Random Probabilistic Circuits

Nicola Di Mauro[1,4] **Gennaro Gala**[1,4] **Marco Iannotta**[1] **Teresa M.A. Basile**[2,3]

[1]Department of Computer Science, University of Bari, Bari, Italy
[2]Department of Physics, University of Bari, Bari, Italy
[3] National Institute for Nuclear Physics (INFN), Bari Division, Bari, Italy
[4] Eindhoven University of Technology, The Netherlands

Abstract

Density estimation could be viewed as a core component in machine learning, since a good estimator could be used to solve many tasks such as classification, regression, and imputing missing values. The main challenge of density estimation is balancing the model expressiveness and its learning and inference complexity. Probabilistic circuits (PCs) model a probability distribution as a computational graph. By imposing specific structural properties on such models many inference tasks become tractable. However, learning PCs usually relies on greedy and time consuming algorithms. In this paper we propose a new unified approach to efficiently learn PCs having several structural properties. We introduce extremely randomized PCs (XPCs), PCs with a random structure. We show their advantage on standard density estimation benchmarks when compared to other density estimators.

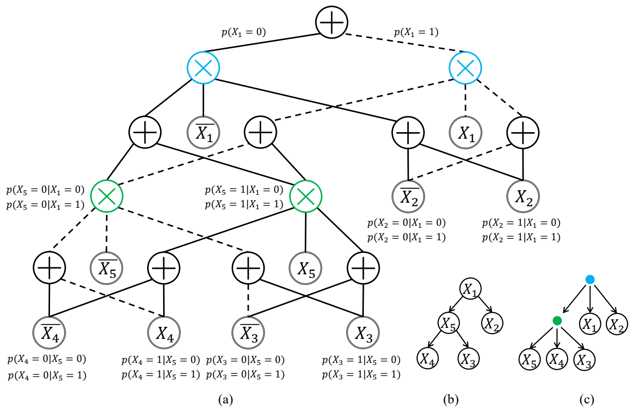

Figure 1: *A PC, its vtree and corresponding CLT*. A smooth, structured-decomposable, and deterministic PC over five RVs $\mathbf{X} = \{X_1, X_2, X_3, X_4, X_5\}$ in (a), where dotted edges denote the *flowing* for the input sample $\{X_1 = 1, X_2 = 0, X_3 = 0, X_4 = 1, X_5 = 0\}$. Its equivalent CLT in (b), and its vtree in (c). Conditional probabilities in (a) represent the weights of the left and right incoming edges.

1 INTRODUCTION

Density estimation is the unsupervised task of reconstructing the joint probability density function (pdf)—learning an estimator—underlying a set of observed samples over specific random variables (RVs). Once such an estimator is learned, it can be used to make *inference*—computing the probability of *queries* about certain RVs states, such as complete evidence queries, marginal queries or conditional queries. A perfect estimator would allow to solve many classical machine learning tasks (e.g., regression, classification, clustering and unsupervised prediction) by casting them in specific types of inference. Indeed, density estimation is recognized as one of the most general and powerful task in Machine Learning (ML).

The main challenge of density estimation is balancing the model representation expressiveness (i.e., the ability to model complex distributions) against its learning and inference complexity. For instance, *Probabilistic Graphical Models* (PGMs), like Bayesian Networks (BNs), can model highly complex probability distributions but exact inference with them is generally intractable—non-polynomial in the size of the network. This is the reason why, a relatively recent probabilistic ML research area focuses on designing and exploiting models that can theoretically guarantee reliable and efficient probabilistic inference. These models belong to the family of *Tractable Probabilistic Models* (TPMs), compact representations for rich probability distributions which allow complex inference routines to be computed exactly and in polynomial time, i.e., *tractably*.

However, it should be noticed that tractability is associated to different classes of queries: computing exact marginals on a TPM may be tractable, while Maximum A Posteriori

(MAP) may not be. TPMs like bounded tree-width graphical models [Meila and Jordan, 2000], Arithmetic Circuits [Darwiche, 2009], Sum-Product Networks (SPNs) [Poon and Domingos, 2011], Cut-Set Networks (CNets) [Rahman et al., 2014] and Probabilistic Sentential Decision Diagrams (PSDDs) [Kisa et al., 2014], promise a good compromise between expressive power and tractable inference. The recently proposed *Probabilistic Circuits* (PCs) framework [Vergari et al., 2020, Choi et al., 2020] allow us to describe, learn, and reason about these TPMs and to categorize them under this single and unified framework.

PCs (see Figure 1) are computational graphs that define a joint probability distribution as recursive mixtures (sum units) and factorizations (product units) of simpler distributions (e.g., parametric distributions such as Gaussians or Bernoullis). Learning PCs can be naturally organized into *structure learning* and *parameter learning*, following the same dichotomy as in PGMs. Differently from other neural density estimators such as NADE [Uria et al., 2016] and autoregressive flows [Papamakarios et al., 2017], PCs enable large classes of tractable inference with little or no compromise in terms of model expressiveness.

However, even if learning TPMs may be done in polynomial time, thanks to several recent algorithmic schemes, making these algorithms scale to high dimensional data is still an issue. In particular, various maximum likelihood based approaches have been proposed using either gradient-based optimization or expectation-maximization (and related schemes), but unfortunately, most structure learners proposed so far are tedious to tune. For instance, tractable SPN learners as LEARNSPN [Gens and Domingos, 2013] and ID-SPN [Rooshenas and Lowd, 2014] spend a substantial amount of their execution time on partitioning RVs into conditionally independent components. On the contrary, Peharz et al. [2019] proposed RAT-SPN, a simple random approach to employ SPNs for deep learning, and demonstrated that tractable models like SPNs can get surprisingly far even without sophisticated structure learning. Also, Di Mauro et al. [2017] showed how mixtures of CNets whose OR tree is learned by performing random conditioning outperform state-of-the-art density estimators on a series of standard benchmark datasets.

Following these trends, we introduce *eXtremely randomized Probabilistic Circuits* (XPCs), smooth and decomposable PCs which can be easily forced to be deterministic and/or structured decomposable without increasing learning complexity. We are proposing a new learning scheme for PCs that can be forced to build networks with specific structural constraints.

