Learning the Structure of Probabilistic Sentential Decision Diagrams

Yitao Liang
Computer Science Department
University of California, Los Angeles
yliang@cs.ucla.edu

Jessa Bekker
Computer Science Department
KU Leuven
jessa.bekker@cs.kuleuven.be

Guy Van den Broeck
Computer Science Department
University of California, Los Angeles
guyvdb@cs.ucla.edu

Abstract

The probabilistic sentential decision diagram (PSDD) was recently introduced as a tractable representation of probability distributions that are subject to logical constraints. Meanwhile, efforts in tractable learning achieved great success inducing complex joint distributions from data without constraints, while guaranteeing efficient exact probabilistic inference; for instance by learning arithmetic circuits (ACs) or sum-product networks (SPNs). This paper studies the efficacy of PSDDs for the standard tractable learning task without constraints and develops the first PSDD structure learning algorithm, called LEARNPSDD. Experiments on standard benchmarks show competitive performance, despite the fact that PSDDs are more tractable and more restrictive than their alternatives. LEARNPSDD compares favorably to SPNs, particularly in terms of model size, which is a proxy for tractability. We report state-of-the-art likelihood results on six datasets. Moreover, LEARNPSDD retains the ability to learn PSDD structures in probability spaces subject to logical constraints, which is beyond the reach of other representations.

1 INTRODUCTION

Tractable learning aims to induce complex, yet tractable probability distributions from data (Domingos et al., 2014; Mauro and Vergari, 2016). The learned tractable model serves as a certificate to the user that any query that arises can always be answered efficiently. In practice, this means that any conditional marginal can be computed in time linear in the size of the learned model.

Current tractable learning efforts stem from two lines of work. First, probabilistic graphical model learning has long targeted sparse models (Meila and Jordan, 2000; Narasimhan and Bilmes, 2004; Chechetka and Guestrin, 2007). Second, the field of knowledge compilation studies tractable representations, such as *arithmetic circuits* (ACs) for probability distributions (Darwiche, 2003), and NNF circuits for Boolean functions (Darwiche and Marquis, 2002). The superior tractability of these circuits derives from their ability to capture local structure and determinism (Boutilier et al., 1996), which makes compilation to circuits a state-of-the-art technique for probabilistic inference (Darwiche et al., 2008; Choi et al., 2013). Recently, circuits have also become the chosen target representation for tractable learners (Lowd and Domingos, 2008; Lowd and Rooshenas, 2013; Gens and Domingos, 2013; Dennis and Ventura, 2015; Bekker et al., 2015), spurring innovation in arithmetic circuit dialects such as *sum-product networks* (SPNs) (Poon and Domingos, 2011; Peharz et al., 2014) and *cutset networks* (Rahman et al., 2014). While closely related, these representations differ significantly in the types of tractable queries and operations they support.

This paper considers the *probabilistic sentential decision diagram* (PSDD) (Kisa et al., 2014b), which is perhaps the most powerful circuit proposed to date. Owing to their intricate structural properties, PSDDs support closed-form parameter learning, MAP inference, complex queries (Bekker et al., 2015), and even efficient multiplication of distributions (Shen et al., 2016), which are all increasingly rare. These strong properties permit learning of PSDDs in probability spaces that are subject to complex logical constraints disallowing large numbers of possible worlds (Kisa et al., 2014a). In this context, knowledge compilation algorithms can build PSDD structures without looking at the data. These structures are large enough that parameter estimation was shown sufficient to learn distributions over game traces (Choi et al., 2016), configurations, and yield state-of-the-art results learning preference distributions (Choi et al., 2015).

These observations raise two questions: (i) are PSDDs amenable to tractable learning when no logical constraints or compiled circuit are available a priori, and (ii) can we still learn PSDDs that are subject to logical constraints while also fitting the data well; that is, perform true structure learning? To answer both questions, we develop LEARNPSDD, which is the first structure learning algorithm for PSDDs. It uses local operations on the PSDD circuit that maintain the desired circuit properties, while steadily increasing model fit. LEARNPSDD is supported by a vtree learning algorithm that captures the data's independencies in a tree structure, which we empirically show to be an essential step of the learning process. Moreover, using expectation maximization on top of LEARNPSDD, we show competitive results on the standard tractable learning benchmarks. When additionally performing bagging, our PSDD learner reports state-of-the-art results on six datasets. Finally, the proposed algorithm is general and retains the ability to learn in logically constrained probability spaces. Here, we empirically show that LEARNPSDD is able to refine the circuits compiled from constraints, yielding superior likelihood scores.

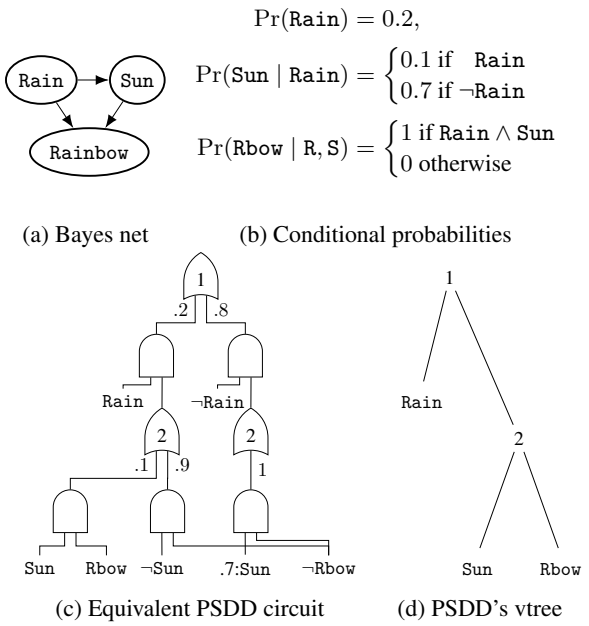


Figure 1: A Bayesian network and its equivalent PSDD.

2 A TRACTABLE REPRESENTATION

This section introduces the notation and circuit representation we employ throughout this paper.

Notation An uppercase letter X denotes a Boolean random variable and a lowercase letter x denotes an assignment to X. Literals X or $\neg X$ respectively assign true or false to variable X. Sets of variables $\mathbf{X}$ and joint assignments $\mathbf{x}$ are denoted in bold. An assignment $\mathbf{x}$ that satisfies logical sentence α is denoted $\mathbf{x} \models \alpha$. Concatenations of sets represent their union. A complete assignment to all variables is a possible world.

PSDDs Probabilistic sentential decision diagrams (PSDDs) are circuit representations of joint probability distributions over binary variables (Kisa et al., 2014b); they are probabilistic extensions of sentential decision diagrams (SDDs) (Darwiche, 2011), which represent Boolean functions as logical circuits. A PSDD is a parametrized directed acyclic graph (DAG), as depicted in Figure 1c. Each inner node is either a logical AND gate with two inputs, or a logical OR gate with an arbitrary number of inputs, and the types of nodes alternate. Each terminal (input) node is a univariate distribution, which could either be X when X is always true, $\neg X$ when it is always false, or $(\theta : X)$ when it is true with probability θ. A decision node is the combination of an OR gate with its AND gate inputs. We refer to the left input of an AND gate as its prime (denoted p) and the right one as its sub (denoted s). The n wires in each decision node are annotated with a normalized probability distribution $\theta_1, \ldots, \theta_n$. Alternatively, we refer to a decision node's labeled AND gates as its elements and represent the decision node itself as a set of elements $\{(p_1, s_1, \theta_1), \ldots, (p_n, s_n, \theta_n)\}$.

