The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Conditional PSDDs: Modeling and Learning with Modular Knowledge

Yujia Shen, Arthur Choi, Adnan Darwiche
Computer Science Department
University of California, Los Angeles
{yujias,aychoi,darwiche}@cs.ucla.edu

Abstract

Probabilistic Sentential Decision Diagrams (PSDDs) have been proposed for learning tractable probability distributions from a combination of data and background knowledge (in the form of Boolean constraints). In this paper, we propose a variant on PSDDs, called *conditional PSDDs,* for representing a family of distributions that are conditioned on the same set of variables. Conditional PSDDs can also be learned from a combination of data and (modular) background knowledge. We use conditional PSDDs to define a more structured version of Bayesian networks, in which nodes can have an exponential number of states, hence expanding the scope of domains where Bayesian networks can be applied. Compared to classical PSDDs, the new representation exploits the independencies captured by a Bayesian network to decompose the learning process into localized learning tasks, which enables the learning of better models while using less computation. We illustrate the promise of conditional PSDDs and structured Bayesian networks empirically, and by providing a case study to the modeling of distributions over routes on a map.

Introduction

The Probabilistic Sentential Decision Diagram (PSDD) is a recently proposed tractable representation for probability distributions (Kisa et al. 2014). The PSDD was motivated by the need to learn distributions in the presence of abundant background knowledge, expressed using Boolean constraints. For example, PSDDs have previously been leveraged to learn distributions in domains that can give rise to massive Boolean constraints, such as user preference rankings (Choi, Van den Broeck, and Darwiche 2015) as well as modeling routes and games (Choi, Tavabi, and Darwiche 2016; Choi, Shen, and Darwiche 2017). The typical approach for constructing a PSDD is to first *compile* Boolean constraints into a tractable logical representation called the Sentential Decision Diagram (SDD). A PSDD is then *learned* from the compiled SDD and a given dataset.

While PSDDs can be quite effective in the presence of massive Boolean constraints, they do not allow users to explicitly represent background knowledge in the form of conditional independence constraints. In contrast, probabilistic graphical models use graphs to represent such knowledge (Darwiche 2009; Koller and Friedman 2009; Murphy 2012; Barber 2012). Moreover, Bayesian networks represent such knowledge while remaining modular in a sense that we shall explain next and exploit later.

A Bayesian network has two components. The first is a directed acyclic graph (DAG) with its nodes representing variables of interest, and its topology encoding conditional independence constraints. The second component consists of a set of conditional probability tables (CPTs), one for each variable in the network. The CPT for a variable defines a set of distributions for that variable, conditioned on the states of its parents in the network. A key property of Bayesian networks is their modularity, which allows the parameter estimation problem under complete data to be decomposed into local CPT estimation problems, with closed-form solutions. However, in the presence of Boolean constraints, Bayesian networks may have very connected topologies, in addition to variables with many states and parents, potentially making them unusable practically.

Bayesian networks and PSDDs can then be viewed as two extremes on a spectrum. On the one hand, Bayesian networks can exploit background knowledge in the form of conditional independencies, but they cannot handle Boolean constraints as effectively as PSDDs. On the other hand, PSDDs can effectively incorporate background knowledge in the form of Boolean constraints, but cannot directly exploit known conditional independencies as do Bayesian networks.

In this paper, we propose a representation that inherits advantages from both representations. First, we propose the *conditional PSDD*, which is a tractable representation of probability distributions that are conditioned on the same set of variables. We then use these PSDDs to represent the conditional probability tables (CPTs) of a Bayesian network. This allows us to inherit the modularity of Bayesian networks and their ability to explicitly encode independence, while also inheriting from PSDDs their ability to effectively incorporate Boolean domain constraints. We refer to the resulting representation as a *structured Bayesian network,* as it also allows nodes with (exponentially) many states. This increases the reach of both PSDDs and Bayesian networks, on the modeling, learning and computational fronts.

This paper is organized as follows. We first review background knowledge as exploited by PSDDs, followed by a review of this representation. We next consider *modular* forms of background knowledge, which are required and exploited

L	K	P	A	Students
0	0	1	0	6
0	0	1	1	54
0	1	1	1	10
1	0	0	0	5
1	0	1	0	1
1	0	1	1	0
1	1	0	0	13
1	1	1	0	8
1	1	1	1	3

Table 1: Student enrollment data. Each column (variable) corresponds to a course, and each row (variable assignment) corresponds to an example. The counts represent the number of times that the example appeared in the dataset. For example, the second row represents those students who took Probability (P) and AI (A), but did not take Logic (L) or KR (K). There were 54 such examples (students) in this case.

by conditional PSDDs. We subsequently introduce the syntax and semantics of conditional PSDDs, while proposing a simple and efficient algorithm for learning their parameters from complete data. We then provide some empirical results to highlight the statistical advantages of conditional PSDDs and structured Bayesian networks, followed by a case study that highlights their representational advantages (modeling distributions over routes on a map). We finally close with some concluding remarks.

Learning with Background Knowledge

A common form of background knowledge is Boolean constraints, which we illustrate next with several examples. The first example we discuss is due to (Kisa et al. 2014), and concerns a computer science department that organizes four courses: Logic (L), Knowledge Representation (K), Probability (P), and Artificial Intelligence (A). The department has data on student enrollments, as in Table 1, and wishes to learn a probabilistic model of student preferences. For example, the department may use the model to infer whether students who take KR are more likely to take Logic than AI. This is a classical machine learning problem, except that we also have background knowledge in the form of program requirements and prerequisites:

– A student must take at least one of Probability or Logic.

– Probability is a prerequisite for AI.

– The prerequisite for KR is either AI or Logic.

Our goal is then to learn a model using both the data in Table 1 and the above knowledge. Effectively, what this knowledge tells us is that some examples will never appear in the dataset because they violate domain constraints — in contrast, for example, to being unlikely or missing for some idiosyncratic reason. This is valuable information and ignoring it can lead to learning a suboptimal model at best, as discussed in (Kisa et al. 2014). More precisely, the domain constraints of this example can be expressed as follows:

$$\begin{gathered} P \vee L \\ A \Rightarrow P \\ K \Rightarrow A \vee L \end{gathered}$$

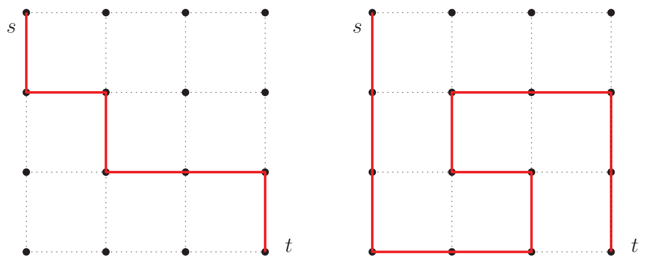


Figure 1: Routes on a 4×4 grid.

Even though there are 16 combinations of courses, this knowledge tells us that only 9 of them are valid choices. Hence, an approach that observes this information must learn a probability distribution that assigns a zero probability to every combination that violates these constraints; otherwise, it will be suboptimal.

