Learning Probabilistic Sentential Decision Diagrams by Sampling

Renato L. Geh[1], Denis D. Mauá[1], Alessandro Antonucci[2]

[1] Department of Computer Science, Institute of Mathematics and Statistics, Universidade de São Paulo, Brazil
{renatolg,ddm}@ime.usp.br
[2] Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, Switzerland
alessandro@idsia.ch

Abstract. Probabilistic circuits are deep probabilistic models with neural-network-like semantics capable of accurately and efficiently answering probabilistic queries without sacrificing expressiveness. Probabilistic Sentential Decision Diagrams (PSDDs) are a subclass of probabilistic circuits able to embed logical constraints to the circuit's structure. In doing so, they obtain extra expressiveness with empirical optimal performance. Despite achieving competitive performance compared to other state-of-the-art competitors, there have been very few attempts at learning PSDDs from a combination of both data and knowledge in the form of logical formulae. Our work investigates sampling random PSDDs consistent with domain knowledge and evaluating against state-of-the-art probabilistic models. We propose a method of sampling that retains important structural constraints on the circuit's graph that guarantee query tractability. Finally, we show that these samples are able to achieve competitive performance even on larger domains.

CCS Concepts: • **Computing methodologies** → *Machine learning*; **Probabilistic reasoning**; • **Theory of computation** → Logic and verification.

Keywords: artificial intelligence, machine learning, probabilistic circuits, probabilistic sentential decision diagrams, propositional logic

1. INTRODUCTION

Probabilistic circuits (PCs) are a class of expressive probabilistic models with operational semantics similar to that of neural networks. Importantly, PCs cover a wide range of tractable inferences, allowing for exact query computation in polytime. For instance, Sum-Product Networks (SPNs) are able to compute conditional marginals in linear time [Poon and Domingos 2011], and Cutset Networks (CNets) admit closed-form parameter estimation [Rahman et al. 2014]. Probabilistic Sentential Decision Diagrams (PSDDs) are a special subclass of PCs capable of efficiently representing combinatorial objects (e.g. hierarchies, rankings, routes, etc) in the form of logical formulae by means of structural restrictions on their circuit [Choi et al. 2015; Kisa et al. 2014; Shen et al. 2019]. Particularly, PSDDs retain linear exact query computation of marginals and closed-form MLE parameter estimation similar to sibling probabilistic circuits, with the added benefit of tractable *maximum-a-posteriori* (MAP) querying. These features make PSDDs attractive estimators capable of not only learning from data, but consistently encoding domain knowledge.

Despite theoretical advantages and empirical competitiveness when compared to other state-of-the-art probabilistic models, there have been very few attempts at learning PSDDs from a combination of data and knowledge [Kisa et al. 2014; Liang et al. 2017]. Recently, [Mattei et al. 2019] have shown promising preliminary work on sampling random structures of PSDDs, though focused on the tautology case. We expand on their work and propose a novel procedure capable of generating random PSDDs consistent with domain knowledge. Furthermore, we show that randomly sampled circuits

This work was supported by CNPq grant 133787/2019-2.

generated by our method are competitive against state-of-the-art PC learners, especially when considering samples as components of an ensemble of PSDDs. We show that taking something as simple as the average of samples is enough for our method to consistently reach competitive performance against other learners.

The article is organised as follows. We first review some necessary background material on vtrees, PSDDs and binary decision diagrams in Section 2. We follow with a description of our PSDD sampling method in Section 3. Experiments and results are discussed in Section 4. We discuss conclusions in Section 5.

2. BACKGROUND

We first review the basic concepts of probabilistic sentential decision diagrams and binary decision diagrams needed for later discussion. We start with some notational preliminaries. We use upper case letters (e.g. X, Y) to denote random variables, and their respective lower case (e.g. x, y) for their instantiations. Boldface upper case letters are used for denoting sets of random variables (e.g. $\mathbf{X}$, $\mathbf{Y}$) with their respective lower case letters meaning their instantiations (e.g. $\mathbf{x}$, $\mathbf{y}$). We denote propositional variables the same way as random variables, and use $\top$ and 1 (and similarly $\bot$ and 0) interchangebly when adequate.