Syntactic Restrictions According to the semantics we will detail later, each PSDD node represents a probability distribution over the random variables that appear below it. However, each AND gate must be *decomposable*, meaning that its inputs represent a distribution over disjoint sets of variables. This is enforced uniformly throughout the circuit by a variable tree (vtree): a full, binary tree, whose leaves are labeled with variables; see Figure 1d. The internal vtree nodes split variables into those appearing in the left subtree $\mathbf{X}$ and those in the right subtree $\mathbf{Y}$. This implies that the corresponding PSDD decision nodes must have primes ranging over $\mathbf{X}$ and subs over $\mathbf{Y}$. We say the corresponding PSDD nodes are *normalized* for the vtree node. Figure 1c labels decision nodes with the vtree node they are normalized for.

Each decision node must be *deterministic*, meaning that for any single possible world, it can have at most one prime assign a non-zero probability to that world. In other words, the supports of all distributions represented by primes must be disjoint within the same decision node. We further assume that all elements assign a non-

zero probability to at least one world.[1]

Semantics Each PSDD node represents a probability distribution, starting with the terminal nodes' univariate distributions. Each decision node q normalized for a vtree node with $\mathbf{X}$ and $\mathbf{Y}$ in its left and right subtrees respectively, represents a distribution over $\mathbf{XY}$ as $\Pr_q(\mathbf{XY}) = \sum_i \theta_i \Pr_{p_i}(\mathbf{X}) \Pr_{s_i}(\mathbf{Y})$. Under these semantics, the PSDD in Figure 1c represents the same distribution as the Bayesian network in Figure 1a.

Each PSDD node's distribution has an intricate support over which it defines a non-zero probability. We refer to this support as the *base* of node q, written $[q]$. The base of a node can alternatively be defined as a logical sentence using the recursion $[q] = \bigvee_i [p_i] \wedge [s_i]$, where $[X] = X$, $[\neg X] = \neg X$, and $[\theta : X] = \text{true}$.

From a top-down perspective, a decision node presents a choice between its prime bases $[p_i]$: at most one is true in each world. Thus, the PSDD is a decision diagram branching on which sentence $[p_i]$ is true. This generalizes decision trees or binary decision diagrams which only branch on the value of a single variable. To reach node q, all the primes on a path to q must be satisfied; they are the sub-context of q. The disjunction of all q's sub-contexts is its *context* γ_q. This notion lets us precisely characterize PSDD parameter semantics: they are conditional probabilities in root node r's distribution:

$$\theta_i = \Pr_r([p_i] \mid \gamma_q).$$

Inference and Learning PSDDs have several desirable properties. The probability of any (partial) assignment $\mathbf{x}$ can be computed in time linear in the PSDD size (Kisa et al., 2014b). Moreover, PSDDs support efficient complex queries, such as count queries (Bekker et al., 2015), and can be multiplied efficiently (Shen et al., 2016). Most pertinently, the maximum-likelihood estimate for each PSDD parameter is calculated in closed form by observing the fraction of complete examples flowing through the relevant wire. More precisely, out of all the examples that agree with the node context γ_q, the parameter estimate is the fraction of examples that also agrees with the prime base $[p_i]$ (Kisa et al., 2014b):

$$\hat{\theta}_i = \frac{\mathcal{D}\#(\gamma_q, [p_i])}{\mathcal{D}\#(\gamma_q)}. \tag{1}$$

To prevent overfitting, Laplace smoothing is used.

[1] In the original definition this was not required. In fact, primes were required to be exhaustive, which can necessitate a zero-probability element (Kisa et al., 2014b). This is an artifact from defining PSDDs as an extension of SDDs, which require exhaustiveness to support negation or disjunction. These logical operations are not used for our (probabilistic) purpose.

In all current PSDD applications, the learner is given a logical sentence α that encodes domain knowledge (e.g., a constraint encoding rankings or game traces). Using knowledge compilation, sentence α is first transformed into an SDD circuit, and second into a PSDD by parameter learning. Prior work does not perform structure learning: no data is used to come up with PSDD structures.

3 VTREE LEARNING

To learn a vtree from data, it is important to understand the assumptions that are implied by a choice of vtree. PSDDs recursively decompose the distribution by conditioning it on the prime bases $[p_i]$. Specifically, each decision node decomposes the distribution into independent distributions over $\mathbf{X}$ and $\mathbf{Y}$, guided by the vtree.

Proposition 1. *(Kisa et al., 2014b) Prime and sub variables are independent in PSDD q, given a prime base:*

$$\begin{aligned}\Pr_q(\mathbf{XY} \mid [p_i]) &= \Pr_q(\mathbf{X} \mid [p_i]) \Pr_q(\mathbf{Y} \mid [p_i]) \\ &= \Pr_{p_i}(\mathbf{X}) \Pr_{s_i}(\mathbf{Y}).\end{aligned}$$

Independence given a logical sentence is called *context-specific independence* (Boutilier et al., 1996). Which context-specific independencies can be exploited, as specified by the vtree, has a crucial impact on PSDD size.

Prior work always obtained its vtree from compiling logical constraints into SDD circuits (Choi and Darwiche, 2013), disregarding the dependencies that are implied by this choice. We propose a novel method that does induce vtrees based on the independencies found in the data.

A common way to quantify the level of independence between two sets of variables is their mutual information:

$$\text{MI}(\mathbf{X}, \mathbf{Y}) = \sum_{\mathbf{x}} \sum_{\mathbf{y}} \Pr(\mathbf{xy}) \log \frac{\Pr(\mathbf{xy})}{\Pr(\mathbf{x}) \Pr(\mathbf{y})}.$$

Intuitively, low mutual information suggests that $\mathbf{X}$ and $\mathbf{Y}$ are almost independent, and that the data distribution can be approximated by a PSDD that satisfies Proposition 1 using only a small number of primes in each decision node. Therefore, we let mutual information guide the learner: our objective is to induce a vtree that minimizes the mutual information between the $\mathbf{X}$ and $\mathbf{Y}$ variables as they are split in each internal vtree node. Additionally, we will aim to balance the vtrees. We observe that this tends to produce smaller PSDDs in practice.

However, estimating mutual information between large $\mathbf{X}$ and $\mathbf{Y}$ requires estimating an exponential number of terms $\Pr(\mathbf{xy})$, each of which is hard to estimate accurately from data. Therefore, we approximate mutual information by average pairwise mutual information:

$$\text{pMI}(\mathbf{X}, \mathbf{Y}) = \text{avg}_{X \in \mathbf{X}, Y \in \mathbf{Y}} \text{MI}(\{X\}, \{Y\}).$$

Algorithm 1: Split(q, i, Zs, m)

Input: q, i: the ith element of node q to split, Zs: mutually exclusive and exhaustive set of variable assignments, m: depth of PartialCopy

Result: The ith element of node q is split on Zs.

```
n2c = ∅                          // maps nodes to copies
RemoveElement(q, (p_i, s_i))
foreach z ∈ Zs do
    PartialCopy(p_i, z, m, n2c)
    PartialCopy(s_i, true, m, n2c)
    AddElement(q, (n2c[p_i], n2c[s_i]))
```

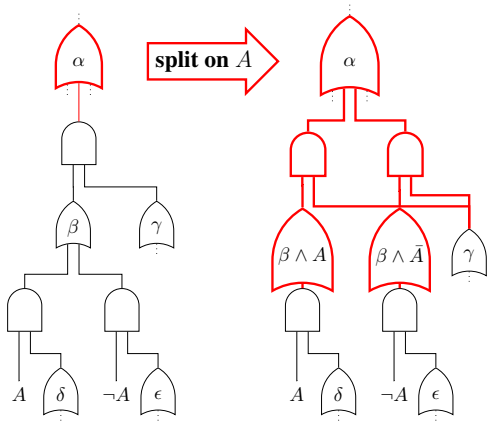


Figure 2: **Minimal Split.** Nodes labels are their base.

