpha_i \wedge \alpha_j \models \mathsf{false}$ for $i \neq j$.*

For example, $(A \wedge B) \vee C$ is an NNF but is not deterministic since the disjuncts $A \wedge B$ and C are consistent. However, the NNF $(A \wedge B) \vee (\neg A \wedge C)$ is deterministic.[5] A deterministic, decomposable NNF will be denoted by d-DNNF. As we shall see, determinism permits one to count the number of theory models in polynomial time. It also allows one to enumerate models in output linear time.

Our last property on NNF is that of smoothness. It is not a crucial property except that it simplifies the formal treatment of NNF operations considerably.

Definition 4 (Smoothness) *A <u>smooth</u> NNF, written s-NNF, is an NNF Δ satisfying the following property: for every disjunction $\alpha = \alpha_1 \vee \ldots \vee \alpha_n$ in Δ, we have $atoms(\alpha) = atoms(\alpha_i)$ for every i.*

A smooth, deterministic, decomposable NNF will be denoted by sd-DNNF.

The d-DNNF in Figure 1 is smooth. We can easily smooth an NNF as follows. For each disjunction $\alpha = \alpha_1 \vee \ldots \vee \alpha_n$, if $atoms(\alpha_i) \neq atoms(\alpha)$, we can replace the disjunct α_i in α with $\alpha_i \wedge \bigwedge_{A \in \Sigma}(A \vee \neg A)$, where $\Sigma = atoms(\alpha) - atoms(\alpha_i)$. This operation preserves equivalence, decomposability and determinism of an NNF. Moreover, the size of resulting s-NNF is $O(mn)$, where n is the size of original NNF and m is the number of propositional atoms.

One way to view the sd-DNNF representation of a theory Δ is as a nested factoring of the sum-of-product representation of Δ. The sd-DNNF representation gives us most of the powerful properties of the sum-of-product representation, yet it does not require as much space. To appreciate this point, we list a number of theories in Table 1 together with the size of their CNF representation, the size of their sd-DNNF representation, and the number of their

[5] Note that although every disjunctive normal form (DNF) is a DNNF, not every DNF is a d-DNNF.

Theory	Models#	CNF Vars#/Cls#	Nodes#	Edges#	sd-DNNF Nodes#	Edges#
Solenoid	27	11/19	42	68	84	172
Valve	24	13/50	77	241	83	171
Four-pipes	25	27/110	165	395	126	302
Chandra21	2097152	24/21	70	102	82	176
Chandra24	16777216	27/24	79	120	87	272
Adder-4	256	17/57	92	258	110	241
Adder-100	2^{200}	401/1401	2204	6402	3182	8017

Table 1: Some propositional theories and their CNF and sd-DNNF representations. Note that CNF is a special class of NNF, hence, the size of a CNF is reported as the size of its corresponding NNF representation.

models.[6] The theories were obtained from *http://www.lifl.fr/KC/*. Note that some of these theories have a very large number of models, yet a very compact sd-DNNF representation. In fact, for most theories in this case, the size of the sd-DNNF representation is very close to the size of the CNF representation, if not better.

We close this section by showing how the models of an sd-DNNF can be enumerated in output linear time, which is a slight improvement on the model enumeration of DNNF [7]. Specifically, suppose that a model is represented as a set of literals and suppose that $\times$ is Cartesian product on sets of models:

$$\{N_1,\ldots,N_n\} \times \{M_1,\ldots,M_m\} \stackrel{def}{=} \{N_1 \cup M_1,\ldots,N_n \cup M_m\}.$$

Then the models of sd-DNNF can be enumerated as follows:

1. $\mathsf{models}(l) = \{\{l\}\}$ where l is a literal.
2. $\mathsf{models}(\alpha_1 \vee \ldots \vee \alpha_n) = \mathsf{models}(\alpha_1) \cup \ldots \cup \mathsf{models}(\alpha_n)$.
3. $\mathsf{models}(\alpha_1 \wedge \ldots \wedge \alpha_n) = \mathsf{models}(\alpha_1) \times \ldots \times \mathsf{models}(\alpha_n)$.

By decomposability, two sets of partial models are multiplied by $\times$ only if they share no atoms. By determinism, two sets of partial models are unioned only if they contain no duplicates. Hence, the time and space complexity of the above procedure is linear in the number of models returned.

3 Counting the Models of sd-DNNF

The polynomial-time operations on NNF which are permitted by the decomposability property have been discussed at length elsewhere [6, 7]. In this

[6]These sd-DNNFs have been computed using a variation of the algorithm in Section 6, which is optimized using unit resolution.

section, we discuss the new operations permitted by the additional properties of determinism and smoothness. As we shall see, these operations on sd-DNNF have implications on a number of AI applications, including truth maintenance, belief revision and diagnosis. Specifically, given an sd-DNNF Δ and a set of literals $\mathbf{S}$, we describe two traversal operations each taking linear time. By the end of the first traversal, we will be able to count the models of $\Delta \cup \mathbf{S}$. By the end of the second traversal, we will be able to count the models of:

1. $\Delta \cup \mathbf{S} \cup \{l\}$ for every literal $l \notin \mathbf{S}$;
2. $\Delta \cup \mathbf{S} \setminus \{l\}$ for every literal $l \in \mathbf{S}$;
3. $\Delta \cup \mathbf{S} \setminus \{l\} \cup \{\neg l\}$ for every literal $l \in \mathbf{S}$.

That is, once we traverse the sd-DNNF twice, we will be able to obtain each of these counts using constant-time, lookup operations.

The traversal will not take place on the sd-DNNF itself, but on a secondary structure that we call the *counting graph.* The counting graph of an sd-DNNF is a function of many variables represented as a rooted DAG.

Definition 5 *The counting graph of an sd-DNNF is a labeled, rooted DAG. It contains a node labeled with l for each literal l, a node labeled with $+$ for each or-node, and a node labeled with $*$ for each and-node in the sd-DNNF. There is an edge between two nodes in the counting graph iff there is an edge between their corresponding nodes in the sd-DNNF.*

Figure 2 depicts the counting graph of the sd-DNNF in Figure 1. The size of a counting graph is therefore equal to the size of its corresponding sd-DNNF. We will see now how such a graph can be used to perform the counting operations we are interested in.

Definition 6 *The value of a node N in a counting graph under a consistent set of literals $\mathbf{S}$ is defined as follows:*

- $\text{VAL}(N) = 0$ *if N is labeled with literal l and $\neg l \in \mathbf{S}$;*
- $\text{VAL}(N) = 1$ *if N is labeled with literal l and $\neg l \notin \mathbf{S}$;*
- $\text{VAL}(N) = \prod_i \text{VAL}(N_i)$ *if N is labeled with $*$, where N_i are the children of N;*
- $\text{VAL}(N) = \sum_i \text{VAL}(N_i)$ *if N is labeled with $+$, where N_i are the children of N.*

The value of a counting graph G under literals $\mathbf{S}$, written $G(\mathbf{S})$, is the value of its root under $\mathbf{S}$.

From here on, we will use G to denote both the counting graph itself and the function represented by this graph.

Here's our first counting result.