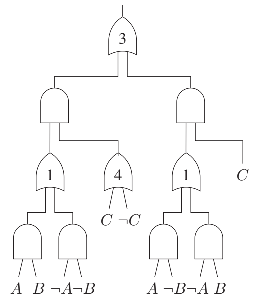
Consider now another example, where the goal is to learn a distribution over routes on a map. Figure 1 depicts two example routes on a map that has the form of a grid (Choi, Tavabi, and Darwiche 2016) — more generally, a map is modeled using an undirected graph. We can represent each edge i in the map by a Boolean variable E_i, and each route as a variable assignment that sets only its corresponding edge variables to true. In this case, each example in the dataset will be a truth assignment to edge variables E_i. Again, some examples will never appear in the dataset as they do not correspond to valid routes. For example, a route that contains disconnected edges is invalid, but other types of invalid routes may also be mandated by the domain. That is, we may already know that only simple routes are possible (i.e., no cycles), or that routes which include edge i will never include some other edge j, and so on. Again, the goal here is to be able to learn from both the dataset and the domain constraints; see (Choi, Tavabi, and Darwiche 2016; Nishino et al. 2017) for how connected-routes and simple-routes can be encoded using Boolean constraints.

We now consider our last illustrative example in which we want to learn a distribution to reason about user preferences. We have n items in this case and a corresponding number of ranks: $1, \ldots, n$. Users are asked to specify the rank j of each item i, which can be represented by a Boolean variable A_{ij}. That is, this variable is true if and only if item i has rank j. Each example in this case is also a variable assignment, which declares the ranks of some items. As is, an example could leave some item i unassigned to a rank, i.e., when variables A_{ij} are false for all ranks j. Moreover, an example could assign the same rank to multiple items, i.e., when both A_{ij} and A_{kj} are true for items $i \neq k$ and rank j.

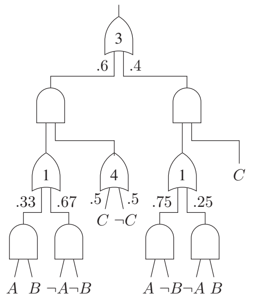
All kinds of constraints may arise in this domain of preference learning (Lu and Boutilier 2011; Huang, Kapoor, and Guestrin 2012). For example, one may know up front that all examples must correspond to total rankings, in which each item is assigned precisely one rank, and each rank is assumed by exactly one item. These constraints were considered in (Choi, Van den Broeck, and Darwiche 2015), which showed how to encode them using Boolean constraints. For

A	B	C	Pr
0	0	0	**0.2**
0	0	1	**0.2**
0	1	0	0.0
0	1	1	**0.1**
1	0	0	0.0
1	0	1	**0.3**
1	1	0	**0.1**
1	1	1	**0.1**

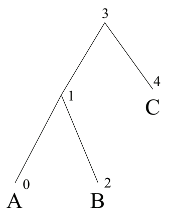
(a) Distribution



(b) SDD



(c) PSDD



(d) Vtree

Figure 2: A probability distribution and its SDD/PSDD representation. The numbers annotating or-gates in (b) & (c) correspond to vtree node IDs in (d). Moreover, while the circuit appears to be a tree, the input variables are shared and hence the circuit is not a tree.

example, when $n = 3$, the constraints are as follows:

– Each item i is assigned exactly one rank, leading to three constraints for $i \in \{1, 2, 3\}$: $(A_{i1} \wedge \neg A_{i2} \wedge \neg A_{i3}) \vee (\neg A_{i1} \wedge A_{i2} \wedge \neg A_{i3}) \vee (\neg A_{i1} \wedge \neg A_{i2} \wedge A_{i3})$.

– Each rank j is assumed by exactly one item, leading to three constraints for $j \in \{1, 2, 3\}$: $(A_{1j} \wedge \neg A_{2j} \wedge \neg A_{3j}) \vee (\neg A_{1j} \wedge A_{2j} \wedge \neg A_{3j}) \vee (\neg A_{1j} \wedge \neg A_{2j} \wedge A_{3j})$.

More generally, one needs a total of $2n$ constraints when considering n items and ranks.

As discussed in (Kisa et al. 2014), learning from both data and domain constraints is challenging for classical learning approaches. For example, when using a probabilistic graphical model, the resulting graph will almost be fully connected which makes both learning and inference very difficult. We next review the framework of Probabilistic Sentential Decision Diagrams (PSDDs), which was particularly introduced to address this challenge.

Probabilistic Sentential Decision Diagrams

PSDDs were motivated by the need to represent probability distributions $Pr(\mathbf{X})$ with many instantiations $\mathbf{x}$ attaining zero probability, $Pr(\mathbf{x}) = 0$ (Kisa et al. 2014). Consider the distribution $Pr(\mathbf{X})$ in Figure 2(a) for an example. The first step in constructing a PSDD for this distribution is to construct a special Boolean circuit that captures its zero entries; see Figure 2(b). The Boolean circuit captures zero entries in the following sense. For each instantiation $\mathbf{x}$, the circuit evaluates to 0 at instantiation $\mathbf{x}$ iff $Pr(\mathbf{x}) = 0$. The second and final step of constructing a PSDD amounts to parameterizing this Boolean circuit (e.g., by learning from data), which amounts to including a local distribution on the inputs of each or-gate; see Figure 2(c).

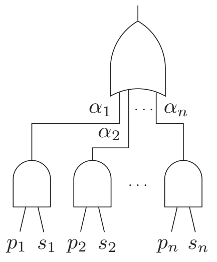


Figure 3: An SDD fragment.

The Boolean circuit underlying a PSDD is known as a Sentential Decision Diagram (SDD) (Darwiche 2011). Understanding SDD circuits is key to understanding PSDDs so we review these circuits next.

First, an SDD circuit is constructed from the fragment shown in Figure 3, where the or-gate can have an arbitrary number of inputs, and the and-gates have precisely two inputs each. Here, each p_i is called a **prime** and each s_i is called a **sub.** For example, the SDD circuit in Figure 2(b) is made up of three of these fragments and terminal SDDs (the Appendix defines terminal SDDs).

Next, each SDD circuit conforms to a tree of variables (called a *vtree*), which is just a binary tree whose leaves are the circuit variables; see Figure 2(d). The conformity is roughly as follows. For each SDD fragment with primes p_i and subs s_i, there must exist a vtree node v where the variables of SDD p_i are those of the left child of v and the variables of SDD s_i are those of the right child of v. For the SDD in Figure 2(b), each or-gate has been labeled with the ID of the vtree node it conforms to. For example, the top fragment conforms to the vtree root (ID=3), with its primes having variables $\{A, B\}$ and its subs having variables $\{C\}$.[1] The final key property of an SDD circuit is this. When the circuit is evaluated under *any* input, precisely one prime p_i of each fragment will be 1. Hence, the fragment output will simply be the value of the corresponding sub s_i.[2]

A PSDD can now be obtained by annotating a distribution $\alpha_1, \ldots, \alpha_n$ on the inputs of each or-gate, where $\sum_i \alpha_i = 1$; see again Figure 3. The distribution specified by a PSDD is as follows. Let $\mathbf{x}$ be an instantiation of the PSDD variables and suppose that we evaluate the underlying SDD circuit at input $\mathbf{x}$. If the SDD evaluates to 0, then $Pr(\mathbf{x}) = 0$. Other-

[1]The Appendix provides a formal definition of conformity. SDDs that conform to a vtree were called *normalized* in (Darwiche 2011), which also defined *compressed* SDDs. These are SDDs in which the subs of a fragment are distinct. We are assuming that SDDs are compressed in this paper.

