Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models*

YooJung Choi

Antonio Vergari

Guy Van den Broeck
Computer Science Department
University of California
Los Angeles, CA, USA

Contents

*. This is a truncated version of the paper up to Section 9. Future sections will be released incrementally.

1. Introduction

Probabilistic models are at the very core of modern machine learning (ML) and artificial intelligence (AI). Indeed, probability theory provides a principled and almost universally adopted mechanism for decision making in the presence of uncertainty. For instance, in machine learning, we assume that our data was drawn from an unknown probability distribution. Getting access to this distribution, in any of its facets, is the "holy grail" of statistical ML. It would reduce many machine learning tasks to simply performing *probabilistic inference*. Similarly, many forms of model-based AI seek to directly represent the mechanism that governs the world around us as a probability distribution in some form.

It is therefore no wonder that much attention in ML has been devoted to *learning the distribution back from the data*. We fit more and more expressive probabilistic models as density estimators that are increasingly close to the data-generating distribution. This approach was popularized recently by progress in deep generative models such as generative adversarial networks (Goodfellow et al., 2014), variational autoencoders (Rezende et al., 2014; Kingma and Welling, 2013) and normalizing flows (Papamakarios et al., 2019). At the same time, increasingly *rich and expressive modeling languages* that can concisely capture complex distributions have been developed through efforts in statistics (Carpenter et al., 2017), programming languages (Holtzen et al., 2020), cognitive science (Griffiths et al., 2010) and AI (Milch et al., 2005; Domingos and Lowd, 2009; Fierens et al., 2015).

However, the increased expressiveness of these probabilistic models, and the ability of modern neural density estimators of scaling learning to large amounts of data comes at a tremendous price: the inability to perform *reliable* and *efficient* probabilistic inference in all but the most trivial of probabilistic reasoning scenarios. Concretely, these aforementioned models resort to various approximation techniques for answering basic questions about the probability distributions they represent. Computing a marginal or conditional probability, an expectation, or the mode of the distribution can only be done through approximations with little to no guarantees. Ironically, as our models get closer to fitting the true distribution with high fidelity, we are also getting further away from our goal of solving problems by probabilistic reasoning, to some extent nullifying the very purpose of probabilistic modeling and learning.

This state of probabilistic generative models stands in stark contrast with the state of the field at its nascence. In a ground-breaking decade, the 1960s saw the introduction of the hidden Markov model (HMM) (Stratonovich, 1960; Baum and Petrie, 1966), the Kalman filter (Kalman, 1960), early applications of naive Bayes classifiers (Bailey, 1965; Boyle et al., 1966), and the Chow-Liu tree learning algorithm (Chow and Liu, 1968). These classical probabilistic models have clear limitations: they are nowhere near as expressive as the models available today. Yet they came with one distinctive and important virtue: efficient algorithms for probabilistic reasoning. They were ***tractable*** probabilistic models that would go on to support scientific and engineering breakthroughs for decades to come.

A trend in probabilistic AI and ML is to focus on designing and exploiting models that can theoretically guarantee reliable and efficient probabilistic inference. These models often go under the umbrella name of *tractable probabilistic models* (TPMs) and allow for complex inference routines to be computed *exactly* and in *polynomial time*. Examples of "classical" TPMs are Kalman filters (Musoff and Zarchan, 2009) and hidden Markov models (Koller

and Friedman, 2009), tree distributions (Chow and Liu, 1968), and bounded-treewidth PGMs (Bach and Jordan, 2002). Although extensively used not only in AI and ML but also control theory, system engineering and statistics, these models are deemed to be be limited in expressive power. More recently, a burgeoning new wave of TPMs has arrived, promising an increase in expressive power and efficiency, with little to no compromise in tractability. These include models such as arithmetic circuits (Darwiche, 2003), probabilistic decision graphs (Jaeger, 2004), and-or search spaces (Marinescu and Dechter, 2005) and multi-valued decision diagrams (Dechter and Mateescu, 2007), sum-product networks (Poon and Domingos, 2011), cutset networks (Rahman et al., 2014) and probabilistic sentential decision diagrams (Kisa et al., 2014).

In this work, we lay the foundations to describe, learn, and reason about these TPM formalisms under a single unified framework, which we name *probabilistic circuits* (PCs). PCs 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). They are expressive deep generative models as they indeed encode several layers of latent variables into large graphs with millions of connections and parameters. Differently from the intractable neural estimators mentioned above, however, PCs allow for exact probabilistic inference in time linear in the size of the circuits and the cost of performing it can be *theoretically certified when the circuit has certain structural properties*.

Specifically, we make the following contributions. First, we introduce the framework of PCs as a unifying theoretical and practical tool that generalizes many previously introduced TPM models, while abstracting away from their syntactic differences. Second, we formalize and systematize many tractable probabilistic inference tasks as classes of functions, named *probabilistic queries*, providing a useful abstraction to talk about the desiderata for real-world probabilistic inference as well as to compare model classes w.r.t. their inference capabilities. Third, we provide precise characterizations of when these query classes can be efficiently computed on PCs, in terms of the presence of certain structural properties in their computational graphs. Lastly, we collect previous connections and draw novel ones between PCs and other representations such as polynomials, hierarchical mixture models, tensor factorizations, and logical circuits, ultimately questioning whether all tractable representations could be represented as PCs and under which conditions.

The rest of the paper is organized as follows. Section 2 introduces the necessary background from probability theory and formalizes the notions of probabilistic query classes and tractable representations. Section 3 builds the PC framework from the ground up and discusses the computation of complete evidence queries with PCs. Marginal inference and related query classes are formalized in Section 4, while also introducing the class of smooth and decomposable PCs as tractable representations for them. Before discussing other query classes of interest, Section 5 draws connections between PCs and other representations such as (hierarchical) mixture models and (multi-linear) polynomials. The tasks of computing the modes of distributions encoded by PCs are discussed in Sections 6 and 8. Section 7 discusses the notion of expressive efficiency and compares tractable PCs under this notion. Section 9 introduces advanced query classes involving pairs of PCs, including the computations of expectations between PCs and metrics to quantify the distance between two PC distributions such as the Kullback-Leibler divergence. Later, Section 10 introduces and

discusses transformations over probability distributions encoded as PCs, and Section 11 provides the reader a compendium to translate the most popular TPM formalisms to PCs, while discussing which structural properties—and hence tractable query classes—are carried over. Finally, before concluding in Section 13, in Section 12 we trace the tractable representations in the AI and ML literature back from logical circuits to generalizations to other semi-rings.

2. Probabilistic Inference: Models, Queries, and Tractability

Probabilistic circuits are ***probabilistic models*** that are ***tractable*** for large classes of ***queries***. This section provides the necessary background to understand those key concepts. First, we discuss how probabilistic models are compact representations of probability distributions and introduce the probability notation used throughout this paper. Second, we formalize probabilistic inference as computing quantities of interest by querying probabilistic models. We will then categorize these queries into families, which will help us characterize their computational differences. Third, we formally define what it means for a family of queries to be tractable. We end this section by stating the scientific questions that arise in this context and that will be answered in the remainder of this paper: for example, what makes a probabilistic model tractable for a family of queries, and what is the price one has to pay for this tractability?

2.1 Probabilistic Models

Probability theory offers a principled way to model and reason about the uncertainty over the world we observe. Next, we briefly refresh probability calculus and its notation.[1]

The world is described in terms of its attributes (or features). Since we have uncertainty about their values, we consider these attributes to be *random variables* (RVs). We denote RVs by uppercase letters (X, Y), and denote sets of RVs by bold uppercase letters $(\mathbf{X}, \mathbf{Y})$. The domain of a RV X is the set of possible values that it can assume, denoted by $\mathsf{val}(X)$. Values of RVs are denoted by a lowercase letter (x, y). When the RV is clear from context, we will abbreviate assignments of the form $X = x$ by simply writing x.

A *state* of the world (or a configuration, possible world, complete state) assigns a value to each RV. A *partial state* assigns a value to some RVs. We denote partial or joint states by bold lowercase letters ($\mathbf{Y} = \boldsymbol{y}$, or simply $\boldsymbol{y}$). We will assume that all states $\boldsymbol{x}$ and sets of RVs $\mathbf{X}$ are indexed by a subscript, i.e., x_i and X_i. The *state space* $\mathsf{val}(\mathbf{X})$ of a set of n RVs $\mathbf{X}$ consists of all possible states $\mathsf{val}(X_1) \times \ldots \times \mathsf{val}(X_n)$.

A *joint probability distribution* over RVs $\mathbf{X}$, denoted by $p(\mathbf{X})$, quantifies the uncertainty over states of $\mathbf{X}$. When the set of variables $\mathbf{X}$ can be partitioned into subsets $\mathbf{Y}, \mathbf{Z}$, we can equivalently write $p(\mathbf{Y}, \mathbf{Z})$. Formally, the state space of our RVs forms a *sample space*. When we define a σ-algebra of events over this sample space, we obtain a *measurable space*. Our joint probability distribution then corresponds to a *probability measure* associated with such a measurable space, resulting in a *probability space*.

1. We refer the reader to Rosenthal (2006); Feller (2008); Koller and Friedman (2009) for additional background and an in-depth treatment of probability theory.

Distributions over discrete RVs are described by *probability mass functions* (PMFs), whereas distributions over continuous RVs are described by *probability density functions* (PDFs). Mixed discrete-continuous distributions are used when dealing with both kinds of RVs. To lighten the notation, we let the function $p(\mathbf{X} = \boldsymbol{x})$ return either a probability or a probability density, depending on context. As most of the statements in this work hold in either scenario, we only make the distinction explicit when necessary.

A ***probabilistic model*** is a particular representation of a probability distributions. For a probabilistic model m that has parameters θ, we will use either $p_{\mathsf{m}}(\mathbf{X})$ or $p_\theta(\mathbf{X})$ to denote the probability distribution that is represented by the probabilistic model.

Events are sets of states that are assigned a probability. In discrete distributions, each (partial) state describes a basic event. Continuous random variables require more care: their partial assignments do not generally have a probability, only a probability density. Thus, to specify an event for a general RV X, we will say that its value comes from an interval $\mathcal{I}$, denoted $X \in \mathcal{I}$. We will write $\mathbf{X} \in \boldsymbol{\mathcal{I}}$ to mean that each X_i in $\mathbf{X}$ has a value from interval $\mathcal{I}_i$. More complex types of events over multiple RVs—those that need to be described in a formal logical language—are discussed in detail in Section 9.

We will assume familiarity with the standard transformations and rules of probability that turn one distribution into another. For example, *marginalization* (or summing out, integrating out) removes a variable from the scope of the distribution. The transformation of *conditioning* removes all probability from states that do not conform to an observed event, often called *evidence*, and re-normalizes all other probability mass accordingly.

2.2 Probabilistic Queries

Intuitively, a probabilistic model can be seen as a *black box* that we can ask questions about the uncertainty around some states and events in the joint probability distribution. These questions involve computing some quantities of interest of the joint probability distribution, for instance the probability mass or density associated with an observed state, the mode of the distribution, one of its moments, its entropy, etc.

Such questions are called ***queries*** in the computer science literature; a term commonly used in databases (Vardi, 1982; Suciu et al., 2011; Van den Broeck et al., 2017), probabilistic graphical models (Koller and Friedman, 2009; Dechter, 2019), and knowledge representation (Cali et al., 2010; Juba, 2019). Queries usually ask for quantities of interest after transforming the distribution in some way. For example, one might ask for the mode of the distribution after conditioning it on evidence and marginalizing out some variables.

Consider the following simple example of decision making under uncertainty.

Example 1 (Traffic jam distribution) *Imagine being a commuter in Los Angeles who needs to decide which route to take to work each day. To avoid traffic jams you could query the probabilistic model* m *embedded in your navigation software. That is, the probabilistic model* m *represents a joint probability distribution* $p_{\mathsf{m}}(\mathbf{X})$ *over RVs* $\mathbf{X} = \{\mathsf{W}, \mathsf{T}, \mathsf{J}_{\mathsf{str}_1}, \ldots, \mathsf{J}_{\mathsf{str}_K}\}$. *Here,* R *is a categorical RV with domain* $\mathsf{val}(\mathsf{W}) = \{\mathsf{Rain}, \mathsf{Snow}, \ldots, \mathsf{Sun}\}$ *for the* W*eather;* T *is a continuous RV with domain* $\mathsf{val}(\mathsf{T}) = [0, 24)$ *indicating the* T*ime of day; and* $\{\mathsf{J}_{\mathsf{str}_i}\}_{i=1}^{K}$ *is a set of binary RVs, each indicating the presence of a traffic jam on the* i*-th street.*

You might want to ask your navigator the following probabilistic inference queries.

q_1: **What is the probability that there is a traffic jam on Westwood Blvd. and that the weather is rainy?**

q_2: **Which time of day is most likely to have a traffic jam on my way to work?**

Both queries are a function of the distribution p_{m} represented by the probabilistic model m inside of your navigator. For instance, the result of query q_1 is the probability mass of the partial state that assigns $\mathsf{W} = \mathsf{Rain}$ and $\mathsf{J}_{\mathsf{Westwood}} = 1$:

$$p_{\mathsf{m}}(\mathsf{W} = \mathsf{Rain}, \mathsf{J}_{\mathsf{Westwood}} = 1). \tag{1}$$

The result of query q_2 is the mode of the distribution over time T after conditioning the distribution on a complex event, and marginalizing out all other variables:

$$\arg\max_{\mathsf{t}} \; p_{\mathsf{m}}\big(\mathsf{T} = \mathsf{t} \;\big|\; \textstyle\bigvee_{i \in \mathsf{route}} \mathsf{J}_{\mathsf{str}_i}\big)\,.$$

Here, $\bigvee_{i \in \mathsf{route}} \mathsf{J}_{\mathsf{str}_i}$ is the event that at least one of the roads on my route to work is jammed.

As one might intuit, computing query q_2 must be at least as hard as computing query q_1: both queries marginalize the distribution, but q_2 also performs maximization while dealing with events that are more complex logical constraints, not just partial states. By looking at the types of transformations and distributional quantities computed by a query, it becomes possible to group queries that have similar characteristics into *query classes*. They will allow us to identify queries that present similar computational challenges, and to formally define useful classes of tractable probabilistic models.

The following example query is representative of a simple but important query class, called *complete evidence* queries (EVI).

Example 2 (EVI query) *Consider again the traffic jam distribution $p_{\mathsf{m}}(\mathbf{X})$ introduced in Example 1. The query "What is the probability that at 12 o'clock on a sunny day there will be a traffic jam only on Westwood Blvd.?" is answered by computing*

$$p_{\mathsf{m}}(\mathsf{W}{=}\mathsf{Sun}, \mathsf{T}{=}12.0, \mathsf{J}_{\mathsf{str}_1}{=}0, \ldots, \mathsf{J}_{\mathsf{Westwood}}{=}1, \ldots, \mathsf{J}_{\mathsf{str}_K}{=}0),$$

where among the K roads, only the traffic jam RV $\mathsf{J}_{\mathsf{Westwood}}$ is set to 1 and all others to 0.

Definition 1 (EVI query class) *The class of complete evidence queries (EVI) consists of all queries that compute $p(\mathbf{X} = \boldsymbol{x})$, where p is a joint probability distribution over RVs $\mathbf{X}$, and $\boldsymbol{x} \in \mathsf{val}(\mathbf{X})$ is a complete state (also called complete evidence).*

Crucially, the EVI query's state $\boldsymbol{x}$ is complete – it assigns a value to each RV in the distribution. For a given model m, answering an EVI query corresponds to computing the complete *likelihood* of that model m given the example $\mathbf{x}$. As such, EVI queries are at the core of maximum-likelihood learning of probabilistic models (Koller and Friedman, 2009).

This paper will present a rich landscape of query classes that goes far beyond the EVI class. For instance, the query q_1 in Example 1 is an instance of a harder query class, called *marginal* queries (MAR), whose goal is to compute the probability (or density) of a *partial* state in the distribution p. We will formally define MAR and discuss its properties in Section 4. Moreover, we will show that its computational challenges are shared with other

interesting query classes such as *conditional* queries (CON) and computing the *moments* of a distribution (MOM) (e.g., its mean or variance).

Instead of the moments of a distribution, we might be interested in its *mode*, that is, the most likely complete state after conditioning the distribution on evidence given by a partial state. Queries of this kind are called *maximum a posteriori* queries (MAP) and require different computational tools than MAR, as we will discuss in Section 6.[2]

Complex decision making in the real-world might require even more sophisticated probabilistic inference routines than EVI, MAR, or MAP queries. The *marginal MAP* (MMAP) query class combines aspects of both MAR and MAP – it requires marginalization over one set of RVs and finding the most likely partial state over another set of RVs. We will discuss the MMAP class in Section 8 while relating it to other difficult classes; in particular the information-theoretic query class of computing the (marginal) *entropy* of a distribution.

The last class of probabilistic queries we will touch upon in this work deal with properties of more than one single model. Such queries ask about the relationship between a distribution and another complex object, which could be a second distribution or some complex event or function. We place these queries under the umbrella class of *pairwise* queries (PAIR), as they share many of the same computational properties. Examples of PAIR queries discussed in Section 9 include the computation of the *Kullback-Leibler divergence* (KLD) between two distributions, the *expectation* (EXP) of a function and the probability of a *complex logical event* (PR), such as the traffic jam event in query q_2 from Example 1.

2.3 Tractable Probabilistic Inference

When we say that a probabilistic model is ***tractable***, we are expecting it to provide two types of guarantees. The first guarantee is that the model is able to perform *exact* inference: the answers to queries are faithful to the model's distribution, and no approximations are involved in obtaining them.[3] The second guarantee is that the query computation can be carried out *efficiently*, that is, in time polynomial in the size of the probabilistic model.

Definition 2 (Tractable probabilistic inference) *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* $\mathcal{O}(poly(|m|))$*. We also say that* $\mathcal{M}$ *is a tractable model for* $\mathbf{Q}$*.*

In Definition 2, the concept of efficiency translates to polytime complexity w.r.t. the *size* of models in a class, $|m|$. Model size can be defined differently for different model classes, but in all cases represents a proxy for the number of computations give the size of the model's input. For classical probabilistic graphical models like Bayesian networks (BNs) and Markov random fields (MRFs), model size can be expressed in terms of the size of their factors (Darwiche, 2009; Koller and Friedman, 2009). For models represented as computational graphs, such as in neural density estimators (Papamakarios et al., 2019) and

2. MAP queries are also called most-probable explanation queries (MPE) in the Bayesian network literature.
3. As such, while probabilistic circuits are a form of deep generative model, this paper will not be considering other deep generative models like GANs (Goodfellow et al., 2014) and RAEs (Ghosh et al., 2019), which do not have an explicit probability distribution, or VAEs (Kingma and Welling, 2013) which do have a well-defined density, but cannot compute it exactly and need to resort to variational approximations.

our probabilistic circuits, 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 one model size. However, for some advanced query classes like PAIR which involve more than one model, we would need to express the dependency w.r.t. all their sizes. In particular, for certain queries in the PAIR class such as computing the probability of a complex event involving disjunctions of simpler event, one of the models involved will be a compact representation for such an event (e.g., compiled into a logical circuit, cf. Section 12). In those cases we will refer to the size of the model encoding the query event, as the *size of the query* and denote it as $|q|$.

2.4 Properties of Tractable Probabilistic Models

It is important to observe that Definition 2 does not state tractability as an absolute property. Tractability can be defined for a family of models only *w.r.t. a class of queries*: ***tractability is a spectrum***. Indeed, a tractable representation for one query class might not admit polynomial time inference for another query class. For a model class, we define its *tractable band* as the set of query classes for which the model class is a tractable representation.

Example 3 (Tractable bands for Bayesian networks) *Let $\mathcal{M}^{\mathbf{BN}}$ be the class of Bayesian Networks over collections of discrete RVs. Then $\mathcal{M}^{\mathbf{BN}}$ is a tractable representation for* EVI *(cf. Section 2.2) since all complete evidence queries can be computed in time linear in the number of RVs considered. However $\mathcal{M}^{\mathbf{BN}}$ is not a tractable representation for* MAR*,* MAP*, nor* MMAP *since computing queries from these classes are respectively #P-complete (Cooper, 1990; Roth, 1996), NP-Hard and NP^{PP}-complete (Park and Darwiche, 2004) problems.*

From this perspective, different model classes can be compared, and ranked by their usefulness for certain application domains, by the extent of their tractable bands. Moreover, when it comes to different model classes with the same tractable bands, it is natural to question whether there are some common aspects in their representations that are sufficient or necessary to support tractable inference for those query classes.

In the next section by introducing the framework of probabilistic circuits (PCs) we will provide a positive answer for many tractable representations. Specifically, the clear operational semantics of PCs will help i) homogenize representation and notation for many tractable representations and ii) provide a clean way to trace the tractable bands of a model class by some structural properties they conform to. Table 1 characterizes the family of probabilistic circuits supporting tractable inference of each query class by their structural properties, as will be shown in future sections.

3. Probabilistic Circuits: Representation

We introduce probabilistic circuits (PCs) as a general and unified computational framework for tractable probabilistic modeling.

This serves two major purposes. The first one is to unify the disparate formalisms proposed so far in the literature for tractable models. Probabilistic circuits reconcile

Table 1: Tractable inference of probabilistic circuits. Rows denote the following query classes: marginal (MAR) and conditional (CON) inference, moments of a distribution (MOM), maximum a-posteriori (MAP) and marginal maximum a-posteriori (MMAP) inference, expectations (EXP) and Kullback-Leibler divergence (KLD). Marks signify that a query class can be computed tractably given certain structure properties (columns): smoothness (Smo.), consistency (Con.), decomposability (Dec.), structured-decomposability (Str.Dec.), determinism (Det.), and marginal determinism (Mar.Det.).

Query Class	Smo.	Con.	Dec.	Str.Dec.	Det.	Mar.Det.	Reference
MAR	✓		✓				Section 4.2
CON	✓		✓				Section 4.2
MOM	✓		✓				Section 4.3
MAP		✓			✓		Section 6
MMAP	✓		✓			✓	Section 8
EXP	✓		✓	✓			Section 9.2
KLD	✓		✓	✓	✓		Section 9.1

and abstract from the different graphical and syntactic formalisms of recently introduced models such as arithmetic circuits (Darwiche, 2003), probabilistic decision graphs (Jaeger, 2004), and-or search spaces (Marinescu and Dechter, 2005) and multi-valued decision diagrams (Dechter and Mateescu, 2007), sum-product networks (Poon and Domingos, 2011), cutset networks (Rahman et al., 2014) and probabilistic sentential decision diagrams (Kisa et al., 2014). Additionally, more classical tractable models such as treewidth-bounded probabilistic graphical models can be naturally cast in the probabilistic circuits framework (Darwiche, 2009). We provide a vocabulary of translations from all these formalisms into PCs in Section 11.

The second purpose of the PC framework is to enable reasoning over the tractable bands of a model class in terms of some well-defined structural properties only. In turn, this allows for a deeper theoretical understanding of which properties are necessary or sufficient for tractable probabilistic representations at large. We introduce and discuss these properties in the context of different query classes in Sections 4-9 and we question if all tractable representations can be cast as PCs in Section 11.7.

In this section, we build the PC framework first in a bottom-up and intuitive fashion by introducing the building blocks of a grammar that PCs provide for tractable probabilistic modeling. Later, we consolidate these notions in a more formal, top-down introduction.

3.1 The Ingredients of Tractable Probabilistic Modeling

Probabilistic circuits encode joint probability distributions in a recursive way, by means of a graph formalism. In essence, probabilistic circuits are *computational graphs* encoding functions that characterize a distribution, for instance a PMF or a PDF. By evaluating such a function w.r.t. to some inputs, a PC will encode the computation to answer certain probabilistic queries, that is, to perform inference. We now introduce the minimal set of computational units needed to build such graphs: *distribution*, *product* and *sum* units.

3.1.1 Input Units: Simple Tractable Distributions

To begin, consider the smallest computational graph of this kind, consisting of a single computational unit. This single unit can represent a whole probability distribution over a bunch of RVs. We name it *distribution unit*, and later we will refer to it as *input unit* as it will constitute the inputs of a whole PC. The computation encoded in such a unit, i.e., the output it emits given some input, is determined by the query class considered and parametric form of the distribution it encodes.

Example 4 (Tractable densities as computational units) *Consider a distribution unit encoding a Gaussian density* $p(X) = \mathcal{N}(X; \mu = 1, \sigma = 0.1)$ *as represented on the left as a circle and labeled by its RV. Then to answer some* EVI *query, when it is fed some observed state* $X = 1.1$ *as evidence (orange), it will output the corresponding PDF* $\mathcal{N}(X = 1.1; 1, 0.1) \approx 2.41$ *(blue):*

Since a computational node defined in this way effectively acts as a black-box encapsulating a distribution function, this formalism is quite versatile. First, we do not need to switch node type to answer queries from different classes. It would suffice to evaluate the encoded distribution accordingly: e.g., evaluating a pointwise density for EVI as in Example 4, marginalizing it over some interval for MAR, or returning its mode to answer MAP queries. Second, we can plug any out-of-the-box probability distribution as long as it is a tractable representation for the query class at hand. Moreover, note that we are not limited to normalized distributions, we just need the function encoded into a input unit to be non-negative and assume it to be tractable for MAR to readily obtain its partition function.

Among the distributions with large tractable bands that can readily be represented by a single distribution unit are the most commonly used univariate distributions, e.g., Bernoulli, Categorical, Gaussian, Poisson, Gamma, and other exponential families. Computing EVI, MAR and MAP queries for them can be done analytically by design.[4] Distribution units are not limited to univariate distributions, however, as many of the above families retain tractability when extended to the multivariate case. Consider for instance the omnipresent multivariate Gaussian distribution over RVs $\mathbf{X}$. It still retains tractable conditioning and marginalization in time cubic in $|\mathbf{X}|$ and constant-time maximization by design (the mean is the mode).

As we can assume to always be able to encode simple distributions in single units this way, distribution units constitute the base case of our recursive scheme to build PCs. The other two units we introduce next provide the inductive step by allowing to compose more and more complex PCs together.

4. When dealing with generalized MAR queries as in Definition 11 we will need to access the corresponding cumulative distribution function (CDF) for these distributions. Sometimes this might not be computable in closed form, as for Gaussians, however efficient accurate approximations will be available.

3.1.2 Product Units: Independent Factorizations

Perhaps the simplest way to decompose a joint distribution into smaller ones is to have it *factorize*; that is, to treat each smaller distribution to be independent from the others.

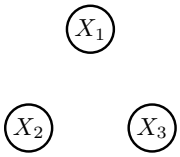
Definition 3 (Factorized models) *Consider the probabilistic model* m *encoding a joint probability distribution over a collection of RVs* $\mathbf{X} = \bigcup_{i=1}^{k} \mathbf{X}_i$ *partitioned into disjoint sets* $\mathbf{X}_i \cap \mathbf{X}_j = \emptyset$ *for any* $i \neq j$ *in* $1, \ldots, k$ *where* $k > 1$*. Model* m *is a factorized model iff*

$$p_{\mathsf{m}}(\mathbf{X}) = \prod_{i=1}^{k} p_{\mathsf{m}_i}(\mathbf{X}_i)$$

where each p_{m_i} *is a distribution over the subset of RVs* $\mathbf{X}_i$*.*

Having a joint distribution decomposing into smaller factors is the backbone assumption made by classical PGMs, where the way in which the joint factorizes is dictated by the dependency structure among the RVs, encoded in a graph formalism (Koller and Friedman, 2009). Among these, the simplest class of factorized models comprises *fully-factorized distributions*, where all RVs in the graph are disconnected; i.e., the joint distribution factorizes into univariate marginal distributions.

