A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference

Anonymous Author(s)
Affiliation
Address
email

Abstract

Circuit representations are becoming the lingua franca to express and reason about tractable generative and discriminative models. In this paper, we show how complex inference scenarios for these models that commonly arise in machine learning—from computing the expectations of decision tree ensembles to information-theoretic divergences of sum-product networks—can be represented in terms of tractable modular operations over circuits. Specifically, we characterize the tractability of simple transformations—sums, products, quotients, powers, logarithms, and exponentials—in terms of sufficient structural constraints of the circuits they operate on, and present novel hardness results for the cases in which these properties are not satisfied. Building on these operations, we derive a unified framework for reasoning about tractable models that generalizes several results in the literature and opens up novel tractable inference scenarios.

1 Introduction

Many core computational tasks in machine learning (ML) and AI involve solving *complex integrals*, such as expectations appearing in training losses or in information-theoretic quantities including entropies or divergences. A fundamental question naturally arises: *under which conditions do these quantities admit tractable computation?* Or equivalently, when can we compute them *reliably and efficiently* without resorting to approximations or heuristics? If we are able to find model classes to tractably compute these quantities of interest—henceforth called *queries*—we can then design efficient algorithms with important applications in learning, approximate inference [44], model compression [28], explainable AI [24, 47, 52] and algorithmic bias detection [23, 7, 6].

This "quest" for tracing the tractability of different queries has been carried out several times, often independently for different model classes in ML and AI and crucially, for each query in isolation. For example, the computation of the Kullback-Leibler divergence (KLD) is known to have a closed form for Gaussians, but only recently has an exact algorithm been derived for a more complex tractable model class such as probabilistic sentential decision diagrams (PSDDs) [28]. On the other hand, tractable computation of the entropy, despite being a sub-routine for the KLD, has only been derived for a different tractable model class—selective sum-product networks (SPNs) [36]—by Shih and Ermon [44]. In the current paradigm, if one were to trace the tractability of a query that has not yet been investigated but still involves the same "building blocks" such as logarithms, integrals and products over distributions, for instance Rényi's alpha divergence [40], they would need to derive a novel custom algorithm for each model class and prove its tractability from scratch.

In this paper, we take a different path and introduce a general framework under which the tractability of complex queries can be traced in a *unified and effortless manner over model classes and query classes*. To abstract from the different model formalisms, we carry our analysis over circuit representations [8] as they subsume many tractable generative models—probabilistic circuits such as Chow-Liu trees [9],

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

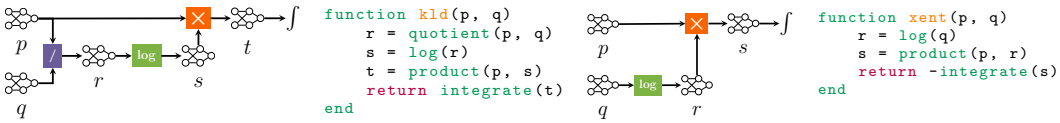


Figure 1: Computational pipelines of the KLD (left) and cross entropy (right) over two distributions p and q encoded as circuits, with the intermediate computations (r, s and t) also represented as circuits. Their corresponding implementations in a few lines of Julia code are shown on their right.

hidden Markov models (HMMs) [39], sum-product networks (SPNs) [38], and other deep mixture models—as well as discriminative ones—including decision trees [25, 10] and deep regressors [23]—thus enabling a unified treatment across model classes.

To generalize our analysis across queries, we propose to represent a single query as a *circuit pipeline*—a computational graph whose intermediate operations transform and combine the input circuits into other circuits. We can first build a set of simple tractable circuit transformations–sums, products, powers, logarithms, and exponentials—and then i) analyze the tractability of a single query by propagating the sufficient conditions for tractability of the intermediate operators in the pipeline; and ii) automatically distill a tractable inference algorithm by composing the operators used. For instance, Fig. 1 shows the pipeline for computing the KLD of p and q, two distributions encoded by circuits. We can identify a general class of models that supports its tractable computation: by tracing the conditions for tractable quotient, logarithm, and product over circuits such that the output circuit (i.e., t) admits tractable integration, we can derive a set of sufficient conditions for the input circuits. Moreover, we can *reuse* the logarithm and product operations in the KLD pipeline to reason about the tractability of cross entropy, in the very same way we can reuse the corresponding code subroutines we provide in Julia to quickly implement algorithms for the two queries in a couple lines of code as shown in Fig. 1. This compositionality greatly speeds up the design of novel tractable algorithms.

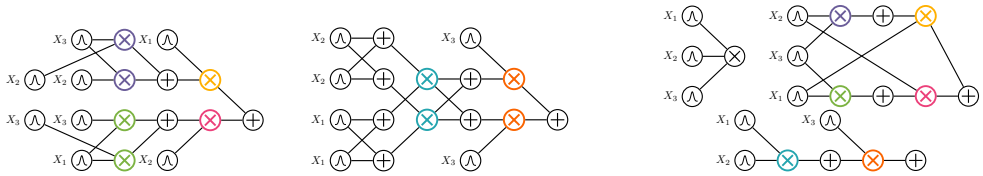
We make the following contributions: (1) a systematic way to compositionally answer many complex queries using simple circuit transformations (Sec. 3), proving sufficient conditions for their tractability and computational hardness when these conditions are unmet (Tab. 1); (2) a unification and generalization of many inference algorithms proposed in the literature so far for specific representations (Sec. 4); (3) novel tractability and hardness results of complex information-theoretic queries including several widely used entropies and divergences (Tab. 2); and (4) a publicly available implementation of these operators in the Juice circuit library [12]. We now start by introducing the circuit language.

2 Circuit Representations

Circuits represent functions as parameterized computational graphs. By imposing certain structural constraints on these graphs, we can guarantee the tractability of certain operations over the encoded functions. Moreover, these constraints help understand how *circuits unify several classical tractable model classes*, such as mixture models, bounded-treewidth probabilistic graphical models (PGMs), decision trees, and compact logical function representations [8, 51]. As such, circuits provide *a language for building and reasoning about tractable representations* and it follows that all our results in the following sections automatically translate to these model classes.

We introduce the basic rules of this language by distinguishing between general circuits and those encoding probability distributions, as some operators in Sec. 3 might be restricted to the latter. Then, we will review the structural constraints we need to characterize different inference scenarios, also known as classes of queries. We denote random variables by uppercase letters (X) and their assignments by lowercase ones (x). Sets of variables and their assignments are denoted by bold uppercase ($\mathbf{X}$) and bold lowercase ($\boldsymbol{x}$) letters, respectively and the set of all their values as $\mathsf{val}(\mathbf{X})$.