[2]This implies that an or-gate will never have more than one 1-input. Also note that an SDD circuit may produce a 1-output for every possible input. These circuits arise when representing strictly positive distributions (with no zero entries).

wise, $Pr(\mathbf{x})$ is the product of all parameters encountered by starting at the output or-gate, and then descending down to every gate and circuit input that evaluates to 1. This PSDD distribution must be normalized as long as the local distributions on or-gates are normalized (Kisa et al. 2014).

The PSDD is a complete and canonical representation of probability distributions. That is, PSDDs can represent any distribution, and there is a unique PSDD for that distribution (under some conditions). A variety of probabilistic queries are tractable on PSDDs, including that of computing the probability of a partial variable instantiation and the most likely instantiation. Moreover, the maximum likelihood parameter estimates of a PSDD are unique given complete data, and these parameters can be computed efficiently using closed-form estimates; see (Kisa et al. 2014) for details. Finally, PSDDs have been used to learn distributions from data and domain constraints by first compiling the constraints into an SDD circuit and then learning its parameters from data. These applications included the learning of distributions over rankings and permutations (Choi, Van den Broeck, and Darwiche 2015), as well as routes and games (Choi, Tavabi, and Darwiche 2016; Choi, Shen, and Darwiche 2017).

Modular Background Knowledge

The PSDD framework can work with background knowledge in the form of arbitrary Boolean constraints since SDD circuits can be compiled from such constraints (Darwiche 2011). In some cases, however, Boolean constraints may be *modular* in a sense that we shall define precisely in this section. Modular Boolean constraints are important because they can be easily integrated with background knowledge in the form of independence constraints. Moreover, they can facilitate the compilation process into SDDs and the learning process itself, leading to improved scalability and to more accurate models. We define modular Boolean constraints next, starting with a motivation from Bayesian networks.

As mentioned earlier, the first component of a Bayesian network is a directed acyclic graph (DAG) over variables $X_1, \ldots, X_n$, which encodes conditional independence statements. In particular, for each variable X_i with parents $\mathbf{P}_i$ and non-descendants $\mathbf{N}_i$, the DAG asserts that X_i is independent of $\mathbf{N}_i$ given $\mathbf{P}_i$. The second component of a Bayesian network contains conditional probability distributions for each variable X_i, $Pr(X_i \mid \mathbf{P}_i)$, also known as the CPT for variable X_i. The independencies encoded by the DAG, together with these conditional distributions, define a unique distribution $Pr(X_1, \ldots, X_n)$ (Darwiche 2009; Koller and Friedman 2009; Murphy 2012; Barber 2012).

Consider now the following key observation. Any set of Boolean constraints Δ implied by the network distribution $Pr(X_1, \ldots, X_n)$ must be modular in the following sense. The constraints Δ can be decomposed into sets $\Delta_1, \ldots, \Delta_n$ such that:[3]

- Δ_i mentions only variable X_i and its parents $\mathbf{P}_i$.
- Δ_i does not constrain the states of parents $\mathbf{P}_i$.

[3]If such a decomposition is not possible, we can establish a contradiction with some of the probabilistic independence constraints encoded by the DAG of the Bayesian network.

A set of constraints that satisfies the above properties will be called constraints for $X_i \mid \mathbf{P}_i$, or simply *conditional constraints* when X_i and $\mathbf{P}_i$ are clear from the context.

Here is now the key implication of the above observation.

If the domain under consideration admits a DAG that captures probabilistic independence, then one can encode any underlying Boolean constraints modularly, i.e., using only conditional constraints.

The catch however is that exploiting this implication fully requires a new class of DAGs for representing independence constraints, compared to what is used in Bayesian networks. We will come back to this point later, after dwelling more on conditional constraints. In particular, we will state two properties of these constraints and provide a concrete example.

The first property of conditional constraints is this: for every parent instantiation $\mathbf{p}_i$, there is at least one state x_i that is compatible with $\mathbf{p}_i$ given Δ_i. That is, while Δ_i may eliminate some states of variable X_i under instantiation $\mathbf{p}_i$, it will never eliminate them all. We will use this property later.

The second property is that conditional constraints can always be expressed as follows. Let $\alpha_1, \ldots, \alpha_m$ be a partition of the states for parents $\mathbf{P}_i$, and let β_j be a non-empty set of states for X_i, $j = 1, \ldots, m$. Any conditional constraints for $X_i \mid \mathbf{P}_i$ can be expressed in the form "if the state of parents $\mathbf{P}_i$ is in α_j, then the state of X_i is in β_j." The converse is also true: any set of constraints of the previous form must be conditional (i.e., cannot eliminate any parent state, or eliminate all states of X_i under a given parent instantiation) .

We will now consider a concrete example of modular constraints, which are extracted from the zero parameters of a Bayesian network over three variables A, B and C. Here, variable C is a child of A and B and has the following CPT:

A	B	C	$Pr(C \mid A, B)$
a_0	b_0	c_0	0.3
a_0	b_0	c_1	0.1
a_0	b_0	c_2	0.6
a_0	b_0	c_3	**0.0**
a_0	b_1	c_0	**0.0**
a_0	b_1	c_1	0.7
a_0	b_1	c_2	0.1
a_0	b_1	c_3	0.2
a_1	b_0	c_0	**0.0**
a_1	b_0	c_1	0.7
a_1	b_0	c_2	0.1
a_1	b_0	c_3	0.2
a_1	b_1	c_0	**0.0**
a_1	b_1	c_1	0.7
a_1	b_1	c_2	0.1
a_1	b_1	c_3	0.2

Variables A and B are binary in this case, with variable C having four states. Moreover, this CPT encodes the following conditional constraints on variable C given A and B:

– if the parents satisfy $a_1 \vee b_1$, then C satisfies $c_1 \vee c_2 \vee c_3$.

– if the parents satisfy $a_0 \wedge b_0$, then C satisfies $c_0 \vee c_1 \vee c_2$.

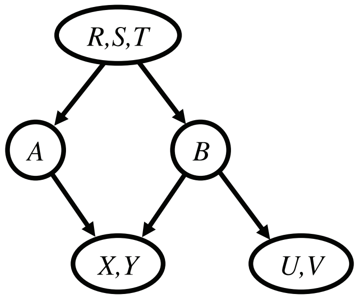


Figure 4: A cluster DAG.

We will next discuss why the integration of conditional Boolean constraints with independence constraints will generally requires a new class of DAGs for representing independence constraints.

As discussed earlier, a Bayesian network over variables $X_1, \ldots, X_n$ requires one to construct a DAG in which nodes correspond to variables $X_1, \ldots, X_n$. When the Boolean constraints are massive, the resulting DAG may end up being almost fully connected, making the Bayesian network unusable practically. To address this issue, we will work with DAGs in which nodes correspond to *clusters* of variables; see Figure 4. The independence semantics are similar to classical Bayesian networks, except that cluster DAGs can be less committing than classical DAGs. For example, the cluster DAG in Figure 4 says that variables $\{X, Y\}$ are independent of $\{R, S, T\}$ (and $\{U, V\}$) given $\{A, B\}$. Cluster DAGs, however, are silent on the independence relationships between variables in the same cluster. As we shall see in the following section, the relationships among variables in the same cluster, and between a cluster and its parent clusters, will be handled by the newly introduced conditional PSDDs.

