Probabilistic Generating Circuits

Honghua Zhang [1] **Brendan Juba** [2] **Guy Van den Broeck** [1]

Abstract

Generating functions, which are widely used in combinatorics and probability theory, encode function values into the coefficients of a polynomial. In this paper, we explore their use as a tractable probabilistic model, and propose probabilistic generating circuits (PGCs) for their efficient representation. PGCs are strictly more expressive efficient than many existing tractable probabilistic models, including determinantal point processes (DPPs), probabilistic circuits (PCs) such as sum-product networks, and tractable graphical models. We contend that PGCs are not just a theoretical framework that unifies vastly different existing models, but also show great potential in modeling realistic data. We exhibit a simple class of PGCs that are not trivially subsumed by simple combinations of PCs and DPPs, and obtain competitive performance on a suite of density estimation benchmarks. We also highlight PGCs' connection to the theory of strongly Rayleigh distributions.

1. Introduction

Probabilistic modeling is an important task in machine learning. Scaling up such models is a key challenge: probabilistic inference quickly becomes intractable as the models become large and sophisticated (Roth, 1996). Central to this effort is the development of *tractable probabilistic models* (TPMs) that guarantee tractable probabilistic inference in the size of the model, yet can efficiently represent a wide range of probability distributions. There has been a proliferation of different classes of TPMs. Examples include bounded-treewidth graphical models (Meila & Jordan, 2000), determinantal point processes (Borodin & Rains, 2005; Kulesza & Taskar, 2012), and various probabilistic circuits (Darwiche, 2009; Kisa et al., 2014; Vergari et al., 2020) such as sum-product networks (Poon & Domingos, 2011).

Ideally, we want our probabilistic models to be as *expressive efficient* (Martens & Medabalimi, 2014) as possible, meaning that they can *efficiently* represent as many classes of distributions as possible, and adapt to a wider spectrum of realistic applications. Often, however, stronger expressive power comes at the expense of tractability: fewer restrictions can make a model more expressive efficient, but it can also make probabilistic inference intractable. We therefore raise the following central research question of this paper: *Does there exist a class of tractable probabilistic models that is strictly more expressive efficient than current TPMs?*

All aforementioned models are usually seen as representing *probability mass functions*: they take assignments to random variables as input and output likelihoods. In contrast, especially in the field of probability theory, it is also common to represent distributions as *probability generating polynomials* (or generating polynomials for short). Generating polynomials are a powerful mathematical tool, but they have not yet found direct use as a probabilistic machine learning representation that permits tractable probabilistic inference.

We make the key observation that the marginal probabilities (including likelihoods) for a probability distribution can be computed by evaluating its generating polynomial in a particular way. Based on this observation, we propose *probabilistic generating circuits* (PGCs), a class of probabilistic models that represent probability generating polynomials compactly as directed acyclic graphs. PGCs provide a partly positive answer to our research question: they are the first known class of TPMs that are strictly more expressive efficient than decomposable probabilistic circuits (PCs), in particular, sum-product networks, and determinantal point processes (DPPs) while supporting tractable marginal inference.

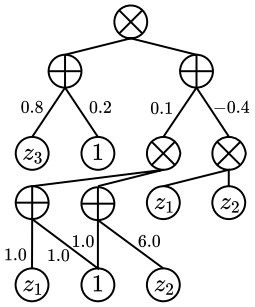
Section 2 formally defines PGCs and establishes their tractability by presenting an efficient algorithm for computing marginals. Section 3 demonstrates the expressive power of PGCs by showing that they subsume PCs and DPPs while remaining strictly more expressive efficient. Section 4 shows that there are PGCs that cannot be represented by PCs with DPPs as leaves. Section 5 evaluates PGCs on standard density estimation benchmarks: even the

[1]Computer Science Department, University of California Los Angeles, USA [2]Computer Science Department, Washington University in St. Louis, Missouri, USA. Correspondence to: Honghua Zhang <hzhang19@cs.ucla.edu>.

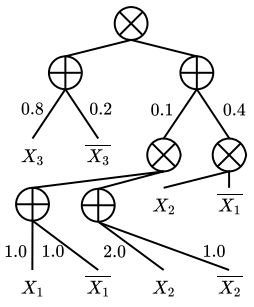
Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

X_1	X_2	X_3	$\Pr_\beta$
0	0	0	0.02
0	0	1	0.08
0	1	0	0.12
0	1	1	0.48
1	0	0	0.02
1	0	1	0.08
1	1	0	0.04
1	1	1	0.16

(a) Table



(b) Probabilistic generating circuit



(c) Probabilistic mass circuit

$$L^\beta = \begin{array}{c} \\ X_1 \\ X_2 \\ X_3 \end{array} \begin{array}{c} \begin{array}{ccc} X_1 & X_2 & X_3 \end{array} \\ \begin{bmatrix} 1 & 2 & 0 \\ 2 & 6 & 0 \\ 0 & 0 & 4 \end{bmatrix} \end{array}$$

$$K^\beta = \begin{array}{c} \\ X_1 \\ X_2 \\ X_3 \end{array} \begin{array}{c} \begin{array}{ccc} X_1 & X_2 & X_3 \end{array} \\ \begin{bmatrix} 0.3 & 0.2 & 0 \\ 0.2 & 0.8 & 0 \\ 0 & 0 & 0.8 \end{bmatrix} \end{array}$$

(d) Kernel L^β and marginal kernel K^β for a DPP

Figure 1. Four different representations for the same probability distribution $\Pr_\beta$.

simplest PGCs outperform other TPM learners on half of the datasets. Then, Section 6 highlights PGCs' connection to strongly Rayleigh distributions. Section 7 summarizes the paper and motivates future research directions.

