# Scalable Learning of Probabilistic Circuits

Renato Lui Geh

Thesis presented to the
Institute of Mathematics and Statistics
of the University of São Paulo
in partial fulfillment
of the requirements
for the degree of
Master of Science

Program: Computer Science
Advisor: Professor Denis Deratani Mauá

This work was supported by CNPq grant #133787/2019-2,
CAPES grant #88887.339583/2019-00 and EPECLIN FM-USP.

São Paulo
November 1, 2021

# Scalable Learning of Probabilistic Circuits

Renato Lui Geh

This is the original version of the thesis prepared by candidate Renato Lui Geh, as submitted to the Examining Committee.

# Acknowledgements

# Abstract

Renato Lui Geh. **Scalable Learning of Probabilistic Circuits**. Thesis (Master's). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2021.

The rising popularity of generative models together with the growing need for flexible and exact inferences has motivated the machine learning community to look for expressive yet tractable probabilistic models. Probabilistic circuits (PCs) are a family of tractable probabilistic models capable of answering a wide range of queries exactly and in polynomial time. Their operational syntax in the form of a computational graph and their principled probabilistic semantics allow their parameters to be estimated by the highly scalable and efficient optimization techniques used in deep learning. Importantly, tractability is tightly linked to constraints on their underlying graph: by enforcing certain structural assumptions, queries like marginals, *maximum a posteriori* or entropy become linear time computable while still retaining great expressivity. While inference is usually straightforward, learning PCs that both obey the needed structural restrictions and exploit their expressive power has proven a challenge. Current state-of-the-art structure learning algorithms for PCs can be roughly divided into three main categories. Most learning algorithms seek to generate a usually tree-shaped circuit from recursive decompositions on data, often through clustering and costly statistical (in)dependence tests, which can become prohibitive in higher dimensional data. Alternatively, other approaches involve constructing an intricate network by growing an initial circuit through structural preserving iterative methods. Besides depending on a sufficiently expressive initial structure, these can possibly take several minutes per iteration and many iterations until visible improvement. Lastly, other approaches involve randomly generating a probabilistic circuit by some criterion. Although usually less performant compared to other methods, random PCs are orders of magnitude more time efficient. With this in mind, this dissertation aims to propose fast and scalable random structure learning algorithms for PCs from two different standpoints: from a logical point of view, we efficiently construct a highly structured binary PC that takes certain knowledge in the form of logical constraints and scalably translate them into a probabilistic circuit; from the viewpoint of data guided structure search, we propose hierarchically building PCs from random hyperplanes. We empirically show that either approach is competitive against state-of-the-art methods of the same class, and that their performance can be further boosted by simple ensemble strategies.

**Keywords:** Probabilistic circuits. Machine learning. Probabilistic models.

# Resumo

Renato Lui Geh. **Aprendizado Escalável de Circuitos Probabilísticos**. Dissertação (Mestrado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2021.

**Palavras-chave:** Circuitos probabilísticos. Aprendizado de máquina. Modelos probabilísticos.

# Nomenclature

## List of Symbols

| | |
|---|---|
| $\sigma$ | Sigmoid function |

## List of Figures

# List of Tables

# List of Algorithms



# Notation

We use the following notation throughout the work. Random variables are written in upper case (e.g. $X$, $Y$) and their values in lower case (e.g. $x$, $y$). We write $\mathcal{X}$ as the sample space and for independence we use $\perp\!\!\!\perp$ to indicate that two variables $X$ and $Y$ are statistically independent, i.e. $X \perp\!\!\!\perp Y$. We identify propositional variables with 0/1-valued random variables, and use them interchangeably. Sets of variables and their joint values are written in boldface (e.g. $\mathbf{X}$, $\mathbf{x}$). Given a Boolean formula $f$, we write $\langle f \rangle$ to denote its semantics, i.e. the Boolean function represented by $f$. For Boolean formulas $f$ and $g$, we write $f \equiv g$ if they are logically equivalent, that is, if $\langle f \rangle = \langle g \rangle$; we abuse notation and write $\phi \equiv f$ to indicate that $\phi = \langle f \rangle$ for a Boolean function $\phi$. We use the notation $[a..b]$, with $b \geq a$ to denote the integer set $\{a, a+1, \ldots, b\} \subset \mathbb{Z}$. Similarly, we use $[b]$ as an equivalent for $[1..b]$. We denote the Iverson bracket as $[\![\phi]\!]$, i.e. a function that returns 1 if $\phi$ is true and 0 otherwise. In the context of graph theory, we use sans serif letters for graph nodes (e.g. $\mathsf{N}$, $\mathsf{S}$, $\mathsf{P}$, $\mathsf{L}$) and bold variants for sets of nodes (e.g. $\mathbf{N}$, $\mathbf{S}$, $\mathbf{P}$, $\mathbf{L}$). We call $\mathrm{Ch}(\mathsf{N})$ the set of all children of a node $\mathsf{N}$, $\mathrm{Pa}(\mathsf{N})$ as the set of all parents, and $\mathrm{Desc}(\mathsf{N})$ the set of all descendants.

# Contents

# Chapter 1

# Introduction

When reasoning about the world, rarely can we find a realistic model that perfectly subsumes all of the needed relationships for flawless prediction. As such, the presence of a reliable uncertainty quantifier in intelligent systems is essential in developing performant yet diagnostible agents. This is made explicitly clear in the case of high-risk settings, such as autonomous vehicles or automated power plant systems, where a wrong prediction could cause disastrous consequences. A well known example in the case of the former is obstacle avoidance: while the agent should be capable of accurately identifying obstructions in its way during normal conditions, so should it be able to identify its own lack of confidence in high uncertainty situations like ones brought by environmental factors, such as severe blizzard or heavy rain. In these situations where predictions are highly unreliable, the safest option might be for the agent to first identify its uncertainty, and second to reach out for human help. Other interesting applications of uncertainty quantification include out-of-distribution detection, which in our previous example could be visualized as the agent identifying a human's driving as abnormally irregular (due to inebriation, infirmity, etc.) and appropriately taking control of the vehicle before a potential accident takes place.

One popular approach to quantifying uncertainty is through probability theory. By abstracting the world as a probability distribution with a finite number of observable random variables that encode a possibly incomplete knowledge of the environment, we are, in theory, able to answer a diverse set of complex queries as long as we have access to the (approximate) true data distribution. Practice is far different from theory, however, as most machine learning models either lie within a very limited range in the tractability spectrum in terms of inference (Vergari, Choi, Peharz, *et al.*, 2020) or are too simplistic for complex real-world problems. Besides, although the majority of recent advances in deep learning claim some probabilistic meaning from the model's output, they are often uncalibrated distributions, a result of focusing on maximizing predictive accuracy at the expense of predictive uncertainty (Guo *et al.*, 2017; Ovadia *et al.*, 2019; Chernikova *et al.*, 2019), ultimately producing overconfident and peculiar results (Szegedy *et al.*, 2013; Wei *et al.*, 2018; Su *et al.*, 2019; Chernikova *et al.*, 2019). Crucially, mainstream deep models (i.e. standard neural networks) usually optimize a conditional distribution over the to-be-predicted random variables – and are thus often called *discriminative* models – and

do not model the actual joint distribution of the data, limiting inference capabilities and uncertainty estimation.

In contrast, *generative* models seek to extract information from the joint (in varying capacities), and have lately seen a sharp rise in interest within deep learning. Despite this, most popular models do not admit either exact or tractable querying of key inference scenarios. For instance, although Generative Adversarial Networks (GANs) allow for efficient sampling (Goodfellow *et al.*, 2014), basic queries such as likelihood or marginals are outside of their capabilities. Similarly, Normalizing Flows (NF) also permit access to efficient sampling, with the added feature of computing likelihoods, but are severely limited by the base distribution when it comes to discrete data (Rezende and Mohamed, 2015; Papamakarios *et al.*, 2021) albeit recent works on discretizing NFs have shown empirically good results (Lippe and Gavves, 2021; Ziegler and Rush, 2019). Variational Auto-Encoders (VAEs) are (under certain conditions) a generalization of NFs (Yu, 2020; Gritsenko *et al.*, 2019) with known extensions for categorical data (Rolfe, 2017; Vahdat, Macready, *et al.*, 2018; Vahdat, Andriyash, *et al.*, 2018), but only permit access to sampling and an upper-bound on the likelihood, with the latter available only after solving a complex optimization task (Kingma and Welling, 2014).

Despite the impressive achievements of the aforementioned generative models on realistically producing samples consistent with evidence, in none of the previous models are complex queries like structured prediction under partial observations, *maximum a posteriori* (MAP), conditional or marginal probabilities tractable. An obvious alternative would be Probabilistic Graphical Models (PGMs), although they too suffer from intractability when dealing with high treewidth networks (R. Dechter, 1998; Koller and Friedman, 2009), severely limiting expressivity. Instead, we draw our attention to an expressive class of models that subsumes several families of probabilistic models with tractable inference capabilities.

Probabilistic Circuits (PCs) define a superclass of probabilistic models distincly specified by recursive compositions of distributions through graphical formalisms. Vaguely speaking, PCs are computational graphs akin to neural networks, but whose network structure and computational units abide by special constraints. Within these specific conditions span a wide range of subclasses, each establishing a distinct set of restrictions on their structure in order to enable different segments within the tractability spectrum. As an example, Sum-Product Networks (SPNs, Poon and P. Domingos, 2011) are usually loosely defined over a couple of constraints: namely *smoothness* and *decomposability*, which in turn enables likelihood, marginal and conditional computations. Arithmetic Circuits (ACs, Darwiche, 2003) add *determinism* to the mix, allowing for tractable computation of MAP probabilities. Similarly, Cutset Networks (CNets, Rahman *et al.*, 2014) employ the same constraints as ACs, but accept more expressive distributions as part of their computational units. Probabilistic Sentential Decision Diagrams (PSDDs, Kisa *et al.*, 2014), Probabilistic Decision Graphs (PDGs, Jaeger, 2004) and And/Or-Graphs (AOGs, Rina Dechter and Mateescu, 2007) all require *smoothness* and *determinism*, but also call for a stronger version of *decomposability*, permitting all queries previously mentioned as well as computation of the Kullback-Leibler divergence and expectation between two circuits (Choi *et al.*, 2020). Usually, PCs represent the joint distribution of the data, although they are sufficiently expressive for generative *and* discriminative modeling (Khosravi *et al.*, 2019; Rashwan

*et al.*, 2018; Rooshenas and Lowd, 2016; Gens and P. Domingos, 2012; Shao *et al.*, 2020). In this dissertation though, we shall focus on the generative side of PCs.