Example 5 (Fully factorized distributions) *Consider a multivariate Gaussian* $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ *over RVs* X_1, X_2, X_3 *with mean* $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3)$ *and diagonal covariance matrix* $\Sigma = \mathsf{diag}(\sigma_1, \sigma_2, \sigma_3)$*, whose graphical model is shown on the left.*

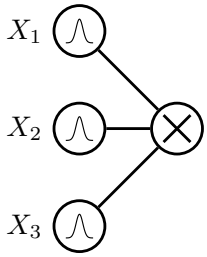


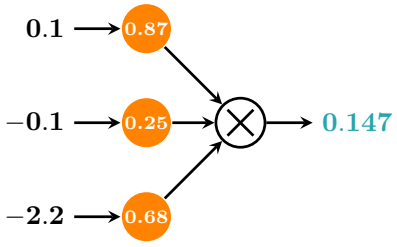
$$\begin{aligned} p(X_1, X_2, X_3) &= p(X_1) \cdot p(X_2) \cdot p(X_3) = \\ &= \mathcal{N}(\mu_1, \sigma_1) \cdot \mathcal{N}(\mu_2, \sigma_2) \cdot \mathcal{N}(\mu_3, \sigma_3) \end{aligned}$$

Then its joint density $p(X_1, X_2, X_3)$ *can be fully factorized as shown above on the right.*

To represent a fully-factorized model as a computational graph we just need to introduce a computational unit that performs *a product* over some input distribution units.

Example 6 (Factorizations as product units) *Consider the factorized multivariate Gaussian shown in Example 5. Then the computational graph below on the left encodes its joint distribution. It comprises three input units, each modeling a univariate Gaussian* $\mathcal{N}(\mu_i, \sigma_i)$ *over RV* X_i *for* $i = 1, 2, 3$*, that feed a product unit.*





To evaluate an EVI query $p(x_1, x_2, x_3)$ (in blue), the output of the product unit is obtained by multiplying the outputs of the input units (in orange), $p(X_i = x_i)$ when evaluated for a certain complete state $\boldsymbol{x} = \{x_1, x_2, x_3\}$. An example of the computations flowing through the graph is shown above on the right for $\boldsymbol{\mu} = \{0, 1, -2\}$ and $\Sigma = \mathsf{diag}(0.2, 0.5, 0.3)$ and for the state $\boldsymbol{x} = \{0.1, -0.1, -2.2\}$.

Factorizations as product units will be pivotal in several tractable inference scenarios in the following sections, as they suggest a divite-et-impera strategy to perform inference: "breaking down" complex inference problems into a collection of smaller ones. However, in order to represent a factorized, but not fully-factorized, model—hence a potentially more expressive model—we need product units to receive inputs not only from distribution units, but also from a different kind of unit: *sums*. Sum units will in fact help model correlations between factors.

3.1.3 Sum Units: Mixture Models

The idea to combine multiple simple distributions into a single model with increased expressiveness is at the core of *mixture models* (McLachlan and Peel, 2004). Here we focus on finite mixture models[5] with positive weights[6] defined as follows.

Definition 4 (Mixture models) *Let $\{p_{\mathsf{m}_i}\}_{i=1}^k$ a finite collection of probabilistic models, each defined over the same collection of RVs $\mathbf{X}$. A mixture model is the probabilistic model defined as the convex combination*

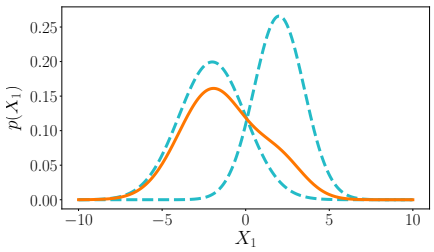
$$p_{\mathsf{m}}(\mathbf{X}) = \sum_{i=1}^{k} \theta_i p_{\mathsf{m}_i}(\mathbf{X})$$

for a set of positive weights (called the mixture parameters) $\theta_i > 0$, $i = 1, \ldots, k$ and $\sum_{i=1}^k \theta_i = 1$.

For continuous RVs, it is very well known that a sufficiently large *Gaussian mixture model* (GMM) can approximate any continuous PDF arbitrarily well (Kostantinos, 2000).

Example 7 (Gaussian mixture models) *Consider the mixture model (orange) of two univariate Gaussians $\mathcal{N}(\mu_1 = -2, \sigma_1 = 2)$ and $\mathcal{N}(\mu_2 = 2, \sigma_2 = 1.5)$ (blue, dashed) as depicted on the left.*

5. Mixture models, especially in the Gaussian case, have been investigated by considering an infinite but countable (Rasmussen, 2000) or uncountable (MacKay, 1995) number of components. The latter case has been recently popularized by deep generative models with continuous latent variables such as variational autoencoders (Kingma and Welling, 2013).
6. Mixtures with positive weights are also called *monotonic* and constitute the norm. While more exotic, non-monotonic mixture models that guarantee densities that are always positive, can offer an exponential saving in the number of components needed to represent a target distribution (Valiant, 1979b). That is, non-monotonic mixtures are more expressive efficient than monotonic ones.

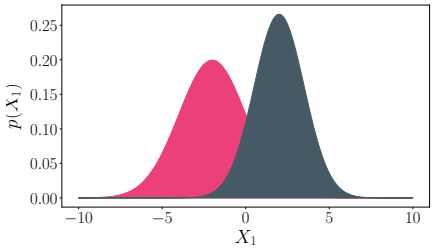


$$p(X_1) = \theta_1 p_1(X_1) + \theta_2 p_2(X_1) = \\ = \theta_1 \mathcal{N}(\mu_1, \sigma_1) + \theta_2 \mathcal{N}(\mu_2, \sigma_2)$$

Then its joint density $p(X_1)$ can be expressed as the weighted sum on the right for the two positive real weights $\theta_1 = 0.8$ and $\theta_2 = 0.2$. Note that the mixture density is more expressive than its components. In fact it captures two modes, something that is not possible doing with the two univariate Gaussian components taken singularly.

The fact that even a mixture model over components encoding fully-factorized models can capture non fully-factorized distributions comes from the fact that every mixture implicitly encodes a categorical latent variable (LV) This LV acts as a selector over mixture components and as such it is responsible for introducing correlations among the component distributions. In fact, the weights in a mixture density can be interpreted as the prior probabilities of setting such an LV to a value indicating a component, and the component distributions as conditional distributions when conditioning happens on the selected value.

Example 8 (Latent variable interpretation of mixture) *Consider the mixture model of the two Gaussians in Example 7. Then, it marginalizes out an implicit categorical LV Z having values in $1, 2$ and its mixture density can be re-written as on the right.*

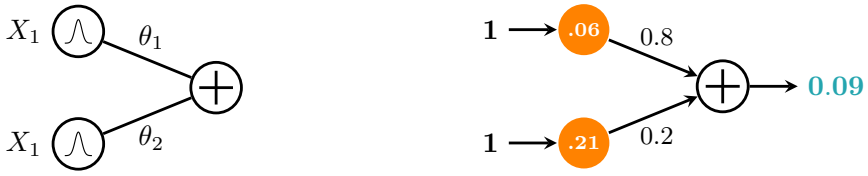


$$p(X_1) = \theta_1 p_{\mathbf{1}}(X_1) + \theta_2 p_{\mathbf{2}}(X_1) = \\ = p(Z = \mathbf{1}) p(X_1 \mid Z = \mathbf{1}) + \\ p(Z = \mathbf{2}) p(X_1 \mid Z = \mathbf{2})$$

Each mixture component as selected by its corresponding LV index is shown above on the left in a different color (purple or grey).

The distribution of a mixture model can be easily represented as computational graphs by introducing a sum unit that computes the weighted average of the inputs it receives. As weights denote mixture components, we graphically represent them as attached to the edges connecting the sum unit to its inputs.

Example 9 (Mixtures as sum units) *Consider the mixture of two Gaussians from Example 7. The computational graph below (left), comprising an input distribution unit for each univariate Gaussian component connected to a sum unit via edges weighted by the mixture weights, represents the mixture density of Example 7.*



To evaluate an EVI query $p(x_1)$ (in blue), the output of the sum unit is obtained by summing the outputs of the input units (in orange) $p_i(X_1 = x_i)$ weighted by w_i for $i = 1, 2$ when evaluated for a certain complete state $\boldsymbol{x} = \{x_1\}$. An example of the computations flowing through the graph is shown above on the right for the input state $x_1 = 1$.

3.2 Probabilistic Circuits: Structure and Parameters

Now that all the building blocks are introduced, we are ready to rigorously define the syntax of PCs as a whole, and introduce the terminology we will use throughout the paper. To begin, it is convenient to distinguish between the *structure* of a PC and its *parameterization*, as for classical PGMs.

Definition 5 (Probabilistic circuits (PCs)) *A probabilistic circuit (PC) $\mathcal{C}$ over RVs $\mathbf{X}$, is a pair $(\mathcal{G}, \boldsymbol{\theta})$, where $\mathcal{G}$ is a computational graph, also called the* **circuit structure** *that is parameterized by $\boldsymbol{\theta}$, also called the* **circuit parameters**, *as defined next. The PC $\mathcal{C}$ computes a function that characterizes a (possibly unnormalized) distrbution $p(\mathbf{X})$.*

Definition 6 (PC structure) *Let $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ be a PC over RVs $\mathbf{X}$. $\mathcal{G}$ is a computational graph in the form of rooted DAG, comprising computational units, also called nodes. The standard evaluation ordering of $\mathcal{G}$, also called feedforward order, is defined as follows.*[7] *If there is an edge $n \to o$ from unit $n \in \mathcal{G}$ to unit $o \in \mathcal{G}$, we say n the input of o and o its output. Let $\mathsf{in}(n)$ denote the set of all input units for unit $n \in \mathcal{G}$ and equivalently, $\mathsf{out}(n)$ denotes the set of its outputs. The input units of $\mathcal{C}$ are all units $n \in \mathcal{G}$ for which $\mathsf{in}(n) = \emptyset$. Analogously, the output unit*[8] *of $\mathcal{C}$, also called its root, is the unit $n \in \mathcal{G}$ for which $\mathsf{out}(n) = \emptyset$. The structure $\mathcal{G}$ comprises three kinds of computational units: input distribution units, product units and sum units, to which a scope is associated as formalized in the following definitions.*

7. The feedforward ordering in the above definition corresponds to the "bottom-up" ordering of several alternative representations of circuits such as arithmetic circuits, sum-product networks and probabilistic sentential decision diagrams (cf. Section 11) There, the natural ordering is assumed to be that of a parent-child relationship, as borrowed from Bayesian networks nomenclature (Koller and Friedman, 2009; Darwiche, 2009). That is, for two units n and c in $\mathcal{G}$, if n is a parent of c and c its child node, then n is the output unit of c and c the input of n. Similarly, their "top-down" ordering corresponds to a backward evaluation in our presentation.

   Note that we adopt the dual-terminology of *units-nodes*, *inputs-leaves* and *output-root* for "backward compatibility" with large portion of the previous literature. Furthermore, when plotting PCs as computational graphs, e.g., in Example 6, we do not graphically show the direction of the arrows connecting inputs to the product to avoid clutter and overcome this ambiguity. Generally, we order inputs before outputs, or equivalently children before parents, from left to right or from bottom to top. This is a graphical convention we will adopt in all graphics in this paper.
8. The structure of a PC $\mathcal{C}$ can be generalized to have multiple output units, in which case $\mathcal{C}$ encodes multiple functions sharing some computations as encoded in the computational graph they have in common (Vergari et al., 2019a; Peharz et al., 2019).

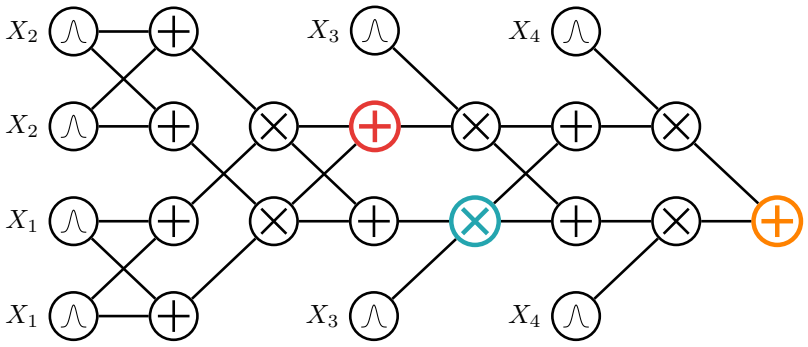
Definition 7 (PC structure: scope) *Let $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ be a PC over RVs $\mathbf{X}$. The computational graph $\mathcal{G}$ is equipped with a scope function ϕ which associates to each unit $n \in \mathcal{G}$ a subset of $\mathbf{X}$, i.e., $\phi(n) \subseteq \mathbf{X}$. For each non-input unit $n \in \mathcal{G}$, $\phi(n) = \cup_{c \in \mathsf{in}(n)} \phi(c)$. The scope of the root of $\mathcal{C}$ is $\mathbf{X}$.*

Definition 8 (PC structure: computational units) *Let $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ be a PC over RVs $\mathbf{X}$. Each unit $n \in \mathcal{G}$ encodes a non-negative function $\mathcal{C}_n$ over its scope: $\mathcal{C}_n : \mathsf{val}(\phi(n)) \to \mathbb{R}_+$. An input unit n in $\mathcal{C}$ encodes a non-negative function that has a support $\mathsf{supp}(\mathcal{C}_n)$ and is parameterized by $\boldsymbol{\theta}_n$.[9] A product unit n defines the product $\mathcal{C}_n(\mathbf{X}) = \prod_{c \in \mathsf{in}(n)} \mathcal{C}_c(\mathbf{X})$. A sum unit n defines the weighted sum $\mathcal{C}_n(\mathbf{X}) = \sum_{c \in \mathsf{in}(n)} \theta_{n,c} \mathcal{C}_c(\mathbf{X})$ parameterized by weights $\theta_{n,c} \geq 0$.[10]*

Definition 9 (PC parameters) *The set of parameters of a PC $\mathcal{C}$ is $\boldsymbol{\theta} = \boldsymbol{\theta}_{\mathsf{S}} \cup \boldsymbol{\theta}_{\mathsf{L}}$ where $\boldsymbol{\theta}_{\mathsf{S}}$ is the set of all sum weights $\theta_{n,c}$ and $\boldsymbol{\theta}_{\mathsf{L}}$ is the set of parameters of all input units in $\mathcal{C}$.*

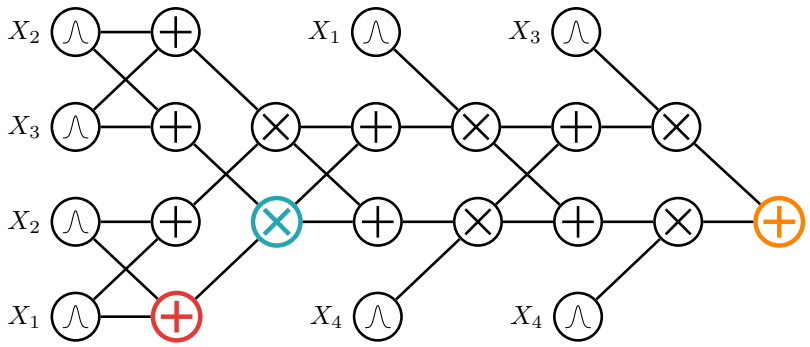
We ground the concepts introduced in the above definitions using the following example.

Example 10 (Probabilistic circuits) *Consider the PC $\mathcal{C}_A$ over continuous RVs $\mathbf{X} = \{X_1, X_2, X_3, X_4\}$ whose structure $\mathcal{G}$ is shown below. Its input distribution units encode univariate Gaussians, two distributions per RV, and are labeled by the RV in their scope (on the left of each unit). This induces a labeling for all inner units. It is easy to verify that the scope of the red sum unit is $\{X_1, X_2\}$ and that the scope of the blue product unit is $\{X_1, X_2, X_3\}$. Sum weights are not shown to avoid clutter. Hence, its parameter set $\boldsymbol{\theta}_A$ comprises the Gaussian unit parameters $\boldsymbol{\theta}_{\mathsf{L}} = \{(\mu_i^j, \sigma_i^j)\}_{i=1,\ldots 4, j=1,2}$ where j denotes one of the two Gaussians, and sum weights $\boldsymbol{\theta}_{\mathsf{S}} = \{\theta_c^s\}_{s=1,\ldots,9, c=1,2}$ where s indicates one of the nine sum units in $\mathcal{C}_A$ and c denotes one of its two inputs. The feedforward ordering is realized by presenting input nodes before outputs, and the circuit's output, the root, is the rightmost sum unit in orange (whose scope is $\mathbf{X}$).*



9. Here we assume that $\mathsf{supp}(\mathcal{C}_n)$ is the *inherent* support for a distribution unit n; i.e., it does not change for arbitrary choices of parameters $\boldsymbol{\theta}_n$. The need for this distinction in structure versus parameterization of a PC will be evident in section 6.
10. The assumption of having normalized weights, i.e., $\sum_c \theta_{n,c} = 1$, delivers the classical intuition on mixture models defining normalized distributions. However, it is not needed because for PCs supporting MAR inference we can always normalize the encoded distribution by locally re-normalizing weights (cf. Section 4).

Consider now another PC $\mathcal{C}_B$ which has the same DAG as computational graph as $\mathcal{C}_A$ and parameters $\boldsymbol{\theta}_B = \boldsymbol{\theta}_A$ but is labeled by a different scope function as shown below.



The different units in $\mathcal{C}_A$ and $\mathcal{C}_B$ that are highlighted by the same color share however the same scope. Even if $\mathcal{C}_A$ and $\mathcal{C}_B$ have the same set of parameters and DAG structure, it is easy to see that they encode two different functions. Section 4 discusses how this difference makes $\mathcal{C}_A$ amenable to tractable inference while $\mathcal{C}_B$ is not.

3.3 Tractable Circuits for Complete Evidence Queries

Before moving to more challenging query classes in the next sections, we consider here the task of evaluate a joint PMF, PDF or mixed mass-density functions as encoded by a PC w.r.t. a complete state. Queries of this kind fall under the class of complete evidence EVI queries, as introduced in Definition 1. Note that in the EVI query class we explicitly refer to *normalized* distributions. We hence assume here that the PCs encode functions that are normalized, postponing to the next section a discussion on how and when we can normalize them if they are not.

Definitions 5-8 yield the semantics of probabilistic circuits as recursive grammars to compose tractable probabilistic models. Indeed, Let $\mathcal{C}_n$ be the sub-circuit rooted at unit n in a PC $\mathcal{C}$, that is the computational graph having n as its output and as units all the units that recursively provide inputs to units in it. $\mathcal{C}_n$ is also a PC, i.e., a computational graph encoding a function over $\phi(n)$, and parameterized by $\boldsymbol{\theta}_n$. This aspect can be captured by the following recursive definition:

Definition 10 (Recursive definition of PCs) *A PC $\mathcal{C}$ over RVs $\mathbf{X}$ is one of the following:*

I) a tractable distribution over $\mathbf{X}$ encoded as a distribution unit,

II) a product of PCs over subsets of $\mathbf{X}$: $\mathcal{C}(\boldsymbol{x}) = \prod_i \mathcal{C}_i(\boldsymbol{x})$, or

III) a positive weighted sum of PCs over subsets of $\mathbf{X}$: $\mathcal{C}(\boldsymbol{x}) = \sum_i w_i \mathcal{C}_i(\boldsymbol{x})$, with $w_i > 0$.

Hence, Definition 10 offers a natural way to answer an EVI query, by following the recursive evaluation of the function encoded in the PC. This also guarantees a polynomial number of computations if intermediate computations are cached at each recursive call.

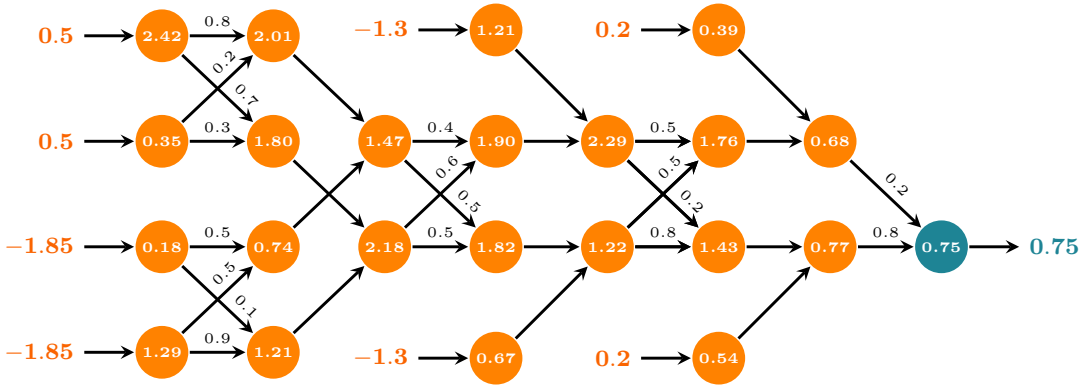
Algorithm 1 EVIQuery$(\mathcal{C}, \boldsymbol{x})$

Input: a PC $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ over RVs $\mathbf{X}$ and a complete state $\boldsymbol{x} \in \mathsf{val}(\mathbf{X})$
Output: $\mathcal{C}(\boldsymbol{x}) := p(\mathbf{X} = \boldsymbol{x})$

```
N ← FeedforwardOrder(G)                          ▷ Order units, inputs before outputs
for each n ∈ N do
    if n is a sum unit then r_n ← Σ_{c∈in(n)} θ_{n,c} r_c
    else if n is a product unit then r_n ← Π_{c∈in(n)} r_c
    else if n is an input distribution unit then r_n ← C_n(x_{φ(n)})
return r_n                                       ▷ the value of the output of C
```

The iterative version to answer an EVI query $p(\boldsymbol{x})$ for a PC $\mathcal{C}$ over RVs $\mathbf{X}$ is summarized in Algorithm 1. After topologically ordering the DAG of $\mathcal{C}$ in a feedforward way, i.e., input before output units, each unit n stores its computation in a local register r_n. The output $p(\boldsymbol{x})$ can be read from the register of the circuit output, the last unit in the ordering, i.e., its root. Examples 4, 6 and 9 in Section 3.1 provided a flavor of such a feedforward evaluation for each computational unit. The following example glues them in a larger circuit.

Example 11 (EVI query computations) *Consider the PC $\mathcal{C}_A$ as shown in Example 10 when the univariate Gaussian units encode the following distributions, ordered from top to bottom, $\mathcal{N}(\mu_1^1 = -1.0, \sigma_1^1 = 2.0)$ and $\mathcal{N}(\mu_1^2 = -2.0, \sigma_1^2 = 0.1)$ for X_1, $\mathcal{N}(\mu_2^1 = 0.6, \sigma_2^1 = 0.1)$ and $\mathcal{N}(\mu_2^2 = 0.0, \sigma_2^2 = 1.0)$ for X_2, $\mathcal{N}(\mu_3^1 = -1.5, \sigma_3^1 = 0.2)$ and $\mathcal{N}(\mu_3^2 = -1.0, \sigma_3^2 = 0.5)$ for X_3 and $\mathcal{N}(\mu_4^1 = 0.0, \sigma_4^1 = 1.0)$ and $\mathcal{N}(\mu_4^2 = 0.0, \sigma_4^2 = 0.1)$ for X_4. The sum weights are reported on the corresponding edges in the picture below.*



Then, how to evaluate the EVI *query $p(\boldsymbol{x})$ for $\boldsymbol{x} = (-1.85, 0.5, -1.3, 0.2)$, according to Algorithm 1, is shown above where intermediate computations as saved in each unit's register, are shown in the orange circles.*

3.4 Beyond Simple PCs

The basic definition for PCs we provided in this section are enough to build a framework that can unify several tractable probabilistic representations, as we will show in the following

sections. In the circuit literature, this basic framework has been extended in a number of interesting ways. We now provide a collection of pointers to these extensions for those readers who are interested in going in depth with, and beyond, PCs. [11]

Non-probabilistic circuits, still represented as computational graphs involving sums and products go under the name of *arithmetic circuits* in computational complexity theory and provide one of simplest and most elegant formalism to reason about the expressive efficiency of model classes and algorithmic complexity (Shpilka and Yehudayoff, 2010).

In the following sections, many of the theoretical results we provide have their roots in analogous results for arithmetic circuits or simpler Boolean circuits, encoding logical formulas. We discuss in depth the strong (and causal!) connection between PCs and probabilistic reasoning over logical formulas encoded as circuits, via weighted model counting (WMC) (Darwiche and Marquis, 2002) in Section 12.2. Inspired by this link, extensions to first-order logical representations have been investigated both for WMC-circuits (cf. Van den Broeck (2013) for a survey) and for PCs (Webb and Domingos, 2013) for which learning routines have also been developed (Nath and Domingos, 2014; Niepert and Domingos, 2015). Further generalizations and connections to tractable circuits to semirings not involving sum and product operations are discussed in Section 12.1.

Inspired by results from complexity circuits, alternative computational graph structures for PCs have been explored in order to increase their expressive efficiency. These include PCs with computational units performing quotients (Sharir and Shashua, 2018) and with sum units with possibly negative weights (Dennis, 2016) in order to realize non-monotonic, but still positive, mixtures (Valiant, 1979b).

Our definitions for the structure and parameters of PCs can be generalized in a number of ways. First, the scope function as introduced in Definition 7 is decoupled from the computational graph of a PC. As such, it can be treated as an additional parameter that can be learned from data independently Trapp et al. (2019), while the computational graph becomes a template for a set of PCs

Second, parameters in a PC could be treated as first-class RVs, where providing a prior distribution over them (e.g., a Dirichlet over the sum weights or a NIG over the parameters of a Gaussian input unit) would yield a Bayesian interpretation of circuits. While appealing from the perspective of robustly modeling uncertainty, inference in Bayesian PCs is generally intractable. Nevertheless, efficient approximations can be carried out by exploiting the tractable inference of PCs as sub-routines for sampling (Vergari et al., 2019b; Trapp et al., 2019) or approximations via variational (Zhao et al., 2016a) or moment-matching (Rashwan et al., 2016; Jaini et al., 2016) optimization. This Bayesian treatment allows to generalize sum units to mixtures with an infinite but countable number of components (Trapp et al., 2019). An analogous take on modeling higher-order uncertainty in PCs comes from the perspective of imprecise probabilities (Walley, 1991): scalar sum weights in PCs can be generalized into interval representations yielding a circuit that encodes not a single distribution but a credal set of distributions (Mauá et al., 2017; Antonucci et al., 2019).

Lastly, input units can be extended beyond simple tractable probability distributions. While several kinds of different parametric (Jaini et al., 2016; Molina et al., 2017; Den-

11. Navigating this body of literature might be easier and more fruitful for readers after reading section 11, as each of the works we point to refers to a disparate TPM representation and might adopt a very different formalism and vocabulary for denoting the concepts we have introduced so far.

nis, 2016; Vergari et al., 2019b) and non-parametric (Molina et al., 2018; Morettin et al., 2020) distributions have been adopted as input units, more recent works focused on intractable models like variational autoencoders (Tan and Peharz, 2019), or classifiers and regressors (Trapp et al., 2020; Khosravi et al., 2020).

4. Tractable Circuits for Marginal Queries

In this section, we extend the PC framework to tractably answer an important class of queries, *marginals*. We first formally define the MAR query class in Section 4.1, and later in Section 4.2 we provide a precise characterization of the model class of PCs that are tractable representations for the class via some structural properties over their computational graphs: *smoothness and decomposability*. Furthermore, in Section 4.3 we show how these properties enable tractable computations of different query classes that share the same computational challenges of the MAR class, such as conditional queries and computing the moments of a distribution. Finally, section 4.4 collects pointers to further readings about representing and learning smooth and decomposable PCs and related tractable formalisms.

4.1 The MAR and CON Query Classes

Marginal queries are of paramount importance when we want to reason about states of the world where not all RVs are fully observed. This might happen because we do not have access to their values, as in the case of *missing values* in a patient record, or because we do not care about specific values for them, as in our traffic jam scenario (cf. Section 2.2).

