Structural Learning of Probabilistic Sentential Decision Diagrams under Partial Closed-World Assumption

Alessandro Antonucci[1] **Alessandro Facchini**[*1] **Lilith Mattei**[1]

[1]Istituto Dalle Molle di Studi per l'Intelligenza Artificiale (IDSIA), Lugano, Switzerland,
{alessandro,alessandro.facchini,lilith.mattei}@idsia.ch

Abstract

Probabilistic sentential decision diagrams are a class of structured-decomposable probabilistic circuits especially designed to embed logical constraints. To adapt the classical LEARNSPN scheme to learn the structure of these models, we propose a new scheme based on a partial closed-world assumption: data implicitly provide the logical base of the circuit. Sum nodes are thus learned by recursively clustering batches in the initial data base, while the partitioning of the variables obeys a given input vtree. Preliminary experiments show that the proposed approach might properly fit training data, and generalize well to test data, provided that these remain consistent with the underlying local base, that is a relaxation of the training data base.

1 INTRODUCTION

Probabilistic Circuits (PCs) have been recently introduced as a general computational framework unifying the different existing formalisms for tractable probabilistic modeling [Choi et al., 2020].

Learning the structure of a PC from data is emerging as a crucial challenge for a widespread application of PCs in the area of machine learning. Different algorithms have been (e.g., Lowd and Domingos [2012]) and still are (e.g., Peharz et al. [2020]) designed to improve the state of the art in this field. Most of these algorithms are somehow inspired by the LEARNSPN scheme of Gens and Domingos [2013], the first algorithm for the structural learning of general PCs, originally designed for sum-product networks [Poon and Domingos, 2011], the most popoular class of PCs.

In this position paper, we focus on the problem of learning from (complete) data the structure of a particular class of PCs, called *Probabilistic Sentential Decision Diagrams* (PSDDs, Kisa et al. [2014]. Roughly speaking, a PSDD is a weighted logical circuit inducing a probability distribution assigning non-zero probability only to states consistent with the Boolean formula encoded by the circuit.

As a matter of fact there are only two main algorithms to learn the structure of a PSDD: LEARNPSDD [Liang et al., 2017] and STRUDEL [Dang et al., 2020]. Despite their differences, both these algorithms start from a circuit subject to local transformations. These local actions are designed to preserve the underlying logical base of the circuit, which should therefore be specified when the initial circuit is learned.

The LEARNSPN architecture is different: the algorithm learns a circuit from an input database by growing it from the root, rather than performing local operations on an initial circuit. LEARNSPN takes as input a database and recursively divides it into sub-databases by finding almost independent subsets of variables and clustering similar instances.

Two structural properties of PSDDs prevent a straightforward application of LEARNSPN. First, PSDDs are structured-decomposable, this roughly meaning that the splitting of the variables into independent sets must agree to some constraint. Second, a PSDD encodes a Boolean formula representing a knowledge base (KB); thus it is desirable to steer the operations of LEARNSPN so that the obtained KB represents the input database. Here we address such situation by making LEARNSPN to learn a PSDD encoding a KB describing the data.

Of course there is no unique way to agree on what "describing the data" means. In database theory, a traditional assumption, called *closed-world assumption* (CWA), states that a database reflects with fidelity the concerned state of affairs, that it is complete [Reiter, 1981]. Stated otherwise, a fact that is true in the world is stored in the database, whereas a fact that is missing in the database just does not hold in the world. This assumption is usually followed in ILP (Inductive Logic Programming), a branch of Machine Learning whose

[*]Supported by the Hasler foundation grant n. 20061.

objective is to induce a set hypothesis represented as logic programs (rules). The opposite of the closed-world assumption is the *open-world assumption* [Reiter, 1981, Imieliński and Lipski Jr, 1989]. Commonly employed in the area of description logic [Baader et al., 2003], this simply states that some facts holding in the real world may be missing, and thus does not presume that the knowledge of a domain is complete.