While inference is usually straightforward, as we shall see in ??, learning the structure of PCs so that they obey the needed structural restrictions requires either careful handcrafted architectures (Poon and P. Domingos, 2011; Cheng *et al.*, 2014; Nath and P. M. Domingos, 2016) or usually involves running costly (in)dependence tests over most (if not all) variables (Gens and P. Domingos, 2013; Jaini, Ghose, *et al.*, 2018; Vergari, Mauro, *et al.*, 2015; Di Mauro *et al.*, 2017), which can become prohibitive in higher dimension data. Alternatively, some learning algorithms resort to structure preserving iterative methods to grow a PC that already initially satisfies desired constraints, adding complexity to the underlying distribution at each iteration (Liang, Bekker, *et al.*, 2017; Dang *et al.*, 2020). However, these can take several iterations until visible improvement and often take several minutes for each iteration when the circuit is big. Common techniques used in deep learning for generating scalable architectures for neural network also pose a problem, as the nature of the needed structural constraints make for sparse computational graphs. To circumvent these issues, work on scaling PCs to higher dimensions has focused mainly on random architectures, with competitive results (Peharz, Vergari, *et al.*, 2020; Mauro *et al.*, 2021; Geh and Denis Deratani Mauá, 2021; Peharz, Lang, *et al.*, 2020). Apart from the scalability side of random structure generation, usual structure learning algorithms often require grid-search for hyperparameter tuning to achieve top quality performance, which is usually not the case for random algorithms. For the usual data scientist or machine learning practicioner, hyperparameter tuning can become exhaustive, especially if the goal is to analyze and infer from large data, and not to achieve top tier performance on benchmark datasets.

In this dissertation, we propose two scalable structural learning algorithms for probabilistic circuits that are especially suited for large data and fast deployment. They both take advantage of random network generation to quickly construct PCs with little to no need for hyperparameters. The first is effective for constructing PCs from binary data with a highly constrained structure, and thus appropriate when complex querying is needed. The second builds less constrained random PCs, but supports both discrete and continuous data.

## 1.1 Contributions and Dissertation Outline

We organize this dissertation as follows. We begin Chapter 2 by formally defining probabilistic circuits, conducting a review of some of the structural constraints that we might impose on PCs, as well as what we may gain from them in terms of tractability. We then list existing formalisms that may be viewed as instances of PCs, and what their structure entail in terms of inference power. In Chapter 3, we address existing PC structure learning algorithms, and which guarantees in terms of tractability each give. We cover the two new structure learners in Chapter 4, providing empirical results on their performance. The final chapter is dedicated to summarizing our research contributions and pointing to potential future work in learning PCs.

Our contributions in this dissertation address the following research topics.

### Scalably learning PCs directly from background knowledge

In Geh and Denis Deratani Mauá (2021), we provide a learning algorithm for PSDDs that learns a PC directly from background knowledge in the form of logical constraints. The algorithm samples a structure from a distribution of possible PSDDs that are weakly consistent with the logical formula. How weak consistency is depends on a parameter that trades permission of false statements as non zero probability events with circuit complexity. We provide the algorithm and empirical results in Section 4.??.

### Using ensembles to strengthen consistency

The PC sampler given by Geh and Denis Deratani Mauá (2021) produces competitive probabilistic models (in terms of likelihood), albeit weak logical models in the sense that it possibly assigns non-zero probability to false variable assignments – as we discuss in Section 4.??, it never assigns zero probability to true statements. By producing many weak models, we not only gain in terms of data fitness, but also consistency: if any one component in the ensemble returns an assignment to be impossible, the whole model should return false.

### Random projections to efficiently learn PCs

Usual methods often employ clustering algorithms for constructing convex combinations of computational units. These can take many iterations to converge or require space quadratic in the number of data points. Instead, in Section 4.?? we present linear alternatives based on random projections (Freund *et al.*, 2008; Dasgupta and Freund, 2008).

# Chapter 2

# Probabilistic Circuits

As we briefly mentioned in the last chapter, Probabilistic Circuits (PCs) are conceptualized as computational graphs under special conditions. In this chapter, Section 2.1 to be more precise, we formally define PCs and give an intuition on their syntax, viewing other probabilistic models through the lenses of the PC framework. In Section 2.2, we describe the special structural constraints that give PCs their inference power over other generative models and state which queries (as far as we know) are enabled from each constraint.

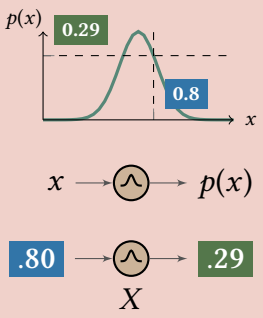
## 2.1 Distributions as Computational Graphs

Probabilistic circuits are directed acyclic graphs usually recursively defined in terms of their computational units. In its simplest form a PC is a single unit with no outgoing edges whose value corresponds to the result of a function. These are often called *input* nodes, and can take any form as long as its value is tractably computable. More concretely, input nodes typically represent probability density (or mass) functions, although they also support inputs as joint probability density functions of complex non-parametric models as well. To simplify notation, from here on out we shall use the term distribution and probability density (resp. mass) function interchangeably and argue that the input node represents a probability distribution.

**Example 2.1.1: Gaussians as probabilistic circuits**

Let $\mathsf{L}_p$ an input node and $p(X) = \mathcal{N}(X; \mu, \sigma^2)$ a univariate Gaussian distribution. Computing any query on $\mathsf{L}_p$ is straightforward: any query on $\mathsf{L}_p$ directly translates to $p$. As an example, suppose $\mu = 0$ and $\sigma^2 = 1$ and we wish to compute $\mathsf{L}_p(x = 0.8)$. The probability of this input shall then be

$$\mathsf{L}_p(x = 0.8) = \mathcal{N}(x = 0.8; \mu = 0, \sigma^2 = 1) = 0.29.$$



Let $\mathsf{L}_p$ a PC input node and denote $p$ as its inherent probability distribution. By definition, any query $f : \mathcal{X} \to \mathcal{Y}$ which is tractable on $p$ is tractable on $\mathsf{L}_p$. We shall denote $\mathsf{L}_p(\mathbf{X}) = p(\mathbf{X})$, and often omit $p$ when its explicit form is not needed. Evidently, a single input node lacks the expressivity for modeling complex models, otherwise we would have just used the input distribution as a standalone model. The expressiveness of PCs comes from recursively combining distributions into complex functions. This can be done through computational units that either compute convex combinations or products of their children. Let us first look at convex combinations, known in the literature as *sum* nodes.

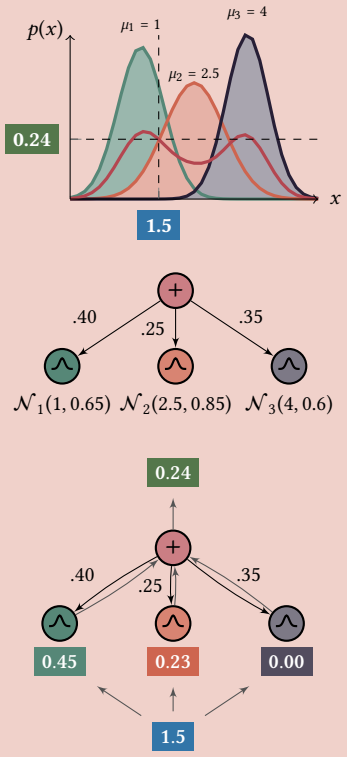
Let $\mathsf{S}$ be a PC sum node, and denote by $\mathrm{Ch}(\mathsf{S})$ the children nodes of $\mathsf{S}$. For every edge $\overrightarrow{\mathsf{S}\,\mathsf{C}}$ coming out of $\mathsf{S}$ and going to $\mathsf{C}$, we attribute a weight $w_{\mathsf{S},\mathsf{C}} > 0$, such that $\sum_{\mathsf{C}\in\mathrm{Ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}} = 1$. A sum node semantically defines a mixture model over its children, essentially acting as a latent variable over the component distributions (Poon and P. Domingos, 2011; Peharz, Gens, Pernkopf, *et al.*, 2016). Its value is the weighted sum of its children $p_{\mathsf{S}}(\mathbf{X} = \mathbf{x}) = \sum_{\mathsf{C}\in\mathrm{Ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}} \cdot p_{\mathsf{C}}(\mathbf{X} = \mathbf{x})$. To simplify notation, we shall use $\mathsf{N}(\mathbf{X} = \mathbf{x})$ as an alias for $p_{\mathsf{N}}(\mathbf{X} = \mathbf{x})$, that is, the probability function given by $\mathsf{N}$'s induced distribution. We often omit the variable assignment $\mathbf{X} = \mathbf{x}$ to $\mathbf{x}$ if the notation is unambiguous.

**Example 2.1.2: Gaussian mixture models as probabilistic circuits**

A Gaussian Mixture Model (GMM) defines a mixture over Gaussian components. Say we wish to compute the probability of $X = x$ for a GMM $\mathcal{G}$ with three components $\mathcal{N}_1(\mu_1 = 1, \sigma_1^2 = 0.65)$, $\mathcal{N}_2(\mu_2 = 2.5, \sigma_1^2 = 0.85)$ and $\mathcal{N}_3(\mu_3 = 4, \sigma_3^2 = 0.6)$, and suppose we have weights set to $\phi = (0.4, 0.25, 0.35)$. Computing the probability of $G$ amounts to the weighted summation

$$\mathcal{G}(X = x) = 0.4 \cdot \mathcal{N}_1(x; \mu_1, \sigma_2^2) + 0.25 \cdot \mathcal{N}_2(x; \mu_2, \sigma_2^2) + 0.35 \cdot \mathcal{N}_3(x; \mu_3, \sigma_3^2),$$

which is equivalent to a computational graph (i.e. a PC) with a sum node whose weights are set to $\phi$ and children are the components of the mixture. The figure on the right shows $\mathcal{G}$ (top) and its corresponding PC (middle). Given $x = 1.5$ (in blue), input nodes are computed following the inference flow (bottom, gray edges) up to the root sum node (in red), where a weighted summation is computed to output the probability (in green).



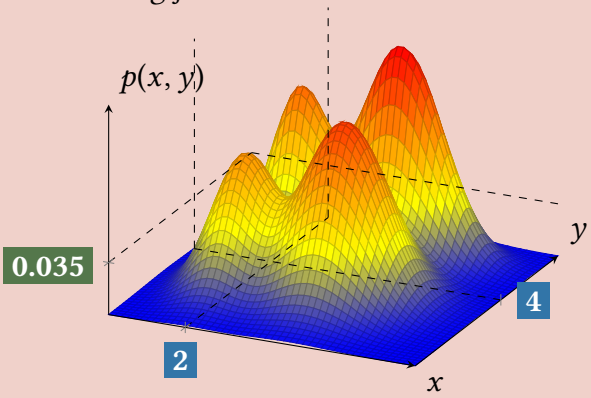

So far, the only nonlinearities present in PCs come from the internal computations of input nodes. In fact, a PC that only contains sums inputs can always be reduced to a sum node rooted PC with a single layer, i.e. a mixture model (see Theorem A.1.1). Adding *product nodes* as another form of nonlinearity increases expressivity sufficiently for PCs

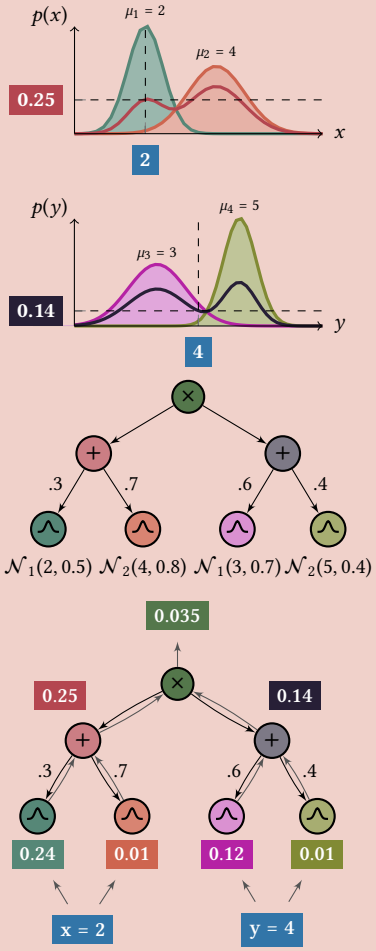
to be capable of representing any (discrete) probability distribution (Darwiche, 2003; Martens and Medabalimi, 2014; Peharz, Tschiatschek, *et al.*, 2015). More importantly, products semantically act as factorizations of their children, indicating an independence relationship between variables from different children. In practice, product nodes are simply products of their childrens' distribution: if P is a PC product node, then its value is given by $\mathrm{P}(\mathbf{X} = \mathbf{x}) = \prod_{\mathrm{C} \in \mathrm{Ch(P)}} \mathrm{C}(\mathbf{X} = \mathbf{x})$.

