Solving $\mathsf{PP}^{\mathsf{PP}}$-Complete Problems Using Knowledge Compilation

Umut Oztok and **Arthur Choi** and **Adnan Darwiche**
Computer Science Department
University of California, Los Angeles
{umut,aychoi,darwiche}@cs.ucla.edu

Abstract

Knowledge compilation has been successfully used to solve beyond NP problems, including some PP-complete and $\mathsf{NP}^{\mathsf{PP}}$-complete problems for Bayesian networks. In this work we show how knowledge compilation can be used to solve problems in the more intractable complexity class $\mathsf{PP}^{\mathsf{PP}}$. This class contains $\mathsf{NP}^{\mathsf{PP}}$ and includes interesting AI problems, such as non-myopic value of information. We show how to solve the prototypical $\mathsf{PP}^{\mathsf{PP}}$-complete problem MAJMAJSAT in linear-time once the problem instance is compiled into a special class of *Sentential Decision Diagrams*. To show the practical value of our approach, we adapt it to answer the *Same-Decision Probability* (SDP) query, which was recently introduced for Bayesian networks. The SDP problem is also $\mathsf{PP}^{\mathsf{PP}}$-complete. It is a value-of-information query that quantifies the robustness of threshold-based decisions and comes with a corresponding algorithm that was also recently proposed. We present favorable experimental results, comparing our new algorithm based on knowledge compilation with the state-of-the-art algorithm for computing the SDP.

1 Introduction

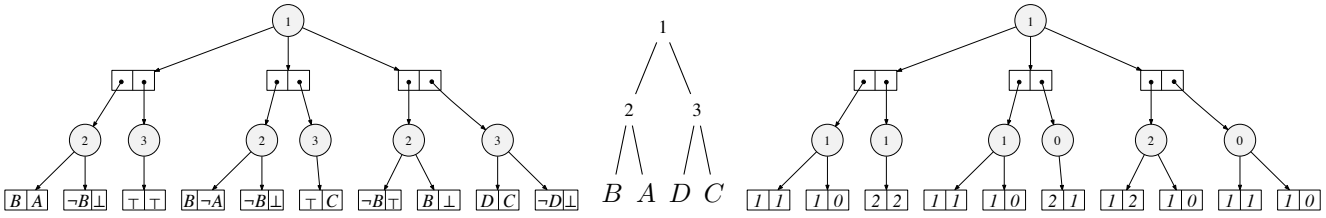
The complexity class $\mathsf{PP}^{\mathsf{PP}}$ is highly intractable being at the second level of the counting hierarchy (Wagner 1986). Despite this difficulty, the $\mathsf{PP}^{\mathsf{PP}}$ class includes some interesting and practical AI problems, such as non-myopic value of information (Krause and Guestrin 2009). Developing effective methods for problems in this class is therefore both significant and difficult.

Our proposed approach for tackling problems in the $\mathsf{PP}^{\mathsf{PP}}$ class will be based on knowledge compilation, which is a well-established research area in AI; see for example (Marquis 1995; Selman and Kautz 1996; Cadoli and Donini 1997; Darwiche and Marquis 2002; Darwiche 2014). The key notion here is to *compile* problem instances into *tractable* representations, allowing one to solve such problems efficiently if the compilation is successful. Although knowledge compilation was originally motivated by the need to push much of the computational overhead into an offline compilation phase, it has been increasingly used as a general methodology for computation. In particular, this approach has been successfully used to solve beyond NP problems, including some problems that are complete for the PP class (Darwiche 2001a; 2003), and the $\mathsf{NP}^{\mathsf{PP}}$ class (Huang, Chavira, and Darwiche 2006; Pipatsrisawat and Darwiche 2009).

In this work, we extend the reach of knowledge compilation techniques to problems in the highly intractable complexity class $\mathsf{PP}^{\mathsf{PP}}$, which contains $\mathsf{NP}^{\mathsf{PP}}$ and can be thought of as its counting analogue. In particular, we introduce a new algorithm for the prototypical $\mathsf{PP}^{\mathsf{PP}}$-complete problem known as MAJMAJSAT. This decision problem is posed with respect to a CNF, asking whether there exists a majority of truth assignments to some variables, under which there is a majority of satisfying truth assignments to the remaining variables (Wagner 1986). Our algorithm is based on compiling the problem instance into a special class of *Sentential Decision Diagrams* (SDDs), which have been recently introduced as a tractable representation of Boolean functions (Darwiche 2011). This new class of SDDs we identify in this paper constrains their structure, allowing one to solve MAJMAJSAT in time linear in the SDD size.

To show the effectiveness and applicability of our approach, we modify our algorithm to solve a $\mathsf{PP}^{\mathsf{PP}}$-complete problem that is of practical interest: The *Same-Decision Probability* (SDP) introduced recently for Bayesian networks (Choi, Xue, and Darwiche 2012). The SDP problem is a value-of-information query that quantifies the robustness of threshold-based decisions. It has been successfully applied as a selection or stopping criterion when making decisions under uncertainty (Chen, Choi, and Darwiche 2014; 2015b; 2015a), and in medical diagnosis (Gimenez et al. 2014). Further, it comes with a corresponding exact algorithm (Chen, Choi, and Darwiche 2013). We empirically compare our proposed approach with the state-of-the-art algorithm for computing the SDP, showing favorable results.

We organize the paper as follows. Section 2 provides some technical background. Section 3 reviews some complexity classes beyond NP. Section 4 introduces the special class of SDDs and the new MAJMAJSAT algorithm that operates on these SDDs. Section 5 introduces the SDP problem and provides a corresponding algorithm. Sections 6 and 7 discuss the compilation of SDDs, and Section 8 presents empirical results. Section 9 provides a broader perspective on this work. We conclude the paper after discussing related work in Section 10. The appendix contains the proofs.



(a) A compressed and normalized SDD (b) A vtree (c) Value registers computed by Algorithm 1

Figure 1: An SDD and a vtree for $(A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$.

2 Technical Background

Upper case letters (e.g., X) will denote variables and lower case letters (e.g., x) will denote their instantiations. That is, x is a *literal* denoting X or $\neg X$. Bold upper case letters (e.g., $\mathbf{X}$) will denote sets of variables and bold lower case letters (e.g., $\mathbf{x}$) will denote their instantiations. We liberally treat an instantiation of a variable set as conjunction of its corresponding literals. Given instantiations $\mathbf{x}$ and $\mathbf{y}$, we say $\mathbf{x}$ is *compatible* with $\mathbf{y}$, denoted $\mathbf{x} \sim \mathbf{y}$, iff $\mathbf{x} \wedge \mathbf{y}$ is satisfiable.

A *Boolean function* $f(\mathbf{Z})$ maps each instantiation $\mathbf{z}$ of variables $\mathbf{Z}$ to 1/true or 0/false. A *trivial* Boolean function maps all its inputs to true (denoted $\top$) or maps them all to false (denoted $\bot$). An instantiation $\mathbf{z}$ *satisfies* function f, denoted $\mathbf{z} \models f$, iff f maps $\mathbf{z}$ to true. In this case, $\mathbf{z}$ is said to be a *model* of function f. The *model count* of function f, denoted $\text{MC}(f)$, is the number of models of f. The *conditioning* of function f on instantiation $\mathbf{x}$, denoted $f|\mathbf{x}$, is the sub-function obtained by setting variables $\mathbf{X}$ to their values in $\mathbf{x}$. We will combine Boolean functions using the traditional Boolean operators, such as $\wedge$ and $\vee$.