We present two algorithms for optimizing a vtree's pMI.

Top-down vtree induction starts with the full variable set and recursively finds splits. Every step divides the variables into two equally-sized subsets with minimal pMI. Finding splits is reduced to a balanced min-cut problem, for which optimized solvers exist (Karypis, 2013).

Bottom-up vtree induction starts with singleton sets of variables at the bottom of the vtree. For each level of the vtree, it pairs two vtrees of the level below, maximizing the pMI of the pairs, in order to minimize the pMI of future pairings at higher levels. Finding pairings of vtrees reduces to the minimum-cost perfect matching problem, for which optimized solvers exist (Kolmogorov, 2009).

Both methods greedily solve the same problem. The difference lies in the direction of the greedy optimization. Top-down induction begins at the root and will therefore get the best splits at the higher levels. Bottom-up starts from the leaves and will therefore get the best pairings at the lower levels. Section 7 will present an empirical comparison showing that bottom-up induction outperforms the top-down approach. Intuitively, most interactions occur between small numbers of variables, which makes the lower levels of the vtree more important.

Algorithm 2: Clone(q, P, m)

Input: q: node to clone, P: parent nodes and elements to redirect to clone, m: depth of PartialCopy

Result: Parents P are redirected to the clone of q.

```
n2c = ∅                          // maps nodes to copies
PartialCopy(q, true, m, n2c)
foreach (π, i) ∈ P do Update(π, (i, q, n2c[q]))
```

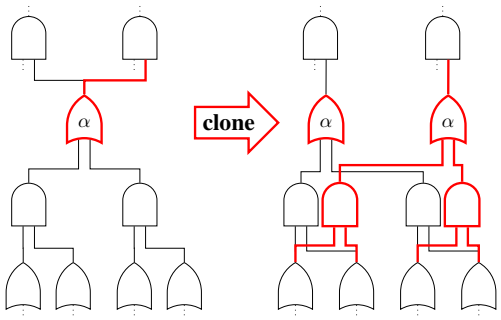


Figure 3: **Minimal Clone.** Base α does not change.

4 PSDD STRUCTURE LEARNING

This section presents the first algorithm to learn PSDD structure from data. The objective is to obtain a compact structure that approximates the data distribution well.

We propose two operations, split and clone, that incrementally change the PSDD structure while keeping the PSDD syntactically sound and the base of the root node unaltered. The soundness criteria guarantees that the learned PSDD follows the syntactic definitions described in Section 2. Not changing the root node's base guarantees that any constraint (i.e., domain knowledge) that is encoded in the PSDD remains intact. Our learner applies these operations greedily to optimize a score function.

4.1 PSDD OPERATIONS

A split or clone operation changes the PSDD structure to represent a different distribution over the same base.

The split operation splits an element (AND node) into multiple elements by constraining the prime. The elements are split based on a mutually exclusive (disjoint) and exhaustive set of partial assignments to the prime variables. This ensures that the decision node remains deterministic. Indeed, for any assignment to the prime variables that had a non-zero probability in the element before the split, there can be at most one element after the split that assigns a non-zero probability to it. To execute a split (Figure 2, Algorithm 1), a new element is created for each partial assignment, where the new prime is a copy of the original prime constrained by the assignment. The

new sub is an unconstrained copy. The original element is removed from its decision node.

The clone operation makes a copy of a node and redirects some of the parents to the copy (Figure 3, Algorithm 2).

Both operations need to make partial copies of a decision node and its descendants. Our algorithm can perform these copies up to some specified depth m. A minimal operation ($m = 0$) copies as few nodes as possible, and a complete operation copies all nodes. Any non-minimal operation ($m > 0$) is equivalent to multiple minimal operations. The complete description of the partial-copy algorithm is given in Appendix A.

Finally, Appendix B proves the following result.

Proposition 2. *Splits and clones maintain a PSDD's syntactic properties and do not alter the base of its root.*

4.2 SPLIT AND CLONE ARE LOCAL

Splits and clones are local operations. Only the node that is modified, the parents that are redirected and the copied descendants are affected. Furthermore, key properties of an operation, such as the required change in PSDD structure and improvement in likelihood are typically not affected by operations elsewhere in the PSDD.

Local operations have four desirable properties. First, the complexity of executing an operation is bounded by the number of elements it affects; cheap operations are thus possible in large PSDDs. Second, the difference in PSDD size after an operation can be easily obtained; it is the difference in the affected elements.

Third, the difference in likelihood can be computed by only looking at elements that are affected. Indeed, Kisa et al. (2014b, long version) prove that the log-likelihood decomposes over the PSDD elements as follows.

Proposition 3. *The log-likelihood of PSDD r given data $\mathcal{D}$ is a sum of log-likelihood contributions per node:*[2]

$$\ln \mathcal{L}(r|\mathcal{D}) = \ln \Pr_r(\mathcal{D}) = \sum_{q \in r} \sum_{i \in q} \ln \theta_{q,i}\, \mathcal{D}\#(\gamma_q, [p_{q,i}]),$$

where $\mathcal{D}\#(\gamma_q, [p_{q,i}])$ is the number of examples that satisfy the node context of q and the base of q's prime $p_{q,i}$.

Fourth, we would like to simulate candidate operations before committing to execute them. Because size and likelihood changes are not affected by other operations, we can cache their values when considering a large number of candidate operations during structure search.

Local operations support principled tractable learning, using exact estimates of likelihood and tractability (size). Many other learners, especially traditional ones, are required to approximate the likelihood and have no ability to reliably determine the tractability of a learned model.

[2]This equation treats terminal nodes as degenerate decision nodes with primes X and $\neg X$, and subs true and false .

4.3 LEARNPSDD ALGORITHM

We build on our split and clone operations to create the first PSDD structure learning algorithm called LEARNPSDD[3]. It incrementally improves the structure of an existing PSDD to better fit the data. In every step, the structure is changed by executing an operation. Learning continues until the log-likelihood on validation data stagnates, or a desired time or size limit is reached. The operation to execute is greedily chosen based on the best likelihood improvement per size increment:

$$\text{score} = \frac{\ln \mathcal{L}(r' \mid \mathcal{D}) - \ln \mathcal{L}(r \mid \mathcal{D})}{\text{size}(r') - \text{size}(r)}$$

where r is the original and r' the updated PSDD.

The algorithm needs to be provided with an initial PSDD and vtree. It can take any PSDD, even one that encodes domain knowledge in its base, as is done in existing applications of PSDDs. It can also be a trivial, maximally uninformative PSDD q whose base $[q] =$ true and whose distribution factorizes completely over the variables. The vtree can either come from compiling those constraints, or can be learn from data as described in Section 3.

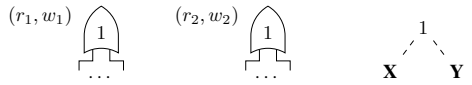
In each iteration, LEARNPSDD considers one clone per node and one split per element. The clone is the best clone for that node where at most k parents are moved to the copy. The split is the best split with the partial assignments limited to one prime variable. Only the scores of the operations that use nodes affected by the previous iteration's operation need to be recalculated.