**Example 2.1.3: Factors as probabilistic circuits**

Say we have two GMMs $\mathcal{G}_1$ and $\mathcal{G}_2$. The first is a mixture model over variable $X$, with component weights $\phi_1 = (0.3, 0.7)$ and gaussians $\mathcal{N}_1(\mu_1 = 2, \sigma_1 = 0.5)$ and $\mathcal{N}_2(\mu_2 = 4, \sigma_2 = 0.8)$. The second is composed of $\mathcal{N}_3(\mu_3 = 3, \sigma_2 = 0.7)$ and $\mathcal{N}_4(\mu_4 = 5, \sigma_2 = 0.4)$, both distributions over variable $Y$ and with weights $\phi_2 = (0.6, 0.4)$.

Suppose $X \perp\!\!\!\perp Y$, yet we wish to compute the joint probability of both $x$ and $y$. If $X \perp\!\!\!\perp Y$, then $p(x, y) = p(x)p(y) = \mathcal{G}_1(x)\mathcal{G}_2(y)$, which corresponds to a factoring of mixtures. This is represented as a product node (in green) over the two mixture models (in red and purple). The resulting joint of this circuit is shown below.




Now that we have introduced the three most important computational units in PCs, we are finally ready to formally define probabilistic circuits.

**Definition 2.1.1** (Probabilistic circuit)**.** *A probabilistic circuits $\mathcal{C}$ is a rooted DAG whose nodes compute any tractable operation of their children, usually either convex combinations, known as* sum *nodes, or* products*. Nodes with no outgoing edges, i.e.* input *nodes, are tractable nonnegative functions whose integrals exist and equal to one. Computing a value from $\mathcal{C}$ amounts to a bottom-up feedforward pass from input nodes to root.*

While we assume that *tractable* operation or function is acceptable, we are usually interested in $\mathcal{O}(1)$ time computable operations, and often assume the same of input functions to simplify analysis. Further, in this dissertation we are only interested in convex combinations and products, and as such only these operations are considered. When a

probabilistic circuit $C$ contains no consecutive sums or products (i.e. for every sum all of its children are either inputs or products and respectively for products) then it is said to be a *standard* form circuit. Any PC can be transformed into a *standardized* circuit, a process we call *standardization* (see Theorem A.1.2).

**Remark 2.1.1: On operators and tractability**

Throughout this work we consider only products and convex combinations (apart from the implicit operations contained within input nodes) as potential computational units. The question of whether any other operator could be used to gain expressivity without loss of tractability is without a doubt an interesting research question, and one that is actively being pursued. However, this is certainly out of the scope of this dissertation, and so we restrict discussion on this topic and only give a brief comment on operator tractability here, pointing to existing literature in this area of research.

A. Friesen and P. Domingos (2016) formalize the notion of replacing sums and products in PCs with any pair of operators in a commutative semiring, giving results on the conditions for marginalization to be tractable. They provide examples of common semirings and to which known formalisms they correspond to. One such example are PCs under the Boolean semiring $(\{0, 1\}, \vee, \wedge, 0, 1)$ for logical inference, which are equivalent to Negation Normal Form (NNF, Barwise, 1982) and constitute an instance of Logic Circuits (LCs), of which Sentential Decision Diagrams (SDDs, Darwiche, 2011) and Binary Decision Diagrams (BDDs, Akers, 1978) are a part of. Another less common semiring in PCs is the real min-sum semiring $(\mathbb{R}_\infty, \min, +, \infty, 0)$ for nonconvex optimization (A. L. Friesen and P. Domingos, 2015).

Recently, Vergari, Choi, Liu, *et al.* (2021) extensively covered tractability conditions and complexity bounds for convex combinations, products, exp (and more generally powers in both naturals and reals), quotients and logarithms, even giving results for complex information-theoretic queries, such as entropies and divergences. Notably, they analyze whether structural constraints (and thus, in a sense, tractability) under these conditions are preserved.

Up to now, we have only considered summations as nonnegative weighted sums. Indeed, in most literature the sum node is defined as a convex combination. However, negative weights have appeared in Logistic Circuits (Liang and Van den Broeck, 2019) for discriminative modeling; and in Probabilistic Generating Circuits (Zhang *et al.*, 2021), a class of tractable probabilistic models that subsume PCs. Denis D. Mauá *et al.* (2017) extend (nonnegative) weights in sum nodes with probability intervals, effectively inducing a credal set (Cozman, 2000) for measuring imprecision.

Other works include PCs with quotients (Sharir and Shashua, 2018), transformations (Pevný *et al.*, 2020), max (Melibari *et al.*, 2016), and einsum (Peharz, Lang, *et al.*, 2020) operations.

Before we address the key components that make PCs interesting tractable probabilistic models, we must first discuss some important concepts that often come up in PC literature.

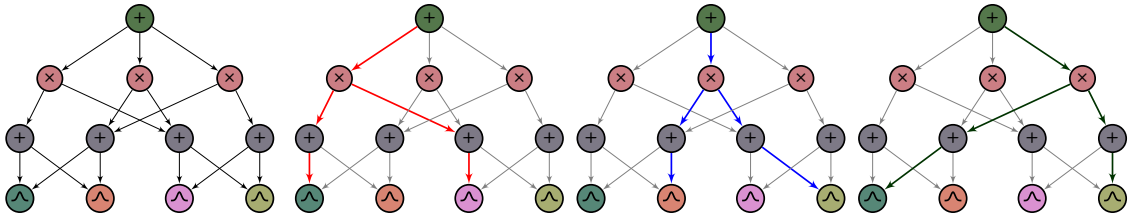



**Figure 2.1:** *A probabilistic circuit (a) and 3 of the 12 possible induced subcircuits (b).*

Mainly, we are interested in defining two notions here: the scope of a unit and induced subcircuits.

In simple terms, the scope of a computational unit N of a PC is merely the set of all variables that appear in the descendants of N. More formally, denote by Sc(N) the set of all variables that appear in N. We inductively compute the scope of circuit by a bottom-up approach: the scope of an input node $\mathsf{L}_p$ is the set of variables that appear in $p$'s distribution[1], and the scope of any other node is the union of all of its childrens' scopes. The notion of scope is essential to the structural constraints seen in Section 2.2.

As an example, take the circuit from Example 2.1.3. The scope of input nodes and are $\mathrm{Sc}() = \mathrm{Sc}() = \{X\}$, while $\mathrm{Sc}() = \mathrm{Sc}() = \{Y\}$. Consequentially, their parent sum nodes will have the same scope as their children $\mathrm{Sc}() = \{X\}$ and $\mathrm{Sc}() = \{Y\}$, yet the root node's scope is $\mathrm{Sc}() = \{X, Y\}$, since its childrens' scopes are distinct. The size of a probabilistic circuit is the number of nodes and edges of its computational graph. We use $|\mathcal{C}|$ to denote the size of a probabilistic circuit $\mathcal{C}$.

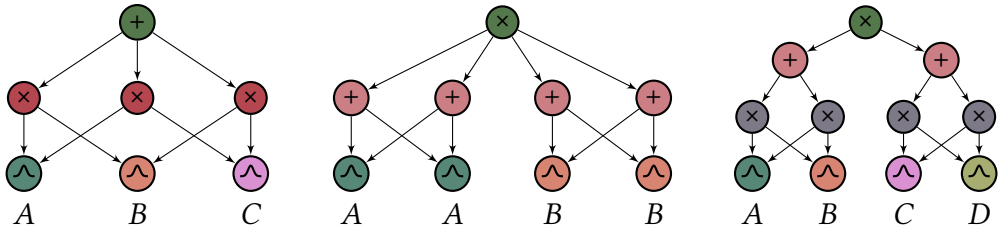
Let $\mathcal{C}$ a probabilistic circuit and node $\mathsf{N} \in \mathcal{C}$. We say that $\mathcal{S}_\mathsf{N}$ is a subcircuit of $\mathcal{C}$ rooted at N if $\mathcal{S}_\mathsf{N}$'s root is N, all nodes and edges in $\mathcal{S}_\mathsf{N}$ are also in $\mathcal{C}$ and $\mathcal{S}$ is also a probabilistic circuit. We now introduce the concept of induced subcircuits (Chan and Darwiche, 2006; Dennis and Ventura, 2015; Peharz, Gens, and P. Domingos, 2014).

**Definition 2.1.2** (Induced subcircuit)**.** *Let $\mathcal{C}$ a probabilistic circuit. An induced subcircuit $\mathcal{S}$ of $\mathcal{C}$ is a subcircuit of $\mathcal{C}$ such that all edges coming out of product nodes in $\mathcal{C}$ are also in $\mathcal{S}$, and of all edges coming out of sum nodes in $\mathcal{C}$, only one is in $\mathcal{S}$.*

Examples of induced subcircuits are visualized in Figure 2.1. When the induced subcircuit is a tree, as is the case in Figure 2.1, they are referred to as induced tree (Zhao, Melibari, *et al.*, 2015; Zhao, Adel, *et al.*, 2016).

So far, by Definition 2.1.1 a PC does not yet necessarily represent a probability distribution. To do so, the structure must obey constraints that we have only previously mentioned in passing. We formally define them next.

[1] Although we previously defined input nodes as mere functions, here we are explicitly associating a random variable to a *probability* function. Indeed, if we are being rigorous, we should define input nodes as a pair of random variable and function. To save space we instead assume, as previously stated, that the function is seen as both the probability (density) function as well as the distribution itself, and thus its scope is the scope of its distribution, i.e. the random variables that come into play in a probability distribution.




(a) (b) (c)

**Figure 2.2:** *Decomposable but unsmooth (a), smooth but undecomposable (b), and smooth and decomposable (c) circuits.*

## 2.2 Deciding What to Constraint

In this section we are interested in studying how the structural constraints in PCs enable different inference tasks. We shall first cover the more basic queries, namely *probability of evidence* (EVI), *marginal probability* (MAR), *conditional probability* (CON) and *maximum a posteriori probability* (MAP). After that we briefly address more complex queries such as mutual information, entropies and *expectation* (EXP).

### 2.2.1 Basic Queries

The most basic inference task we are interested in computing is the probability of evidence. To unlock this, we must introduce two structural constraints known as *smoothness* and *decomposability*.