SDDs: A Boolean function $f(\mathbf{XY})$, where $\mathbf{X}$ and $\mathbf{Y}$ are disjoint, can always be decomposed into the following form:

$$f(\mathbf{XY}) = \big(p_1(\mathbf{X}) \wedge s_1(\mathbf{Y})\big) \vee \ldots \vee \big(p_n(\mathbf{X}) \wedge s_n(\mathbf{Y})\big)$$

such that the sub-functions $p_i(\mathbf{X})$ are consistent, mutually exclusive, and exhaustive.[1] A decomposition of this kind is called an $(\mathbf{X}, \mathbf{Y})$-*partition*, and denoted $\{(p_1, s_1), \ldots, (p_n, s_n)\}$. In this case, each p_i is called a *prime,* each s_i is called a *sub*. Moreover, an $(\mathbf{X}, \mathbf{Y})$-partition is *compressed* when its subs are distinct, i.e., $s_i \neq s_j$ for $i \neq j$ (Darwiche 2011). For instance, consider the Boolean function $f = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$ represented as a DNF. Choosing $\mathbf{X} = \{A, B\}$ and $\mathbf{Y} = \{C, D\}$ yields the following $(\mathbf{X}, \mathbf{Y})$-partition:

$$\{(\underbrace{A \wedge B}_{\text{prime}}, \underbrace{\top}_{\text{sub}}), (\underbrace{\neg A \wedge B}_{\text{prime}}, \underbrace{C}_{\text{sub}}), (\underbrace{\neg B}_{\text{prime}}, \underbrace{D \wedge C}_{\text{sub}})\}.$$

SDDs will result from the recursive decomposition of a Boolean function using $(\mathbf{X}, \mathbf{Y})$-partitions. This recursive decomposition process requires a structure to determine the $\mathbf{X}/\mathbf{Y}$ variables of each partition. For that, we use a *vtree*, which is a full binary tree whose leaves are labeled with variables; see Figure 1(b). For an internal vtree node v, we will use v^l and v^r to denote the left and right children of v. We will denote variables appearing inside v by $vars(v)$. We will also not distinguish between vtree node v and the subtree rooted at v. We next define SDDs (Darwiche 2011).

Definition 1. *α is an SDD that is normalized for vtree v iff:*

- *$\alpha = \bot$ or $\alpha = \top$, where v is a leaf node;*
- *$\alpha = X$ or $\alpha = \neg X$, where v is a leaf node labeled by X;*
- *$\alpha = \{(p_1, s_1), \ldots, (p_n, s_n)\}$, where v is an internal node, $p_1, \ldots, p_n$ are SDDs that are normalized for v^l, and $s_1, \ldots, s_n$ are SDDs that are normalized for v^r.*

A constant or literal SDD is called a *terminal*; otherwise it is called a *decomposition*. SDDs are depicted graphically as in Figure 1(a). Here, each decomposition is represented by a circle, where each element is depicted by a paired box $\boxed{p \mid s}$. The left box corresponds to a prime p and the right box corresponds to its sub s. A prime p or sub s are either a constant, literal, or pointer to a decomposition node. Decomposition nodes are labeled by the vtree nodes they are normalized for.

An SDD is *compressed* iff each of its decompositions is a compressed partition. For a fixed vtree, a Boolean function has a unique SDD that is compressed and normalized. If an SDD is normalized for vtree v, then its model count is defined over $vars(v)$. For example, the terminal SDDs $\bot$, $\top$, X and $\neg X$ have model counts of 0, 2, 1, and 1, respectively. We will not distinguish between an SDD node α and the Boolean function that α represents. We will also identify an SDD by its root node.

3 Complexity Classes Beyond NP

In this section, we will review some complexity classes beyond NP that are relevant to our work. In particular, our focus will be on the complexity classes NP, PP, NP^{PP}, and PP^{PP}, which are related in the following way:

$$\text{NP} \subseteq \text{PP} \subseteq \text{NP}^{\text{PP}} \subseteq \text{PP}^{\text{PP}}.$$

NP is the class of decision problems that can be solved by a non-deterministic polynomial-time Turing machine. PP is the class of decision problems that can be solved by a non-deterministic polynomial-time Turing machine, which has more accepting than rejecting paths. NP^{PP} and PP^{PP} are the corresponding classes assuming a PP oracle. That is, the corresponding Turing machine has an access to a PP oracle.

[1] $p_i \neq \bot$ for all i; $p_i \wedge p_j = \bot$ for $i \neq j$; and $\bigvee_i p_i = \top$.

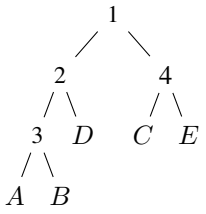


Figure 2: An $\mathbf{X}$-constrained vtree, where $\mathbf{X} = \{A, B, D\}$.

Given a CNF Δ representing Boolean function $f(\mathbf{XY})$, we have the following decision problems which are respectively complete for the above complexity classes.

SAT is the prototypical NP-complete problem, asking if there is an instantiation $\mathbf{xy}$ that satisfies CNF Δ.

MAJSAT is the prototypical PP-complete problem, asking if there are a majority of instantiations $\mathbf{xy}$ that satisfy CNF Δ.

E-MAJSAT is the prototypical $\mathsf{NP}^{\mathsf{PP}}$-complete problem, asking if there is an instantiation $\mathbf{x}$ for which a majority of instantiations $\mathbf{y}$ satisfy CNF $\Delta|\mathbf{x}$.[2]

MAJMAJSAT is the prototypical $\mathsf{PP}^{\mathsf{PP}}$-complete problem, asking if there are a majority of instantiations $\mathbf{x}$ for which a majority of instantiations $\mathbf{y}$ satisfy CNF $\Delta|\mathbf{x}$.

In the remainder of the paper, we will focus on a *functional* variant of MAJMAJSAT, denoted $\text{MMS}(f, \mathbf{X}, T)$, which we introduce as follows:[3]

> Given a threshold T, how many instantiations $\mathbf{x}$ are there, for which the number of instantiations $\mathbf{y}$ that satisfy $f|\mathbf{x}$ is greater than or equal to T?

Formally, $\text{MMS}(f, \mathbf{X}, T)$ can be defined as follows:

$$\text{MMS}(f, \mathbf{X}, T) = \sum_{\mathbf{x}} [\text{MC}(f|\mathbf{x}) \geq T], \qquad (1)$$

where $[P]$ is an indicator function equal to 1 if P is true and equal to 0 otherwise.

We will next show how to solve MAJMAJSAT in linear-time when the underlying Boolean function is represented as a special type of SDD.

4 Solving MAJMAJSAT using SDDs

In this section, we will present a new algorithm that solves the prototypical $\mathsf{PP}^{\mathsf{PP}}$-complete problem MAJMAJSAT, using a special type of SDDs which we will introduce next.

Constrained SDDs

The new class of SDDs will basically have a constrained structure, due to the following special type of vtrees.

[2]The CNF $\Delta|\mathbf{x}$ is obtained by replacing variables $\mathbf{X}$ in CNF Δ with their values in instantiation $\mathbf{x}$.

[3]Solving the functional version would immediately imply a solution to the decision problem.

Algorithm 1: $MMS(S, \mathbf{X}, T)$

Input:
$\mathbf{X}$: set of variables
T : threshold
S : (root) $\mathbf{X}$-constrained SDD
Data:
$vr()$: value registers (one for each SDD node)
Output: computes $\text{MMS}(S, \mathbf{X}, T)$

1 **foreach** *SDD node α in S (children before parents)* **do**
2 **if** *α is a terminal* **then** $vr(\alpha) \leftarrow \text{MC}(\alpha)$
3 **else** $vr(\alpha) \leftarrow \sum_{(p_i, s_i) \in \alpha} vr(p_i) \times vr(s_i)$
4 **if** *α is $\mathbf{X}$-constrained* **then**
5 $vr(\alpha) \leftarrow 1$ if $vr(\alpha) \geq T$; else 0
6 **return** $vr(S)$

Definition 2. *A vtree node v is $\mathbf{X}$-constrained, denoted $v_{\mathbf{X}}$, iff v appears on the right-most path of the vtree and $\mathbf{X}$ is the set of variables outside v. A vtree is $\mathbf{X}$-constrained iff it has an $\mathbf{X}$-constrained node.*

Figure 2 shows an $\mathbf{X}$-constrained vtree for $\mathbf{X} = \{A, B, D\}$, where vtree node $v = 4$ is the $\mathbf{X}$-constrained node. A vtree can have at most one $\mathbf{X}$-constrained node. Moreover, an $\mathbf{X}$-constrained vtree constrains the variable order obtained by a left-right traversal of the vtree, pushing variables $\mathbf{X}$ in front. The reverse is not necessarily true. For example, the left-right traversal of the vtree in Figure 2 puts variables $\mathbf{X} = \{A, B\}$ in front, yet the vtree is not $\mathbf{X}$-constrained.

To compute $\text{MMS}(f, \mathbf{X}, T)$, we will represent the Boolean function f using an $\mathbf{X}$-constrained SDD.

Definition 3. *An SDD is $\mathbf{X}$-constrained iff it is normalized for an $\mathbf{X}$-constrained vtree. An SDD node is $\mathbf{X}$-constrained iff it is normalized for the $\mathbf{X}$-constrained vtree node.*

Intuitively, an $\mathbf{X}$-constrained SDD node corresponds to the conditioning of the SDD f on some instantiation $\mathbf{x}$, and will be used to compute the indicator function $[\text{MC}(f|\mathbf{x}) \geq T]$.