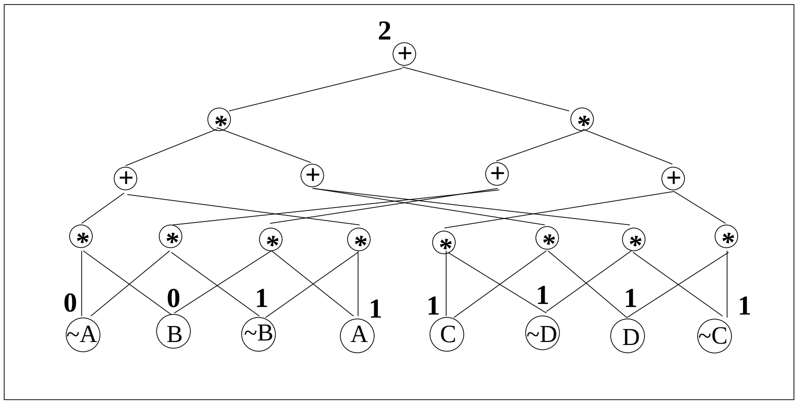

Figure 2: A counting graph of the sd-DNNF Δ in Figure 1 evaluated under $\mathbf{S} = A, \neg B$. The evaluation indicates that $\Delta \cup \mathbf{S}$ has two models ($\{A, \neg B, C, D\}$ and $\{A, \neg B, \neg C, \neg D\}$ in this case).

Theorem 1 *Let Δ be an sd-DNNF, $\mathbf{S}$ be a consistent set of literals, and let G be the counting graph of Δ. The value of G under $\mathbf{S}$ is the number of models of $\Delta \cup \mathbf{S}$:*

$$G(\mathbf{S}) = \mathit{Models}\#(\Delta \cup \mathbf{S}).$$

Note that $G(\mathbf{S}) > 0$ iff $\Delta \cup \mathbf{S}$ is consistent. Therefore, by traversing the counting graph once we can test the consistency of sd-DNNF Δ conjoined with any set of literals $\mathbf{S}$. Figure 2 depicts the counting graph of the sd-DNNF Δ in Figure 1, evaluated under the literals $\mathbf{S} = A, \neg B$. This indicates that $\Delta \cup \{A, \neg B\}$ has two models.

We now present the central result in this paper. First, we note that when viewing a counting graph G as a function of many variables, we will use V_l to denote the variable (node) labeled with literal l. Second, we can talk about the partial derivative of function G with respect to any of its variables V_l, $\partial G / \partial V_l$ and we can also talk about the value of this derivative under a set of literals $\mathbf{S}$, $\partial G(\mathbf{S}) / \partial V_l$. Due to the decomposability of DNNF, the function G is linear in each of its variables. Therefore, the change to the count $G(\mathbf{S})$ as a result of adding, removing or flipping a literal in $\mathbf{S}$ can be obtained from the partial derivatives, without having to re-evaluate the counting graph G. This leads to the following result:

Theorem 2 *Let Δ be an sd-DNNF, $\mathbf{S}$ be a consistent set of literals, and let G be the counting graph of Δ. We have:*

Assertion: *When $\neg l \notin \mathbf{S}$:*

$$\mathit{Models}\#(\Delta \cup \mathbf{S} \cup \{l\}) = \partial G(\mathbf{S}) / \partial V_l.$$

Retraction: *When $l \in \mathbf{S}$:*

$$\textit{Models\#}(\Delta \cup \mathbf{S} \setminus \{l\}) = \partial G(\mathbf{S})/\partial V_l + \partial G(\mathbf{S})/\partial V_{\neg l}.$$

Flipping: *When $l \in \mathbf{S}$:*

$$\textit{Models\#}(\Delta \cup \mathbf{S} \setminus \{l\} \cup \{\neg l\}) = \partial G(\mathbf{S})/\partial V_{\neg l}.$$

Therefore, if we can compute the partial derivatives of G with respect to each of its variables, and evaluate these derivatives at $\mathbf{S}$, we can then count the models of $\Delta \cup \mathbf{S}$ under the assertion of new literals not in $\mathbf{S}$, and under the retraction or flipping of literals in $\mathbf{S}$. Figure 3 depicts the value of each of these partial derivatives for the sd-DNNF in Figure 1. The counting graph is evaluated under literals $\mathbf{S} = A, \neg B, C$ and the partial derivatives are shown below each variable. According to these derivatives, we have:

Assertion: $\textit{Models\#}(\Delta \cup \mathbf{S} \cup \{D\}) = 1$ and $\textit{Models\#}(\Delta \cup \mathbf{S} \cup \{\neg D\}) = 0$. This immediately tells us that $\Delta \cup \mathbf{S} \models D$.

Retraction: $\textit{Models\#}(\Delta \cup \mathbf{S} \setminus \{A\}) = 1 + 1 = 2$; $\textit{Models\#}(\Delta \cup \mathbf{S} \setminus \{\neg B\}) = 1 + 1 = 2$; and $\textit{Models\#}(\Delta \cup \mathbf{S} \setminus \{C\}) = 1 + 1 = 2$. Therefore, retracting any literal in $\mathbf{S}$ increases the number of models to 2.

Flipping: $\textit{Models\#}(\Delta \cup \mathbf{S} \setminus \{A\} \cup \{\neg A\}) = 1$; $\textit{Models\#}(\Delta \cup \mathbf{S} \setminus \{\neg B\} \cup \{B\}) = 1$; $\textit{Models\#}(\Delta \cup \mathbf{S} \setminus \{C\} \cup \{\neg C\}) = 1$. Therefore, flipping any literal in $\mathbf{S}$ will not change the number of models (although it does change the model itself).

For another example, consider Figure 4(a) which depicts an sd-DNNF for the theory $\Delta = (A \supset B) \wedge (B \supset C)$, and Figure 4(b) which depicts its corresponding counting graph. The value of this counting graph for $\mathbf{S} = \emptyset$ is shown, and its partial derivatives are also computed. According to this Figure, Δ has 4 models. Moreover, by Theorem 2:

- $\Delta \cup \{A\}$ has 1 model since $\partial G(\mathbf{S})/\partial V_A = 1$. The model is: $\{A, B, C\}$.
- $\Delta \cup \{\neg C\}$ has 1 model since $\partial G(\mathbf{S})/\partial V_{\neg C} = 1$. The model is: $\{\neg A, \neg B, \neg C\}$.
- $\Delta \cup \{B\}$ has 2 models since $\partial G(\mathbf{S})/\partial V_B = 2$. The models are: $\{A, B, C\}$ and $\{\neg A, B, C\}$.

There is one missing link now: How do we compute the partial derivatives of a counting graph with respect to each of its variables? This actually turns out to be easy due to results in [14, 16] which show how to evaluate and simultaneously compute all partial derivatives of a function by simply traversing its *computation graph* in linear time. Although [16] casts such computation in terms of summing weights of paths in such a graph, we present a more direct implementation here. In particular, if we let PD(N) denote the partial

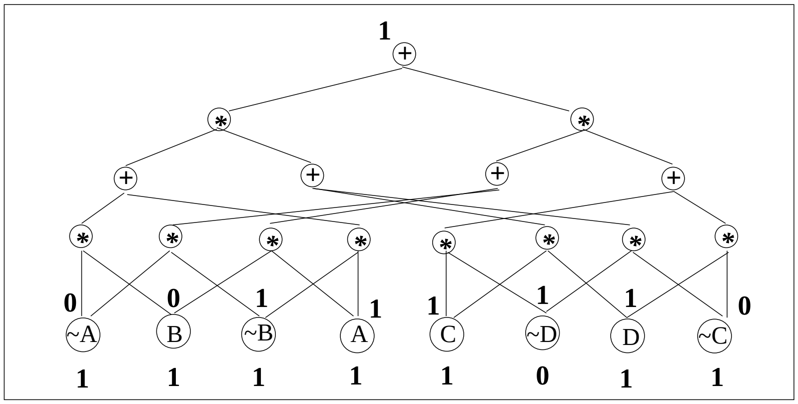

Figure 3: A counting graph of the sd-DNNF Δ in Figure 1 evaluated under literals $\mathbf{S} = A, \neg B, C$, indicating one model of $\Delta \cup \mathbf{S}$ ($\{A, \neg B, C, D\}$ in this case). Partial derivatives are shown below the leaves.