Definition 2.1 (Circuit)**.** A circuit p over variables $\mathbf{X}$ is a parameterized computational graph encoding a function $p(\mathbf{X})$ and comprising three kinds of computational units: *input*, *product*, and *sum*. Each inner unit n (i.e., product or sum unit) receives inputs from other units, denoted $\mathsf{in}(n)$. If n is an input unit, it encodes a parameterized function $p_n(\phi(n))$ over variables $\phi(n) \subseteq \mathbf{X}$, also called its *scope*. Instead, if n is a sum unit, it encodes $\sum_{c \in \mathsf{in}(n)} \theta_c p_c(\phi(n))$ where $\theta_c \in \mathbb{R}$ are the sum parameters; while if it is a product unit, it encodes $\prod_{c \in \mathsf{in}(n)} p_c(\phi(n))$. The scope of an inner unit is the union of the scopes of its inputs: $\phi(n) = \bigcup_{c \in \mathsf{in}(n)} \phi(c)$. The output unit of the circuit is



(a) Decomposable (b) Structured-decomposable (c) Omni-compatible

Figure 2: Examples of circuits with different structural properties. The feedforward order is from left to right; input units are labeled by their scopes; and sum parameters are omitted for visual clarity. Product units of the rearranged omni-compatible circuits encoding $p(X_1) \cdot p(X_2) \cdot p(X_3)$ are shown in (c) and color-coded with those of matching scope in (a) and (b).

the last unit (i.e., with out-degree 0) in the graph, encoding $p(\mathbf{X})$. The *support* of p is the set of all complete states for $\mathbf{X}$ for which the output of p is non-zero: $\mathsf{supp}(p) = \{\boldsymbol{x} \in \mathsf{val}(\mathbf{X}) \mid p(\boldsymbol{x}) \neq 0\}$.

Circuits can be understood as compact representations of polynomials with exponentially many terms, whose indeterminates are the functions encoded by the input units. These functions are assumed to be simple enough to allow tractable computations of the operations discussed in this paper. Fig. 2 shows some examples of circuits. A *probabilistic circuit* (PC) [8] represents a (possibly unnormalized) probability distribution by encoding its probability mass, density, or a combination thereof.

Definition 2.2 (Probabilistic circuit)**.** A PC over variables $\mathbf{X}$ is a circuit encoding a function p that is non-negative for all values of $\mathbf{X}$; i.e., $\forall \boldsymbol{x} \in \mathsf{val}(\mathbf{X}) : p(\boldsymbol{x}) \geq 0$.

From here on, we will assume a PC to have positive sum parameters and input units that model valid (unnormalized) distributions, which is a sufficient condition to satisfy the above definition. Moreover, w.l.o.g. we will assume that each layer of a circuit alternates between sum and product units and that every product unit n receives only two inputs c_1, c_2, i.e., $p_n(\mathbf{X}) = p_{c_1}(\mathbf{X}) \cdot p_{c_2}(\mathbf{X})$. These conditions can easily be enforced on a circuit in exchange for only a polynomial increase in its size [49, 50].

Computing (functions of) $p(\mathbf{X})$, or in other words performing *inference*, can be done by evaluating its computational graph. Hence, the computational cost of inference on a circuit is a function of its *size*, defined as the number of edges in it and denoted as $|p|$. For instance, querying the value of p for a complete assignment $\boldsymbol{x}$ equals its *feedforward* evaluation—inputs before outputs—and therefore is linear in $|p|$. Other common inference scenarios such as function integration—which translate to *marginal inference* in the context of probability distributions—can be tackled in linear time with circuits that exhibit certain structural properties, as discussed next.

Structural Properties of Circuits. Structural constraints on the computational graph of a circuit in terms of its scope or support provide sufficient and/or necessary conditions for certain queries to be tractably computed. We now define the structural properties needed for the query classes that this work will focus on, referring to Choi et al. [8] for more details.

Definition 2.3 (Smoothness)**.** A circuit is *smooth* if for every sum unit n, its inputs depend on the same variables: $\forall\, c_1, c_2 \in \mathsf{in}(n), \phi(c_1) = \phi(c_2)$.

Smooth PCs generalize shallow mixture models [30] to deep and hierarchical models. For instance, a Gaussian mixture model (GMM) can be represented as a smooth PC with a single sum unit over as many input units as mixture components, each encoding a (multivariate) Gaussian density.

Definition 2.4 (Decomposability)**.** A circuit is *decomposable* if the inputs of every product unit n depend on disjoint sets of variables: $\mathsf{in}(n) = \{c_1, c_2\}, \phi(c_1) \cap \phi(c_2) = \emptyset$.

Decomposable product units encode local factorizations. That is, a decomposable product unit n over variables $\mathbf{X}$ encodes $p_n(\mathbf{X}) = p_1(\mathbf{X}_1) \cdot p_2(\mathbf{X}_2)$ where $\mathbf{X}_1$ and $\mathbf{X}_2$ form a partition of $\mathbf{X}$. Taken together, decomposability and smoothness are a sufficient and necessary condition for performing tractable integration over arbitrary sets of variables in a single feedforward pass, as they enable larger integrals to be efficiently decomposed into smaller ones [15, 8]. Next Proposition formalizes it.

Proposition 2.1 (Tractable integration, Choi et al. [8])**.** *Let p be a smooth and decomposable circuit over $\mathbf{X}$ with input functions that can be tractably integrated. Then for any variables $\mathbf{Y} \subseteq \mathbf{X}$ and their assignment $\boldsymbol{y}$, the integral $\int_{\boldsymbol{z} \in \mathsf{val}(\mathbf{Z})} p(\boldsymbol{y}, \boldsymbol{z}) d\mathbf{Z}$ can be computed exactly in $\Theta(|p|)$ time, where $\mathbf{Z}$ denotes $\mathbf{X} \setminus \mathbf{Y}$.*

Table 1: ***Tractability and hardness of simple circuit operations***. Tractable conditions on inputs translate to conditions on outputs. E.g., for the quotient p/q, if p and q are compatible (Cmp) and q is deterministic (Det), then the output is decomposable (Dec); also (+) deterministic if p is deterministic; and structured-decomposable (SD) if both p and q are. Hardness results are for representing the output as a smooth (Sm) and decomposable circuit without some input condition.