We introduce the notion of decomposition of a formula.

Definition 2.1 $(\mathbf{X}, \mathbf{Y})$*-decomposition.* Let $\phi(\mathbf{x}, \mathbf{y}) = (p_1(\mathbf{x}) \wedge s_1(\mathbf{y})) \vee \ldots \vee (p_k(\mathbf{x}) \wedge s_k(\mathbf{y}))$ a Boolean function over $\mathbf{X} \cup \mathbf{Y}$. Consider a set of tuples $\mathcal{D} = \{(p_i, s_i)\}_{i=1}^{k}$, where p_i and s_i are formulae over $\mathbf{X}$ and $\mathbf{Y}$ respectively, and $\mathbf{X} \cap \mathbf{Y} = \emptyset$. We say that $\mathcal{D}$ is an $(\mathbf{X}, \mathbf{Y})$-decomposition of ϕ if and only if:

$$\phi(\mathbf{x}, \mathbf{y}) = \bigvee_{i=1}^{k} p_i(\mathbf{x}) \wedge s_i(\mathbf{y}),$$

for each $\mathbf{x} \in \{\top, \bot\}^{|\mathbf{X}|}$ and $\mathbf{y} \in \{\top, \bot\}^{|\mathbf{Y}|}$.

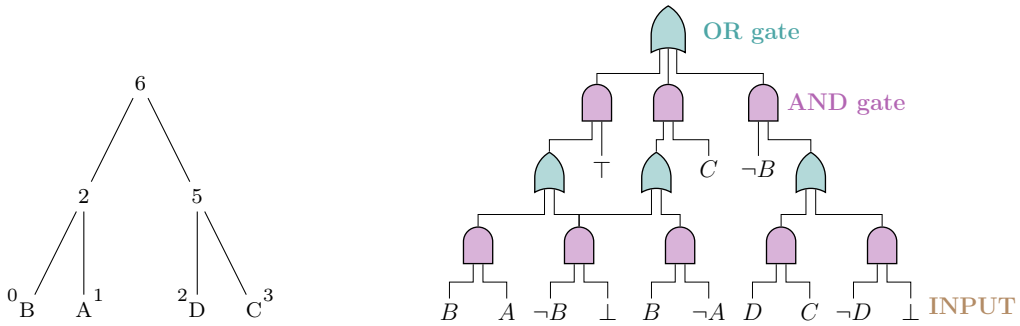
A decomposition is said to be *strongly deterministic* when $p_i(\mathbf{x}) \wedge p_j(\mathbf{y}) = \bot$ and $i \neq j$. When this is true, the pair (p_i, s_i) is called an *element* of the decomposition, and p_i and s_i are called the prime and sub respectively for each $i = 1, \ldots, k$. The size of a decomposition is the number of elements k. When the set of primes of a strongly deterministic decomposition are mutually exclusive and consistent, the decomposition is said to be an $(\mathbf{X}, \mathbf{Y})$-partition. The prime of a partition cannot be a $\bot$, while $\top$ is a prime if and only if the partition has size one.

Definition 2.2 *Compression.* Let $\alpha = \{(p_1, s_1), \ldots, (p_k, s_k)\}$ be an $(\mathbf{X}, \mathbf{Y})$-partition. Partition α is *compressed* if and only if $s_i \neq s_j$ for $i \neq j$.

We may compress an uncompressed partition through the following operation. Let $\alpha = \{(p_1, s_1), \ldots, (p_i, s), \ldots, (p_j, s), \ldots, (p_k, s_k)\}$ be an uncompressed partition. Compress α by removing element (p_j, s) and replacing the i-th element with a disjunction over the primes. The resulting compressed partition is $\{(p_1, s_1), \ldots, (p_i \vee p_j, s), \ldots, (p_k, s_k)\}$.