We close this section by pointing out that in this paper, variables in a cluster DAG are assumed to be binary. Consider again the CPT above in which variable C has four states. By replacing this variable with two binary variables, X and Y, we get the following CPT:

A	B	X	Y	$Pr(X, Y \mid A, B)$
a_0	b_0	x_0	y_0	0.3
a_0	b_0	x_0	y_1	0.1
a_0	b_0	x_1	y_0	0.6
a_0	b_0	x_1	y_1	0.0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Here, state c_0 is encoded by state x_0, y_0, state c_1 is encoded by x_0, y_1 and so on, leading to the conditional constraints:

– if $a_1 \vee b_1$, then $x_1 \vee y_1$.

– if $a_0 \wedge b_0$, then $x_0 \vee y_0$.

We will refer to this example in the following section.

Conditional PSDDs

We will now introduce the *conditional PSDD* for representing a family of distributions that are conditioned on the same set of variables. That is, a conditional PSDD will play the role of a CPT in a Bayesian network. More precisely, a conditional PSDD will quantify the relationship between a cluster and its parents in a cluster DAG, leading to what we shall call a *structured Bayesian network.* Not only will this allow us to integrate Boolean and independence constraints into the learning process, but it will sometimes allow us to scale to networks whose nodes have exponentially many states (we will show how conditional PSDDs can be learned efficiently from complete data in the following section).

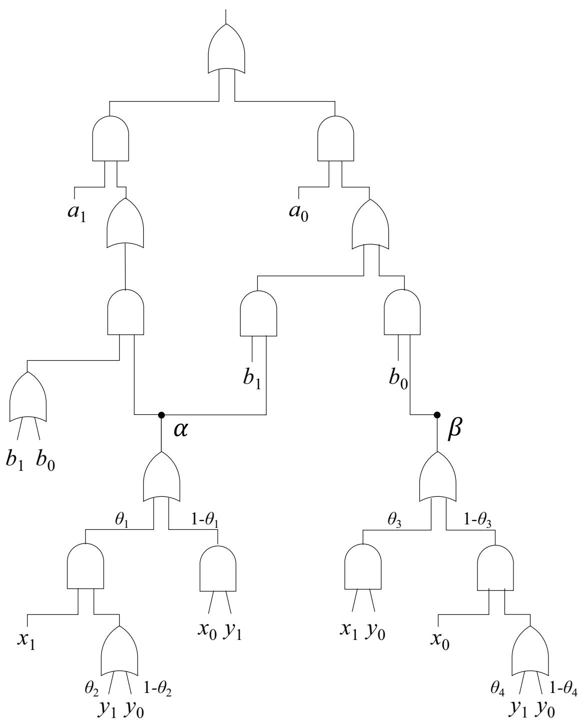


Figure 5: A conditional PSDD (vtree on left of Figure 7).

The first step in obtaining a conditional PSDD is to compile conditional constraints into an SDD. Compiling the constraints from the previous section leads to the SDD in Figure 5. We will next discuss two key properties of this SDD.

The first property concerns the SDDs labeled α and β. SDD α represents the Boolean expression $x_1 \vee y_1$, which captures the states of X, Y under $a_1 \vee b_1$. SDD β represents the Boolean expression $x_0 \vee y_0$, which captures the states of X, Y under $a_0 \wedge b_0$. Each of these SDDs can be parameterized into a PSDD to yield a distribution over the corresponding states of variables X, Y; see Figure 5.

The second property of the SDD in Figure 5 is that the output of the circuit under any input that satisfies $a_1 \vee b_1$ will be the state of α; see Figure 6. Similarly, the output of the circuit under any input that satisfies $a_0 \wedge b_0$ will be the state of β. That is, depending on the values of parents A and B, the circuit selects either PSDD α or PSDD β. This is why the circuit in Figure 5 is called a *conditional PSDD:* it represents a set of PSDDs for variables X, Y, each conditioned on some state of parents A and B.[4]

[4]According to this conditional PSDD, the distribution over vari-

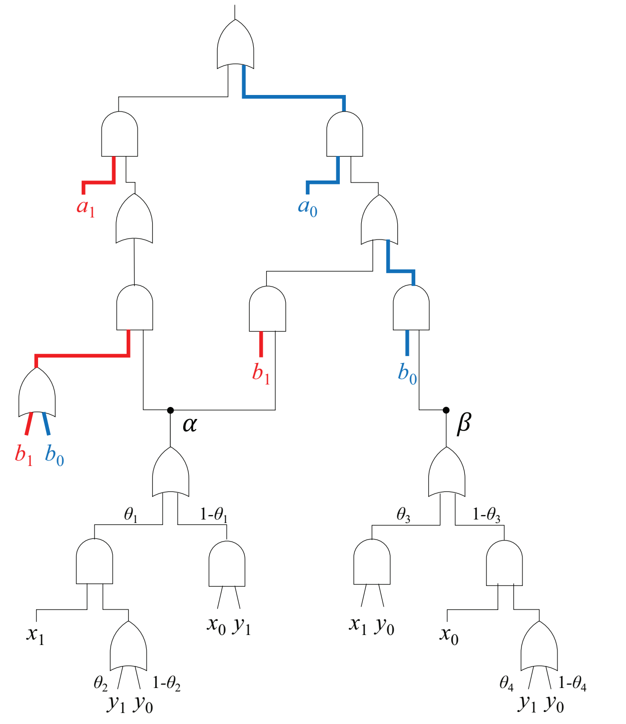


Figure 6: A partial evaluation of the conditional PSDD of Figure 5, under the input $A = a_1, B = b_1$. Wires are colored red if they are fixed to high, and colored blue if they are fixed to low. The states of uncolored wires depend on the states (values) of the inputs X and Y. In this example, the output of the circuit is the same as the value of α.

The above properties are due to choosing a specific vtree when constructing an SDD and because the SDD was compiled from modular constraints. We formalize these next.

Definition 1 (Conditional Vtrees) *Let v be a vtree for variables $\mathbf{X} \cup \mathbf{P}$ which has a node u that contains precisely the variables $\mathbf{X}$. If node u can be reached from node v by only following right children, then v is said to be a vtree for $\mathbf{X} \mid \mathbf{P}$ and u is said to be its $\mathbf{X}$-node.*

Conditional vtrees were introduced in (Oztok, Choi, and Darwiche 2016) for a different purpose, under the name of $\mathbf{P}$-constrained vtrees. Figure 7 depicts some examples of conditional vtrees for $\mathbf{X} = \{X, Y\}$ and $\mathbf{P} = \{A, B\}$. The figure also marks the $\mathbf{X}$-nodes of these vtrees.

ables X, Y is independent of the specific state of parents A and B, once we know that these parents satisfy $a_1 \vee b_1$. This corresponds to *context-specific independence* (Boutilier et al. 1996), as it says that X, Y is independent of A given b_1, and independent of B given a_1. That is, the independence is conditioned on a variable taking a specific value (X, Y are neither independent of A given B nor independent of B given A). Decision trees have also been used in a similar context by (Friedman and Goldszmidt 1998), but these trees are representationally less compact than conditional PSDDs. The latter are based on SDDs, which are decision graphs (not trees) that branch on sentences instead of variables, leading to exponentially smaller representations (Meinel and Theobald 1998; Xue, Choi, and Darwiche 2012; Bova 2016).