The operation depth parameter m is fixed during learning. The larger this parameter, the more elements are added and the larger the log-likelihood improvement per operation. A large m speeds up the learning but learns larger PSDDs, which are more prone to overfitting. Appendix C provides more implementation details.

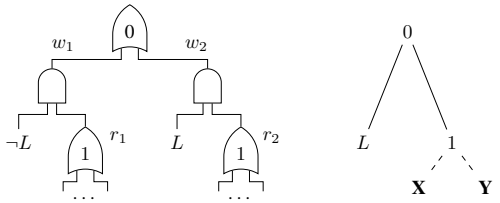
5 ENSEMBLES OF PSDDS

This section extends LEARNPSDD to induce mixtures of PSDDs. A mixture of PSDDs $\mathcal{M}$ is a set of pairs (r_i, w_i) where each r_i is a component PSDD and w_i is its mixture weight. A mixture of n PSDDs must have $\sum_{i=1}^{n} w_i = 1$. We further assume that all r_i are normalized for the same vtree. A mixture of PSDDs $\mathcal{M}$ represents the probability distribution $\Pr_{\mathcal{M}}(\mathbf{X}) = \sum_{i=1}^{n} w_i \Pr_{r_i}(\mathbf{X})$.

[3]Open-source code and experiments are available at `https://github.com/UCLA-StarAI/LearnPSDD`.



(a) Mixture of two PSDDs with outline of a common vtree.



(b) Equivalent single PSDD and vtree with latent variable L.

Figure 4: Representing ensembles as a single PSDD.

An ensemble of PSDDs is equivalent to a single PSDD with latent variables. More precisely, by adding $\lceil \log(n) \rceil$ Boolean variables $\mathbf{L}$ to the top of the vtree (encoding an n-valued latent component identifier), and mixing between the component PSDDs with an additional decision node, one can capture the distribution $\Pr_{\mathcal{M}}(\mathbf{X})$ in a single PSDD circuit. Figure 4 depicts this reduction.

Because the latent variables $\mathbf{L}$ are not observable, the mixture weights w_i cannot be learned from data in closed form. Instead, we appeal to expectation maximization (EM) for optimizing the likelihood $\mathcal{L}(\mathcal{M} \mid \mathcal{D})$ given dataset $\mathcal{D} = \left\{\mathbf{d}^{(1)}, \mathbf{d}^{(2)}, \ldots, \mathbf{d}^{(M)}\right\}$.

We propose EM-LEARNPSDD, a variant of the (soft) structural EM algorithm (Friedman, 1998), to learn the structure and parameters of ensembles of PSDDs. In soft EM, each example $\mathbf{x}^{(j)}$ takes part in each component (that is, each PSDD r_i) with weight $\alpha_{i,j}$, resulting in weighted datasets $\bar{\mathcal{D}}_i$ for each component. Weights $\alpha_{i,j}$ represent the probability that example $\mathbf{x}^{(j)}$ belongs to distribution r_i, and therefore $\sum_i \alpha_{i,j} = 1$ for all j.

EM-LEARNPSDD consists of two nested learners: an outer EM for structure learning and an inner EM for parameter learning. The outer E-step is the inner learner. The outer M-step uses LEARNPSDD to improve the structure of all PSDD components given the weighted datasets $\bar{\mathcal{D}}_i$. It also updates the component weights as $w_i = \sum_{j=1}^{M} \alpha_{i,j} / \sum_{k=1}^{n} \sum_{j=1}^{M} \alpha_{k,j}$.

The inner E-step redistributes the data over components r_i. For every example $\mathbf{d}^{(j)}$, it updates the weights in each component's weighted dataset $\bar{\mathcal{D}}_i$ as

$$\alpha_{i,j} = \frac{\Pr_{r_i}(\mathbf{d}^{(j)})}{\sum_{k=1}^{n} \Pr_{r_k}(\mathbf{d}^{(j)})}.$$

The inner M-step learns the parameters in r_i from $\bar{\mathcal{D}}_i$ using closed-form estimates (employing a weighted version of Equation 1). Internal EM steps alternate until convergence, or for a maximum number of iterations. We find empirically that a maximum of 3 inner EM iterations is sufficient to improve the parameters and warrants moving to another iteration of the outer EM structure learner.

The initial weighted datasets $\bar{\mathcal{D}}_i$ are found by k-means clustering on $\mathcal{D}$ and softening these clusters by weighting an example in the cluster with $1 - (n-1)\epsilon$ and one not in the cluster with ϵ (default 0.05). K-means clustering empirically provides a better starting point for EM-LEARNPSDD: worlds belonging to the same component distribution tend to be closer in Euclidean distance.

6 RELATED WORK

The sentential decision diagram (SDD) is a tractable representation that is closely related to the PSDD. Despite being a purely logical circuit, one can reduce statistical models to a weighted model counting task on an SDD encoding (Choi et al., 2013). Bekker et al. (2015) learn Markov networks that have a compact SDD for weighted model counting. The learning algorithm uses bottom-up compilation to incrementally add factors to the SDD. It selects features based on a likelihood vs. size trade-off. Adding features is a global modification and requires all parameters to be re-learned by convex optimization.

PSDDs can be reduced to sum-product networks (SPNs), which are a syntactic variation on arithmetic circuits (ACs). A PSDD can be turned into an equivalent SPN by replacing AND nodes by products and OR nodes by sums. Several learning algorithms for SPNs exist. LearnSPN induces an SPT (an SPN tree structure) by splitting on latent variables (Gens and Domingos, 2013). O-SPN and L-SPN improve this algorithm by merging parts of the SPT back into a DAG (Rahman and Gogate, 2016b). Vergari et al. (2015) describe various improvements to reduce overfitting in LearnSPN. SearchSPN shares with LEARNPSDD that it uses local operators (a combination of a type of minimal split on a latent variable with a type of minimal clone) (Dennis and Ventura, 2015).

Probabilistic decision graphs (PDGs) have a variable forest that defines the dependencies between variables, much like vtrees (Jaeger et al., 2006). To induce a variable forest, one learns small PDGs for different forests and chooses the best one. PDG structure learning applies split, merge and redirect operations to the graph in a fixed order, much like L-SPN and O-SPN.

Beyond these, there is a vast literature on tractable learning algorithms that are less related to LEARNPSDD, include ACBN (Lowd and Domingos, 2008), ACMN (Lowd and Rooshenas, 2013), ID-SPN (Rooshenas and Lowd, 2014) and ECNet (Rahman and Gogate, 2016a).

7 EXPERIMENTS

Next, we evaluate the performance of LEARNPSDD and EM-LEARNPSDD, and provide deeper insights into PSDD learning. Section 7.2 evaluates how vtrees affect the learner. Section 7.3 to 7.6 demonstrate that PSDDs are amenable to learning in probability spaces without logical constraints. Lastly, Section 7.7 shows that LEARNPSDD is able to capture a logically constrained probabilistic space while also fitting the data well.

7.1 SETUP

We evaluate our learners on a standard benchmark suite consisting of 20 real-world datasets (Van Haaren and Davis, 2012); see Table 1. These datasets have been used in various previous works for evaluating the performance of assorted tractable model learners (Gens and Domingos, 2013; Lowd and Rooshenas, 2013; Adel et al., 2015; Rooshenas and Lowd, 2014; Rahman and Gogate, 2016a). These datasets do not assume any prior domain knowledge, and are not associated with any logical constraints. Our experiments run for 24 hours or until convergence on the validation set, whichever happens earlier. The experiments run on servers with 16-core 2.6GHz Intel Xeon CPUs and 256GB RAM.