**Definition 2.2.1** (Smoothness)**.** *A probabilistic circuit $\mathcal{C}$ is said to be* smooth *if for every sum node $S$ in $\mathcal{C}$,* $\mathrm{Sc}(C_1) = \mathrm{Sc}(C_2)$ *for* $C_1, C_2 \in \mathrm{Ch}(S)$.

**Definition 2.2.2** (Decomposability)**.** *A probabilistic circuit $\mathcal{C}$ is said to be* decomposable *if for every product node $P$ in $\mathcal{C}$,* $\mathrm{Sc}(C_1) \cap \mathrm{Sc}(C_2) = \emptyset$ *for* $C_1, C_2 \in \mathrm{Ch}(P)$.

For any *smooth* and *decomposable* PC, computing EVI is done in linear time in the number of edges. In fact this is true for MAR and CON as well. To compute marginals, it is sufficient to compute the corresponding marginals with respect to each input node and proceed to propagate values bottom-up. For conditionals, we simply compute two passes: one where we marginalize the conditional variables and the other any other variables that are not present in our query. These procedures are formalized in the theorem below.

**Theorem 2.2.1.** *Let $\mathcal{C}$ a* smooth *and* decomposable *PC. Any one of EVI, MAR or CON can be computed in linear time (in the number of edges of $\mathcal{C}$).*

Importantly, EVI and CON are both special cases of MAR in the sense that they are marginalizations over different intervals (see Page 19). More specifically, EVI is a marginalization over an empty set and CON is simply the quotient between two marginalization passes. Algorithmically, this means that the only distinction between these three queries is on what to do on the input nodes. We say that a variable assignment $\mathbf{x}$ for circuit $\mathcal{C}$ is *complete* if for every variable $X \in \mathrm{Sc}(\mathcal{C})$, $\mathbf{x}$ assigns a value to $X$; otherwise it is said to be a *partial* assignment. Algorithm 1 and Algorithm 2 show the exact algorithmic procedures

**Algorithm 1** EVI

**Input** A probabilistic circuit $\mathcal{C}$ and complete assignment $\mathbf{x}$
**Output** Probability $\mathcal{C}(\mathbf{x})$

$\mathbf{N} \leftarrow$ ReverseTopologicalOrder($\mathcal{C}$)
Let $v$ a hash function mapping a node to its probability
**for** each $\mathsf{N} \in \mathbf{N}$ **do**
    **if** $\mathsf{N}$ is an input **then** $v_{\mathsf{N}} \leftarrow \mathsf{N}(\mathbf{x})$
    **else if** $\mathsf{N}$ is a sum **then** $v_{\mathsf{N}} \leftarrow \sum_{\mathsf{C} \in \mathrm{Ch}(\mathsf{N})} w_{\mathsf{N},\mathsf{C}}\, v_{\mathsf{C}}$
    **else if** $\mathsf{N}$ is a product **then** $v_{\mathsf{N}} \leftarrow \prod_{\mathsf{C} \in \mathrm{Ch}(\mathsf{N})} v_{\mathsf{C}}$
**return** $v_R$, where $R$ is $\mathcal{C}$'s root

**Algorithm 2** MAR

**Input** A probabilistic circuit $\mathcal{C}$ and partial assignment $\mathbf{x}$
**Output** Probability $\int \mathcal{C}(\mathbf{x}, \mathbf{y})\, \mathrm{d}\mathbf{y}$

$\mathbf{N} \leftarrow$ ReverseTopologicalOrder($\mathcal{C}$)
Let $v$ a hash function mapping a node to its probability
**for** each $\mathsf{N} \in \mathbf{N}$ **do**
    **if** $\mathsf{N}$ is an input **then** $v_{\mathsf{N}} \leftarrow \int \mathsf{N}(\mathbf{x}, \mathbf{y})\, \mathrm{d}\mathbf{y}$
    **else if** $\mathsf{N}$ is a sum **then** $v_{\mathsf{N}} \leftarrow \sum_{\mathsf{C} \in \mathrm{Ch}(\mathsf{N})} w_{\mathsf{N},\mathsf{C}}\, v_{\mathsf{C}}$
    **else if** $\mathsf{N}$ is a product **then** $v_{\mathsf{N}} \leftarrow \prod_{\mathsf{C} \in \mathrm{Ch}(\mathsf{N})} v_{\mathsf{C}}$
**return** $v_R$, where $R$ is $\mathcal{C}$'s root

to extract EVI and MAR queries from $\mathcal{C}$. For CON, it suffices to run two MARs.

Because EVI, MAR and CON are the most basic forms of querying in PCs, we shall refer to these as *base queries*. Although smoothness and decomposability are sufficient conditions for tractable base queries, they are not necessary conditions. As a matter of fact, decomposability can be replaced with a third weaker constraint known as *consistency*. Denote by Desc(N) the set of all descendants of N.

**Definition 2.2.3** (Consistency)**.** *A probabilistic circuit $\mathcal{C}$ is said to be* consistent *if for any product node $P$ in $\mathcal{C}$, it holds that for every two children $C_1, C_2 \in \mathrm{Ch}(P)$ there does not exist two input nodes $L_p^1 \in \mathrm{Desc}(C_1)$ and $L_q^2 \in \mathrm{Desc}(C_2)$ whose scope is the same and $p(\mathbf{x}) \neq q(\mathbf{x})$ for any $\mathbf{x} \in \mathcal{X}$.*

In Peharz, Tschiatschek, *et al.* (2015), the authors show that although smooth and consistent PCs are more succinct (Darwiche and Marquis, 2002) compared to smooth and decomposable circuits, the gain is only mild, further proving that any smooth and consistent PC can be polynomially translated to a decomposable equivalent. In practice, because constructing decomposable circuits (and verifying decomposability) is much easier compared to doing the same with consistency, we instead focus on smooth and decomposable PCs.

Suppose we wish to compute the most probable assignment of a variable, say for classi-

**Algorithm 3** MAP

**Input** A probabilistic circuit $\mathcal{C}$ and evidence assignment $\mathbf{x}$
**Output** Probability $\max_{\mathbf{y}} \mathcal{C}(\mathbf{y}|\mathbf{x})$

$\mathbf{N} \leftarrow$ ReverseTopologicalOrder($\mathcal{C}$)
Let $v$ a hash function mapping a node to its probability
**for** each $\mathsf{N} \in \mathbf{N}$ **do**
    **if** $\mathsf{N}$ is an input **then** $v_{\mathsf{N}} \leftarrow \max_{\mathbf{y}} \mathsf{N}(\mathbf{y}, \mathbf{x})$
    **else if** $\mathsf{N}$ is a sum **then** $v_{\mathsf{N}} \leftarrow \max_{\mathsf{C} \in \mathsf{Ch}(\mathsf{N})} w_{\mathsf{N},\mathsf{C}}\, v_{\mathsf{C}}$
    **else if** $\mathsf{N}$ is a product **then** $v_{\mathsf{N}} \leftarrow \prod_{\mathsf{C} \in \mathsf{Ch}(\mathsf{N})} v_{\mathsf{C}}$
**return** $v_R/\mathbf{x}$, where $R$ is $\mathcal{C}$'s root

**Algorithm 4** ARGMAP

**Input** A probabilistic circuit $\mathcal{C}$ and evidence assignment $\mathbf{x}$
**Output** The most probable state $\arg\max_{\mathbf{y}} \mathcal{C}(\mathbf{y}|\mathbf{x})$

Compute $\max_{\mathbf{y}} \mathcal{C}(\mathbf{y}|\mathbf{x})$ and store values in $v$
$\mathbf{N} \leftarrow$ MaxInducedTree($\mathcal{C}$, $v$)
Call $\mathbf{y}$ the set of initially empty arg max states
**for** each input node $\mathsf{L} \in \mathbf{N}$ **do**
    $\mathbf{y} \leftarrow \mathbf{y} \cup \arg\max_{\mathbf{z}} \mathsf{N}(\mathbf{z}, \mathbf{x})$
**return** $\mathbf{y}$

fication or image reconstruction. To do so, we must compute the conditional query

$$\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) = \max_{\mathbf{y}} \frac{p(\mathbf{y}, \mathbf{x})}{p(\mathbf{x})} = \frac{\max_{\mathbf{y}} p(\mathbf{y}, \mathbf{x})}{p(\mathbf{x})}, \tag{2.1}$$

often called the *maximum a posteriori* (MAP) probability. Although it seems at first like MAP is no harder than computing a CON, it turns out that for smooth and decomposable PCs, MAP is unfortunately intractable (Conaty *et al.*, 2017; Mei *et al.*, 2018). To unlock access to MAP we must make the circuit *deterministic*.

**Definition 2.2.4** (Determinism)**.** *A probabilistic circuit $\mathcal{C}$ is said to be* deterministic *if for every sum node $S \in \mathcal{C}$ only one child of $S$ has nonnegative value at a time for any complete assignment.*

At this point, we must introduce a graphical notation for *indicator* nodes. An indicator node is an input node whose function is the characteristic function $f(x) = [\![x = k]\!]$, i.e. a degenerate function with all of its mass on $k$ and zero anywhere else. A special case is when $X$ is binary and $k = 1$, in which case we say the input node is a literal node, denoting by the usual propositional notation $X$ for when $k = 1$ and $\neg X$ for $k = 0$. Graphically, we shall use ◎ for indicators and the textual propositional notation for literals.

With determinism, we now have access to MAP.

**Theorem 2.2.2.** *Let $\mathcal{C}$ a smooth, decomposable and* deterministic *PC. MAP is computable in linear time (in the number of edges of $\mathcal{C}$).*

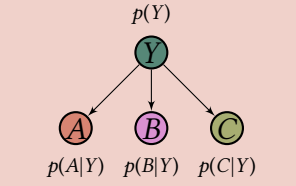
When a PC is smooth, decomposable and deterministic, the MAP is easily computable by simply replacing sum nodes with a max operation and performing a feedforward EVI pass. This is commonly called the Max-Product algorithm, shown more formally in Algorithm 3. To find the states that maximize Equation (2.1) in a given circuit $\mathcal{C}$, we first compute the MAP probabilities through the usual bottom-up pass, and then find the maximum (in terms of probability) induced tree $\mathcal{M}$ rooted at $\mathcal{C}$. This maximum induced tree can be retrieved by a top-down pass selecting the most probable sum child nodes according to the probabilities set by MAP. Since $\mathcal{C}$ is decomposable, there cannot exist a node in $\mathcal{M}$ with more than one parent, meaning it is by construction a tree whose leaves are input nodes with scopes whose union is the scope of $\mathcal{C}$. This reduces the problem to a divide-and-conquer approach where each input node is individually maximized (see Algorithm 4 and Page 20 for a formal proof on its validity).