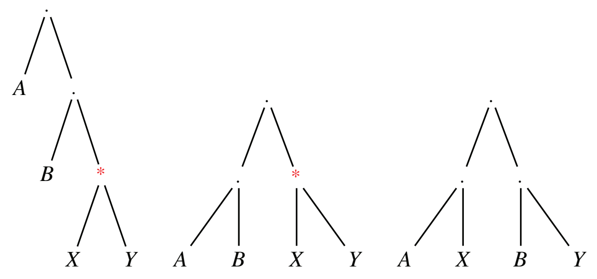


Figure 7: Two vtrees for $\mathbf{X} \mid \mathbf{P}$ with $\mathbf{X} = \{X, Y\}$ and $\mathbf{P} = \{A, B\}$ (left and center) and a vtree that is not for $\mathbf{X} \mid \mathbf{P}$ (right). The $\mathbf{X}$-nodes of the first two vtrees are starred.

We are now ready to formally define conditional PSDDs and their underlying SDDs.

Definition 2 (Conditional and Modular SDDs) *An SDD circuit for $\mathbf{X} \mid \mathbf{P}$ is one that conforms to a vtree for $\mathbf{X} \mid \mathbf{P}$. The circuit is also modular for $\mathbf{X} \mid \mathbf{P}$ if it evaluates to 1 under each instantiation $\mathbf{p}$ and some instantiation $\mathbf{x}$.*

Modularity can be described in two equivalent ways: (1) the SDD does not constrain the states of parents $\mathbf{P}$, or (2) variables $\mathbf{X}$ have at least one possible state given any instantiation of parents $\mathbf{P}$. If modularity is violated by some parent instantiation $\mathbf{p}$, then we cannot represent the conditional distribution $Pr(\mathbf{X} \mid \mathbf{p})$ for that instantiation.

Definition 3 (Conditional PSDDs) *An or-gate that feeds only from variables in $\mathbf{X}$ will be called an $\mathbf{X}$-gate. A PSDD for $\mathbf{X} \mid \mathbf{P}$ is a modular SDD for $\mathbf{X} \mid \mathbf{P}$ in which every $\mathbf{X}$-gate is parameterized.*

With $\mathbf{X} = \{X, Y\}$ and $\mathbf{P} = \{A, B\}$, we have four $\mathbf{X}$-gates in Figure 5. The following theorem shows that a conditional PSDD is guaranteed to contain a PSDD for each conditional distribution $Pr(\mathbf{X} \mid \mathbf{p})$.

Theorem 1 *Consider a modular SDD γ for $\mathbf{X} \mid \mathbf{P}$ and let u be the $\mathbf{X}$-node of its vtree. For each parent instantiation $\mathbf{p}$, there is a unique $\mathbf{X}$-gate g that (1) conforms to u and (2) has the same value as SDD γ under every circuit input $\mathbf{p}, \mathbf{x}$.*

The or-gate g will then be the root of an SDD that captures the possible states of variables $\mathbf{X}$ under parent instantiation $\mathbf{p}$. Moreover, the parameterization of this SDD leads to a PSDD for the conditional distribution $Pr(\mathbf{X} \mid \mathbf{p})$. It is critical to observe here that multiple parent instantiations $\mathbf{p}$ may map to the same or-gate g. That is, the number of PSDDs in a conditional PSDD is not necessarily exponential in the number of parents $\mathbf{P}$. More generally, the size of a conditional PSDD is neither necessarily exponential in the number of variables $\mathbf{X}$ nor their parents $\mathbf{P}$. This is the reason why structured Bayesian networks—which are cluster DAGs that are quantified by conditional PSDDs—can be viewed as Bayesian networks in which nodes can have an exponential number of states. We will later discuss a real-world application of structured Bayesian networks at length.[5]

[5]In a normalized and compressed SDD, two nodes cannot represent the same Boolean function if they conform to the same vtree

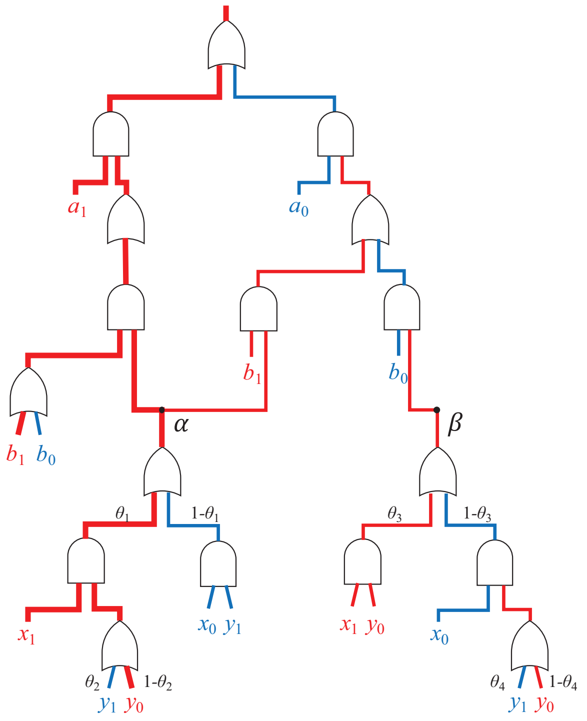


Figure 8: Evaluating the SDD of a conditional PSDD under the circuit input $A{=}a_1, B{=}b_1, X{=}x_1, Y{=}y_0$ (which can be viewed as an example in the dataset). Each wire is colored by its state (red for high and blue for low). The wires in bold are visited when descending from the root gate down to every gate and circuit input that evaluates to 1.

Learning Conditional PSDDs

Learning a conditional PSDD is akin to learning the CPT of a Bayesian network in that we are learning a set of distributions for variables $\mathbf{X}$ given their parents $\mathbf{P}$. As we shall see next, this turns out to be easy under complete data as it amounts to a process of counting examples in the dataset, which can be described using closed-form equations.

The one difference with learning CPTs is that we are now learning from constraints as well, which requires compiling the given constraints into an SDD circuit that conforms to a vtree for $\mathbf{X} \mid \mathbf{P}$. There are usually many such vtrees for a given $\mathbf{X}$ and $\mathbf{P}$ so the learning process tries to choose one that minimizes the SDD size (there is a unique SDD once the vtree is fixed). A method that searches for conditional vtrees has already been proposed in (Oztok, Choi, and Darwiche 2016), which we adapted in our experiments that we present in the next section.