2. Probabilistic Generating Circuits

In this section we establish probabilistic generating circuits (PGCs) as a class of tractable probabilistic models. We first introduce generating polynomials as a representation for probability distributions and propose PGCs for their compact representations. Then, we show that marginal probabilities for a PGC can be computed efficiently.

2.1. Probability Generating Polynomials

It is a common technique in combinatorics to encode sequences as *generating polynomials*. In particular, probability distributions over binary random variables can be represented by *probability generating polynomials*.

Definition 1. Let $\Pr(\,)$ be a probability distribution over binary random variables $X_1; X_2; \ldots; X_n$, then the *probability generating polynomial* (or generating polynomial for short) for the distribution is defined as

$$g(z_1; \ldots; z_n) = \sum_{S} {}_S z^S \qquad (1)$$

where ${}_S = \Pr(\{X_i = 1 \mid i \in S\}; \{X_i = 0 \mid i \notin S\})$ and $z^S = \prod_{i \in S} z_i$.

As an illustrating example, we consider the probability distribution $\Pr$ specified as a table in Figure 1a. By definition, the generating polynomial for distribution $\Pr$ is given by

$$\begin{aligned} g = {} & 0.16 z_1 z_2 z_3 + 0.04 z_1 z_2 + 0.08 z_1 z_3 + 0.02 z_1 \\ & + 0.48 z_2 z_3 + 0.12 z_2 + 0.08 z_3 + 0.02. \end{aligned} \qquad (2)$$

We see from Equation 2 that the generating polynomial for a distribution simply "enumerates" all possible variable assignments term-by-term, and the coefficient of each term corresponds to the probability of an assignment. The probability for the assignment $X_1 = 0; X_2 = 1; X_3 = 1$, for example, is 0.48, which corresponds to the coefficient of the term $z_2 z_3$. That is, given an assignment, we can evaluate its probability by directly reading off the coefficient for the corresponding term. We can also evaluate marginal probabilities by summing over the coefficients for a *set* of terms. For example, the marginal probability $\Pr(X_2 = 0; X_3 = 0)$ is given by $\Pr(X_1 = 0; X_2 = 0; X_3 = 0) + \Pr(X_1 = 1; X_2 = 0; X_3 = 0)$, which corresponds to the sum of the constant term and the coefficient for the term z_1.

2.2. Compactly Representing Generating Polynomials

Equation 2 also illustrates that the size of a term-by-term representation for a generating polynomial is *exponential* in the number of variables. As the number of variables increases, it quickly becomes impractical to compute probabilities by extracting coefficients from these polynomials. Hence, to turn generating polynomials into tractable models, we need a data structure to represent them more efficiently. We thus introduce a new class of probabilistic circuits called *probabilistic generating circuits* to represent generating polynomials compactly as directed acyclic graphs (DAGs). We first present the formal definition for PGCs.

Definition 2. A *probabilistic generating circuit* (PGC) is a directed acyclic graph consisting of three types of nodes:

1. *Sum nodes* $\bigoplus$ with weighted edges to children;
2. *Product nodes* $\bigotimes$ with unweighted edges to children;
3. *Leaf nodes*, which are z_i or constants.

A PGC has one node of out-degree 0 (edges are directed from children to parents), and we refer to it as the *root* of the PGC. The *size* of a PGC is the number of edges in it.

Each node in a PGC *represents* a polynomial, (i) each leaf in a PGC represents the polynomial z_i or a constant, (ii) each sum node represents the weighted sum over the polynomials represented by its children, and (iii) each product node

represents the unweighted product over the polynomials represented by its children. The polynomial represented by a PGC is the polynomial represented by its root.

We have now fully specified the *syntax* of PGCs, but a PGC with valid syntax does not necessarily have valid *semantics*. Because of the presence of negative parameters, it is not guaranteed that the polynomial represented by a PGC is a probability generating polynomial: it might contain terms that are not multiaffine or have negative coefficients (e.g. $1{:}2z_1z_2^3$). In practice, however, we show in Section 4 that, by certain compositional operations, we can construct PGCs that are guaranteed to have valid semantics for any parameterization.

Continuing our example, we observe that the generating polynomial g in Equation 2 can be re-written as:

$$(0{:}1(z_1 + 1)(6z_2 + 1) \quad 0{:}4z_1z_2)(0{:}8z_3 + 0{:}2) \qquad (3)$$

Based on Equation 3, we can immediately construct a PGC that compactly represents g, as shown in Figure 1b. In this way, generating polynomials for high-dimensional distributions may become feasible to represent by PGCs.

2.3. Tractable Inference with PGCs

We now show that the computation of marginals is tractable for PGCs. As briefly mentioned in Section 2.1, we can compute probabilities by extracting the coefficients of generating polynomials, which is much trickier when they are represented as deeply-nested DAGs; as shown in Figure 1b, it is impossible to directly read off any coefficient. We circumvent this problem by making the key observation that we can "zero-out" the terms we don't want from generating polynomials by evaluating them in a certain way. For example, when evaluating a marginal probability with X_1 set to 0, we zero-out all terms that contain z_1 by setting z_1 to 0. We generalize this idea as follows.