However, in many cases, it may be desirable to have an intermediate perspective: some parts of the observed data may be complete, and some others may be incomplete. First treated by Motro [1989], this view has later been instantiated in various forms under various names (local closed-world assumption, partial closed-world assumption, partial completeness, and others; see, e.g., Galárraga et al. [2013], Darari et al. [2013], Dong et al. [2014])

In this work we follow a similar intermediate perspective. More precisely, we extract a KB from a database $\mathscr{D}$ by recursively applying an instance of a *partial closed-world assumption* (PCWA). That is, each step of the procedure extracts a sub-database $\mathscr{D}'$ and a subset $\boldsymbol{X}'$ of variables for which $\mathscr{D}'$ is assumed to reflect completely the concerned possible states of affairs (given the constraints provided by the previous step). This idea is made precise in Section 3.

The paper is organised as follows. In Section 2 we provide the necessary background about PSDDs. Our contribution is in Section 3, where we present a PCWA based structural learning algorithm for PSDDs inspired by the LEARNSPN schema. In Section 4 we run preliminary experiments. Conclusions and outlooks are in Section 5.

2 BACKGROUND

2.1 BASICS

We focus on a set of Boolean variables $\boldsymbol{X} := (X_1, \ldots, X_n)$. Let $\mathscr{D}$ denote a database of joint states of $\boldsymbol{X}$, to be also called *records* and denoted as $\boldsymbol{x} = (x_1, \ldots, x_n)$. The database obtained by taking only the columns of $\mathscr{D}$ relative to variables in $\boldsymbol{X}' \subseteq \boldsymbol{X}$ is denoted as $\mathscr{D}^{\boldsymbol{X}'}$. When coping with a single variable $X \in \boldsymbol{X}$, with a small abuse of notation, we intend as $\mathscr{D}^{X=1}$ the (sub)set of true instances of $\mathscr{D}^X$, and we analogously intend $\mathscr{D}^{X=0}$. Notation $|\mathscr{D}|$ is used instead to denote the number of records in $\mathscr{D}$. If the records are indexed by the integers $\{1, \ldots, m\}$, with $m := |\mathscr{D}|$, we denote as $\mathscr{D}_J$ the database obtained by taking only the records indexed by the elements of J, with $J \subseteq \{1, \ldots, m\}$. Let $\boldsymbol{x}^j = x_1^j, \ldots, x_n^j$ denote the j-th record of $\mathscr{D}$. According to CWA, we assume the database $\mathscr{D} = \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^m\}$ complete, this meaning that it completely describes the possible state of (the concerned aspects of) the world. Hence, $\mathscr{D}$ induces a Disjunctive Normal Form (DNF)

$$\phi(\mathscr{D}) := \bigvee_{j=1}^{m} \bigwedge_{i=1}^{n} L_i^j \,, \tag{1}$$

whose conjunctive clauses are the ones describing the records belonging to $\mathscr{D}$. That is, for each record $\boldsymbol{x}^j := (x_1^j, \ldots, x_n^j) \in \mathscr{D}$, the corresponding conjunctive clause under the CWA coincides with $L_1^j \wedge \cdots \wedge L_n^j$ where, for $i \leq n$

$$L_i^j = \begin{cases} X_i & \text{when } x_i^j = 1 \,, \\ \neg X_i & \text{otherwise.} \end{cases} \tag{2}$$

2.2 PSDDS

Inspired by Choi et al. [2020], in this section we introduce PSDDs as a class of PCs that, in addition to specific structural properties (e.g., determinism and structured-decomposability), encode Boolean formulae.

2.2.1 Probabilistic Circuits

A PC $\mathscr{C}$ over $\boldsymbol{X}$ is a rooted DAG $\mathscr{G}$ annotated by parameters $\boldsymbol{\theta}$ in order to represent a joint probability mass function $\mathbb{P}(\boldsymbol{X})$. Formally, each leaf (terminal node) g of $\mathscr{G}$ is called an *input unit* and it is associated with a variable $X_g \in \boldsymbol{X}$ and a (possibly degenerate) probability mass function $d_g(X_g)$. Non-leaves nodes can be *sum units* and *product units*. Parameters are associated with both input units and sum units. In case of a sum unit, these can be seen as the probabilities of a mass function weighting its outgoing arcs, while for input units a mass function π is directly provided. In practice, if $\mathsf{ch}(g)$ are the children of a node g (also called its *inputs*) and $\boldsymbol{X}_g$ the set of variables associated to the input units reachable from g, and called its *scope*, the corresponding probability mass function $\mathbb{P}_g(\boldsymbol{X}_g)$ is defined as follows:

$$\mathbb{P}_g(\boldsymbol{x}_g) := \begin{cases} \pi_g(\boldsymbol{x}_g) & \text{if } g \text{ is an input unit,} \\ \prod_{\ell \in \mathsf{ch}(g)} \mathbb{P}_\ell(\boldsymbol{x}_\ell) & \text{if } g \text{ is a product unit,} \\ \sum_{\ell \in \mathsf{ch}(g)} \theta_{\ell,g} \mathbb{P}_\ell(\boldsymbol{x}_\ell) & \text{if } g \text{ is a sum unit.} \end{cases}$$

As we focus here on Boolean variables, we can identify input units with literals or logical constants, sum units with disjunctions, and product units with conjunctions. Graph $\mathscr{G}$ can be consequently seen as a logical circuit encoding a Boolean formula. Both logical and probabilistic elements are considered by the definition of PSDDs provided in the next section.

2.2.2 Defining PSDDs

PSDDs have been originally presented as a probabilistic extension of a class of logical circuits called *sentential decision*

diagrams [Darwiche, 2011]. The extension is obtained by respecting the formula in the underlying circuit. Thus, unlike PCs, the position of the variables in a PSDD is constrained by a meta-structure called *vtree*.

A vtree over $\boldsymbol{X}$ is just a full binary tree whose leaves are in one-to-one correspondence with the variables in $\boldsymbol{X}$ (see, e.g., Figure 1). We intend vtrees as recursive objects and denote them with their root node. In practice, each node u of vtree v is a(sub-)vtree. Notation $\boldsymbol{v}$ is used for the variables of v. Moreover, we denote as v^l and v^r the sub-vtrees of v rooted at its left and right children.

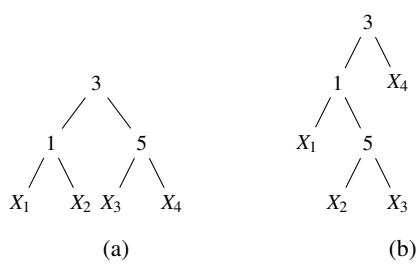

Figure 1: Two vtrees over $\boldsymbol{X} = \{X_1, X_2, X_3, X_4\}$.

Using PCs terminology, we can define a PSDD associated to a vtree v by the properties of its structure and its parameters.

PSDD Structure. PSDD graph $\mathscr{G}$ is defined as follows.

- Input units are associated with the leaves of vtree v, i.e., have scope $\{X\}$ for some $X \in \boldsymbol{v}$. They can be literals (X or $\neg X$) or the logical constant $\top$ (*true*).
- A product unit n has two inputs. Notation $n = (p, s)$ is used, p is called its *prime* and s its *sub*.
- A sum unit n is associated with an internal node u of v, its inputs are product units and are called its *elements*. The notation $n := \{(p_i, s_i)\}_{i=1}^k$ is used. Its primes and subs are either sum or input units. The scope of n and of its elements is $\boldsymbol{u}$. The scope of the primes is $\boldsymbol{u^l}$ and the scope of the subs is $\boldsymbol{u^r}$.
- The root is a sum unit.

PSDD Parameters. Parameters $\boldsymbol{\theta}$ are defined as follows.

- Each input unit $\top$ associated with variable X is annotated with a weight $\theta \in]0, 1[$.
- Each sum unit $\{(p_i, s_i)\}_{i=1}^k$ is annotated with non-negative weights $\theta_1, \ldots, \theta_n$ such that $\Sigma_{i=1}^k \theta_i = 1$.

PSDD Logical Base Each PSDD node n encodes a Boolean formula $\langle n \rangle$ called its (logical) *base* and defined recursively as follows.