A New Algorithm to Solve MAJMAJSAT

We present the pseudocode of our approach in Algorithm 1, which takes as input a set of variables $\mathbf{X}$, a threshold value T, and an $\mathbf{X}$-constrained SDD S, to compute $\text{MMS}(S, \mathbf{X}, T)$. The algorithm performs a single bottom-up pass over S (Lines 1–5). For each visited SDD node α, the algorithm applies one of several actions, depending on the vtree node v to which α is normalized for.

If vtree node v is not an ancestor of the $\mathbf{X}$-constrained vtree node $v_{\mathbf{X}}$, the algorithm computes the model count of α.

Lemma 1. *Let α be an SDD node normalized for v, where v is not an ancestor of $v_{\mathbf{X}}$. Line 2 or Line 3 computes MC(α).*

If $v = v_{\mathbf{X}}$, α must be equal to $S|\mathbf{x}$ for some instantiation $\mathbf{x}$. So, we also compute the indicator function $[\text{MC}(\alpha) \geq T]$ (Lines 4–5), and pass either 0 or 1 to the ancestors of α. Because of this, if v is an ancestor of $v_{\mathbf{X}}$, the algorithm basically counts the instantiations $\mathbf{y}$ for which the model count

of $\alpha|\mathbf{y}$ is above the threshold T, where $\mathbf{Y}$ is the subset of $\mathbf{X}$ appearing in v.

Lemma 2. *Let α be an SDD node normalized for v, where v is an ancestor of $v_{\mathbf{X}}$ or $v = v_{\mathbf{X}}$. Then,*

$$vr(\alpha) = \sum_{\mathbf{y}} \big[MC(\alpha|\mathbf{y}) \geq T\big],$$

where $\mathbf{Y} = vars(v) \cap \mathbf{X}$.

The above cases ensure that the algorithm computes Equation (1) at the root of SDD S (Line 6).

Proposition 1. *Algorithm 1 computes $MMS(S, \mathbf{X}, T)$.*

As the algorithm performs a single pass over S (Lines 1–5), and at each node it takes a constant amount of time, the time complexity of Algorithm 1 is linear in the size of S.

Proposition 2. *Algorithm 1 takes time linear in the size of S.*

We now demonstrate how Algorithm 1 works on constrained SDDs. Consider the SDD S in Figure 1(a). This SDD is normalized for the vtree in Figure 1(b). This vtree is $\mathbf{X}$-constrained for $\mathbf{X} = \{A, B\}$, where the root vtree node is $\mathbf{X}$-constrained. Given a threshold $T = 3$, Figure 1(c) shows the value registers of SDD nodes computed by Algorithm 1, upon the call $MMS(S, \mathbf{X}, T)$, by labeling the nodes with the corresponding values. Accordingly, the root of SDD S returns 1, meaning that there exists only one instantiation $\mathbf{x}$ of variables $\mathbf{X}$ such that the model count of $S|\mathbf{x}$ exceeds the given threshold $T = 3$.

We close this section with the following remark. When the Boolean function is represented as a general SDD (serving the role of a PP oracle), MAJMAJSAT would be PP-hard. This is still true when we use other well-known compilation languages that support polynomial-time model counting, such as d-DNNFs (Darwiche 2001a) and OBDDs (Bryant 1986). To solve the problem in linear-time, however, we need an additional property, as in $\mathbf{X}$-constrained vtrees. We are not aware if there is a weaker property that would suffice.

5 Computing the SDP using SDDs

We will now modify our algorithm to solve another $\mathsf{PP}^{\mathsf{PP}}$-complete problem, which is of practical interest: The *Same-Decision Probability* (SDP) for Bayesian networks (Choi, Xue, and Darwiche 2012). The input to this problem is a probability distribution represented by the Bayesian network, together with some variable sets and a threshold. Intuitively, the SDP is used to quantify the robustness of decisions against new evidence. That is, given initial evidence $\mathbf{e}$, one makes a decision d based on whether $\mathsf{Pr}(d|\mathbf{e})$ surpasses a given threshold T. The SDP is then the probability that this decision would stay the same after observing the state of new evidence (which is also given as input). We will actually define an abstraction of this problem in which the distribution is represented by a *weighted Boolean function.* This abstraction will facilitate the computation of SDP through compilation into SDDs.

SDP on Weighted Boolean Functions

We start by defining weighted Boolean functions, which simply augment a Boolean function with a weight function.

Definition 4. *A weighted Boolean function is a pair $\big(f, W\big)$ where f is a Boolean function over variables $\mathbf{Z}$ and W is a weight function that maps literals of $\mathbf{Z}$ to real numbers.*

Given a weighted Boolean function (f, W), one is typically interested in computing its *weighted* model count, which is formally defined as follows:

$$\sum_{\mathbf{z} \models f} \Big(\prod_{\ell \in \mathbf{z}} W(\ell) \Big).$$

That is, the weighted model count of f is the summation of the weights of the models of f, where the weight of a model is the product of its literals' weights. The weighted model count subsumes the model count when the weight function W assigns the weight 1 to each literal.

We also define the weighted model count with respect to an instantiation $\mathbf{x}$, also called *evidence* $\mathbf{x}$:

$$\phi_{(f,W)}(\mathbf{x}) = \sum_{\mathbf{z} \models f,\, \mathbf{z} \sim \mathbf{x}} \Big(\prod_{\ell \in \mathbf{z}} W(\ell) \Big).$$

That is, the weighted model count of f under evidence $\mathbf{x}$ is the summation of the weights of the models of f compatible with $\mathbf{x}$. We will omit W from the subscript whenever it is clear from the context, and write $\phi_f(\mathbf{x})$ instead. Under no evidence (i.e., $\mathbf{x} \equiv \top$), we often drop $\mathbf{x}$ and write ϕ_f. In this case, ϕ_f reduces to the weighted model count.

We finally define the *conditional* weighted model count:

$$\phi_f(\mathbf{x} \mid \mathbf{y}) = \frac{\phi_f(\mathbf{x}\,\mathbf{y})}{\phi_f(\mathbf{y})}.$$

Note here that $\phi_f(. \mid \top)$ is a probability distribution. Moreover, $\phi_f(. \mid \mathbf{y})$ is a probability distribution conditioned on instantiation $\mathbf{y}$. This shows how a weighted Boolean function can be used to represent a probability distribution, allowing us to define the same-decision probability on weighted Boolean functions.

Definition 5. *Consider a weighted Boolean function $\big(f, W\big)$. Let $\mathbf{E}, \mathbf{H}$ and $\{D\}$ be mutually disjoint variables of f. Given an instantiation d, a threshold T, and an instantiation $\mathbf{e}$, the same-decision probability is defined as follows:*

$$SDP_f(d, \mathbf{H}, \mathbf{e}, T) = \sum_{\mathbf{h}} \big[\phi_f(d \mid \mathbf{h}\,\mathbf{e}) \geq T\big]\, \phi_f(\mathbf{h} \mid \mathbf{e}). \tag{2}$$

Here, $[P]$ is the indicator function which is 1 if P is true, and 0 otherwise.

We remark that the classical definition of SDP on Bayesian networks is based on the probability distribution defined by the Bayesian network. Equation (2), however, replaces that distribution by its weighted Boolean function representation. Similarly to its Bayesian network analogue, the SDP on weighted Boolean functions is highly intractable, assuming a CNF representation of the function.

Theorem 1. *The problem of deciding whether the SDP on a weighted CNF is greater than a number p is $\mathsf{PP}^{\mathsf{PP}}$-complete.*

Algorithm 2: $SDP(d, \mathbf{H}, \mathbf{e}, T, S)$

Input:
d : hypothesis
$\mathbf{H}$: query variables
$\mathbf{e}$: evidence
T : threshold
S : $\mathbf{H}$-constrained SDD
Data:
$vr_1(), vr_2()$: value registers (one for each SDD node)
Output: computes $\text{SDP}_S(d, \mathbf{H}, \mathbf{e}, T)$

1 **foreach** *SDD node* α *in* S *(children before parents)* **do**
2 **if** α *is a terminal* **then**
3 $vr_1(\alpha) \leftarrow \phi_\alpha$ if $\alpha \sim \mathbf{e}$; else 0
4 **else** $vr_1(\alpha) \leftarrow \sum_{(p_i,s_i)\in\alpha} vr_1(p_i) \times vr_1(s_i)$
5 **foreach** *SDD node* α *in* S *(children before parents)* **do**
6 **if** α *is a terminal* **then**
7 $vr_2(\alpha) \leftarrow \phi_\alpha$ if $\alpha \sim d\,\mathbf{e}$; else 0
8 **else** $vr_2(\alpha) \leftarrow \sum_{(p_i,s_i)\in\alpha} vr_2(p_i) \times vr_2(s_i)$
9 **if** α *is* $\mathbf{H}$*-constrained* **then**
10 $vr_2(\alpha) \leftarrow vr_1(\alpha)$ if $\frac{vr_2(\alpha)}{vr_1(\alpha)} \geq T$; else 0
11 $\phi(\mathbf{e}) \leftarrow vr_1(S)$
12 $Q \leftarrow vr_2(S)$
13 **return** $\frac{Q}{\phi(\mathbf{e})}$

As we shall see next, if a Boolean function is represented by an $\mathbf{H}$-constrained SDD, the SDP problem becomes tractable. Indeed, as long as the set of variables $\mathbf{H}$ does not change, each instance of SDP with different parameters $d, \mathbf{e}$, and T can be solved in linear-time in the SDD size. That is, our knowledge compilation approach would effectively solve exponentially many queries in linear-time.