Lemma 1. *Let $g(z_1, \ldots, z_n)$ be a probability generating polynomial for $\Pr(\,)$, then for $A, B \quad f1, \ldots, ng$ with $A \setminus B = ;$, the marginal probability can be computed by:*

$$\Pr(fX_i = 1g_{i2A}; fX_i = 0g_{i2B})$$
$$= \mathrm{coef}_{jAj}\left(g(fz_i = tg_{i2A}; fz_i = 0g_{i2B}; fz_i = 1g_{i\not\in A[B})\right);$$

where t is an indeterminate for polynomials, and $\mathrm{coef}_k(g(t))$ denotes the coefficient for the term t^k in $g(t)$.

Lemma 1 basically says that for a generating polynomial, its marginals can be computed by evaluating the generating polynomial in the *polynomial ring* $\mathsf{R}[t]$. With a bottom-up pass, this result naturally extends to generating polynomials represented as PGCs: we first evaluate the leaf nodes to t, 1 or 0 based on the assignment; then, for the sum nodes, we compute the weighted sum over the polynomials that their children evaluate to; for the product nodes, we compute the product over the polynomials that their children evaluate to. Note that, as we are taking sums and products over univariate polynomials of degree n, the time complexities for the naive algorithms are $O(n)$ and $O(n^2)$, respectively. In light of this, it is not hard to see that computing marginal probabilities is polynomial-time with respect to the size of the PGCs.

Theorem 1. *For PGCs of size m representing distributions on n binary random variables, marginal probabilities (including likelihoods) are computable in time $O(mn \log n \log\log n)$.*

The $O(mn \log n \log\log n)$ complexity in the theorem consists of two parts: the $O(m)$ part is the time complexity of a bottom-up pass for a PGC of size m and the $O(n \log n \log\log n)$ part is contributed by the time complexity of computing the product of two degree-n polynomials with fast Fourier transform (Schönhage & Strassen, 1971; Cantor & Kaltofen, 1991).

3. PGCs Subsume Other Probabilistic Models

To this point, we have introduced PGCs as a probabilistic model and shown that they support tractable marginals. Next, we show that PGCs are *strictly* more expressive efficient than other TPMs by showing that PGCs tractably subsume decomposable probabilistic circuits and determinantal point processes.

3.1. PGCs Subsume Other Probabilistic Circuits

We start by introducing the basics of probabilistic circuits (Vergari et al., 2020; Choi et al., 2020). Just like PGCs, each PC also represents a polynomial with respect to variables X_i and $\overline{X_i}$. The syntax of probabilistic circuits (PCs) is basically the same as PGCs except for the following:

1. the variables in PCs are X_is and $\overline{X_i}$s rather than z_is; they are inherently different in the sense that X_is and $\overline{X_i}$s are the random variables themselves, while z_is are symbolic formal objects;
2. the edge weights of PCs must be non-negative;
3. unlike PGCs, which represent probability generating polynomials, all of the existing PCs represent probability mass functions (as polynomials), so we sometimes refer to them as *probabilistic mass circuits*.

Figure 1c shows an example PC that represents the distribution $\Pr$. For a given assignment $\mathbf{X} = \mathbf{x}$, the PC A *evaluates* to a number $A(\mathbf{x})$, which is obtained by (i) replacing X_i variable leaves by x_i, (ii) replacing $\overline{X_i}$ variable leaves by $1 \quad x_i$, (iii) evaluating product nodes as taking the

product over their children, and (iv) evaluating sum nodes as taking a weighted sum over their children. Finally, a PC A with variable leaves $\mathbf{X} = (X_1, \ldots, X_n)$ represents the probability distribution $\Pr(\mathbf{X} = \mathbf{x}) \;/\; A(\mathbf{x})$.

For an arbitrary PC, most probabilistic inference tasks, including marginals and MAP inference, are computationally hard in the circuit size. In order to guarantee the efficient evaluation of queries it is therefore necessary to impose further constraints on the structure of the circuit. In this paper we consider two well-known structural properties of probabilistic circuits (Darwiche & Marquis, 2002; Choi et al., 2020):

Definition 3. For a PC, we denote the input variables that a node depends on as its *scope*; then,

1. A $\bigotimes$ node is *decomposable* if the scopes of its children are disjoint.
2. A $\bigoplus$ node is *smooth* if the scopes of its children are the same.

A PC is decomposable if all of its $\bigotimes$ nodes are decomposable; a PC is smooth if all of its $\bigoplus$ nodes are smooth.

Let A be a PC over $X_1, \ldots, X_n$. If A is decomposable and smooth, then we can efficiently compute its marginals: for disjoint $A, B \subseteq \{1, \ldots, n\}$ the marginal probability $\Pr(\bigwedge_{i \in A} X_i = 1, \bigwedge_{i \in B} X_i = 0)$ is given by the evaluation of A with the following inputs.

$$\begin{cases} X_i = 1, \overline{X_i} = 0 & \text{if } i \in A \\ X_i = 0, \overline{X_i} = 1 & \text{if } i \in B \\ X_i = 1, \overline{X_i} = 1 & \text{otherwise.} \end{cases}$$

Many TPMs are certain forms of decomposable PCs. Examples include sum-product networks (SPNs) (Poon & Domingos, 2011; Peharz et al., 2019), And-Or graphs (Mateescu et al., 2008), probabilistic sentential decision diagrams (PSDDs) (Kisa et al., 2014), arithmetic circuits (Darwiche, 2009), cutset networks (Rahman & Gogate, 2016) and bounded-treewidth graphical models (Meila & Jordan, 2000) such as Chow-Liu trees (Chow & Liu, 1968) and hidden Markov models (Rabiner & Juang, 1986).