Operation		Tractability				Hardness	
		Input conditions	Output conditions	Time Complexity			
SUM	$\theta_1 p + \theta_2 q$	(+Cmp)	(+SD)	$\mathcal{O}(\lvert p\rvert+\lvert q\rvert)$	(Prop. B.1)	NP-hard for Det out	[43]
PRODUCT	$p \cdot q$	Cmp (+Det, +SD)	Dec (+Det, +SD)	$\mathcal{O}(\lvert p\rvert\lvert q\rvert)$	(Thm. 3.2)	#P-hard w/o Cmp	(Thm. 3.1)
POWER	$p^n, n \in \mathbb{N}$	SD (+Det)	SD (+Det)	$\mathcal{O}(\lvert p\rvert^n)$	(Thm. 3.3)	#P-hard w/o SD	(Thm. 3.3 and B.2)
	$p^\alpha, \alpha \in \mathbb{R}$	Sm, Dec, Det (+SD)	Sm, Dec, Det (+SD)	$\mathcal{O}(\lvert p\rvert)$	(Thm. 3.5)	#P-hard w/o Det	(Thm. 3.4)
QUOTIENT	p/q	Cmp; q Det (+p Det,+SD)	Dec (+Det,+SD)	$\mathcal{O}(\lvert p\rvert\lvert q\rvert)$	(Thm. B.4)	#P-hard w/o Det	(Thm. B.3)
LOG	$\log(p)$	Sm, Dec, Det	Sm, Dec	$\mathcal{O}(\lvert p\rvert)$	(Thm. 3.6)	#P-hard w/o Det	(Thm. 3.6)
EXP	$\exp(p)$	linear	SD	$\mathcal{O}(\lvert p\rvert)$	(Prop. 3.1)	#P-hard	(Thm. 3.7)

As the complex queries we focus on in this work involve integration as the last step, it is therefore needed that any intermediate operation preserves at least decomposability; smoothness is less of an issue, as it can be enforced in polytime [45]. A key additional constraint over scope decompositions is *compatibility*. Intuitively, two decomposable circuits are compatible if they can be rearranged in polynomial time[1] such that their respective product units, once matched by scope, decompose in the same way. We formalize this with the following inductive definition.

Definition 2.5 (Compatibility)**.** Two circuits p and q over variables $\mathbf{X}$ are *compatible* if (1) they are smooth and decomposable and (2) any pair of product units $n \in p$ and $m \in q$ with the same scope can be rearranged into binary products that are mutually compatible and decompose in the same way: $(\phi(n) = \phi(m)) \implies (\phi(n_i) = \phi(m_i),\ n_i$ and m_i are compatible) for some rearrangement of the inputs of n (resp. m) into n_1, n_2 (resp. m_1, m_2).

We can derive from compatibility the following properties pertaining to a single circuit, which will be useful in our analysis later.

Definition 2.6 (Special types of compatibility)**.** A circuit is *structured-decomposable* if it is compatible with itself. A decomposable circuit p over $\mathbf{X}$ is *omni-compatible* if it is compatible with any smooth and decomposable circuit over $\mathbf{X}$.

Not all decomposable circuits are structured-decomposable (see Figs. 2a and 2b), but some can be rearranged to be compatible with any decomposable circuit. For instance, in Fig. 2c, the fully factorized product unit $p(\mathbf{X}) = p_1(X_1) \cdot p_2(X_2) \cdot p_3(X_3)$ can be rearranged into $p_1(X_1) \cdot (p_2(X_2) \cdot p_3(X_3))$ and $p_2(X_2) \cdot (p_1(X_1) \cdot p_3(X_3))$ to match the yellow and pink products in Fig. 2a. We can easily see that omni-compatible circuits must assume the form of mixtures of fully-factorized models; i.e., $\sum_i \theta_i \prod_j p_{i,j}(X_j)$. For example, an additive ensemble of decision trees over variables $\mathbf{X}$ can be represented as an omni-compatible circuit (see Ex. D.1 in the Appendix). Also note that if two circuits are compatible and neither is omni-compatible, then both must be structured decomposable.

Definition 2.7 (Determinism)**.** A circuit is *deterministic* if the inputs of every sum unit n have disjoint supports: $\forall\, c_1, c_2 \in \mathsf{in}(n), c_1 \neq c_2 \implies \mathsf{supp}(c_1) \cap \mathsf{supp}(c_2) = \emptyset$.

Analogously to decomposability, determinism induces a recursive partitioning, but this time over the support of a circuit. For a deterministic sum unit n, the partitioning of its support can be made explicit by introducing an indicator function per each of its inputs, i.e., $\sum_{c \in \mathsf{in}(n)} \theta_c p_c(\boldsymbol{x}) = \sum_{c \in \mathsf{in}(n)} \theta_c p_c(\boldsymbol{x}) [\![\boldsymbol{x} \in \mathsf{supp}(p_c)]\!]$. Determinism allows for tractable maximization of a circuit [13, 8]. While we do not consider maximization queries in this work, determinism will still play a crucial role in the next sections. Moreover, bounded-treewidth PGMs, such as Chow-Liu trees [9] and thin junction trees [1], can efficiently be represented as a smooth, deterministic, and decomposable PC via *compilation* [13, 11]. Probabilistic sentential decision diagrams (PSDDs) [26] are deterministic and structured-decomposable PCs that can be efficiently learned from data [11].

3 From Simple Circuit Transformations…

This section aims to build an atlas of simple operations over circuits which can then be composed into more complex queries via circuit pipelines—computational graphs whose units are tractable operators

[1] By changing the order in which n-ary product units are turned into a series of binary product units.

over circuits. To compose two operators, we would need that the output circuits of one satisfy the structural properties required for the inputs of the other. As such, for each of these operations we are interested in characterizing (1) its tractability in terms of the structural properties of its input circuits, and (2) its closure w.r.t. these properties, i.e. whether they are preserved in the output circuit, in order to compose many operations together in a pipeline, while (3) providing an efficient algorithmic implementation for it. As we are interested in pipelines for queries involving integration, we would expect the output circuits to at least retain decomposability (see Prop. 2.1). For a pipeline in which all operators can be computed tractably, a simple tractable algorithm can be then distilled for it. Furthermore, our analysis will highlight if one needs to resort to approximations, by tracing the hardness of representing the output of an operator as a decomposable circuit when some property of its inputs is unmet. Tab. 1 summarizes all our results. For space constraints, we discuss the main theorems next and report their complete statements and proofs in the Appendix.

Sum of Circuits. The operation of summing two circuits $p(\mathbf{Z})$ and $q(\mathbf{Y})$ is defined as $s(\mathbf{X}) = \theta_1 \cdot p(\mathbf{Z}) + \theta_2 \cdot q(\mathbf{Y})$ for $\mathbf{X} = \mathbf{Z} \cup \mathbf{Y}$ and two real parameters $\theta_1, \theta_2 \in \mathbb{R}$. This operation, which is at the core of additive ensembles of tractable representations,[2] can be realized by introducing a single sum unit that takes as input p and q. Summation applies to any input circuits, regardless of structural assumptions, and it preserves several properties (see Prop. B.1 in the Appendix). In particular, if p and q are decomposable then s is also decomposable; moreover, if they are compatible then s is structured-decomposable as well as compatible with p and q. However, representing a sum as a deterministic circuit is known to be NP-hard [43], even for compatible and deterministic inputs.