A New Algorithm to Compute SDPs

We now present our method to compute the SDP on weighted Boolean functions, using constrained SDDs. The pseudocode of our approach is described in Algorithm 2, which takes as input an SDP instance with parameters $d, \mathbf{H}, \mathbf{e}, T$ and an $\mathbf{H}$-constrained SDD S, to compute $\text{SDP}_S(d, \mathbf{H}, \mathbf{e}, T)$. The algorithm is a slight modification of Algorithm 1 presented earlier. It performs two bottom-up passes over S and maintains two value registers per SDD node. This is because it needs to compute (conditional) weighted model counts under evidence.

Given an SDD node α normalized for vtree node v, and evidence $\mathbf{e}$, we let $\mathbf{e}_v$ denote the subset of the instantiation $\mathbf{e}$ that pertains to the variables of vtree v. Algorithm 2 then computes $\phi_\alpha(\mathbf{e}_v)$ for each node α in S, during the first pass (Lines 1–4). The result is cached in the register $vr_1(\alpha)$.

Lemma 3. *Let α be an SDD node normalized for vtree v. Then, $vr_1(\alpha) = \phi_\alpha(\mathbf{e}_v)$.*

In the second pass (Lines 5–10), the algorithm mimics Algorithm 1. First, if v is not an ancestor of $v_\mathbf{H}$, it simply computes a weighted model count.

Lemma 4. *Let α be an SDD node normalized for vtree v, where v is not an ancestor of $v_\mathbf{H}$. Lines 7–8 compute $\phi_\alpha(d\,\mathbf{e}_v)$ if D is contained in vtree v, and $\phi_\alpha(\mathbf{e}_v)$ otherwise.*

Next, if $v = v_\mathbf{H}$, then α must be equal to $S|\mathbf{h}$ for some instantiation $\mathbf{h}$. Here, we also compute the indicator function $\big[\phi_\alpha(d \mid \mathbf{e}) \geq T\big]$ (Lines 9–10) and pass either 0 or $\phi_\alpha(\mathbf{e})$ to the ancestors of α. This way, if v is an ancestor of $v_\mathbf{H}$, it basically computes the following quantity:

$$\sum_{\mathbf{y}} \big[\phi_\alpha(d \mid \mathbf{y}, \mathbf{e}) \geq T\big]\, \phi_\alpha(\mathbf{y}\,\mathbf{e}),$$

where $\mathbf{Y}$ is the subset of $\mathbf{H}$ appearing in v.

Lemma 5. *Let α be an SDD node normalized for vtree v, where v is an ancestor of $v_\mathbf{H}$ or $v = v_\mathbf{H}$. Then,*

$$vr_2(\alpha) = \sum_{\mathbf{y}} \big[\phi_\alpha(d \mid \mathbf{y}\,\mathbf{e}) \geq T\big] \phi_\alpha(\mathbf{y}\,\mathbf{e}),$$

where $\mathbf{Y} = vars(v) \cap \mathbf{H}$.

We now obtain the following quantity at the root of SDD S:

$$Q = \sum_{\mathbf{h}} \big[\phi_S(d \mid \mathbf{h}\,\mathbf{e}) \geq T\big]\, \phi_S(\mathbf{h}\,\mathbf{e}).$$

The quantity Q is not equal to the SDP (note $\phi_S(\mathbf{h}, \mathbf{e})$ instead of $\phi_S(\mathbf{h} \mid \mathbf{e})$). Dividing Q by $\phi_S(\mathbf{e})$, which is computed by the first pass, gives the desired result (Line 13).

Proposition 3. *Algorithm 2 computes $SDP_S(d, \mathbf{H}, \mathbf{e}, T)$.*

The algorithm takes two passes over SDD S (Lines 1–4 and Lines 5–10). During each pass, the work it performs at each SDD node takes a constant amount of time. Hence, the time complexity of Algorithm 2 is linear in the size of S.

Proposition 4. *Algorithm 2 takes time linear in the size of S.*

6 Compiling X-Constrained SDDs

Our algorithms require the representation of Boolean functions as constrained SDDs, which also require the construction of constrained vtrees. Since the SDD size depends critically on the corresponding vtree, identifying good constrained vtrees is quite critical for compiling successfully. Moreover, once compilation is completed, further queries can be answered in time linear in the SDD size. Hence, the smaller the compiled SDDs, the more efficient further inference will be. We will next describe a method for obtaining $\mathbf{X}$-constrained vtrees that tend to yield smaller SDD sizes.

There are two different methods for generating vtrees. The first method, which is *static*, identifies an appropriate vtree before the compilation starts. This method requires a preprocessing step of the Boolean function representation. The second method, which is *dynamic*, searches for an appropriate vtree during the compilation process. That is, while compiling the SDD, one can use a search algorithm that tries to identify vtrees leading to smaller SDD sizes. Although this search might be costly, it is needed most of the time to successfully finish compilation.

To get $\mathbf{X}$-constrained vtrees, we will combine static and dynamic approaches as follows. Our initial vtree will be

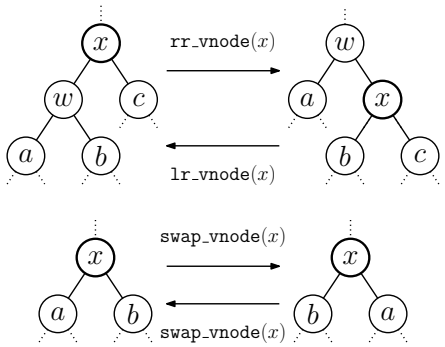


Figure 3: Rotate a vtree node x right and left, and swap its children. Nodes a, b, and c may represent leaves or subtrees.

right-linear, where variables $\mathbf{X}$ appear first in the left-right traversal of the vtree.[4] Here, one can use some heuristic method to find a variable order for variables $\mathbf{X}$ and another for the remaining variables.[5] By combining these two variable orders, one can obtain an $\mathbf{X}$-constrained vtree, which is right-linear. Then, during compilation, we can employ a dynamic vtree search algorithm as long as the search algorithm ensures that the new vtree remains $\mathbf{X}$-constrained. In our case, we modified the search algorithm introduced by (Choi and Darwiche 2013). This algorithm essentially navigates the search space of vtrees by applying three different vtree operations on vtree nodes, namely, right rotation, left rotation, and children swapping. These operations are depicted in Figure 3. To make sure the new vtree will remain $\mathbf{X}$-constrained, all we need is to restrict the use of some vtree operations at certain vtree nodes. In particular, the restrictions below suffice for our purpose:

- Right rotation can be done on any vtree node.
- Left rotation can be done on any vtree node unless it is the $\mathbf{X}$-constrained vtree node.
- Children swapping can be done on any vtree node unless it is an ancestor of the $\mathbf{X}$-constrained vtree node.

7 Compiling Bayesian Networks into Weighted SDDs

We defined the SDP problem on weighted Boolean functions, while the original SDP problem is defined on Bayesian networks. Our experimental results in the next section assume a Bayesian network input. We therefore need to capture the probability distribution of a Bayesian network as a weighted Boolean function, represented by an SDD. We discuss this process briefly in this section as it is described in detail by (Darwiche 2002) and (Choi, Kisa, and Darwiche 2013). The reader is also referred to (Darwiche 2009) for an introduction to Bayesian networks.

[4] A vtree is right-linear if the left child of each internal node is a leaf. In this case, the SDD would correspond to an OBDD.

[5] We use the minfill heuristic which is commonly used in the Bayesian network literature.