derivative of G with respect to a node N in the counting graph, then $\textsc{pd}(N)$ is the summation of contributions made by parents M of N:

$$\textsc{pd}(N) \quad = \quad \begin{cases} 1, & \text{N is the root node;} \\ \sum_{M} \textsc{cpd}(M, N), & \text{otherwise;} \end{cases}$$

where the contribution of parent M to its child N is computed as follows:

$$\textsc{cpd}(M, N) = \begin{cases} \textsc{pd}(M), & M \text{ is a } +\text{node;} \\ \textsc{pd}(M) \prod_{K \neq N} \textsc{val}(K), & M \text{ is a } \star\text{node;} \end{cases}$$

where K is a child of M. This computation can be performed by first traversing the counting graph once to evaluate it, assigning $\textsc{val}$ to each node N, and then traversing it a second time, assigning $\textsc{pd}$ for each node N. We are then mainly interested in $\textsc{val}(N)$ where N is the root node, and $\textsc{pd}(N)$ where N is a leaf node.

Therefore, both the value of a counting graph under some literals $\mathbf{S}$ and the values of each of its partial derivatives under $\mathbf{S}$ can be computed by traversing the graph twice. Once to compute the values, and another to compute the partial derivatives.

We close this section by pointing out that partial differentiation turns out to play a key role in probabilistic reasoning as well. Specifically, we present a comprehensive framework for probabilistic reasoning in [8] based on compiling a Bayesian network into a polynomial and then reducing a large number of

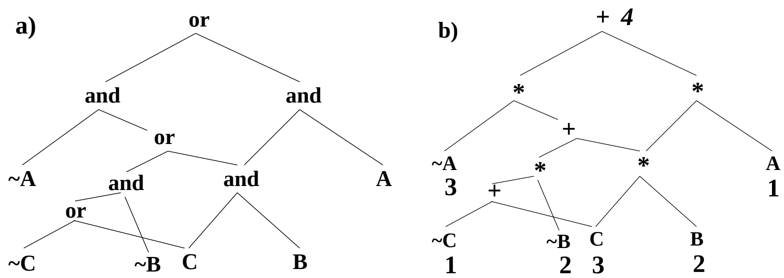

Figure 4: An sd-DNNF for the theory $A \supset B, B \supset C$, and its corresponding counting graph evaluated and differentiated under $\mathbf{S} = \emptyset$.

probabilistic queries into the computation of partial derivatives of the compiled polynomial.

4 Complete, Linear-Time Truth Maintenance

We now turn to some applications of the results in the previous section. We first consider truth maintenance systems and show how our results allow us to implement complete truth maintenance systems which take time linear in the size of an sd-DNNF theory.

A truth maintenance system takes a set of clauses Γ and a set of literals $\mathbf{S}$ and tries to determine for each literal l whether $\Gamma \cup \mathbf{S} \models l$. The most common truth maintenance system is the one based on closing $\Gamma \cup \mathbf{S}$ under unit resolution [12]. Such a system takes linear time, but is incomplete. Given that the set of literals in $\mathbf{S}$ changes to $\mathbf{S}'$, the goal of a truth maintenance system is then to update the truth of each literal under the new "context" $\mathbf{S}'$. Sometimes, clauses in Γ are retracted and/or asserted. A truth maintenance system is expected to update the truth of literals under such clausal changes too.

Our model-counting results allow us to implement a complete truth maintenance system as follows. We compile the theory Γ into an sd-DNNF Δ and construct the counting graph G of Δ. Given any set of literal $\mathbf{S}$, we evaluate G under $\mathbf{S}$ and compute its partial derivatives also under $\mathbf{S}$. This can be done in time linear in the size of Δ. We are now ready to answer all queries expected from a truth maintenance system by simple, constant-time, look-up operations:

Literal l is entailed by $\Delta \cup \mathbf{S}$ iff $\Delta \cup \mathbf{S} \cup \{\neg l\}$ has no models: $\partial G(\mathbf{S})/\partial V_{\neg l} = 0$.

Retracting literal l from $\mathbf{S}$ will render $\Delta \cup \mathbf{S}$ consistent iff $\Delta \cup \mathbf{S} \setminus \{l\}$ has at least one model: $\partial G(\mathbf{S})/\partial V_l + \partial G(\mathbf{S})/\partial V_{\neg l} > 0$.

Flipping literal l in $\mathbf{S}$ will render $\Delta \cup \mathbf{S}$ consistent iff $\Delta \cup \mathbf{S} \setminus \{l\} \cup \{\neg l\}$ has at least one model: $\partial G(\mathbf{S})/\partial V_{\neg l} > 0$. [7]

If we want to reason about the effect of asserting and retracting clauses in theory Γ, we can replace each clause α in Γ by $C_\alpha \supset \alpha$, where C_α is a new atom. We then compile the extended theory Γ into sd-DNNF Δ. To assert all clauses initially, we have to include all atoms C_α in the set of literals $\mathbf{S}$. The assertion/retraction of clauses can then be emulated by the assertion/retraction of atoms C_α. For example, in case of a contradiction, we can ask whether retracting a clause α will resolve the contradiction by asking whether $\Delta \cup \mathbf{S} \setminus \{C_\alpha\}$ has more than one model:

$$\partial G(\mathbf{S})/\partial V_{C_\alpha} + \partial G(\mathbf{S})/\partial V_{\neg C_\alpha} > 0.$$

5 Complete, Linear-Time Belief Revision

We now turn to a second major application of model counting on sd-DNNF: the implementation of a very common class of belief revision systems, which is adopted in model-based diagnosis and in certain forms of default reasoning [15, 11]. The problem here is as follows. We have a set of special atoms $\mathbf{D} = \{d_1, \ldots, d_n\}$ in the theory Δ which represent defaults. Typically, we assume that all of these defaults are true, allowing us to draw some default conclusions. We then receive some evidence $\mathbf{S}$ (a set of consistent literals) which is inconsistent with $\Delta \cup \mathbf{D}$. We therefore know that not all defaults are true and some must be retracted—that is, some d_is will have to be replaced by $\neg d_i$ in $\mathbf{D}$. Our goal then is to identify a set of literals $\mathbf{D}'$ such that

1. $d_i \in \mathbf{D}'$ or $\neg d_i \in \mathbf{D}'$ for all i;
2. $\Delta \cup \mathbf{D}' \cup \mathbf{S}$ is consistent;
3. the number of negative literals in $\mathbf{D}'$ is minimized;

and then report the truth of every literal under the new set of defaults $\mathbf{D}'$. Note that there may be more than one set of defaults $\mathbf{D}'$ that satisfies the above properties. In such a case, a literal holds after the revision process only if it holds under $\Delta \cup \mathbf{D}' \cup \mathbf{S}$ for every $\mathbf{D}'$.

This revision process is a special case of what is known as *cardinality-maximizing base revision.* Deciding entailment after such a revision process is known to be $\Delta_2^p[O(\log n)]$-complete [15, 11].