- For input units: $\langle X \rangle := X$, $\langle \neg X \rangle := \neg X$, and $\langle \top \rangle := \top$
- If $n = (p, s)$ is a product unit, $\langle n \rangle := \langle p \rangle \wedge \langle s \rangle$

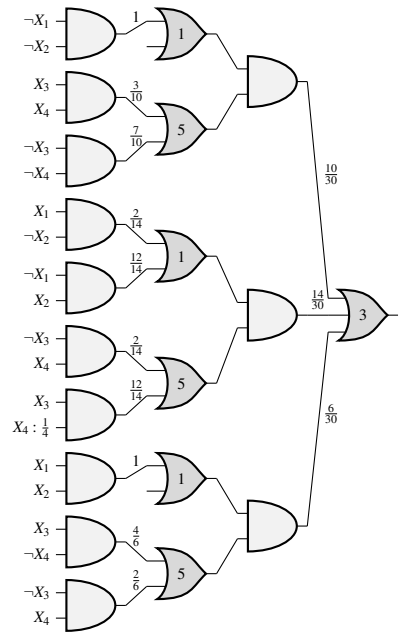

Figure 2: A PSDD over four variables trained from database in Table 1. Its vtree is depicted in Figure 1a.

- If n is a sum unit, $n = \{n_i\}_{i=1}^k$, $\langle n \rangle := \bigvee_{i=1}^k \langle n_i \rangle$.

In fact, product units represent AND gates and sum units represent OR gates. The primes of a sum unit are defined as exclusive, i.e., $\langle p_i \rangle \wedge \langle p_j \rangle \equiv \bot$ for $i \neq j$.

Induced Probability Mass Function. Each unit n induces a probability mass function $\mathbb{P}_n$ over the variables of its scope.

- If n is an input unit with scope $\{X\}$, $\mathbb{P}_n$ is a univariate mass function: $\mathbb{P}_\top(X) := (\theta, 1-\theta)$; $\mathbb{P}_X(X) := (1, 0)$; and $\mathbb{P}_{\neg X}(X) := (0, 1)$.
- Otherwise, let $(\boldsymbol{X}, \boldsymbol{Y})$ be the scope of n, where $\boldsymbol{X}$ are its left variables and $\boldsymbol{Y}$ its right ones.
  - If $n = (p, s)$ is a product unit,
  $$\mathbb{P}_n(\boldsymbol{x}, \boldsymbol{y}) := \mathbb{P}_p(\boldsymbol{x})\mathbb{P}_s(\boldsymbol{y}). \tag{3}$$
  - If $n = \{n_i\}_{i=1}^k$ is a sum unit,
  $$\mathbb{P}(\boldsymbol{x}, \boldsymbol{y}) := \theta_i \mathbb{P}_{n_i}(\boldsymbol{x}, \boldsymbol{y}), \tag{4}$$
  for the unique $1 \leq i \leq k$ such that $\boldsymbol{x} \models \langle p_i \rangle$. Moreover, each input of a sum unit assigns non-zero probability to at least one world.[1]

Let us note that the exclusivity of the primes makes PSDDs a special class of *deterministic* PCs. Indeed, for each decision

[1] This means that, unlike in the original definition, and as done in [Liang et al., 2017], we do not consider elements encoding the false, aka *dead branches*.

node $n = \{(p_i, s_i)\}_{i=1}^{k}$ and for each world $(\boldsymbol{xy})$, there is at most one prime p_i such that $\mathbb{P}_{p_i}(\boldsymbol{x}) > 0$. Moreover, the induced probability mass function of a PSDD factorizes through its variables by following its vtree. This makes PSDDs structured-decomposable PCs.

As a demonstrative example, Figure 2 depicts a PSDD over four variables consistent with the vtree in Figure 1.a and whose parameters have been trained from the database in Table 1 as done in [Kisa et al., 2014]. Labels on the decision nodes denote the vtree nodes for which the sub-PSDD is associated.

#	X_1	X_2	X_3	X_4
3	0	0	1	1
7	0	0	0	0
2	1	0	0	1
3	0	1	1	1
9	0	1	1	0
2	1	1	0	1
4	1	1	1	0
0	1	0	1	1
0	1	0	1	0
0	0	1	0	1

Table 1: A database for the joint states of four Boolean variables. The last three records are unobserved.