Example 12 (Marginal query) *Consider the probability distribution p_{m} defined over the RVs $\mathbf{X}$ as in the traffic jam scenario in Example 1. Then question q_1 can be answered by the* MAR *query as defined in Equation 1 by computing*

$$p_{\mathsf{m}}(\mathsf{W} = \mathsf{Rain}, \mathsf{J}_{\mathsf{Westwood}} = 1) = \int_{t=0}^{24} \sum_{\mathbf{j}} p_{\mathsf{m}}(\mathsf{W} = \mathsf{Rain}, \mathsf{T} = t, \mathsf{J}_{\mathsf{Westwood}} = 1, \mathbf{J} = \mathbf{j})\, dT.$$

where $\mathbf{J}$ indicates all the jam binary RVs with the exception of $\mathsf{J}_{\mathsf{Westwood}}$ and $\mathbf{j}$ is a state for them.

As this example suggests, the key difference of marginal queries from complete evidence ones is that they admit *partial states* as evidence. As this effectively amounts to computing integrals and summations over complete evidence probabilities, we can define a more general class of marginal queries where these operations[12] are taken over subsets of the RV domains.

Definition 11 (MAR query class) *Let $p(\mathbf{X})$ be a joint distribution over RVs $\mathbf{X}$. The class* MAR *of marginal queries over p is the set of functions that compute:*

$$p(\mathbf{E} = \boldsymbol{e}, \mathbf{Z} \in \boldsymbol{\mathcal{I}}) = \int_{\boldsymbol{\mathcal{I}}} p(\boldsymbol{z}, \boldsymbol{e})\ d\mathbf{Z} \tag{2}$$

12. For the sake of simplicity, from here on, we will adopt the more general integral symbol to subsume both multi-dimensional finite integrals for continuous RVs and nested summations for discrete ones.

where $\boldsymbol{e} \in \mathsf{val}(\mathbf{E})$ *is a partial state for any subset of RVs* $\mathbf{E} \subseteq \mathbf{X}$*, and* $\mathbf{Z} = \mathbf{X} \setminus \mathbf{E}$ *is the set of* k *RVs to be integrated over intervals* $\boldsymbol{\mathcal{I}} = \mathcal{I}_1 \times \cdots \times \mathcal{I}_k$ *each of which is defined over the domain of its corresponding RV in* $\mathbf{Z}$*:* $\mathcal{I}_i \subseteq \mathsf{val}(Z_i)$ *for* $i = 1, \ldots, k$*.*

Note that the integration is over a Cartesian product of intervals (i.e., a hypercube). Therefore, computing a marginal query involves integrating out k variables from the complete evidence computation, i.e., solving a k-dimensional integral of the form:

$$\int_{\mathcal{I}_1} \int_{\mathcal{I}_2} \cdots \int_{\mathcal{I}_k} p(z_1, z_2, \ldots, z_k, \boldsymbol{e})\, dZ_k \cdots dZ_2\, dZ_1.$$

When RVs $\mathbf{Z}$ are marginalized over their whole domains, i.e., $\mathcal{I}_i = \mathsf{val}(Z_i)$, we retrieve the classical definition of marginal queries (Darwiche, 2009; Koller and Friedman, 2009), denoted by the shorthand $p(\mathbf{E} = \boldsymbol{e})$. Furthermore, the general query class MAR also includes queries on the joint *cumulative distribution function* (CDF) of a distribution p when integration is performed over open intervals for all RVs. Lastly, it naturally follows that $\mathsf{EVI} \subset \mathsf{MAR}$.

A query class that shares the same computational challenges of MAR is that of *conditional queries* (CON), i.e., queries that compute the probability of a partial state *conditioned* on another event also given as partial state.

Example 13 (Conditional query) *Consider the probability distribution* p_{m} *defined over the RVs* $\mathbf{X}$ *as in the traffic jam scenario in Example 1. Then the question "What is the probability that there will be a traffic jam only on Westwood Blvd. at 12 o'clock?" can be answered by the following* CON *query:*

$$p_{\mathsf{m}}(\mathsf{J}_{\mathsf{Westwood}} = 1, \mathsf{J}_1 = 0, \ldots, \mathsf{J}_{k-1} = 0 \mid \mathsf{T} = 12.0)$$

where $\mathsf{J}_1, \ldots, \mathsf{J}_{k-1}$ *are the traffic indicator for all streets with the exception of Westwood Blvd.*

One can easily define the class of conditional queries in terms of the marginal query class by noting that any conditional query can be rewritten as a ratio of marginal queries.

Definition 12 (CON query class) *Let* $p(\mathbf{X})$ *be a joint distribution over RVs* $\mathbf{X}$*. The class of conditional queries* CON *is the set of queries that compute functions of the form*

$$p(\mathbf{Q} = \boldsymbol{q} \mid \mathbf{E} = \boldsymbol{e}, \mathbf{Z} \in \boldsymbol{\mathcal{I}}) = \frac{p(\mathbf{Q} = \boldsymbol{q}, \mathbf{E} = \boldsymbol{e}, \mathbf{Z} \in \boldsymbol{\mathcal{I}})}{p(\mathbf{Q} \in \mathsf{val}(\mathbf{Q}), \mathbf{E} = \boldsymbol{e}, \mathbf{Z} \in \boldsymbol{\mathcal{I}})} = \frac{\int_{\boldsymbol{\mathcal{I}}} p(\boldsymbol{q}, \boldsymbol{e}, \boldsymbol{z})\ d\mathbf{Z}}{\int_{\mathsf{val}(\mathbf{Q})} \int_{\boldsymbol{\mathcal{I}}} p(\boldsymbol{q}, \boldsymbol{e}, \boldsymbol{z})\ d\mathbf{Z}\ d\mathbf{Q}} \tag{3}$$

where $\boldsymbol{e} \in \mathsf{val}(\mathbf{E})$ *and* $\boldsymbol{q} \in \mathsf{val}(\mathbf{Q})$ *are partial states any subsets of RVs* $\mathbf{Q}, \mathbf{E} \subset \mathbf{X}$*, and* $\mathbf{Z} = \mathbf{X} \setminus (\mathbf{E} \cup \mathbf{Q})$ *is the set of* k *RVs to be integrated over intervals* $\boldsymbol{\mathcal{I}} = \mathcal{I}_1 \times \cdots \times \mathcal{I}_k$ *each of which is defined over the domain of its corresponding RV in* $\mathbf{Z}$*:* $\mathcal{I}_i \subseteq \mathsf{val}(Z_i)$ *for* $i = 1, \ldots, k$*.*

Next, we define a class of PCs that deliver tractable inference for both MAR and CON.

4.2 Smooth and Decomposable PCs

Solving one of the definite multivariate integrals as those needed to answer queries from MAR or CON is in general a #P-Hard problem (Baldoni et al., 2011) and it is no wonder that this task is hard for many common probabilistic models such as Bayesian networks (Darwiche, 2009), Markov random fields (Koller and Friedman, 2009), variational autoencoders (Rezende et al., 2014; Kingma and Welling, 2013) and normalizing flows (Papamakarios et al., 2019). However, restricting the computational graphs of PCs to have certain structural properties can guarantee a linear time computation for all possible queries in MAR and CON. By looking at the simplest probabilistic models that can be turned into PCs, we can understand what these properties are and how the computations can be generalized to a general algorithmic scheme in PCs.

In the simplest case, answering MAR queries equates to collecting the output of the single distribution unit that encodes the distribution.

Example 14 (Tractable densities for MAR) *Consider an input distribution unit encoding a Gaussian density* $p(X) = \mathcal{N}(X; \mu = 1, \sigma = 0.1)$ *as defined in Example 4. Consider asking it to compute the* MAR *query* $p(X < 1.1)$*, the input distribution unit will output* ≈ 0.84*:*

or simply 1.0 *when queried for its partition function* Z*; that is, integrating over all* $\mathbb{R}$*:*

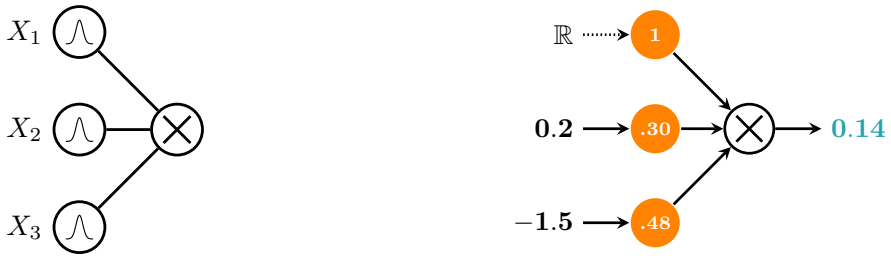
where the interval for integration is shown as an input connected with a dotted line.

Consider a factorized probabilistic model over the partitioning $\mathbf{X} = \mathbf{X}_1 \cup \ldots \cup \mathbf{X}_D$ of the form $p_{\mathsf{m}}(\mathbf{X}) = \prod_{i=1}^{D} p_{\mathsf{m}_i}(\mathbf{X}_i)$ as introduced in Definition 3. Then, the marginalization integral of Equation 2 can be "broken down" as a product of simpler integrals:

$$\int_{\boldsymbol{\mathcal{I}}_1} p_{\mathsf{m}_1}(\boldsymbol{z}_1, \boldsymbol{e}_1)\, d\mathbf{Z}_1 \int_{\boldsymbol{\mathcal{I}}_2} p_{\mathsf{m}_2}(\boldsymbol{z}_2, \boldsymbol{e}_2) d\mathbf{Z}_2 \cdots \int_{\boldsymbol{\mathcal{I}}_D} p_{\mathsf{m}_D}(\boldsymbol{z}_D, \boldsymbol{e}_D)\, d\mathbf{Z}_D. \tag{4}$$

where the evidence RVs and the marginals are partitioned into the sets $\mathbf{E} = \mathbf{E}_1 \cup \ldots \cup \mathbf{E}_D$ and $\mathbf{Z} = \mathbf{Z}_1 \cup \ldots \cup \mathbf{Z}_D$ according to the partitioning of $\mathbf{X}$ and which also induces the partitioning over the multivariate interval $\boldsymbol{\mathcal{I}}_1 \times \ldots \times \boldsymbol{\mathcal{I}}_D$. That is, independence among the factors enables the independent computation of the smaller integrals. Operationally, we can then solve the sub-integrals of the inputs of a product unit and compose them into a product in a *divide-et-impera* fashion.

Example 15 (Marginal queries for factorized models) *Consider the factorized multivariate Gaussian introduced in Example 5 and whose PC representation from Example 6 is shown below on the left. Then the computational graph below on the right illustrates how to compute the* MAR *query* $p(X_2 = 0.2, X_3 = -1.5) \approx 0.14$ *(in blue).*

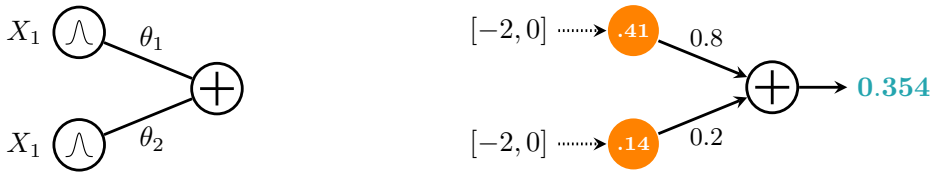


Lastly, consider a sum unit encoding a mixture model of the form $p_{\mathsf{m}}(\mathbf{X}) = \sum_{i=1}^{k} \theta_i p_i(\boldsymbol{x})$ as introduced in Definition 4. For such model, the integral of Equation 2 simplifies as the weighted sum of integrals evaluated w.r.t. each mixture component:

$$\sum_{i=1}^{k} \theta_i \int_{\boldsymbol{\mathcal{I}}} p_i(\boldsymbol{z}, \boldsymbol{e}) \; d\mathbf{Z}. \tag{5}$$

Again, this suggest that the integration of a PC encoding a mixture is deferred to computing integration for the circuits that are inputs to the sum unit, weighting their results in the sum output.

Example 16 (Marginal queries for mixture models) *Consider the mixture of Gaussians introduced in Example 7, whose PC representation has been introduced in Example 9 and is shown below on the left. Then the computational graph on the right illustrates how to compute the* MAR *query* $p(-2 \leq X_1 \leq 0) \approx 0.354$ *(in blue).*



A general scheme to compute MAR queries with PCs can therefore be found by recursively repeating all these steps while evaluating the PC in the usual feedforward way, as the next Definition specifies.

Definition 13 (MAR query computations) *Let $\mathcal{C}$ be a PC over RVs $\mathbf{X}$ and $\boldsymbol{e} \in \mathsf{val}(\mathbf{E})$ a partial state for RVs $\mathbf{E} \subseteq \mathbf{X}$ and let $\boldsymbol{\mathcal{I}}_{\mathbf{Z}} = \mathcal{I}_{Z_1} \times \cdots \times \mathcal{I}_{Z_k}$ be a multidimensional and possibly open interval for RVs $\mathbf{Z} = \mathbf{X} \setminus \mathbf{E}$. Then, we say that $\mathcal{C}$ computes the* MAR *query $p(\mathbf{E} = \boldsymbol{e}, \mathbf{Z} \in \boldsymbol{\mathcal{I}}_{\mathbf{Z}})$ (cf. Definition 11) if the output of $\mathcal{C}$ given $\boldsymbol{e}$ and $\boldsymbol{\mathcal{I}}_{\mathbf{Z}}$, denoted $\mathcal{C}(\boldsymbol{e}; \boldsymbol{\mathcal{I}}_{\mathbf{Z}})$ is given by evaluating $\mathcal{C}$ according to Algorithm 2.*

The general evaluation scheme of a PC for MAR queries, therefore differs from the computation of complete evidence (cf. Algorithm 1) only in the evaluation of the input distribution units (line 5 of Algorithm 2), where computations of a partial state is restricted

Algorithm 2 MARQUERY($\mathcal{C}, \boldsymbol{e}, \boldsymbol{\mathcal{I}}_{\mathbf{Z}}$)

Input: a PC $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ over RVs $\mathbf{X}$, a partial state $\boldsymbol{e} \in \mathsf{val}(\mathbf{E})$ with $\mathbf{E} \subset \mathbf{X}$ and a set of integration domains $\boldsymbol{\mathcal{I}}_{\mathbf{Z}}$ for RVs $\mathbf{Z} = \mathbf{X} \setminus \mathbf{E}$

Output: $\mathcal{C}(\boldsymbol{x}, \boldsymbol{\mathcal{I}}_{\mathbf{Z}}) := p(\mathbf{E} = \boldsymbol{e}, \mathbf{Z} \in \boldsymbol{\mathcal{I}}_{\mathbf{Z}})$

1: $\mathsf{N} \leftarrow$ FEEDFORWARDORDER($\mathcal{G}$) ▷ Order units, inputs before outputs
2: **for each** $n \in \mathsf{N}$ **do**
3: **if** n is a sum unit **then** $\mathsf{r}_n \leftarrow \sum_{c \in \mathsf{in}(n)} \theta_{n,c} \mathsf{r}_c$
4: **else if** n is a product unit **then** $\mathsf{r}_n \leftarrow \prod_{c \in \mathsf{in}(n)} \mathsf{r}_c$
5: **else if** n is an input distribution unit **then** $\mathsf{r}_n \leftarrow \mathcal{C}_n(\boldsymbol{e}_{\phi(n)}; \boldsymbol{\mathcal{I}}_{\mathbf{Z}_{\phi(n)}})$
6: **return** r_n ▷ the value of the output of $\mathcal{C}$

to the scope of the input unit, denoted as $\mathcal{C}_n(\boldsymbol{e}_{\phi(n)}; \boldsymbol{\mathcal{I}}_{\mathbf{Z}_{\phi(n)}})$. In essence, it reverts to complete evidence computations when $\mathbf{Z}_{\phi(n)} = \emptyset$, i.e., the RVs to integrate are outside the scope of input distribution unit n, or otherwise follows the base case as illustrated in Example 14. Quite remarkably, certain PCs have the ability of delegating the computation of integrals to input distributions as just illustrated for *any possible* MAR query.

Definition 14 *A PC $\mathcal{C}$ encoding $p(\mathbf{X})$* **computes marginals** *if the partial state computation $\mathcal{C}(\boldsymbol{e}; \boldsymbol{\mathcal{I}}_{\mathbf{Z}})$ is equal to the distribution marginal $\int_{\boldsymbol{\mathcal{I}}_{\mathbf{Z}}} p(\boldsymbol{e}, \boldsymbol{z})\, d\mathbf{Z}$ for all possible subsets $\mathbf{E} \subseteq \mathbf{X}$ and $\mathbf{Z} = \mathbf{X} \setminus \mathbf{E}$, partial states $\boldsymbol{e} \in \mathsf{val}(E)$, and intervals $\mathcal{I}$.*

Note that a PC that does not compute marginals for a certain distribution p is not necessarily intractable for certain marginal queries—there could be other polytime routines than Algorithm 2 for those partial state computation. Instead, PCs that do compute marginals allow every marginal integration to exactly decompose according the PC structure, in a top-down fashion described in Examples 14–16 such that its feedforward evaluation retrieves the exact query value. Intuitively, one can think of these PCs as compactly storing in their computational graphs the set of all computational graphs computing the marginals for every subset of RVs, partial states and intervals. A question still remains: when is a PC guaranteed to compute marginals? The answer lies in restricting its computational graph, enforcing two structural properties: *decomposability and smoothness*. The next definitions and examples formally introduce and illustrate the class of decomposable and smooth PCs.

Definition 15 (Decomposability) *A product node n is* **decomposable** *if the scopes of its input units do not share variables: $\phi(c_i) \cap \phi(c_j) = \emptyset$, $\forall\, c_i, c_j \in \mathsf{in}(n), i \neq j$. A PC is decomposable if all of its product units are decomposable.*

Example 17 (Decomposable PCs) *Consider the PC $\mathcal{C}_A$ as defined in Example 10. The product unit highlighted in blue with scope $\{X_1, X_2, X_3\}$ is decomposable, as its two inputs have scopes $\{X_1, X_2\}$ and $\{X_3\}$, respectively. It is easy to verify that all other products in $\mathcal{C}_A$ are decomposable as well, and therefore $\mathcal{C}_A$ classifies as a decomposable PC. Consider instead the blue product of the circuit $\mathcal{C}_B$: it is not decomposable as X_2 appears in the scope of both its inputs. Therefore $\mathcal{C}_B$ is not a decomposable PC.*

All factorized models employed as simple PC examples are decomposable by design, as to guarantee probabilistic independence each factor does not share RVs with the others. However, note that a decomposable PC does not necessarily encode a factorized distribution over its scope, even if all of its product units are decomposable. This is due to the presence of sum units which introduce correlations among the distributions encoded by its input sub-circuits.

Definition 16 (Smoothness) *A sum node n is* **smooth** *if its inputs all have identical scopes: $\phi(c) = \phi(n)$, $\forall\, c \in \mathsf{in}(n)$. A circuit is smooth if all of its sum units are smooth.*

Example 18 (Smooth PCs) *Consider the PC $\mathcal{C}_A$ as defined in Example 10. The sum unit highlighted in red with scope $\{X_1, X_2\}$ is smooth, as its two inputs have the same scope. As all other sum units in $\mathcal{C}_A$ are smooth, $\mathcal{C}_A$ is a smooth circuit. On the other hand, consider instead the red sum in the circuit $\mathcal{C}_B$ again with scope $\{X_1, X_2\}$; it is not smooth as its inputs have scopes $\{X_1\}$ and $\{X_2\}$ respectively. Therefore $\mathcal{C}_B$ is not a smooth circuit.*

Commonly, smoothness is implicitly assumed when dealing with valid mixture models, as was the case for all the GMM examples we used up to now. As a matter of fact, PCs can be seen as a generalization of classical mixture models, more precisely as hierarchical mixture models. As we will see later in Section 7, if a PC is decomposable but not smooth we can *smooth* it in polytime, i.e., apply a transformation that outputs a smooth PC while not altering its encoded distribution.

It is evident how decomposability is a sufficient property for tractable computation of marginals in factorized models: when present, the decomposition of larger integrals is always allowed over smaller and disjoint scopes (cf. Eq. 4). Equivalently, the safe exchange of summation and integration over mixture components (cf. Eq. 5) comes from the applicability of the Fubini Theorem. These properties in a PC, in addition to the assumption that input distributions allow tractable marginals, are sufficient to guarantee the tractable computation of any marginal query (Darwiche, 2003; Peharz et al., 2015).

Proposition 17 *Let $\mathcal{G}$ be a circuit structure that is smooth and decomposable. Then for any parameterization $\boldsymbol{\theta}$, the probabilistic circuit $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ computes marginals.*

Moreover, a CON query can be evaluated as a ratio of two MAR queries, as shown in Definition 12. Thus, a circuit that allows tractable marginals also allows tractable conditional inference by extension.

Corollary 18 *Let $\mathcal{C}$ be a smooth and decomposable PC over RVs $\mathbf{X}$ encoding $p(\mathbf{X})$. Suppose the input distribution units of $\mathcal{C}$ allow tractable marginal inference. Then the CON query $p(\mathbf{Q} = \boldsymbol{q} \mid \mathbf{E} = \boldsymbol{e}, \mathbf{Z} \in \boldsymbol{\mathcal{I}})$ can be computed in time linear in the size of $\mathcal{C}$ for any subsets $\mathbf{Q}, \mathbf{E} \subset \mathbf{X}$ and $\mathbf{Z} = \mathbf{X} \setminus (\mathbf{E} \cup \mathbf{Q})$, partial states $\boldsymbol{q} \in \mathsf{val}(\mathbf{Q})$ and $\boldsymbol{e} \in \mathsf{val}(\mathbf{E})$, and intervals $\boldsymbol{\mathcal{I}}$.*

It is less apparent, however, if these properties are also *necessary* for such computations to be efficient. As the next theorem shows, a PC *needs* to be smooth and decomposable to tractably compute marginals.

Theorem 19 *Let $\mathcal{G}$ be a circuit structure such that for any parameterization $\boldsymbol{\theta}$, the probabilistic circuit $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ encodes a distribution over RVs $\mathbf{X}$ and computes marginals. Then, $\mathcal{G}$ must be decomposable and smooth.*

To prove Theorem 19, we first introduce the *shallow representation* of a probabilistic circuit, which will prove to be handier to operate on. Any PC $\mathcal{C}$ over RVs $\mathbf{X}$ can be "unrolled" into an equivalent PC $\mathcal{S}$ of the form

$$\mathcal{S}(\mathbf{x}) = \sum_{i=1}^{K} \theta_i \prod_{j=1}^{M_i} L_{ij}(\mathbf{x}).$$

that is comprising a single sum unit over a (potentially exponential) number of product units, K, each multiplying M_i input distribution units of $\mathcal{C}$, denoted here as L_{ij}. This shallow representation is insightful for understanding the relationships between PCs, mixture models and polynomial representations, as will be discussed in depth in Section 5.

To turn a deep PC $\mathcal{C}$ into its shallow representation $\mathcal{S}$, one can apply the distributive law of multiplication over addition in the classical algebraic semiring recursively, inputs before outputs. The construction follows the recursive definition of PCs (cf. Definition 10) and proceeds as follows. If $\mathcal{C}$ is a single input distribution unit, its corresponding $\mathcal{S}$ will comprise a sum unit over a single product unit which is weighted by $\theta = 1$ and fed by input distribution $L := \mathcal{C}$. Alternatively, if $\mathcal{C}$ consists of a sum unit n over R sub-circuits with weights $\theta_1, \ldots, \theta_R$, then its shallow representation $\mathcal{S}$ will comprise a single sum unit n' whose inputs are all the product units appearing in the shallow representations $\mathcal{S}_1, \ldots, \mathcal{S}_R$ of its sub-circuits, whose weights are multiplied by $\theta_1, \ldots, \theta_R$. Instead, if $\mathcal{C}$ consists of a product unit n over R sub-circuits, then its shallow representation is obtained by performing the *cross-product* of the shallow representations $\mathcal{S}_1, \ldots, \mathcal{S}_R$ of its sub-circuits. That is, each product unit in $\mathcal{S}$ is obtained by multiplying R product units, one from each $\mathcal{S}_i$, and is weighted by the product of their weights.

This above construction highlights that every shallow representation $\mathcal{S}_n$ for a sub-circuit $\mathcal{C}_n$ shares its scope and that for every product unit n in $\mathcal{C}$ there are several product units in $\mathcal{S}$ built by multiplying n with other units in $\mathcal{C}$. We will call the units in $\mathcal{S}$ participating in this 1-to-many mapping, *corresponding* products. Moreover, note that the shallow representation $\mathcal{S}$ is obtained by manipulating the DAG structure of $\mathcal{C}$, and thus the construction is invariant to how the input distribution unit is evaluated. In particular, for any partial state and interval, evaluating $\mathcal{S}$ and $\mathcal{C}$ according to Algorithm 2 return the same values.

We will first prove that a circuit that tractably computes marginals for any parameterization must be decomposable. The proof utilizes the following relationship between decomposability of a PC and that of its shallow representation.

Proposition 20 *Let $\mathcal{C}$ be a PC over RVs $\mathbf{X}$ and $\mathcal{S}$ its shallow representation. If $\mathcal{S}$ is decomposable, then $\mathcal{C}$ is decomposable.*

Proof Suppose $\mathcal{C}$ is not decomposable, then there must be at least one product unit n in $\mathcal{C}$ that is not decomposable, i.e., at least two inputs units of n, say c_1 and c_2, have overlapping scopes. This can happen for two reasons. First, the sub-circuits $\mathcal{C}_{c_1}$ and $\mathcal{C}_{c_2}$ could share at least one input distribution unit L. In this case, all product units in $\mathcal{S}$ corresponding to n

must multiply (at least) two copies of L, and thus they cannot be decomposable. Second, the sub-circuit $\mathcal{C}_{c_1}$ could contain an input distribution unit L_1 and $\mathcal{C}_{c_2}$ another unit L_2 that has an overlapping scope with L_1. In this case, all product units in $\mathcal{S}$ corresponding to n must multiply (at least once) L_1 and L_2, hence are not decomposable. In both cases, $\mathcal{S}$ will not be decomposable. ■

Suppose that $\mathcal{C}$ is not decomposable. From Proposition 20, we know that its shallow representation $\mathcal{S}$ is not decomposable either. Moreover, we know that there must exist a product unit in $\mathcal{S}$ that either contains two copies of the same input distribution unit, or two input units having overlapping scopes. Let Z be one of the shared variables in such non-decomposable product unit. Consider now an arbitrary MAR query of the form $p(\mathbf{E} = \boldsymbol{e}, Z \in \mathcal{I})$ where $\boldsymbol{e}$ is a partial state on $\mathbf{E} = \mathbf{X} \setminus \{Z\}$ and $\mathcal{I}$ an integration interval as defined in Definition 11. Computing such a query by evaluating Algorithm 2 would yield the following computation:

$$\begin{aligned}\mathcal{C}(\boldsymbol{e}; \mathcal{I}) &= \sum_{i=1}^{K} \theta_i \prod_{j=1}^{M_i} L_{ij}(\boldsymbol{e}_{\phi(L_{ij})}; \mathcal{I}_{\phi(L_{ij})}) \\ &= \sum_{i=1}^{K} \theta_i \left(\prod_{j: Z \notin \phi(L_{ij})} L_{ij}(\boldsymbol{e}_{\phi(L_{ij})}) \right) \left(\prod_{j: Z \in \phi(L_{ij})} \int_{\mathcal{I}} L_{ij}(\boldsymbol{e}_{\phi(L_{ij})}, z) dZ \right). \end{aligned} \tag{6}$$

Note that because $\mathcal{C}$ is not decomposable, there must exist an i such that multiple input units include Z in their scope (i.e., $Z \in \phi(L_{ij})$). On the other hand, the MAR query evaluates to

$$\begin{aligned} p(\mathbf{E} = \boldsymbol{e}, Z \in \mathcal{I}) &= \int_{\mathcal{I}} \sum_{i=1}^{K} \theta_i \prod_{j=1}^{M_i} L_{ij}(\boldsymbol{e}, z) dZ \\ &= \sum_{i=1}^{K} \theta_i \left(\prod_{j: Z \notin \phi(L_{ij})} L_{ij}(\boldsymbol{e}_{\phi(L_{ij})}) \right) \left(\int_{\mathcal{I}} \prod_{j: Z \in \phi(L_{ij})} L_{ij}(\boldsymbol{e}_{\phi(L_{ij})}, z) dZ \right). \end{aligned} \tag{7}$$

Equations 6 and 7 are not equal in general for all choice of parameters, and thus Algorithm 2 does not necessarily return the marginal given a non-decomposable PC.