One way to formalize this revision process is through $\mathbf{D}$-minimization, which is a generalization of the operation of minimization that we defined in [7] (where $\mathbf{D}$ contains all atoms).

Definition 7 *If $\mathbf{D}$ is a set of atoms, then the* $\underline{\mathbf{D}\text{-}\textit{cardinality}}$ *of a model m, written* $\mathsf{card}(m, \mathbf{D})$*, is the number of atoms in $\mathbf{D}$ that the model sets to false.*

[7] Note that the flipping of literals is outside the scope of classical truth maintenance systems in the sense that they must retract l and then assert $\neg l$, taking linear time, to perform the above operation.

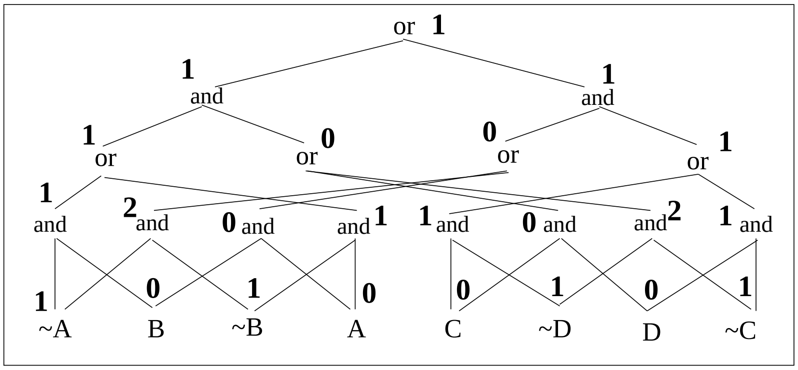

Figure 5: Assigning a $\mathbf{D}$-cardinality to each node in an sd-DNNF with $\mathbf{D} = \{A, B, C, D\}$.

Definition 8 *Let Δ be a propositional theory and let $\mathbf{D}$ be a set of atoms in Δ. The <u>$\mathbf{D}$-minimization</u> of theory Δ, written* $\mathsf{minimize}(\Delta, \mathbf{D})$*, is defined as a theory with the following models:*

$$\mathsf{models}(\mathsf{minimize}(\Delta, \mathbf{D})) = \{m \models \Delta : m' \models \Delta \textit{ only if } \mathsf{card}(m, \mathbf{D}) \leq \mathsf{card}(m', \mathbf{D})\}.$$

Here, the models m and m' are over atoms(Δ).

Therefore, the belief revision process can be viewed as $\mathbf{D}$-minimizing $\Delta \cup \mathbf{S}$ and then closing the result under literal entailment. We will next show how to conjoin an sd-DNNF Δ with literals $\mathbf{S}$, and then how to $\mathbf{D}$-minimize the result while guaranteeing that the final outcome remains to be an sd-DNNF. We can then use the technique discussed in the previous sections to close the minimization under literal entailment.

First, we show how to conjoin an sd-DNNF Δ with a set of literals $\mathbf{S}$ to yield another sd-DNNF equivalent to $\Delta \cup \mathbf{S}$ (which we will also write as $\Delta \wedge \mathbf{S}$). For this we need the notion of conditioning:

Definition 9 (Conditioning) *[6, 7] Let Δ be a propositional sentence, and let $\mathbf{S}$ be a consistent set of literals. The conditioning of Δ on $\mathbf{S}$, written $\Delta \mid \mathbf{S}$, is the sentence obtained by replacing each atom X of Δ by* true *(resp.* false*) if X (resp. $\neg X$) is a positive (resp. negative) literal of $\mathbf{S}$.*

For example, conditioning the DNNF $(\neg A \wedge \neg B) \vee (B \wedge C)$ on $\{B, D\}$ gives $(\neg A \wedge \neg \mathsf{true}) \vee (\mathsf{true} \wedge C)$, and conditioning it on $\{\neg B, D\}$ gives $(\neg A \wedge \neg \mathsf{false}) \vee (\mathsf{false} \wedge C)$. In what follows, we assume that $\neg\mathsf{true}$ will be written as false and that $\neg\mathsf{false}$ will be written as true.

Theorem 3 *[6, 7] $\Delta \wedge \mathbf{S}$ is equivalent to $(\Delta \mid \mathbf{S}) \wedge \mathbf{S}$.*

Theorem 4 *If Δ is sd-DNNF, then $(\Delta \mid \mathbf{S}) \wedge \mathbf{S}$ is sd-DNNF.*

Therefore, decomposability, determinism and smoothness are all closed under conditioning. Theorem 3 can then be used to conjoin any sd-DNNF with a consistent set of literals, to yield another sd-DNNF representing the conjunction.

We now describe the process of $\mathbf{D}$-minimizing an sd-DNNF Δ, which is done in two steps. First, we assign a cardinality to each node in the sd-DNNF. Then we prune some of the edges connecting or-nodes to their children.

The following two theorems follow immediately from smoothness, determinism, and decomposability.

Theorem 5 *Consider the following function,* $\mathsf{mCard}(., \mathbf{D})$*, defined on NNF:*

- $\mathsf{mCard}(l, \mathbf{D}) = 1$ *if l is a negative literal whose atom is in $\mathbf{D}$, and* $= 0$ *otherwise.*
- $\mathsf{mCard}(\alpha_1 \vee \ldots \vee \alpha_n, \mathbf{D}) = \min(\mathsf{mCard}(\alpha_1, \mathbf{D}), \ldots, \mathsf{mCard}(\alpha_n, \mathbf{D}))$.
- $\mathsf{mCard}(\alpha_1 \wedge \ldots \wedge \alpha_n, \mathbf{D}) = \mathsf{mCard}(\alpha_1, \mathbf{D}) + \ldots + \mathsf{mCard}(\alpha_n, \mathbf{D})$.

If Δ is an sd-DNNF, then $\mathsf{mCard}(\Delta, \mathbf{D}) = min_{m \models \Delta} \mathsf{card}(m, \mathbf{D})$ where m is a model over $atoms(\Delta)$.

Note that by traversing an sd-DNNF only once, we are able to compute $\mathsf{mCard}(\alpha, \mathbf{D})$ for every node α in the sd-DNNF.

Theorem 6 *We have the following:*

- $\mathsf{minimize}(l, \mathbf{D}) = l$ *if l is a literal.*
- $\mathsf{minimize}(\bigvee_i \alpha_i, \mathbf{D}) = \bigvee_i \mathsf{minimize}(\alpha_i, \mathbf{D})$*, where* $\mathsf{mCard}(\alpha, \mathbf{D}) = \mathsf{mCard}(\alpha_i, \mathbf{D})$.
- $\mathsf{minimize}(\bigwedge_i \alpha_i, \mathbf{D}) = \bigwedge_i \mathsf{minimize}(\alpha_i, \mathbf{D})$.

Therefore, once we have computed $\mathsf{mCard}(\alpha, \mathbf{D})$ for every node α in an sd-DNNF, we can traverse the sd-DNNF once and end up $\mathbf{D}$-minimizing it by simply disconnecting the edges between any or-node and its children that have higher cardinality. Note that this minimization process will preserve decomposability, smoothness and determinism.

Figure 5 depicts the result of assigning cardinalities to the sd-DNNF of Figure 1, and Figure 6 depicts the result of deleting some of its edges. This is the minimized sd-DNNF and it has four models: $\{\neg A, B, C, D\}$; $\{A, \neg B, C, D\}$; $\{A, B, \neg C, D\}$; and $\{A, B, C, \neg D\}$.