2.3 LEARNSPN

To conclude the review of the background material, let us consider the pseudo-code in Algorithm 1 where an adaptation of the classical LEARNSPN scheme for the learning of PCs from a database is depicted [Gens and Domingos, 2013]. This is a recursive procedure, where the database subject to horizontal (line 4) and vertical (line 5) splits. By horizontal split we intend a partition in clusters of similar records based on a clustering algorithm generically denoted as CLUSTER and returning the index sets of the different clusters. A vertical split is intended instead a partition of the variables in the database achieved by some independence test and denoted as PARTITION. When single columns are found (line 1), univariate mass functions are directly obtained from the data. The recursion is achieved by expressing the output mass function as a weighted sum of the mass functions associated to the different clusters (sums in line 8) and expressing them as a product of the mass functions associated with the different groups of variables (products in line 8).

3 LEARNING PSDDS UNDER PCWA

The goal of this section is to show how the LEARNSPN scheme as sketched in Section 2.3 can be modified in order to force the output of Algorithm 1 to be a valid PSDD. Remarkably, this will be shown to correspond to a PCWA approach. Similarly to LEARNSPN, we might obtain a PSDD by recursively perform horizontal and vertical splits of $\mathscr{D}$. Yet, unlike LEARNSPN, the horizontal splits should be driven by the vtree, which should be available as an input.

Algorithm 1 LEARNSPN($\mathscr{D}, \boldsymbol{X}$) (adaptated from Gens and Domingos [2013])
Return a PC given database $\mathscr{D}$ over $\boldsymbol{X}$

1: **if** $|\boldsymbol{X}| = 1$ **then**
2: **return** $\pi(\boldsymbol{X}) \leftarrow \mathscr{D}$
3: **else**
4: $\{J_i\}_{i=1}^{k} \leftarrow$ CLUSTER($\mathscr{D}^{\boldsymbol{X}}$)
5: **for** $i \leftarrow 1, \ldots, k$ **do**
6:   $\{\boldsymbol{X}_i^j\}_{j=1}^{m_i} \leftarrow$ PARTITION($\boldsymbol{X}, \mathscr{D}_{J_i}$)
7: **end for**
8: **return** $\sum_{i=1}^{k} \frac{|\mathscr{D}_{J_i}|}{|\mathscr{D}|} \prod_{j=1}^{m_i}$ LEARNSPN($\mathscr{D}_{J_i}^{\boldsymbol{X}_j^i}, \boldsymbol{X}_j^i$)
9: **end if**

Algorithm 2 SLOPP($\mathscr{D}, v$)
Return a PSDD given database $\mathscr{D}$ and vtree v and over $\boldsymbol{X}$

1: **if** $|\boldsymbol{v}| = 1$ **then**
2: $X \leftarrow$ unique variable in $\boldsymbol{v}$
3: **if** $|\mathscr{D}^{X=1}| = |\mathscr{D}|$ **then**
4:   **return** X
5: **else if** $|\mathscr{D}^{X=0}| = |\mathscr{D}|$ **then**
6:   **return** $\neg X$
7: **else**
8:   **return** $(\top, \frac{|\mathscr{D}^{X=1}|}{|\mathscr{D}|})$
9: **end if**
10: **else**
11: $\{J_i\}_{i=1}^{k} \leftarrow$ CLUSTER($\mathscr{D}^{\boldsymbol{v}^l}$)
12: **for** $i = 1, \ldots, k$ **do**
13:   $p_i \leftarrow$ SLOPP($\mathscr{D}_{J_i}^{\boldsymbol{v}^l}, v^l$)
14:   $s_i \leftarrow$ SLOPP($\mathscr{D}_{J_i}^{\boldsymbol{v}^r}, v^r$)
15: **end for**
16: **return** $\{(p_i, s_i, \frac{|\mathscr{D}_{J_i}|}{|\mathscr{D}|})\}_{i=1}^{k}$
17: **end if**