We will now show that a PC that supports tractable marginal computations is not only decomposable but also smooth. Again, we leverage the following result about shallow representations.

Proposition 21 *Let $\mathcal{C}$ be a decomposable PC over RVs $\mathbf{X}$ and $\mathcal{S}$ its shallow representation. If $\mathcal{S}$ is smooth, then $\mathcal{C}$ is smooth.*

Proof Suppose $\mathcal{C}$ is not smooth. Then there must be at least one sum unit n in $\mathcal{C}$ that is not smooth; i.e., there exists an input c of n such that an RV $X \in \phi(n)$ does not appear in $\phi(c)$. By the recursive construction of shallow representations, $\mathcal{S}_c$ also does not include X in its scope, and thus such is the case for at least one product unit of $\mathcal{S}_n$. Moreover, the product units of $\mathcal{S}_n$ either appear as is in $\mathcal{S}$ or are multiplied with other product units. However,

because $\mathcal{C}$ is decomposable, multiplication with shared scopes never occurs; hence, no other copy of RV X will be introduced in the scope of product units of $\mathcal{S}_n$ while completing the construction of the shallow representation $\mathcal{S}$. Therefore, there must exist a product unit of $\mathcal{S}$ whose scope does not include X, and $\mathcal{S}$ cannot be smooth. ■

Next, suppose $\mathcal{C}$ is decomposable but not smooth. Then from Proposition 21, we know that its shallow representation $\mathcal{S}$ is also not smooth and that there exists a product unit of $\mathcal{S}$ whose scope does not include a variable in $\mathbf{X}$, say Z. Let us again consider a MAR query $p(\mathbf{E} = \boldsymbol{e}, Z \in \mathcal{I})$ where $\boldsymbol{e}$ is a partial state on $\mathbf{E} = \mathbf{X} \setminus \{Z\}$ and $\mathcal{I}$ is an interval on Z. Suppose I refers the set of product units of $\mathcal{S}$ without Z in their scopes. Moreover, because $\mathcal{C}$—and by extension $\mathcal{S}$—is decomposable, every product unit of $\mathcal{S}$ will have at most one input distribution unit that depends on Z; w.l.o.g., let us denote such unit $j = M_i$. Then the MAR query evaluates to

$$\begin{aligned} p(\mathbf{E} = \boldsymbol{e}, Z \in \mathcal{I}) &= \int_{\mathcal{I}} \sum_{i=1}^{K} \theta_i \prod_{j=1}^{M_i} L_{ij}(\boldsymbol{e}, z) dZ \\ &= \sum_{i \in I} \theta_i \prod_{j=1}^{M_i} L_{ij}(\boldsymbol{e}_{\phi(L_{ij})}) \left(\int_{\mathcal{I}} dZ \right) + \sum_{i \notin I} \theta_i \prod_{j=1}^{M_i - 1} L_{ij}(\boldsymbol{e}_{\phi(L_{ij})}) \left(\int_{\mathcal{I}} L_{iM_i}(\boldsymbol{e}_{\phi(L_{iM_i})}, z) dZ \right). \end{aligned} \tag{8}$$

On the other hand, evaluating Algorithm 2 would return

$$\begin{aligned} \mathcal{C}(\boldsymbol{e}; \mathcal{I}) &= \sum_{i=1}^{K} \theta_i \prod_{j=1}^{M_i} L_{ij}(\boldsymbol{e}_{\phi(L_{ij})}; \mathcal{I}_{\phi(L_{ij})}) \\ &= \sum_{i \in I} \theta_i \prod_{j=1}^{M_i} L_{ij}(\boldsymbol{e}_{\phi(L_{ij})}) + \sum_{i \notin I} \theta_i \prod_{j=1}^{M_i - 1} L_{ij}(\boldsymbol{e}_{\phi(L_{ij})}) \left(\int_{\mathcal{I}} L_{iM_i}(\boldsymbol{e}_{\phi(L_{iM_i})}, z) dZ \right). \end{aligned} \tag{9}$$

Equation 8 is not equal to Equation 9 in general. Hence, Algorithm 2 is not guaranteed to compute the MAR query without smoothness. For instance, if Z is a discrete variable and $\mathcal{I} = \mathsf{val}(Z)$, each term for $i \in I$ in Equation 9 is missing a factor of $\int_{\mathcal{I}} dZ = |\mathsf{val}(Z)|$ compared to Equation 8, and thus the output of Algorithm 2 lower bounds the marginal.

Therefore, decomposability and smoothness precisely describe all circuit structures that allow for marginals under any parameterization.

4.3 Tractable Computation of the Moments of a Distribution

In essence, smoothness and decomposability enable the tractable computation of the MAR query class by breaking down a large integration problem to smaller and easy-to-compute integrals. This powerful idea can be exploited to tractably compute other kinds of probabilistic query classes involving multivariate integrals. For instance, this is the case for computing the mean and the variance of a probability distribution, or more generally, any of its moments.

Algorithm 3 MOMQUERY($\mathcal{C}, \mathbf{k}$)

Input: a PC $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ over D RVs $\mathbf{X}$, degree parameters $\mathbf{k} \in \mathbb{Z}^D_{\geq 0}$
Output: $\mathbb{M}_{\mathcal{C}}(\mathbf{k})$

```
N ← FEEDFORWARDORDER(G)                    ▷ Order units, inputs before outputs
for each n ∈ N do
    if n is a sum unit then r_n ← Σ_{c∈in(n)} θ_{n,c} r_c
    else if n is a product unit then r_n ← Π_{c∈in(n)} r_c
    else if n is an input distribution unit then r_n ← M_{C_n}(k_{φ(n)})
return r_n                                 ▷ the value of the output of C
```

Definition 22 (MOM query class) *Let $p(\mathbf{X})$ be a joint distribution over RVs $\mathbf{X} = \{X_1, \ldots, X_D\}$. The class* MOM *of moment queries over p is the set of functions that compute:*

$$\mathbb{M}_p(\mathbf{k}) := \int_{\mathsf{val}(\mathbf{X})} x_1^{k_1} x_2^{k_2} \ldots x_D^{k_D} p(\boldsymbol{x}) d\mathbf{X} \tag{10}$$

where $\mathbf{k} = (k_1, \ldots, k_D)$ is a vector of non-negative integers.

Example 19 (Moment query) *Consider the D-dimensional multivariate Gaussians $\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$, whose joint density is given by $p(\mathbf{X})$. Then the first order moment $\mathbb{M}_p(\mathbf{k})$ for $k_i = 1$, $k_j = 0$, $\forall\, j \neq i$ corresponds to the mean of X_i:*

$$\mathbb{M}_p(\mathbf{k}) = \int_{\mathsf{val}(\mathbf{X})} x_1^0 \ldots x_i^1 \ldots x_D^0 p(\boldsymbol{x}) d\mathbf{X} = \int_{\mathsf{val}(\mathbf{X})} x_i p(\boldsymbol{x}) d\mathbf{X} = \mathbb{E}_p[X_i] = \mu_i.$$

Moreover, the second order moment $\mathbb{M}_p(\mathbf{k})$ for $k_i = k_j = 1$, $k_l = 0$, $\forall\, l \neq i, j$ evaluates to:

$$\mathbb{M}_p(\mathbf{k}) = \int_{\mathsf{val}(\mathbf{X})} x_1^0 \ldots x_i^1 \ldots x_j^1 \ldots x_D^0 p(\boldsymbol{x}) d\mathbf{X} = \int_{\mathsf{val}(\mathbf{X})} x_i x_j p(\boldsymbol{x}) d\mathbf{X} = \mathbb{E}_p[X_i X_j] = \Sigma_{i,j} + \mu_i \mu_j,$$

using the fact that $\Sigma_{i,j}$, the covariance of X_i and X_j, is equal to $\mathbb{E}_p[X_i X_j] - \mathbb{E}_p[X_i]\mathbb{E}_p[X_j]$.

Evidently, the class of moment queries is closely related to that of marginal queries in that they both involve multivariate integrals of the distribution. In fact, moment queries can also be computed via a feedforward evaluation of a smooth and decomposable PC.

Proposition 23 *Let $\mathcal{C}$ be a PC over RVs $\mathbf{X} = \{X_1, \ldots, X_D\}$ and $\mathbf{k} = (k_1, \ldots, k_D)$ be non-negative integers. If $\mathcal{C}$ is smooth and decomposable, then Algorithm 3 given $\mathcal{C}$ and $\mathbf{k}$ evaluates to the moment query $\mathbb{M}_{\mathcal{C}}(\mathbf{k})$.*

Therefore, a smooth and decomposable PC can compute moment queries tractably as long as the input distribution units support tractable computation of given moment query.

We now prove above proposition by induction. As the base case, if $\mathcal{C}(\mathbf{X})$ comprises a single input distribution, then Algorithm 3 simply computes its moment of degree $\mathbf{k}$.

Next, suppose the root of $\mathcal{C}(\mathbf{X})$ is a smooth sum unit, i.e., $\mathcal{C}(\mathbf{X}) = \sum_{i=1}^{n} \theta_i \mathcal{C}_i(\mathbf{X})$. Then, we can push the integration in Equation 10 down to the inputs of the sum unit:

$$\begin{aligned}\mathbb{M}_{\mathcal{C}}(\mathbf{k}) &= \int_{\mathsf{val}(\mathbf{X})} x_1^{k_1} x_2^{k_2} \dots x_D^{k_D} \sum_{i=1}^{n} \theta_i \mathcal{C}_i(\boldsymbol{x}) d\mathbf{X} \\ &= \sum_{i=1}^{n} \theta_i \int_{\mathsf{val}(\mathbf{X})} x_1^{k_1} x_2^{k_2} \dots x_D^{k_D} \mathcal{C}_i(\boldsymbol{x}) d\mathbf{X} = \sum_{i=1}^{n} \theta_i \mathbb{M}_{\mathcal{C}_i}(\mathbf{k}).\end{aligned}$$

Thus, if Algorithm 3 computes the moment of a sum unit's inputs, it also computes the moment of the sum unit.

Lastly, suppose the root of $\mathcal{C}(\mathbf{X})$ is a decomposable product unit, i.e., $\mathcal{C}(\mathbf{X}) = \prod_{i=1}^{n} \mathcal{C}_i(\mathbf{X}_i)$ for a partitioning of RVs $\mathbf{X} = \mathbf{X}_1 \cup \dots \cup \mathbf{X}_n$. Then the multivariate integration in Equation 10 decomposes as the following, similar to the decomposition of MAR queries:

$$\begin{aligned}\mathbb{M}_{\mathcal{C}}(\mathbf{k}) &= \int_{\mathsf{val}(\mathbf{X})} x_1^{k_1} x_2^{k_2} \dots x_D^{k_D} \prod_{i=1}^{n} \mathcal{C}_i(\boldsymbol{x}_i) d\mathbf{X} \\ &= \int_{\mathsf{val}(\mathbf{X}_1)} \dots \int_{\mathsf{val}(\mathbf{X}_n)} \boldsymbol{x}_1^{\mathbf{k}_1} \dots \boldsymbol{x}_n^{\mathbf{k}_n} \prod_{i=1}^{n} \mathcal{C}_i(\boldsymbol{x}_i) d\mathbf{X}_1 \dots d\mathbf{X}_n \\ &= \prod_{i=1}^{n} \int_{\mathsf{val}(\mathbf{X}_i)} \boldsymbol{x}_i^{\mathbf{k}_i} \mathcal{C}_i(\boldsymbol{x}_i) d\mathbf{X}_i = \prod_{i=1}^{n} \mathbb{M}_{\mathcal{C}_i}(\mathbf{k}_i),\end{aligned}$$

where $\mathbf{k}_i$ is the vector consisting of entries of $\mathbf{k}$ that corresponds to RVs in $\mathbf{X}_i$. Therefore, Algorithm 3 returns the moment query of any unit if it correctly computes the moment of its input units.

4.4 MAR and Beyond

Here we provide the interested readers additional pointers to relevant literature about the topics touched in this section. Decomposability and smoothness are the two "barebone" properties that the many TPMs that can be cast as PCs commonly assume (cf. Table 2 and Section 11).

In the literature of sum-product networks (SPNs) (Poon and Domingos, 2011), smoothness is called *completeness*. In their original formulation in Poon and Domingos (2011), as probabilistic models over binary RVs, SPNs computing tractable MAR are called *valid*. Sufficient conditions for validity in the case of binary RVs are smoothness and *consistency*, where the latter is a generalization of decomposability. It was later shown by Peharz et al. (2015) that decomposability and smoothness are in fact sufficient for a circuit with tractable, but arbitrary, input nodes to tractably compute MAR. However, consistency still plays a role in characterizing a tractable class of PCs, but for a different query class, MAP, as we will show later in Section 6. Lastly, smoothness and decomposability are proven to be sufficient and necessary conditions for a stricter version of validity, called *stronger validity* by Martens and Medabalimi (2014). Their results leverage the link between SPNs and (set-)multilinear polynomials (cf. Section 5) and properties of the latter representation. Our theoretical result in this section arrives to an analogous, but with a slightly more general conclusion, by extending it to all parameterization of a general PC.

If a PC over discrete RVs is decomposable but not smooth, the latter property can be enforced in polytime with a polynomial increase in its size (Darwiche, 2001b; Shih et al., 2019). However, for a PC over continuous RVs with unbounded support the above algorithms might yield a PC whose normalizing constant is unbounded.

Learning smooth and decomposable PCs, or variants thereof, from data has been addressed in a number of ways in the literature. From the interpretation of smooth sum nodes as mixture models—and hence of PCs as deep mixture models (cf. Section 5)—it follows that the likelihood function of their parameters is not concave, in general. Practical parameter learning schemes for smooth and decomposable PCs include expectation-maximization (Peharz et al., 2016, 2020), variants of (stochastic) gradient descent (Peharz et al., 2019; Jaini et al., 2018; Sharir et al., 2016), (online) Bayesian moment matching alternatives (Rashwan et al., 2016; Jaini et al., 2016; Zhao et al., 2016b), (collapsed) variational optimization routines (Zhao et al., 2016a) or Gibbs sampling schemes for Bayesian optimization (Vergari et al., 2019b; Trapp et al., 2019). Furthermore, the ability of smooth and decomposable PCs to efficiently marginalize over any set of features helps devise hybrid generative-discriminative classifiers (Peharz et al., 2019) and safe semi-supervised learning schemes to train them (Trapp et al., 2017).

Analogously, the clear probabilistic semantic of smooth sums and decomposable products inspired many structure learning routines that leverage commonly adopted ML approaches to learn mixtures and (local) factorizations via clustering and independence testing. The first notable approach is by Dennis and Ventura (2012) using k-means to group RVs in a data matrix. Peharz et al. (2013) take a "bottom-up" approach to learn PCs by greedily merging candidate structures via an information-bottleneck criterion. Gens and Pedro (2013) propose a general high-level scheme called LearnSPN which essentially performs hierarchical co-clustering over the data matrix, by alternately clustering data samples—corresponding to sum nodes—and splits on data columns—corresponding to product nodes—via independence tests. Since then, there have been several improvements of the basic LearnSPN scheme, such as regularization by employing multivariate input distributions and ensembles (Vergari et al., 2015; Rooshenas and Lowd, 2014), by performing an SVD-decomposition (Adel et al., 2015), merging tree-shaped PCs into DAG-shaped ones (Rahman and Gogate, 2016), learning product nodes via multi-view clustering over variables (Jaini et al., 2018), and lowering their complexity by approximate independence testing (Di Mauro et al., 2018). Other variants of LearnSPN include learning PCs with specific inductive biases in terms of the data likelihood distribution (Molina et al., 2017) or on heterogeneous data (Molina et al., 2018; Vergari et al., 2019b).

5. The Many Faces of Probabilistic Circuits

We now present several interpretations of PCs that directly stem from their operational semantics. They will help pose PCs in the broader landscape of probabilistic modeling and will be useful in the following sections.[13]

13. Note that, if not stated otherwise, we introduce these interpretations in the context of unconstrained PCs and as such, they hold also for constrained PCs, as they are proper subclasses of the unconstrained case. Clearly, the interpretations we provide later for constrained PCs will not be valid for the superclass.

5.1 PCs are *not* PGMs.

Even if they are *probabilistic* models expressed via a *graphical* formalism, PCs are *not* PGMs in the classical sense (Koller and Friedman, 2009). In fact, classical PGMs such as Bayesian networks and Markov random fields have a clear *representational semantics*, while the semantics of PCs is clearly *operational*. That is, units in the computational graphs of PCs directly represent how to evaluate the probability distributions they encode, i.e., how to answer probabilistic queries. On the other hand, nodes in a graph of a PGM denote RVs. Edges connecting units in PCs define the order of execution of operations in answering a query, while in PGMs they encode the (conditional) independence assumptions known between the RVs.

Evaluating a query in PGMs is a task demanded to external algorithms that come in many flavors (e.g., variable elimination, message-passing, recursive conditioning, etc.) and whose complexity can vary based on the structural property of the graph they exploit. Section 11.1 will discuss how several PGMs can be readily cast as computational graphs in the framework of PCs. The process of translating one graphical representation into the other, while preserving the underlying probability distribution is called *compilation*.

5.2 PCs are neural networks.

If PCs do not share the same semantics of PGMs, they do with neural networks (NNs). In fact, computational graphs in PCs are peculiar NNs where neurons are constrained to be either input distribution, sum or product units. While sum units output linear transformations of their inputs as in a common pre-activation function in perceptrons, product units implement a form of *multiplicative interaction* (Jayakumar et al., 2019) which can be found in attention mechanism and many modern gating units (Ha et al., 2016; Bahdanau et al., 2014).

As such, a PC is a NN containing two forms of non-linearity: the first provided by the input distribution units warping inputs via their densities or masses, the second by the product units. It is possible to retrieve the interpretation of a more classical feedforward perceptron where a non-linear transformation follows a linear transformation (without bias) by reparameterizing computations in PCs to alternate between the linear and log domain when considering sum and product units (Vergari et al., 2019a). Computational graphs of constrained PCs are sparser than NNs. Mapping PCs to tensorized representations for efficient GPU computations is therefore harder, although recent efforts are closing this gap (Sharir et al., 2016; Vergari et al., 2019a; Peharz et al., 2019, 2020).

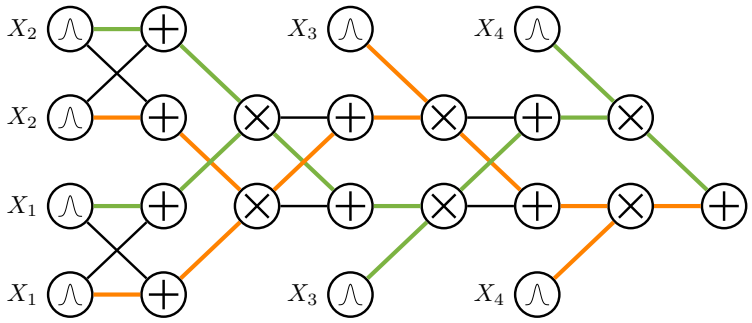
5.3 PCs are polynomials

The adoption of sum and product units as inner neurons in PCs yields the interpretation of PCs as polynomial multivariate functions whose indeterminates are the density functions encoded by the input distribution units. We already exploited this interpretation while constructing bottom-up the *shallow representation of a PC* in order to prove Theorem 19. In the following, we provide a more formal characterization in a top-down fashion, by firstly introducing the notion of an *induced sub-circuit*.

Definition 24 (Induced sub-circuit) *Let $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ be a PC over RVs $\mathbf{X}$. An induced sub-circuit $\mathcal{T}$ built from $\mathcal{G}$ is a DAG recursively built as follows. The output n of $\mathcal{C}$ is the output of $\mathcal{T}$. If n is a product unit in $\mathcal{C}$, then every unit $c \in \mathsf{in}(n)$ and edge $c \to n$ are in $\mathcal{T}$. If n is a sum unit in $\mathcal{C}$, then exactly one of its input unit c and the corresponding weighted edge $c \to n$ are in $\mathcal{T}$.*

Note that each input distribution unit in a sub-circuit $\mathcal{T}$ of a graph $\mathcal{G}$ is also a input unit in $\mathcal{G}$. Furthermore, the DAG $\mathcal{G}$ can be represented as the collection $\{\mathcal{T}_i\}_i$ of all the induced sub-circuit one can enumerate by taking different input units at every sum unit in $\mathcal{G}$. The following example provides some intuition.

Example 20 (Induced sub-circuit) *Consider the PC $\mathcal{C}_A$ over RVs $\mathbf{X} = \{X_1, X_2, X_3, X_4\}$ as shown in Example 10. Two possible induced sub-circuits in it are highlighted in green and orange below.*



It is easy to verify that the number of all distinct sub-circuits in PC $\mathcal{C}_A$ is 32.

Given this "unrolled" representation of $\mathcal{G}$ as a collection of induced sub-circuits, we can now define the polynomial representation of a PC $\mathcal{C}$.

Definition 25 (Circuit polynomial) *Let $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ be a PC over RVs $\mathbf{X}$. For a complete state $\boldsymbol{x} \in \mathsf{val}(\mathbf{X})$, $\mathcal{C}$ computes the following polynomial:*

$$\mathcal{C}(\boldsymbol{x}) = \sum_{\mathcal{T}_i \in \mathcal{G}} \left(\prod_{\theta_j \in \boldsymbol{\theta}_{\mathcal{T}_i}} \theta_j \right) \prod_{c \in \mathcal{T}_i} \mathcal{C}_c^{d_c}(\boldsymbol{x}) = \sum_{\mathcal{T}_i} \theta_{\mathcal{T}_i} \prod_{c \in \mathcal{T}_i} \mathcal{C}_c^{d_c}(\boldsymbol{x}). \tag{11}$$

where $\mathcal{T}_i \in \mathcal{G}$ is a sub-circuit tree in the computational graph as previously defined, $\boldsymbol{\theta}_{\mathcal{T}_i}$ is the collection of weights attached to weighted edges in $\mathcal{T}_i$, $c \in \mathcal{T}_i$ denotes one of its input unit and d_c denotes how many times an input unit is reachable in $\mathcal{T}_i$.

Note that the number of induced sub-circuits, and hence terms in the polynomial representation, *can be exponential in number of the RVs*. While this representation might seem impractical, circuit polynomials facilitates the design of learning schemes (Zhao et al., 2016b; Vergari et al., 2019b), help verifying properties and proving statements about PCs (cf. section 4.2) and provides intuition about the expressiveness of PCs, as discussed next.

A circuit polynomial is a *multilinear* polynomial when d_c becomes 1 for all possible input units. Additionally, it becomes *set-multilinear* (Shpilka and Yehudayoff, 2010) when the sets of scopes of the input distributions participating in the polynomial terms are disjoint, i.e., when the circuit is decomposable (Martens and Medabalimi, 2014). Note that in such a case the induced sub-circuit is a *tree* (and sometimes called induced tree (Zhao et al., 2015)).

5.4 PCs are hierarchical mixture models

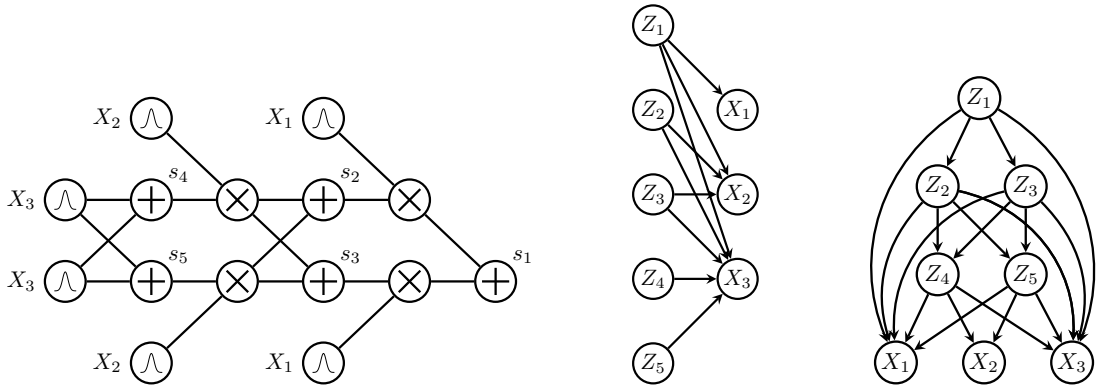
As discussed in Section 3.2, a smooth[14] sum unit in a PC encodes a mixture model whose components are the distributions represented by its input sub-circuits. Moreover, as Example 8 suggests, one can interpret a smooth PC as *marginalizing out a categorical LV* associated to each of its sum units. As a result, PCs are *hierarchical latent variable models*, and more precisely *deep mixture models*. In the following we discuss the LV semantics of PCs and what it entails.

First, note that a circuit polynomial representation (cf. Definition 25) highlights how the hierarchy over the discrete LVs in a PC can be collapsed into a single LV, corresponding to the single sum unit in the PC shallow representation.The number of states that this single LV could assume will correspond to the number of the induced sub-circuits of the PC considered. In other words, a deep circuit *compactly* encodes a mixture with an exponential number of components. This notion of compactness, or expressive efficiency, will be rigorously formalized as function of model size in Section 7. Consequently, each finite and shallow mixture model can be turned into a smooth and shallow PC (if each component represents a tractable distribution). It is an open question under which conditions a shallow mixture model can also be turned into a compact, deep (decomposable) PC in polytime.

The above duality of shallow and deep PCs translates into different ways of graphically representing the dependencies among the associated LVs, i.e., retrieving an i-map structure for the LV hierarchy (Koller and Friedman, 2009). Spanning from the induced-tree representation of a PC, Zhao et al. (2015) translates the dependency structure of a PC into a Bayesian network that is a bipartite graph, where LVs and observed RVs form the two sets of nodes. From this perspective, there are no edges connecting the LVs, and these are conditionally independent one from another given the observed RVs. Alternatively, Peharz et al. (2016) build a DAG whose leaves are the observed RVs and each LV is a node depending on all the other LVs that are associated to sum units that follow it in a feedforward topological order. Retrieving a finer-grain dependency structure has been discussed in Butz et al. (2020), where under certain assumptions concerning the PC being compiled from a PGM (Darwiche and Marquis, 2002), it is possible to identify the original PGM structure (with the LVs made explicit) in a process called *decompilation*.

Example 21 (PCs as hierarchical LV models) *Consider the smooth and decomposable PC $\mathcal{C}$ over RVs $\mathbf{X} = \{X_1, X_2, X_3\}$ as depicted below on the left, whose sum units are labeled as $s_1, \ldots, s_5$. Then, the corresponding LVs $Z_1, \ldots . Z_5$ can be explicitly represented in the bipartite Bayesian network in the center (Zhao et al., 2015) or in a DAG on the right*

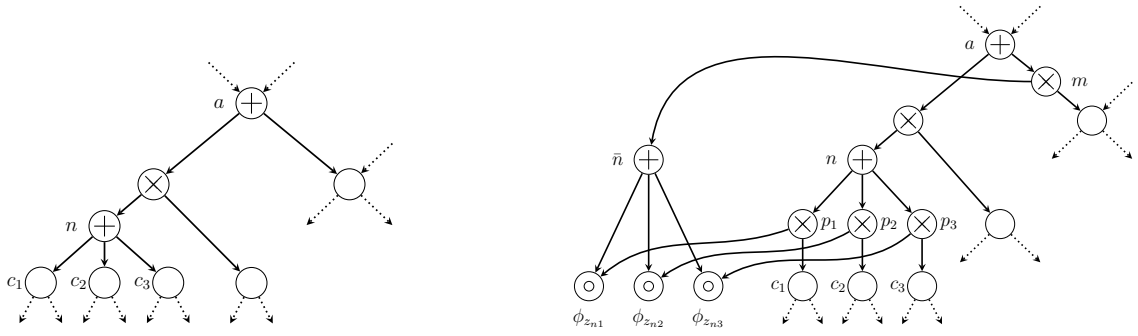
14. Here we assume smoothness because mixture models are classically combinations of homogeneous distributions, i.e., distributions over the same collection of RVs. While smoothness can easily be enforced in a decomposable PC over discrete RVs (Darwiche, 2001b; Shih et al., 2019), the same may not be true for continuous or non-decomposable PCs.



where there is an edge between two LVs if there is an output-input relationship between the corresponding sum units in the computational graph of $\mathcal{C}$.