Definition 2.3 *Vtree.* A vtree over variables $\mathbf{X}$ is a full binary tree whose leaves are in one-to-one correspondence with the variables in $\mathbf{X}$.

A vtree sets a total ordering of variables $\mathbf{X}$. Let v be a vtree node. We use the notation $v^{\rightarrow}$ and $v^{\leftarrow}$ for v's left and right children respectively. The ordering defined by the vtree is obtained by a left-right traversal of the binary tree. The vtree in Figure 1a induces a total order (B, A, D, C). We



(a) A vtree over variables (A, B, C, D).

(b) SDD logic circuit according to vtree on the left.

Fig. 1: SDD (b) for $\phi(A, B, C, D) = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$ and its corresponding vtree (a).

say that the scope of a vtree node is the set of variables mentioned in itself and its descendants. We denote the scope of a vtree node v as $\mathrm{Sc}(v)$.

Probabilistic sentential decision diagrams are probabilistic extensions on Sentential Decision Diagrams (SDDs) [Kisa et al. 2014]. SDDs are compact network representations of propositional knowledge bases [Darwiche 2011].

Definition 2.4 *Sentential decision diagram.* A Sentential Decision Diagram (SDD) is a logic circuit composed of leaf input nodes, and logic gates as inner nodes. An SDD $\mathcal{L}$ respects a vtree $\mathcal{V}$ if and only if, for every node $v \in \mathcal{V}$ and a corresponding node $\mathrm{N} \in \mathcal{L}$:

(1) v is a leaf and $\mathrm{N} \in \{\top, \bot\}$ is a constant terminal node; or
(2) v is a leaf with variable X and $\mathrm{N} \in \{X, \neg X\}$ is a literal terminal node; or
(3) v is an internal node and N is a partition in the form of a disjunction node with $\mathrm{N}(\mathbf{x}) = \bigvee_{\mathrm{C} \in \mathrm{Ch(N)}} p_{\mathrm{C}}(\mathbf{x}) \wedge s_{\mathrm{C}}(\mathbf{x})$, where p_{C} and s_{C} are C's prime and sub that respect vtrees $v^{\leftarrow}$ and $v^{\rightarrow}$ respectively.

An SDD encodes a propositional formula. More specifically, every internal node of an SDD is either an element (p_i, s_i) or an $(\mathbf{X}, \mathbf{Y})$-partition. In practice, elements act as AND logic gates, and partitions as OR gates. Figure 1b shows an SDD encoding the Boolean function $\phi(A, B, C, D) = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$ and respecting the vtree shown on its left. The value of an SDD is computed through a bottom-up traversal of the circuit, substituting literals with their values. The value of SDD $\mathcal{L}$ in Figure 1b is $\mathcal{L}(A = 0, B = 1, C = 1, D = 0) = 1$. An SDD is said to be compressed if and only if all of its partitions are compressed.

Definition 2.5 *Probabilistic sentential decision diagram.* A Probabilistic Sentential Decision Diagram (PSDD) is an SDD with the following parameterizations:

—Terminal $\top$ leaves are replaced with a Bernoulli distribution subject to some parameter;
—Nonnegative and normalized weights are added to outgoing disjunction node edges.

The knowledge *base* of a PSDD is its underlying SDD. The value of a PSDD is computed similarly to SDDs, with the added multiplication of weighted edges: the value of a Bernoulli terminal node is the value of its distribution, and the value of a disjunction node N is given by $\mathrm{N}(\mathbf{x}) = \sum_{\mathrm{C} \in \mathrm{Ch(N)}} w_{\mathrm{N,C}} p_{\mathrm{C}}(\mathbf{x}) s_{\mathrm{C}}(\mathbf{x})$.

In order to efficiently manipulate formulae during circuit sampling, we require an efficient data structure with polynomial time apply and reduce operations. Binary Decision Diagrams (BDDs) are

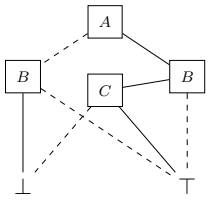


Fig. 2: A BDD for $\phi(A, B, C) = (A \vee \neg B) \wedge (\neg B \vee C)$.