Once the vtree and the SDD are fixed, the parameters of the PSDDs (embedded in our conditional PSDD) can be learned in closed form when the data is complete. This

node (Darwiche 2011). Hence, in the given definition of conditional PSDDs, each PSDD node conforming to the $\mathbf{X}$-node of the vtree will have a unique space over the variables $\mathbf{X}$, and hence a unique distribution for that space. This can be relaxed if one uses auxiliary variables, to distinguish distributions over the same space.

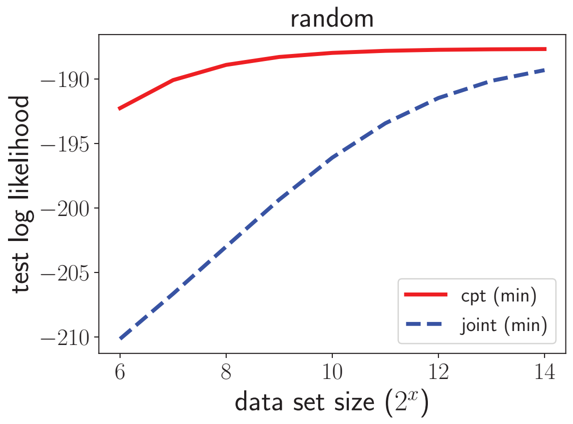


Figure 9: Conditional vs. joint PSDDs on random networks.

can be done using a method similar to the one proposed in (Kisa et al. 2014) for learning the parameters of a classical PSDD. In particular, each parameter is simply estimated by counting the number of examples in the dataset that “touches” that parameter in the PSDD. We can visualize this process using the conditional PSDD of Figure 8, for which the underlying SDD was evaluated under the example $A{=}a_1, B{=}b_1, X{=}x_1, Y{=}y_0$. The parameters touched by this example are the ones encountered by starting at the output or-gate, and then descending down to every gate and circuit input that evaluates to 1. According to this definition, parameters θ_1 and $1-\theta_2$ are touched in Figure 8.

Suppose now that $\mathcal{D}\#(\theta_i)$ and $\mathcal{D}\#(1-\theta_i)$ denote the number of examples in dataset $\mathcal{D}$ that touch each corresponding parameter. The (maximum-likelihood) parameters are then given by[6]

$$\theta_i^{\mathrm{ml}} = \frac{\mathcal{D}\#(\theta_i)}{\mathcal{D}\#(\theta_i) + \mathcal{D}\#(1-\theta_i)}.$$

Again, this parameter estimation algorithm is analogous to the one given by (Kisa et al. 2014) for classical PSDDs, except that we do not estimate parameters for gates that are not $\mathbf{X}$-gates. Moreover, the above parameter estimates are the maximum likelihood estimates for the structured Bayesian network that is quantified using conditional PSDDs.

Experimental Results

We now empirically evaluate our proposed approach using conditional PSDDs by contrasting it to the classical way in which PSDDs are used. In particular, given data and a cluster DAG with corresponding modular constraints, we learn a model in two ways. First, we compile the constraints into an SDD that we then parameterize using the data. This is the classical method which we shall refer to as “joint PSDD.” Second, we compile each set of conditional constraints into

[6]This equation is based on or-gates with two inputs. However, it can be generalized easily to or-gates with an arbitrary number of inputs, since all underlying concepts still apply to the general case.

a conditional PSDD and learn its parameters from the data. This is the proposed method which we refer to as "conditional PSDD." We finally compare the quality of the two models learned. In each case, we also try to minimize the size of compiled SDD as is classically done.

The used data and constraints are obtained by randomly generating Bayesian network structures over 100 variables $X_1, \ldots, X_{100}$. In particular, the structure of the network is obtained by randomly selecting k parents for each variable X_i from the last s variables that precede X_i, i.e., from $X_{i-s}, \ldots, X_{i-1}$ (if they are available). For our experiments, we assumed $k = s = 4$. Each variable X_i was assumed to have $2^4 = 16$ states (hence, we used 4 SDD variables to represent that space, leading to clusters of size 4). The modular constraints were generated as follows. For each variable X_i, we divide the space of parent instantiations by asserting a random parity constraint of length $\frac{k}{2}$ where k is the number of SDD variables used to represent the parents. We then divide the space recursively, until we have a partition of size 8. For each member of the partition, we constrain variable X_i using a parity constraint of length $\frac{k}{2}$ where k is now the number of SDD variables used to represent the child (i.e., $k = 4$). Note that a parity constraints eliminates half the values of variable X_i.

We simulate datasets from this Bayesian network using forward sampling. That is, we traverse the network in topological order and draw a sample from each node given the sample of the parents. Each dataset is used to learn the conditional PSDDs of the cluster DAG, as well as the corresponding joint PSDD.

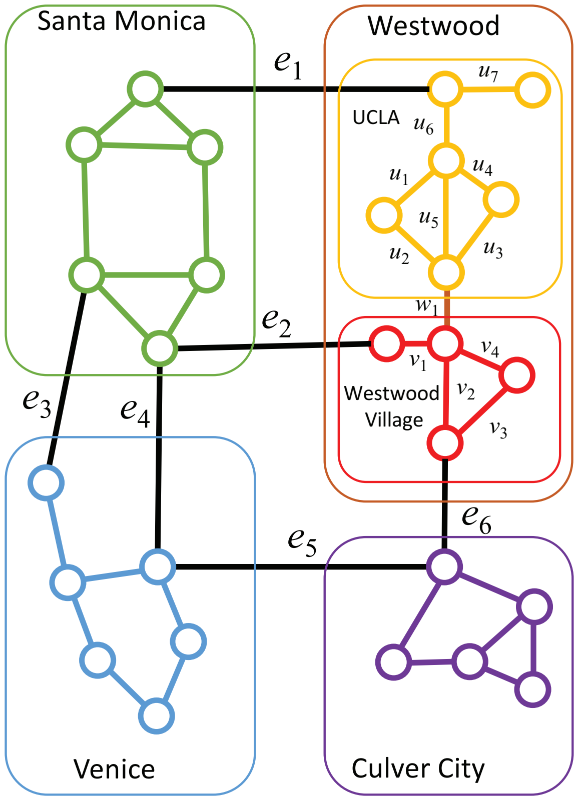
Figure 9 highlights the results. On the x-axis, we increase the size of the training set used. On the y-axis, we evaluate the test-set log likelihood (larger is better). We simulated 10 random networks, and for each network we simulated 7 different train/test pairs; hence, each point represents an average over 70 learning problems. For each learning problem, we increased the size of training sets from 2^6 to 2^{14}, and used an independent test set of size 2^{12}. We observe, in Figure 9, that the conditional PSDD obtains much better likelihoods than the joint PSDD, especially with smaller datasets. On average, the joint PSDDs had 45,618 free parameters, while the conditional PSDDs had on average 648 free parameters in total. Hence, with smaller datasets, we expect the conditional PSDDs to be more robust to overfitting.