XPCs leverage on random conditionings and employ naïve factorizations (*aka* PoB, Product of Bernoullis) and Chow-Liu Trees (CLTs) [Chow and Liu, 1968] as multivariate leaf distributions. To satisfy structured decomposability, we propose a simple yet effective approach to normalize several CLTs for the same vtree. While the log-likelihood of an XPC is slightly worse than other state-of-the-art (SOTA) competitors, ensembles of XPC (EXPCs) perform as SOTA density estimators on a series of standard benchmark datasets, yet employing only a fraction of the time needed to learn the competitors.

2 RELATED WORKS

Randomly learning the structure of PCs is a problem already tackled in some recent works. The first kind of random PCs have been proposed in [Di Mauro et al., 2017] where the authors introduced random CNets, a deterministic PC, whose OR tree is learned by performing random conditioning. Here we do not limit to single variable conditioning, and furthermore our proposed learning scheme is not limited to deterministic circuits as CNets.

In [Peharz et al., 2019] the authors proposed a simple and scalable method to build random and tensorized SPNs (RAT-SPNs), avoiding the necessity of a structure learner. In particular, they first construct a random region graph [Dennis and Ventura, 2012] subsequently populated with arrays of SPN nodes, hence leading to a random hierarchical tensorial decomposition. For the generative case, the parameters of the network are learned using the classical expectation-maximization (EM) algorithm [Dempster et al., 1977] derived for SPNs as in [Peharz et al., 2017]. In this paper the construction of random region graph is data-driven and we avoid the time consuming EM algorithm for parameter optimization. Furthermore, our learning scheme can impose other structural constraints to the obtained PCs than that imposed on SPNs.

Finally, in [Ventola et al., 2019] the authors proposed random sum-product forests (RSPFs), an ensemble approach for mixing multiple randomly generated SPNs, with an approach similar to that proposed in [Di Mauro et al., 2017]. Even in this case the parameters of the networks are learned using the time consuming EM algorithm and the approach is restricted to learn SPNs.

3 PROBABILISTIC CIRCUITS

Notation. Upper-case letters are used to denote random variables (RVs), while lower-case ones for their assignments. Upper-case and lower-case bold letters are used, resp., to denote set of RVs and their joint values. In this paper we consider Boolean RVs, i.e., variables having values in $\{0,1\}$.

3.1 STRUCTURAL PROPERTIES

A probabilistic circuit $\mathcal{C}$, defined over a set of RVs $\mathbf{X}$, is a computational graph, represented with a directed acyclic graph (DAG), encoding a probability distribution $P_{\mathcal{C}}(\mathbf{X})$.

PCs have three kinds of nodes only: input distributions (leaves), product nodes (⊗-node) and sum nodes (⊕-node). Given a node n of the DAG, let $\mathcal{C}_n$ denote the sub-circuit rooted at n, $\mathsf{ch}(n)$ its child nodes, and $P_{\mathcal{C}_n}$ its encoding distribution.

An input distribution n encodes a tractable probability distribution P_n over some RVs $\phi(n) \subseteq \mathbf{X}$. A product node defines a factorized distribution $P_n(\mathbf{X}) = \prod_{c \in ch(n)} P_c(\mathbf{X})$. A sum node n represents a mixture distribution $P_n(\mathbf{X}) = \sum_{c \in ch(n)} \theta_{nc} P_c(\mathbf{X})$. The RVs over which an input distribution n is defined is named its *scope* and it is denoted as $\phi(n)$. The scope of a product or a sum node n is defined as $\phi(n) = \cup_{c \in ch(n)} \phi(c)$. In this paper we consider univariate input distributions.

A PC $\mathcal{C}$ over RVs $\mathbf{X}$ supports computing the likelihood $p_{\mathcal{C}}(\mathbf{x})$ given a complete configuration $\mathbf{x}$ (a complete evidence query) by evaluating the circuit bottom up: starting from the input distributions and computing the output of children before parents. Additional properties of the PC extend the set of probabilistic queries that are guaranteed to be answered exactly and in time linear in the size of the PC—its number of edges.

Definition 3.1. A probabilistic circuit is *smooth* if for every ⊕-node, the children have the same scope.

In a smooth PC ⊕-nodes encode mixtures of distributions defined over the same sets of RVs.

Definition 3.2. A probabilistic circuit is *decomposable* if for every ⊗-node its children have disjoint scope.

In a decomposable PC ⊗-nodes model a factorized probability distributions.

Smooth and decomposable PCs enable the tractable computation of any marginal query [Darwiche, 2000]. In this way, each node in a PC recursively defines a distribution over its scope: a) leaves are distribution by definition, b) ⊕-nodes are mixtures of their child distributions, and c) ⊗-nodes are factorized distributions.

Definition 3.3. A probabilistic circuit is *deterministic* if for every ⊕-node n and assignment $\mathbf{x}$, at most one of the children of n have a non-zero output.

In a deterministic circuit ⊕-nodes define a mixture model whose components have disjoint support, thus enabling tractable MAP inference [Chan and Darwiche, 2006] and closed-form parameter learning [Kisa et al., 2014].

In order to introduce the definition of structured version of PCs it is necessary to give the notion of *vtree*. Even if a vtree has been defined to be a binary tree [Pipatsrisawat and Darwiche, 2008, Kisa et al., 2014], its definition could be extended to general n-ary trees as in the following.

Definition 3.4. A *vtree* over a set of $\mathbf{X}$ RVs is a n-ary tree encoding a hierarchical decomposition of the RVs. Each leaf in a vtree denotes a RV, while an internal node indicates how to decompose a set of RVs into many subsets.

A PC is said to be *normalized* for a vtree if the scope of every ⊗-nodes decomposes over its children as its corresponding node in the vtree (see Figure 1 for an example of a vtree and a corresponding normalized structured decomposable PC. Figure 3, in Appendix A, is another example of structured decomposable PC).

A PC is *structured decomposable* (SD) if its ⊗-nodes cannot decompose in arbitrary ways, but must agree on a vtree. In particular, a structured-decomposable PC encodes a probability distribution in a computational graph by recursively decomposing it into smaller distributions according to a hierarchical partitioning of the random variables, also called vtree. A structured decomposable PC provides several classes of advanced probabilistic queries computable exactly and efficiently [Kisa et al., 2014].