We call our procedure SLOPP (structural learning of PSDDs under PCWA). Algorithm 2 depicts the SLOPP workflow. This is a recursive procedure, that starts growing the circuit from the top. A clustering algorithm (line 11) is executed on the left columns of the database. The clusters obtained in this way are used to generate the primes (line 13), while the subs are made of the records for the right variables induced by the same clusters (line 14). The probabilities assigned to these product units are proportional to the cardinality of the corresponding clusters (line 16). When we finally obtain databases over single variables (lines 1-10) univariate mass functions or simple literals are specified depending on the frequencies of the two Boolean states in the column.

The only parameters of the algorithm are the number of clusters to be returned (or the criteria to select this number) and the minimum amount of records required to run the clustering algorithm. If the number of records is smaller than this threshold, we return a single cluster with all the records.

As discussed in Section 2, given a database $\mathscr{D}$, CWA allows to describe the possible states of the affairs induced by the database as a DNF $\phi(\mathscr{D})$. The PSDD we learn represents a *relaxation* of $\phi(\mathscr{D})$, i.e, $\phi(\mathscr{D})$ logically implies the formula encoded by the learned PSDD. This is formalised by the following result, whose derivation is detailed in the appendix.

Proposition 1 *Algorithm 2 returns a valid PSDD representing a relaxation of the formula induced by the input database, i.e.,* $\phi(\mathscr{D})$ *logically implies* $\langle \text{SLOPP}(\mathscr{D}, v) \rangle$.

The aforementioned relaxation happens because SLOPP applies a PCWA at each recursive call, as we shall explain with a simple example. Consider Table 1 and vtree v in Figure 1a whose left variables are $\boldsymbol{v}^l = \{X_1, X_2\}$ and whose right variables are $\boldsymbol{v}^r = \{X_3, X_4\}$. The horizontal lines in that table separate the output of a clustering of the records in $\mathscr{D}^{\boldsymbol{v}^l}$, these inducing the primes of the root node, whose corresponding partition $\mathscr{D}^{\boldsymbol{v}^r}$ induces its subs. For instance, the second element is induced by the records corresponding to the third, fourth and fifth line in Table 1. In particular prime p_2 is induced by the sub-table described by the third, fourth and fifth lines restricted to the first two columns (associated to X_1 and X_2), whereas the restriction of those three lines to the last two columns (associated to X_3 and X_4) induces the corresponding sub s_2. By combining the records of these two sub-tables, we see that three new *virtual records*, $[1,0,1,1], [1,0,1,0]$ and $[0,1,0,1]$ arise. These three virtual records have zero counts, as showed in Table 1. This notwithstanding, they are considered to be possible by the PSDD returned by SLOPP. Indeed, this PSDD, depicted in Figure 2, induces a mass function that assigns non-zero probability exactly to both the instances recorded in $\mathscr{D}$ and to the virtual records arising from it. Stated otherwise, these records coincide with the joint states satisfying the Boolean formula encoded by the PSDD that is returned by SLOPP when applied to Table 1.

4 PRELIMINARY EXPERIMENTS

A first implementation of Algorithm 2 has been achieved within the Juice[2] (Julia) library for PCs [Dang et al., 2021]. The code, freely available as a simple Julia notebook,[3] is highly experimental and not optimized for fast performance. For this reason, in this position paper, we discuss the results of a very preliminary validation based only on two small databases as available in Juice and whose features are detailed in Table 2. A deeper analysis based on a larger benchmark is a necessary future work.

Name	$\lvert\mathscr{D}_{\text{train}}\rvert$	$\lvert\mathscr{D}_{\text{test}}\rvert$	$\lvert\boldsymbol{X}\rvert$
Nltcs	16181	3236	16
Plants	17412	3482	69

Table 2: Characteristics of the benchmark databases.

Given a complete database of Boolean variables, SLOPP needs a vtree over those variables. Here we adopt the state-of-the-art techniques proposed by Dang et al. [2021] and based on Chow-Liu trees as included in the (Juice implementation of the) STRUDEL algorithm. The PSDD returned by this algorithm is used to evaluate the performance of our method, together with a fully-factorised model intended to provide a trivial baseline level.