A decomposable PC can always be "smoothed" (i.e. transformed into a smooth and decomposable PC) in polynomial time with respect to its size (Darwiche, 2001; Shih et al., 2019). Hence, when we are trying to show that decomposable PCs can be transformed into equivalent PGCs in polynomial time, we can always assume without loss of generality that decomposable PCs are also smooth. Our first observation is that the probability mass functions represented by smooth and decomposable PCs are very similar to the corresponding generating polynomials:

Proposition 1. *Let A be a smooth and decomposable PC that represents the probability distribution $\Pr$ over random variables $X_1, \ldots, X_n$. Then A represents a probability mass polynomial of the form:*

$$m(z_1, \ldots, z_n) = \sum_{S \subseteq \{1, \ldots, n\}} \alpha_S \prod_{i \in S} X_i \prod_{i \notin S} \overline{X_i} \tag{4}$$

where $\alpha_S = \Pr(\bigwedge_{i \in S} X_i = 1, \bigwedge_{i \notin S} X_i = 0)$.

Note that Equation 4 is closely related to the network polynomials (Darwiche, 2003) defined for Bayesian Networks. By comparing Equation 4 to Equation 1 in the definition of generating circuits, we find that they look very similar, except for the absence of the negative random variables $\overline{X_i}$ in Equation 1, which gives us the following corollary:

Corollary 1. *Let A be a smooth and decomposable PC. By replacing all $\overline{X_i}$ in A by 1 and X_i by z_i, we obtain a PGC that represents the same distribution.*

Corollary 1 establishes that PGCs subsume decomposable PCs and in turn, the TPMs subsumed by decomposable PCs. This raises the question of whether PGCs are *strictly* more expressive efficient and we give a positive answer:

Theorem 2. *PGCs are strictly more expressive efficient than decomposable PCs; that is, there exists a class of probability distributions that can be represented by polynomial-size PGCs but the sizes of any decomposable PCs that represent the distributions are at least exponential in the number of random variables.*

We take determinantal point processes (DPPs) as this separating class of distributions and prove Theorem 2 by showing the following two results: (1) DPPs, in general, cannot be represented by polynomial-size decomposable PCs, and (2) DPPs are tractably subsumed by PGCs. The first result has already been proved in previous works:

Theorem 3 (Zhang et al. (2020); Martens & Medabalimi (2014))**.** *There exists a class of DPPs such that the the size of any decomposable PCs that represent them is exponential in the number of random variables.*

In the next section, we complete this proof by showing that any DPP can be represented by a PGC of polynomial size in the number of random variables.

3.2. PGCs subsume Determinantal Point Processes

In this section, we focus on showing that determinantal point processes (DPPs) can be tractably represented by PGCs. We start by introducing the basics for DPPs.

At a high level, a unique property of DPPs is that they are tractable representations of probability distributions that express *global negative dependence*, which makes them very useful in many applications (Mariet & Sra, 2016), such as document and video summarization (Chao et al., 2015;

Lin & Bilmes, 2012), recommender systems (Zhou et al., 2010), and object retrieval (Affandi et al., 2014).

In machine learning, DPPs are most often represented by means of an *L-ensemble* (Borodin & Rains, 2005):[1]

Definition 4. A probability distribution $\Pr$ over n binary random variables $\mathbf{X} = (X_1, \ldots, X_n)$ is an *L-ensemble* if there exists a (symmetric) positive semidefinite matrix $L \in \mathbb{R}^{n \times n}$ such that for all $\mathbf{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$,

$$\Pr(\mathbf{X} = \mathbf{x}) \propto \det(L_{\mathbf{x}}), \tag{5}$$

where $L_{\mathbf{x}} = [L_{ij}]_{x_i = 1, x_j = 1}$ denotes the submatrix of L indexed by those i, j where $x_i = 1$ and $x_j = 1$. The matrix L is called the *kernel* for the L-ensemble. To ensure that the distribution is properly normalized, it is necessary to divide Equation 5 by $\det(L + I)$, where I is the $n \times n$ identity matrix (Kulesza & Taskar, 2012).

Consider again the example distribution $\Pr$. It is actually a DPP whose kernel is given by the matrix L in Figure 1d. The probability of the assignment $\mathbf{X} = (1, 0, 1)$, for example, is given by

$$\Pr(\mathbf{X} = (1,0,1)) = \frac{\det(L_{\{1,3\}})}{\det(L + I)} = \frac{1}{50}\begin{vmatrix} 1 & 0 \\ 0 & 4 \end{vmatrix} = 0.08.$$

To compute marginal probabilities for L-ensembles, we also need *marginal kernels*, which characterize DPPs in general, as an alternative to L-ensemble kernels.

Definition 5. A probability distribution $\Pr$ is a DPP over n binary random variables $X_1, \ldots, X_n$ if there exists a positive semdidefinite matrix $K \in \mathbb{R}^{n \times n}$ such that for all $A \subseteq \{1, \ldots, n\}$

$$\Pr(\{X_i = 1\}_{i \in A}) = \det(K_A), \tag{6}$$

where $K_A = [K_{ij}]_{i \in A, j \in A}$ denotes the submatrix of K indexed by elements in A.