**Example 2.2.1: Naïve Bayes as probabilistic circuits**

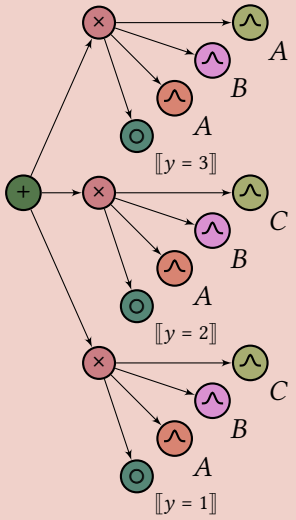
Suppose we have samples of per capita census measurements on three different features, say age $A$, body mass index $B$ and average amount of cheese consumed daily $C$ from three different cities $Y$. Assuming $A$, $B$ and $C$ are independent, given a sample $x = (a, b, c)$ we can use Gaussian naïve Bayes to predict $x$'s class

$$p(y|a, b, c) = p(y)p(a|y)p(b|y)p(c|y).$$

In PC terms, $p(y)$ are the prior probabilities, i.e. sum weights, for each class and $p(z|y)$ are Gaussian input nodes corresponding to the distributions of each feature in each city. To make sure that these are in fact conditional distributions, we introduce indicator variables "selecting" $Y$'s state. Since the PC resulting is deterministic, we can compute the MAP for classification in linear time by simply replacing the root node with a max, which is exactly equivalent to finding the highest likelihood of $x$ for each city $y$.




Gaussian naïve Bayes




Equivalent PC

## 2.2.2 Complex Queries

Although base queries cover most of the needs of the usual data scientist, more complex tasks involving information-theoretic measures, logical queries or distributional divergences are also (tractably) computable under the right conditions. Particularly, we are interested in a key component for tractability in all these tasks: the notion of *vtrees* and *structure decomposability*, a stronger variant of decomposability where variable partitionings on product nodes follow a hierarchy. This hierarchy is easily visualized through a *vtree* (variable tree), a data structure that defines a (partial) ordering of variables.

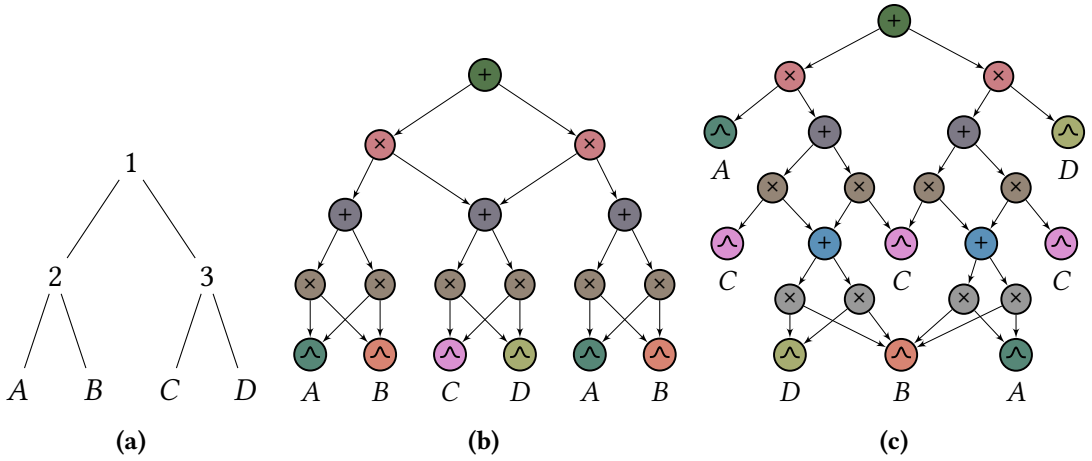



**Figure 2.3:** *A vtree (a) defining an order* $(A, B, C, D)$*, a 2-standard structure decomposable probabilistic circuit that respects the vtree (b), and a 2-standard decomposable probabilistic circuit that does not (c).*

**Definition 2.2.5** (Vtree)**.** *A variable tree* $\mathcal{V}$*, or* vtree*, over a set of variables* $\mathbf{X}$ *is a binary tree whose leaf nodes have a one-to-one and onto mapping* $\phi_{\mathcal{V}}$ *with* $\mathbf{X}$*.*

We shall adopt the same scope definition and notation $\text{Sc}(\cdot)$ for vtrees as in PCs. Let $v$ a vtree node from a vtree $\mathcal{V}$. If $v$ is a leaf node, its scope is $\phi_{\mathcal{V}}(v)$, i.e. the leaf's assigned variable; otherwise its scope is the union of the scope of its children. For an inner node $v$, we shall call its left child $v^{\leftarrow}$ and right child $v^{\rightarrow}$. Every inner node $v$ of a vtree $\mathcal{V}$ defines a variable *partitioning* of the scope $(\text{Sc}(v^{\leftarrow}), \text{Sc}(v^{\rightarrow}))$, while the leaves of $\mathcal{V}$ define a partial ordering of $\text{Sc}(\mathcal{V})$. We are especially interested in the scope partitioning aspect of vtrees.

**Definition 2.2.6.** *A product node* $P$ respects *a vtree node* $v$ *if* $P$ *contains only two children* $\text{Ch}(P) = \{C_1, C_2\}$*, and* $\text{Sc}(C_1) = \text{Sc}(v^{\leftarrow})$ *and* $\text{Sc}(C_2) = \text{Sc}(v^{\rightarrow})$*.*

Obviously the above definition is vague with regards to which child (i.e. graphically, left or right) of P should respect the scope of which $v$ child. We therefore assume a fixed order for P's children and say that the (graphically) left child is called the *prime* and (graphically) right child the *sub*. This ultimately means that the scope of the prime (resp. sub) of P must be the same as the scope of the left (resp. right) child of $v$. Although the graphical concept of left and right is needed for easily visualizing the scope partitioning of a product with respect to a vtree node, we do not use it strictly. In fact, when the situation is unambiguous, we compactly represent the computational graph without adhering to the left-right convention in favor of readability.

We say that a vtree is linear, if either it is left-linear or right-linear. A left- (resp. right) linear vtree is a vtree whose inner nodes all have leaf nodes on their right (resp. left) child. Similarly, a vtree is said to be left- (resp. right) leaning if the number of leaf nodes as right (resp. left) children is much higher then left (resp. right) children. Otherwise, it is a balanced vtree. Figure 2.3a shows a balanced vtree.

Now that we understand what a vtree is, we can properly introduce *structure decomposability*, a stronger variant of decomposability. We say that a PC is 2-standard if it is standard and all of its product nodes have exactly two children.

**Definition 2.2.7** (Structure decomposability)**.** *Let $\mathcal{C}$ a 2-standard probabilistic circuit and $\mathcal{V}$ a vtree with same scope as $\mathcal{C}$. $\mathcal{C}$ is said to be* structure decomposable *if every $i$-th product layer of $\mathcal{C}$ respects every $i$-th inner node layer of $\mathcal{V}$.*

Although we assume a *2-standard* PC in the above definition, its assumption was only for convenience, and does not imply in a loss of expressivity. As a matter of fact, any PC can be 2-standardized, i.e. to standardize the circuit such that every product node only has two children (see Theorem A.1.3). Intuitively, structure decomposability merely states that every two product nodes whose scopes are the same must partition their scopes (between their two children) exactly the same (and according to their corresponding vtree node).

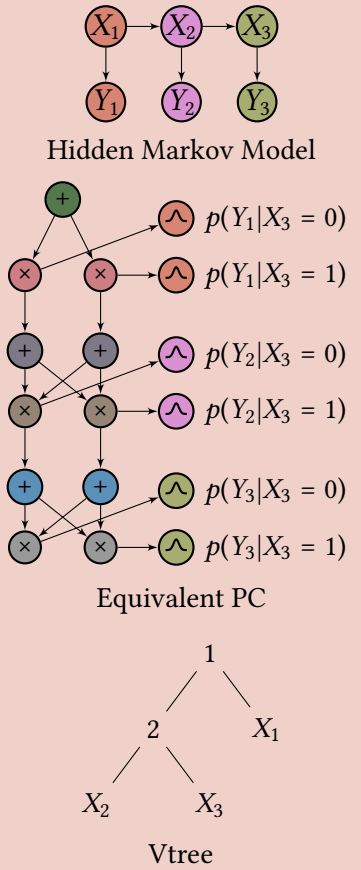
Figure 2.3 shows a vtree $\mathcal{V}$ and two probabilistic circuits, say $\mathcal{C}_1$ for the one in the middle and $\mathcal{C}_2$ for the one on the right. Notice how $\mathcal{C}_1$ respects $\mathcal{V}$, as each ⊗ respects the split at vtree node 1 (namely $\{A, B\}$). The primes are then ⊕ whose scopes are $\{A, B\}$, while the sub is the one with two parents and scope $\{C, D\}$. For each of these, their children ⊗ also respect $\mathcal{V}$: they either encode he same split as 2 or as 3, depending on whether they are descendants from the sub or prime of 1. Although $\mathcal{C}_2$ is decomposable, it does *not* respect $\mathcal{V}$, as ⊗ encode different variable partionings $(\{A\}, \{B, C, D\})$ and $(\{A, B, C\}, \{D\})$. In fact, it is not structure decomposable, as it does not respect any vtree.

**Example 2.2.2: Hidden Markov models as probabilistic circuits**

Say we wish to model a temporal structured dependence between three latent binary variables, for example the presence of a subject $X_1$, verb $X_2$ and object $X_3$ in a natural language phrase. Each observation $Y_i$ is a fragment (of $X_i$) taken from a complete sentence $\mathbf{y} = (y_1, y_2, y_3)$. The first-order Hidden Markov Model (HMM) (on the right) models the joint probability of sentences

$$p(X_{1..3}, Y_{1..3}) = p(X_1) \prod_{i=2}^{3} p(X_i|X_{i-1}) \prod_{i=1}^{3} p(Y_i|X_i).$$

This is computationally equivalent to the PC on the right. Each input node $p(Y_i|X_i)$ is a conditional distribution model (possibly another PC) for each assignment (here two) of $X_i$, meaning that if $p(Y_i|X_i = 0) > 0$, then $p(Y_i|X_i = 1) = 0$ and vice-versa. Further, notice how every product (except for ⊗s which are redundant nodes) follows the partitionings imposed by the vtree below it. What this means is that this PC is not only smooth, but structure decomposable and deterministic.




Despite our structure decomposability definition relying on a vtree, there is at least

**(a)**

one alternative definition that defines it in terms of *circuit compatibility*. Essentially, a circuit $\mathcal{C}_1$ is *compatible* with $\mathcal{C}_2$ if they can be 2-standardized and rearranged such that any two products with same scope, one from $\mathcal{C}_1$ and the other $\mathcal{C}_2$, partition the scope the same way (Vergari, Choi, Liu, *et al.*, 2021). A structure decomposable PC is then defined as a PC that is compatible with a copy of itself. In summary, the two definitions of structure decomposability are equivalent, except compatibility implicitly assumes an arrangement of product scopes that is analogous to a vtree.

The notion of a vtree (or compatibility for that matter) is key to more complex queries. For instance, given two probabilistic circuits $\mathcal{C}_1$ and $\mathcal{C}_2$, computing cross entropy between the two is $\mathcal{O}(|\mathcal{C}_1||\mathcal{C}_2|)$ as long as both have the same vtree and at least the circuit that needs to be log-computable is also deterministic. Likewise, computing the Kullback-Leibler (KL) divergence between $\mathcal{C}_1$ and $\mathcal{C}_2$ requires that the two share the same vtree and both be deterministic. Mutual Information (MI), in turn, calls for the circuit to be smooth, structure decomposable and an even stronger version of determinism where sums can only have one nonnegative valued child for any *partial* assignment at a time. For a more detailed insight on the tractability of these (and other) queries, as well as proofs on these results, we point to the comprehensive study of Vergari, Choi, Liu, *et al.* (2021).