Product of Circuits. The product of two circuits $p(\mathbf{Z})$ and $q(\mathbf{Y})$ can be expressed as $m(\mathbf{X}) = p(\mathbf{Z}) \cdot q(\mathbf{Y})$ for variables $\mathbf{X} = \mathbf{Z} \cup \mathbf{Y}$. If $\mathbf{Z}$ and $\mathbf{Y}$ are disjoint, the product m is already decomposable. Otherwise, Shen et al. [43] proved that representing the product of two decomposable circuits as a decomposable circuit is NP-hard, even if they are deterministic. We prove a novel result that it is #P-hard even for structured-decomposable and deterministic circuits.

Theorem 3.1 (Hardness of product)**.** *If p and q are two structured-decomposable and deterministic circuits, then computing their product as a decomposable circuit is #P-hard.*

Shen et al. [43] also introduced an efficient algorithm for the product of two structured-decomposable and deterministic PCs that are compatible (namely PSDDs). We generalize this result by proving that compatibility alone is sufficient for the tractable product computation of any two circuits.

Theorem 3.2 (Tractable product)**.** *If p and q are two compatible circuits, then computing their product as a decomposable circuit that is compatible with them can be done in $\mathcal{O}(|p|\,|q|)$ time.*

The proof is by construction and leads to Alg. 3 in the Appendix. In the following, we provide a sketch of the algorithm for the case $\mathbf{X} = \mathbf{Z} = \mathbf{Y}$. Intuitively, the idea is to "break down" the construction of the product circuit in a recursive manner by exploiting compatibility. The base case is where p and q are input units with simple parametric forms. Their product can be represented as a single input unit as long as we can find a simple parametric form for it, as is the case for products of exponential families such as (multivariate) Gaussians. Next, we consider the inductive steps where p and q are two sum or product units. If p and q are compatible product units, they decompose $\mathbf{X}$ the same way for some ordering of inputs; i.e., $p(\mathbf{X}) = p_1(\mathbf{X}_1)p_2(\mathbf{X}_2)$ and $q(\mathbf{X}) = q_1(\mathbf{X}_1)q_2(\mathbf{X}_2)$. Then, their product m as a decomposable circuit can be constructed recursively from the products of their inputs: $m(\mathbf{X}) = (p_1q_1)(\mathbf{X}_1) \cdot (p_2q_2)(\mathbf{X}_2)$. On the other hand, if p and q are smooth sum units, written as $p(\mathbf{X}) = \sum_i \theta_i p_i(\mathbf{X})$ and $q(\mathbf{X}) = \sum_j \theta'_j q_j(\mathbf{X})$, we can obtain their product m recursively by distributing product over sum. In other words, $m(\mathbf{X}) = \sum_{i,j} \theta_i \theta'_j (p_i q_j)(\mathbf{X})$. Note that if both input circuits are also deterministic, m is also deterministic since $\mathsf{supp}(p_i q_j) = \mathsf{supp}(p_i) \cap \mathsf{supp}(q_j)$ are disjoint for different i, j. Combining these, the algorithm will recursively compute the product of each pair of units in p and q with matching scopes. Assuming efficient products for input units, the overall complexity is $\mathcal{O}(|p|\,|q|)$, which yields a compact circuit m of size $\mathcal{O}(|p|\,|q|)$. This upper bound is loose and in practice product circuits will be much smaller as our experiments show (Sec. 5), especially if inputs are deterministic as products of units with disjoint supports will be "pruned" away.

Powers of a Circuit. The α-power of a PC $p(\mathbf{X})$ for an $\alpha \in \mathbb{R}$ is denoted as $p^\alpha(\mathbf{X})$ and is an operation needed to compute generalizations of the entropy of a PC and related divergences (Sec. 4). Let us first consider natural powers ($\alpha \in \mathbb{N}$) which can be computed even for general circuits.

[2] If p and q are PCs, then s is a PC encoding a monotonic mixture model if $\theta_1, \theta_2 > 0$ and $\theta_1 + \theta_2 = 1$.

Theorem 3.3 (Natural powers)**.** *If p is a structured-decomposable circuit, then for any $\alpha \in \mathbb{N}$, its power can be represented as a structured-decomposable circuit in $\mathcal{O}(|p|^{\alpha})$ time. Otherwise, if p is only smooth and decomposable, then computing $p^{\alpha}(\mathbf{X})$ as a decomposable circuit is #P-hard.*

The proof for tractability easily follows by directly applying the product operation repeatedly. However, the exponential dependence on α is unavoidable unless P=NP as we demonstrate in Thm. B.2 in the Appendix, thus rendering the operation intractable for large α.

Turning our attention to non-natural $\alpha \in \mathbb{R}$, and restricting our attention to PCs, structured-decomposability is not sufficient to tractably compute α-powers, which we will show in the next theorem for $\alpha = -1$. First, as zero raised to a negative power is undefined, we instead consider the *restricted α-power* of a PC, denoted as $p^{\alpha}(\boldsymbol{x})|_{\mathsf{supp}(p)}$ and equal to $(p(\boldsymbol{x}))^{\alpha}$ if $\boldsymbol{x} \in \mathsf{supp}(p)$ and 0 otherwise. Note that this is equivalent to the α-power if $\alpha \geq 0$. Abusing notation, we will also denote this by $p^{\alpha}(\boldsymbol{x})[\![\boldsymbol{x} \in \mathsf{supp}(p)]\!]$, where $[\![\cdot]\!]$ stands for indicator functions.

Theorem 3.4 (Hardness of reciprocals)**.** *If p is a structured-decomposable circuit over variables $\mathbf{X}$, then computing $p^{-1}(\mathbf{X})\big|_{\mathsf{supp}(p)}$ as a decomposable circuit is #P-hard.*

The key property that enables efficient computation of power circuits is determinism. More interestingly, we do not require structured-decomposability, but only smoothness and decomposability.