We close this section by pointing out that the above results have direct application to model-based diagnosis, where Δ is the device description, $\mathbf{S}$ is the device observation and $\mathbf{D}$ contains the health modes $ok_1, \ldots, ok_n$. Initially, we assume that all device components are working normally, but then find some observation $\mathbf{S}$ such that $\Delta \cup \mathbf{D} = \{ok_1, \ldots, ok_n\} \cup \mathbf{S}$ is inconsistent.

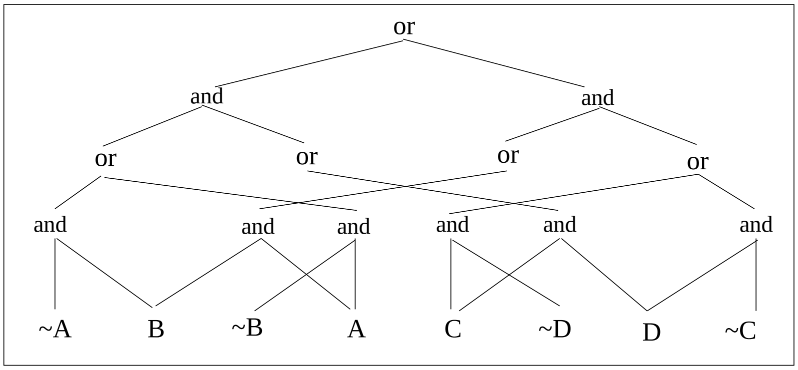

Figure 6: A minimized sd-DNNF.

To regain consistency we must postulate that some of the components are not healthy, therefore, flipping some of the ok_is into $\neg ok_i$ in the set $\mathbf{D}$. Assuming a smallest number of faults, we want to minimize the number of unhealthy components needed to regain consistency. A set $\mathbf{D'}$ such that:

1. $ok_i \in \mathbf{D'}$ or $\neg ok_i \in \mathbf{D'}$ for all i;
2. $\Delta \cup \mathbf{D'} \cup \mathbf{S}$ is consistent;
3. the number of negative literals in $\mathbf{D'}$ is minimized;

is called a *minimum-cardinality diagnosis* [5, 9].

Traditionally, the main goal of model-based diagnosis has been to enumerate such diagnoses. Another practical problem, however, which has received much less attention in model-based diagnosis is that of predicting the behavior of the device (the values of its ports) given an abnormal observation. If the observation $\mathbf{S}$ is normal, then $\Delta \cup \mathbf{D} \cup \mathbf{S}$ is consistent, and all we need is to close this theory under literal entailment to figure out the value of each port in the device. However, if the observation $\mathbf{S}$ is abnormal, then $\Delta \cup \mathbf{D} \cup \mathbf{S}$ is inconsistent, and we need is to close $\mathsf{minimize}(\Delta \cup \mathbf{S}, \mathbf{D})$ under literal entailment as shown above. This allows us to predict the value of each device port under the (default) assumption that a smallest number of faults have actually materialized in the broken device. All of this can be done in linear time once the original device model is made available in sd-DNNF.

6 Compiling sd-DNNF

We present two complexity results in this section.

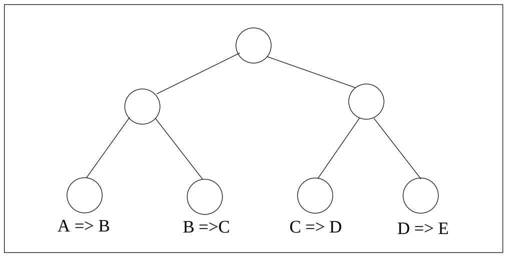

Figure 7: A decomposition tree for the theory $A \supset B$, $B \supset C$, $C \supset D$, $D \supset E$.

The first result is a strengthening of our result in [6, 7], where we presented an algorithm for compiling a CNF into a DNNF. We show here that the presented algorithm can be modified slightly so it will also ensure the smoothness and determinism of the resulting NNF, without affecting its time and space complexity.

The second result is a strengthening of a result we have in [7], where we showed that:

1. if a propositional theory has a polynomially-sized FBDD representation, then it has a polynomially-sized DNNF representation;

2. the opposite is not true.

Here we show that this result is true even with the addition of the smoothness and determinism properties. Therefore, sd-DNNFs are a strictly more efficient representation than FBDDs, which are an important generalization of OBDDs (in which the restriction of a fixed-variable ordering is removed) [4, 13, 17].

We start with our first result, which is an extension of a structure-based algorithm that we introduced in [6, 7] for converting a CNF into a DNNF. The algorithm utilizes a *decomposition tree,* which is a binary tree the leaves of which correspond to the CNF clauses—see Figure 7. Each decomposition tree has a width and the complexity of the algorithm is exponential only in the width of used tree. The algorithm is given with a slight modification (on Line 04) in Figure 8. The pseudocode is explained in Appendix A for completeness. Our main concern here, however, is the following result:

Theorem 7 *Let N be the root of decomposition tree T used in Figure 8. Then* CNF2SDDNNF(N, true) *will return Δ in sd-DNNF, where Δ contains the clauses attached to the leaves of T. Moreover, the time and space complexity of the algorithm is $O(nw2^w)$, where n is the number of clauses in Δ and w is the width of decomposition tree T.*

Algorithm cnf2sddnnf

/* N is a node in a decomposition tree */
/* α is an instantiation */

$\text{CNF2SDDNNF}(N, \alpha)$
01. $\psi \leftarrow project(\alpha, atoms(N))$
02. if $\text{CACHE}_N(\psi) \neq \text{NIL}$, return $\text{CACHE}_N(\psi)$
03. if N is a leaf node,
04. then $\gamma \leftarrow \text{CL2SDDNNF}(N, \alpha)$
05. else $\gamma \leftarrow \bigvee_\beta \text{CNF2SDDNNF}(N_l, \alpha \wedge \beta) \wedge$
06. $\text{CNF2SDDNNF}(N_r, \alpha \wedge \beta) \wedge \beta$
07. where β ranges over all instantiations
08. of $atoms(N_l) \cap atoms(N_r) - atoms(\alpha)$
09. $\text{CACHE}_N(\psi) \leftarrow \gamma$
10. return γ

Figure 8: Compiling a CNF into sd-DNNF.

The width of a decomposition tree is defined in Appendix A. The class of CNF theories with bounded treewidth is also given in Appendix A, and we have shown in [6, 7] that for this class of theories, one can construct in linear time a decomposition tree of bounded width. Therefore, one can compile an sd-DNNF of linear size for this class of theories. Note that the CNF representation of n-bit adders has a bounded treewidth, hence, such device descriptions can be easily compiled into sd-DNNF (see Table 1).

We now turn to our second result, relating to Binary Decision Diagrams (BDDs) which are among the most successful representations of propositional theories [4]. A BDD is indeed only a special case of an NNF, which is depicted using specialized notation [7]. Figure 9 depicts a BDD which is a DAG in which each leaf node is labeled with 0 or 1, and each internal node is labeled with an atom and has two children, labeled low (dashed edge) and high (solid edge).

Each node N in a BDD which is labeled with atom A represents a disjunction $(A \wedge \alpha) \vee (\neg A \wedge \beta)$, where α is the NNF represented by the high-child of node N and β is the NNF represented by the low-child of node N. The leaf node 0 represents false and the leaf node 1 represents true. If a BDD has n nodes, then it can be represented by an NNF of size $O(n)$.