A particularly interesting class of queries that becomes tractable when circuits are structure decomposable is expectation (EXP). One notable example from this class is computing the probability of logical events. This leads us to logic circuits, a parallel version of probabilistic circuits for logical reasoning.

## 2.3 Probabilistic Circuits as Knowledge Bases

We briefly and superficially mentioned in remark 2.1.1 that PCs under a Boolean semiring with conjunctions and disjunctions as operators are known as Logic Circuits (LCs). In this section, we formally define LCs and more precisely show the connection between PCs and LCs.

Logic circuits are computational graphs just like PCs, but whose input are always Booleans, scope is over propositional variables and computational units define either a conjunction, disjunction, literal ou constant. Similar to PCs, computing the satisfiability of an assignment is done by a bottom-up feedforward evaluation of the circuit.

In terms of notation, we shall use ⋁ for disjunction nodes, ⋀ for conjunction nodes, $\bot$ and $\top$ for true and false constant nodes respectively, and $X$ and $\neg X$ for propositional nodes.

**Example 2.3.1: BDDs as logic circuits**

# Appendix A

# Appendix

## A.1 Proofs

**Theorem A.1.1.** *Let $\mathcal{C}$ a probabilistic circuit whose first $l$ layers are composed solely of sum nodes. Call $\mathbf{N}$ the set of all nodes in layer $l+1$. $\mathcal{C}$ is equivalent to a PC $\mathcal{C}'$ whose root is a sum node with $\mathbf{N}$ as children.*

*Proof.* We adapt a similar proof due to Jaini, Poupart, *et al.* (2018). Every sum node is of the form

$$\mathrm{S}(\mathbf{x}) = \sum_{\mathrm{C} \in \mathrm{Ch}(\mathrm{S})} w_{\mathrm{S},\mathrm{C}} \cdot \mathrm{C}(\mathbf{x}).$$

Particularly, every child C in a sum node in layer $1 \leq i \leq l-1$, is a sum node, and so for the first layer we have that

$$\begin{aligned}\mathrm{S}(\mathbf{x}) &= \sum_{\mathrm{C}_1 \in \mathrm{Ch}(\mathrm{S})} w_{\mathrm{S},\mathrm{C}_1} \sum_{\mathrm{C}_2 \in \mathrm{Ch}(\mathrm{C}_1)} w_{\mathrm{C}_1,\mathrm{C}_2}\, \mathrm{C}_2(\mathbf{x}) \\ &= \sum_{\mathrm{C}_1 \in \mathrm{Ch}(\mathrm{S})} \sum_{\mathrm{C}_2 \in \mathrm{Ch}(\mathrm{C}_1)} w_{\mathrm{S},\mathrm{C}_1} w_{\mathrm{C}_1,\mathrm{C}_2}\, \mathrm{C}_2(\mathbf{x}).\end{aligned}$$

Define a one-to-one mapping that takes a tuple $(\mathrm{C}_1, \mathrm{C}_2)$ where $\mathrm{C}_1 \in \mathrm{Ch}(\mathrm{S})$ and $\mathrm{C}_2 \in \mathrm{Ch}(\mathrm{C}_1)$ and returns a (unique) path from S to every grandchild $\mathrm{C}_2$ of S. Call $\mathbf{K}$ the set of all paths, and $w_{\mathrm{S},\mathrm{C}_1}$ and $w_{\mathrm{C}_1,\mathrm{C}_2}$ the weights for one such path. We can merge these two weights into a single weight $w'_{\mathrm{S},\mathrm{C}_2} = w_{\mathrm{S},\mathrm{C}_1} \cdot w_{\mathrm{C}_1,\mathrm{C}_2}$, yielding

$$\mathrm{S}(\mathbf{x}) = \sum_{(w_{\mathrm{S},\mathrm{C}_1}, w_{\mathrm{C}_1,\mathrm{C}_2}) \in \mathbf{K}} w'_{\mathrm{S},\mathrm{C}_2}\, \mathrm{C}_2(\mathbf{x}).$$

This ensures that two consecutive sum layers can be collapsed into a single layer. Particularly, for the first (root) and second layers, the above transformation generates a circuit with one fewer layer and whose root has $\mathcal{O}(nm)$ edges, where $n$ and $m$ are the number of edges coming from the original root and its children respectively. We can apply this

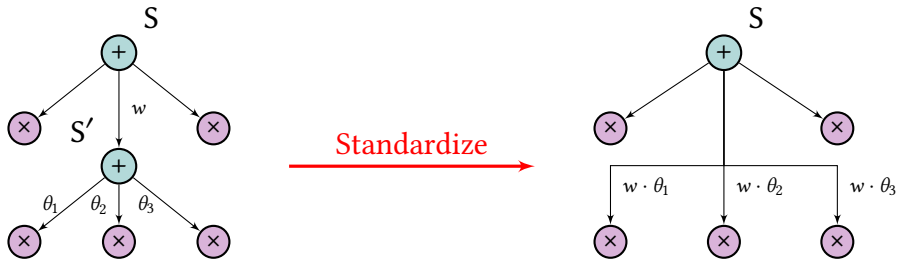
procedure until there are no more consecutive sum nodes. This results in a PC of the form

$$\mathrm{S}(\mathbf{x}) = \sum_{\mathrm{C} \in \mathrm{Ch}(S)} w_{\mathrm{S,C}}\, \mathrm{N}(\mathbf{x}),$$

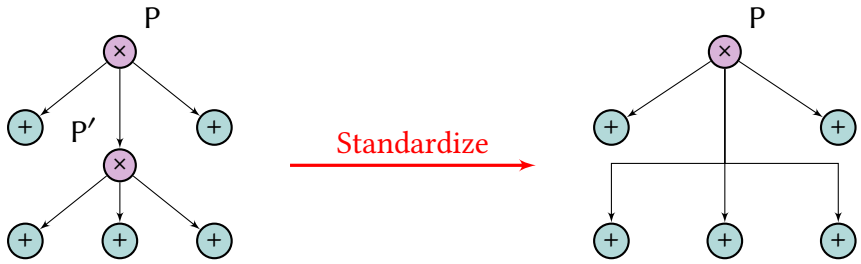
where $\mathrm{N} \in \mathbf{N}$. The number of children of the resulting root sum node will be exponential on the number of edges of its children. □

**Theorem A.1.2** (Standardization)**.** *Any probabilistic circuit $\mathcal{C}$ can be reduced to a circuit where every sum node contains only products or inputs and every product node contains only sums or inputs.*

*Proof.* If $\mathcal{C}$ is already standard we are done. Otherwise, there exists either (i) a sum node S with a sum S′ as child; or (ii) a product node P with a product P′ as child. We first address (i): let $w$ be the weight of edge $\overrightarrow{\mathrm{S\,S'}}$ and $\theta_i$ the weights from all edges coming out from S′.




Connect S with every child of S′, assigning as weight $w \cdot \theta_i$ for each child $i$. Delete S′ and all edges coming out from it. The resulting circuit is computationally equivalent but now without a consecutive pair of sums. This transformation is visualized by the figure above. We do a similar procedure in (ii), but now instead remove P′ and connect all children of P′ to P, as we show below.



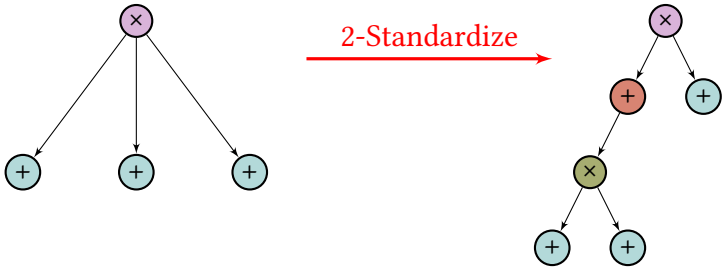

□

**Theorem A.1.3** (2-Standardization)**.** *Any probabilistic circuit $\mathcal{C}$ can be transformed into a circuit where every sum node contains only products or inputs and every product node contains only* two *sums or inputs.*

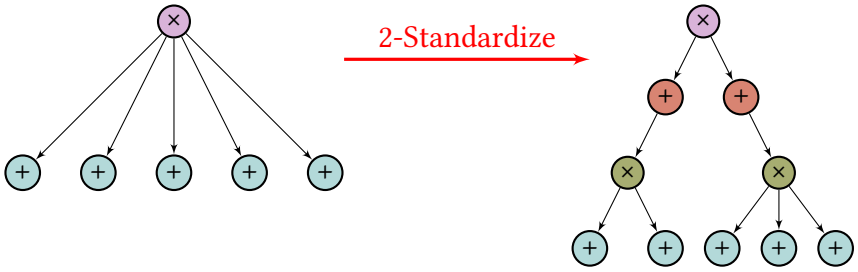
*Proof.* For sums, apply the same standardization procedure as Theorem A.1.2. Let P a product and call $n = |\mathrm{Ch(P)}|$. If $n = 1$ and Ch(P) is a product, then remove P and connect

all previous parents of P with its child. If $n = 1$ and Ch(P) is not a product, remove P and apply the standardization procedure for sums on all of Pa(P).

For $n > 2$, we simply need to split into 2-products recursively. We prove this by induction. The base case is when $n = 2$, which is already done, or $n = 3$, in which case we need apply the transformation below.




Where ⊕ and ⊗ are newly introduced nodes. When $n > 3$, we create two products $P_1$ and $P_2$, each connected with a sum and product, and with $\lfloor \frac{n}{2} \rfloor$ and $\lceil \frac{n}{2} \rceil$ potential children. By the induction hypothesis, we can recursively binarize the subsequent grandchildren products.




As an example, we have $n = 5$ in the figure above. We introduce the sums ⊕ and products ⊗ and then recursively apply the transformation again on the ⊗s.