7.2 IMPACT OF VTREES

Because a PSDD's structure is so strongly constrained by the vtree it is normalized for, we would expect vtrees to play a crucial role in determining the size of a PSDDs, and more importantly, the quality of the probability distributions we can learn with the given data and PSDD size available. To evaluate their impact, and the learners described in Section 3, we generate 3 vtrees per dataset: (i) using top-down induction, (ii) using bottom-up induction, and (iii) a balanced vtree with a random variable ordering. We run LEARNPSDD for five hours, with the operation depth parameter m set to 3, using these 3 vtrees. We compare the learned PSDDs in terms of their quality (log-likelihood) as a function of learning time, and their size (number of parameters) as a function of quality.

Figure 5 shows the experimental results for a representative dataset (plants). As expected, bottom-up induction learns superior vtrees, followed by top-down and finally random. The PSDD with a better vtree achieves a higher log-likelihood and is more tractable (smaller). Moreover, a better vtree reduces learning time. In all three cases, LEARNPSDD starts from a trivial initial PSDD. Hence, the log-likelihood is the same for all vtrees at the start of learning. Bottom-up vtree induction is used for the remaining experiments.

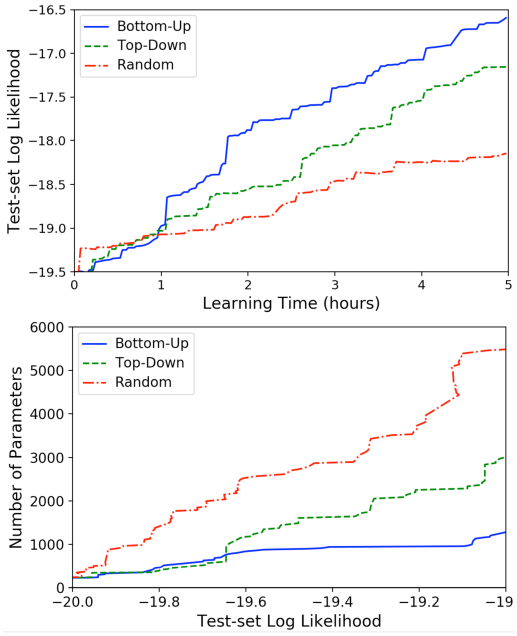


Figure 5: Bottom-up-induced vtrees result in better PSDDs, with higher likelihood and fewer parameters (bottom figure) and are learned in less time (top figure).

7.3 EVALUATION OF LEARNPSDD

As discussed in Section 6, the approach LEARNPSDD takes is closely related to SearchSPN. We therefore evaluate LEARNPSDD's performance in comparison to SearchSPN. The operation copy depth m of LEARNPSDD is fixed to 3. As shown in Table 1, LEARNPSDD achieves a better test-set log-likelihood than SearchSPN (or ties) in 5 datasets while being competitive in most of the other datasets. In general, LEARNPSDD's performance is weaker than SearchSPN. This is expected, because SearchSPN uses many thousands of latent variables, while LEARNPSDD uses none, and PSDDs are necessarily more restrictive than SPNs.

7.4 EVALUATION OF EM-LEARNPSDD

The structural EM algorithm that augments LEARNPSDD essentially decomposes the optimization problems for parameter and structure learning. With a new layer of modeling power, EM-LEARNPSDD is expected to learn more effectively. Therefore, we reduce the depth parameter m to learn smaller PSDD components while retaining a high-quality mixture overall. This experiment uses a combination of minimal operations (80%) and operations with a depth of 3 (20%). Minimal operations get chosen most often, resulting in smaller circuits for the same number of operations. We determine the number of components

Table 1: Comparison among LEARNPSDD, EM-LEARNPSDD, SearchSPN, merged L-SPN and merged O-SPN in terms of performance (log-likelihood) and model size (number of parameters). Sizes for SearchSPN are not reported in the original paper. We use the following notation: (1) LL: Average test-set log-likelihood; (2) Size: Number of parameters in the learned model; (3) † denotes a better LL between LEARNPSDD and SearchSPN; (4) * denotes a better LL between LEARNPSDD and EM-LEARNPSDD; (5) Bold likelihoods denote the best LL among EM-LEARNPSDD, merged L-SPN and merged O-SPN.

Datasets	|Var|	|Train|	|Valid|	|Test|	LearnPSDD		EM-LearnPSDD		SearchSPN	Merged L-SPN		Merged O-SPN	
					LL	Size	LL	Size	LL	LL	Size	LL	Size
NLTCS	16	16181	2157	3236	$-6.03^{\dagger *}$	3170	$\mathbf{-6.03}^*$	2147	-6.07	-6.04	3988	-6.05	1152
MSNBC	17	291326	38843	58265	$-6.05^{\dagger}$	8977	$\mathbf{-6.04}^*$	3891	-6.06	-6.46	2440	-6.08	9478
KDD	64	1800992	19907	34955	$-2.16^{\dagger}$	14974	$\mathbf{-2.12}^*$	9182	$\mathbf{-2.16}$	-2.14	6670	-2.19	16608
Plants	69	17412	2321	3482	-14.93	13129	-13.79^*	13951	$-13.12^{\dagger}$	$\mathbf{-12.69}$	47802	-13.49	36960
Audio	100	15000	2000	3000	-42.53	13765	-41.98^*	9721	$-40.13^{\dagger}$	$\mathbf{-40.02}$	10804	-42.06	6142
Jester	100	9000	1000	4116	-57.67	11322	-53.47^*	7014	$-53.08^{\dagger}$	$\mathbf{-52.97}$	10002	-55.36	4996
Netflix	100	15000	2000	3000	-58.92	10997	-58.41^*	6250	$-56.91^{\dagger}$	$\mathbf{-56.64}$	11604	-58.64	6142
Accidents	111	12758	1700	2551	-34.13	10489	-33.64^*	6752	$-30.02^{\dagger}$	$\mathbf{-30.01}$	13322	-30.83	6846
Retail	135	22041	2938	4408	-11.13	4091	$\mathbf{-10.81}^*$	7251	$-10.97^{\dagger}$	-10.87	2162	-10.95	3158
Pumsb-Star	163	12262	1635	2452	-34.11	10489	-33.67^*	7965	$-28.69^{\dagger}$	$\mathbf{-24.11}$	17604	-24.34	18338
DNA	180	1600	400	1186	-89.11^*	6068	-92.67	14864	$-81.76^{\dagger}$	$\mathbf{-85.51}$	4320	-87.49	1430
Kosarek	190	33375	4450	6675	$-10.99^{\dagger}$	11034	-10.81^*	10179	-11.00	$\mathbf{-10.62}$	5318	-10.98	6712
MSWeb	294	29441	32750	5000	$-10.18^{\dagger}$	11389	-9.97^*	14512	-10.25	$\mathbf{-9.90}$	16484	-10.06	12770
Book	500	8700	1159	1739	-35.90	15197	-34.97^*	11292	$-34.91^{\dagger}$	$\mathbf{-34.76}$	11998	-37.44	11916
EachMovie	500	4524	1002	591	-56.43^*	12483	-58.01	16074	$-53.28^{\dagger}$	$\mathbf{-52.07}$	15998	-58.05	19846
WebKB	839	2803	558	838	-163.42	10033	-161.09^*	18431	$-157.88^{\dagger}$	$\mathbf{-153.55}$	20134	-161.17	10046
Reuters-52	889	6532	1028	1530	-94.94	10585	-89.61^*	9546	$-86.38^{\dagger}$	$\mathbf{-83.90}$	46232	-87.49	28334
20NewsGrp.	910	11293	3764	3764	-161.41	12222	-161.09^*	18431	$-153.63^{\dagger}$	$\mathbf{-154.67}$	43684	-161.46	29016
BBC	1058	1670	225	330	-260.83	10585	$\mathbf{-253.19}^*$	20327	$-252.13^{\dagger}$	-253.45	21160	-260.59	8454
AD	1556	2461	327	491	-30.49^*	9666	-31.78	9521	$-16.97^{\dagger}$	-16.77	49790	$\mathbf{-15.39}$	31070

by conducting a grid search over $\{3, 5, 7, 9\}$ on validation data and report the best result for each datasets. EM-LEARNPSDD surpasses or ties the performance of LEARNPSDD in 17 datasets and it learns smaller models in 13 datasets; see Table 1. EM-LEARNPSDD is superior to LEARNPSDD in 12 datasets by being more accurate and more tractable at the same time.