It also follows immediately that each NNF which results from this translation is deterministic. Therefore, BDDs are a special case of deterministic NNFs. A Free BDD (FBDD) [13, 17] is a BDD where each atom appears only once on any path from the root of the BDD to any of its leaves. It is not hard to see that this condition implies that the NNF corresponding to an FBDD is also decomposable [7]. Therefore, FBDDs are a special case of deterministic,

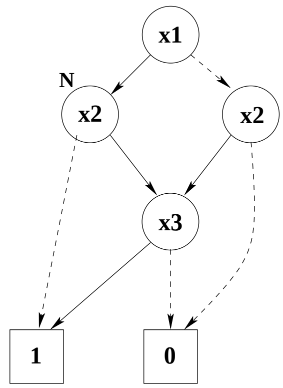

Figure 9: A BDD. High edges are solid, low edges are dashed. The NNF corresponding to node N is: $(\neg x_2 \wedge \mathsf{true}) \vee (x_2 \wedge (x_3 \wedge \mathsf{true} \vee \neg x_3 \wedge \mathsf{false}))$, which simplifies to $\neg x_2 \vee (x_2 \wedge x_3)$.

decomposable NNFs. Since any NNF can be smoothed in polynomial time, we get the first result that any propositional theory that has a polynomial FBDD representation also has a polynomial sd-DNNF representation. The opposite is not true, however.

Theorem 8 *[3] There are propositional theories Δ_1 and Δ_2 such that:*

- *each of Δ_1 and Δ_2 has a polynomial FBDD representation;*
- *$\Delta_1 \wedge \Delta_2$ is inconsistent.*
- *$\Delta_1 \vee \Delta_2$ has no polynomial FBDD representation.*

Since Δ_1 and Δ_2 have polynomial FBDD representations, they must also have polynomial sd-DNNF representations. Let us call these representations Γ_1 and Γ_2. It immediately follows that $\Gamma_1 \vee \Gamma_2$ is decomposable, deterministic NNF, which can be smoothed in polynomial-time, leading to a polynomially-sized sd-DNNF representation of the theory $\Delta_1 \vee \Delta_2$.

We close this section by pointing out that since smooth, deterministic, decomposable NNF admits a large number of polynomial-time operations, the scalability of this representation may be questionable (the size of theories when compiled into sd-DNNF). The above results, however, shed strong light on this scalability, showing that sd-DNNFs are strictly more space efficient than FBDDs, which are strictly more space efficient than OBDDs. The latter representation, however, is viewed among the most practical representations of propositional theories, and has been finding increasing usage in AI applications, such as planning, diagnosis and Markov decision processes.

7 Conclusion

We have identified the class of decomposable negation normal form, DNNF, in previous work and showed that it is highly tractable by identifying a large set of influential logical operations which take polynomial time on DNNF. In this paper, we have identified two extra properties, smoothness and determinism, that increase the tractability of DNNF, allowing one to count the models of a theory in time linear in its size and under incremental changes to its content (addition/removal/flipping of literals).

Interestingly enough, two of our complexity results on DNNF continue to hold for the class of smooth, deterministic DNNF, shedding new light on the scalability of this new representation. The new polynomial-time operations permitted by sd-DNNF have applications to building linear-time, complete truth-maintenance and belief-revision systems, which we also explored in this paper.

Acknowledgement

I wish to thank Pierre Marquis for many valuable discussions on this subject and for his comments on an earlier version of this paper. This work has been partially supported by an NSF research grant number IIS-9988543 and a MURI grant number N00014-00-1-0617.

A Converting CNF to sd-DNNF

The pseudocode in Figure 8 uses a number of notations that we explain below.

An instantiation is a consistent conjunction of literals. An instantiation over atoms Σ includes one literal for each atom in Σ. Each node N in the decomposition tree has an associated cache CACHE_N, which maps instantiations into sd-DNNFs; N_l and N_r are the left and right children of node N, respectively; $\mathit{clause}(N)$ returns the clause attached to leaf node N; $\mathit{atoms}(N)$ are the atoms of clauses appearing under node N; $\mathit{atoms}(\alpha)$ returns the atoms appearing in instantiation α; $\mathit{project}(\alpha, \Sigma)$ returns the subset of instantiation α pertaining to atoms Σ; and $\text{CL2SDDNNF}(N, \alpha)$ returns an sd-DNNF of $\mathit{clause}(N) \mid \alpha$ mentioning all atoms in $\mathit{atoms}(\mathit{clause}(N))$.

For completeness, we include the definitions of width for a decomposition tree and a CNF.

For internal node N in a decomposition tree, $\mathit{atoms}^l(N)$ are atoms that appear in the left subtree of node N; $\mathit{atoms}^r(N)$ are atoms that appear in the right subtree of node N; $\mathit{atoms}^p(N)$ are atoms that appear in clauses attached to leaves not in the subtree rooted at node N.

Definition 10 *[6, 7] Let N be a node in a decomposition tree T. The <u>cluster</u> of node N is defined as follows:*

- *If N is a leaf node, then its cluster is $\mathit{atoms}(\mathit{clause}(N))$.*

- *If N is an internal node, then its cluster is $(atoms^l(N) \cap atoms^r(N)) \cup (atoms^p(N) \cap atoms(N))$.*

The width of a decomposition tree is the size of its maximal cluster minus one.

Definition 11 *[10] Let Δ be a propositional theory in clausal form. The interaction graph for Δ is the undirected graph G constructed as follows. The nodes of G are the atoms of Δ. There is an edge between two atoms in G iff the atoms appear in the same clause of Δ. The treewidth of Δ is the treewidth of its connectivity graph.*

The treewidth of an undirected graph is a standard notion of graph theory—see [2] for example.

B Proofs

Proof of Theorem 1

The proof is by induction on the structure of the sd-DNNF and its corresponding counting graph. We want to show that if N is a node in the counting graph which corresponds to subsentence α in the sd-DNNF, then the value of N represents $\mathit{Models}\#(\{\alpha\} \cup \mathbf{S}_\alpha)$, where $\mathbf{S}_\alpha$ is the subset of $\mathbf{S}$ which atoms appear in α. If α is the given sd-DNNF, then $\mathbf{S}_\alpha = \mathbf{S}$.

The base case follows immediately from Definition 6. For the inductive step, suppose that this holds for the conjuncts/disjuncts α_i of some conjunction/disjunction α. Let N, N_i be the counting graph nodes corresponding to α and α_i. We have two cases:

1. $\alpha = \bigwedge_i \alpha_i$: Note that $\mathbf{S}_\alpha$ is partitioned into $\mathbf{S}_{\alpha_i}$. Moreover, since the conjuncts α_i do not share atoms, we have that

$$\begin{aligned}
\mathit{Models}\#(\{\alpha\} \cup \mathbf{S}_\alpha) &= \mathit{Models}\#(\{\bigwedge_i \alpha_i\} \cup \mathbf{S}_\alpha) \\
&= \prod_i \mathit{Models}\#(\{\alpha_i\} \cup \mathbf{S}_{\alpha_i}) \\
&= \prod_i \textsc{val}(N_i) \\
&= \textsc{val}(N).
\end{aligned}$$

2. $\alpha = \bigvee_i \alpha_i$: Note that $\mathbf{S}_\alpha = \mathbf{S}_{\alpha_i}$. Moreover, since the disjuncts α_i are logically disjoint, we have that

$$\begin{aligned}
\mathit{Models}\#(\{\alpha\} \cup \mathbf{S}_\alpha) &= \mathit{Models}\#(\{\bigvee_i \alpha_i\} \cup \mathbf{S}_\alpha) \\
&= \sum_i \mathit{Models}\#(\{\alpha_i\} \cup \mathbf{S}_{\alpha_i})
\end{aligned}$$

$$= \sum_i \text{VAL}(N_i)$$
$$= \text{VAL}(N). \blacksquare$$

Proof of Theorem 2

First, note that by the decomposability of DNNFs:

- the function represented by the counting graph G is linear in each of its variables V_l, which correspond to a leaf node in G.
- the function will never multiply two expressions, one containing V_l and the other containing $V_{\neg l}$.