When Ch(P) are product nodes we do the same procedure as before, but with the added post-process addition of a sum node connecting ⊗ to every Ch(P). □

**Theorem 2.2.1.** *Let $\mathcal{C}$ a* smooth *and* decomposable *PC. Any one of EVI, MAR or CON can be computed in linear time (in the number of edges of $\mathcal{C}$).*

*Proof.* For a sum node S, we have the following marginalization query

$$\begin{aligned}\int \mathrm{S}(\mathbf{x}, \mathbf{y})\, d\mathbf{y} &= \int \sum_{\mathrm{C} \in \mathrm{Ch}(\mathrm{S})} w_{\mathrm{S,C}}\, \mathrm{C}(\mathbf{x}, \mathbf{y})\, d\mathbf{y} \\ &= \sum_{\mathrm{C} \in \mathrm{Ch}(\mathrm{S})} w_{\Sigma,\mathrm{C}} \int \mathrm{C}(\mathbf{x}, \mathbf{y})\, d\mathbf{y}.\end{aligned}$$

Analogously, for a product node

$$\int \mathrm{P}(\mathbf{x}, \mathbf{y})\, d\mathbf{y} = \int \prod_{\mathrm{C} \in \mathrm{Ch}(\mathrm{P})} \mathrm{C}(\mathbf{x}, \mathbf{y})\, d\mathbf{y} = \prod_{\mathrm{C} \in \mathrm{Ch}(\mathrm{P})} \int \mathrm{C}(\mathbf{x}, \mathbf{y})\, d\mathbf{y}.$$

This ensures that marginals are pushed down to children. This can be done recursively until C is an input node $\mathrm{L}_p$, in which case we marginalize $\mathbf{y}$ according to $p$, which by definition should be tractable and here we assume can be done in $\mathcal{O}(1)$. We have proved the case for MAR. For EVI, we simply assign $\mathbf{y} = \emptyset$ with input nodes acting as probability density functions. Conditionals can easily be computed by an EVI or MAR followed by a second pass marginalizing the conditional variables $p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{x},\mathbf{y})}{p(\mathbf{y})}$ which are both done in linear time as we have seen here. □

**Theorem 2.2.2.** *Let C a smooth, decomposable and* deterministic *PC. MAP is computable in linear time (in the number of edges of C).*

*Proof.* For a sum node S, we want to compute the following query

$$\max_{\mathbf{y}} \mathrm{S}(\mathbf{y}|\mathbf{x}) = \frac{1}{\mathrm{S}(\mathbf{x})} \max_{\mathbf{y}} \mathrm{S}(\mathbf{y}, \mathbf{x}) = \frac{1}{\mathrm{S}(\mathbf{x})} \max_{\mathbf{y}} \sum_{\mathrm{C} \in \mathrm{Ch}(\mathrm{S})} w_{\mathrm{S,C}}\, \mathrm{C}(\mathbf{y}, \mathbf{x}),$$

yet notice that for any assignment of $\mathbf{x}$ and $\mathbf{y}$ only one $\mathrm{C} \in \mathrm{Ch}(\mathrm{S})$ must have a nonnegative value by the definition of determinism, so we may replace the summation with a maximization over the children, giving

$$\max_{\mathbf{y}} \mathrm{S}(\mathbf{y}|\mathbf{x}) = \frac{1}{\mathrm{S}(\mathbf{x})} \max_{\mathbf{y}} \max_{\mathrm{C} \in \mathrm{Ch}(\mathrm{S})} w_{\mathrm{S,C}}\, \mathrm{C}(\mathbf{y}, \mathbf{x}) = \frac{1}{\mathrm{S}(\mathbf{x})} \max_{\mathrm{C} \in \mathrm{Ch}(\mathrm{S})} \max_{\mathbf{y}} w_{\mathrm{S,C}}\, \mathrm{C}(\mathbf{y}, \mathbf{x}).$$

For a product node P, we compute

$$\max_{\mathbf{y}} \mathrm{P}(\mathbf{y}|\mathbf{x}) = \frac{1}{\mathrm{P}(\mathbf{x})} \max_{\mathbf{y}} \mathrm{P}(\mathbf{y}, \mathbf{x}) = \frac{1}{\mathrm{P}(\mathbf{x})} \max_{\mathbf{y}} \prod_{\mathrm{C} \in \mathrm{Ch}(\mathrm{P})} \mathrm{C}(\mathbf{y}, \mathbf{x}) = \frac{1}{\mathrm{P}(\mathbf{x})} \prod_{\mathrm{C} \in \mathrm{Ch}(\mathrm{P})} \max_{\mathbf{y}} \mathrm{C}(\mathbf{y}, \mathbf{x}).$$

This is equivalent to an inductive top-down pass where we maximize instead of sum until we reach all input nodes, in which case we simply maximize the supposedly tractable functions. Once these are computed, we unroll the induction, maximizing over all values. □

# References


[Akers 1978] S. B. Akers. "Binary decision diagrams". In: *IEEE Transactions on Computers* 27.6 (1978), pp. 509–516 (cit. on p. 8).

[Barwise 1982] Jon Barwise. *Handbook of Mathematical Logic*. 1982 (cit. on p. 8).

[Chan and Darwiche 2006] Hei Chan and Adnan Darwiche. "On the robustness of most probable explanations". In: *Proceedings of the 22nd Conference in Uncertainty in Artificial Intelligence*. 2006 (cit. on p. 9).

[Cheng *et al.* 2014] Wei-Chen Cheng, Stanley Kok, Hoai Vu Pham, Hai Leong Chieu, and Kian Ming A. Chai. "Language modeling with sum-product networks". In: *Fifteenth Annual Conference of the International Speech Communication Association*. 2014 (cit. on p. 3).

[Chernikova *et al.* 2019] A. Chernikova, A. Oprea, C. Nita-Rotaru, and B. Kim. "Are self-driving cars secure? evasion attacks against deep neural networks for steering angle prediction". In: *2019 IEEE Security and Privacy Workshops*. 2019, pp. 132–137 (cit. on p. 1).

[Choi *et al.* 2020] YooJung Choi, Antonio Vergari, and Guy Van den Broeck. "Probabilistic circuits: a unifying framework for tractable probabilistic models". In: (2020). In preparation (cit. on p. 2).

[Conaty *et al.* 2017] Diarmaid Conaty, Cassio Polpo de Campos, and Denis Deratani Mauá. "Approximation complexity of maximum A posteriori inference in sum-product networks". In: *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence*. 2017 (cit. on p. 12).

[Cozman 2000] Fabio G. Cozman. "Credal networks". In: *Artificial Intelligence* 120.2 (2000), pp. 199–233. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(00)00029-1. URL: https://www.sciencedirect.com/science/article/pii/S0004370200000291 (cit. on p. 8).

[Dang *et al.* 2020] Meihua Dang, Antonio Vergari, and Guy Van den Broeck. "Strudel: learning structured-decomposable probabilistic circuits". In: *Proceedings of the 10th International Conference on Probabilistic Graphical Models*. PGM. 2020 (cit. on p. 3).

[Darwiche 2003] Adnan Darwiche. "A differential approach to inference in bayesian networks". In: *Journal of the ACM* 50.3 (2003), pp. 280–305 (cit. on pp. 2, 7).

[Darwiche 2011] Adnan Darwiche. "SDD: a new canonical representation of propositional knowledge bases". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*. 2011, pp. 819–826 (cit. on p. 8).

[Darwiche and Marquis 2002] Adnan Darwiche and Pierre Marquis. "A knowledge compilation map". In: *J. Artif. Int. Res.* 17.1 (Sept. 2002), pp. 229–264. ISSN: 1076-9757 (cit. on p. 11).

[Dasgupta and Freund 2008] Sanjoy Dasgupta and Yoav Freund. "Random projection trees and low dimensional manifolds". In: *Proceedings of the fortieth annual ACM symposium on Theory of computing*. STOC. 2008, pp. 537–546 (cit. on p. 4).

[R. Dechter 1998] R. Dechter. "Bucket elimination: a unifying framework for probabilistic inference". In: *Learning in Graphical Models*. Ed. by Michael I. Jordan. Dordrecht: Springer Netherlands, 1998, pp. 75–104. ISBN: 978-94-011-5014-9. DOI: 10.1007/978-94-011-5014-9_4. URL: https://doi.org/10.1007/978-94-011-5014-9_4 (cit. on p. 2).

[Rina Dechter and Mateescu 2007] Rina Dechter and Robert Mateescu. "And/or search spaces for graphical models". In: *Artificial Intelligence* 171.2 (2007), pp. 73–106. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2006.11.003. URL: https://www.sciencedirect.com/science/article/pii/S000437020600138X (cit. on p. 2).

[Dennis and Ventura 2015] Aaron Dennis and Dan Ventura. "Greedy structure search for sum-product networks". In: *Proceedings of the 24th International Conference on Artificial Intelligence*. 2015, pp. 932–938 (cit. on p. 9).

[Di Mauro *et al.* 2017] Nicola Di Mauro, Floriana Esposito, Fabrizio G. Ventola, and Antonio Vergari. "Alternative variable splitting methods to learn sum-product networks". In: *AI*IA 2017 Advances in Artificial Intelligence*. Ed. by Floriana Esposito, Roberto Basili, Stefano Ferilli, and Francesca A. Lisi. Cham: Springer International Publishing, 2017, pp. 334–346. ISBN: 978-3-319-70169-1 (cit. on p. 3).

[Freund *et al.* 2008] Yoav Freund, Sanjoy Dasgupta, Mayank Kabra, and Nakul Verma. "Learning the structure of manifolds using random projections". In: *Advances in Neural Information Processing Systems*. Vol. 20. NeurIPS. 2008 (cit. on p. 4).

[A. Friesen and P. Domingos 2016] Abram Friesen and Pedro Domingos. "The sum-product theorem: a foundation for learning tractable models". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, June 2016, pp. 1909–1918. URL: https://proceedings.mlr.press/v48/friesen16.html (cit. on p. 8).

[A. L. Friesen and P. Domingos 2015] Abram L. Friesen and Pedro Domingos. "Recursive decomposition for nonconvex optimization". In: *Proceedings of the 24th International Conference on Artificial Intelligence*. IJCAI'15. Buenos Aires, Argentina: AAAI Press, 2015, pp. 253–259. ISBN: 9781577357384 (cit. on p. 8).

[Geh and Denis Deratani Mauá 2021] Renato Lui Geh and Denis Deratani Mauá. "Learning probabilistic sentential decision diagrams under logic constraints by sampling and averaging". In: *Proceedings of The 37th Uncertainty in Artificial Intelligence Conference*. Proceedings of Machine Learning Research. PMLR, 2021 (cit. on pp. 3, 4).

[Gens and P. Domingos 2012] Robert Gens and Pedro Domingos. "Discriminative learning of sum-product networks". In: *Advances in Neural Information Processing Systems 25*. NIPS, 2012, pp. 3239–3247 (cit. on p. 3).

[Gens and P. Domingos 2013] Robert Gens and Pedro Domingos. "Learning the structure of sum-product networks". In: *Proceedings of the 30th International Conference on Machine Learning*. ICML. 2013, pp. 873–880 (cit. on p. 3).

[Goodfellow *et al.* 2014] Ian Goodfellow *et al.* "Generative adversarial nets". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., 2014, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf (cit. on p. 2).

[Gritsenko *et al.* 2019] Alexey A. Gritsenko, Jasper Snoek, and Tim Salimans. "On the relationship between normalising flows and variational- and denoising autoencoders". In: *Deep Generative Models for Highly Structured Data, ICLR 2019 Workshop, New Orleans, Louisiana, United States, May 6, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=HklKEUUY%5C_E (cit. on p. 2).

[Guo *et al.* 2017] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. "On calibration of modern neural networks". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1321–1330 (cit. on p. 1).

[Jaeger 2004] Manfred Jaeger. "Probabilistic decision graphs-combining verification and ai techniques for probabilistic inference". In: *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* 12.1 supp (Jan. 2004), pp. 19–42. ISSN: 0218-4885. DOI: 10.1142/S0218488504002564. URL: https://doi.org/10.1142/S0218488504002564 (cit. on p. 2).

[Jaini, Ghose, *et al.* 2018] Priyank Jaini, Amur Ghose, and Pascal Poupart. "Prometheus: directly learning acyclic directed graph structures for sum-product networks". In: *International Conference on Probabilistic Graphical Models*. PGM. 2018, pp. 181–192 (cit. on p. 3).

[Jaini, Poupart, *et al.* 2018] Priyank Jaini, Pascal Poupart, and Yaoliang Yu. "Deep homogeneous mixture models: representation, separation, and approximation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio *et al.* Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/c5f5c23be1b71adb51ea9dc8e9d444a8-Paper.pdf (cit. on p. 17).