Suppose we are given a Bayesian network $\mathcal{N}$ that induces a probability distribution $\Pr$ over variables $\mathbf{Z}$. The weighted Boolean function (f, W) which represents distribution $\Pr(\mathbf{Z})$ has the following variables and weights.

Indicator variables: Function f has an *indicator variable* I_x for each value x of variable X in $\mathbf{Z}$. Moreover, for each variable I_x, the corresponding weights are $W(I_x) = 1$ and $W(\neg I_x) = 1$.

Parameter variables: Function f has a *parameter variable* $P_{x|\mathbf{u}}$ for each Bayesian network parameter $\theta_{x|\mathbf{u}}$. Moreover, for each variable $P_{x|\mathbf{u}}$, the corresponding weights are $W(P_{x|\mathbf{u}}) = \theta_{x|\mathbf{u}}$ and $W(\neg P_{x|\mathbf{u}}) = 1$.

Using the above variables, (Darwiche 2002) describes an efficient CNF representation of the Boolean function f. Hence, one can compile this CNF into a constrained SDD for the purpose of computing the SDP. However, we use a more recent approach which compiles the Boolean function f directly into an SDD, therefore bypassing the construction of a CNF (Choi, Kisa, and Darwiche 2013). See also (Roth 1996) for a theoretical treatment of reducing probabilistic inference to weighted model counting, and (Sang, Beame, and Kautz 2005; Chavira and Darwiche 2008) for further practical approaches and reviews of encoding Bayesian networks into weighted CNFs.

8 Experiments

We now empirically evaluate our proposed SDD-based approach for solving $\mathsf{PP}^{\mathsf{PP}}$-complete problems. To our knowledge, the only existing system for solving $\mathsf{PP}^{\mathsf{PP}}$-complete problems is the SDP algorithm of (Chen, Choi, and Darwiche 2013), which is implemented in a beta version of the Bayesian network modeling and reasoning system, SAMIAM.[6] Hence, we compare our approach to SAMIAM's, in the task of computing SDPs in Bayesian networks. Note that both systems produce exact solutions, hence our evaluation is based on the efficiency of computing the SDP.

We evaluated our systems using the Deterministic QMR (DQMR) benchmarks,[7] which are deterministic versions of the classical diagnostic QMR benchmarks. These are two-layer networks where a top layer represents diseases and the bottom layer represents symptoms; logical-ORs are used (by DQMR) instead of noisy-ORs (by QMR) to represent the disease-symptom interactions. The DQMR benchmark consists of 120 Bayesian networks. For each network, we created an SDP problem at random, where the decision variable was chosen at random, and 10 evidence variables were selected at random and set to true (which is the more challenging setting for logical-OR networks). Further, we evaluated three settings on the number of query variables $\mathbf{H}$: 10, 20 and 30. To compute SDPs using our algorithm, we compiled Bayesian networks into SDDs as discussed in Section 7. The constrained vtrees are constructed as discussed in Section 6. The SDP algorithm of SAMIAM was run with

[6] SAMIAM is available at `http://reasoning.cs.ucla.edu/samiam`. We obtained a beta version from the authors.

[7] `http://www.cs.rochester.edu/users/faculty/kautz/Cachet`

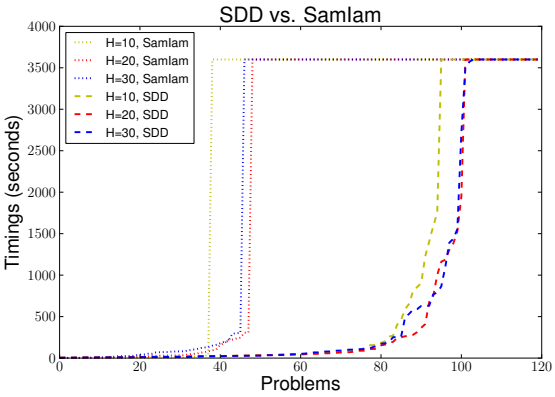


Figure 4: Experimental results on DQMR-networks.

default settings. Finally, our experiments were performed on a 2.6GHz Intel Xeon E5-2670 CPU with a 1 hour time limit and a memory limit of 12GB RAM.

Figure 4 highlights the results. On the x-axis, we sorted the 120 problem instances by running time (y-axis), where a running time of $3,600$s corresponds to an out-of-time or an out-of-memory. We make a few observations. First, our SDD-based approach solves at worst 79% of the 120 instances, over all cases (where we vary the number of query variables $\mathbf{H}$). In contrast, SAMIAM solves at best 39%, over all cases. Next, we remark that the timing for the SDD-based approach includes both the compilation time, and the SDD evaluation time to compute the SDP. In practice, compilation would be performed in an offline phase. Further, evaluation time on an SDD is relatively negligible (typically on the order of milliseconds). This is quite significant, as one expects to use the SDP in an online fashion in practice (one would continue to make observations, and re-compute the SDP until the SDP is high enough, and further observations are unlikely to be relevant). In contrast, the SDP algorithm of SAMIAM is in general invoked from scratch.

Finally, we want to stress the important role that SDDs and the dynamic vtree search played in our experiments. For that, we disabled the dynamic search and ran the experiment again. In this case, the compiled SDDs would be OBDDs as the initial vtrees were right-linear. However, this approach finished compilation for a very few instances (5 out of 360).[8]

9 Knowledge Compilation for Solving Beyond NP Problems

We will now present a perspective on this work from the broader context of solving beyond NP problems using knowledge compilation. A number of prototypical problems for beyond NP classes are defined on CNF representations of Boolean functions. Moreover, these problems become tractable when the Boolean functions are represented as *negation normal form* (NNF) circuits that satisfy certain properties. Recall that an NNF circuit contains only $\wedge$-gates or $\vee$-gates, with inputs being literals or constants. Among the well known subsets of NNF circuits are DNNFs, d-DNNFs, SDDs and OBDDs, each of which results from imposing specific properties on NNF circuits (Darwiche and Marquis 2002). One can therefore solve various NP-hard problems by *compiling* the input CNF into a NNF circuit with certain properties.

For example, the prototypical NP-complete problem SAT becomes tractable when the NNF circuit is *decomposable*, leading to DNNFs (Darwiche 2001a). Similarly, the prototypical PP-complete problem MAJSAT becomes tractable when the NNF circuit is both decomposable and *deterministic*, leading to d-DNNF (Darwiche 2001b). Further, the prototypical $\mathsf{NP}^{\mathsf{PP}}$-complete problem E-MAJSAT becomes tractable when the NNF circuit is decomposable, deterministic, and certain variables of interest appear in certain positions (Huang, Chavira, and Darwiche 2006). This work identifies a new property of SDDs that makes the prototypical $\mathsf{PP}^{\mathsf{PP}}$-complete problem MAJMAJSAT tractable. This new property is the notion of $\mathbf{X}$-constrained vtrees, which basically constrains the structure of SDDs, allowing one to solve MAJMAJSAT in time linear in the size of constrained SDD. Note that SDDs result from imposing stronger versions of decomposability and determinism on NNF circuits (Darwiche 2011). The tradition of solving Beyond NP problems using knowledge compilation can perhaps be traced to (Coste-Marquis et al. 2005), which observed that Quantified Boolean Formulas (QBFs) can be solved in linear time when the formula is represented as an OBDD whose variable order satisfies certain properties. QBF is complete for the complexity class PSPACE (Papadimitriou 1994).