3.2 TRACTABLE INFERENCE

A probabilistic model is tractable when it provides exact inference–answers to queries are not approximated–and the query computation can be obtained efficiently–in time polynomial in the size of the probabilistic model [Choi et al., 2020].

Definition 3.5. A class of queries $\mathbf{Q}$ is *tractable* on a family of probabilistic models $\mathcal{M}$ iff any query $q \in \mathbf{Q}$ on a model $m \in \mathcal{M}$ can be computed in time $\mathcal{O}(\text{poly}(|m|))$.

The concept of efficiency translates to polytime complexity w.r.t. the size of models in a class, $|m|$. For models represented as computational graphs, such as our PCs, model size will directly translate to the number of edges in the graph. The complexity of answering queries in the above definition depends only on the model size and not on its structural properties.

3.3 STRUCTURE LEARNING

Structure learning for PCs corresponds to learning both its structure and parameters approximating the data distribution. For advanced inference task is necessary to require structured decomposability and determinism. Two structure learner for those PCs have been recently proposed: LEARNPSDD [Liang et al., 2017] and STRUDEL [Dang et al., 2020].

LEARNPSDD performs a local search over the space of possible structured-decomposable PCs, given a vtree as input. A first hierarchical clustering is performed over the RVs in order to learn a vtree. Next the local search start from a fully factored PC conforming to a vtree–all RVs are considered to be independent. Each local step changes the circuit while preserving its semantics and structural properties of smoothness, determinism and structured decomposability. Candidates are proposed using two structural transformations–split and clone–to all nodes in the circuits. LEARNPSDD does not use the dependencies discovered during the learning of the vtree, and is very slow making difficult the learning of a mixture model.

In order to overcome these shortcomings, in [Dang et al., 2020] the authors proposed to extract a vtree from the best graphical model that can be learned in tractable time, and then compile it into a structured-decomposable PC as a more informative starting point. The complexity of the learning algorithm has been reduced adopting a greedier local search employing a single transformation, and using circuit *flows* to speed up parameter learning and likelihood computation. The resulting STRUDEL algorithm is a simpler, faster structure learning scheme, enabling learning of large mixtures.

4 EXTREMELY RANDOMIZED CIRCUITS

Here we present our proposed PC structure learning method exploiting a random conditioning approach without optimizing the internal parameters of the circuit, thus leading to a fast learning algorithm.

4.1 CHOW-LIU TREES

A Chow-Liu Tree (CLT), see Figure 1 for an example, is a direct tree-structured Bayesian network minimizing the Kullback-Leibler divergence with the data distribution and supporting linear time marginals and MAP inference. More formally, a CLT is a first-order dependency tree $\mathcal{T}$ over $\mathbf{X} = \{X_i\}_{i=1}^d$ RVs equipped with parameters $\theta_{X_i|X_{\tau(i)}}$ which encodes a factorized probability distribution of the form:

$$q(\mathbf{X}) = \prod_{i=1}^{d} p(X_i|X_{\tau(i)}), \tag{1}$$

where $X_{\tau(i)}$ is the parent of X_i in $\mathcal{T}$ and p is the data probability distribution [1].

A CLT is learned according to the classic Chow-Liu algorithm [Chow and Liu, 1968]: the CLT structure is based on the maximum spanning tree derived by the empirical pairwise mutual information (MI) matrix of $\mathbf{X}$, and the related parameters $\boldsymbol{\theta}$ are estimated from the data. CLTs can be quickly compiled into smooth, deterministic and structured-decomposable PCs [Dang et al., 2020].

[1]Note that, if X_i is the root then $\tau(i) = 0$ and $p(X_i|X_0) = p(X_i)$.

4.2 REGION GRAPH

In this paper we restrict to the class of PCs following a tree-shaped *region graph*. Region graphs can viewed as a vectorised representation of PCs, and have been already used in many SPN learners [Dennis and Ventura, 2012, Peharz et al., 2013, 2019, Trapp et al., 2019].

Definition 4.1. Given a set of RVs $\mathbf{X}$, a *region graph* is a couple $(\mathcal{R}, \phi)$ where $\mathcal{R}$ is a connected DAG containing both *regions* nodes (⊟) and *partitions* nodes (◫). Let $\mathbf{R}$ be the set of all ⊟-nodes and $\mathbf{P}$ be the set of all ◫-nodes. The scope function is defined as a function $\phi : \mathbf{R} \cup \mathbf{P} \to 2^{\mathbf{X}}$, assigning each node in $\mathcal{R}$ a subset of $\mathbf{X}$ ($2^{\mathbf{X}}$ denotes the power set of $\mathbf{X}$). A region graph has the following properties: i) the root R is a ⊟-node and $\phi(R) = \mathbf{X}$; ii) all leaves are ⊟-nodes; iii) $\mathcal{R}$ is bipartite, i.e., all children of ⊟-node are ◫-nodes and vice versa; iv) if Q is either a ⊟-node with children or a ◫-node, then $\phi(Q) = \bigcup_{Q' \in \mathsf{ch}(Q)} \phi(Q')$; v) for a ◫-node P we have $\forall R, R' \in \mathsf{ch}(P) : \phi(R) \cap \phi(R') = \emptyset$; vi) for a ⊟-node R we have $\forall P \in \mathsf{ch}(R) : \phi(R) = \phi(P)$.

Given a region graph $(\mathcal{R}, \phi)$, we can obtain its corresponding PC structure $\mathcal{C}$ as follows. The root ⊟-node in $\mathcal{R}$ is replace with a ⊕-node as a root in $\mathcal{C}$. Each leaf ⊟-node in $\mathcal{R}$ introduces a leaf node in $\mathcal{C}$, while each other ⊟-node corresponds to a ⊕-node in $\mathcal{C}$. For each ◫-node in $\mathcal{R}$, we introduce a ⊗-node in $\mathcal{C}$. All the nodes introduced in $\mathcal{C}$ have the same scope as the nodes in $\mathcal{R}$. It is immediate to observe that such a PC satisfies the smoothness (Definition 3.1) and decomposability (Definition 3.2) structural properties.