Hence, the partial derivative with respect to V_l does not depend on the value of either V_l or $V_{\neg l}$.

Second, we have:

$$\mathit{Models}\#(\Delta \cup \mathbf{S}) = \mathit{Models}\#(\Delta \cup \mathbf{S} \cup \{\neg l\}) + \mathit{Models}\#(\Delta \cup \mathbf{S} \cup \{l\}).$$

Suppose that $\neg l \notin \mathbf{S}$, and let $\mathbf{S}' = \mathbf{S} \cup \{\neg l\}$. The difference between the counting graph G evaluated at $\mathbf{S}$ and $\mathbf{S}'$ is that the leaf node V_l will have the value 1 under $\mathbf{S}$ and the value 0 under $\mathbf{S}'$. Therefore:

$$\mathit{Models}\#(\Delta \cup \mathbf{S} \cup \{\neg l\}) = \mathit{Models}\#(\Delta \cup \mathbf{S}) + (-1)\partial G(\mathbf{S})/\partial V_l.$$

Hence,

$$\partial G(\mathbf{S})/\partial V_l = \mathit{Models}\#(\Delta \cup \mathbf{S}) - \mathit{Models}\#(\Delta \cup \mathbf{S} \cup \{\neg l\}) = \mathit{Models}\#(\Delta \cup \mathbf{S} \cup \{l\}).$$

This proves the first part of the theorem.

The third part follows similarly: suppose $l \in \mathbf{S}$ and let $\mathbf{S}' = \mathbf{S} \setminus \{l\} \cup \{\neg l\}$. The difference between the counting graph G evaluated at $\mathbf{S}$ and $\mathbf{S}'$ is that

- the leaf node $V_{\neg l}$ will have the value 0 under $\mathbf{S}$ and the value 1 under $\mathbf{S}'$;
- the leaf node V_l will have the value 1 under $\mathbf{S}$ and the value 0 under $\mathbf{S}'$.

Hence,

$$\mathit{Models}\#(\Delta \cup \mathbf{S} \setminus \{l\} \cup \{\neg l\}) = \mathit{Models}\#(\Delta \cup \mathbf{S}) + (+1)\partial G(\mathbf{S})/\partial V_{\neg l} + (-1)\partial G(\mathbf{S})/\partial V_l.$$

We also have $\mathit{Models}\#(\Delta \cup \mathbf{S} \cup \{l\}) = \mathit{Models}\#(\Delta \cup \mathbf{S})$ since $l \in \mathbf{S}$, $\mathit{Models}\#(\Delta \cup \mathbf{S} \cup \{l\}) = \partial G(\mathbf{S})/\partial V_l$ by the first part and $\neg l \notin \mathbf{S}$, and, hence, $\mathit{Models}\#(\Delta \cup \mathbf{S}) = \partial G(\mathbf{S})/\partial V_l$. Hence,

$$\begin{aligned}
&\mathit{Models}\#(\Delta \cup \mathbf{S} \setminus \{l\} \cup \{\neg l\}) \\
&= \mathit{Models}\#(\Delta \cup \mathbf{S}) + (+1)\partial G(\mathbf{S})/\partial V_{\neg l} + (-1)\partial G(\mathbf{S})/\partial V_l \\
&= \mathit{Models}\#(\Delta \cup \mathbf{S}) + (+1)\partial G(\mathbf{S})/\partial V_{\neg l} + (-1)\mathit{Models}\#(\Delta \cup \mathbf{S}) \\
&= \partial G(\mathbf{S})/\partial V_{\neg l}.
\end{aligned}$$

We now prove the second part. Let $\mathbf{S}' = \mathbf{S} \setminus \{l\}$. Then

$$Models\#(\Delta \cup \mathbf{S}') = Models\#(\Delta \cup \mathbf{S}' \cup \{l\}) + Models\#(\Delta \cup \mathbf{S}' \cup \{\neg l\}).$$

Moreover, $Models\#(\Delta \cup \mathbf{S}' \cup \{l\}) = Models\#(\Delta \cup \mathbf{S} \cup \{l\})$, which equals $\partial G(\mathbf{S})/\partial V_l$ by the first part. Similarly, $Models\#(\Delta \cup \mathbf{S}' \cup \{\neg l\}) = Models\#(\Delta \cup \mathbf{S} \setminus \{l\} \cup \{\neg l\})$, which equals $\partial G(\mathbf{S})/\partial V_{\neg l}$ by the third part. Hence,

$$Models\#(\Delta \cup \mathbf{S} \setminus \{l\}) = \partial G(\mathbf{S})/\partial V_l + \partial G(\mathbf{S})/\partial V_{\neg l}. \blacksquare$$

Proof of Theorem 4

We have shown in [6, 7] that if Δ is a DNNF, then $(\Delta \mid \mathbf{S}) \wedge \mathbf{S}$ is also a DNNF. That $(\Delta \mid \mathbf{S}) \wedge \mathbf{S}$ is smooth follows immediately, since $atoms(\alpha \mid \mathbf{S}) = atoms(\alpha) - atoms(\mathbf{S})$. Hence, if $atoms(\alpha) = atoms(\beta)$, then $atoms(\alpha \mid \mathbf{S}) = atoms(\beta \mid \mathbf{S})$. That $(\Delta \mid \mathbf{S}) \wedge \mathbf{S}$ is deterministic follows from the following property of conditioning: the models of $\alpha \mid \mathbf{S}$ are the models of $\alpha \wedge \mathbf{S}$ after having removed from them $atoms(\mathbf{S})$.[8] Now suppose that $atoms(\alpha) = atoms(\beta)$, $\alpha \wedge \beta \models \mathsf{false}$, m_1 is a model of α, and m_2 is a model of β. Then m_1 and m_2 must be conflicting. Moreover, we must have two cases:

1. m_1 and m_2 are conflicting on an atom appearing in $\mathbf{S}$.
   Then either $m_1 \not\models \alpha \mid \mathbf{S}$ or $m_2 \not\models \beta \mid \mathbf{S}$.

2. m_1 and m_2 are conflicting on an atom not appearing in $\mathbf{S}$.
   Then models m_1 and m_2 will remain to be conflicting even after removing $atoms(\mathbf{S})$ from them.

Given the two cases, every model of $\alpha \mid \mathbf{S}$ will be conflicting with every model of $\beta \mid \mathbf{S}$ and, hence, $(\alpha \mid \mathbf{S}) \wedge (\beta \mid \mathbf{S}) \models \mathsf{false}$. $\blacksquare$

Proof of Theorem 5

This theorem follows immediately from the following property of sd-DNNF:

1. $\mathsf{models}(l) = \{\{l\}\}$ where l is a literal.