To cluster records we use the k-means algorithm. Here we set the number of clusters k constant (and equal to two and three), while a minimum database size d to create a cluster is also specified (and if this is not the case a single cluster with all the data is used).

As a consequence of Proposition 1, the PSDD returned by SLOPP might be consistent even with records not present in the training set. Yet, it might be possible that testing records not available in the training set would be inconsistent with the PSDD. We discard those records from the test database and we denote as γ their number. This clearly gives an unfair advantage to SLOPP if compared with STRUDEL or with the fully-factorised model, as both these PSDDs encode the tautology (i.e., no KB). Yet, for such a preliminary study, we are interested in evaluating the strength of such advantage in terms of performance or its relevance as shown by γ. Tables 3 and 4 depict the test log-likelihood on the consistent records for different values of the parameters.

k	d	LL_{SL}	LL_{FF}	LL_{ST}	γ
2	20	-22'897	-35'821	**-21'787**	6
2	50	-24'787	-35'832	**-21'803**	5
3	20	**-19'998**	-35'688	-21'629	16
3	50	**-20'325**	-35'710	-21'654	18

Table 3: Test log-likelihood with Nltcs for SLOPP (SL), STRUDEL (ST), and fully factorised models (FF). Different experiments with different number of clusters k and threshold on the minimum cluster size d are reported.

As a positive remark we notice comparable performance levels between SLOPP and STRUDEL and a relatively low number of inconsistent test records. The most critical point recorded during our simulations is the huge size of the

[2] `github.com/Juice-jl`.

[3] `github.com/IDSIA-papers/2021-TPM`.

k	d	LL_{SL}	LL_{FF}	LL_{ST}	γ
2	20	**-37'998**	-138'842	-40'397	579
2	50	-41'284	-139'416	**-40'752**	567

Table 4: Test log-likelihood with Plants. The same setup as in Table 3 is considered.

PSDDs returned by SLOPP: on average these are roughly one order of magnitude larger than the ones produced by STRUDEL. We conjecture that this might be related to the fact that the current implementation of our algorithm is only coping with singly-connected topologies, thus creating unnecessarily large circuits. Forcing the creation of multiple connections in the circuit could lead to smaller circuits and possibly prevent overfitting issues.

5 CONCLUSIONS

A new structural learning algorithm for PSDDs has been presented. The algorithm uses the input data as a knowledge base to be relaxed by a PCWA approach. A very preliminary validation suggests that the algorithm might lead to reliable models to be used in frameworks where (some form of) PCWA is a tenable hypothesis. Besides an extensive experimental analysis, the necessary future work involves the dynamic generation of multiple connections when an element involving the same sub-circuit is created. A relaxation of this idea might lead to even smaller models. A similar approach might be also adopted for the dynamic generation of the model vtree.

PROOFS

Proof of Proposition 1 It is an easy exercise to verify that the output of SLOPP is a valid PSDD.

To see that the formula encoded by $\text{SLOPP}(\mathscr{D}, v)$ is a relaxation of $\phi(\mathscr{D})$, we proceed by induction on the number n of variables in $\mathscr{D}$. If $n = 1$, v is a leaf and $\text{SLOPP}(\mathscr{D}, v)$ is a trivial PSDD consisting in a single unit X, $\neg X$ or $(\top, \frac{|\mathscr{D}^{X=1}|}{|\mathscr{D}|})$. In each case the encoded formula is by definition $\phi(\mathscr{D})$. Assume that the statement holds for a number of variables strictly smaller than a fixed $n > 1$. Then, for a database $\mathscr{D}$ over n variables and associated vtree v, consider $\text{SLOPP}(\mathscr{D}, v) = \{(p_i, s_i, \theta_i)\}_{i=1}^{k}$. For $1 \leq i \leq k$, $p_i = \text{SLOPP}(\mathscr{D}_{J_i}^{\boldsymbol{v^l}}, v^l)$ and $s_i = \text{SLOPP}(\mathscr{D}_{J_i}^{\boldsymbol{v^r}}, v^r)$, where $J_1, \ldots, J_k$ are the clusters obtained from $\mathscr{D}^{\boldsymbol{v^l}}$.