4.3 RANDOM REGION GRAPH

Here, we propose a random conditioning approach to build a region graph based on the satisfaction of logical constraints. The construction of a random region graph as been already proposed in [Peharz et al., 2019]. Our approach differs from that since the construction is data driven, thus leading to a fast method for learning both the structure and the parameters of a random PC without relying on parameter optimization. Indeed, while in [Peharz et al., 2019] the graph is randomly defined, here its structure is guided exploiting a random partitioning of the data.

The notation $\mathbf{x} \vDash \Gamma$ denotes that the assignment $\mathbf{x}$ satisfies the logical constraint Γ, and $|\Gamma|$ the length of the constraint. For instance, given the assignment $\mathbf{x} = \{X_1 = 0, X_2 = 1, X_3 = 1, X_4 = 0\}$ and two constraints $\Gamma_1 = \{X_1 = 0 \land X_3 = 1\}$ and $\Gamma_2 = \{X_2 = 0 \land X_3 = 1\}$, then it follows that $\mathbf{x} \vDash \Gamma_1$ and $\mathbf{x} \nvDash \Gamma_2$.

Smooth and decomposable PC. Each region or partition node Q in $\mathcal{R}$ refers to a data slice $\mathcal{S}$ of the entire dataset $\mathcal{D}$ used to learn $\mathcal{R}$, in the following denoted as $Q_{\mathcal{S}}$. The process to build the random region graph is reported in Algorithm 1.

The root region R in $\mathcal{R}$ should be a $\boxminus$-node, and it refers to the entire dataset $\mathcal{D}$ (lines 1-2). The algorithm iteratively extend the graph replacing a leaf region with a sub-graph in the following manner. In order to extend a randomly selected leaf region $R_{\mathcal{S}}$ (line 4), we horizontally split its corresponding data slice $\mathcal{S}$ into non overlapping sub-slices as described in Algorithm 2. In particular, given the $R_{\mathcal{S}}$ region, l RVs $\mathbf{C}$ from its scope $\phi(R_{\mathcal{S}})$ are randomly chosen. Then, k logical constraints Γ_i, among the 2^l possible ones, over the selected $\mathbf{C}$ RVs are chosen at random.

Algorithm 1 RandomRegionGraph($\mathcal{D}$, δ, l, k)

```
Input: a set of samples $\mathcal{D}$ over a set of RVs $\mathbf{X}$, a minimum number of examples per partition $\delta$, a split arity $k$, a conjunction length $l$
Output: a *random* region graph $\mathcal{G}$
 1: $\mathcal{G} \leftarrow$ InsRoot($\boxminus_{\mathcal{D}}$)
 2: $\mathcal{P} \leftarrow \{$Root($\mathcal{G}$)$\}$
 3: while $\mathcal{P} \neq \varnothing$ do
 4:     $P \leftarrow$ a *random* region $\boxminus_{\mathcal{S}}$ from $\mathcal{P}$
 5:     $\mathbf{C} \leftarrow$ a subset of $l$ *random* RVs of $\mathbf{X}$
 6:     $\mathcal{S} \leftarrow$ RandRegions($P, \delta, \mathbf{C}, k$)
 7:     if $\mathcal{S} \neq \varnothing$ then
 8:         for each $\mathcal{Q} \in \mathcal{S}$ do
 9:             AddSub($P, \boxbar_{\mathcal{Q}}$)
10:         for each $\mathcal{Q} \in$ sub($P$) do
11:             AddSub($\mathcal{Q}, \boxminus_{\mathcal{Q}[\mathbf{C}]}$)
12:             AddSub($\mathcal{Q}, \boxminus_{\mathcal{Q}[\mathbf{X}_P - \mathbf{C}]}$)
13:             $\mathcal{P} \leftarrow \mathcal{P} \cup \{\boxminus_{\mathcal{Q}[\mathbf{X}_{\mathcal{P}} - \mathbf{C}]}\}$
14: return $\mathcal{G}$
```

Algorithm 2 RandRegions($\boxminus_{\mathcal{S}}$, δ, $\mathbf{C}$, k)

```
Input: a region $\boxminus_{\mathcal{S}}$ over a data slice $\mathcal{S}$, a min. number of instances per partition $\delta$, a set of RVs $\mathbf{C}$, a split arity $k$
Output: a set of sub-slices $\mathcal{H} \cup \{\mathcal{R}\}$ of $\mathcal{S}$
 1: $\mathcal{R} \leftarrow \mathcal{S}$
 2: $\mathcal{H} \leftarrow \varnothing$
 3: while $|\mathcal{H}| \neq k$ do
 4:     $\Gamma \leftarrow$ a *random* logical constraints over the RVs $\mathbf{C}$
 5:     $\mathcal{Q} \leftarrow \{\mathbf{x} \in \mathcal{R} : \mathbf{x} \models \Gamma\}$
 6:     if $|\mathcal{Q}| \geq \delta$ and $|\mathcal{R} - \mathcal{Q}| \geq \delta$ then
 7:         $\mathcal{H} \leftarrow \mathcal{H} \cup \{\mathcal{Q}\}$
 8:         $\mathcal{R} \leftarrow \mathcal{R} - \mathcal{Q}$
 9: if $\mathcal{H} \neq \varnothing$ then
10:     return $\mathcal{H} \cup \{\mathcal{R}\}$
11: else
12:     return $\varnothing$
```

The k chosen logical constraints Γ_i are used to split the $\boxminus$-node $R_{\mathcal{S}}$ into $k+1$ $\boxbar$-nodes. Each constraint Γ_i, $i = 1, \ldots, k$, leads to a $\boxbar$-node $R'_{\mathcal{S}'_i}$ if at least δ samples $\mathcal{S}'_i$ of the data slice $\mathcal{S}$ satisfy Γ_i. A remaining $\boxbar$-node contains all the samples in $\mathcal{S}$ not satisfying any constraint. The corresponding $\oplus$-node in the circuit to the $\boxminus$-node $R_{\mathcal{S}}$ in $\mathcal{R}$ has $k+1$ branches whose weights are estimated as $|\mathcal{S}'_i| / |\mathcal{S}|$.