Operation	Description	Complexity
Reduce	canonical form of ϕ	$\mathcal{O}(n \cdot \log n)$
Apply	$\phi_1 \oplus \phi_2$	$\mathcal{O}(n_1 \cdot n_2)$
Restrict	$\phi\|_x$	$\mathcal{O}(n \cdot \log n)$
Forget	$\phi\|_x \vee \phi\|_{\neg x}$	$\mathcal{O}(n^2)$

Table I: Common operations for BDDs.

rooted DAGs capable of canonically representing a Boolean function [Akers 1978; Bryant 1986; Lee 1959].

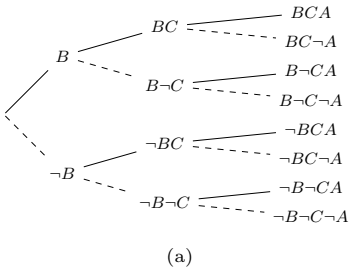
Definition 2.6 *Binary decision diagram.* A Binary Decision Diagram (BDD) is a circuit where leaves are $\bot$ or $\top$ terminals, and inner nodes are variables. Every inner node has two outgoing edges: one for *false* and another for *true* assignments. Let N be an inner node, we denote N's children as N^- and N^+ for the false and true assignment edge destinations respectively. The value of a terminal node is false if $\bot$ and true if $\top$. The value of an inner node representing variable X is the value of its N^- child if $x = 0$, or N^+ if $x = 1$. The value of a BDD is the value of its root.

Figure 2 shows a BDD for formula $\phi(A, B, C) = (A \vee \neg B) \wedge (\neg B \vee C)$. Dashed edges represent false assignments, and solid true assignments. The assignment $\mathbf{x} = \{A = 1, B = 0, C = 1\}$ gives a value of true for the BDD, matching that of ϕ. Table I shows complexity for common operations on BDDs, which are polynomial in the number of vertices n. We make use of this feature to efficiently construct our circuits out of formula decompositions.

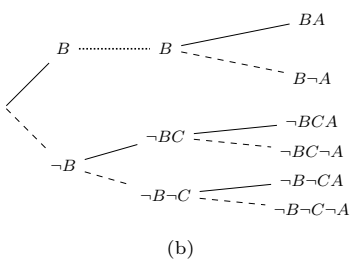
3. SAMPLING PSDDS

Given a formula ϕ and a dataset D, and denote by $\mathbf{X}$ the set of all variables in D and by $\mathbf{Y}$ the set of variables mentioned in ϕ, we wish to sample a PSDD consistent with ϕ. Firstly, we require a vtree to construct the circuit from. A vtree may be easily sampled following the procedure discussed in [Mattei et al. 2019], where the task of sampling vtrees corresponds to sampling matching parentheses. Once a vtree is fixed, we may generate a PSDD by recursively decomposing ϕ in a top-down fashion. This decomposition boils down to generating mutually exclusive and exhaustive primes in order to construct a partition consistent with the original formula. Algorithm 1 gives an overview of the sampling procedure.

At each recursive call of SamplePSDD, a partition of ϕ is generated by Partition, returning a collection of sets of elements grouped by same sub. We propose generating elements in the following way. Let $\mathbf{O}$ be a random variable ordering out of the intersection between the local scope defined



(a)



(b)

Fig. 3: On the left (a), a full assignment tree for ordering $\mathbf{O} = (B, C, A)$. On the right (b), a tree from the same ordering, but marginalizing C on the upper branch. Dashed lines are false assignments, solid lines are true assignments, and dotted lines denote a marginalization branch collapse.

Algorithm 1 SamplePSDD

Input Formula ϕ, vtree node v, marginalization probability p.
Output A sampled PSDD.