The interpretation of PCs as deep LV models suggests that the model likelihood of a PC is non-convex. Therefore, circuit parameters are often learned via expectation-maximization (EM) schemes (Dempster et al., 1977). Furthermore, explicitly representing the LVs that a smooth and decomposable PC marginalizes out opens the way to exploit these circuits as feature extractors for representation learning (Vergari et al., 2018, 2019a). In these scenarios, a useful theoretical tool to operate on is the *augmented circuit* $\mathcal{A}$ (Peharz et al., 2016) associated to a smooth and decomposable PC $\mathcal{C}$. An augmented PC materializes the computation involving the LVs $\mathbf{Z}$ of its base PC into some additional units in its computational graph. In other words, it encodes the joint distribution $p(\mathbf{X}, \mathbf{Z})$ and allows to perform probabilistic inference on it with the usual PC inference routines. In a nutshell, augmenting a PC $\mathcal{C}$ over RVs $\mathbf{X}$ into $\mathcal{A}$ is a two-step procedure that requires i) making LVs $\mathbf{Z}$ explicit by introducing a collection of k indicator input distribution units for each $Z_n \in \mathbf{Z}$ associated to a sum unit n in $\mathcal{C}$ and having $\mathsf{val}(Z_n) = k$, ii) while properly connecting them to "switch" on and off the inputs of n and finally iii) *smoothing* the augmented circuit (Darwiche, 2001b; Shih et al., 2019), i.e., making sure each sum node in it is smooth w.r.t. the newly introduced $\mathbf{Z}$.

Example 22 (Augmented computational graph of a PC) *Consider the fragment of the computational graph of one PC $\mathcal{C}$ as depicted on the left, the circuit $\mathcal{A}$ on the right shows $\mathcal{C}$ augmented w.r.t. its sum unit n. To augment a sum node n, first the categor-*



ical LV Z_n with $\mathsf{val}(Z_n) = \{z_{n1}, z_{n2}, z_{n3}\}$ is made explicit by introducing indicator units $\{[\![Z_n = z_{ni}]\!]\}_{i=1}^{3}$. These indicators are multiplied to the inputs of n by the introduction of auxiliary product units p_i. To ensure smoothness of $\mathcal{A}$, sum units that receive input from n or from one of its outputs must be smooth w.r.t. Z_n. If that is not the case, like for the sum unit m in the figure, an additional sum unit $\bar{n}$, also called the twin sum Peharz et al. (2016) is introduced to collect the output of the indicator units and is linked as one input to the non-smooth branch of the m.

For a smooth and decomposable PC $\mathcal{C}$ over $\mathbf{X}$, its augmented computational graph for the augmented joint distribution $p_{\mathcal{A}}(\mathbf{X}, \mathbf{Z})$ is not only smooth and decomposable, but also satisfies another property called determinism, which we will discuss in Section 6. Furthermore, if the input distributions of $\mathcal{C}$ were exponential families, then the augmented joint distribution $p_{\mathcal{A}}(\mathbf{X}, \mathbf{Z})$ would be an exponential family as well.[15] In such a case, optimizing the parameters of $\mathcal{C}$ via EM is equivalent to follow the natural gradient induced by the Fisher information metric of the underlying density Sato (1999).

5.5 Syntactic Transformations

As a PC encodes a function that characterizes a probability distribution in a computational graph, a transformation w.r.t. that distribution, e.g., setting evidence or marginalization as discussed in section 2.1, will also induce a transformation of the computational graph of the PC. We will review these transformations over computational graphs in Section 5.6. Conversely, one could transform the computational graph of a PC but without altering the distribution function encoded in it. We call this kind of operations *syntactical transformations*, some of which we will review below. We say a PC is in its *canonical form* if any application of the following syntactical transformation does not change the computational graph structure.

First, we can easily *make input distributions unique*: if two distinct input distribution units of a PC encode the same function, we can remove one and add its outputs to the set of output units of the remaining unit.

Second, we can transform a PC to have *alternating sum and product units* without changing its distribution; i.e., sum units are fed inputs only by product units or input distribution units and vice versa. To achieve this ordering, one has to iteratively collapse adjacent units of the same type until no such units are left. Consider two computational units n and i in a PC $\mathcal{C}$ where both n and i are of the same kind (e.g., two sum units), and i feeds n as well as other units $c_1, \ldots, c_P$ of the opposite type (e.g., , products). First, we can copy i and disconnect this copy from n (it will still feed $c_1, \ldots, c_P$). Then, i can be collapsed into n by redirecting all of its input units as inputs of n, thereby preserving the distribution represented by $\mathcal{C}$.

Lastly, sum weights of a PC in canonical form is assumed to be non-zero, as we can always efficiently *prune zero weights*. That is, we can simply remove all edges with zero weight and iteratively, outputs before inputs, remove units whose set of outputs is empty.

15. Note that this is not true for $p_{\mathcal{C}}(\mathbf{X})$, in the same way that a mixture of exponential families is not an exponential family distribution.

Note that above syntactic transformations can be performed in time polynomial in the size of the circuit. As such, in the following sections, we will assume a PC to be in its canonical form unless specified otherwise.

5.6 Distribution Transformations

In progress.

5.7 Beyond Basic Representations

In progress.

6. Tractable Circuits for MAP Queries

This section introduces another family of tractable PCs, namely those that can tractably answer MAP queries. We first formally define the class of MAP queries in Section 6.1, and in Section 6.2 characterize the family of PCs that are tractable for MAP in terms of the structural properties of their computational graphs—namely, determinism and consistency.

6.1 The MAP Query Class

As mentioned briefly in previous sections, MAP queries relate to the mode of a distribution, as in the following example.

Example 23 *Consider the probability distribution p_{m} defined over the RVs $\mathbf{X}$ as in the traffic jam example. Then the question "Which combination of roads is most likely to be jammed on Monday at 9:30am?" can be answered by the following* MAP *query:*[16]

$$\arg\max_{\mathbf{j}} p_{\mathsf{m}}(\mathbf{J} = \mathbf{j}, \mathsf{D} = \mathsf{Mon}, \mathsf{T} = 9.5)$$

That is, we want to compute the mode of the distribution among states that agree with the partial state $\mathsf{D} = \mathsf{Mon}, \mathsf{T} = 9.5$.

Definition 26 (MAP query class) *Let $p(\mathbf{X})$ be a joint distribution over RVs $\mathbf{X}$. The class of maximum a posteriori queries (*MAP*) is the set of queries that compute:*

$$\arg\max_{\boldsymbol{q} \in \mathsf{val}(\mathbf{Q})} p(\boldsymbol{q} \mid \boldsymbol{e}) = \arg\max_{\boldsymbol{q} \in \mathsf{val}(\mathbf{Q})} p(\boldsymbol{q}, \boldsymbol{e}) \tag{12}$$

where $\boldsymbol{e} \in \mathsf{val}(\mathbf{E}), \boldsymbol{q} \in \mathsf{val}(\mathbf{Q})$ are partial states for an arbitrary partitioning of the RVs $\mathbf{X}$; i.e., $\mathbf{Q} \cup \mathbf{E} = \mathbf{X}$ and $\mathbf{Q} \cap \mathbf{E} = \emptyset$.

Note that the right-hand side of Equation 12 follows from the fact that maximization is not affected by the normalization constant $p(\boldsymbol{e})$. Sometimes one might be interested not

16. In the Bayesian networks literature, MAP queries are often referred to as *most probable explanation* (*MPE*) queries (Darwiche, 2009) and MAP refers to marginal MAP queries (Koller and Friedman, 2009), where one performs maximization on some subset of $\mathbf{Q}$, marginalizing over the remaining variables (Koller and Friedman, 2009).

only in the mode value of the distribution but also in its associated probability or density. To categorize these queries we can introduce a class in which the arg max operation is replaced by a simple maximization.[17] In fact, to categorize PCs that support tractable MAP inference, we will ask whether they can compute the joint MAP probability, i.e., $\max_{\boldsymbol{q}} p(\boldsymbol{q}, \boldsymbol{e})$. As we will show later, we can also obtain the MAP state using only a single additional pass through such PC.

6.2 Determinism and Consistency

It is well-known that answering a MAP query is in general NP-hard (Shimony, 1994), including for many probabilistic graphical models. Nevertheless, analogous to tractable MAR inference using PCs, enforcing certain structural properties allow us to answer MAP queries in linear time in the size of the circuit. Again, we first study the simplest forms of probabilistic circuits to understand the necessary properties.

For distribution units—the smallest type of PCs—answering MAP queries is as simple as outputting the maximum value or the mode of the encoded distribution.

Example 24 (Tractable densities for MAP) *Consider an input distribution unit encoding a Gaussian density* $p(X) = \mathcal{N}(X; \mu = 1, \sigma = 0.1)$ *as defined in Example 4. Consider asking it to compute the* MAR *query* $\max_x p(x)$*; the distribution unit will output the density at its mode, which is equal to its mean* 1.0*:*

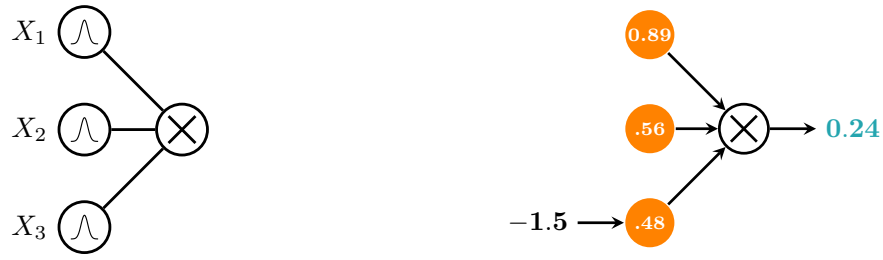
Next, consider a factorized probabilistic model over the partitioning $\mathbf{X} = \mathbf{X}_1 \cup \ldots \cup \mathbf{X}_D$. Analogous to marginalization, the joint maximization problem of Equation 12 can be decomposed in smaller ones which can be solved independently:

$$\max_{\boldsymbol{q} \in \mathsf{val}(\mathbf{Q})} p(\mathbf{Q} = \boldsymbol{q}, \mathbf{E} = \boldsymbol{e}) = \max_{\boldsymbol{q}_1 \in \mathsf{val}(\mathbf{Q}_1)} p(\mathbf{Q}_1 = \boldsymbol{q}_1, \boldsymbol{e}_1) \times \ldots \times \max_{\boldsymbol{q}_D \in \mathsf{val}(\mathbf{Q}_D)} p(\mathbf{Q}_D = \boldsymbol{q}_D, \boldsymbol{e}_D)$$

where the factors over RVs $\{\mathbf{X}_i\}_{i=1}^D$ induce a corresponding partitioning $\{\mathbf{Q}_i, \boldsymbol{e}_i\}_{i=1}^D$ over query RVs $\mathbf{Q}$ and evidence $\boldsymbol{e}$.

Example 25 (MAP queries for factorized models) *Consider the factorized multivariate Gaussian from Example 6 shown below on the left. Then the computational graph below on the right illustrates how to compute the* MAP *query* $\max_{x_1, x_2} p(x_1, x_2, X_3 = -1.5) \approx 0.24$ *(in blue).*

17. If one wants to query the MAP probability and not just the MAP state (i.e., max instead of arg max), then the query and its complexity depends on whether one conditions on the evidence or not (de Campos, 2020). In this section, we focus on querying the MAP state, or equivalently the MAP probability without conditioning.



Lastly, the simplest type of sum units is the mixture model. Consider the mixture of Gaussians from Example 9, shown below on the left. To compute the MAP query $\max_x p(x)$, one may suggest to maximize each mixture component independently, analogous to how the integrals are "broken down" to each component. The computational graph on the right illustrates the output of such computation in blue.

However, this is not equal to the answer $\max_x p(x) = 0.161$. Simply put, one cannot compute the maximum of a convex combination (i.e., mixture) by taking the convex combination of the maximum values of each component. Furthermore, even if the component model classes are tractable representations for MAP, the induced mixture class is *not* tractable for MAP. Recall from Example 8 that we can interpret a mixture model by associating a categorical latent variable (LV) that acts as a switch in selecting the mixture components. This allows us to see why MAP is hard for mixture models over RVs $\mathbf{X}$: maximization over $\mathbf{X}$ requires to first marginalize Z and hence corresponds to performing marginal MAP inference (de Campos, 2011).

While computing MAP inference is hard in general for mixture models, it is tractable for a subclass, represented as sum units satisfying a structural property called *determinism*. Before we discuss the properties necessary for tractable MAP inference, let us define what it means for a circuit to compute MAP, using the notion of maximizer circuits.

Definition 27 (Distribution maximizer) *Let $\mathcal{C}_{\mathsf{L}}$ be an input function of a PC, characterizing some distribution, then its associated function maximizer, denoted as $\mathcal{C}_{\mathsf{L}}^{\max}$, computes $\max_{\boldsymbol{y} \in \mathsf{val}(\mathbf{Y})} \mathcal{C}_{\mathsf{L}}(\boldsymbol{y})$ where $\mathbf{Y} = \phi(n)$.*

Recall from Section 3.1 that input units support tractable computation of MAP queries. Thus, evaluating a distribution maximizer is also tractable.

Definition 28 (Maximizer circuit) *For a given circuit $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ over RVs $\mathbf{X}$, let $\mathcal{C}_{\max} = (\mathcal{G}_{\max}, \boldsymbol{\theta}_{\max})$ be its* **maximizer circuit** *where: (1) $\mathcal{G}_{\max}$ is obtained by replacing every sum node n in $\mathcal{G}$ with a max node, i.e., computing the function $\mathcal{C}_n(\mathbf{X}) = \max_{c \in \mathsf{ch}(n)} \theta_{n,c} \mathcal{C}_c(\mathbf{X})$; and (2) $\boldsymbol{\theta}_{\max}$ is obtained by replacing every input distribution with its distribution maximizer.*

Algorithm 4 MAP($\mathcal{C}, \boldsymbol{e}$)

Input: a PC $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ over RVs $\mathbf{X}$ and a partial state $\boldsymbol{e} \in \mathsf{val}(\mathbf{E})$ for $\mathbf{E} \subseteq \mathbf{X}$
Output: $\max_{\boldsymbol{q} \in \mathsf{val}(\mathbf{Q})} \mathcal{C}(\boldsymbol{q}, \boldsymbol{e})$ for $\mathbf{Q} = \mathbf{X} \setminus \mathbf{E}$

```
N ← FeedforwardOrder(G)                    ▷ Order units, inputs before outputs
for each n ∈ N do
    if n is a sum unit then r_n ← max_{c∈in(n)} θ_{n,c} r_c
    else if n is a product unit then r_n ← ∏_{c∈in(n)} r_c
    else if n is an input unit then r_n ← C_n^max(e_{φ(n)})
return r_n                                 ▷ the value of the output of C
```

Definition 29 *Given a probabilistic circuit $\mathcal{C}$ and its maximizer circuit $\mathcal{C}_{\max}$, we say $\mathcal{C}_{\max}$* **computes the MAP of** *$\mathcal{C}$ if $\mathcal{C}_{\max}(\mathbf{y}) = \max_{\mathbf{z}} \mathcal{C}(\mathbf{y}, \mathbf{z})$ for all subset $\mathbf{Y} \subseteq \mathbf{X}$ and $\mathbf{Z} = \mathbf{X} \setminus \mathbf{Y}$ and its instantiation $\mathbf{y}$.*

Equivalently, we say that $\mathcal{C}$ computes the MAP if the output of Algorithm 4 given $\mathcal{C}$ and any evidence $\boldsymbol{e}$ is equal to the MAP query $\max_{\boldsymbol{q}} \mathcal{C}(\boldsymbol{q}, \boldsymbol{e})$.

Let us now introduce a structural property that enables tractable MAP computations.

Definition 30 (Determinism) *A sum node is* **deterministic** *if, for any fully-instantiated input, the output of at most one of its children is nonzero. A circuit is deterministic if all of its sum nodes are deterministic.*

Determinism is our first structural property that constrains the output of a node, instead of its scope. Note that it is still a restriction on the circuit structure and not its parameters, as the inherent support of leaf nodes given by the structure cannot be altered by the parameters. Chan and Darwiche (2006) showed that maximizer circuits of smooth, decomposable, and deterministic circuits compute the MAP. That is, these properties are sufficient conditions for MAP computations using maximizer circuits.

We now turn our focus to identifying the necessary conditions. First, we observe that decomposability is not in fact necessary, and that a strictly weaker restriction, namely consistency, is enough. We adopt the notion of consistency introduced by Poon and Domingos (2011) for Boolean variables and generalize it to arbitrary (continuous or discrete) random variables as the following:

Definition 31 (Consistency) *A product node is* **consistent** *if each variable that is shared between multiple children nodes only appears in a single leaf node, in the subcircuit rooted at the product node.*[18] *A circuit is consistent if all of its product nodes are consistent.*

Clearly, any decomposable product node is also consistent by definition.

Proposition 32 *Let $\mathcal{G}$ be a circuit structure that is consistent and deterministic. Then for any parameterization $\boldsymbol{\theta}$, the probabilistic circuit $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ computes MAP.*

18. Recall from Section 5.5 that we assume the circuits to be in their canonical form, and thus input distribution units are unique.

We will prove above proposition by inductively showing that Algorithm 4 outputs the MAP for any consistent and deterministic PC. As the base case, if $\mathcal{C}$ is a single distribution unit, the algorithm returns the output of its distribution maximizer which computes MAP by definition. Next, suppose the output unit of $\mathcal{C}$ is a consistent product unit and that Algorithm 4 correctly answers the MAP query for input units to the output unit. Given a query $\max_{\boldsymbol{q}} \mathcal{C}(\boldsymbol{q}, \boldsymbol{e})$, let $\mathbf{Q}_{\text{shared}}$ and $\mathbf{E}_{\text{shared}}$ be the variables in $\mathbf{Q}$ and $\mathbf{E}$, respectively, that appear in more than one input unit. Moreover, we say $\mathbf{Q}_i$ and $\mathbf{E}_i$ are the variables that are in the scope of only the i-th input unit; i.e., $\mathbf{Q} = \mathbf{Q}_{\text{shared}} \cup \mathbf{Q}_1 \cup \cdots \cup \mathbf{Q}_k$ and $\mathbf{E} = \mathbf{E}_{\text{shared}} \cup \mathbf{E}_1 \cup \cdots \cup \mathbf{E}_k$. Because the PC is consistent, each variable in $\mathbf{Q}_{\text{shared}}$ must appear in exactly one distribution unit. Therefore, maximizing each of the k input units independently will result in a consistent MAP state for $\mathbf{Q}_{\text{shared}}$; then the MAP query on $\mathcal{C}$ can be expressed as:

$$\max_{\boldsymbol{q}} \mathcal{C}(\boldsymbol{q}, \boldsymbol{e}) = \max_{\boldsymbol{q}_{\text{shared}}, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_k} \prod_{i=1}^{k} \mathcal{C}_i(\boldsymbol{q}_{\text{shared}}, \boldsymbol{q}_i, \boldsymbol{e}_{\text{shared}}, \boldsymbol{e}_i) = \prod_{i=1}^{k} \max_{\boldsymbol{q}_{\text{shared}}, \boldsymbol{q}_i} \mathcal{C}_i(\boldsymbol{q}_{\text{shared}}, \boldsymbol{q}_i, \boldsymbol{e}_{\text{shared}}, \boldsymbol{e}_i).$$

That is, Algorithm 4 correctly computes the MAP for $\mathcal{C}$. Furthermore, say the output unit of $\mathcal{C}$ is a deterministic sum unit, and again assume that Algorithm 4 computes MAP for its input units. Given any complete evidence, at most one of the input units will return a nonzero value, and thus the sum output unit will evaluate to either zero or the nonzero input, behaving like a maximizer unit. Thus, the MAP query on $\mathcal{C}$ can be broken down as follows:

$$\max_{\boldsymbol{q}} \mathcal{C}(\boldsymbol{q}, \boldsymbol{e}) = \max_{\boldsymbol{q}} \sum_{i=1}^{k} \theta_i \mathcal{C}_i(\boldsymbol{q}, \boldsymbol{e}) = \max_{\boldsymbol{q}} \max_{i} \theta_i \mathcal{C}_i(\boldsymbol{q}, \boldsymbol{e}) = \max_{i} \max_{\boldsymbol{q}} \mathcal{C}_i(\boldsymbol{q}, \boldsymbol{e}).$$

Hence, Algorithm 4, with a single feedforward pass, computes the MAP probability for any consistent and deterministic PC.

To conclude the proof that consistency and determinism are sufficient conditions to tractably answer MAP queries, we describe how to retrieve the MAP state via a backward pass through the PC. After evaluating the circuit as in Algorithm 4 for the MAP probability, we can simply follow the edges that contributed to the output and collect the modes at the input distribution units. Concretely, the MAP state of a sum unit is simply the MAP state of its input unit with the maximum value:

$$\arg\max_{\boldsymbol{q}} \mathcal{C}(\boldsymbol{q}, \boldsymbol{e}) = \arg\max_{\boldsymbol{q}} \theta_i \mathcal{C}_i(\boldsymbol{q}, \boldsymbol{e}), \quad \text{where } i = \arg\max_{i=1,\ldots,k} \max_{\boldsymbol{q}} \theta_i \mathcal{C}_i(\boldsymbol{q}, \boldsymbol{e}).$$

The MAP state of a product unit is obtained by concatenating the MAP states of its inputs, as each input will assign a state to a subset of variables. Note that thanks to consistency, any shared variable will be assigned the same state by input units with such variable in their scopes:

$$\arg\max_{\boldsymbol{q}} \mathcal{C}(\boldsymbol{q}, \boldsymbol{e}) = \left\{ \arg\max_{\boldsymbol{q}_{\text{shared}}, \boldsymbol{q}_i} \mathcal{C}_i(\boldsymbol{q}_{\text{shared}}, \boldsymbol{q}_i, \boldsymbol{e}_{\text{shared}}, \boldsymbol{e}_i) \right\}_{i=1,\ldots,k}.$$

This concludes the proof that any consistent and deterministic PC can answer MAP queries tractably. We will next show that these are indeed necessary conditions for tractable MAP inference.

Theorem 33 *Suppose $\mathcal{G}$ is a circuit structure such that for any parameterization $\boldsymbol{\theta}$, the maximizer circuit $\mathcal{C}_{\max} = (\mathcal{G}_{\max}, \boldsymbol{\theta}_{\max})$ computes the MAP of $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$. Then, $\mathcal{G}$ must be consistent and deterministic.*

We will prove above theorem by first showing necessity of consistency then determinism. Let the scope of $\mathcal{C}$ be $\mathbf{X}$, and consider the MAP query $\max_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} \mathcal{C}(\boldsymbol{x})$, i.e., MAP with no evidence $\boldsymbol{e}$. Maximizer circuit $\mathcal{C}_{\max}$ computes the MAP if and only if any product unit n with inputs $\mathsf{in}(n)$ satisfies $\max_{\mathbf{x}} \mathcal{C}_n(\mathbf{x}) = \prod_{c \in \mathsf{in}(n)} \max_{\mathbf{x}} \mathcal{C}_c(\mathbf{x})$. Suppose there exists an inconsistent product unit n. Let c_1 and c_2 denote its inputs where a variable X appears in input distribution units L_1 and L_2, respectively. We can assume w.l.o.g that c_1 and c_2 are consistent, and thus $\arg\max_{\mathbf{x}} \mathcal{C}_{c_1}(\mathbf{x})$ (resp. $\arg\max_{\mathbf{x}} \mathcal{C}_{c_2}(\mathbf{x})$) must agree with $\arg\max_{\mathbf{x}} \mathcal{C}_{L_1}(\mathbf{x})$ (resp. $\arg\max_{\mathbf{x}} \mathcal{C}_{L_2}(\mathbf{x})$) on the assignment to X. However, we can parameterize the input distributions at L_1 and L_2 such that their respective MAP assignments do not agree on the value of X. In other words, the maximizer circuit can return a value that does not correspond to a consistent assignment of the variables. Therefore, any circuit that computes the MAP must be consistent.

Next, we show that such circuit must also deterministic, by adapting the proof from Choi and Darwiche (2017) that any smooth, decomposable circuit computing the MAP must also be deterministic. Suppose $\mathcal{G}$ is not deterministic, and let n be a non-deterministic sum unit. Hence, there exists one or more complete assignments (denote the set of such assignments $\mathcal{X}$) such that more than one input units of n evaluate to non-zero values. Since $\mathcal{G}$ must be consistent, at least one of those inputs must make the output of the PC be non-zero. To show this, suppose all inputs in $\mathcal{X}$ lead to a circuit output of zero. Then there must exist a unit in the path from n to the circuit output unit that is always multiplied with a unit that outputs 0 for all inputs in $\mathcal{X}$. Note that the variables not in the scope of n can be assigned freely, and thus the output of zero must be caused by assignments to variables in the scope of n. This is only possible if the circuit is inconsistent. Thus, the PC structure does not prohibit the assignments in $\mathcal{X}$ from leading to a non-zero output of the PC. Then we can choose the parameters of the PC such that its MAP state is some $\mathbf{x}$ in $\mathcal{X}$. To evaluate the circuit output $\mathcal{C}(\mathbf{x})$, unit n must perform addition of more than one non-zero inputs. In other words, using the polynomial interpretation of the PC as in Section 5.3, the output of $\mathcal{C}$ given $\mathbf{x}$ cannot be given by a single induced sub-circuit, but rather is a sum of multiple induced sub-circuits. On the other hand, any evaluation of the maximizer circuit corresponds to an induced sub-circuit, because every sum unit has been transformed into a max unit which selects exactly one of its inputs (see Definition 24). Thus, one cannot retrieve the MAP probability $\mathcal{C}(\mathbf{x})$ by transforming it into a maximizer unit, that is, without performing additions.

7. Expressive Efficiency

As discussed in Section 5.4, probabilistic circuits can encode mixture models, and in particular Gaussian mixture models (GMMs) if the input distribution units encode Gaussian densities. Thus, PCs, just like GMMs, are universal approximators of probability densities; i.e., they are *expressive*. However, expressiveness does not describe how *compactly* a model can encode a distribution. For example, the shallow representation of a PC (i.e., its circuit

polynomial), as described in Section 5.3, may be exponentially larger than the original deep PC, even though both are equally expressive.

In order to characterize the ability of PCs to compactly model distributions, we utilize the notion of *expressive efficiency*, also often referred to as *succinctness* (Darwiche and Marquis, 2002).

Definition 34 (Expressive efficiency) *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two families of probabilistic circuits. We say $\mathcal{M}_1$ is* **as expressive efficient as** *$\mathcal{M}_2$ if there exists a polynomial function f such that every PC $\mathcal{C}_2 \in \mathcal{M}_2$ has a circuit $\mathcal{C}_1 \in \mathcal{M}_1$ that represents the same distribution and has size $|\mathcal{C}_1| \leq f(|\mathcal{C}_2|)$.*

Recall that tractability of a family of models is defined with respect to the size of the model (cf. Definition 2). Hence, expressive efficiency is an essential property when characterizing and comparing tractable probabilistic models. For instance, suppose two families of PCs are both tractable for a query class of interest, say MAR, and that one is more expressive efficient than the other. In such a case, one may prefer to use the former circuit family, as MAR inference would be more efficient on it than on the latter.