Now, each obtained $\boxbar$-node $R'_{\mathcal{S}'_i}$ becomes a child of the $\boxminus$-node $R_{\mathcal{S}}$, and a parent of two new $\boxminus$-nodes (a vertical split of the sub-slice): the first region $R''_{\mathcal{S}'_i[\mathbf{C}]}$ on the sub-slice $\mathcal{S}'_i[\mathbf{C}]$ and the second one $R''_{\mathcal{S}'_i[\phi(R_{\mathcal{S}})-\mathbf{C}]}$ on the remaining variables, where with $\mathcal{S}[\mathbf{C}]$ we denote the selection of the columns $\mathbf{C}$ from the slice $\mathcal{S}$. The first k obtained $R''_{\mathcal{S}'_i[\mathbf{C}]}$ regions are called $\mathcal{Q}$-regions, while the last one is called $\mathcal{R}$-region. All the $R''_{\mathcal{S}'_i[\phi(R_{\mathcal{S}})-\mathbf{C}]}$ regions are called $\mathcal{S}$-regions. During the iterative process, only $\mathcal{S}$-regions are considered for the graph extension.

At the end of the iterative process all the leaf $\boxminus$-nodes corresponding to $\mathcal{Q}$-regions or $\mathcal{R}$-regions are represented in the corresponding circuit as a PoB with Laplace correction, while those corresponding to $\mathcal{S}$-regions are modeled as CLTs with Laplace correction. This process provides a random region graph that leads to a smooth and decomposable PC (see Figure 2, in Appendix A).

The parameters of every sub-circuit are estimated from the data slice corresponding to the related region. Differently from [Peharz et al., 2019], where the graph is randomly constructed, here, however, its construction is guided by the random chosen variables and by the corresponding data partitioning, conditioned on the random logical constraints.

Deterministic PC. In order to impose determinism, given an assignment, the *flow* for every sum node in the circuit must be unique (where with flows we indicate the 'activated' edges in the circuit for an input assignment). Hence, first of all, we remove the Laplace corrections from the sub-circuits corresponding to $\mathcal{Q}$-regions and $\mathcal{S}$-regions. Furthermore, $\mathcal{R}$-regions are modeled as a deterministic PC which evaluates to 0 only for samples having an assignment satisfying the chosen constraints for its sibling regions, thus ensuring determinism for the whole circuit.

Structured Decomposable PC. In order to impose structural decomposability we cannot chose l RVs at random for each region to split since the circuit must be normalized for a vtree (see Section 3.1). Hence, a fixed random ordering $\sigma(\mathbf{X})$ of the RVs is chosen at the beginning of the iterative process and, in Algorithm 1 line 5, the variables are always pushed from that ordering.

However, providing a fixed random ordering $\sigma(\mathbf{X})$ of the RVs is necessary but not sufficient to learn structured decomposable PCs. Indeed, we cannot independently learn the CLTs in each $\mathcal{S}$-region, because there is not guarantee they share the same vtree when learned on their corresponding data slice. Therefore, it is necessary to learn for every $\mathcal{S}$-region a CLT sharing the same structure.

The following theorem proves what is the best first-order dependency tree approximating many distributions.

Theorem 1. *Let $\{p_i\}_{k=1}^n$ be probability distributions over $\mathbf{X} = \{X_i\}_{i=1}^d$ and $\{q_i\}_{k=1}^n$ their corresponding approximations based on a same first-order dependency tree $\mathcal{T}$. It is possible to prove that the first-order dependency tree $\widehat{\mathcal{T}}$ minimizing $\sum_{k=1}^n \mathbb{KL}(p_k \| q_k)$ is:*

$$\widehat{\mathcal{T}} = \arg\max_{\mathcal{T}} \sum_{k=1}^{n} \sum_{i=1}^{d} \mathsf{MI}_k(X_i; X_{\tau(i)}), \qquad (2)$$

where MI_k is the mutual information on p_k and $X_{\tau(i)}$ is the parent of X_i in $\mathcal{T}$.

Theorem 1 proves that the best first-order dependency tree for many $\mathcal{S}$-regions–over the same scope–is based on the maximum spanning tree derived by the matrix obtained by adding the empirical MI computed on each data slice (proof in Appendix B).

However, it may happen to have $\mathcal{S}$-regions over different scopes (see Figure 3, in Appendix A) thus avoiding to straightforwardly apply the Theorem 1. Furthermore, we can note that the conditioning process implicitly imposes a partial structure to the vtree. In particular, let $\mathbb{C} = \{\mathbf{C}_i\}_{i=1}^s$ be the ordered set in which $\mathbf{C}_i$ contains the RVs of the i-th conditioning traversing top-down the region graph. Let $\mathbf{F} = \mathbf{X} - \bigcup_{i=1}^s \mathbf{C}_i$ be the set of variables never involved in a conditioning. Hence, only on those variables $\mathbf{F}$ we can learn the best dependence tree using the result in Theorem 1.

Hence, to complete the vtree we can proceed as follows. For each $\mathcal{S}$-partition, we accumulate the estimated MI just for those variables in $\mathbf{F}$. Let $\mathcal{T}_{\mathbf{C}_1}, \mathcal{T}_{\mathbf{C}_2}, \ldots, \mathcal{T}_{\mathbf{C}_s}$ be the dependence trees resp. over the variables in $\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_s$, and $\mathcal{T}_{\mathbf{F}}$ that over $\mathbf{F}$. We build the complete dependence tree over $\mathbf{X}$ joining the dependence trees such that, the root of $\mathcal{T}_{\mathbf{C}_{i+1}}$ becomes a child of $\mathcal{T}_{\mathbf{C}_i}$ for $i = 1, \ldots, s-1$, and the root of $\mathcal{T}_{\mathbf{F}}$ becomes a child of $\mathcal{T}_{\mathbf{C}_s}$. Now, each CLT for a $\mathcal{S}$-region is compiled using the sub-tree in the vtree associated to the variables in its scope.