The marginal kernel K for the L-ensemble that represents the distribution $\Pr$ is shown in Figure 1d, along with its kernel L. One can use a generalized version of Equation 6 to compute the marginal probabilities $\Pr(\{X_i = 1\}_{i \in A}, \{X_j = 0\}_{j \in B})$ efficiently, where $A, B \subseteq \{1, \ldots, n\}$. We refer to Kulesza & Taskar (2012) for further details.

PCs and DPPs support tractable marginals in strikingly different ways, and we wonder whether these two tractable languages can be captured by a unified framework. As mentioned in Section 3.1, it has already been proved that PCs cannot tractably represent DPPs in general. We now show that PGCs also tractably subsume DPPs, providing a positive answer to this open problem.

[1] Although not every DPP is an L-ensemble, Kulesza & Taskar (2012) show that DPPs that assign non-zero probability to the empty set (the all-false assignment) are L-ensembles.

The key to constructing a PGC representation for DPPs is that their generating polynomials can be written as determinants over polynomial rings.

Lemma 2 (Borcea et al. (2009))**.** *The generating polynomial for an L-ensemble with kernel L is given by*

$$g_L = \frac{1}{\det(L + I)} \det(LZ + I). \tag{7}$$

With $Z = \operatorname{diag}(z_1, \ldots, z_n)$, the generating polynomial for a DPP with marginal kernel K is given by

$$g_K = \det(I - K + KZ). \tag{8}$$

Note that the generating polynomials presented in Lemma 2 are just mathematical objects; to use them as tractable models, we need to represent them in the framework of PGCs. So let us examine Equations (7) and (8) in detail. The entries in the matrices $LZ + I$ and $I - K + KZ$ are degree-one univariate polynomials, which can be easily represented as PGCs. Thus, to compactly represent DPPs' generating polynomials as PGCs, we only need to compactly represent the determinant function as a PGC.

There are a variety of polynomial-time division-free algorithms for computing determinants over rings (Bird, 2011; Samuelson, 1942; Berkowitz, 1984; Mahajan & Vinay, 1997) and we take Bird's algorithm (Bird, 2011) as an example. Bird's algorithm is simply an iteration of certain matrix multiplications and requires $O(nM(n))$ additions and multiplications, where $M(n)$ is the number of basic operations needed for a matrix multiplication. We conservatively assume that $M(n)$ is upper-bounded by n^3. Thus when we encode Bird's algorithm as a PGC, the PGC contains at most $O(n^4)$ sum and product nodes, each with a constant number of edges. Together with Lemma 2, it follows that DPPs are tractably subsumed by PGCs.

Theorem 4. *Any DPP over n binary random variables can be represented by a PGC of size $O(n^4)$.*

We conclude this section with the following remarks:

1. DPPs cannot represent any positive dependence; for example, $\Pr(X_i = 1, X_j = 1) > \Pr(X_i = 1)\Pr(X_j = 1)$ can never happen for a DPP. On the other hand, since PGCs are fully general, they are strictly more expressive than DPPs.

2. In practice, when representing DPPs in the language of PGCs, we do not need to explicitly construct the sum nodes and product nodes to form the circuit structures. Recall from Lemma 1 that marginals are tractable as long as we can efficiently *evaluate* the PGCs over polynomial rings. Thus we can simply apply Bird's algorithm, for example, to compute the determinants from Lemma 2.

3. Since nonsymmetric DPPs (Gartrell et al., 2019) are defined in the same way as standard DPPs, except for their kernels L need not be symmetric, they are also tractably subsumed by PGCs.

4. Beyond PCs and DPPs

In the previous section we have demonstrated the expressive power of PGCs by showing that they are strictly more expressive efficient than decomposable PCs and DPPs. It is well-known, however, that PCs can use arbitrary families of tractable distributions at their leaves, including DPPs. In this section, we construct a simple class of PGCs that are more interesting than PCs with DPP leaves.

4.1. Basic Compositional Operations for PGCs

We start by defining the *sum*, *product* and *hierarchical composition* operations for PGCs.

Proposition 2. *Let $A; B$ $\mathbb{N}^+$; denote $f z_i g_{i2A}$ by $\mathbf{z}_A$ and $f X_i g_{i2A}$ by $\mathbf{X}_A$. Let $f(\mathbf{z}_A)$ and $g(\mathbf{z}_B)$ be the generating polynomials for distributions $\Pr_f(\mathbf{X}_A)$ and $\Pr_g(\mathbf{X}_B)$, then,*

***Sum**: let 2 $[0; 1]$, then $f + (1 \quad)g$ is the generating polynomial for the probability distribution $\Pr_{sum}$ where*

$$\Pr_{sum}(\mathbf{X}_A = \mathbf{a}; \mathbf{X}_B = \mathbf{b}) = \Pr_f(\mathbf{X}_A = \mathbf{a}) + (1 \quad)\Pr_g(\mathbf{X}_B = \mathbf{b}):$$

***Product**: if A and B are disjoint (i.e. f and g depend on disjoint sets of variables), then fg is the generating polynomial for the probability distribution $\Pr_{prod}$ where*

$$\Pr_{prod}(\mathbf{X}_A = \mathbf{a}; \mathbf{X}_B = \mathbf{b}) = \Pr_f(\mathbf{X}_A = \mathbf{a})\Pr_g(\mathbf{X}_B = \mathbf{b}):$$

The sum and product operations described above are basically the same as those for PCs: the sum distribution $\Pr_{\text{sum}}$ is just a mixture over two distributions $\Pr_f$ and $\Pr_g$, and the product distribution $\Pr_{\text{prod}}$ is the point-wise product of $\Pr_f$ and $\Pr_g$. The *hierarchical composition* is much more interesting.