2. $\mathsf{models}(\alpha_1 \vee \ldots \vee \alpha_n) = \mathsf{models}(\alpha_1) \cup \ldots \cup \mathsf{models}(\alpha_n)$.

3. $\mathsf{models}(\alpha_1 \wedge \ldots \wedge \alpha_n) = \mathsf{models}(\alpha_1) \times \ldots \times \mathsf{models}(\alpha_n)$.

Specifically, since literal l has a single model $\{l\}$, the minimum $\mathbf{D}$-cardinality of l is simply the $\mathbf{D}$-cardinality of the model $\{l\}$. The models of a disjunction are simply the union of its disjuncts' models. Hence, its minimum $\mathbf{D}$-cardinality is the minimum $\mathbf{D}$-cardinality of any of its disjuncts. Finally, the models of a conjunction are simply the Cartesian product of its conjuncts' models, which share no variables. Therefore, the minimum $\mathbf{D}$-cardinality of a conjunction is the sum of the minimum $\mathbf{D}$-cardinalities of its conjuncts. $\blacksquare$

[8] To see this, represent α as a disjunction of its models, $m_1 \vee \ldots \vee m_n$, and then condition on $\mathbf{S}$. This conditioning will eliminate any models that are inconsistent with $\mathbf{S}$. For those models that are consistent with $\mathbf{S}$, each literal which appears in $\mathbf{S}$ will be replaced by true.

Proof of Theorem 6

This theorem follows immediately from the following property of sd-DNNF:

1. $\mathsf{models}(l) = \{\{l\}\}$ where l is a literal.

2. $\mathsf{models}(\alpha_1 \vee \ldots \vee \alpha_n) = \mathsf{models}(\alpha_1) \cup \ldots \cup \mathsf{models}(\alpha_n)$.

3. $\mathsf{models}(\alpha_1 \wedge \ldots \wedge \alpha_n) = \mathsf{models}(\alpha_1) \times \ldots \times \mathsf{models}(\alpha_n)$.

Specifically, since a literal l has only a single model $\{l\}$, we have $\mathsf{models}(\mathsf{minimize}(l, \mathbf{D})) = \{\{l\}\}$ according to Definition 8 and Theorem 6. The remaining two cases follow directly from smoothness and decomposability. ■

Proof of Theorem 7

The only difference between this version and the one in [6, 7] is that we have $\textsc{cl2sddnnf}(N, \alpha)$ instead of $\mathit{clause}(N) \mid \alpha$ on Line 04, therefore, converting a clause to an sd-DNNF at the boundary condition. The complexity and correctness (with respect to DNNF) is therefore unchanged. The only thing we need to show is that the resulting NNF is also smooth and deterministic.

Determinism follows because the only place where a disjunction is introduced by the algorithm is on Line 04 and 05. For disjunctions introduced on Line 04, the disjuncts are logically inconsistent by definition of $\textsc{cl2sddnnf}(N, \alpha)$. For disjunctions introduced on Line 05, the disjuncts are logically inconsistent since the different β's are logically inconsistent.

Smoothness follows by induction on the structure of a decomposition tree. First, for disjunctions introduced on Line 04, the disjuncts mention the same set of atoms by definition of $\textsc{cl2sddnnf}(N, \alpha)$. Second, for disjunctions introduced on Line 05, it suffice to show that $\mathit{atoms}(\textsc{cnf2sddnnf}(N, \alpha)) = \mathit{atoms}(\textsc{cnf2sddnnf}(N, \alpha'))$ when $\mathit{atoms}(\alpha) = \mathit{atoms}(\alpha')$, which we will show by induction. When N is leaf, this follows from the definition of $\textsc{cl2sddnnf}(N, \alpha)$. Suppose that N is not leaf. Then $\textsc{cnf2sddnnf}(N, \alpha) =$

$$\bigvee_{\beta} \textsc{cnf2sddnnf}(N_l, \alpha \wedge \beta) \wedge \textsc{cnf2sddnnf}(N_r, \alpha \wedge \beta) \wedge \beta,$$

and $\textsc{cnf2sddnnf}(N, \alpha') =$

$$\bigvee_{\beta} \textsc{cnf2sddnnf}(N_l, \alpha' \wedge \beta) \wedge \textsc{cnf2sddnnf}(N_r, \alpha' \wedge \beta) \wedge \beta.$$

Hence, $\mathit{atoms}(\textsc{cnf2sddnnf}(N, \alpha)) = \mathit{atoms}(\textsc{cnf2sddnnf}(N, \alpha'))$ follows from the induction hypothesis. ■

Proof of Theorem 8

Consider the following two propositional theories from [3], which are defined over a set of variables X_{ij} where $1 \leq i, j \leq n$:

- Δ_1: m is a model of Δ_1 iff
  - $m(X_{ij}) = \mathsf{true}$ for *some* i and *all* j;
  - m sets an *odd* number of X_{ij} to true.
- Δ_2: m is a model of Δ_2 iff
  - $m(X_{ij}) = \mathsf{true}$ for *some* j and *all* i;
  - m sets an *even* number of X_{ij} to true.

It is shown in [3] that each of these theories have a polynomially-sized FBDD representation, yet their disjunction cannot be represented polynomially using FBDDs. Note also that, by definition, Δ_1 and Δ_2 are logically inconsistent. ■

References

[1] Jon Barwise, editor. *Handbook of Mathematical Logic.* North-Holland, Amsterdam, 1977.

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

[3] B. Bollig and I. Wegner. Complexity theoretical results on partitioned binary decision diagrams. *Theory of Computing Systems*, 32:487–503, 1999.

[4] R. E. Bryant. Symbolic Boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[5] Adnan Darwiche. Model–based diagnosis using structured system descriptions. *Journal of Artificial Intelligence Research*, 8:165–222, 1998.

[6] Adnan Darwiche. Compiling knowledge into decomposable negation normal form. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 284–289, 1999.

[7] Adnan Darwiche. Decomposable negation normal form. Technical Report D–109, Cognitive Systems Laboratory, Computer Science Department, UCLA, Los Angeles, Ca 90095, 1999. To appear in Journal of ACM.

[8] Adnan Darwiche. A differential approach to inference in Bayesian networks. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 123–132, 2000.

[9] Johan de Kleer, Alan K. Mackworth, and Raymond Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2-3):197–222, 1992.

[10] Rina Dechter and Irina Rish. Directional resolution: The Davis-Putnam procedure, revisited. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 134–145, 1994.

[11] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates and counterfactuals. *Artificial Intelligence*, 57:227–270, 1992.

[12] Kenneth D. Forbus and Johan de Kleer. *Building Problem Solvers*. MIT Press, 1993.

[13] J. Gergov and C. Meinel. Efficient analysis and manipulation of OBDDs can be extended to FBDDs. *IEEE Transactions on Computers*, 43(10):1197–1209, 1994.

[14] Masao. Simultaneous computation of functions, partial derivatives and estimates of rounding error. *Japan J. Appl. Math.*, 1:223–252, 1984.

[15] B. Nebel. How hard is it to revise a belief base? In D. Dubois and H. Prade, editors, *Handbook of Defeasible Reasoning and Uncertainty Managment*, pages 77–145. Kluwer Academic, 1998.

[16] Graz Rote. Path problems in graphs. *Computing Suppl.*, 7:155–189, 1990.

[17] D. Sieling and I. Wegener. Graph driven BDDs — a new data structure for Boolean functions. *Theoretical Computer Science*, 141:283–310, 1995.