4.4 MIXTURES OF XPCS

Recently, a lot of attention has been devoted to learn mixtures of PCs to greatly improve their performance as density estimators, encoding a distribution as the following:

$$m(\mathbf{X}) = \sum_{i=1}^{k} w_i q_i(\mathbf{X}),$$

where k is the number of components $q_i(\mathbf{X})$ and w_i are the weights such that $\sum_{i=1}^k w_i = 1$.

Putting many XPCs in a mixture results in a non-deterministic circuit, that we call EXPC, since it introduces a sum node over many XPC marginalizing a latent variable [Peharz et al., 2017]—all the branches of this sum node are 'activated' for an input assignment.

An ensemble of structured decomposable XPCs remains not structured decomposable unless all of them are normalized for the same vtree [Dang et al., 2020], a constraint very difficult to fulfill in general. However, in our learning scheme we propose to just learn a structured decomposable mixture using the same ordering $\sigma(\mathbf{X})$ for each involved XPC.

Therefore, while non-structured decomposable EXPCs can rely on higher randomness, structured decomposable ones are highly dependent on such a fixed ordering, and their randomness only lies in choosing the logical constraints for horizontal splits. Moreover, since this ordering is shared among all XPCs, it directly affects the ensemble performance.

We noticed that choosing the ordering $\sigma(\mathbf{X})$ at random provides poor results. Instead, we propose the following greedy procedure. The algorithm iteratively selects blocks of l variables from $\mathbf{X}$. A first variable is selected having the highest MI w.r.t. the other ones; the remaining $l-1$ ones are those having the highest MI w.r.t. the first one. After the selection of the block of l variables, the process continues selecting the next block on the remaining variables. In this way the conditioning is done on strong conditionally dependent variables thus providing better results. All the experiments for structured decomposable PCs use this procedure for selecting the variable ordering.

It is possible to note that vertical splits are automatically introduced due to the random conditioning proposed approach. Furthermore, the proposed heuristic for variable ordering (and consequently for a vtree structure) imposes a correlation among the variables involved in the logical constraints, thus providing effective product nodes.

The weights for each component in the mixture are simply set to be uniform among the ensemble, i.e., $w_i = 1/k$ for each $i = 1, \ldots, k$. Even if this is not the optimal choice, however in this way we avoid to optimize them using time consuming procedures like EM.

5 RESULTS

In this section we empirically evaluate our proposed approach[2] [Gala, 2021]. We evaluate our learner on 20 real-world benchmark datasets [Haaren and Davis, 2012] (reported in Table 3, Appendix C), already used to evaluate different tractable density estimators. In particular, we aim to answer the following research questions: **Q1**) how much accurate are single XPCs when compared to other density estimators? And what is the effect of increasing the length of the constraints on XPCs. **Q2**) What is the learning time and circuit size obtained with the proposed approach. **Q3**) Are ensemble of XPCs competitive to those learned using

[2]`https://github.com/gengala/Random-Probabilistic-Circuits`.

other approaches?

In our grid searches some hyper-parameters have been tuned, while keeping fixed to 0.01, for computational reasons, the smoothing parameter α used in the CLTs and PoBs construction for the Laplace correction. We stopped to grow the region graph as soon as its leaves are more than 200. Experiments have been executed on a 8-core Intel Xeon CPU E5-1620 v3 @ 3.50GHz with 16 GB RAM.

Let us denote a deterministic XPC as XPC_{Det} and a structured decomposable XPC as XPC^{SD}.

5.1 Q1) SINGLE MODELS

We evaluate different XPCs and we investigate how they perform by varying the values for l (the length of the logical constraint) and k (the number of constraints for the horizontal splits) in order to asses whether the proposed random region graph method works. In particular, we are interested in XPCs with $l > 1$, since XPCs with $l = 1$ are basically CNets, extensively studied in literature.

Due to the randomness of XPCs, we report the results averaged on 40 different runs for each kind of model in the following grid search space: $l = \{1,2,3\}$, $k = \{2,3,4,8\}$ and $\delta = \{16,32,64,128,256,512\}$[3] (the minimum number of samples per slice for splitting).

The results of such a grid search (reported in Table 4, Appendix D), shows that XPCs with $l > 1$ obtain better results, i.e., just 33 XPCs with $l = 1$ out of 120 are marked as best ones, proving that conditioning on more than one variable at the same time is more effective. The results of XPC wins over XPC_{Det} 15 out of 20 times, and XPC^{SD} wins 13 out of 20 over $\text{XPC}^{\text{SD}}_{\text{Det}}$. This obviously reflects the higher expressiveness of non-deterministic PCs over deterministic ones. Furthermore, XPC^{SD} wins 15 out of 20 over XPC and $\text{XPC}^{\text{SD}}_{\text{Det}}$ wins 15 out of 20 over XPC_{Det}, even though structured decomposable PCs are more restrictive than non-structured decomposable ones, since their structure have to fulfill a vtree. However, as reported in Section 4.4, this could be explained by the adopted greedy heuristic used to select the variables on which is chosen the conditioning, and thus providing a correct conditional probability estimation. A random vtree does not represent the conditional dependencies among the variables the greedy heuristic instead provides.

Table 1 compares, in the first two columns, the results of a single XPC_{Det}, averaged by 40 runs, against those of another deterministic circuit such a XCNet, as reported in [Di Mauro et al., 2017]. The results show how the two models rank the same. In the remaining columns we evaluate $\text{XPC}^{\text{SD}}_{\text{Det}}$ sharing the same structural constraints of the two competitors LEARNPSDD [Liang et al., 2017] and STRUDEL, whose values are taken from [Dang et al., 2020]. $\text{XPC}^{\text{SD}}_{\text{Det}}$ log-likelihoods are in line with those of its competitors despite its random structure and the extremely fast learning time.

[3]Configurations with $k \notin [2, 2^l]$ are discarded.

5.2 Q2) LEARNING TIME AND CIRCUIT SIZE

XPCs learning time depends mainly on learning CLTs and regardless of whether or not we force structural decomposability and/or determinism. More specifically, under the same region graph, every XPC type requires the same learning time. In terms of circuit sizes, deterministic XPCs need a greater number of edges in order to model $\mathcal{R}$-partitions, but nevertheless the size increase is negligible for small values of l. In general, learning times and circuit sizes are obviously inversely proportional to δ given that smaller δ values generate deeper region graphs and then bigger circuits.