Theorem 3.5 (Tractable real powers)**.** *If p is a smooth, decomposable, and deterministic PC, then for any $\alpha \in \mathbb{R}$, its restricted power can be represented as a smooth, decomposable, and deterministic circuit that is compatible with p in $\mathcal{O}(|p|)$ time.*

Again, the proof is done by construction and detailed in Sec. B.4. The key insight is that restricted powers "break down" over a smooth and deterministic sum unit p. That is, $\left(\sum_i \theta_i p_i(\boldsymbol{x})[\![\boldsymbol{x} \in \mathsf{supp}(p_i)]\!]\right)^{\alpha} [\![\boldsymbol{x} \in \mathsf{supp}(p)]\!] = \sum_i \theta_i^{\alpha} p_i^{\alpha}(\boldsymbol{x})[\![\boldsymbol{x} \in \mathsf{supp}(p_i)]\!]$. This follows from the fact that for any $\boldsymbol{x}$, at most one indicator $[\![\boldsymbol{x} \in \mathsf{supp}(p_i)]\!]$ evaluates to 1. As such, when multiplying a deterministic sum unit with itself, each input will only have overlapping support with itself, thus effectively matching product units only with themselves. This is why decomposability suffices. In conclusion, this recursive decomposition of the power of a circuit will result in the power circuit having the same structure as the original circuit, with input functions and sum parameters replaced by their α-powers. The space and time complexity of the algorithm is $\mathcal{O}(|p|)$ for smooth, deterministic, and decomposable PCs, even for natural powers. This will be a key insight to compactly multiply circuits with the same support structure, such as when computing logarithms and entropies (Sec. 4).

We can already see an example of how simple operators can be composed to derive other tractable ones. Consider the quotient of two circuits $p(\mathbf{X})$ and $q(\mathbf{X})$, denoted as $p(\mathbf{X})/q(\mathbf{X})$, and restricted to $\mathsf{supp}(q)$. The quotient, appearing in queries such as KLD or Itakura-Saito divergence (Sec. 4), can be computed by first taking the reciprocal circuit (i.e., the (-1)-power) of q, followed by its product with p. Thus, if q is deterministic and compatible with p, we can take its reciprocal—which will have the same structure as q—and multiply with p to obtain the quotient as a decomposable circuit as we show in Thm. B.4 in the Appendix. There, Thm. B.3 instead proves that the quotient between p and a non-deterministic q is #P-hard even if they are compatible.

Logarithms of a PC. The logarithm of a PC $p(\mathbf{X})$, denoted $\log p(\mathbf{X})$, is fundamental in computing quantities such as entropies and divergences between distributions (Sec. 4). Since the log is undefined for 0 we will again consider the *restricted logarithm*, denoted as $\log p(\boldsymbol{x})|_{\mathsf{supp}(p)}$ and equal to $\log p(\boldsymbol{x})$ if $\boldsymbol{x} \in \mathsf{supp}(p)$ and 0 otherwise.

Theorem 3.6 (Logarithms)**.** *If p is a smooth, deterministic and decomposable PC, then its restricted logarithm can be represented as a decomposable circuit in $\mathcal{O}(|p|)$ time. Otherwise, if p is only smooth and decomposable, or even structured-decomposable, computing its restricted logarithm as a decomposable circuit is #P-hard.*

Note that while the input of the logarithm operator must be a PC, its output can be a general circuit. Moreover, if p is structured decomposable, then so is its logarithm. The tractability proof is again by construction and is detailed in Sec. B.6 in the Appendix. We point out that determinism again allows the restricted log to decompose over the support of the PC, but this time the output circuit is *not* deterministic. Nevertheless, the inputs of the newly introduced sum units can be clearly partitioned into groups sharing the same support of the corresponding product units in p. This implies that whenever we have to multiply a deterministic circuit and its logarithmic circuit—for instance to

Table 2: ***Tractability and hardness of information-theoretic queries over circuits.*** Tractability given some conditions over the input circuits; computational hardness when some of these are unmet.

Query		Tract. Conditions	Hardness	Reference
CROSS ENTROPY	$-\int p(\boldsymbol{x}) \log q(\boldsymbol{x})\, \mathrm{d}\mathbf{X}$	Cmp, q Det	#P-hard w/o Det	Thm. C.1
SHANNON ENTROPY	$-\sum p(\boldsymbol{x}) \log p(\boldsymbol{x})$	Sm, Dec, Det	coNP-hard w/o Det	Thm. C.2
RÉNYI ENTROPY	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{N}$	SD	#P-hard w/o SD	Thm. C.5
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{R}_+$	Sm, Dec, Det	#P-hard w/o Det	Thm. C.6
MUTUAL INFORMATION	$\int p(\boldsymbol{x}, \boldsymbol{y}) \log(p(\boldsymbol{x}, \boldsymbol{y})/(p(\boldsymbol{x})p(\boldsymbol{y})))$	Sm, SD, Det*	coNP-hard w/o SD	Thm. C.3
KULLBACK-LEIBLER DIV.	$\int p(\boldsymbol{x}) \log(p(\boldsymbol{x})/q(\boldsymbol{x}))d\mathbf{X}$	Cmp, Det	#P-hard w/o Det	Thm. C.4
RÉNYI'S ALPHA DIV.	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{N}$	Cmp, q Det	#P-hard w/o Det	Thm. 4.1 and C.7
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{R}$	Cmp, Det	#P-hard w/o Det	Thm. 4.1 and C.7
ITAKURA-SAITO DIV.	$\int [p(\boldsymbol{x})/q(\boldsymbol{x}) - \log(p(\boldsymbol{x})/q(\boldsymbol{x})) - 1] d\,\mathbf{X}$	Cmp, Det	#P-hard w/o Det	Thm. C.8
CAUCHY-SCHWARZ DIV.	$-\log \frac{\int p(\boldsymbol{x}) q(\boldsymbol{x}) d\mathbf{X}}{\sqrt{\int p^2(\boldsymbol{x}) d\mathbf{X} \int q^2(\boldsymbol{x}) d\mathbf{X}}}$	Cmp	#P-hard w/o Cmp	Thm. C.9
SQUARED LOSS	$\int (p(\boldsymbol{x}) - q(\boldsymbol{x}))^2 d\,\mathbf{X}$	Cmp	#P-hard w/o Cmp	Thm. C.10

compute its Shannon entropy (Sec. 4)—we can leverage the sparsifying effect of non-overlapping supports and perform only a linear number of products (cf. product and power operators).

Exponentials of a Circuit. The exponential of a circuit $p(\mathbf{X})$, denoted $\exp(p(\mathbf{X}))$, is the inverse operation of the logarithm and is a fundamental operation when representing distributions such as log-linear models [27]. Similarly to the logarithm, building a decomposable circuit that encodes an exponential of a circuit is hard in general.

Theorem 3.7 (Hardness of exponentials)**.** *If p is a smooth and decomposable circuit, then, computing its exponential as a decomposable circuit is #P-hard, even if p is structured-decomposable.*

Unlike the logarithm however, restricting the operation to deterministic circuits does not help with tractability, since the issue comes from product units: the exponential of a product is neither a sum nor product of exponentials. Nevertheless, it is easy to see that if p encodes a linear sum over its variables, i.e., $p(\mathbf{X}) = \sum_i \theta_i X_i$, we could easily represent its exponential as a circuit comprising a single decomposable product unit, hence tractably.