Modeling Route Distributions

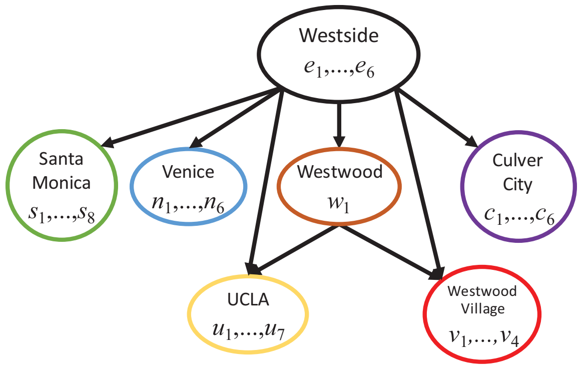
We now discuss a real-world application that is too complex to approach using a Bayesian network alone, or a PSDD alone, but where Bayesian networks and PSDDs can be mixed together to capitalize on the benefits of each. This will be made possible by utilizing structured Bayesian networks: cluster DAGs that are quantified with conditional PSDDs.

The application we consider is that of learning a distribution over routes on a map. Route distributions are of great practical importance as they can be used to estimate traffic jams, predict specific routes, and even project the impact of interventions, such as closing certain routes on a map.

As discussed previously, routes correspond to paths on a graph where each edge of the graph represents a street or highway, and each node represents an intersection. PSDDs were first proposed as a representation for route distributions in (Choi, Tavabi, and Darwiche 2016), but more as a proof of concept since these PSDDs did not scale to real-world maps. Subsequently, (Choi, Shen, and Darwiche 2017) was able to handle route distributions over real-world maps learned from GPS data by using the notion of a *hierarchical map*. We next



(a) map



Westside $e_1,\ldots,e_6$
Santa Monica $s_1,\ldots,s_8$
Venice $n_1,\ldots,n_6$
Westwood w_1
Culver City $c_1,\ldots,c_6$
UCLA $u_1,\ldots,u_7$
Westwood Village $v_1,\ldots,v_4$

(b) cluster DAG

Figure 10: A hierarchical map of neighborhoods in the Los Angeles Westside, and the corresponding cluster DAG. Each neighborhood and the roads within are depicted using colors, while roads connecting regions are depicted in black. The roads of Santa Monica, Venice and Culver City are unlabeled in the map, for clarity.

show how these hierarchical maps can be generalized and represented using cluster DAGs, while making more explicit the independence assumptions that underlie them.

In particular, (Choi, Shen, and Darwiche 2017) proposed decomposing a given graph (map) into regions as shown in Figure 10(a). This example contains a simplified graph representing the Los Angeles Westside, where nodes (intersections) of the graph have been partitioned into regions. The Los Angeles Westside is first partitioned into four sub-regions: Santa Monica, Westwood, Venice and Culver City. Westwood is further partitioned into two smaller sub-regions: UCLA and Westwood Village.

Using this hierarchical decomposition, we can plan a route between two nodes, recursively. Suppose we want a route from Santa Monica to Culver City. First, we plan a route at the top level through the regions Santa Monica, Westwood, Venice and Culver City, using the edges $e_1, \ldots, e_6$ that cross between different regions. We next decide, recursively, three sub-routes: (1) in the Santa Monica region, we plan a route from our source node to the edge e_1, (2) in the Westwood Region, we plan a route from edge e_1 to edge e_6, and (3) in the Culver City region, we plan a route from edge e_6 to the destination. Continuing with this recursive process, for the route in the Westwood region, we plan sub-routes through UCLA and Westwood Village from edge e_1 to edge w_1 to edge e_6.[7]

A hierarchical decomposition of a map can be modeled using a cluster DAG as follows; see Figure 10. Each internal cluster represents a region, with its variables being the set of edges that cross between its sub-regions. In Figure 10(b), the root cluster represents the Los Angeles Westside region and its variables represent edges $e_1, \ldots, e_6$ that are used to cross between its four sub-regions. Each leaf cluster represents the set of edges that are strictly contained in that sub-region. The parent-child relationship in the cluster DAG is based on whether an edge in the parent region can be used to enter the sub-region represented by the child cluster. For example, the UCLA region has Westside as a parent since edge e_1 can be used to enter the UCLA region directly from Santa Monica. The main point here is that a cluster DAG can be automatically generated from a hierarchical map.[8]

The use of hierarchical maps implies key independence assumptions that are made explicit by the corresponding cluster DAG. In particular, a hierarchical map implies the following. Once we know how we are entering a sub-region R, the route we take inside that sub-region is independent of the route we may take in any other sub-region that is not a descendant of R. For example, the route we take inside the UCLA region (edges $u_1, \ldots, u_7$) is independent of the route used in any other region, once we know how we plan to enter or exit the UCLA region (edges w_1 and $e_1, \ldots, e_6$).[9] Such independencies can be easily read off the cluster DAG using the Markovian assumption of Bayesian networks.

We now get to the final part of using structured Bayesian networks for representing and learning route distributions. To induce a distribution over routes in a hierarchical map, one needs to quantify the corresponding cluster DAG using conditional PSDDs. That is, for each cluster with variables $\mathbf{X}$, and whose parent clusters have variables $\mathbf{P}$, one needs to provide a conditional PSDD for $\mathbf{X} \mid \mathbf{P}$. As discussed earlier, this conditional PSDD will be based on conditional constraints for $\mathbf{X} \mid \mathbf{P}$ which define the underlying conditional SDD. The conditional PSDD is then obtained from this SDD by learning parameters from data.

The conditional constraints of a cluster can also be generated automatically from a hierarchical map and are of two types. The first type of conditional constraints rules out disconnected routes in cluster $\mathbf{X}$ (does not depend on the state of parent cluster). The second type rules out routes in region $\mathbf{X}$ which are no longer possible given how we enter or exit that region (depends on the state of parent cluster). Clearly, the root cluster only has constraints of the first type.

Conclusion

We proposed conditional PSDDs and cluster DAGs, showing how they lead to a new probabilistic graphical model: The structured Bayesian network. The new model was motivated by two needs. The first is to learn probability distributions from both data and background knowledge in the form of Boolean constraints. The second is to model and exploit background knowledge that takes the form of independence constraints. The first need has been addressed previously by PSDDs, while the second has been addressed classically using probabilistic graphical models, including Bayesian networks. Structured Bayesian networks inherits the advantages of both of these representations, including closed-form parameter estimates under complete data. We presented empirical results to show the promise of structured Bayesian networks in learning better models than PSDDs, when independence information is available. We also presented a case study on route distributions, showing the promise of structured Bayesian networks as a modeling and learning tool in complex domains.

Acknowledgments

This work has been partially supported by NSF grant #IIS-1514253, ONR grant #N00014-15-1-2339 and DARPA XAI grant #N66001-17-2-4032. We thank Isha Verma for comments and discussions relating to this paper.

Proof of Theorem 1

The proof requires a few formal definitions. Let f be a Boolean function and $\mathbf{x}$ be an instantiation of some of its variables. Then $f|_{\mathbf{x}}$ denotes the *subfunction* obtained from f by fixing the values of variables $\mathbf{X}$ to $\mathbf{x}$. In the following proofs, we use Boolean functions and SDDs exchangeably.

[7](Choi, Shen, and Darwiche 2017) assumed simple routes (no cycles) at each level of the hierarchy by excluding routes that enter or leave the same region more than once. Whether this constitutes a good approximation depends on the hierarchical decomposition used and corresponding queries (Choi, Shen, and Darwiche 2017).

[8]The proposal of (Choi, Shen, and Darwiche 2017) corresponds to a two-layer cluster DAG.