Let $L = \text{Sc}(v^{\leftarrow})$ the scope of the left child of v.
$\mathbb{E} \leftarrow$ Partition(ϕ, L, p), where $\mathbb{E}$ is the set of sets of elements with same sub.
Create an OR logic gate S.
for each $\mathbf{E} = \{e_1, e_2, \ldots, e_l\} \in \mathbb{E}$ **do**
    if $|\mathbf{E}| = 1$ **then**
        Let $e = (p, s)$ the only element in $\mathbf{E}$, where p and s are prime and sub.
        C $\leftarrow$ CreateElement(e, v)
        Add C as a child of S.
    else
        Sample a combination of elements $\mathbf{K}$ to compress out of $\mathbf{E}$.
        Compress all elements in $\mathbf{K}$ into a single element $\hat{e} = (\hat{p}, \hat{s})$.
        C $\leftarrow$ CreateElement$(\hat{e}, v)$
        Add C as a child of S.
        for each other element $e = (p, s) \in \mathbf{E}$ such that $e \notin \mathbf{K}$ **do**
            C $\leftarrow$ CreateElement(e, v)
            Add C as a child of S.
Assign random weights to S's children.
return S

by the vtree node and variables mentioned in ϕ. Algorithm 2 induces a binary assignment tree from $\mathbf{O}$, where inner nodes are partial ordered assignments, edges indicate variable values, and leaves are sampled primes. Figure 3a shows an assignment tree for $\mathbf{O} = (B, C, A)$.

Exhaustively constructing every possible world is clearly intractable. Our method exploits polynomial time operations on BDDs to trim an exponential number of partitions: at each assignment tree

Algorithm 2 Partition

Input Formula ϕ, scope $\mathbf{X}$, marginalization probability p.
Output A collection of sets of elements grouped by their sub.

Let $\mathbf{Y}$ be the set of variables mentioned in ϕ.
Let $\mathbf{Z} = \mathbf{X} \cap \mathbf{Y}$.
Let $\mathbf{O}$ be a random variable ordering of $\mathbf{Z}$.
Let $\mathbf{E}$ be a set of elements initially empty.
if $\mathbf{Z} = \emptyset$ **then return** a stochastically sampled partition with marginalization probability p
function RecursivelyPartition(formula ϕ, ordering $\mathbf{O}$, prime π)
    if $\mathbf{O} = \emptyset$ **or** (not first call **and** $\phi \equiv \top$) **then**
        $\mathbf{E} \leftarrow \mathbf{E} \cup (\pi, \phi)$
    else
        Pop X from queue $\mathbf{O}$.
        if $X \notin \phi$ **then**
            RecursivelyPartition$(\phi, \mathbf{O}, \pi)$
        else
            Attribute $\alpha \leftarrow \phi|_x$ and $\beta \leftarrow \phi|_{\neg x}$.
            if $\alpha \not\equiv \bot$ **then** RecursivelyPartition$(\alpha, \mathbf{O}, \pi \wedge x)$
            if $\beta \not\equiv \bot$ **then** RecursivelyPartition$(\beta, \mathbf{O}, \pi \wedge \neg x)$
Call RecursivelyPartition$(\phi, \mathbf{O}, \top)$.
return $\mathbf{E}$ grouped by sub

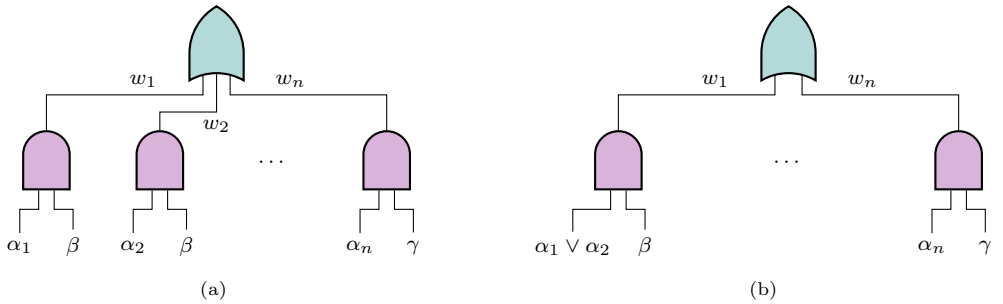


Fig. 4: Compressing two elements (α_1, β) and (α_2, β) in (a) amounts to merging the two AND gates and applying a disjunction over all merged primes, resulting in the compression in (b).