Proposition 3 (hierarchical composition)**.** *Let $\Pr_g$ be a probability distribution with generating polynomial $g(z_1; : : : ; z_n)$. Let $A_1; : : : ; A_n$ be disjoint subsets of $\mathbb{N}^+$ and $f_1(\mathbf{z}_{A_1}); : : : ; f_n(\mathbf{z}_{A_n})$ be generating polynomials for $\Pr_i$. We define the hierarchical composition of g and f_is by*

$$g_{\text{comp}} = g\big|_{z_i = f_i};$$

which is the generating polynomial obtained by substituting z_i in g by f_is. In particular, g_{comp} is a well-defined generating polynomial that represents a valid probability distribution.

Unlike the sum and product operations, the hierarchical composition operation for PGCs does not have an immediate analogue for PCs. This operation is a simple yet powerful way to construct classes of PGCs; Figure 2 shows an example, which we illustrate in detail in the next section.

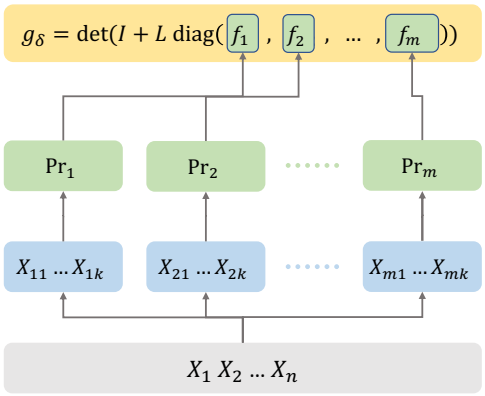


Figure 2. An example of the hierarchical composition for PGCs. We partition n binary random variables into m parts, each with k variables. Then, variables from part i are modeled by the PGC $\Pr_i$ with generating polynomial f_i. Let $g_L = \det(I + L\mathrm{diag}(z_1, \ldots, z_n))$ be the generating polynomial for a DPP with kernel L. Then g_δ is the hierarchical composition of g_L and f_is. We refer to this architecture for g_δ as a determinantal PGC.

4.2. A Separating Example

Now we construct a simple class of PGCs that are not trivially subsumed by PCs with DPPs as leaves. Figure 2 gives an outline of its structure. We construct a model of a probability distribution over n random variables $X_1; : : : ; X_n$. For simplicity we assume $n = mk$ and partition the variables into m parts, each with k variables. Changing notation, we write $f X_{11}; : : : ; X_{1k} g; : : : ; f X_{m1}; : : : ; X_{mk} g$. For $1 \quad i \quad m$, let $\Pr_i$ be a PGC over the random variables $X_{i1}; : : : ; X_{ik}$ with generating polynomial $f_i(z_{i1}; : : : ; z_{ik})$. Let $\Pr_L$ be a DPP with kernel L and generating polynomial $g_L(z_1; : : : ; z_m)$. Then, the generating polynomial $g = g_L\big|_{z_i = f_i}$, namely the hierarchical composition of g_L and f_i, defines a PGC $\Pr$, which we refer to as a *determinantal PGC* (DetPGC).

DPPs are great at modeling negative dependencies but cannot represent positive dependencies between variables: for the DPP $\Pr_L$, $\Pr_L(X_i = 1; X_j = 1) > \Pr_L(X_i = 1)\Pr_L(X_j = 1)$ can never happen. Our construction of DetPGCs aims to equip a DPP model to capture local positive dependencies. To understand how DetPGCs actually behave, we compute the marginal probability $\Pr(X_{ik} = 1; X_{jl} = 1)$ by Lemma 1.

When X_{ik} and X_{jl} belong to the same group (i.e. $i = j$):

	DPP	Strudel	EiNet	MT	SimplePGC
nltcs	9.23	6.07	6.02	**6.01**	6.05*
msnbc	6.48	**6.04**	6.12	6.07	6.06$^{\dagger\circ}$
kdd	2.45	2.14	2.18	**2.13**	2.14*†
plants	31.20	13.22	13.68	**12.95**	13.52†
audio	49.31	42.20	**39.88**	40.08	40.21*
jester	63.88	54.24	**52.56**	53.08	53.54*
netflix	64.18	57.93	**56.54**	56.74	57.42*
accidents	35.61	**29.05**	35.59	29.63	30.46†
retail	11.43	**10.83**	10.92	**10.83**	10.84†
pumsb	51.98	24.39	31.95	**23.71**	29.56†
dna	82.19	87.15	96.09	85.14	**80.82**$^{*\dagger\circ}$
kosarek	13.35	10.70	11.03	**10.62**	10.72†
msweb	11.31	**9.74**	10.03	9.85	9.98†
book	41.22	34.49	34.74	34.63	**34.11**$^{*\dagger\circ}$
movie	83.55	53.72	**51.71**	54.60	53.15$^{*\circ}$
webkb	180.61	**154.83**	157.28	156.86	155.23$^{\dagger\circ}$
reuters	107.44	86.35	87.37	**85.90**	87.65
20ng	174.43	**153.87**	153.94	154.24	154.03°
bbc	278.15	256.53	**248.33**	261.84	254.81$^{*\circ}$
ad	63.20	16.52	26.27	**16.02**	21.65†

(a) Results on the Twenty Datasets benchmark.