Proposition 3.1 (Tractable exponential of a linear circuit)**.** *If p is a linear circuit, then its exponential can be represented as an omni-compatible circuit in $\mathcal{O}(|p|)$ time.*

Note that if we were to add an additional deterministic sum unit over many omni-compatible circuits built in this way, we would retrieve a mixture of truncated exponentials [32, 55]. This is the largest class of tractable exponentials we know so far, and enlarging its boundaries is an open problem. Our compositional atlas is now complete: If we were to add an additional circuit operator to it, it would take the form of the already discussed powers, logarithms or exponentials as proven in Sec. B.3.

4 … to Complex Compositional Queries

In this section, we show how our atlas of simple tractable operators can be effectively used to systematically find a tractable model class for *any advanced query that comprises these operators*. We will show its practical utility by quickly coming up with tractability proofs as well as distilling efficient algorithms for several entropy and divergence queries that are largely used in ML. We will then discuss how our discovered tractable circuit classes subsume some previously known results in the literature and prove novel hardness results for when the structural properties of these circuits are unmet. Tab. 2 summarizes our results.

We now showcase how a short tractability proof can be easily distilled, using Rényi's α-divergence[3] [40] as an example. Note that no tractable algorithm was available for it yet. A proof can be built by inferring the sufficient conditions to tractably compute each operator in the pipeline—starting from the last before the integral and proceeding backwards according to Tab. 1.

Theorem 4.1 (Tractable alpha divergence)**.** *The Rényi's α-divergence between two distributions p and q, defined as $(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x})\, d\mathbf{X}$, can be computed exactly in $\mathcal{O}(|p|^{\alpha}\, |q|)$ time for $\alpha \in \mathbb{N}, \alpha > 1$ if p and q are compatible and q is deterministic, or in $\mathcal{O}(|p|\, |q|)$ time for $\alpha \in \mathbb{R}, \alpha \neq 1$ if p and q are both deterministic and compatible.*

[3] Several alternative formulations of α-divergences can be found in the literature such as Amari's [31] and Tsallis's [34] divergences. However, as they share the same core operations—real powers and products of circuits—our results easily extend to them as well.

Proof. A circuit pipeline for Rényi's α-divergence involves first computing $r = p^\alpha$ and $s = q^{1-\alpha}$, then $t = r \cdot s$ and finally integrate it.[4] Therefore we require t to be a smooth and decomposable circuit (Prop. 2.1), which in turn requires r and s to be compatible (Thm. 3.2). To conclude the proof, we need to compute two compatible circuits r and s in polytime, which can be done according to Thm. 3.5 or Thm. 3.3 depending on the value of α. As these theorems state, p^α and $q^{1-\alpha}$ will be compatible with p and q , respectively, with sizes $\mathcal{O}(|p|^\alpha)$ and $\mathcal{O}(|q|)$ for a natural power α or $\mathcal{O}(|p|)$ and $\mathcal{O}(|q|)$ for a real-valued α. As such, t could be computed in $\mathcal{O}(|p|^\alpha\,|q|)$ time for $\alpha \in \mathbb{N}$ or $\mathcal{O}(|p|\,|q|)$ for $\alpha \in \mathbb{R}$ (Thm. 3.2). □

We leave the formal theorems and proofs for the other queries listed in Tab. 2 to Sec. C in the Appendix for space constraints. We remark again that our technique can be used beyond this query list and *can be applied to any complex query that involves a pipeline comprising the operations we discussed in Sec. 3 and culminating in an integration*.

Shannon entropy Smooth, decomposable and deterministic PCs enable the exact computation of Shannon entropy (Thm. C.2) and this tractability result translates to bounded-treewidth PGMs such as Chow-Liu trees and polytrees as they are special cases (Sec. 2). Our framework provides a more succinct tractability proof for the computation of Shannon entropy derived by Shih and Ermon [44], which we complete by demonstrating in Thm. C.2 that it is coNP-hard for non-deterministic PCs.

Rényi entropy For non-deterministic PCs we can employ the tractable computation of Rényi entropy of order α [40], which recovers Shannon Entropy for $\alpha \to 1$. As the logarithm is taken after integration of the power circuit, the tractability and hardness follow directly from those of the power operation (Thm. 3.3 and 3.5).

Cross entropy As hinted by the presence of a logarithm, the cross entropy is #P-hard to compute without determinism, even for compatible PCs (Thm. C.1). Nevertheless, given our atlas the cross entropy can be tractably computed in $\mathcal{O}(|p|\,|q|)$ if p and q are deterministic and compatible.

Mutual information Let a joint distribution $p(\mathbf{X}, \mathbf{Y})$ and its marginals $p(\mathbf{X})$ and $p(\mathbf{Y})$ be represented as PCs. Then the mutual information (MI) over these three PCs can be computed via a pipeline involving product, quotient, and log operators and it is tractable if all circuits are compatible and deterministic. On the other hand, if the marginal distributions cannot be represented as compact deterministic PCs, we prove it to be coNP-hard (Thm. C.3).

Divergences Liang and Van den Broeck [28] proposed an efficient algorithm to compute the KLD tailored for PSDDs.[5] This has been the only tractable divergence available for PCs so far. We greatly extend this panorama with our atlas by introducing Rényi's α-divergences which generalize several other divergences such as the KLD when $\alpha \to 1$, Hellinger's squared divergence when $\alpha = 2^{-1}$, and the $\mathcal{X}^2$-divergence when $\alpha = 2$ [16]. As Thm. 4.1 states, they are tractable for compatible and deterministic PCs, as is the Itakura-Saito divergence [53] (Thm. C.8). For non-deterministic PCs, we characterize the tractability of the squared loss and the Cauchy-Schwarz divergence [19]. The latter has applications in mixture models for approximate inference [46] and has been derived in closed-form only for mixtures of simple parametric forms like Gaussians [22], Weibull and Rayligh distributions [33]. Our results generalize them to deep mixture models [38].