This section studies the expressive efficiency of the two families of tractable PCs we have seen so far: namely, smooth and decomposable PCs for MAR inference, and consistent and deterministic PCs for MAP inference. We will explore how each of the structural properties affect expressive efficiency and directly compare the expressive efficiency of the two tractable circuit families.

7.1 Expressive Efficiency of Circuits for Marginals

We first study the effect of smoothness and decomposability on the expressive efficiency of PCs. As proven in Section 4, a smooth and decomposable PC allows for tractable computation of marginal queries, i.e., in time polynomial in the size of the circuit. Because marginal queries are well known to be #P-hard (Roth, 1996), this immediately implies that not every distribution can be compactly represented by a smooth and decomposable PC, unless the polynomial hierarchy collapses.

On the other hand, one can easily smooth a probabilistic circuit while preserving decomposabiilty. Suppose $\mathcal{C}_2$ is a decomposable PC over RVs $\mathbf{X}$. Then a smooth and decomposable PC $\mathcal{C}_1$ can be constructed from $\mathcal{C}_2$ as the following. For every non-smooth sum unit n, we replace each of its inputs c with a product unit c' that takes as input c and newly introduced input distribution units:

$$c' = c \cdot \prod_{X \in \phi(n) \setminus \phi(c)} u(X), \tag{13}$$

where $u(X)$ is an unnormalized uniform distribution over X that outputs 1 for every value of X.[19] For instance, $u(X)$ for a Boolean variable X can be written as $[\![X = 1]\!] + [\![X = 0]\!]$. Clearly, the resulting circuit is smooth as all inputs of n have the same scope, and it is still decomposable as the newly added product units are over disjoint variable sets. Moreover, the smoothed circuit $\mathcal{C}_1$ still represents the same distribution as $\mathcal{C}_2$, i.e., for every complete

19. Here we assume that the partition function of the original PC is finite, which is reasonable if one intends to compute marginals using the circuit.

evidence $\boldsymbol{x}$, $\mathcal{C}_1(\boldsymbol{x}) = \mathcal{C}_2(\boldsymbol{x})$. This can be argued recursively using the fact that every unit c of $\mathcal{C}_1$ that is replaced as in Equation 13 still returns the same value for any complete evidence query:

$$\mathcal{C}_{c'}(\boldsymbol{x}) = \mathcal{C}_c(\boldsymbol{x}) \cdot \prod_{X \in \phi(n) \setminus \phi(c)} u(\boldsymbol{x}) = \mathcal{C}_c(\boldsymbol{x}) \cdot 1.$$

Lastly, constructing $\mathcal{C}_1$ as above incurs a polynomial—quadratic, to be specific—size increase as Equation 13 adds at most $|\mathbf{X}|$ number of edges and will be applied at most $|\mathcal{C}_2|$ times. Therefore, the family of smooth and decomposable circuits are equally expressive efficient as that of decomposable ones.

Moreover, combining the fact that smooth and decomposable PCs are strictly less expressive efficient than unconstrained ones and are equally expressive efficient as decomposable ones, we can infer that decomposable PCs must also be strictly less expressive efficient than unconstrained PCs (unless the polynomial hierarchy collapses). Note that we have yet to address the expressive efficiency of smooth PCs compared to unconstrained ones. We will show in the next section that smoothing a non-decomposable circuit is in fact hard.

7.2 Expressive Efficiency of Circuits for MAP

Next, we study the expressive efficiency of circuits when enforcing consistency and determinism, the necessary and sufficient conditions for tractable MAP inference.

Theorem 35 *There exists a function with a circuit of linear size that can compute the* MAP *but no poly-size circuit that computes its* MAR. *(Assuming that the polynomial hierarchy does not collapse)*

Consider a circuit $\mathcal{C}$ of the following form over Boolean variables $\mathbf{X} = \{X_1, \ldots, X_n\}, \mathbf{Y} = \{Y_1, \ldots, Y_r\}$:

$$\prod_{i}^{r} (Y_i \cdot Z_{i1} + (\neg Y_i) \cdot Z_{i2}), \tag{14}$$

where each $Z_{ij} \in \mathbf{X}$. Note that above circuit is consistent and deterministic, and thus allows for computation of MAP using its maximizer circuit. Next, we will show that computing the marginal of above function is a #P-hard problem. The proof is by reduction from SAT′ which was shown to be #P-complete by Valiant (1979a).

- SAT′: Given a monotone 2CNF $\bigwedge_i^r (Z_{i1} \vee Z_{i2})$ where $Z_{ij} \in \mathbf{X}$, output $|\{(\mathbf{x}, \mathbf{t}) : \mathbf{t} \in \{1, 2\}^r, \mathbf{x} \text{ makes } Z_{i,t_i} \text{ true } \forall\, i\}|$.

Note that for a given $\mathbf{x}$, the number of $(\mathbf{x}, \mathbf{t})$ that is counted by SAT′ is 0 if $\mathbf{x}$ does not satisfy the 2CNF formula, and otherwise 2^m where m is the number of clauses i such that both literals Z_{i1} and Z_{i2} are set to true. Given a monotone 2CNF, let us construct a consistent and deterministic circuit by changing the logical AND into a product unit, OR into a sum unit, and adding auxiliary variables Y_i for each clause as in Equation 14. Then for any given $\mathbf{x}$, the marginal $\mathcal{C}(\mathbf{x})$ (with $\mathbf{Y}$ unobserved) computes $\prod(Z_{i1} + Z_{i2})$ which is 0 if $\mathbf{x}$ does not satisfy the formula and 2^m otherwise; the marginal $\mathcal{C}(.)$ is then the solution to

the original SAT′ problem. Hence, there cannot exist a poly-sized circuit for this function that allows marginals, unless the polynomial hierarchy collapses. Furthermore, this implies that the family of smooth and decomposable PCs are not as expressive efficient as that of consistent and deterministic PCs.

In addition, above proof in fact shows the following stronger result: a consistent and non-decomposable circuit cannot be smoothed efficiently, unless the polynomial hierarchy collapses. Consider the following circuit obtained by marginalizing out $\mathbf{Y}$ from Equation 14, which was used as an intermediate step to prove Theorem 35: $\prod_i^r (Z_{i1} + Z_{i2})$. This PC over Boolean RVs $\mathbf{X}$ is consistent but not decomposable. As shown previously, computing the partition function of this PC (i.e., MAR inference) corresponds to the solution to a SAT′ problem, which is #P-hard. Moreover, smoothness and consistency are in fact sufficient conditions for tractable MAR inference for PCs over Boolean RVs (Poon and Domingos, 2011). Therefore, there exists no polynomial sized smooth and consistent circuit for above consistent PC, unless the polynomial hierarchy collapses.

Note that Theorem 35 may not be surprising considering the fact that answering MAP queries is an NP-complete problem (Shimony, 1994), whereas MAR inference is #P-complete. That is, one may expect the family of PCs that can tractably answer a harder class of queries, namely MAR, to be more expressive efficient. However, not only are smooth and decomposable PCs unable to readily answer MAP queries via maximizer circuit representations, but they are in fact not strictly more expressive efficient than the PCs for tractable MAP.

Theorem 36 (Choi and Darwiche (2017)) *There exists a function with linear-size circuit that can compute marginals but no poly-size circuit that computes its MAP. (Assuming that the polynomial hierarchy does not collapse)*

We briefly describe the proof to above theorem shown in Choi and Darwiche (2017). Any naive Bayes network over discrete RVs can be represented as a linear-size smooth and decomposable probabilistic circuit (see Section 11.1). The marginal feature distribution from a naive Bayes distribution can be represented as a PC by marginalizing out the class variable, i.e., setting all input distribution units for the class variable to 1. A poly-size circuit that computes the MAP of this marginal distribution then computes the marginal MAP of the original naive Bayes in polytime. However, marginal MAP is known to be NP-hard for naive Bayes (de Campos, 2011). Therefore, even though marginal inference is computationally harder than MAP inference, there still exist distributions with compact PC representation for tractable marginals but not tractable MAP. Furthermore, this immediately implies that the family of consistent and deterministic PCs are strictly less expressive efficient than unconstrained PCs.

8. Tractable Circuits for Marginal MAP Queries

Let us next study tractable circuits for more advanced query classes, an example being the class of marginal MAP queries. We first formally define the query class and introduce a structural property—marginal determinism—that will enable tractable inference of marginal MAP queries in conjunction with previously discussed properties.

8.1 The MMAP Query Class

As briefly discussed in Section 2.2, marginal MAP queries subsume aspects of both marginal and MAP inference.

Example 26 *Consider the probability distribution p_{m} defined over the RVs $\mathbf{X}$ as in the traffic jam example. Then the question "Which combination of roads is most likely to be jammed on Monday?" can be answered by the following marginal MAP query:*

$$\arg\max_{\mathbf{j}} p_{\mathsf{m}}(\mathbf{J} = \mathbf{j}, \mathsf{D} = \mathsf{Mon}),$$

where $\mathbf{J}$ denote the set of traffic jam indicator variables.

In other words, we wish to find the state that maximizes the distribution that agrees with $\mathsf{D} = \mathsf{Mon}$ while marginalizing out the Time variable. Note the close resemblance to the query in Example 23: in fact the only difference is that T is neither a query variable nor in the evidence. Next, we define a generalized version of the marginal MAP query class and show how this subsumes the more classical definition.

Definition 37 (MMAP query class) *Let $p(\mathbf{X})$ be a joint distribution over RVs $\mathbf{X}$. The class of marginal MAP queries* MMAP *is the set of queries that compute:*

$$\arg\max_{\boldsymbol{q} \in \mathsf{val}(\mathbf{Q})} p(\mathbf{Q} = \boldsymbol{q} \mid \mathbf{E} = \boldsymbol{e}, \mathbf{Z} \in \boldsymbol{\mathcal{I}}) = \arg\max_{\boldsymbol{q} \in \mathsf{val}(\mathbf{Q})} \int_{\boldsymbol{\mathcal{I}}} p(\boldsymbol{q}, \boldsymbol{e}, \boldsymbol{z})\ d\mathbf{Z} \tag{15}$$

where $\mathbf{Q}$, $\mathbf{E}$, and $\mathbf{Z}$ form a partitioning of RVs $\mathbf{X}$, $\boldsymbol{e} \in \mathsf{val}(\mathbf{E})$ is a partial state, and $\boldsymbol{\mathcal{I}} = \mathcal{I}_1 \times \cdots \times \mathcal{I}_k$ are intervals, each of which is defined over the domain of its corresponding RV in $\mathbf{Z}$: $\mathcal{I}_i \subseteq \mathsf{val}(Z_i)$ for $i = 1, \ldots, k$.

Similar to MAP queries, the right-hand side of Equation 15 holds because maximization is not affected by the normalization constant $p(\mathbf{E} = \boldsymbol{e}, \mathbf{Z} \in \boldsymbol{\mathcal{I}})$. Moreover, as shown for marginal queries, the integral over $\boldsymbol{\mathcal{I}}$ consists of k integrals as follows:

$$\arg\max_{\boldsymbol{q} \in \mathsf{val}(\mathbf{Q})} \int_{\mathcal{I}_1} \int_{\mathcal{I}_2} \cdots \int_{\mathcal{I}_k} p(\boldsymbol{q}, \boldsymbol{e}, z_1, z_2, \ldots, z_k)\, dZ_k \cdots dZ_2\, dZ_1.$$

When $\mathbf{Q}$ and $\mathbf{E}$ are clear from context, we use the shorthand $\arg\max_{\boldsymbol{q}} p(\boldsymbol{q}, \boldsymbol{e})$ to implicitly denote marginalizing $\mathbf{Z} = \mathbf{X} \setminus (\mathbf{Q} \cup \mathbf{E})$ over their domains $\mathsf{val}(\mathbf{Z})$, analogous to the shorthand for marginal queries. This corresponds to a more commonly used definition of marginal MAP for PGMs.

The class of marginal MAP queries clearly subsumes both MAP and marginal queries by setting the query variable set $\mathbf{Q}$ to $\mathbf{X}$ and $\emptyset$, respectively. Again, we will show that marginal MAP queries can be computed tractably via bottom-up evaluations of circuits.

Algorithm 5 MMAP$(\mathcal{C}, \boldsymbol{e}, \boldsymbol{\mathcal{I}})$

Input: a PC $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ over RVs $\mathbf{X}$, a partial state $\boldsymbol{e} \in \mathsf{val}(\mathbf{E})$ for $\mathbf{E} \subseteq \mathbf{X}$, and a set of integration domains $\boldsymbol{\mathcal{I}}$ for $\mathbf{Z} \subseteq \mathbf{X}$.
Output: $\max_{\boldsymbol{q} \in \mathsf{val}(\mathbf{Q})} \mathcal{C}(\boldsymbol{q}, \boldsymbol{e}, \boldsymbol{\mathcal{I}})$ for $\mathbf{Q} = \mathbf{X} \setminus (\mathbf{E} \cup \mathbf{Z})$

```
N ← FeedforwardOrder(G)                          ▷ Order units, inputs before outputs
for each n ∈ N do
    if n is a sum unit then
        if φ(n) ∩ Q = ∅ then r_n ← Σ c ∈ in(n) θ_{n,c} r_c
        else if φ(n) ∩ Q ≠ ∅ then r_n ← max_{c∈in(n)} θ_{n,c} r_c
    else if n is a product unit then r_n ← Π_{c∈in(n)} r_c
    else if n is an input unit then r_n ← C_n^{+,max}(e_{φ(n)}; I_{Z_{φ(n)}})
return r_n                                       ▷ the value of the output of C
```

8.2 Marginal Determinism

Definition 38 (Sum-maximizer circuit) *Let $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ be a probabilistic circuit over RVs $\mathbf{X}$. We say $\mathcal{C}_{+,\max} = (\mathcal{G}_{+,\max}, \boldsymbol{\theta}_{\max})$ is a* **sum-maximizer circuit** *for $\mathcal{C}$ if: (1) $\mathcal{G}_{+,\max}$ is obtained by replacing a subset of sum nodes with max nodes; and (2) $\boldsymbol{\theta}_{\max}$ is obtained by replacing every input distribution with its distribution marginal maximizer, which marginalizes out some variables and maximizes over others.*

Definition 39 *Given a subset of variables $\mathbf{Q} \subseteq \mathbf{X}$, we say a sum-maximizer circuit $\mathcal{C}_{+,\max}$* **computes the marginal MAP of $\mathcal{C}$ over $\mathbf{Q}$** *if $\mathcal{C}_{+,\max}(\boldsymbol{e}; \mathcal{I}) = \max_{\boldsymbol{q}} \int_{\mathcal{I}} \mathcal{C}(\boldsymbol{z}, \boldsymbol{q}, \boldsymbol{e})\, d\mathbf{Z}$ for any subset $\mathbf{E} \subseteq \mathbf{X}$ and instantiation $\boldsymbol{e}$, and intervals $\mathcal{I}$ on $\mathbf{Z} = \mathbf{X} \setminus (\mathbf{E} \cup \mathbf{Q})$.*

First, if a circuit computes the marginal MAP with no query variables (i.e., computes marginals), we know that it must be smooth and decomposable from Theorem 19. In addition, if a circuit computes the marginal MAP with $\mathbf{X}$ as the query variable set (i.e., computes MAP), then it must be consistent and deterministic from Theorem 33. Nevertheless, unlike for MAR or MAP queries, whether a circuit computes the marginal MAP query depends on a subset of query variables $\mathbf{Q}$. Therefore, the structural properties that allow for tractable marginal MAP computation also depend on such subset.

Definition 40 (Marginal determinism) *Given a subset of variables $\mathbf{Q} \subseteq \mathbf{X}$, a sum node is* **marginal deterministic w.r.t. $\mathbf{Q}$** *if for any partial state $\boldsymbol{q} \in \mathsf{val}(\mathbf{Q})$, the output of at most one of its input units is nonzero. A circuit is marginal deterministic w.r.t. $\mathbf{Q}$ if all sum nodes containing variables in $\mathbf{Q}$ are marginal deterministic.*

Just like determinism, marginal determinism is also a structural property as it is a restriction on the supports of certain nodes restricted to the set of variables $\mathbf{Q}$, which are determined by the inherent support of input distribution units and the computational graph structure. We are now ready to prove the sufficient conditions for tractable computation of marginal MAP queries using circuits.

Proposition 41 *Suppose $\mathcal{G}$ is a circuit structure that is smooth, decomposable, and marginal deterministic w.r.t. a subset of variables $\mathbf{Q} \subseteq \mathbf{X}$. Let $\mathcal{G}_{+,\max}$ be obtained from $\mathcal{G}$ by replacing any sum node whose scope includes variables in $\mathbf{Q}$ with max nodes. Then, for any parameterization $\boldsymbol{\theta}$, $\mathcal{C}_{+,\max} = (\mathcal{G}_{+,\max}, \boldsymbol{\theta}_{\max})$ computes the marginal MAP of $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ over $\mathbf{Q}$.*

We prove above statement by induction. First, an input unit for $\mathbf{Q}$, $\mathbf{Z}$, or $\mathbf{E}$ variable is maximized, marginalized, or complete-evidence computed, respectively; this computes the correct marginal MAP for the input distributions.

Next, suppose the inputs of the root node compute the marginal MAP of the corresponding subcircuits. The root node can then be one of three types: max, sum, and product. Consider a max node. Then $\mathbf{Q}$ must be non-empty by construction of sum-maximizer circuits. The root computes the marginal MAP due to smoothness and marginal determinism as follows:

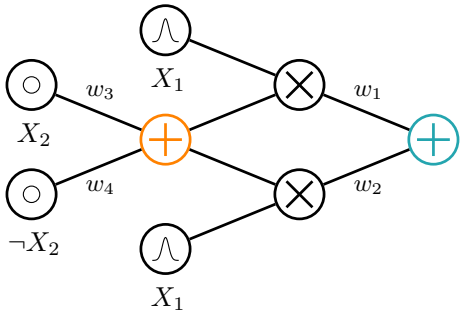
$$\begin{aligned}
&\max_{\boldsymbol{q}} \int_{\mathcal{I}} \mathcal{C}(\mathbf{Z}, \boldsymbol{q}, \boldsymbol{e})\, d\mathbf{Z} = \max_{\boldsymbol{q}} \int_{\mathcal{I}} \sum_i \theta_i \mathcal{C}_i(\mathbf{Z}, \boldsymbol{q}, \boldsymbol{e})\, d\mathbf{Z} \\
&= \max_{\boldsymbol{q}} \sum_i \theta_i \int_{\mathcal{I}} \mathcal{C}_i(\mathbf{Z}, \boldsymbol{q}, \boldsymbol{e})\, d\mathbf{Z} = \max_{\boldsymbol{q}} \max_i \theta_i \int_{\mathcal{I}} \mathcal{C}_i(\mathbf{Z}, \boldsymbol{q}, \boldsymbol{e})\, d\mathbf{Z} \qquad (16) \\
&= \max_i \theta_i \left(\max_{\boldsymbol{q}} \int_{\mathcal{I}} \mathcal{C}_i(\mathbf{Z}, \boldsymbol{q}, \boldsymbol{e})\, d\mathbf{Z} \right) = \max_i \theta_i \mathcal{C}_{i_{+,\max}}(\boldsymbol{e}; \mathcal{I}) = \mathcal{C}_{+,\max}(\boldsymbol{e}; \mathcal{I})
\end{aligned}$$

Equation 16 is obtained using marginal determinism: for each $\boldsymbol{q}$, at most one term i will be non-zero, and thus the sum is equivalent to maximization. Furthermore, if the root is a sum node, $\mathbf{Q}$ must be empty by construction. Therefore, the sum-maximizer circuit does not have any max node. Such circuit then computes the marginal, which is equivalent to marginal MAP given no $\mathbf{Q}$ variables. Lastly, suppose the root is a decomposable product node where the variable sets $\mathbf{Z}, \mathbf{Q}, \mathbf{E}$ are partitioned into $\mathbf{Z}_i, \mathbf{Q}_i, \mathbf{E}_i$, respectively, for each child note $i = 1, \ldots, k$. Then it also computes the marginal MAP as follows:

$$\begin{aligned}
&\max_{\boldsymbol{q}} \int_{\mathcal{I}} \mathcal{C}(\mathbf{Z}, \boldsymbol{q}, \boldsymbol{e})\, d\mathbf{Z} = \max_{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_k} \int_{\mathcal{I}} \prod_i \mathcal{C}_i(\mathbf{Z}_i, \boldsymbol{q}_i, \boldsymbol{e}_i)\, d\mathbf{Z} \\
&= \max_{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_k} \prod_i \left(\int_{\mathcal{I}_i} \mathcal{C}_i(\mathbf{Z}_i, \boldsymbol{q}_i, \boldsymbol{e}_i)\, d\mathbf{Z}_i \right) = \prod_i \left(\max_{\boldsymbol{q}_i} \int_{\mathcal{I}_i} \mathcal{C}_i(\mathbf{Z}_i, \boldsymbol{q}_i, \boldsymbol{e}_i)\, d\mathbf{Z}_i \right) \\
&= \prod_i \mathcal{C}_{i_{+,\max}}(\boldsymbol{e}_i; \mathcal{I}_i) = \mathcal{C}_{+,\max}(\boldsymbol{e}; \mathcal{I})
\end{aligned}$$

We can apply above observations recursively down to the input distribution units to conclude the proof of Proposition 41. Hence, assuming that computing marginal MAP is tractable for each input distribution, it is also tractable for the PC.

Nevertheless, while smoothness, decomposability, and marginal determinism lead to tractable computation of marginal MAP queries, these conditions are not always necessary for a sum-maximizer circuit to compute MMAP of its associated probabilistic circuit. Consider the following circuit for example.



For $\mathbf{Q} = \{X_2\}$, above circuit is not marginal deterministic as the output unit (in blue) is not marginal deterministic w.r.t. to X_2. However, the sum-maximizer circuit where just the sum node over X_2 (in orange) is replaced with a max node computes the marginal MAP over $\mathbf{Q}$. We can easily see this from the circuit polynomial:

$$\begin{aligned}\mathcal{C}(x_1, x_2) &= w_1 f_1(x_1)(w_3[\![x_2 = 1]\!] + w_4[\![x_2 = 0]\!]) + w_2 f_2(x_1)(w_3[\![x_2 = 1]\!] + w_4[\![x_2 = 0]\!]) \\ &= (w_1 f_1(x_1) + w_2 f_2(x_1)) \cdot (w_3[\![x_2 = 1]\!] + w_4[\![x_2 = 0]\!]),\end{aligned}$$

where f_1, f_2 denote the leaf distributions on X_1. Because the circuit polynomial decomposes, the marginal MAP over $\mathbf{Q}$ is:

$$\begin{aligned}\max_{x_2} \int \mathcal{C}(x_1, x_2) dX_1 &= \max_{x_2}(w_3[\![X_2 = 1]\!] + w_4[\![X_2 = 0]\!]) \int (w_1 f_1(x_1) + w_2 f_2(x_1)) dX_1 \\ &= \max\{w_3, w_4\} \left(w_1 \int f_1(x_1) dX_1 + w_2 \int f_2(x_1) dX_1 \right).\end{aligned}$$

Note that this is exactly the polynomial computed by the sum-maximizer circuit described previously. Therefore, the properties in Proposition 41 are not the necessary conditions for tractable MMAP computation, which are currently left open.

Remark 42 *Approaches to similar Algorithm 5 were first proposed by Huang et al. (2006)*[20] *and Oztok and Darwiche (2015) for PCs over Boolean RVs. We relax the structural constraints required to apply the algorithm. In particular, they require that every sum node is associated with a variable (or a set of variables) with respect to which it is marginal deterministic, and that variables appear in the circuit in such an order that no node whose scope includes a variable in* $\mathbf{Q}$ *is an input to a node that is not associated with any variable in* $\mathbf{Q}$*. Such circuit must not only be marginal deterministic w.r.t.* $\mathbf{Q}$ *but is also deterministic, while Proposition 41 does not require the PC to be deterministic to apply Algorithm 5.*

8.3 Tractable Computation of Information-Theoretic Measures

We have shown how marginal determinism can be leveraged to break down a hard query such as marginal MAP. Here we demonstrate how to exploit this idea to tractably compute other types of queries—namely, information-theoretic measures such as marginal entropy and mutual information.

20. Huang et al. (2006) proposed an algorithm to compute upper bounds on marginal MAP probability on decision DNNFs, but it can be inferred that given a certain variable ordering in the circuit, the algorithm exactly computes MMAP.

Given a probabilistic circuit $\mathcal{C}$ representing a normalized distribution over RVs $\mathbf{X}$, its *marginal entropy* over the set of RVs $\mathbf{Y} \subseteq \mathbf{X}$ is defined as:[21]

$$\mathbb{H}_{\mathcal{C}}(\mathbf{Y}) = -\int_{\mathsf{val}(\mathbf{Y})} p_{\mathcal{C}}(\boldsymbol{y}) \log p_{\mathcal{C}}(\boldsymbol{y}) d\mathbf{Y}.$$

Proposition 43 *Suppose $\mathcal{C}$ is a PC over variables $\mathbf{X}$ that is smooth, decomposable, and marginal deterministic w.r.t. $\mathbf{Y}$. Then its marginal entropy $\mathbb{H}_{\mathcal{C}}(\mathbf{Y})$ can be computed in time linear in its size, if its input distributions allow tractable computation of marginal entropy.*

First, because $\mathcal{C}$ is smooth and decomposable, it computes marginals, and the marginal entropy can be written in terms of the partial state computation as follows:

$$\mathbb{H}_{\mathcal{C}}(\mathbf{Y}) = -\int_{\mathsf{val}(\mathbf{Y})} p_{\mathcal{C}}(\boldsymbol{y}) \log p_{\mathcal{C}}(\boldsymbol{y}) d\mathbf{Y} = -\int_{\mathsf{val}(\mathbf{Y})} \mathcal{C}(\boldsymbol{y}) \log \mathcal{C}(\boldsymbol{y}) d\mathbf{Y}.$$

If the root of $\mathcal{C}$ is a decomposable product node $\mathcal{C}(\mathbf{X}) = \prod_i \mathcal{C}_i(\mathbf{X}_i)$, and $\mathbf{Z} = \mathbf{X} \setminus \mathbf{Y}$ then:

$$\mathbb{H}_{\mathcal{C}}(\mathbf{Y}) = -\int_{\mathsf{val}(\mathbf{Y})} \left(\prod_i \mathcal{C}(\boldsymbol{y}_i)\right) \log \left(\prod_i \mathcal{C}(\boldsymbol{y}_i)\right) \, d\mathbf{Y} = -\sum_i \int_{\mathsf{val}(\mathbf{Y})} \left(\prod_i \mathcal{C}(\boldsymbol{y}_i)\right) \log \mathcal{C}(\boldsymbol{y}_i) \, d\mathbf{Y}$$

$$= \sum_i \left(-\int_{val(\mathbf{Y}_i)} \mathcal{C}_i(\boldsymbol{y}_i) \log \mathcal{C}_i(\boldsymbol{y}_i) \, d\mathbf{Y}_i\right) \left(\prod_{j \neq i} \int_{\mathsf{val}(\mathbf{Y}_j)} \mathcal{C}_j(\boldsymbol{y}_j) \, d\mathbf{Y}_j\right) = \sum_i \mathbb{H}_{\mathcal{C}_i}(\mathbf{Y}_i) \prod_{j \neq i} Z_j$$

where Z_j denotes the partition function for $\mathcal{C}_j$. On the other hand, if the root of $\mathcal{C}$ is sum node that is smooth and marginal deterministic w.r.t. $\mathbf{Y}$ such that $\mathcal{C}(\mathbf{X}) = \sum_{i=1}^k w_i \mathcal{C}_i(\mathbf{X})$:

$$\begin{aligned}
\mathbb{H}_{\mathcal{C}}(\mathbf{Y}) &= -\int_{\mathsf{val}(\mathbf{Y})} \left(\sum_{i=1}^k w_i \mathcal{C}_i(\boldsymbol{y})\right) \log \left(\sum_{i=1}^k w_i \mathcal{C}_i(\boldsymbol{y})\right) d\mathbf{Y} \\
&= -\sum_{i=1}^k w_i \int_{\mathsf{val}(\mathbf{Y})} [\![\boldsymbol{y} \in \mathsf{supp}(\mathcal{C}_i)|_{\mathbf{Y}}]\!] \mathcal{C}_i(\boldsymbol{y}) \log \left(\sum_{i=1}^k w_i \mathcal{C}_i(\boldsymbol{y})\right) d\mathbf{Y} \qquad (17) \\
&= -\sum_{i=1}^k w_i \int_{\mathsf{supp}(\mathcal{C}_i)|_{\mathbf{Y}}} \mathcal{C}_i(\boldsymbol{y}) \log(w_i \mathcal{C}_i(\boldsymbol{y})) d\mathbf{Y} \qquad (18) \\
&= -\sum_{i=1}^k w_i \left(\int_{\mathsf{supp}(\mathcal{C}_i)|_{\mathbf{Y}}} \mathcal{C}_i(\boldsymbol{y}) \log \mathcal{C}_i(\boldsymbol{y}) d\mathbf{Y} + \int_{\mathsf{supp}(\mathcal{C}_i)|_{\mathbf{Y}}} \mathcal{C}_i(\boldsymbol{y}) \log(w_i) d\mathbf{Y}\right) \\
&= \sum_{i=1}^k w_i \left(\mathbb{H}_{\mathcal{C}_i}(\mathbf{Y}) - \log(w_i) \cdot Z_i\right)
\end{aligned}$$

where again Z_i is the partition function for $\mathcal{C}_i$. The key idea is in Equations 17 and 18 which hold due to marginal determinism. A marginal deterministic sum node represents a mixture of components whose supports, restricted to $\mathbf{Y}$, are non-overlapping. Hence, we

21. Note that for continuous and mixed continuous-discrete RVs, this is the definition of the differential entropy and for discrete RVs it is the classical information theory entropy.

can break down the computation of marginal entropy, which involves computing a logarithm of a mixture, into computations over the smaller components. Recursively applying above results, we can tractably compute the marginal entropy as long as it is tractable for the input distribution units.