[9]In this particular case, only w_1 and e_1 are relevant to how we enter or exit the UCLA region. However, this independence is only visible in the conditional PSDD for the UCLA region as it is implied by the conditional constraints not the cluster DAG.

A *terminal SDD* is either a variable (X), its negation ($\neg X$), false ($\bot$), or true (an or-gate with inputs X and $\neg X$). Finally, we need to formally state the definition of *conformity,* which we discussed in connection to SDDs and vtrees.

Definition 4 (Conformity) *An SDD circuit n is said to conform to a vtree v iff*

– *v is a leaf with variable X, and α is a terminal SDD over variable X.*
– *v is internal, and n is an SDD fragment (see Figure 3), where the primes $p_1, ..., p_n$ are SDDs that conform to the left vtree v^l, and the subs $s_1, ..., s_n$ are SDDs that conform to the right vtree v^r.*

The proof of Theorem 1 follows directly from two lemmas that we state and prove next.

Lemma 1 *Consider a modular SDD γ for $\mathbf{X} \mid \mathbf{P}$ that conforms to vtree v, and let u be the $\mathbf{X}$-node of vtree v. For each parent instantiation $\mathbf{p}$, there is an $\mathbf{X}$-gate g that represents the subfunction $\gamma|_{\mathbf{p}}$.*

Proof The proof is by induction on the level of node u in the vtree.

- Base Case: $u = v$. Then $\mathbf{P} = \emptyset$ and there is a unique (empty) instantiation $\mathbf{p}$, where $\gamma|_{\mathbf{p}} = \gamma$.
- Inductive Step $u \neq v$: Then SDD γ is a fragment; see Figure 3. Since SDD γ conforms to v, each prime p_i conforms to v^l and each sub s_i conforms to v^r. Since u is the $\mathbf{X}$-node, it can be reached from the root v by iteratively following right children. Hence, $\mathbf{X} \subseteq \mathsf{vars}(v^r)$ and $\mathsf{vars}(v^l) \subseteq \mathbf{P}$. Let $\mathbf{P}^l = \mathsf{vars}(v^l)$ (parent variables to the left of v) and $\mathbf{P}^r = \mathbf{P} \setminus \mathsf{vars}(v^l)$ (parent variables on the right of v). Further, let $\mathbf{p}^l$ and $\mathbf{p}^r$ denote the corresponding sub-instantiations of $\mathbf{p}$. In an SDD, the primes p_i are mutually-exclusive and exhaustive, hence $\gamma|_{\mathbf{p}^l} = s_i$ for some unique i. Moreover, since γ is modular for $\mathbf{X} \mid \mathbf{P}$, then $\gamma|_{\mathbf{p}^l}$ must be modular for $\mathbf{X} \mid \mathbf{P}^r$. Since $\gamma|_{\mathbf{p}^l} = s_i$ further conforms to v^r, then by induction there is an $\mathbf{X}$-gate g representing $s_i|_{\mathbf{p}^r}$. Gate g is also the one representing $\gamma|_{\mathbf{p}}$ since: $\gamma|_{\mathbf{p}} = (\gamma|_{\mathbf{p}^l})|_{\mathbf{p}^r} = s_i|_{\mathbf{p}^r} = g$. □

Lemma 2 *Consider a modular SDD γ for $\mathbf{X} \mid \mathbf{P}$ that conforms to vtree v, and let u be the $\mathbf{X}$-node of vtree v. For each parent instantiation $\mathbf{p}$, if there is an $\mathbf{X}$-gate g that represents $\gamma|_{\mathbf{p}}$, then gate g (1) conforms to u, (2) evaluates to 1 on $\mathbf{x}$ iff γ evaluates to 1 on $\mathbf{p}, \mathbf{x}$, and (3) is unique.*

The proof uses a property of normalized and compressed SDDs that underlie conditional PSDDs. That is, in such SDDs, we cannot have two distinct or-gates that conform to the same vtree node yet represent the same Boolean function (Darwiche 2011).

Proof We prove each part in turn.

1. By Lemma 1, there is an $\mathbf{X}$-gate g representing $\gamma|_{\mathbf{p}}$. From the proof of Lemma 1, this $\mathbf{X}$-gate g is obtained by following a path in the circuit γ through the subs. By conformity, each of these subs comform to the corresponding right child of a vtree node. Hence, g must conform to u.
2. For a given instantiation $\mathbf{p}$, we have $\gamma(\mathbf{p}, \mathbf{x}) = \gamma|_{\mathbf{p}}(\mathbf{x})$, which is equivalent to $g(\mathbf{x})$ from the proof of Lemma 1.
3. Suppose that there exists a gate h that conforms to u and which evaluates to 1 on $\mathbf{x}$ iff γ evaluates to 1 on $\mathbf{p}, \mathbf{x}$. It follows that $h(\mathbf{x}) = g(\mathbf{x})$ for all $\mathbf{x}$. Since the SDD is normalized and compressed, we must have $h = g$. □

References

Barber, D. 2012. *Bayesian Reasoning and Machine Learning*. Cambridge University Press.

Boutilier, C.; Friedman, N.; Goldszmidt, M.; and Koller, D. 1996. Context-specific independence in Bayesian networks. In *UAI*, 115–123.

Bova, S. 2016. SDDs are exponentially more succinct than OBDDs. In *AAAI*, 929–935.

Choi, A.; Shen, Y.; and Darwiche, A. 2017. Tractability in structured probability spaces. In *NIPS*.

Choi, A.; Tavabi, N.; and Darwiche, A. 2016. Structured features in naive Bayes classification. In *AAAI*.

Choi, A.; Van den Broeck, G.; and Darwiche, A. 2015. Tractable learning for structured probability spaces: A case study in learning preference distributions. In *IJCAI*.

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 *Proceedings of IJCAI*, 819–826.

Friedman, N., and Goldszmidt, M. 1998. Learning bayesian networks with local structure. In *Learning in graphical models*. Springer. 421–459.

Huang, J.; Kapoor, A.; and Guestrin, C. 2012. Riffled independence for efficient inference with partial rankings. *J. Artif. Intell. Res. (JAIR)* 44:491–532.

Kisa, D.; Van den Broeck, G.; Choi, A.; and Darwiche, A. 2014. Probabilistic sentential decision diagrams. In *KR*.

Koller, D., and Friedman, N. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Lu, T., and Boutilier, C. 2011. Learning Mallows models with pairwise preferences. In *ICML*, 145–152.

Meinel, C., and Theobald, T. 1998. *Algorithms and Data Structures in VLSI Design: OBDD — Foundations and Applications*. Springer.

Murphy, K. P. 2012. *Machine Learning: A Probabilistic Perspective*. MIT Press.

Nishino, M.; Yasuda, N.; Minato, S.; and Nagata, M. 2017. Compiling graph substructures into sentential decision diagrams. In *AAAI*, 1213–1221.

Oztok, U.; Choi, A.; and Darwiche, A. 2016. Solving PP^{PP}-complete problems using knowledge compilation. In *KR*, 94–103.

Xue, Y.; Choi, A.; and Darwiche, A. 2012. Basing decisions on sentences in decision diagrams. In *AAAI*, 842–849.