10 Related Work

$\mathsf{PP}^{\mathsf{PP}}$-complete problems, such as MAJMAJSAT and SDP, are particularly challenging as they are still PP-hard given access to a PP-oracle. Analogously, $\mathsf{NP}^{\mathsf{PP}}$-complete problems, such as the MAP problem in Bayesian networks, are still NP-hard given access to a PP-oracle (Park 2002). In this context, search-based MAP algorithms have been proposed that assume an oracle for exact inference in Bayesian networks, which is PP-complete (Park and Darwiche 2003). Knowledge compilation techniques have also been employed here (Huang, Chavira, and Darwiche 2006; Pipatsrisawat and Darwiche 2009), where Bayesian networks were compiled into d-DNNFs, which were in turn used as oracles. Further, a relaxed notion of constraining a d-DNNF structure was used to produce tighter upper-bounds on the MAP solution, which improves the efficiency of search. This notion of constraining a d-DNNF is similar, but distinct from the notion of a constrained vtree which we introduced in this paper (SDDs and vtrees have stronger semantics). Further, we remark that since $\mathsf{NP}^{\mathsf{PP}} \subseteq \mathsf{PP}^{\mathsf{PP}}$, we consider more demanding problems, compared to the above prior work; in fact, our approach can be leveraged to solve MAP as well (we do not pursue this here). Another distinction is that the above works compiled Bayesian networks to d-DNNFs, but through an

[8] As the approach in (Choi, Kisa, and Darwiche 2013) is based on SDDs, using dynamic search while ensuring that the result is an OBDD is nontrivial. For a detailed empirical analysis on SDDs and OBDDs, see (Choi and Darwiche 2013) who report orders-of-magnitude better performance in favour of SDDs.

intermediate CNF representation. In contrast, we can bypass this step using SDDs, which can be more efficient in practice (Choi, Kisa, and Darwiche 2013). This is enabled by an efficient `Apply`[9] operation, which SDDs support, whereas d-DNNFs do not (Darwiche and Marquis 2002). Indeed, this also explains why we used in our experiments the bottom-up SDD compiler (Choi and Darwiche 2013), rather than the top-down SDD compiler (Oztok and Darwiche 2015); the latter compiles CNFs into SDDs, whereas the former can compile Bayesian networks into SDDs directly, using `Apply`. Moreover, the top-down compiler requires a special type of vtree as input, called a *decision vtree* (Oztok and Darwiche 2014). Thus, to use the top-down compiler for our purposes, we need to construct decision vtrees that are also $\mathbf{X}$-constrained. However, this requires a corresponding heuristic for identifying such vtrees, which is nontrivial.

11 Conclusion

We applied techniques from the domain of knowledge compilation to tackle some $\mathsf{PP}^{\mathsf{PP}}$-complete problems, which are highly intractable but can be practically important. In particular, we identified a special class of SDDs on which some $\mathsf{PP}^{\mathsf{PP}}$-complete problems can be solved in time linear in the size of the SDD. We proposed two algorithms based on this class of SDDs, one to solve the prototypical $\mathsf{PP}^{\mathsf{PP}}$-complete problem of MAJMAJSAT and another to solve the same-decision probability (SDP) problem in Bayesian networks. Empirically, our SDD-based approach significantly outperformed the state-of-the-art algorithm for computing the SDP on some benchmarks, solving over twice as many instances.

Acknowledgments. This work has been supported by ONR grant #N00014-15-1-2339 and NSF grant #IIS-1514253.

A Soundness of Algorithm 1

We show the soundness of Algorithm 1 whose proof was outlined earlier. We start with a few complementary lemmas.

Lemma 6. *Let $f = \{(p_1, s_1), \ldots, (p_n, s_n)\}$ be an $(\mathbf{X}, \mathbf{Y})$-partition. Then, for each $\mathbf{x} \models p_i(\mathbf{X})$, $s_i(\mathbf{Y}) = f(\mathbf{XY})|\mathbf{x}$.*

Proof. Take some instantiation $\mathbf{x} \models p_i(\mathbf{X})$. Since primes are mutually exclusive, $\mathbf{x} \not\models p_j(\mathbf{X})$ for $j \neq i$. Thus, $f|\mathbf{x}$ can be obtained as follows:

$$\begin{aligned} f|\mathbf{x} &= \big(p_1|\mathbf{x} \wedge s_1|\mathbf{x}\big) \vee \ldots \vee \big(p_n|\mathbf{x} \wedge s_n|\mathbf{x}\big) \\ &= \big(\bot \wedge s_1\big) \vee \ldots \vee \big(\top \wedge s_i\big) \vee \ldots \vee \big(\bot \wedge s_n\big) \\ &= s_i. \end{aligned}$$ □

Lemma 7. *Let $f = \{(p_1, s_1), \ldots, (p_n, s_n)\}$ be an $(\mathbf{X}, \mathbf{Y})$-partition and $\mathbf{e} = \mathbf{e}^l\mathbf{e}^r$ be evidence over f where $\mathbf{e}^l$ and $\mathbf{e}^r$ are the (partial) instantiations of $\mathbf{X}$ and $\mathbf{Y}$, respectively. Then, $\phi_f(\mathbf{e}) = \sum_i \phi_{p_i}(\mathbf{e}^l) \times \phi_{s_i}(\mathbf{e}^r)$.*

Proof. As f is an $(\mathbf{X}, \mathbf{Y})$-partition, the sub-functions $p_i(\mathbf{X})$ and $s_i(\mathbf{Y})$ are defined over disjoint sets of variables. So, $\phi_{p_i \wedge s_i}(\mathbf{e}) = \phi_{p_i}(\mathbf{e}^l) \times \phi_{s_i}(\mathbf{e}^r)$. Since p_i's are mutually exclusive, $(p_i \wedge s_i) \wedge (p_j \wedge s_j) = \bot$ for $i \neq j$. Hence, $\phi_f(\mathbf{e}) = \sum_i \phi_{p_i}(\mathbf{e}^l) \times \phi_{s_i}(\mathbf{e}^r)$. □

[9]The `Apply` operation combines two SDDs using any Boolean operator, and has its origins in the OBDD literature (Bryant 1986).

Corollary 1. *Let $f = \{(p_1, s_1), \ldots, (p_n, s_n)\}$ be an $(\mathbf{X}, \mathbf{Y})$-partition. Then, MC$(f) = \sum_i$ MC$(p_i) \times$ MC(s_i).*

Proof. Follows from Lemma 7, as MC$(f) = \phi_f(\top)$ when the weight function W assigns 1 to each literal. □

We are now ready to prove Lemma 1 and Lemma 2, which perform case analysis on vtree nodes.

Lemma 1. *Let α be an SDD node normalized for v, where v is not an ancestor of $v_{\mathbf{X}}$. Line 2 or Line 3 computes MC(α).*

Proof. If α is a terminal SDD, Line 2 clearly computes MC(α). Suppose that α is a decomposition SDD. Let $\alpha = \{(p_1, s_1), \ldots, (p_n, s_n)\}$. As v is not an ancestor of $v_{\mathbf{X}}$, Line 5 will never be executed for any descendants of α. So, Line 3 computes $\sum_i$ MC$(p_i) \times$ MC(s_i), which is equal to MC(α) by Corollary 1. □

Lemma 2. *Let α be an SDD node normalized for v, where v is an ancestor of $v_{\mathbf{X}}$ or $v = v_{\mathbf{X}}$. Then,*

$$vr(\alpha) = \sum_{\mathbf{y}} \big[MC(\alpha|\mathbf{y}) \geq T\big],$$

where $\mathbf{Y} = vars(v) \cap \mathbf{X}$.

Proof. The proof is by induction on the distance of v to $v_{\mathbf{X}}$.

Base case: Suppose $v = v_{\mathbf{X}}$. As v is not an ancestor of $v_{\mathbf{X}}$, Line 2 or Line 3 computes MC(α) by Lemma 1. So, Line 5 computes $vr(\alpha) = \big[\text{MC}(\alpha) \geq T\big]$. Note that $\mathbf{Y} = \emptyset$, and so the only possible instantiation of $\mathbf{Y}$ is $\mathbf{y} = \top$. As $\alpha = \alpha|\mathbf{y}$, $vr(\alpha) = \sum_{\mathbf{y}} \big[\text{MC}(\alpha|\mathbf{y}) \geq T\big]$.

Inductive step: Suppose that v is an ancestor of $v_{\mathbf{X}}$ and that the lemma holds for SDDs nodes that are normalized for v^r. Let $\mathbf{Y}^l = vars(v^l)$ and $\mathbf{Y}^r = vars(v^r) \cap \mathbf{X}$. Note that $\mathbf{Y} = \mathbf{Y}^l \cup \mathbf{Y}^r$. Let α be $\{(p_1, s_1), \ldots, (p_n, s_n)\}$. As each p_i is normalized for v^l, which is not an ancestor of $v_{\mathbf{X}}$, $vr(p_i) = \text{MC}(p_i)$ by Lemma 1. Also, as each s_i is normalized for v^r, $vr(s_i) = \sum_{\mathbf{y}^r} \big[\text{MC}(s_i|\mathbf{y}^r) \geq T\big]$ by the induction hypothesis. So, Line 3 computes the following:

$$\begin{aligned} vr(\alpha) &= \sum_i vr(p_i) \times vr(s_i) \\ &= \sum_i \text{MC}(p_i) \Big(\sum_{\mathbf{y}^r} \big[\text{MC}(s_i|\mathbf{y}^r) \geq T\big] \Big) \\ &= \sum_i \Big(\sum_{\mathbf{y}^l \models p_i} 1 \Big) \Big(\sum_{\mathbf{y}^r} \big[\text{MC}(s_i|\mathbf{y}^r) \geq T\big] \Big) \\ &= \sum_i \sum_{\mathbf{y}^l \models p_i,\, \mathbf{y}^r} \big[\text{MC}(s_i|\mathbf{y}^r) \geq T\big] \\ &= \sum_i \sum_{\mathbf{y}^l \models p_i,\, \mathbf{y}^r} \big[\text{MC}(\alpha|\mathbf{y}^l\mathbf{y}^r) \geq T\big] \text{ (by Lemma 6)} \\ &= \sum_{\mathbf{y}^l,\, \mathbf{y}^r} \big[\text{MC}(\alpha|\mathbf{y}^l\mathbf{y}^r) \geq T\big] \text{ (as } p_i\text{'s are partition)} \\ &= \sum_{\mathbf{y}} \big[\text{MC}(\alpha|\mathbf{y}) \geq T\big] \text{ (as } \mathbf{Y} = \mathbf{Y}^l \cup \mathbf{Y}^r). \end{aligned}$$ □