An immediate implication of Proposition 43 is the tractable computation of joint entropy (Shih and Ermon, 2020), a special case of marginal entropy over the entire set of variables $\mathbf{X}$.

Corollary 44 *Suppose $\mathcal{C}$ is a smooth, deterministic, and decomposable PC over variables $\mathbf{X}$. Then its joint entropy $\mathbb{H}_{\mathcal{C}}(\mathbf{X})$ can be computed in time linear in its size, if its input distributions allow tractable computation of entropy.*

This follows directly from the fact that marginal determinism w.r.t. $\mathbf{X}$ is equivalent to determinism.

In addition, tractable computation of marginal entropy also leads to that of a number of other information-theoretic measures. For instance, consider the mutual information between subsets of variables. Formally, given a probabilistic circuit $\mathcal{C}$ representing a normalized distribution over RVs $\mathbf{X}$ and $\mathbf{Y}, \mathbf{Z} \subset \mathbf{X}$, the *mutual information* between $\mathbf{Y}$ and $\mathbf{Z}$ is defined as:

$$\mathsf{MI}_{\mathcal{C}}(\mathbf{Y}; \mathbf{Z}) = \int_{\mathsf{val}(\mathbf{Z})} \int_{\mathsf{val}(\mathbf{Y})} p_{\mathcal{C}}(\boldsymbol{y}, \boldsymbol{z}) \log \frac{p_{\mathcal{C}}(\boldsymbol{y}, \boldsymbol{z})}{p_{\mathcal{C}}(\boldsymbol{y}) p_{\mathcal{C}}(\boldsymbol{z})} \, d\mathbf{Y} \, d\mathbf{Z}.$$

It is easy to check that mutual information can also be written in terms of marginal entropy:

$$\mathsf{MI}_{\mathcal{C}}(\mathbf{Y}; \mathbf{Z}) = \mathbb{H}_{\mathcal{C}}(\mathbf{Y}) + \mathbb{H}_{\mathcal{C}}(\mathbf{Z}) - \mathbb{H}_{\mathcal{C}}(\mathbf{Y} \cup \mathbf{Z}).$$

Therefore, if $\mathcal{C}$ is smooth, decomposable and marginal deterministic w.r.t. $\mathbf{Y}$, $\mathbf{Z}$ and $\mathbf{Y} \cup \mathbf{Z}$, then we can tractably compute $\mathsf{MI}_{\mathcal{C}}(\mathbf{Y}; \mathbf{Z})$.

8.4 Expressive Efficiency of Circuits for Marginal MAP

Unlike the structural properties in previous sections, marginal determinism is defined with respect to a subset of variables $\mathbf{Q} \subseteq \mathbf{X}$. Thus, the expressive efficiency of PCs for tractable marginal MAP naturally also depends on the set $\mathbf{Q}$. Specifically, for each $\mathbf{Q}$, we consider the family of PCs that are smooth, decomposable, and marginal deterministic w.r.t. $\mathbf{Q}$.

First, the family of tractable MMAP circuits for $\mathbf{Q} = \emptyset$ corresponds to that of smooth and decomposable PCs (i.e., tractable circuits for MAR), whereas for $\mathbf{Q} = \mathbf{X}$ it corresponds to smooth, decomposable, and deterministic PCs (i.e., tractable for both MAR and MAP). As we saw in Section 7.2, the former is strictly more expressive efficient than the latter. Next we consider the expressive efficiency of PCs that are tractable for MMAP w.r.t. some non-empty $\mathbf{Q} \subset \mathbf{X}$.

Recall that any naive Bayes network over discrete RVs can be represented by a linear-sized smooth, deterministic, and decomposable PC. Such circuit is tractable for MMAP w.r.t. both $\mathbf{Q} = \emptyset$ and $\mathbf{Q} = \mathbf{X}$. Nevertheless, marginal MAP w.r.t. $\mathbf{Q} = \mathbf{X} \setminus \{C\}$, the set of variables excluding the class variable, is known to be NP-hard for naive Bayes (de Campos, 2011). Therefore, the family of PCs tractable for MMAP w.r.t. some $\mathbf{Q}$ is not necessarily as expressive efficient as those for MMAP w.r.t. either a subset or superset of $\mathbf{Q}$.

Lastly, let us remark on why we allow the dependence on query variable set $\mathbf{Q}$ when determining tractability of PCs for MMAP. Indeed, the family of tractable PCs for MAR or MAP was required to be tractable for all possible subsets $\mathbf{E}$. To answer this, we look to the expressiveness of marginal deterministic PCs. Let us consider the family of PCs that are smooth, decomposable, and marginal deterministic w.r.t. all possible subsets $\mathbf{Q} \subseteq \mathbf{X}$. Clearly, a fully factorized distribution, i.e., a PC with decomposable product units and no sum units, satisfies this property. If a PC in this family has a sum unit, its input units must have the same scope by smoothness, and it must be marginal deterministic with respect to every variable in its scope. Thus, the support of the sum node $n = \sum_i n_i$ will be of the form

$$\bigsqcup_i \prod_{X \in \phi(n)} S_{i,X},$$

where $S_{i,X} \subset \mathsf{val}(X)$ such that $S_{i,X} \cap S_{j,X} = \emptyset$ for any $i \neq j$. In other words, the supports of the input units will not only be disjoint but will be a Cartesian product of disjoint subsets on each variable. An immediate consequence is that the support of such sum node cannot be $\mathsf{val}(\phi(n))$. Therefore, this family of PCs cannot contain any distribution whose support is $\mathsf{val}(\mathbf{X})$ without being fully factorized,[22] and thus is not expressive.

9. Tractable Circuits for Pairwise Queries

In the previous sections, we have discussed a number of queries that operate on a single distribution. However, one may also wish to describe properties of a pair of distributions; for instance, how similar two given distributions over the same set of RVs are.

In this section, we study some examples of such pairwise queries, and show how probabilistic circuits can again be used for tractable computation of such queries. Similar to how single-distribution queries were computed via bottom-up evaluations of circuits satisfying certain structural properties, we will show that many pairwise queries can also be tractably computed in a bottom-up fashion by considering pairs of circuit nodes.

9.1 Kullback–Leibler Divergence

An example of pairwise query is the *Kullback–Leibler (KL) divergence*, which is a way to measure how different two distributions are.

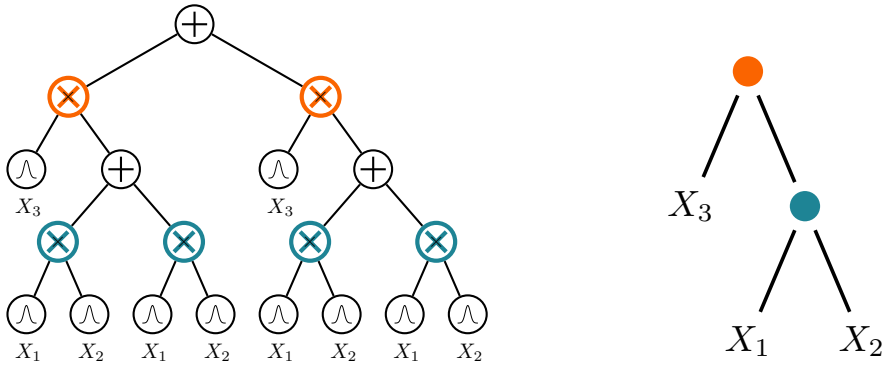
Example 27 *Consider the probability distribution p_{m} defined over the RVs $\mathbf{X}$ as in the traffic jam example. Then the question "How different is the distribution of traffic jam on a rainy day compared to a sunny day?" can be answered by the following* KLD *query:*

$$\mathbb{KL}(p_{\mathsf{m}}(. \mid \mathsf{W}{=}\mathsf{Rain}) \,\|\, p_{\mathsf{m}}(. \mid \mathsf{W}{=}\mathsf{Sun})) = \int_{\mathsf{val}(\mathbf{X})} p_{\mathsf{m}}(\boldsymbol{x} \mid \mathsf{W} = \mathsf{Rain}) \log \frac{p_{\mathsf{m}}(\boldsymbol{x} \mid \mathsf{W} = \mathsf{Rain})}{p_{\mathsf{m}}(\boldsymbol{x} \mid \mathsf{W} = \mathsf{Sun})} d\mathbf{X}.$$

Formally, the KL divergence (also called relative entropy) between probabilistic circuits $\mathcal{C}$ and $\mathcal{C}'$ representing normalized distributions over RVs $\mathbf{X}$ is defined as:

$$\mathbb{KL}(\mathcal{C} \,\|\, \mathcal{C}') = \int_{\mathsf{val}(\mathbf{X})} \mathcal{C}(\boldsymbol{x}) \log \frac{\mathcal{C}(\boldsymbol{x})}{\mathcal{C}'(\boldsymbol{x})} d\mathbf{X}.$$

22. Here we assume that trivially satisfying the structural constraints by representing the distribution in a single distribution unit is not tractable for MMAP.



(a) A structured decomposable circuit (b) A vtree

Figure 1: A structured decomposable PC over $\mathbf{X} = \{X_1, X_2, X_3\}$ and its corresponding vtree.

Let us now introduce the key structural property that enables tractable pairwise query computation.

Definition 45 (Structured decomposability) *A circuit is* **structured decomposable** *if it is decomposable and any pair of its product nodes n and m with the same scope decomposes in the same way: $(\phi(n) = \phi(m)) \Rightarrow (\forall\, i, \phi(\mathsf{in}(n)_i) = \phi(\mathsf{in}(m)_i))$ for some ordering of input units.*

Figure 1a shows an example PC that is structured decomposable. Note that the definition of structured decomposability implies that product nodes with the same scope have the same number of inputs. The decomposition of variables can be described using a ***vtree***: a tree structure whose leaves correspond to (possibly sets of) variables, with each variable appearing exactly once in a vtree leaf node.[23] Then a circuit is structured decomposable if every product node decomposes according to (i.e., normalized for) an internal vtree node. That is, the scope of the i-th input unit is precisely the variables appearing under the i-th sub-vtree node. Consider, for example, Figure 1b which depicts the vtree that the circuit in Figure 1a is normalized for. Each product node is highlighted as the same color as its corresponding vtree node. Furthermore, each input distribution unit of a structured decomposable PC corresponds to a leaf node in the associated vtree; i.e., its scope is exactly the variable(s) appearing in its corresponding vtree node.

Structured decomposability allows us to easily describe whether two probabilistic circuits have compatible scopes at each level; that is, whether they are normalized for the same vtree. This is a key property we exploit in tractably computing pairwise queries, such as the KL divergence query.

Proposition 46 *Suppose $\mathcal{C}$ and $\mathcal{C}'$ are probabilistic circuits over variables $\mathbf{X}$ that are smooth, deterministic, and structured decomposable w.r.t. the same vtree. Then the KL*

23. A vtree was initially defined as a full binary tree whose leaves are in one-to-one correspondence with variables (Pipatsrisawat and Darwiche, 2008). We adopt a more general definition that allows for more than two children nodes and subsets of variables as leaves. The notion of structured decomposability is also generalized in a similar fashion.

divergence $\mathbb{KL}(\mathcal{C} \parallel \mathcal{C}')$ *can be computed tractably, given that it is tractable on the input distributions.*

First, observe that the KL divergence is bounded only if $\mathsf{supp}(\mathcal{C}) \subseteq \mathsf{supp}(\mathcal{C}')$, and is defined as infinite if there is an assignment $\boldsymbol{x}$ with non-zero $\mathcal{C}(\boldsymbol{x})$ but zero $\mathcal{C}'(\boldsymbol{x})$ probability. As it will later become clear, it is useful to consider the *intersectional divergence* (Liang and Van den Broeck, 2017) between circuits:

$$D_I(\mathcal{C} \parallel \mathcal{C}') = \int_{\mathsf{supp}(\mathcal{C}) \cap \mathsf{supp}(\mathcal{C}')} \mathcal{C}(\boldsymbol{x}) \log \frac{\mathcal{C}(\boldsymbol{x})}{\mathcal{C}'(\boldsymbol{x})} d\mathbf{X}.$$

If KL divergence is bounded, then it is equal to the intersectional divergence. We will later show that checking whether the KL divergence is bounded can also be done tractably using circuits.

We now proceed to show how to compute the intersectional divergence tractably, by considering the implications of structural properties at product and sum units. First, we assume $\mathcal{C}$ and $\mathcal{C}'$ are in their canonical forms, as noted in Section 5.5; in particular, they have alternating sums and products. This ensures that the PCs are of the same depth and that each recursive step of the algorithm considers a pair of sub-circuits whose roots are of the same type and are structured decomposable w.r.t. the same vtree.

Now suppose that the roots are structured decomposable product units with k inputs. Structured decomposability w.r.t. a shared vtree dictates that the variables are partitioned the same way in both circuits. We say the i-th children, namely $\mathcal{C}_i$ and $\mathcal{C}'_i$, depend on variables $\mathbf{X}_i$. Then the divergence decomposes as the following:

$$\begin{aligned}
D_I(\mathcal{C} \parallel \mathcal{C}') &= \int_{\mathsf{supp}(\mathcal{C}) \cap \mathsf{supp}(\mathcal{C}')} \left(\prod_i \mathcal{C}_i(\boldsymbol{x}_i) \right) \log \frac{\prod_j \mathcal{C}_j(\boldsymbol{x}_j)}{\prod_j \mathcal{C}'_j(\boldsymbol{x}_j)} d\mathbf{X} \\
&= \int_{\mathsf{supp}(\mathcal{C}_1) \cap \mathsf{supp}(\mathcal{C}'_1)} \cdots \int_{\mathsf{supp}(\mathcal{C}_k) \cap \mathsf{supp}(\mathcal{C}'_k)} \left(\prod_i \mathcal{C}_i(\boldsymbol{x}_i) \right) \sum_j \log \frac{\mathcal{C}_j(\boldsymbol{x}_j)}{\mathcal{C}'_j(\boldsymbol{x}_j)} d\mathbf{X}_k \dots d\mathbf{X}_1 \\
&= \sum_j \left(\int_{\mathsf{supp}(\mathcal{C}_j) \cap \mathsf{supp}(\mathcal{C}'_j)} \mathcal{C}_j(\boldsymbol{x}_j) \log \frac{\mathcal{C}_j(\boldsymbol{x}_j)}{\mathcal{C}'_j(\boldsymbol{x}_j)} d\mathbf{X}_j \right) \prod_{i \neq j} \int_{\mathsf{supp}(\mathcal{C}_i) \cap \mathsf{supp}(\mathcal{C}'_i)} \mathcal{C}_i(\boldsymbol{x}_i) d\mathbf{X}_i \\
&= \sum_j D_I(\mathcal{C}_j \parallel \mathcal{C}'_j) \prod_{i \neq j} \int_{\mathsf{supp}(\mathcal{C}_i) \cap \mathsf{supp}(\mathcal{C}'_i)} \mathcal{C}_i(\boldsymbol{x}_i) d\mathbf{X}_i. \qquad (19)
\end{aligned}$$

Thus, the intersectional divergence of product units can be computed using divergence of their input units and marginals over the intersections of supports. Next, for smooth and

deterministic sum units:

$$
\begin{aligned}
D_I(\mathcal{C} \parallel \mathcal{C}') &= \int_{\mathsf{supp}(\mathcal{C})\cap\mathsf{supp}(\mathcal{C}')} \sum_i \theta_i \mathcal{C}_i(\boldsymbol{x}) \log \frac{\sum_i \theta_i \mathcal{C}_i(\boldsymbol{x})}{\sum_j \theta'_j \mathcal{C}'_j(\boldsymbol{x})} d\mathbf{X} \\
&= \sum_i \theta_i \int_{\mathsf{supp}(\mathcal{C})\cap\mathsf{supp}(\mathcal{C}')} [\![\boldsymbol{x} \in \mathsf{supp}(\mathcal{C}_i)]\!] \mathcal{C}_i(\boldsymbol{x}) \log \frac{\sum_i \theta_i \mathcal{C}_i(\boldsymbol{x})}{\sum_j \theta'_j \mathcal{C}'_j(\boldsymbol{x})} d\mathbf{X} \qquad (20) \\
&= \sum_i \theta_i \int_{\mathsf{supp}(\mathcal{C}_i)\cap(\sqcup_j \mathsf{supp}(\mathcal{C}'_j))} \mathcal{C}_i(\boldsymbol{x}) \log \frac{\theta_i \mathcal{C}_i(\boldsymbol{x})}{\sum_j \theta'_j \mathcal{C}'_j(\boldsymbol{x})} d\mathbf{X} \qquad (21) \\
&= \sum_{i,j} \theta_i \int_{\mathsf{supp}(\mathcal{C}_i)\cap\mathsf{supp}(\mathcal{C}'_j)} \mathcal{C}_i(\boldsymbol{x}) \log \frac{\theta_i \mathcal{C}_i(\boldsymbol{x})}{\theta'_j \mathcal{C}'_j(\boldsymbol{x})} d\mathbf{X} \qquad (22) \\
&= \sum_{i,j} \theta_i \left(\log \frac{\theta_i}{\theta'_j} \int_{\mathsf{supp}(\mathcal{C}_i)\cap\mathsf{supp}(\mathcal{C}'_j)} \mathcal{C}_i(\boldsymbol{x}) d\mathbf{X} + D_I(\mathcal{C}_i \parallel \mathcal{C}'_j) \right). \qquad (23)
\end{aligned}
$$

Equations 20 and 21 hold due to determinism of $\mathcal{C}$; whereas, Equation 22 is derived from $\mathcal{C}'$ being deterministic, thus its support partitioned into supports of its input units: $\mathsf{supp}(\mathcal{C}') = \sqcup_j \mathsf{supp}(\mathcal{C}'_j)$. The intersectional divergence again breaks down into that of input units and marginals over intersections of supports. Hence, if we can tractably compute the marginal probability of a PC over the support of another PC, we can tractably compute the intersectional divergence.

Such marginals can be computed tractably by assumption for input distribution units and by the following equations for product and sum units, respectively.

$$
\int_{\mathsf{supp}(\mathcal{C}')} \prod_i \mathcal{C}_i(\boldsymbol{x}_i) d\mathbf{X} = \prod_i \int_{\mathsf{supp}(\mathcal{C}'_i)} \mathcal{C}_i(\boldsymbol{x}_i) d\mathbf{X}_i \qquad (24)
$$

$$
\int_{\sqcup_j \mathsf{supp}(\mathcal{C}'_j)} \sum_i \theta_i \mathcal{C}_i(\boldsymbol{x}) d\mathbf{X} = \sum_{i,j} \theta_i \int_{\mathsf{supp}(\mathcal{C}'_j)} \mathcal{C}_i(\boldsymbol{x}) d\mathbf{X} \qquad (25)
$$

This is very similar to computing the marginal over a Cartesian product of intervals. The key difference is that the integration is over a more complex domain, defined by the structure of another circuit. Moreover, recall that the intersectional divergence computes the KL divergence only if the support of $\mathcal{C}$ is a subset of the support of $\mathcal{C}'$. This holds when $\int_{\mathsf{supp}(\mathcal{C})\cap\mathsf{supp}(\mathcal{C}')} \mathcal{C}(\boldsymbol{x}) d\mathbf{X} = 1$, which we can compute tractably for smooth, deterministic, and structured decomposable circuits. Therefore, we have all the ingredients for tractable computation of KL divergence. We can use a recursive algorithm with caching, shown in Algorithm 6, to compute the marginal of a circuit w.r.t. another and then the KL divergence between them in polynomial time. This concludes the proof of Proposition 46.

To conclude this section, we note that above result also allows for tractable computation of other information-theoretic measures using probabilistic circuits. For example, we can tractably compute the cross entropy of probabilistic circuits using its relation to KL divergence. The cross entropy between two normalized PCs $\mathcal{C}$ and $\mathcal{C}'$ over RVs $\mathbf{X}$ is defined as

$$
\mathbb{H}(\mathcal{C}, \mathcal{C}') = -\int_{\mathsf{val}(\mathbf{X})} \mathcal{C}(\boldsymbol{x}) \log \mathcal{C}'(\boldsymbol{x}) d\mathbf{X}.
$$

Algorithm 6 KLD$(\mathcal{C}, \mathcal{C}')$ ▷ Cache recursive calls to achieve polynomial complexity

Require: smooth, deterministic, and structured decomposable PCs $\mathcal{C}$ and $\mathcal{C}'$

if InterMAR$(\mathcal{C}, \mathcal{C}') < 1$ **then return** ∞ **else return** InterDiv$(\mathcal{C}, \mathcal{C}')$

function InterMAR(n,m)

if n, m are input units **then return** $\mathcal{C}_n(\mathsf{supp}(\mathcal{C}'_m))$

else if n, m are product units **then return** $\prod_i$ InterMAR(n_i, m_i)

else if n, m are sum units **then return** $\sum_{i,j} \theta_i \cdot$ InterMAR(n_i, m_j)

function InterDiv(n,m)

if n, m are input units **then return** $D_I(\mathcal{C}_n \,\|\, \mathcal{C}'_m)$

else if n, m are product units **then**

return $\sum_i$ InterDiv$(n_i, m_i) \prod_{j \neq i}$ InterMAR(n_j, m_j)

else if n, m are sum units **then**

return $\sum_{i,j} \theta_i$(InterDiv$(n_i, m_j) + \log \frac{\theta_i}{\theta'_j} \cdot$ InterMAR(n_i, m_j))

From this definition, we can easily derive the following expression of cross entropy in terms of entropy and KL-divergence:

$$\mathbb{H}(\mathcal{C}, \mathcal{C}') = \mathbb{H}_{\mathcal{C}}(\mathbf{X}) + \mathbb{KL}(\mathcal{C} \,\|\, \mathcal{C}').$$

Thus, we can tractably compute the cross entropy between two PCs if they are smooth, deterministic, and structured decomposable w.r.t. the same vtree (assuming tractable computation on input distributions), as these properties imply tractable computation of joint entropy (see Corollary 44) as well as KL divergence.[24]

9.2 Expectation

Let us next study another type of pairwise query—expectation—whose tractable computation is enabled by structured decomposability. For example, consider the traffic jam scenario from Section 2.2; expectation queries will allow us to answer questions such as "How likely is my route to work have traffic jam on a weekday?"

Definition 47 (EXP query class) *Let $\mathcal{C}$ be a normalized PC over RVs $\mathbf{X}$, and $\mathcal{C}'$ another PC over the same set of variables. Then the expectation of $\mathcal{C}'$ w.r.t. $\mathcal{C}$ is:*

$$\mathbb{E}_{\mathcal{C}}[\mathcal{C}'] = \int_{\mathsf{val}(\mathbf{X})} \mathcal{C}(\boldsymbol{x}) \cdot \mathcal{C}'(\boldsymbol{x})\, d\mathbf{X}. \tag{26}$$

The EXP query class is quite general and can represent a range of queries depending on the function or distribution defined by $\mathcal{C}'$. In this section, we focus on a notable example that was hinted to earlier: *the probability of logical events* (Choi et al., 2015). While the query classes considered so far dealt with probabilities of events given by assignments or

24. In fact, we can tractably compute the cross entropy $\mathbb{H}(\mathcal{C}, \mathcal{C}')$ if both PCs are smooth and structured decomposable and $\mathcal{C}'$ is deterministic; that is, $\mathcal{C}$ need not be deterministic. We omit the details of the proof here, but how the cross entropy over sum units breaks down can easily be derived in steps similar to Equations 21–23.

Cartesian products of simple intervals, we now turn our attention to probabilities of events with a more intricate structure.

We will represent events as logical formulas involving conjunctions and disjunctions, with atoms consisting of assignments (equality) for discrete RVs and inequality constraints over continuous RVs.[25] For instance the following event over RVs $\mathbf{X} = X_1, X_2$

$$(X_1 = \nu_1) \wedge ((X_2 \geq \nu_2) \vee (X_2 \leq \nu_3))$$

denotes the subset $\{x_1, x_2 \mid x_1 = \nu_1, x_2 \in \mathsf{val}(X_2)\} \cap (\{x_1, x_2 \mid x_1 \in \mathsf{val}(X_1), x_2 \in \mathsf{val}(X_2), x_2 \geq \nu_2\} \cup \{x_1, x_2 \mid x_1 \in \mathsf{val}(X_1), x_2 \in \mathsf{val}(X_2), x_2 \leq \nu_3\})$ over the state space of $\mathbf{X}$ for $\nu_1 \in \mathsf{val}(X_1)$ and $\nu_2, \nu_3 \in \mathsf{val}(X_2)$.

Suppose $\mathcal{C}$ is a PC over RVs $\mathbf{X}$, and a circuit $\mathcal{C}'$ defines a logical formula α over $\mathbf{X}$. Then the expectation $\mathbb{E}_{\mathcal{C}}[\mathcal{C}']$ precisely computes the probability of event given by α w.r.t. the distribution defined by $\mathcal{C}$, denoted $p_{\mathcal{C}}(\alpha)$. We have in fact already seen an example of such query: the class of marginal queries. For instance, consider the following query for the probability of a logical event:

$$p(X_1 = 2 \wedge (X_2 \geq 5) \wedge (X_2 \leq 10)).$$

Above query can be compactly written as

$$p(X_1 = 2, 5 \leq X_2 \leq 10) = p(X_1 = 2, X_2 \in [5, 10]),$$

which corresponds to a MAR query. In other words, a marginal query computes the probability of an event given by a conjunction of literals. Formally, the marginal probability $p_{\mathcal{C}}(\boldsymbol{e}, \boldsymbol{\mathcal{I}})$ for a given evidence $\boldsymbol{e} \in \mathsf{val}(\mathbf{E})$ and intervals $\boldsymbol{\mathcal{I}} = \mathcal{I}_1 \times \cdots \times \mathcal{I}_k$ s.t. $\mathcal{I}_i \in \mathsf{val}(Z_i)$, where each interval $\mathcal{I}_i$ is of the form $a_i \leq Z_i < b_i$, can be interpreted as the probability of a logical formula α where

$$\alpha = \bigwedge_{e \in \boldsymbol{e}} (E = e) \wedge \bigwedge_{i} (a_i \leq Z_i) \wedge (Z_i < b_i).$$

Note that α is simply a linear-sized conjunction, and as shown in Section 4, marginal queries can be tractably computed without constructing a second circuit that represents α. Nevertheless, the expectation query can be useful to compute probabilities of more complex events. For instance, recall the earlier example query for "the probability of a traffic jam on my route to work on a weekday." This query can be written as:

$$p\left(\neg(\mathsf{D} = \mathsf{Sat} \vee \mathsf{D} = \mathsf{Sun}) \wedge \bigvee_{i \in \mathsf{route}} \mathsf{J}_{\mathsf{str}_i}\right).$$

We refer to Section 12.2 on details about representing a logical formula with a PC, exploiting the connection to logical circuits.