7.5 COMPARISON WITH SPN LEARNERS

SPNs have been demonstrated to be quite effective for tractable learning in probability spaces that are not subject to logical domain constraints. SPN learners have generated state-of-the-art results in the 20 benchmark datasets (Rooshenas and Lowd, 2014; Rahman and Gogate, 2016b). Specifically, merged L-SPN and O-SPN are the first few SPN structure learners that consider a heuristic merging strategy and therefore produce SPNs that have a significant advantage in size with no loss in performance. In fact, merging shows an improvement in test-set log-likelihood for most datasets (Rahman and Gogate, 2016b). We compare our EM-LEARNPSDD with merged L-SPN and merged O-SPN.

Our experiments show that EM-LEARNPSDD is competitive with merged L-SPN and O-SPN. This result is surprising because PSDDs are much more restrictive than SPNs. EM-LEARNPSDD outperforms O-SPN on likelihood in 11 datasets, learns smaller models in 14 datasets, and wins on both measures in 6 datasets; EM-LEARNPSDD outperforms L-SPN on likelihood in 6 datasets, learns smaller models in 14 datasets and wins on both in 2 datasets. See Table 1 for the full results.

7.6 COMPARISON WITH STATE OF THE ART

In this section, we demonstrate that we can achieve near state-of-the art performance using our EM-LEARNPSDD algorithm. It was shown in previous studies that bagged ensembles with expectation maximization can significantly improve results on many of the 20 datasets (Rahman and Gogate, 2016a,b). We therefore build bagging ensembles on top of EM-LEARNPSDD. The result is still equivalent to a single PSDD, by a translation similar to the one shown in Figure 4 for mixture models, except the w_i for bagging represent a uniform distribution. Our goal with this experiment is to match or exceed the state-of-the-art. This is a very strong baseline, consisting of five competitive tractable model learners: (1) ACMN (Lowd and Rooshenas, 2013), (2) ID-SPN (Rooshenas and Lowd, 2014), (3) SPN-SVD (Adel et al., 2015), (4) ECNet (Rahman and Gogate, 2016a) and (6) Merged L-SPN (Rahman and Gogate, 2016b).

When fixing the number of bags to 10, EM-LEARNPSDD is competitive with the state of the art and surpasses it on 6 out of 20 datasets; see Table 2.

Overall, the experiments outlined so far have incrementally demonstrated that the PSDD structure learning algorithms proposed in this paper (LEARNPSDD and EM-

Table 2: Comparison of test-set log-likelihood between LearnPSDD and the state of the art († denotes best).

Datasets	|Var|	LearnPSDD Ensemble	Best-to-Date
NLTCS	16	$-5.99^{\dagger}$	-6.00
MSNBC	17	$-6.04^{\dagger}$	$-6.04^{\dagger}$
KDD	64	$-2.11^{\dagger}$	-2.12
Plants	69	-13.02	$-11.99^{\dagger}$
Audio	100	-39.94	$-39.49^{\dagger}$
Jester	100	-51.29	$-41.11^{\dagger}$
Netflix	100	$-55.71^{\dagger}$	-55.84
Accidents	111	-30.16	$-24.87^{\dagger}$
Retail	135	$-10.72^{\dagger}$	-10.78
Pumsb-Star	163	-26.12	$-22.40^{\dagger}$
DNA	180	-88.01	$-80.03^{\dagger}$
Kosarek	190	$-10.52^{\dagger}$	-10.54
MSWeb	294	-9.89	$-9.22^{\dagger}$
Book	500	-34.97	$-30.18^{\dagger}$
EachMovie	500	-58.01	$-51.14^{\dagger}$
WebKB	839	-161.09	$-150.10^{\dagger}$
Reuters-52	889	-89.61	$-80.66^{\dagger}$
20NewsGrp.	910	-155.97	$-150.88^{\dagger}$
BBC	1058	-253.19	$-233.26^{\dagger}$
AD	1556	-31.78	$-14.36^{\dagger}$

LEARNPSDD) perform competitively in classical probability spaces without domain constraints. This is despite the fact that PSDDs are more tractable and have more syntactic properties than their alternatives.

7.7 EVALUATION IN A CONSTRAINED SPACE

PSDDs pay for their desirable properties, such as their ability to encode domain knowledge into their base, and ability to answer complex queries, by being a more restrictive representation. The experiments so far do not directly exploit these desirable properties, to allow for a comparison with other tractable learners. They therefore only experience the restrictiveness. However, the next experiments show that in practical domains, and spaces with domain constraints in particular, having these desirable properties can be a great advantage.

Many real-world datasets contain discrete multi-valued data, instead of being only binary. The straightforward way to use general ACs for multi-valued domains, is to introduce a binary variable for each value of the multi-valued variable. Unfortunately, in the learned distribution, it will then be possible for a multi-valued variable to have multiple values simultaneously. PSDDs can easily cope with this by encoding into the base that binary variables belonging to the same multi-valued variable must be mutually exclusive, and at least one must be true.

Table 3: Incorporating domain constraints improves the quality of the learned distributions. Compared settings: (i) unconstrained LEARNPSDD, (ii) constrained PSDD (no LEARNPSDD), and (iii) constrained LEARNPSDD.

Datasets	No Constraint	PSDD	LEARNPSDD
Adult	-18.41	-14.14	-12.86
CovType	-14.39	-8.81	-7.32

To assess the advantage of PSDD in this setting, we compare three learning approaches: (i) LEARNPSDD without domain constraints, (ii) parameter learning on an SDD that is compiled from the constraints (as in prior work, for example Kisa et al. (2014b)), and (iii) applying LEARNPSDD on the initial PSDD obtained from (ii). We use the same vtree in all settings and run LEARNPSDD for 5 hours. We conduct the experiments on two real-world datasets from the UCI repository: Adult and CoverType. Continuous features are discretized into four equal-sized bins. Adult has 14 original (125 binary) variables and CoverType has 12 original (84 binary) variables. Adult and CoverType respectively contain 32,562 and 581,012 examples.

As expected, learning structure on top of the constraints yields the best models. Interestingly, only using the constraints to come up with the SDD structure strongly outperforms unconstrained structure learning, which shows that ignoring constraints complicates learning significantly. The improvement is due to the fact that the probabilities of many impossible assignments (given the multi-valued constraint) are set to 0 and hence the probabilities of the remaining assignments correspondingly increase.

8 CONCLUSIONS

The two questions we raised at the beginning of this paper both receive a strong positive answer. LEARNPSDD is an effective algorithm for learning PSDD structures. It achieves some state-of-the-art results learning classical probability distributions that are not subject to constraints. Moreover, it can just as easily induce structure over logically constrained spaces without losing any domain-specific information.