Since the root of SDD S is normalized for an ancestor of $v_{\mathbf{X}}$, by Lemma 2 we conclude that Algorithm 1 returns $\sum_{\mathbf{x}} \big[\text{MC}(S|\mathbf{x}) \geq T\big]$ on Line 6, which is the same as MMS$(S, \mathbf{X}, T)$. Hence, Proposition 1 holds.

B Soundness of Algorithm 2

We next prove the soundness of Algorithm 2 whose proof was outlined earlier. We start by showing two lemmas.

Lemma 8. *Let α be a function over variables $\mathbf{Z}$, and let $\mathbf{Y} \subseteq \mathbf{Z}$. Then $\phi_\alpha(\mathbf{y}\,\mathbf{e}) = \phi_\mathbf{y}\,\phi_{\alpha|\mathbf{y}}(\mathbf{e})$.*

Proof. The lemma holds due to the following.

$$\phi_\alpha(\mathbf{y}\,\mathbf{e}) = \sum_{\substack{\mathbf{z}\models\alpha\\ \mathbf{z}\sim\mathbf{y}\,\mathbf{e}}} \phi_\mathbf{z} = \sum_{\substack{\mathbf{z}\models\alpha\\ \mathbf{z}\sim\mathbf{y}\,\mathbf{e}}} \phi_\mathbf{y}\,\phi_{\mathbf{z}|\mathbf{y}} = \phi_\mathbf{y} \sum_{\substack{\mathbf{x}\models\alpha|\mathbf{y}\\ \mathbf{x}\sim\mathbf{e}}} \phi_\mathbf{x}$$
$$= \phi_\mathbf{y}\,\phi_{\alpha|\mathbf{y}}(\mathbf{e}). \quad \square$$

Lemma 9. *$\phi_\alpha(d \mid \mathbf{y}\,\mathbf{e}) = \phi_{\alpha|\mathbf{y}}(d \mid \mathbf{e})$, where α is defined over variables $\mathbf{Z}$ and $\mathbf{Y} \subseteq \mathbf{Z}$.*

Proof. The following holds due to Lemma 8.

$$\phi_\alpha(d \mid \mathbf{y}\,\mathbf{e}) = \frac{\phi_\alpha(d\,\mathbf{y}\,\mathbf{e})}{\phi_\alpha(\mathbf{y}\,\mathbf{e})} = \frac{\phi_\mathbf{y}\,\phi_{\alpha|\mathbf{y}}(d\,\mathbf{e})}{\phi_\mathbf{y}\,\phi_{\alpha|\mathbf{y}}(\mathbf{e})} = \phi_{\alpha|\mathbf{y}}(d \mid \mathbf{e}).$$
$\square$

We now prove the lemmas that show Algorithm 2 is sound.

Lemma 3. *Let α be an SDD node normalized for vtree v. Then, $vr_1(\alpha) = \phi_\alpha(\mathbf{e}_v)$.*

Proof. $vr_1(\alpha)$ is computed during the first pass over S (Lines 1–4). If α is a terminal SDD, Line 3 clearly computes $vr_1(\alpha) = \phi_\alpha(\mathbf{e}_v)$. Suppose α is an $(\mathbf{X}, \mathbf{Y})$-partition. Let $\alpha = \{(p_1, s_1), \ldots, (p_n, s_n)\}$, and let $\mathbf{e}_v = \mathbf{e}^l\mathbf{e}^r$ where $\mathbf{e}^l$ and $\mathbf{e}^r$ are the partial instantiations over $\mathbf{X}$ and $\mathbf{Y}$ respectively. Line 4 then computes $\sum_i \phi_{p_i}(\mathbf{e}^l) \times \phi_{s_i}(\mathbf{e}^r)$, which is equal to $\phi_\alpha(\mathbf{e}_v)$ by Lemma 7. $\square$

Lemma 4. *Let α be an SDD node normalized for vtree v, where v is not an ancestor of $v_\mathbf{H}$. Lines 7–8 compute $\phi_\alpha(d\,\mathbf{e}_v)$ if D is contained in vtree v, and $\phi_\alpha(\mathbf{e}_v)$ otherwise.*

Proof. Since v is not an ancestor of $v_\mathbf{H}$, Line 10 will never be executed for the descendants of α. Hence, the computations of Lines 7–8 is analogous to the computation of Lines 1–4, as in Lemma 3, except where we include d as part of the evidence $\mathbf{e}_v$ (if variable D is contained in vtree node v). That is, we compute $vr_2(\alpha) = \phi_\alpha(d\,\mathbf{e}_v)$, if D is contained in v, and $vr_2(\alpha) = \phi_\alpha(\mathbf{e}_v)$ otherwise. $\square$

Lemma 5. *Let α be an SDD node normalized for vtree v, where v is an ancestor of $v_\mathbf{H}$ or $v = v_\mathbf{H}$. Then,*

$$vr_2(\alpha) = \sum_\mathbf{y} \big[\phi_\alpha(d \mid \mathbf{y}\,\mathbf{e}) \geq T\big]\phi_\alpha(\mathbf{y}\,\mathbf{e}),$$

where $\mathbf{Y} = vars(v) \cap \mathbf{H}$.

Proof. The proof is similar to the proof of Lemma 2, and is done by induction on the distance of v to $v_\mathbf{H}$.

Base case: Suppose $v = v_\mathbf{H}$. As v is not an ancestor of $v_\mathbf{H}$, Line 7 or Line 8 computes $\phi_\alpha(d\,\mathbf{e})$ by Lemma 4. Moreover, $vr_1(\alpha) = \phi_\alpha(\mathbf{e})$ by Lemma 3. So, Line 10 computes $vr_2(\alpha) = \big[\phi_\alpha(d \mid \mathbf{e}) \geq T\big]\phi_\alpha(\mathbf{e})$. Note that $\mathbf{Y} = \emptyset$, and so the only possible instantiation of $\mathbf{Y}$ is $\mathbf{y} = \top$. As $\alpha = \alpha|\mathbf{y}$, $vr_2(\alpha) = \sum_\mathbf{y} \big[\phi_\alpha(d \mid \mathbf{y}\,\mathbf{e}) \geq T\big]\phi_\alpha(\mathbf{y}\,\mathbf{e})$.

Inductive step: Suppose that v is an ancestor of $v_\mathbf{H}$ and that the lemma holds for SDD nodes that are normalized for v^r. Let $\mathbf{Y}^l = vars(v^l)$ and $\mathbf{Y}^r = vars(v^r) \cap \mathbf{H}$. Note that $\mathbf{Y} = \mathbf{Y}^l \cup \mathbf{Y}^r$. Let α be $\{(p_1, s_1), \ldots, (p_n, s_n)\}$. Note that each p_i is normalized for v^l, and neither D nor $\mathbf{E}$ appears in $vars(v^l)$. Hence, via Lemma 4, $vr_2(p_i) = \phi_{p_i}(\top) = \phi_{p_i}$. Further, as each s_i is normalized for v^r, we have $vr_2(s_i) = \sum_{\mathbf{y}^r} \big[\phi_{s_i}(d \mid \mathbf{y}^r\,\mathbf{e}) \geq T\big]\phi_{s_i}(\mathbf{y}^r\,\mathbf{e})$ by the induction hypothesis. So, Line 8 computes the following (justifications are provided at the end):