Expectation queries Among other complex queries that can be abstracted into the general form of an expectation of a circuit f w.r.t. a PC p, i.e., $\mathbb{E}_{\boldsymbol{x}\sim p(\mathbf{X})}\left[f(\boldsymbol{x})\right]$, there are the moments of distributions, such as means and variances. They can be efficiently computed for any smooth and decomposable PC, as f is an omni-compatible circuit (Prop. D.1). This result generalizes the moment computation for simple models such as GMMs and HMMs as they can be encoded as smooth and decomposable PCs (Sec. 2). If f is the indicator function of a logical formula, the expectation computes its probability w.r.t. the distribution p. Choi et al. [4] proposed an algorithm tailored to formulas f over binary variables, encoded as SDDs [14] w.r.t. distributions that are PSDDs. We generalize this result to mixed continuous-discrete distributions encoded as structured-decomposable PCs that are not necessarily deterministic and to logical formulas in the language of satisfiability modulo theories [2] over linear arithmetics with univariate literals (Prop. D.2). Lastly, if f encodes constraints over the output distribution of a deep network we retrieve the *semantic loss* [54]. If f encodes a classifier or a

[4] Note that all the operations outside integration are tractable, therefore we can skip them.

[5] Note that our tractability proof in Thm. C.4 is only a few lines long and does not require the lengthy and ad-hoc algebraic derivations of [28].

regressor, then $\mathbb{E}_p[f]$ refers to computing its expected predictions w.r.t. p [24]. Our results generalize the results reported in Van den Broeck et al. [47] such as computing the expectations of decision trees and their ensembles [25] (cf. Prop. D.3) as well as those of *deep regression circuits* [23].[6]

5 Experiments

We prototyped the tractable operators defined in Sec. 3 as subroutines in Julia in the Juice.jl framework [12] to showcase how our modular atlas can practically and quickly help implement tractable algorithms for novel query classes.[7] To this extent, we distilled algorithms for the Shannon, Rényi, and cross entropies and for the KL, α, and Cauchy-Schwarz divergences. We then ran them on deterministic and structured-decomposable circuits learned as in Dang et al. [11] from 20 publicly available real-world benchmark datasets [29, 48].

Tab. 3 and Tab. 4 in Sec. E in the Appendix report all the intermediate circuit sizes in their respective pipelines as well as the time taken to build and execute the pipelines. First, the intermediate circuits created in the pipeline do not blow up in size: as predicted by our theoretical analysis, the size of the logarithm circuit grows by a linear factor ($\sim$3–4x). Moreover, the size of the product circuit $p \cdot q$ is only slightly larger than $\max(|p|, |q|)$ when p and q are deterministic, much smaller than the theoretical bound of $\mathcal{O}(|p|\,|q|)$.

In terms of execution time, our algorithms run in less than a second for most circuits and peak at slightly more than one minute to compute a pipeline of the KLD, whose output circuit has more than 3 million edges on the DNA dataset (Tab. 3). A custom and highly optimized implementation of the KLD for PSDDs by Liang and Van den Broeck [28] runs up to ten times faster on smaller circuits but surprisingly takes $\sim$220 seconds for DNA, highlighting that our compositional atlas is a promising way to distill tractable algorithms. We emphasize that the aim of these experiments, however, is to demonstrate that *our compositional framework can quickly distill new tractable algorithms for queries that were not available before*, such as Rényi entropy and α and Cauchy-Schwarz divergences.

6 Discussion and Conclusions

This work introduced a unified framework to reason about tractable model classes for complex queries composed of simpler operations. This rich atlas of operators can be used to solve many queries common in probabilistic ML and AI as well as novel inference scenarios.

Darwiche and Marquis [15] is the work most closely related to ours: they define operators over *logical circuits*, encoding Boolean functions as computational graphs with AND and OR gates, for which structural properties analogous to those discussed in Sec. 2 can be defined. Our results generalize their work on logical tractable operators such as disjunctions and conjunctions—the analogous to our (deterministic) sums and products—while also extending it to powers, logarithms and exponentials as well as complex queries such as divergences, which have no direct counterpart in the logical domain. Algorithms to tractably multiply two probabilistic models have been proposed for probabilistic decision graphs (PDGs) first [17] and PSDDs later [43]. Despite the different syntax, both model classes can be encoded as structured-decomposable and deterministic circuits in our language [8].[8]

Our property-driven analysis closes many open questions about the tractability and hardness of queries for several model classes that are special cases of circuits. At the same time, our hardness results might limit its general applicability by restricting tractable inference of some query classes to only certain inputs (Sec. 3). However, we see this as an opportunity: now that we can analyze which operators in a pipeline are tractable or not, we could substitute the latter by an approximate inference algorithm distilled from the pipeline. Lastly, we plan to extend our analysis to other queries involving not only integration but also maximization—that is, understanding what are the operators that make MAP inference over probabilistic circuits or optimization over general circuits tractable.

[6] Despite the name, regression circuits do not conform to our definition of circuits in Def. 2.1. Nevertheless, we can translate them to our format in polytime as we illustrate in Alg. 9 in the Appendix.

[7] Code is attached as supplementary material and will be made public upon acceptance.

[8] PDGs and PSDDs entangle compatibility with determinism in terms of special notions of hierarchical scope partitioning, namely pseudo forests [17, 18] and vtrees [37], respectively. As we showed in Thm. 3.2, compatibility is sufficient for tractable multiplication, and as discussed in the previous section many algorithms tailored for PSDDs [4, 43, 23] can be generalized to *non-deterministic* distributions in our framework.

References

[1] Francis R Bach and Michael I Jordan. Thin junction trees. In *NIPS*, volume 14, pages 569–576, 2001.

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

[3] 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.

[4] Arthur Choi, Guy Van den Broeck, and Adnan Darwiche. Tractable learning for structured probability spaces: A case study in learning preference distributions. In *Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2015, pages 2861–2868, 2015.

[5] YooJung Choi, Adnan Darwiche, and Guy Van den Broeck. Optimal feature selection for decision robustness in bayesian networks. In *IJCAI*, pages 1554–1560, 2017.

[6] YooJung Choi, Meihua Dang, and Guy Van den Broeck. Group fairness by probabilistic modeling with latent fair decisions. *arXiv preprint arXiv:2009.09031*, 2020.

[7] YooJung Choi, Golnoosh Farnadi, Behrouz Babaki, and Guy Van den Broeck. Learning fair naive bayes classifiers by discovering and eliminating discrimination patterns. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 10077–10084, 2020.

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

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

[10] Alvaro H. C. Correia, Robert Peharz, and Cassio P. de Campos. Joints in random forests. In *NeurIPS*, 2020.

[11] Meihua Dang, Antonio Vergari, and Guy Van den Broeck. Strudel: Learning structured-decomposable probabilistic circuits. In *PGM*, Proceedings of Machine Learning Research, 2020.

[12] Meihua Dang, Pasha Khosravi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. Juice: A julia package for logic and probabilistic circuits. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (Demo Track)*, Feb 2021.

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

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

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

[16] Alison L Gibbs and Francis Edward Su. On choosing and bounding probability metrics. *International statistical review*, 70(3):419–435, 2002.

[17] 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.