In Table 5, Appendix D, reports the average circuit size, the average training time (in seconds) and the δ value of the 40 best models (bold ones) of Table 4. The statistics are quite stable unless few outliers such as the XPC^{SD} over the AD dataset.

Table 2 reports the learning times (in seconds) and circuit sizes for $\text{XPC}^{\text{SD}}_{\text{Det}}$ (averaged over 40 runs) and STRUDEL best models, both executed on the same machine for execution time comparison. As reported in [Dang et al., 2020] the learning time and circuit size have been already compared to those obtained with LEARNPSDD showing the superiority of STRUDEL. STRUDEL models are larger PCs than $\text{XPC}^{\text{SD}}_{\text{Det}}$ thus leading to efficient inference in our case. Furthermore, the learning time required to learn $\text{XPC}^{\text{SD}}_{\text{Det}}$ models is only a tiny fraction of the time required by the competitor so as to extremely speed up learning. To the best of our knowledge, XPCs are the fastest and accurate TPMs to learn among those available in literature.

5.3 Q3) MIXTURE MODELS

To investigate the performance of ensembles of XPCs (EXPCs) we run a grid search in the space formed by $l = \{1,2,3\}$, $k = \{2,3,4,8\}$, $\delta = \{16, 32, 64, 128, 256, 512\}$ and $M = \{2, 5, 10, 15, 20, 25, 30, 40\}$ for every EXPC type, where M is the number of components in the mixture. Learning times and circuit sizes of EXPCs scale linearly in M. Next, we compare EXPCs to other state-of-the-art ensembling techniques and much more sophisticated single models such as ID-SPN.

Table 6, in Appendix E, shows that most of best test-set log-likelihoods is associated to EXPCs having $l > 1$ and this, once again, justifies the constraint-based approach. In particular, just 13 EXPCs having $l = 1$ out of 80 (i.e., 16.25%) are marked as best ones. By comparing non-rounded test-set log-

Table 1: Average test-set log-likelihoods for XCNet, XPC_{Det}, $\text{XPC}^{\text{SD}}_{\text{Det}}$, LEARNPSDD and STRUDEL.

DATASET	XPC_{Det}	XCNet	$\text{XPC}^{\text{SD}}_{\text{Det}}$	LEARNPSDD	STRUDEL
NLTCS	-6.10	**-6.06**	-6.09	**-6.03**	-6.06
MSNBC	-6.18	**-6.09**	-6.21	**-6.04**	-6.05
KDD	-2.22	**-2.19**	-2.20	**-2.17**	**-2.17**
PLANTS	-13.96	**-13.43**	-14.59	**-13.49**	-13.72
AUDIO	**-42.65**	-42.66	-41.97	**-41.51**	-42.26
JESTER	**-56.00**	-56.10	-54.94	**-54.63**	-55.30
NETFLIX	-59.28	**-59.21**	-58.73	**-58.53**	-58.68
ACCIDENTS	-31.88	**-31.58**	-31.03	**-28.29**	-29.46
RETAIL	**-10.95**	-11.44	-10.98	-10.92	**-10.90**
PUMSB-STAR	-25.90	**-25.55**	-26.56	-25.40	**-25.28**
DNA	-87.75	**-87.67**	-87.46	**-83.02**	-87.10
KOSAREK	**-11.29**	-11.70	-10.99	-10.99	**-10.98**
MSWEB	**-10.19**	-10.47	-10.12	**-9.93**	-10.19
BOOK	**-37.51**	-42.36	-36.83	-36.06	**-35.77**
EACHMOVIE	-62.62	**-60.71**	-59.99	**-55.41**	-59.47
WEBKB	**-163.33**	-167.45	**-161.26**	-161.42	-160.50
ROUTERS-52	**-94.29**	-99.52	**-88.92**	-93.30	-92.38
20NEWS-GRP	**-163.95**	-172.60	**-159.37**	-160.43	-160.77
BBC	-261.96	**-261.79**	-260.62	-260.24	**-258.96**
AD	**-16.40**	-18.70	**-16.39**	-20.13	-16.52
AVG. RANK	1.5	1.5	2.45	1.55	2

Table 2: Learning times (in seconds) and circuit sizes for $\text{XPC}^{\text{SD}}_{\text{Det}}$ (averaged over 40 runs) and STRUDEL best models.

DATASET	$\text{XPC}^{\text{SD}}_{\text{Det}}$	STRUDEL	$\text{XPC}^{\text{SD}}_{\text{Det}}$	STRUDEL
NLTCS	0.05	172.5	4401	13827
MSNBC	0.31	3182.7	4887	35629
KDD	1.02	532.7	13040	16984
PLANTS	0.1	10805.6	13960	454141
AUDIO	0.17	930.9	29317	87090
JESTER	0.11	748.5	20273	78342
NETFLIX	0.21	530.1	39868	45328
ACCIDENTS	0.09	2213.3	11921	99277
RETAIL	0.09	50.3	6651	6609
PUMSB-STAR	0.12	216.8	8866	17395
DNA	0.02	211.2	2616	24106
KOSAREK	0.3	657.4	20938	73086
MSWEB	0.27	224.5	12135	3177
BOOK	0.38	1589.9	13678	140351
EACHMOVIE	0.5	10686.1	21369	988953
WEBKB	0.87	1631.0	17122	84299
ROUTERS-52	1.52	3875.6	36440	198905
20NEWS-GRP	3.34	4425.4	65881	204983
BBC	0.77	317.3	14578	22962
AD	1.69	113.2	22093	18151

likelihoods, EXPC wins over EXPC_{Det} 12 out of 20 times, and EXPC^{SD} wins 14/20 over $\text{EXPC}^{\text{SD}}_{\text{Det}}$. Therefore, regardless of whether we use deterministic or non-deterministic components, EXPCs do not show a significant difference in log-likelihood. In fact, an ensemble can be expressive enough even employing deterministic components. Finally, non-structured decomposable EXPC wins over structured decomposable ones 19 out of 20 times, the only exception occurs for ROUTERS-52. This proves the limited expressiveness of structured decomposable circuits compared to non-structured decomposable ones.