25. This logical language to describe events is a fragment of the Satisfiability Modulo Theory (SMT) language (Barrett and Tinelli, 2018) where literals are constrained to be univariate predicates. Computing the probability of events involving multivariate SMT literals, e.g., $(X + Y \leq 5)$ involving additional predicates such as linear arithmetic ones over the reals, poses additional and non-trivial computational challenges, and goes beyond the scope of this work. We refer the reader interested in these advanced classes of probabilistic queries to the literature of weighted model integration (Belle et al., 2015) where tractable representations for them have been recently investigated (Zeng and Van den Broeck, 2019; Zeng et al., 2020).

Algorithm 7 $\textsc{Exp}(n, m)$ ▷ Cache recursive calls to achieve polynomial complexity

Require: smooth and structured decomposable PC nodes n and m
  if n is an input unit **then return** $\mathbb{E}_{\mathcal{C}_n}[\mathcal{C}'_m]$
  else if n,m are product units **then return** $\prod_i \textsc{Exp}(n_i, m_i)$
  else if n,m are sum units **then return** $\sum_{i,j} \theta_i \theta'_j \textsc{Exp}(n_i, m_j)$

Proposition 48 *Suppose $\mathcal{C}$ and $\mathcal{C}'$ are probabilistic circuits over variables $\mathbf{X}$, and are smooth and structured decomposable w.r.t. the same vtree. Then $\mathbb{E}_{\mathcal{C}}[\mathcal{C}']$, the expectation of $\mathcal{C}'$ w.r.t. $\mathcal{C}$, can be computed tractably, if the expectation of input distributions is tractable.*

Analogous to computation of KL divergence, the expectation query on a pair of PC nodes can be broken down into that of their inputs. First, if the roots are product units such that $\mathcal{C}(\boldsymbol{x}) = \prod_{i=1}^{k} \mathcal{C}_i(\boldsymbol{x}_i)$ (similar for $\mathcal{C}'$), we have:

$$\mathbb{E}_{\mathcal{C}}[\mathcal{C}'] = \int \mathcal{C}(\boldsymbol{x}) \cdot \mathcal{C}'(\boldsymbol{x}) \, d\mathbf{X} = \int \left(\prod_i \mathcal{C}_i(\boldsymbol{x}_i) \right) \left(\prod_i \mathcal{C}'_i(\boldsymbol{x}_i) \right) \, d\mathbf{X}$$
$$= \int \prod_i \mathcal{C}_i(\boldsymbol{x}_i) \mathcal{C}'_i(\boldsymbol{x}_i) \, d\mathbf{X} = \prod_i \left(\int \mathcal{C}_i(\boldsymbol{x}_i) \mathcal{C}'_i(\boldsymbol{x}_i) \, d\mathbf{X}_i \right) = \prod_i \mathbb{E}_{\mathcal{C}_i}[\mathcal{C}'_i].$$

In other words, the expectation of structured decomposable product units is simply the product of expectation of their input units.

Next, if the roots are sums, then their children nodes all depend on the same set of variables (namely $\mathbf{X}$) by smoothness. Hence, the expectation can be broken down as follows:

$$\mathbb{E}_{\mathcal{C}}[\mathcal{C}'] = \int \mathcal{C}(\boldsymbol{x}) \cdot \mathcal{C}'(\boldsymbol{x}) \, d\mathbf{X} = \int \left(\sum_i \theta_i \mathcal{C}_i(\boldsymbol{x}) \right) \left(\sum_j \theta'_j \mathcal{C}'_j(\boldsymbol{x}) \right) \, d\mathbf{X}$$
$$= \int \sum_{i,j} \theta_i \theta'_j \mathcal{C}_i(\boldsymbol{x}) \mathcal{C}'_j(\boldsymbol{x}) \, d\mathbf{X} = \sum_{i,j} \theta_i \theta'_j \int \mathcal{C}_i(\boldsymbol{x}) \mathcal{C}'_j(\boldsymbol{x}) \, d\mathbf{X} = \sum_{i,j} \theta_i \theta'_j \mathbb{E}_{\mathcal{C}_i}[\mathcal{C}'_j].$$

Therefore, the expectation of smooth sum nodes can be computed as a weighted sum of the expectations of each pair of children nodes.

We can apply above observations recursively down to the distribution units, as shown in Algorithm 7. Again, the algorithm assumes that both circuits are in their canonical forms, with alternating sum and product units at each layer. Moreover, nodes may have multiple outputs, resulting in multiple recursive calls with the same pair of circuit nodes. These values can be cached to avoid redundant computations. Then, the complexity of the algorithm is loosely upper-bounded by $\mathcal{O}(|\mathcal{C}| \cdot |\mathcal{C}'|)$ assuming tractable computation of expectations for distribution units.

References

Tameem Adel, David Balduzzi, and Ali Ghodsi. Learning the structure of sum-product networks via an svd-based algorithm. In ***UAI***, pages 32–41, 2015.

Alessandro Antonucci, Alessandro Facchini, and Lilith Mattei. Credal sentential decision diagrams. In ***International Symposium on Imprecise Probabilities: Theories and Applications***, pages 14–22, 2019.

Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for# sat and bayesian inference. In ***44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.***, pages 340–351. IEEE, 2003.

Francis R Bach and Michael I Jordan. Thin junction trees. In ***Advances in Neural Information Processing Systems***, pages 569–576, 2002.

Stephen H Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-loss markov random fields and probabilistic soft logic. ***The Journal of Machine Learning Research***, 18(1):3846–3912, 2017.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. ***arXiv preprint arXiv:1409.0473***, 2014.

NTJ Bailey. Probability methods of diagnosis based on small samples. In ***Mathematics and computer science in biology and medicine***, pages 103–107. Her Majesty's Stationery Office, London, 1965.

Velleda Baldoni, Nicole Berline, Jesus De Loera, Matthias Köppe, and Michèle Vergne. How to integrate a polynomial over a simplex. ***Mathematics of Computation***, 80 (273):297–325, 2011.

Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In ***Handbook of Model Checking***, pages 305–343. Springer, 2018.

Jon Barwise. ***Handbook of mathematical logic***. Elsevier, 1982.

Leonard E. Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. ***Ann. Math. Statist.***, 37(6):1554–1563, 12 1966. doi: 10.1214/aoms/1177699147. URL `https://doi.org/10.1214/aoms/1177699147`.

Vaishak Belle and Luc De Raedt. Semiring programming: A framework for search, inference and learning. ***arXiv preprint arXiv:1609.06954***, 2016.

Vaishak Belle, Andrea Passerini, and Guy Van den Broeck. Probabilistic inference in hybrid domains by weighted model integration. In ***Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI)***, pages 2770–2776, 2015.

Armin Biere, Marijn Heule, and Hans van Maaren. ***Handbook of satisfiability***, volume 185. IOS press, 2009.

Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. ***Journal of the ACM (JACM)***, 44(2):201–236, 1997.

James A Boyle, William R Greig, David A Franklin, Ronald MCG Harden, W Watson Buchanan, and Edward M McGirr. Construction of a model for computer-assisted diagnosis: application to the problem of non-toxic goitre. ***QJM: An International Journal of Medicine***, 35(4):565–588, 1966.

Randal E Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. ***ACM Computing Surveys (CSUR)***, 24(3):293–318, 1992.

Cory J Butz, Jhonatan S Oliveira, and Robert Peharz. Sum-product network decompilation. In ***International Conference on Probabilistic Graphical Models***, 2020.

Andrea Cali, Georg Gottlob, Thomas Lukasiewicz, Bruno Marnette, and Andreas Pieris. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In ***2010 25th Annual IEEE Symposium on Logic in Computer Science***, pages 228–242. IEEE, 2010.

Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. ***Journal of statistical software***, 76(1), 2017.

Hei Chan and Adnan Darwiche. On the robustness of most probable explanations. In ***Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence***, pages 63–71, 2006.

Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. ***Artificial Intelligence***, 172(6-7):772–799, 2008.

Arthur Choi and Adnan Darwiche. On relaxing determinism in arithmetic circuits. In ***Proceedings of the Thirty-Fourth International Conference on Machine Learning (ICML)***, 2017.

Arthur Choi, Doga Kisa, and Adnan Darwiche. Compiling probabilistic graphical models using sentential decision diagrams. In ***European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty***, pages 121–132. Springer, 2013.

Arthur Choi, Guy Van den Broeck, and Adnan Darwiche. Tractable learning for structured probability spaces: A case study in learning preference distributions. In ***Twenty-Fourth International Joint Conference on Artificial Intelligence***, 2015.

C Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. ***IEEE transactions on Information Theory***, 14(3):462–467, 1968.

Karine Chubarian and György Turán. Interpretability of bayesian network classifiers: Obdd approximation and polynomial threshold functions. In ***International Symposium on Artificial Intelligence and Mathematics (ISAIM)***, 2020.

Gregory F Cooper. The computational complexity of probabilistic inference using bayesian belief networks. ***Artificial intelligence***, 42(2-3):393–405, 1990.

Adnan Darwiche. Decomposable negation normal form. ***Journal of the ACM (JACM)***, 48(4):608–647, 2001a.

Adnan Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. ***Journal of Applied Non-Classical Logics***, 11(1-2): 11–34, 2001b.

Adnan Darwiche. A differential approach to inference in bayesian networks. ***Journal of the ACM (JACM)***, 50(3):280–305, 2003.

Adnan Darwiche. ***Modeling and reasoning with Bayesian networks***. Cambridge university press, 2009.

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

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. ***Journal of Artificial Intelligence Research***, 17:229–264, 2002.

Sanjoy Dasgupta. Learning polytrees. ***arXiv preprint arXiv:1301.6688***, 2013.

Cassio de Campos. Almost no news on the complexity of map in bayesian networks. In ***Proceedings of the 10th International Conference on Probabilistic Graphical Models***, 2020.

Cassio P de Campos. New complexity results for map in bayesian networks. In ***IJCAI***, volume 11, pages 2100–2106, 2011.

Rina Dechter. Bucket elimination: A unifying framework for reasoning. ***Artificial Intelligence***, 113(1-2):41–85, 1999.

Rina Dechter. Reasoning with probabilistic and deterministic graphical models: exact algorithms. ***Synthesis Lectures on Artificial Intelligence and Machine Learning***, 13(1):1–199, 2019.

Rina Dechter and Robert Mateescu. And/or search spaces for graphical models. ***Artificial intelligence***, 171(2-3):73–106, 2007.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. ***Journal of the Royal Statistical Society: Series B (Methodological)***, 39(1):1–22, 1977.

Aaron Dennis and Dan Ventura. Learning the architecture of sum-product networks using clustering on variables. In ***Advances in Neural Information Processing Systems***, pages 2033–2041, 2012.

Aaron Dennis and Dan Ventura. Greedy structure search for sum-product networks. In ***Twenty-Fourth International Joint Conference on Artificial Intelligence***, 2015.

Aaron W Dennis. Algorithms for learning the structure of monotone and nonmonotone sum-product networks. 2016.

Mattia Desana and Christoph Schnörr. Expectation maximization for sum-product networks as exponential family mixture models. ***arXiv preprint arXiv:1604.07243***, 2016.

Nicola Di Mauro, Antonio Vergari, and Teresa MA Basile. Learning bayesian random cutset forests. In ***International Symposium on Methodologies for Intelligent Systems***, pages 122–132. Springer, 2015.

Nicola Di Mauro, Antonio Vergari, and Floriana Esposito. Learning accurate cutset networks by exploiting decomposability. In ***Congress of the Italian Association for Artificial Intelligence***, pages 221–232. Springer, 2015.

Nicola Di Mauro, Antonio Vergari, Teresa MA Basile, and Floriana Esposito. Fast and accurate density estimation with extremely randomized cutset networks. In ***Joint European conference on machine learning and knowledge discovery in databases***, pages 203–219. Springer, 2017.

Nicola Di Mauro, Floriana Esposito, Fabrizio Giuseppe Ventola, and Antonio Vergari. Sum-product network structure learning by efficient product nodes discovery. ***Intelligenza Artificiale***, 12(2):143–159, 2018.

Pedro Domingos and Daniel Lowd. Markov logic: An interface layer for artificial intelligence. ***Synthesis lectures on artificial intelligence and machine learning***, 3(1):1–155, 2009.

Willliam Feller. ***An introduction to probability theory and its applications***, volume 2. John Wiley & Sons, 2008.

Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted Boolean formulas. ***Theory and Practice of Logic Programming***, 15:358–401, 5 2015. ISSN 1475-3081. doi: 10.1017/S1471068414000076. URL `http://starai.cs.ucla.edu/papers/FierensTPLP15.pdf`.

Abram Friesen and Pedro Domingos. The sum-product theorem: A foundation for learning tractable models. In ***International Conference on Machine Learning***, pages 1909–1918, 2016.

Abram L Friesen and Pedro Domingos. Recursive decomposition for nonconvex optimization. In ***Twenty-Fourth International Joint Conference on Artificial Intelligence***, 2015.

Robert Gens and Domingos Pedro. Learning the structure of sum-product networks. In ***International conference on machine learning***, pages 873–880, 2013.

Partha Ghosh, Mehdi SM Sajjadi, Antonio Vergari, Michael Black, and Bernhard Schölkopf. From variational to deterministic autoencoders. ***arXiv preprint arXiv:1903.12436***, 2019.

Carla P Gomes, Ashish Sabharwal, and Bart Selman. Model counting. 2008.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In ***Advances in neural information processing systems***, pages 2672–2680, 2014.

Thomas L. Griffiths, Nick Chater, Charles Kemp, Amy Perfors, and Joshua B. Tenenbaum. Probabilistic models of cognition: exploring representations and inductive biases. ***Trends in Cognitive Sciences***, 14(8):357 – 364, 2010. ISSN 1364-6613. doi: https://doi.org/10.1016/j.tics.2010.05.004. URL `http://www.sciencedirect.com/science/article/pii/S1364661310001129`.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. ***arXiv preprint arXiv:1609.09106***, 2016.

Steven Holtzen, Guy Van den Broeck, and Todd Millstein. Scaling exact inference for discrete probabilistic programs. ***Proc. ACM Program. Lang. (OOPSLA)***, 2020. doi: https://doi.org/10.1145/342820.

Jinbo Huang, Mark Chavira, and Adnan Darwiche. Solving MAP exactly by searching on compiled arithmetic circuits. In ***Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)***, pages 143–148, 2006.

Manfred Jaeger. Probabilistic decision graphs—combining verification and ai techniques for probabilistic inference. ***International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems***, 12(supp01):19–42, 2004.

Manfred Jaeger, Jens D Nielsen, and Tomi Silander. Learning probabilistic decision graphs. ***International Journal of Approximate Reasoning***, 42(1-2):84–100, 2006.

Priyank Jaini, Abdullah Rashwan, Han Zhao, Yue Liu, Ershad Banijamali, Zhitang Chen, and Pascal Poupart. Online algorithms for sum-product networks with continuous variables. In ***Conference on Probabilistic Graphical Models***, pages 228–239, 2016.

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***, pages 181–192, 2018.

Siddhant M Jayakumar, Wojciech M Czarnecki, Jacob Menick, Jonathan Schwarz, Jack Rae, Simon Osindero, Yee Whye Teh, Tim Harley, and Razvan Pascanu. Multiplicative interactions and where to find them. In ***International Conference on Learning Representations***, 2019.

Brendan Juba. Query-driven pac-learning for reasoning. ***CoRR***, abs/1906.10118, 2019. URL `http://arxiv.org/abs/1906.10118`.

R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. ***Journal of Basic Engineering***, 82(1):35–45, 03 1960. ISSN 0021-9223. doi: 10.1115/1.3662552.

Pasha Khosravi, Antonio Vergari, YooJung Choi, Yitao Liang, and Guy Van den Broeck. Handling missing data in decision trees: A probabilistic approach. ***arXiv preprint arXiv:2006.16341***, 2020.

Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. Algebraic model counting. ***Journal of Applied Logic***, 22:46–62, 2017.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. ***arXiv preprint arXiv:1312.6114***, 2013.

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

Daphne Koller and Nir Friedman. ***Probabilistic graphical models: principles and techniques***. MIT press, 2009.

N Kostantinos. Gaussian mixtures and their applications to signal processing. ***Advanced signal processing handbook: theory and implementation for radar, sonar, and medical imaging real time systems***, pages 3–1, 2000.

Alex Kulesza, Ben Taskar, et al. Determinantal point processes for machine learning. ***Foundations and Trends® in Machine Learning***, 5(2–3):123–286, 2012.

Yitao Liang and Guy Van den Broeck. Towards compact interpretable models: Shrinking of learned probabilistic sentential decision diagrams. In ***IJCAI 2017 Workshop on Explainable Artificial Intelligence (XAI)***, 2017.

Yitao Liang and Guy Van den Broeck. Learning logistic circuits. In ***Proceedings of the AAAI Conference on Artificial Intelligence***, volume 33, pages 4277–4286, 2019.

Yitao Liang, Jessa Bekker, and Guy Van den Broeck. Learning the structure of probabilistic sentential decision diagrams. In ***Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI)***, 2017.

Julissa Giuliana Villanueva Llerena and Denis Deratani Mauá. Robust analysis of map inference in selective sum-product networks. In ***International Symposium on Imprecise Probabilities: Theories and Applications***, pages 430–440, 2019.

David JC MacKay. Bayesian neural networks and density networks. ***Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment***, 354(1):73–80, 1995.

Radu Marinescu and Rina Dechter. And/or branch-and-bound for graphical models. In ***IJCAI***, pages 224–229, 2005.

Radu Marinescu and Rina Dechter. Memory intensive and/or search for combinatorial optimization in graphical models. ***Artificial Intelligence***, 173(16-17):1492–1524, 2009.

James Martens and Venkatesh Medabalimi. On the expressive efficiency of sum product networks. ***arXiv preprint arXiv:1411.7717***, 2014.

Robert Mateescu, Rina Dechter, and Radu Marinescu. And/or multi-valued decision diagrams (aomdds) for graphical models. ***Journal of Artificial Intelligence Research***, 33:465–519, 2008.

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***, pages 205–216, 2017.

Geoffrey J McLachlan and David Peel. ***Finite mixture models***. John Wiley & Sons, 2004.

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

Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L Ong, and Andrey Kolobov. Blog: probabilistic models with unknown objects. In ***Proceedings of the 19th international joint conference on Artificial intelligence***, pages 1352–1359, 2005.

Alejandro Molina, Sriraam Natarajan, and Kristian Kersting. Poisson sum-product networks: A deep architecture for tractable multivariate poisson distributions. In ***Thirty-First AAAI Conference on Artificial Intelligence***, 2017.

Alejandro Molina, Antonio Vergari, Nicola Di Mauro, Sriraam Natarajan, Floriana Esposito, and Kristian Kersting. Mixed sum-product networks: A deep architecture for hybrid domains. In ***Thirty-second AAAI conference on artificial intelligence***, 2018.

Paolo Morettin, Samuel Kolb, Stefano Teso, and Andrea Passerini. Learning weighted model integration distributions. In ***AAAI***, pages 5224–5231, 2020.

Howard Musoff and Paul Zarchan. ***Fundamentals of Kalman filtering: a practical approach***. American Institute of Aeronautics and Astronautics, 2009.

Aniruddh Nath and Pedro M Domingos. Learning tractable statistical relational models. In ***Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence***, 2014.

Mathias Niepert and Pedro M Domingos. Learning and inference in tractable probabilistic knowledge bases. In ***UAI***, pages 632–641, 2015.

Lars Otten and Rina Dechter. Anytime and/or depth-first search for combinatorial optimization. ***Ai Communications***, 25(3):211–227, 2012.

Umut Oztok and Adnan Darwiche. A top-down compiler for sentential decision diagrams. In ***Twenty-Fourth International Joint Conference on Artificial Intelligence***, 2015.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. ***arXiv preprint arXiv:1912.02762***, 2019.

James D Park and Adnan Darwiche. Complexity results and approximation strategies for map explanations. ***Journal of Artificial Intelligence Research***, 21:101–133, 2004.

Robert Peharz, Bernhard C Geiger, and Franz Pernkopf. Greedy part-wise learning of sum-product networks. In ***Joint European Conference on Machine Learning and Knowledge Discovery in Databases***, pages 612–627. Springer, 2013.

Robert Peharz, Robert Gens, and Pedro Domingos. Learning selective sum-product networks. In ***LTPM workshop***, 2014.

Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. On theoretical properties of sum-product networks. In ***Artificial Intelligence and Statistics***, pages 744–752, 2015.

Robert Peharz, Robert Gens, Franz Pernkopf, and Pedro Domingos. On the latent variable interpretation in sum-product networks. ***IEEE transactions on pattern analysis and machine intelligence***, 39(10):2030–2044, 2016.

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

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

Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In ***AAAI***, volume 8, pages 517–522, 2008.

Knot Pipatsrisawat and Adnan Darwiche. A new d-dnnf-based bound computation algorithm for functional e-majsat. In ***Twenty-First International Joint Conference on Artificial Intelligence***, 2009.

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In ***2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)***, pages 689–690. IEEE, 2011.

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

Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In ***Joint European conference on machine learning and knowledge discovery in databases***, pages 630–645. Springer, 2014.

Parikshit Ram and Alexander G Gray. Density estimation trees. In ***Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining***, pages 627–635, 2011.

Abdullah Rashwan, Han Zhao, and Pascal Poupart. Online and distributed bayesian moment matching for parameter learning in sum-product networks. In ***Artificial Intelligence and Statistics***, pages 1469–1477, 2016.

Carl Edward Rasmussen. The infinite gaussian mixture model. In ***Advances in neural information processing systems***, pages 554–560, 2000.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. ***arXiv preprint arXiv:1401.4082***, 2014.

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

Jeffrey S Rosenthal. ***A first look at rigorous probability theory***. World Scientific Publishing Company, 2006.

Dan Roth. On the hardness of approximate reasoning. ***Artificial Intelligence***, 82(1–2): 273–302, 1996.

Tian Sang, Paul Beame, and Henry A Kautz. Performing bayesian inference by weighted model counting. In ***AAAI***, volume 5, pages 475–481, 2005.

Masa-aki Sato. Fast learning of on-line em algorithm. ***Rapport Technique, ATR Human Information Processing Research Laboratories***, 1999.

Xiaoting Shao, Alejandro Molina, Antonio Vergari, Karl Stelzner, Robert Peharz, Thomas Liebig, and Kristian Kersting. Conditional sum-product networks: Imposing structure on deep probabilistic architectures. ***arXiv preprint arXiv:1905.08550***, 2019.

Or Sharir and Amnon Shashua. Sum-product-quotient networks. In ***International Conference on Artificial Intelligence and Statistics***, pages 529–537, 2018.

Or Sharir, Ronen Tamari, Nadav Cohen, and Amnon Shashua. Tensorial mixture models. ***arXiv preprint arXiv:1610.04167***, 2016.

Andy Shih and Stefano Ermon. Probabilistic circuits for variational inference in discrete graphical models. In ***NeurIPS***, 2020.

Andy Shih, Guy Van den Broeck, Paul Beame, and Antoine Amarilli. Smoothing structured decomposable circuits. In ***Advances in Neural Information Processing Systems***, pages 11412–11422, 2019.

Solomon Eyal Shimony. Finding maps for belief networks is np-hard. ***Artificial Intelligence***, 68(2):399–410, 1994.

Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. ***Foundations and Trends® in Theoretical Computer Science***, 5 (3–4):207–388, 2010. ISSN 1551-305X. doi: 10.1561/0400000039. URL `http://dx.doi.org/10.1561/0400000039`.

R. L. Stratonovich. Conditional markov processes. ***Theory of Probability & Its Applications***, 5(2):156–178, 1960. doi: 10.1137/1105015.

Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. Probabilistic databases. ***Synthesis lectures on data management***, 3(2):1–180, 2011.

Ping Liang Tan and Robert Peharz. Hierarchical decompositional mixtures of variational autoencoders. In ***International Conference on Machine Learning***, pages 6115–6124, 2019.

Martin Trapp, Tamas Madl, Robert Peharz, Franz Pernkopf, and Robert Trappl. Safe semi-supervised learning of sum-product networks. ***UAI***, 2017.

Martin Trapp, Robert Peharz, Hong Ge, Franz Pernkopf, and Zoubin Ghahramani. Bayesian learning of sum-product networks. In ***Advances in Neural Information Processing Systems***, pages 6347–6358, 2019.

Martin Trapp, Robert Peharz, Carl E Rasmussen, and Franz Pernkopf. Deep structured mixtures of gaussian processes. In ***The 23rd International Conference on Artificial Intelligence and Statistics***, 2020.

Leslie G Valiant. The complexity of enumeration and reliability problems. ***SIAM Journal on Computing***, 8(3):410–421, 1979a.

Leslie G Valiant. Negation can be exponentially powerful. In ***Proceedings of the eleventh annual ACM symposium on Theory of computing***, pages 189–196, 1979b.

Guy Van den Broeck. Lifted inference and learning in statistical relational models (eerste-orde inferentie en leren in statistische relationele modellen). 2013.

Guy Van den Broeck, Dan Suciu, et al. Query processing on probabilistic data: A survey. ***Foundations and Trends® in Databases***, 7(3-4):197–341, 2017.

Moshe Y Vardi. The complexity of relational query languages. In ***Proceedings of the fourteenth annual ACM symposium on Theory of computing***, pages 137–146, 1982.

Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In ***Joint European Conference on Machine Learning and Knowledge Discovery in Databases***, pages 343–358. Springer, 2015.

Antonio Vergari, Robert Peharz, Nicola Di Mauro, Alejandro Molina, Kristian Kersting, and Floriana Esposito. Sum-product autoencoding: Encoding and decoding representations using sum-product networks. In ***Thirty-Second AAAI Conference on Artificial Intelligence***, 2018.

Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Visualizing and understanding sum-product networks. ***Machine Learning***, 108(4):551–573, 2019a.

Antonio Vergari, Alejandro Molina, Robert Peharz, Zoubin Ghahramani, Kristian Kersting, and Isabel Valera. Automatic bayesian density analysis. In ***Proceedings of the AAAI Conference on Artificial Intelligence***, volume 33, pages 5207–5215, 2019b.

Peter Walley. Statistical reasoning with imprecise probabilities. 1991.

W Austin Webb and Pedro Domingos. Tractable probabilistic knowledge bases with existence uncertainty. In ***Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence***, 2013.

Zhe Zeng and Guy Van den Broeck. Efficient search-based weighted model integration. ***Proceedings of UAI***, 2019.

Zhe Zeng, Paolo Morettin, Fanqi Yan, Antonio Vergari, and Guy Van den Broeck. Scaling up hybrid probabilistic inference with logical and arithmetic constraints via message passing. ***arXiv preprint arXiv:2003.00126***, 2020.

Han Zhao, Mazen Melibari, and Pascal Poupart. On the relationship between sum-product networks and bayesian networks. In ***International Conference on Machine Learning***, pages 116–124, 2015.

Han Zhao, Tameem Adel, Geoff Gordon, and Brandon Amos. Collapsed variational inference for sum-product networks. In ***International Conference on Machine Learning***, pages 1310–1318, 2016a.

Han Zhao, Pascal Poupart, and Geoffrey J Gordon. A unified approach for learning the parameters of sum-product networks. In ***Advances in neural information processing systems***, pages 433–441, 2016b.