	DPP	Strudel	EiNet	MT	SimplePGC
apparel	9.88	9.51	9.24	9.31	**9.10**$^{*\dagger\circ}$
bath	8.55	8.38	8.49	8.53	**8.29**$^{*\dagger\circ}$
bedding	8.65	8.50	8.55	8.59	**8.41**$^{*\dagger\circ}$
carseats	4.74	4.79	4.72	4.76	**4.64**$^{*\dagger\circ}$
diaper	10.61	9.90	9.86	9.93	**9.72**$^{*\dagger\circ}$
feeding	11.86	11.42	11.27	11.30	**11.17**$^{*\dagger\circ}$
furniture	4.38	4.39	4.38	4.43	**4.34**$^{*\dagger\circ}$
gear	9.14	9.15	9.18	9.23	**9.04**$^{*\dagger\circ}$
gifts	3.51	**3.39**	3.42	3.48	3.47°
health	7.40	7.37	7.47	7.49	**7.24**$^{*\dagger\circ}$
media	8.36	**7.62**	7.82	7.93	7.69$^{\dagger\circ}$
moms	3.55	3.52	**3.48**	3.54	3.53°
safety	4.28	4.43	4.39	4.36	**4.28**$^{*\dagger\circ}$
strollers	5.30	5.07	5.07	5.14	**5.00**$^{*\dagger\circ}$
toys	8.05	**7.61**	7.84	7.88	7.62$^{\dagger\circ}$

(b) Results on the Amazon Baby Registries benchmark.

Figure 3. Experiment results on the Twenty Datasets and the Amazon Baby Registries, comparing the performance of DPP, Strudel, EiNet, MT and SimplePGC in terms of average log-likelihood. Bold numbers indicate the best log-likelihood. For SimplePGC, annotations , $^{\circ}$ and mean better log-likelihood compared to Strudel, EiNet and MT, respectively.

$$\Pr(X_{ik}=1, X_{il}=1) = \Pr_L(X_i=1)\Pr_i(X_{ik}=1, X_{il}=1);$$

that is, when two variables belong to the same group, the dependencies between them are dominated by $\Pr_i$, giving space for positive dependencies.

When X_{ik} and X_{jl} belong to different groups (i.e. $i \neq j$):

$$\Pr(X_{ik}=1, X_{jl}=1) \quad \Pr(X_{ik}=1)\Pr(X_{jl}=1);$$

that is, random variables from different groups are still negatively dependent, just like variables in the DPP $\Pr_L$.

We stress that we construct DetPGCs merely to illustrate how the flexibility of PGCs permits us to develop TPMs that capture structure that is beyond the reach of the standard suite of TPMs, including PCs, DPPs, and standard combinations thereof. We are *not* proposing DetPGCs as an "optimal" PGC structure for probabilistic modeling. Nevertheless, as we will see, even the simple DetPGC model may be a better model than PCs or DPPs for some kinds of real-world data.

5. Experiments

This section evaluates PGCs' ability to model real data on density estimation benchmarks. We use a weighted sum over DetPGCs as our model. This simple method achieves state-of-the-art performance on half the benchmarks, illustrating the potential of PGCs in real-world applications.

5.1. Datasets

We evaluate PGCs on two density estimation benchmarks:

1. **Twenty Datasets** (Van Haaren & Davis, 2012), which contains 20 real-world datasets ranging from retail to biology. These datasets have been used to evaluate various tractable probabilistic models (Liang et al., 2017; Dang et al., 2020; Peharz et al., 2020).

2. **Amazon Baby Registries**, which contains 15 datasets,[2] each representing a collection of registries or "baskets" of baby products from a specific category such as "apparel" and "bath". We randomly split each dataset into train (70%), valid (10%) and test (20%) sets. This benchmark has been commonly used to evaluate DPP learners (Gillenwater et al., 2014; Mariet & Sra, 2015; Gartrell et al., 2019).

5.2. Model Structure

The model we use in our experiments is a weighted sum over DetPGCs, the example constructed in Section 4.2, which we refer to as SimplePGCs. Recall from Figure 2 that a DetPGC is the hierarchical composition of a DPP and some "leaf" PGCs $\Pr_1, \ldots, \Pr_m$. For SimplePGC, we make the simplest choice by setting the $\Pr_i$s to be the fully general PGCs. In addition, to partition the input variables into

[2]The original benchmark had 17 datasets. We omit the datasets with fewer than 10 variables: decor and pottytrain.

m groups as shown in Figure 2, we use a simple greedy algorithm that aims at putting pairs of positively dependent variables into the *same* groups. The structure of a SimplePGC is also governed by two hyperparameters: the number of DetPGCs in the weighted sum (denoted by C) and the *maximum number of variables* (i.e. k in Figure 2) allowed in each group (denoted by K). We tune C and K by a grid search over the following ranges: $K \in \{1, 2, 5, 7\}$ and $C \in \{1, 4, 7, 10, 20\}$. Note that our model reduces to a mixture over DPPs when $K = 1$.

We implement SimplePGC in PyTorch and learn the parameters by maximum likelihood estimation (MLE). In particular, we use Adam (Kingma & Ba, 2014) as the optimizing algorithm to minimize the negative log likelihoods given the training sets. Regularization is done by setting the *weight_decay* parameter in Adam. For further details regarding the construction and implementation of SimplePGCs, please see the Appendix.

5.3. Baselines

We compare SimplePGC against four baselines: DPPs, Strudel, Einsum Networks and Mixture of Trees.