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

[19] Robert Jenssen, Jose C Principe, Deniz Erdogmus, and Torbjørn Eltoft. The cauchy–schwarz divergence and parzen windowing: Connections to graph theory and mercer kernels. *Journalof the Franklin Institute*, 343(6):614–629, 2006.

[20] Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the ising model. *SIAM Journal on computing*, 22(5):1087–1116, 1993.

[21] Wolfgang B Jurkat. On cauchy's functional equation. *Proceedings of the American Mathematical Society*, 16(4):683–686, 1965.

[22] Kittipat Kampa, Erion Hasanbelliu, and Jose C Principe. Closed-form cauchy-schwarz pdf divergence for mixture of gaussians. In *The 2011 International Joint Conference on Neural Networks*, pages 2578–2585. IEEE, 2011.

[23] Pasha Khosravi, YooJung Choi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. On tractable computation of expected predictions. In *Advances in Neural Information Processing Systems*, pages 11169–11180, 2019.

[24] Pasha Khosravi, Yitao Liang, YooJung Choi, and Guy Van den Broeck. What to expect of classifiers? reasoning about logistic regression with missing features. In *IJCAI*, pages 2716–2724, 2019.

[25] Pasha Khosravi, Antonio Vergari, YooJung Choi, Yitao Liang, and Guy Van den Broeck. Handling missing data in decision trees: A probabilistic approach. In *Proceedings of The Art of Learning with Missing Values, Workshop at ICML*, 2020.

[26] Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *Proceedings of the 14th international conference on principles of knowledge representation and reasoning (KR)*, pages 1–10, 2014.

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

[28] 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)*, August 2017.

[29] Daniel Lowd and Jesse Davis. Learning markov network structure with decision trees. In *2010 IEEE International Conference on Data Mining*, pages 334–343. IEEE, 2010.

[30] Geoffrey J McLachlan, Sharon X Lee, and Suren I Rathnayake. Finite mixture models. *Annual review of statistics and its application*, 6:355–378, 2019.

[31] Thomas P. Minka. Expectation propagation for approximate bayesian inference. In *UAI*, pages 362–369. Morgan Kaufmann, 2001.

[32] Serafín Moral, Rafael Rumí, and Antonio Salmerón. Mixtures of truncated exponentials in hybrid bayesian networks. In *ECSQARU*, volume 2143 of *Lecture Notes in Computer Science*, pages 156–167. Springer, 2001.

[33] Frank Nielsen. Closed-form information-theoretic divergences for statistical mixtures. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 1723–1726. IEEE, 2012.

[34] Manfred Opper, Ole Winther, and Michael J Jordan. Expectation consistent approximate inference. *Journal of Machine Learning Research*, 6(12), 2005.

[35] Umut Oztok, Arthur Choi, and Adnan Darwiche. Solving $\mathrm{PP}^{\mathrm{PP}}$-complete problems using knowledge compilation. In *Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 94–103, 2016.

[36] Robert Peharz, Robert Gens, and Pedro Domingos. Learning selective sum-product networks. In *LTPM workshop*, volume 32, 2014.

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

[38] 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.

[39] Lawrence Rabiner and Biinghwang Juang. An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16, 1986.

[40] Alfréd Rényi et al. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*. The Regents of the University of California, 1961.

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

[42] Prasanna K Sahoo and Palaniappan Kannappan. *Introduction to functional equations*. CRC Press, 2011.

[43] Yujia Shen, Arthur Choi, and Adnan Darwiche. Tractable operations for arithmetic circuits of probabilistic models. In *Advances in Neural Information Processing Systems*, pages 3936–3944, 2016.

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

[45] Andy Shih, Guy Van den Broeck, Paul Beame, and Antoine Amarilli. Smoothing structured decomposable circuits. In *NeurIPS*, pages 11412–11422, 2019.

[46] Linh Tran, Maja Pantic, and Marc Peter Deisenroth. Cauchy-schwarz regularized autoencoder. *arXiv preprint arXiv:2101.02149*, 2021.

[47] Guy Van den Broeck, Anton Lykov, Maximilian Schleich, and Dan Suciu. On the tractability of shap explanations. In *Proceedings of the 35th Conference on Artificial Intelligence (AAAI)*, 2021.

[48] Jan Van Haaren and Jesse Davis. Markov network structure learning: A randomized feature generation approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 2012.

[49] 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.

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

[51] Antonio Vergari, Nicola Di Mauro, and Guy Van den Broeck. Tractable probabilistic models: Representations, algorithms, learning, and applications. *Tutorial at UAI*, 2019.

[52] Eric Wang, Pasha Khosravi, and Guy Van den Broeck. Probabilistic sufficient explanations. *arXiv preprint arXiv:2105.10118*, 2021.

[53] Bo Wei and Jerry D Gibson. Comparison of distance measures in discrete spectral modeling. Master's thesis, Citeseer, 2001.

[54] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 5498–5507. PMLR, 2018.

[55] 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. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 10990–11000. PMLR, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Contributions are clearly stated in lines 54-60 and they closely match the referred paragraphs/tables.
   (b) Did you describe the limitations of your work? [Yes] We provide several hardness results to show how the tractable operators and queries we introduced cannot be used when some structural property is unmet. See Tables 1 and 2 for a summary. Moreover, we discuss in Section 6 some possible ways to overcome these limitations. At the same time, we provide an empirical comparison of algorithms distilled with your modular framework noting that some already existing optimized optimizations can be faster on certain circuits (see discussion in Section 5).
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] Our work can be posed within the scope of basic theoretical research about probabilistic inference. As such, it is highly difficult to forsee any negative societal impact made practical by it.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] All theorems in the main text and in the Appendix formally state all assumptions.
   (b) Did you include complete proofs of all theoretical results? [Yes] All proofs for all theorem statements are delegated to the Appendix for space constraints, but they are complete and detailed.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code and instructions (in the form of one Jupyter notebook) are included in the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] All details for reproducibility are specified in Appendix E and in the attached supplementary material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A] Our experiments concern prototyping some probabilistic inference algorithms and therefore we report the circuit sizes and inference times for the operators we introduce. These values are not affected by randomicity.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Details about computing resources can be found in Appendix E.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cited the used circuit learning algorithm [11] as well as the papers introducing the datasets we employed [29, 48].
   (b) Did you mention the license of the assets? [Yes] We specified both the used algorithm and data are publicly available in Section 5.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We included our code in the supplementary material as well as Jupyter notebooks to run it and reproduce our experiments.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We did not collect data ourselves, but using consolidated and public available benchmarks.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We did not collect data ourselves, but using consolidated and public available benchmarks which do not contain personal information.

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We did not perform crowdsourced experiments.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] We did not perform crowdsourced experiments.
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] We did not perform crowdsourced experiments.