Acknowledgements

The authors thank Arthur Choi and Yujia Shen for helpful discussions. This research was conducted while JB was a visiting student at UCLA. JB is supported by IWT (SB/141744). This work is partially supported by NSF grants #IIS-1657613, #IIS-1633857 and DARPA XAI grant #N66001-17-2-4032.

References

T. Adel, D. Balduzzi, and A. Ghodsi. Learning the structure of sum-product networks via an SVD-based algorithm. *UAI*, pages 32–41, 2015.

J. Bekker, J. Davis, A. Choi, A. Darwiche, and G. Van den Broeck. Tractable learning for complex probability queries. In *NIPS*, pages 2242–2250, 2015.

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

A. Chechetka and C. Guestrin. Efficient principled learning of thin junction trees. In *NIPS*, pages 273–280, 2007.

A. Choi and A. Darwiche. Dynamic minimization of sentential decision diagrams. In *AAAI*, 2013.

A. Choi, D. Kisa, and A. Darwiche. Compiling probabilistic graphical models using sentential decision diagrams. In *ECSQARU*, pages 121–132, 2013.

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

A. Choi, N. Tavabi, and A. Darwiche. Structured features in naive Bayes classification. In *AAAI*, pages 3233–3240, 2016.

A. Darwiche. A differential approach to inference in Bayesian networks. *JACM*, 50(3):280–305, 2003.

A. Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *IJCAI*, pages 819–826, 2011.

A. Darwiche and P. Marquis. A knowledge compilation map. *JAIR*, 17:229–264, 2002.

A. Darwiche, R. Dechter, A. Choi, V. Gogate, and L. Otten. Results from the probablistic inference evaluation of UAI-08. 2008.

A. Dennis and D. Ventura. Greedy structure search for sum-product networks. *IJCAI*, 2015.

P. Domingos, M. Niepert, and D. L. (Eds.). ICML workshop on learning tractable probabilistic models. 2014.

N. Friedman. The Bayesian structural em algorithm. *UAI*, pages 129–138, 1998.

R. Gens and P. Domingos. Learning the structure of sum-product networks. In *ICML*, pages 873–880, 2013.

M. Jaeger, J. D. Nielsen, and T. Silander. Learning probabilistic decision graphs. *IJAR*, 42(1):84–100, 2006.

G. Karypis. METIS a software package for partitioning graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, version 5.1.0. Technical report, University of Minnesota, 2013.

D. Kisa, G. Van den Broeck, A. Choi, and A. Darwiche. Probabilistic sentential decision diagrams: Learning with massive logical constraints. In *LTPM*, 2014a.

D. Kisa, G. Van den Broeck, A. Choi, and A. Darwiche. Probabilistic sentential decision diagrams. In *KR*, pages 1–10, 2014b.

V. Kolmogorov. Blossom v: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1):43–67, 2009.

D. Lowd and P. Domingos. Learning arithmetic circuits. In *UAI*, pages 383–392, 2008.

D. Lowd and A. Rooshenas. Learning markov networks with arithmetic circuits. In *AISTATS*, pages 406–414, 2013.

N. D. Mauro and A. Vergari. PGM tutorial on learning sum-product networks. 2016.

M. Meila and M. I. Jordan. Learning with mixtures of trees. *JMLR*, 1:1–48, 2000.

C. Meinel and T. Theobald. *Algorithms and Data Structures in VLSI Design: OBDD-foundations and applications*. Springer Science & Business Media, 2012.

M. Narasimhan and J. Bilmes. PAC-learning bounded tree-width graphical models. In *UAI*, 2004.

R. Peharz, R. Gens, and P. Domingos. Learning selective sum-product networks. In *LTPM*, 2014.

H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *UAI*, 2011.

T. Rahman and V. Gogate. Learning ensembles of cutset networks. In *AAAI*, pages 3301–3307, 2016a.

T. Rahman and V. Gogate. Merging strategies for sum-product networks: From trees to graphs. In *UAI*, 2016b.

T. Rahman, P. Kothalkar, and V. Gogate. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of Chow-Liu trees. In *ECML PKDD*, pages 630–645, 2014.

A. Rooshenas and D. Lowd. Learning sum-product networks with direct and indirect variable interactions. In *ICML*, pages 710–718, 2014.

Y. Shen, A. Choi, and A. Darwiche. Tractable operations for arithmetic circuits of probabilistic models. In *NIPS*, 2016.

J. Van Haaren and J. Davis. Markov network structure learning: A randomized feature generation approach. In *AAAI*, pages 1148–1154, 2012.

A. Vergari, N. Di Mauro, and F. Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *ECML PKDD*, pages 343–358, 2015.

Algorithm 3: PartialCopy($q, \mathbf{z}, m, n2c$)

Input: q: node to copy, $\mathbf{z}$: variable assignment, m: depth of copy, $n2c$: map of nodes to copy

Result: constrained copy of q in $n2c$

1 $E = \emptyset$
2 $\mathbf{X}$ *and* $\mathbf{Y}$ *are the partition variables of* q
3 $\mathbf{z}_p = \exists_{\mathbf{Y}} \mathbf{z}$; $\mathbf{z}_s = \exists_{\mathbf{X}} \mathbf{z}$
4 **for** $i \leftarrow 1$ **to** n **do**
5   **if** $\mathbf{z}_p \models [p_i] \wedge \mathbf{z}_s \models [s_i]$ **then**
6         $p' = p_i$; $s' = s_i$
7   **if** $m > 0$ **or** $[p_i] \not\Rightarrow \mathbf{z}_p$ **then**
8   **if** $p_i \notin n2c$ **then**
                PartialCopy($p_i, \mathbf{z}_p, m-1, n2c$)
9             $p' = n2c[p_i]$
10   **if** $m > 0$ **or** $[s_i] \not\Rightarrow \mathbf{z}_s$ **then**
11   **if** $s_i \notin n2c$ **then**
                PartialCopy($s_i, \mathbf{z}_s, m-1, n2c$)
12             $s' = n2c[s_i]$
13         $E = E \cup [(p', s')]$
14 $n2c[q] = $ NewNode(E)

A PARTIAL COPY OF A PSDD

A copy of a node creates a new fold for that node and its descendants up to a specified level (Algorithm 3, lines 8,11). The elements of the copy beyond the specified level redirect to nodes of the original PSDD (line 6). Optionally, the copy can be constrained to a partial assignment for some variables. In this case, only descendants that agree with the assignment are kept in the copy (line 5) and nodes beyond the specified level may have to be copied to enforce the constraint (lines 7,10).

B PROOF OF SYNTACTIC VALIDITY OF OPERATIONS

Definition 1 (Valid PSDD node)**.** *A PSDD node q that is normalized for a vtree node v is valid if: (1) all primes p_i are valid nodes and normalized for the left child of v; (2) all subs s_i are valid nodes and normalized for the right child of v; (3) the primes are mutual exclusive: $\forall i \neq j, [p_i] \wedge [p_j] = \bot$; (4) all elements are satisfiable: $\forall i, [p_i] \wedge [s_i] \neq \bot$.*

A valid operation keeps the PSDD syntactically sound and does not alter the base of the root node.