[Khosravi *et al.* 2019] Pasha Khosravi, Yitao Liang, YooJung Choi, and Guy Van den Broeck. "What to expect of classifiers? reasoning about logistic regression with missing features". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 2716–2724. DOI: 10.24963/ijcai.2019/377. URL: https://doi.org/10.24963/ijcai.2019/377 (cit. on p. 2).

[Kingma and Welling 2014] Diederik P. Kingma and Max Welling. "Auto-encoding variational bayes". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2014. URL: http://arxiv.org/abs/1312.6114 (cit. on p. 2).

[Kisa *et al.* 2014] Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. "Probabilistic sentential decision diagrams". In: *Knowledge Representation and Reasoning Conference* (2014) (cit. on p. 2).

[Koller and Friedman 2009] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. ISBN: 0262013193 (cit. on p. 2).

[Liang, Bekker, *et al.* 2017] Yitao Liang, Jessa Bekker, and Guy Van den Broeck. "Learning the structure of probabilistic sentential decision diagrams". In: *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence*. 2017 (cit. on p. 3).

[Liang and Van den Broeck 2019] Yitao Liang and Guy Van den Broeck. "Learning logistic circuits". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (July 2019), pp. 4277–4286. DOI: 10.1609/aaai.v33i01.33014277. URL: https://ojs.aaai.org/index.php/AAAI/article/view/4336 (cit. on p. 8).

[Lippe and Gavves 2021] Phillip Lippe and Efstratios Gavves. "Categorical normalizing flows via continuous transformations". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=-GLNZeVDuik (cit. on p. 2).

[Martens and Medabalimi 2014] James Martens and Venkatesh Medabalimi. "On the expressive efficiency of sum product networks". In: *CoRR* abs/1411.7717 (2014). arXiv: 1411.7717. URL: http://arxiv.org/abs/1411.7717 (cit. on p. 7).

[Denis D. Mauá *et al.* 2017] Denis D. Mauá, Fabio G. Cozman, Diarmaid Conaty, and Cassio P. Campos. "Credal sum-product networks". In: *Proceedings of the Tenth International Symposium on Imprecise Probability: Theories and Applications*. Ed. by Alessandro Antonucci, Giorgio Corani, Inés Couso, and Sébastien Destercke. Vol. 62. Proceedings of Machine Learning Research. PMLR, Oct. 2017, pp. 205–216. URL: https://proceedings.mlr.press/v62/mau%C3%A117a.html (cit. on p. 8).

[Mauro *et al.* 2021] Nicola Di Mauro, Gennaro Gala, Marco Iannotta, and Teresa M. A. Basile. "Random probabilistic circuits". In: *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*. 2021 (cit. on p. 3).

[Mei *et al.* 2018] Jun Mei, Yong Jiang, and Kewei Tu. "Maximum a posteriori inference in sum-product networks". In: *AAAI Conference on Artificial Intelligence*. 2018 (cit. on p. 12).

[Melibari *et al.* 2016] Mazen Melibari, Pascal Poupart, and Prashant Doshi. "Sum-product-max networks for tractable decision making". In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. IJCAI'16. New York, New York, USA: AAAI Press, 2016, pp. 1846–1852. ISBN: 9781577357704 (cit. on p. 8).

[Nath and P. M. Domingos 2016] Aniruddh Nath and Pedro M Domingos. "Learning tractable probabilistic models for fault localization". In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016 (cit. on p. 3).

[Ovadia *et al.* 2019] Yaniv Ovadia *et al.* "Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 2019 (cit. on p. 1).

[Papamakarios *et al.* 2021] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. "Normalizing flows for probabilistic modeling and inference". In: *Journal of Machine Learning Research* 22.57 (2021), pp. 1–64. URL: http://jmlr.org/papers/v22/19-1028.html (cit. on p. 2).

[Peharz, Gens, and P. Domingos 2014] Robert Peharz, Robert Gens, and Pedro Domingos. "Learning selective sum-product networks". In: *Workshop on Learning Tractable Probabilistic Models*. 2014 (cit. on p. 9).

[Peharz, Gens, Pernkopf, *et al.* 2016] Robert Peharz, Robert Gens, Franz Pernkopf, and Pedro Domingos. "On the latent variable interpretation in sum-product networks". In: *IEEE transactions on pattern analysis and machine intelligence* 39.10 (2016), pp. 2030–2044 (cit. on p. 6).

[Peharz, Lang, *et al.* 2020] Robert Peharz, Steven Lang, *et al.* "Einsum networks: fast and scalable learning of tractable probabilistic circuits". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 7563–7574. URL: https://proceedings.mlr.press/v119/peharz20a.html (cit. on pp. 3, 8).

[Peharz, Tschiatschek, *et al.* 2015] Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. "On theoretical properties of sum-product networks". In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*. 2015, pp. 744–752 (cit. on pp. 7, 11).

[Peharz, Vergari, *et al.* 2020] Robert Peharz, Antonio Vergari, *et al.* "Random sum-product networks: a simple and effective approach to probabilistic deep learning". In: *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*. Ed. by Ryan P. Adams and Vibhav Gogate. Vol. 115. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 334–344. URL: https://proceedings.mlr.press/v115/peharz20a.html (cit. on p. 3).

[Pevný *et al.* 2020] Tomáš Pevný, Václav Smídl, Martin Trapp, Ondřej Poláček, and Tomáš Oberhuber. "Sum-product-transform networks: exploiting symmetries using invertible transformations". In: *Proceedings of the 10th International Conference on Probabilistic Graphical Models*. Ed. by Manfred Jaeger and Thomas Dyhre Nielsen. Vol. 138. Proceedings of Machine Learning Research. PMLR, Sept. 2020, pp. 341–352. URL: https://proceedings.mlr.press/v138/pevny20a.html (cit. on p. 8).

[Poon and P. Domingos 2011] Hoifung Poon and Pedro Domingos. "Sum-product networks: a new deep architecture". In: *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*. 2011, pp. 337–346 (cit. on pp. 2, 3, 6).

[Rahman *et al.* 2014] Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate. "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*. 2014, pp. 630–645 (cit. on p. 2).

[Rashwan *et al.* 2018] Abdullah Rashwan, Pascal Poupart, and Chen Zhitang. "Discriminative training of sum-product networks by extended baum-welch". In: *Proceedings of the Ninth International Conference on Probabilistic Graphical Models*. Vol. 72. Proceedings of Machine Learning Research. 2018, pp. 356–367 (cit. on p. 2).

[Rezende and Mohamed 2015] Danilo Rezende and Shakir Mohamed. "Variational inference with normalizing flows". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1530–1538. URL: https://proceedings.mlr.press/v37/rezende15.html (cit. on p. 2).

[Rolfe 2017] Jason Tyler Rolfe. "Discrete variational autoencoders". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=ryMxXPFex (cit. on p. 2).

[Rooshenas and Lowd 2016] Amirmohammad Rooshenas and Daniel Lowd. "Discriminative structure learning of arithmetic circuits". In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. Ed. by Arthur Gretton and Christian C. Robert. Vol. 51. Proceedings of Machine Learning Research. Cadiz, Spain: PMLR, Sept. 2016, pp. 1506–1514. URL: https://proceedings.mlr.press/v51/rooshenas16.html (cit. on p. 3).

[Shao *et al.* 2020] Xiaoting Shao *et al.* "Conditional sum-product networks: imposing structure on deep probabilistic architectures". In: *Proceedings of the 10th International Conference on Probabilistic Graphical Models*. Ed. by Manfred Jaeger and Thomas Dyhre Nielsen. Vol. 138. Proceedings of Machine Learning Research. PMLR, Sept. 2020, pp. 401–412. URL: https://proceedings.mlr.press/v138/shao20a.html (cit. on p. 3).

[Sharir and Shashua 2018] Or Sharir and Amnon Shashua. "Sum-product-quotient networks". In: *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*. Ed. by Amos Storkey and Fernando Perez-Cruz. Vol. 84. Proceedings of Machine Learning Research. PMLR, Sept. 2018, pp. 529–537. URL: https://proceedings.mlr.press/v84/sharir18a.html (cit. on p. 8).

[Su *et al.* 2019] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. "One pixel attack for fooling deep neural networks". In: *IEEE Transactions on Evolutionary Computation* 23.5 (2019), pp. 828–841 (cit. on p. 1).

[Szegedy *et al.* 2013] Christian Szegedy *et al.* "Intriguing properties of neural networks". In: *arXiv preprint arXiv:1312.6199* (2013) (cit. on p. 1).

[Vahdat, Andriyash, *et al.* 2018] Arash Vahdat, Evgeny Andriyash, and William Macready. "Dvae#: discrete variational autoencoders with relaxed boltzmann priors". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio *et al.* Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/9f53d83ec0691550f7d2507d57f4f5a2-Paper.pdf (cit. on p. 2).

[Vahdat, Macready, *et al.* 2018] Arash Vahdat, William G. Macready, Zhengbing Bian, Amir Khoshaman, and Evgeny Andriyash. "DVAE++: discrete variational autoencoders with overlapping transformations". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 5042–5051. URL: http://proceedings.mlr.press/v80/vahdat18a.html (cit. on p. 2).

[Vergari, Choi, Liu, *et al.* 2021] Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. "A compositional atlas of tractable circuit operations: from simple transformations to complex information-theoretic queries". In: *CoRR* abs/2102.06137 (2021). arXiv: 2102.06137 (cit. on pp. 8, 16).

[Vergari, Choi, Peharz, *et al.* 2020] Antonio Vergari, YooJung Choi, Robert Peharz, and Guy Van den Broeck. *Probabilistic Circuits: Representations, Inference, Learning and Applications.* AAAI Tutorial. 2020 (cit. on p. 1).

[Vergari, Mauro, *et al.* 2015] Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. "Simplifying, regularizing and strengthening sum-product network structure learning". In: *ECML/PKDD*. 2015 (cit. on p. 3).

[Wei *et al.* 2018] Wenqi Wei *et al.* "Adversarial examples in deep learning: characterization and divergence". In: *arXiv preprint arXiv:1807.00051* (2018) (cit. on p. 1).

[Yu 2020] Ronald Yu. *A Tutorial on VAEs: From Bayes' Rule to Lossless Compression.* 2020. arXiv: 2006.10273 [cs.LG] (cit. on p. 2).

[Zhang *et al.* 2021] Honghua Zhang, Brendan Juba, and Guy Van Den Broeck. "Probabilistic generating circuits". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 12447–12457. url: https://proceedings.mlr.press/v139/zhang21i.html (cit. on p. 8).

[Zhao, Adel, *et al.* 2016] Han Zhao, Tameem Adel, Geoff Gordon, and Brandon Amos. "Collapsed variational inference for sum-product networks". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, June 2016, pp. 1310–1318. url: http://proceedings.mlr.press/v48/zhaoa16.html (cit. on p. 9).

[Zhao, Melibari, *et al.* 2015] Han Zhao, Mazen Melibari, and Pascal Poupart. "On the relationship between sum-product networks and Bayesian networks". In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37. Proceedings of Machine Learning Research. 2015, pp. 116–124 (cit. on p. 9).

[Ziegler and Rush 2019] Zachary M. Ziegler and Alexander M. Rush. "Latent normalizing flows for discrete sequences". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 7673–7682. url: http://proceedings.mlr.press/v97/ziegler19a.html (cit. on p. 2).