$$vr_2(\alpha) = \sum_i vr_2(p_i) \times vr_2(s_i)$$
$$= \sum_i \phi_{p_i}\Big(\sum_{\mathbf{y}^r} \big[\phi_{s_i}(d \mid \mathbf{y}^r\,\mathbf{e}) \geq T\big]\phi_{s_i}(\mathbf{y}^r\,\mathbf{e})\Big)$$
$$= \sum_i \sum_{\mathbf{y}^l \models p_i,\, \mathbf{y}^r} \phi_{\mathbf{y}^l} \big[\phi_{s_i}(d \mid \mathbf{y}^r\,\mathbf{e}) \geq T\big]\phi_{s_i}(\mathbf{y}^r\,\mathbf{e})$$
$$= \sum_i \sum_{\mathbf{y}^l \models p_i,\, \mathbf{y}^r} \phi_{\mathbf{y}^l} \big[\phi_{\alpha|\mathbf{y}^l}(d \mid \mathbf{y}^r\,\mathbf{e}) \geq T\big]\phi_{\alpha|\mathbf{y}^l}(\mathbf{y}^r\,\mathbf{e}) \quad (3)$$
$$= \sum_i \sum_{\mathbf{y}^l \models p_i,\, \mathbf{y}^r} \big[\phi_{\alpha|\mathbf{y}^l}(d \mid \mathbf{y}^r\,\mathbf{e}) \geq T\big]\phi_\alpha(\mathbf{y}^l\,\mathbf{y}^r\,\mathbf{e}) \quad (4)$$
$$= \sum_i \sum_{\mathbf{y}^l \models p_i,\, \mathbf{y}^r} \big[\phi_\alpha(d \mid \mathbf{y}^l\,\mathbf{y}^r\,\mathbf{e}) \geq T\big]\phi_\alpha(\mathbf{y}^l\mathbf{y}^r\,\mathbf{e}) \quad (5)$$
$$= \sum_{\mathbf{y}^l,\, \mathbf{y}^r} \big[\phi_\alpha(d \mid \mathbf{y}^l\,\mathbf{y}^r\,\mathbf{e}) \geq T\big]\phi_\alpha(\mathbf{y}^l\mathbf{y}^r\,\mathbf{e}) \quad (6)$$
$$= \sum_\mathbf{y} \big[\phi_\alpha(d \mid \mathbf{y}\,\mathbf{e}) \geq T\big]\phi_\alpha(\mathbf{y}\,\mathbf{e}). \quad (7)$$

Equations (3), (4), and (5) are due to Lemma 6, Lemma 8, and Lemma 9, respectively. Equation (6) holds as primes are partitions. Equation (7) holds as $\mathbf{Y} = \mathbf{Y}^l \cup \mathbf{Y}^r$. $\square$

As the root of SDD S is normalized for an ancestor of $v_\mathbf{H}$, by Lemma 5, $vr_2(S) = \sum_\mathbf{h} \big[\phi_S(d \mid \mathbf{h}\,\mathbf{e}) \geq T\big]\phi_S(\mathbf{h}\,\mathbf{e})$. We also know that $vr_1(S) = \phi_S(\mathbf{e})$. So, Algorithm 2 returns $\text{SDP}_S(d, \mathbf{H}, \mathbf{e}, T)$ on Line 13. Hence, Proposition 3 holds.

C Complexity of SDP on Weighted CNFs

We now show that SDP on weighted CNFs is a $\mathsf{PP}^{\mathsf{PP}}$-complete problem (Theorem 1). In particular, we reduce SDP on a weighted CNF to and from SDP on a Bayesian network, which is $\mathsf{PP}^{\mathsf{PP}}$-complete. First, we can encode a weighted CNF to a Bayesian network and vice-versa, where the weighted model count of a CNF is equivalent to the probability of evidence in a Bayesian network. We can encode a weighted CNF to a Bayesian network, e.g., as shown by (Choi, Xue, and Darwiche 2012) (we additionally encode CNF weights as priors in the network). We can encode a Bayesian network as a weighted CNF, e.g., as shown by (Chavira and Darwiche 2008) (using ENC1). The reductions are polynomial-time, and results in a one-to-one correspondence between the weighted CNF models and the (non-zero) rows of the joint distribution induced by the network. Hence, the corresponding SDPs are equivalent.

References

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

Cadoli, M., and Donini, F. M. 1997. A Survey on Knowledge Compilation. *AI Commun.* 10(3,4):137–150.

Chavira, M., and Darwiche, A. 2008. On Probabilistic Inference by Weighted Model Counting. *AIJ* 172(6-7):772–799.

Chen, S.; Choi, A.; and Darwiche, A. 2013. An Exact Algorithm for Computing the Same-Decision Probability. In *IJCAI*, 2525–2531.

Chen, S.; Choi, A.; and Darwiche, A. 2014. Algorithms and Applications for the Same-Decision Probability. *JAIR* 49:601–633.

Chen, S.; Choi, A.; and Darwiche, A. 2015a. Computer Adaptive Testing Using the Same-Decision Probability. In *12th Annual Bayesian Modeling Applications Workshop (BMAW)*.

Chen, S.; Choi, A.; and Darwiche, A. 2015b. Value of Information Based on Decision Robustness. In *AAAI*, 3503–3510.

Choi, A., and Darwiche, A. 2013. Dynamic Minimization of Sentential Decision Diagrams. In *AAAI*, 187–194.

Choi, A.; Kisa, D.; and Darwiche, A. 2013. Compiling Probabilistic Graphical Models using Sentential Decision Diagrams. In *ECSQARU*, 121–132.

Choi, A.; Xue, Y.; and Darwiche, A. 2012. Same-Decision Probability: A Confidence Measure for Threshold-Based Decisions. *IJAR* 53(9):1415–1428.

Coste-Marquis, S.; Berre, D. L.; Letombe, F.; and Marquis, P. 2005. Propositional Fragments for Knowledge Compilation and Quantified Boolean Formulae. In *AAAI*, 288–293.

Darwiche, A., and Marquis, P. 2002. A Knowledge Compilation Map. *JAIR* 17:229–264.

Darwiche, A. 2001a. Decomposable Negation Normal Form. *JACM* 48(4):608–647.

Darwiche, A. 2001b. On the Tractable Counting of Theory Models and its Application to Truth Maintenance and Belief Revision. *JANCL* 11(1-2):11–34.

Darwiche, A. 2002. A Logical Approach to Factoring Belief Networks. In *KR*, 409–420.

Darwiche, A. 2003. A Differential Approach to Inference in Bayesian Networks. *JACM* 50(3):280–305.

Darwiche, A. 2009. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

Darwiche, A. 2011. SDD: A New Canonical Representation of Propositional Knowledge Bases. In *IJCAI*, 819–826.

Darwiche, A. 2014. Tractable Knowledge Representation Formalisms. In Bordeaux, L.; Hamadi, Y.; and Kohli, P., eds., *Tractability*. Cambridge University Press. 141–172.

Gimenez, F. J.; Wu, Y.; Burnside, E.; and Rubin, D. L. 2014. A Novel Method to Assess Incompleteness of Mammography Reports. In *AMIA*, 1758–1767.

Huang, J.; Chavira, M.; and Darwiche, A. 2006. Solving MAP Exactly by Searching on Compiled Arithmetic Circuits. In *AAAI*, 1143–1148.

Krause, A., and Guestrin, C. 2009. Optimal Value of Information in Graphical Models. *JAIR* 35:557–591.

Marquis, P. 1995. Knowledge Compilation Using Theory Prime Implicates. In *IJCAI*, 837–845.

Oztok, U., and Darwiche, A. 2014. On Compiling CNF into Decision-DNNF. In *CP*, 42–57.

Oztok, U., and Darwiche, A. 2015. A Top-Down Compiler for Sentential Decision Diagrams. In *IJCAI*, 3141–3148.

Papadimitriou, C. M. 1994. *Computational complexity*. Addison-Wesley.

Park, J., and Darwiche, A. 2003. Solving MAP Exactly using Systematic Search. In *UAI*, 459–468.

Park, J. 2002. MAP Complexity Results and Approximation Methods. In *UAI*, 388–396.

Pipatsrisawat, K., and Darwiche, A. 2009. A New d-DNNF-Based Bound Computation Algorithm for Functional EMA-JSAT. In *IJCAI*, 590–595.

Roth, D. 1996. On the Hardness of Approximate Reasoning. *AIJ* 82(1-2):273–302.

Sang, T.; Beame, P.; and Kautz, H. A. 2005. Performing Bayesian Inference by Weighted Model Counting. In *AAAI*, 475–482.

Selman, B., and Kautz, H. A. 1996. Knowledge Compilation and Theory Approximation. *JACM* 43(2):193–224.

Wagner, K. W. 1986. The Complexity of Combinatorial Problems with Succinct Input Representation. *Acta Informatica* 23(3):325–356.