Algorithm 3 CreateElement

Input Element $e = (p, s)$, vtree node v.
Output A conjunction node following element e.
1: Let $L = \mathrm{Sc}(v^{\leftarrow})$ the scope of the left child of v.
2: Let $R = \mathrm{Sc}(v^{\rightarrow})$ the scope of the right child of v.
3: Create an AND logic gate P.
4: **if** $|L| = 1$ **then** create prime π as a literal node consistent with p.
5: **else** $\pi \leftarrow$ SamplePSDD$(p, v^{\leftarrow})$
6: **if** $|R| = 1$ **then** create sub σ as a terminal node consistent with s.
7: **else** $\sigma \leftarrow$ SamplePSDD$(s, v^{\rightarrow})$
8: Add π and σ as prime and sub of P.
9: **return** P

node, we marginalize the corresponding variable x if $\phi|_x \equiv \phi|_{\neg x}$. On a reduced BDD, this is equivalent to checking if X is mentioned in its graph. Figure 3b shows an assignment tree for the same ordering but with variable C marginalized. Once primes have been constructed, we condition ϕ on the assigned values for each prime, resulting in a sub.

A special case arises when $\mathbf{Z} = \emptyset$ in Algorithm 2, which occurs when either $\phi \equiv \top$ or any variable assignment is idempotent in ϕ (i.e. ϕ does not mention any variables to be sampled). In such cases, we may decompose ϕ in terms of $\mathbf{X}$ however we like without altering the original formula, and thus some stochasticity may be added to increase sample variability. Particularly, we may choose to either marginalize or further branch out a variable with some probability p.

The set of generated elements configures a potentially uncompressed partitioning of ϕ. Different PSDDs may be sampled depending on how we compress elements with same sub. We propose a simple sampling procedure to uniformly sample a subspace of possible compressions. Let $\mathbf{E}$ be a set of elements with same sub, and define $n = |\mathbf{E}|$. The number of all possible combinations of n choose k elements, for each $1 \leq k \leq n$, is given by the n-th row of the Pascal Triangle, and thus we may sample a size k, and subsequently a k-sized combination by sampling from the n-th row and then extracting a random combination using Floyd's algorithm in $\mathcal{O}(k)$ time [Bentley and Floyd 1987].

The sampled combination is then compressed, resulting in a single compressed and $n - k$ uncompressed elements. Since elements are conjunction nodes and partitions are disjunctions, a logic circuit is naturally derived from the formula decomposition. Figure 4 shows a PSDD compression. Recursive calls require additional treatment of base cases, which are handled by Algorithm 3. When either $|\mathrm{Sc}(v^{\leftarrow})| = 1$ or $|\mathrm{Sc}(v^{\rightarrow})| = 1$, a leaf node is constructed. In such cases, primes are always literals, and subs take the form of either literals, $\bot$'s or Bernoulli distributions.

Data	#vars	Worst	Best	Average	LSPN-Opt	LSPN-CV	CLT	CNet
1	10	-449.89	**-413.51**	-415.60	-422.37	-444.62	-456.49	-585.23
2	15	-745.20	**-693.51**	-695.82	-695.09	-739.49	-803.41	-1070.29
3	20	-1024.01	-969.51	-971.80	**-959.03**	-1003.93	-1075.31	-1855.05
4	25	-1268.82	-1208.22	-1210.27	**-1185.28**	-1254.47	-1290.94	-2033.58
5	30	-1548.92	**-1440.27**	-1442.58	-1441.90	-1543.35	-1535.54	-2048.16
6	100	-5169.14	-4995.53	-4997.83	**-4958.06**	-5232.04	-5712.73	-10326.73
7	100	-5329.17	-5153.51	-5155.82	**-4900.65**	-5206.54	-5710.64	-9948.88
8	14	-472.15	**-423.32**	-425.62	-490.21	-506.62	-486.06	-601.31

Table II: Log-likelihood values for test dataset. Boldface entries denote best performance. Underlined entries show second best.