Lemma 1 (PartialCopy($q, \mathbf{z}, m, n2c$) is valid)**.** *If the following conditions are satisfied: (1) q is valid; (2) $n2c$ is valid (this means that it only contains entries $q \to q'$ where q and q' are normalized for the same vtree, valid and $[q'] = [q] \wedge \mathbf{z}_q$, where $\mathbf{z}_q$ is the projection of the assignment $\mathbf{z}$ to the variables in the vtree of node q); (3) $\mathbf{z}$ only contains variables in the vtree of q and is satisfiable in q: $\mathbf{z} \models [q]$.*

Proof. Proof by induction.

Note the following preconditions hold and we use them in our proof: (1) $n2c$ includes q; (2) $n2c$ is valid.

Base case: $m = 0$ and $[q] \to \mathbf{z}$. In the base case, only one decision node is added according to $n2c$ which is q', the copy of q. Because q and q' have the same elements, q' is valid and $[q] = [q']$. Because $\mathbf{z}$ is implied by $[q]$, $[q'] = [q] \wedge \mathbf{z}_q$.

Induction step: To use the inductive assumption, we first show that the preconditions hold for the calls of PartialCopy. Because q is valid, so are its primes and subs. By induction and precondition, $n2c$ is valid. Finally, $\mathbf{z}_p$ and $\mathbf{z}_s$ only contain the relevant variables because the others are forgotten using existential quantification.

The first postcondition is satisfied because q is added to $n2c$ in line 14. In terms of the second postcondition, we consider 3 cases: (i) The entry is already in $n2c$ when PartialCopy is called, then it is valid because of the precondition. (ii) The entry is added by a recursive call of PartialCopy, then it is valid because of induction. (iii) The entry is $q \to q'$, where q' has an element (p'_i, s'_i) for every element $(p_i, s_i) \in q$, except for those that do not agree with the assignment: $p_i \wedge \mathbf{z}_p = \bot$ or $s_i \wedge \mathbf{z}_s = \bot$. p'_i and s'_i are normalized for the correct vtrees because they either are the original children, or they come from $n2c$ which is valid by the precondition and induction.

We proceed to prove the mutual exclusivity of the copied primes, the satisfiability of the copied elements and the correctness of the base of the copied decision node.

The primes of q' are mutually exclusive:

$$
\begin{aligned}
[p'_i] \wedge [p'_j] &= [p_i] \wedge \mathbf{z}_p \wedge [p_j] \wedge \mathbf{z}_p \\
&= [p_i] \wedge [p_j] \wedge \mathbf{z}_p \\
&= \bot
\end{aligned}
$$

All elements of q' are satisfiable because all the elements of q are satisfiable and elements that would become unsatisfied by conditioning on $\mathbf{z}$ are removed.

The base of q' is the base of q constraint by $\mathbf{z}$:

$$
[q'] = \bigvee_{i \in q : \mathbf{z}_p \models [p_i] \wedge \mathbf{z}_s \models [s_i]} [p'_i] \wedge [s'_i]
$$

$$= \bigvee_{i \in q} [p_i] \wedge \mathbf{z}_p \wedge [s_i] \wedge \mathbf{z}_s$$
$$= \mathbf{z} \wedge \bigvee_{i \in q} [p_i] \wedge [s_i]$$
$$= \mathbf{z} \wedge [q]$$

□

Proposition 4 (Split(q, i, Zs, m) is valid)**.** *If the following conditions are satisfied: (1) q is valid. (2) All $\mathbf{z} \in Zs$ only contain variables of the left children of q's vtree and are satisfiable in the ith element of q: $\mathbf{z} \vDash [p_i] \wedge [s_i]$. (3) All $\mathbf{z} \in Zs$ are mutually exclusive and exhaustive.*

Proof. Note that the following postconditions hold and we use them in our proof: (1) q is valid; (2) the base of q is not altered: $[q] = [q_{\text{old}}]$.

The primes and subs of q are normalized for the correct vtree because q is valid and $n2c$ is valid (Lemma 1).

The primes of q are mutually exclusive if: (i) the original primes are mutually exclusive, (ii) the new primes are mutually exclusive and (iii) every pair of an original prime and a new prime is mutually exclusive.

All the elements of q are satisfiable, because the precondition states that all the assignments must be satisfiable in the split element.

The original base of q is $[q_{\text{old}}] = \bigvee_j [p_j] \wedge [s_j]$. After the split, the base is:

$$[q] = \bigvee_{j \neq i} [p_j] \wedge [s_j] \vee \bigvee_{\mathbf{z} \in Zs} [p_{i,\mathbf{z}}] \wedge [s_i]$$
$$= \bigvee_{j \neq i} [p_j] \wedge [s_j] \vee \bigvee_{\mathbf{z} \in Zs} [p_i] \wedge \mathbf{z} \wedge [s_i]$$
$$= \bigvee_{j \neq i} [p_j] \wedge [s_j] \vee \Big([p_i] \wedge [s_i] \wedge \bigvee_{\mathbf{z} \in Zs} \mathbf{z} \Big)$$
$$= \bigvee_{j \neq i} [p_j] \wedge [s_j] \vee \Big([p_i] \wedge [s_i] \Big)$$
$$= [q_{\text{old}}]$$

□

Proposition 5 (Clone(q, P, m) is valid)**.** *If the following conditions are satisfied: (1) q is valid; (2) $\forall (\pi, i) \in P$, q is either $p_{\pi,i}$ or $s_{\pi,i}$.*

Proof. Note the following postconditions hold and we use them in our proof: (1) $\forall (\pi, i) \in P$, π is valid; (2) $\forall (\pi, i) \in P$, the base of π is not altered: $[\pi] = [\pi_{\text{old}}]$. Because of lemma 1 and the preconditions, q' is a valid node with the same vtree and base as q. Redirecting the parents to this node therefore keeps the parents valid and also remain the base as unaltered. □

C IMPLEMENTATION DETAILS

We discuss implementation details of LEARNPSDD.

Data In The Nodes The training data is explicitly kept in the PSDD nodes during learning. Every node contains a bitset that indicates which examples agree with the context of that node. This speeds up parameter estimation and log-likelihood calculations, which are needed for every execution and simulation of an operation. For simulation of an operation, a bitmask is used to represent the examples that are moved to a copy.

Unique Node Cache To avoid duplicate calculations when doing inference, the PSDD should not have duplicate nodes. This is accomplished using the unique-node technique, where a cache of the nodes is kept and it is checked every time before creating a new node (Meinel and Theobald, 2012). In general, two nodes are considered equal if they have the same (p, s, θ) elements. During learning, however, we adapt this by considering two nodes different if they might evolve to a different structure, based on the training data that it contains. There are two reasons for a node not to change. First, if the node's base is a complete assignment, i.e. if all descendants of this node have only one element, then there are possible LEARNPSDD operations. A clone would be useless in this case because all the parameters would remain as 1. Second, if the node contains no data. Such a node cannot contribute to the log-likelihood and has therefore no reason to change.

The number of added nodes is no longer a local characteristic of an operation, as it depends on the nodes available in the cache. To cope with this, we consider nodes that can be cached as free nodes: they are not counted in the score. This makes sense because if the node is already in the cache, it does not need to be added, otherwise adding it to the cache can make subsequent operations less expensive to simulate or execute.

SDDs In The Nodes SDDs are kept in the nodes to represent their base. This is not really needed, because the base is implied by the structure of the PSDD. However, during structure learning, PSDDs grow bigger, while SDDs do not. Therefore, if the base needs to be checked, doing this on the SDD is more efficient. Note that before any structure learning is done, the SDD is larger than the PSDD because SDD's primes need to be exhaustive and therefore the SDD may have elements for subs that represent false. However, PSDDs are expected to grow larger than the corresponding SDDs during structure learning.