Table 7, in Appendix E, reports the log-likelihoods for best EXPCs, SOTA ensembling techniques and single model competitors. Among mixture models, EXPCs provide the best average rank; while, when we include single models competitors, they provide the second best average rank. However, it should be noticed that EXPCs are obviously bigger PCs than single model competitors but nevertheless can be learned employing only a fraction of the time.

Table 3 reports the log-likelihoods of structured decomposable EXPCs and their natural competitor STRUDEL, both executed on the same machine. As we can see our proposed approach ranks better than STRUDEL, thus providing its validity.

Table 3: Average test-set log-likelihoods for structured decomposable mixture models.

DATASET	EXPC^{SD}	$\text{EXPC}^{\text{SD}}_{\text{Det}}$	STRUDEL
NLTCS	**-6.05**	**-6.05**	-6.08
MSNBC	-6.18	-6.17	**-6.04**
KDD	**-2.15**	-2.16	-2.16
PLANTS	-14.19	-14.21	**-13.73**
AUDIO	**-40.91**	-40.97	-41.48
JESTER	**-53.43**	-53.51	-55.04
NETFLIX	**-57.58**	-57.69	-58.25
ACCIDENTS	-31.02	-30.99	**-29.07**
RETAIL	-10.94	**-10.90**	**-10.90**
PUMSB-STAR	-26.06	-26.05	**-24.17**
DNA	-86.61	**-85.09**	-87.21
KOSAREK	**-10.77**	-10.81	-10.89
MSWEB	-9.93	-9.94	**-9.76**
BOOK	**-34.75**	-35.14	-35.89
EACHMOVIE	**-54.82**	-55.26	-55.76
WEBKB	**-153.67**	-154.23	-159.85
ROUTERS-52	**-84.70**	-85.09	-90.12
20NEWS-GRP	**-153.75**	-154.21	-158.79
BBC	**-248.34**	-248.79	-257.40
AD	-15.50	-15.59	**-15.39**
AVG. RANK	**1.6**	2	2.25

6 CONCLUSIONS

We introduced XPCs, simple and customizable probabilistic circuits based on random logical constraints-based conditioning. XPCs are smooth and decomposable by default but can be easily forced to be deterministic and/or structured decomposable without increasing the learning complexity. When learned in ensembles, XPCs achieve the state-of-the-art results for density estimation on several benchmark datasets. Due to their simplicity to implement, fast learning times and accurate inference performances XPCs are promising tractable density estimators.

Many lines of work are possible for XPCs. Firstly, we plan to employ greedy orderings also for non-structured decomposable circuits just to improve their log-likelihoods as single models. Secondly, we plan to extend XPCs for continuous RVs and to employ a DAG rather than a tree structure to partition the data. Thirdly, we plan to investigate how EXPCs perform using bagged datasets, as done by mixtures of LEARNPSDD like in [Dang et al., 2020]. Finally, due to their fast learning, we plan to investigate how sophisticated learners as LEARNPSDD and STRUDEL perform if we provide XPCs as initial PCs.

Author Contributions

N. Di Mauro and G. Gala had an equal contribution.

Acknowledgements

The authors would like to thank Antonio Vergari for inspiring discussions and valuable feedback.

References

Hei Chan and Adnan Darwiche. On the robustness of most probable explanations. In *UAI*, 2006.

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

C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Trans. Inf. Theory*, 14(3):462–467, 1968.

Meihua Dang, Antonio Vergari, and Guy Van den Broeck. Strudel: Learning structured-decomposable probabilistic circuits. In *PGM*, 2020.

Adnan Darwiche. A differential approach to inference in bayesian networks. In Craig Boutilier and Moisés Goldszmidt, editors, *UAI*, pages 123–132, 2000.

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

A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B*, 1977.

Aaron W. Dennis and Dan Ventura. Learning the architecture of sum-product networks using clustering on variables. In *NIPS*, pages 2042–2050, 2012.

Nicola Di Mauro, Antonio Vergari, Teresa Maria Altomare Basile, and Floriana Esposito. Fast and accurate density estimation with extremely randomized cutset networks. In *ECML*, pages 203–219, 2017.

Gennaro Gala. gengala/Random-Probabilistic-Circuits: First release, May 2021. URL `https://doi.org/10.5281/zenodo.4775258`.

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

Jan Van Haaren and Jesse Davis. Markov network structure learning: A randomized feature generation approach. In *AAAI*, 2012.

Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *ICPKRR*, 2014.

Yitao Liang, Jessa Bekker, and Guy Van den Broeck. Learning the structure of probabilistic sentential decision diagrams. In *UAI*, 2017.

Marina Meila and Michael I. Jordan. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1:1–48, 2000.

George Papamakarios, Iain Murray, and Theo Pavlakou. Masked autoregressive flow for density estimation. In *NIPS*, pages 2338–2347, 2017.

Robert Peharz, Bernhard C. Geiger, and Franz Pernkopf. Greedy part-wise learning of sum-product networks. In *ECML/PKDD*, pages 612–627, 2013.

Robert Peharz, Robert Gens, Franz Pernkopf, and Pedro M. Domingos. On the latent variable interpretation in sum-product networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(10):2030–2044, 2017.

Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Xiaoting Shao, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *UAI*, volume 115, pages 334–344, 2019.

Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In *AAAI*, pages 517–522, 2008.

Hoifung Poon and Pedro M. Domingos. Sum-product networks: A new deep architecture. In *UAI*, 2011.

Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In *ECML/PKDD*, pages 630–645, 2014.

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

Martin Trapp, Robert Peharz, Hong Ge, Franz Pernkopf, and Zoubin Ghahramani. Bayesian learning of sum-product networks. In *NIPS*, pages 6344–6355, 2019.

Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *Journal of Machine Learning Research*, 17:205:1–205:37, 2016.

Fabrizio Ventola, Karl Stelzner, Alejandro Molina, and Kristian Kersting. Random sum-product forests with residual links. *CoRR*, abs/1908.03250, 2019.

Antonio Vergari, YooJung Choi, Robert Peharz, and Guy Van den Broeck. Probabilistic circuits: Representations, inference, learning and applications. In *Tutorial at the The 34th AAAI Conference on Artificial Intelligence*, 2020.