Data	Logic formula
1	$\phi_1 = (X_1 \wedge X_3) \vee (X_4 \wedge \neg X_2) \vee (X_5 \wedge \neg X_{10})$
2	$\phi_2 = (X_1 \wedge X_3) \vee (X_4 \wedge \neg X_2) \vee (X_5 \wedge \neg X_{10}) \vee (X_{12} \wedge \neg X_{13} \wedge X_{15} \wedge \neg X_{14})$
3	$\phi_3 = (X_1 \wedge X_3) \vee (X_4 \wedge \neg X_2) \vee (X_5 \wedge \neg X_{10}) \vee (X_{12} \wedge \neg X_{13} \wedge X_{15} \wedge \neg X_{14})$
4	$\phi_4 = (X_1 \wedge X_3) \vee (X_4 \wedge \neg X_2) \vee (X_5 \wedge \neg X_{10}) \vee (X_{12} \wedge \neg X_{13} \wedge X_{15} \wedge \neg X_{14})$
5	$\phi_5 = (X_2 \vee X_{30}) \wedge (\neg X_{15} \vee \neg X_{10}) \wedge (\neg X_{25} \vee X_5 \vee X_{15} \vee \neg X_1) \wedge (X_1 \vee X_{15} \vee \neg X_{30})$
6	$\phi_6 = (X_{10} \vee X_{30}) \wedge (\neg X_1 \vee X_5) \wedge (\neg X_{10} \vee X_{14} \vee X_{23}) \wedge (X_2 \vee \neg X_{27} \vee X_{35}) \wedge (X_{98} \vee \neg X_{78} \vee \neg X_{27} \vee X_8)$
7	$\phi_7 = \top$
8	$\phi_8 = d_0 \vee d_1 \vee d_2 \vee d_3 \vee d_4 \vee d_5 \vee d_6 \vee d_7 \vee d_8 \vee d_9$

Table III: Prior domain knowledge as logical restrictions for each generated dataset.

4. EXPERIMENTS

We evaluated sampled PSDDs on several artificial datasets paired with prior knowledge in the form of logical formulae. Ground-truth PSDDs were sampled from a logical formula and set arbitrary weights, and datasets of size 100 each sampled from the resulting circuits. We generated the structure of 10 PSDDs from randomly sampled vtrees according to the algorithm previously described. Weights were learned through the usual closed-form MLE parameter learning for PSDDs [Kisa et al. 2014]. For each dataset, we evaluated worst, best and average performance of sampled models against several structure learning algorithms for probabilistic circuits. Namely the LearnSPN[1] sum-product network structure learner (LSPN-Opt, LSPN-CV) [Gens and Pedro 2013], the Strudel Chow-Liu Tree algorithm[2] (CLT) [Dang et al. 2020], and a cutset network learner[3] (CNet) [Rahman et al. 2014].

Table II shows log-likelihood performance for sampled PSDDs against other probabilistic circuits. Samples are consistently competitive and even the average of sampled likelihoods is often above the next better learning algorithm, showing that domain knowledge takes an important role, even when structure is random. Table III shows the domain restrictions used to generate the ground-truth distributions. These formulae were passed as arguments to sampled PSDDs. Dataset 8 refers to a digit identification task on a possibly faulty seven-segment digit display [Mattei et al. 2020]. Each segment corresponds to an observable variable representing whether the segment has come on (or off). For each of these segments, an additional latent variable for the intended (or predicted) state of the display is created. Logical restrictions corresponding to each configuration d_i of digit i are then imposed on the latent variables.