Moreover, $\langle \text{SLOPP}(\mathscr{D}, v) \rangle = \bigvee_{i=1}^{k} \langle p_i \rangle \wedge \langle s_i \rangle$, and by induction hypothesis $\langle p_i \rangle$ and $\langle s_i \rangle$ are relaxations of $\phi(\mathscr{D}_{J_i}^{\boldsymbol{v^l}})$ and $\phi(\mathscr{D}_{J_i}^{\boldsymbol{v^r}})$, respectively.

This means that $\langle p_i \rangle$ implies $\phi(\mathscr{D}_{J_i}^{\boldsymbol{v^l}})$ and $\langle s_i \rangle$ implies $\phi(\mathscr{D}_{J_i}^{\boldsymbol{v^r}})$, and thus we get that $\langle p_i \rangle \wedge \langle s_i \rangle$ implies their conjunction. We can immediately conclude that $\langle \texttt{SLOPP}(\mathscr{D}, v) \rangle$ implies $\bigvee_{i=1}^{k} \phi(\mathscr{D}_{J_i}^{\boldsymbol{v^l}}) \wedge \phi(\mathscr{D}_{J_i}^{\boldsymbol{v^r}})$.

Now, for $1 \leq i \leq k$, both $\phi(\mathscr{D}_{J_i}^{\boldsymbol{v^l}})$ and $\phi(\mathscr{D}_{J_i}^{\boldsymbol{v^r}})$ are DNFs whose conjunctive clauses are induced by the rows of $\mathscr{D}_{J_i}^{\boldsymbol{v^l}}$ and of $\mathscr{D}_{J_i}^{\boldsymbol{v^r}}$, respectively. By taking their conjunction we combine their clauses and clearly get, among others, $\phi(\boldsymbol{x}^i)$, the conjunctive clause induced by the ith record of $\mathscr{D}$. This yields the desired conclusion.

References

Franz Baader, Diego Calvanese, Deborah McGuinness, Peter Patel-Schneider, Daniele Nardi, et al. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. 2020.

Meihua Dang, Antonio Vergari, and Guy Van den Broeck. Strudel: Learning Structured-Decomposable Probabilistic Circuits. In *PGM*, 2020. URL http://starai.cs.ucla.edu/papers/DangPGM20.pdf.

Meihua Dang, Pasha Khosravi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. Juice: A Julia package for logic and probabilistic circuits. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (Demo Track)*, Feb 2021. URL http://starai.cs.ucla.edu/papers/DangAAAI21.pdf.

Fariz Darari, Werner Nutt, Giuseppe Pirro, and Simon Razniewski. Completeness statements about rdf data sources and their use for query answering. In *International Semantic Web Conference*, pages 66–83. Springer, 2013.

Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610, 2014.

Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd international conference on World Wide Web*, pages 413–422, 2013.

Robert Gens and Pedro Domingos. Learning the Structure of Sum-Product Networks. In *International conference on machine learning*, 2013.

Tomasz Imieliński and Witold Lipski Jr. Incomplete information in relational databases. In *Readings in Artificial Intelligence and Databases*, pages 342–360. Elsevier, 1989.

Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2014.

Yitao Liang, Jessa Bekker, and Guy Van den Broeck. Learning the Structure of Probabilistic Sentential Decision Diagrams. In *UAI*, 2017.

Daniel Lowd and Pedro Domingos. Learning arithmetic circuits. *arXiv preprint arXiv:1206.3271*, 2012.

Amihai Motro. Integrity= validity+ completeness. *ACM Transactions on Database Systems (TODS)*, 14(4):480–502, 1989.

Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *International Conference on Machine Learning*, pages 7563–7574. PMLR, 2020.

Hoifung Poon and Pedro Domingos. Sum-Product Networks: A New Deep Architecture. In *ICCV*. IEEE, 2011.

Raymond Reiter. On closed world data bases. In *Readings in artificial intelligence*, pages 119–140. Elsevier, 1981.