Overall, sampled PSDDs have shown to be competitive against other probabilistic circuits. Taking the best sample out of the 10 consistently outperforms most learners, even LearnSPN, which produces less constrained probabilistic circuits. Even when no prior knowledge is expected, as is the case for dataset 7, sampled PSDDs remain competitive. Furthermore, it is worth mentioning that LearnSPN's hyperparameters were optimized in two ways: through a grid search on a k-fold cross

[1]https://gitlab.com/pgm-usp/pyspn
[2]https://github.com/Juice-jl/ProbabilisticCircuits.jl
[3]https://github.com/SPFlow/SPFlow

validation regime (LSPN-CV), and through a more optimistic hyperparameter optimization on the whole test-set (LSPN-Opt). On the other hand, our SamplePSDD algorithm takes no parameters aside from the probability of marginalizing a variable for special cases. Even so, we randomly assigned this probability at each run, to further explore sample space.

5. CONCLUSIONS

We show a simple yet effective algorithm for sampling PSDDs consistent with a propositional logic formula. Sampled PSDDs achieve comparable results against other state-of-the-art probabilistic circuit learners, often outperforming them in log-likelihood. This remains true even when no domain knowledge is given.

REFERENCES

Akers, S. B. Binary decision diagrams. *IEEE Trans. Comput.* 27 (6): 509–516, June, 1978.

Bentley, J. and Floyd, B. Programming pearls: A sample of brilliance. *Commun. ACM* 30 (9): 754–757, Sept., 1987.

Bryant, R. E. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.* 35 (8): 677–691, Aug., 1986.

Choi, A., den Broeck, G. V., and Darwiche, A. Tractable learning for structured probability spaces: A case study in learning preference distributions. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, Q. Yang and M. J. Wooldridge (Eds.). AAAI Press, pp. 2861–2868, 2015.

Dang, M., Vergari, A., and den Broeck, G. V. Strudel: Learning structured-decomposable probabilistic circuits (to appear). *Proceedings of the Tenth International Conference on Probabilistic Graphical Models*, 2020.

Darwiche, A. Sdd: A new canonical representation of propositional knowledge bases. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*. IJCAI'11. AAAI Press, Barcelona, Catalonia, Spain, pp. 819–826, 2011.

Gens, R. and Pedro, D. Learning the structure of sum-product networks. In *Proceedings of the 30th International Conference on Machine Learning*. Proceedings of Machine Learning Research, vol. 28. PMLR, Atlanta, Georgia, USA, pp. 873–880, 2013.

Kisa, D., den Broeck, G. V., Choi, A., and Darwiche, A. Probabilistic sentential decision diagrams. *Knowledge Representation and Reasoning Conference*, 2014.

Lee, C. Y. Representation of switching circuits by binary-decision programs. *The Bell System Technical Journal* 38 (4): 985–999, 1959.

Liang, Y., Bekker, J., and den Broeck, G. V. Learning the structure of probabilistic sentential decision diagrams. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017*, G. Elidan, K. Kersting, and A. T. Ihler (Eds.). AUAI Press, 2017.

Mattei, L., Antonucci, A., Mauá, D. D., Facchini, A., and Villanueva Llerena, J. Tractable inference in credal sentential decision diagrams. *International Journal of Approximate Reasoning* vol. 125, pp. 26 – 48, 2020.

Mattei, L., Soares, D. L., Antonucci, A., Mauá, D. D., and Facchini, A. Exploring the space of probabilistic sentential decision diagrams. In *3rd Workshop of Tractable Probabilistic Modeling*. PMLR, 2019.

Poon, H. and Domingos, P. Sum-product networks: A new deep architecture. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*. UAI'11. AUAI Press, Arlington, Virginia, USA, pp. 337–346, 2011.

Rahman, T., Kothalkar, P., and Gogate, V. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In *Proceedings of the 2014th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part II*. ECMLPKDD'14. Springer-Verlag, Berlin, Heidelberg, pp. 630–645, 2014.

Shen, Y., Goyanka, A., Darwiche, A., and Choi, A. Structured bayesian networks: From inference to learning with routes. *Proceedings of the AAAI Conference on Artificial Intelligence* vol. 33, pp. 7957–7965, 07, 2019.