DPP: As mentioned, SimplePGC reduces to a mixture over DPPs when the hyperparameter $K = 1$. DPPs are learned via SGD. We expect PGCs to outperform DPPs on most datasets and be at least as good on all datasets.

Strudel: Strudel (Dang et al., 2020) is an algorithm for learning the circuit structure of structured-decomposable PCs. We include them as one of the state-of-the-art tractable density estimators.

Einsum Networks: Einsum Networks (Peharz et al., 2020) (EiNets) are a deep-learning-style implementation design for PCs. Compared to Strudel, EiNets are more related to SimplePGC in the sense that they are both fixed-structure models where only parameters are learned. The hyperparameters are chosen by a grid search as suggested by Peharz et al. (2020).

Mixture of Trees The Mixture of Trees (Meila & Jordan, 2000) (MT) model is a mixture model over Chow-Liu trees (Chow & Liu, 1968). MTs are included as a representative of tractable graphical models with simple yet expressive structures. For learning, we run the *mtlearn* algorithm implemented in the Libra-tk library (Lowd & Rooshenas, 2015); the number of components in MT is chosen by a grid search from 2 to 30 with step size 2, as suggested by Rooshenas & Lowd (2014).

5.4. Results and Analysis

Figure 3 shows the experiment results. We first compare SimplePGC against DPPs. On both benchmarks, SimplePGC performs significantly better than DPPs on almost every dataset except for 4 datasets from the Amazon Baby Registries benchmark, where SimplePGC performs at least as well as DPPs.

Overall, SimplePGC achieves competitive performance when compared against Strudel, EiNet and MT on both benchmarks. On the Twenty Datasets benchmark, SimplePGC obtains better average log-likelihood than at least one of the baselines (Strudel, EiNet and MT) on 19 out of the 20 datasets and, in particular, SimplePGC obtains higher log-likelihood than *all* of them on 2 datasets. Such results are remarkable, given the fact that SimplePGC is just a simple hand-crafted PGC architecture with little fine-tuning, while Strudel, EiNet and MT follow from a long line of research aiming to perform well on exactly the Twenty Datasets benchmark.

The performance of SimplePGC on the Amazon Baby Registries benchmark is even more impressive: SimplePGC beats *all* of baselines on 11 out of 15 datasets and beats at least one of them on all datasets. One possible reason that SimplePGC performs much better than the other baselines on this benchmark is because these datasets exhibit relatively strong negative dependence and SimplePGCs' DPP-like structure allows them to capture negative dependence well.

We also conducted one-sample t-test for the results; for further details please refer to the Appendix.

6. PGCs and Strongly Rayleigh Distributions

At a high level, the study of PCs and graphical models mainly focuses on constructing classes of *models* that guarantee tractable exact inference. A separate line of research in probabilistic machine learning, however, aims at identifying classes of *distributions* that support tractable sampling, where generating polynomials play an essential role. For example, a well-studied class of distributions are the *strongly Rayleigh (SR) distributions* (Borcea et al., 2009; Li et al., 2016), which were first defined in the field of probability theory for studying negative dependence:

Definition 6. A polynomial $f \in \mathbb{R}[z_1, \ldots, z_n]$ is *real stable* if whenever the imaginary part $\mathrm{Im}(z_i) > 0$ for $1 \le i \le n$, $f(z_1, \ldots, z_n) \neq 0$. We say that a distribution over $X_1, \ldots, X_n$ is *strongly Rayleigh* (SR) if its generating polynomial is real stable.

SR distributions contain many important subclasses such as DPPs and the spanning tree/forest distributions, which have various applications. From Section 3.2, we already know that PGCs can compactly represent DPPs. We now show that PGCs can represent spanning tree distributions in polynomial-size.

We first define the spanning tree distributions. Let $G = (V; E)$ be a connected graph with vertex set $V = f1; : : : ; ng$ and edge set E. Associate to each edge $e \; 2 \; E$ a variable z_e and a weight $w_e \; 2 \; \mathsf{R} \; _0$. If $e = fi; jg$, let A_e be the $n \quad n$ matrix where $A_{ii} = A_{jj} = 1$, $A_{ij} = A_{ji} = \quad 1$ and all other entries equal to 0. Then the *weighted Laplacian* of G is given by $L(G) = \sum_{e2E} w_e z_e A_e$;

By the Principal Minors Matrix-Tree Theorem (Chaiken & Kleitman, 1978),

$$f_G = \det(L(G)_{nfig}) = \sum_{T \text{ a spanning tree of G}} w^{\text{edges}(T)} z^{\text{edges}(T)}$$

is the (un-normalized) generating polynomial for the spanning tree distribution, and we denote it by Pr_G. Here $L(G)_{nfig}$ denotes the principal minor of $L(G)$ by removing its ith row and column.

As shown in the equation above, Pr_G is supported on the spanning trees of G, and the probability of each spanning tree is proportional to the product of its edge weights. Pr_G is a strongly Rayleigh distribution (Borcea et al., 2009), and, to the best of our knowledge, it is not a DPP unless the edge weights are the same. By the same argument as in Section 3.2, we claim that Pr_G can be tractably represented by PGCs.

Thus, we see that there is another natural class of SR distributions – spanning tree distributions – that can be represented by PGCs. More generally, generating polynomials play a key role in the study of a number of other classes of distributions, including the Ising model (Jerrum & Sinclair, 1993), exponentiated strongly Rayleigh (ESR) distributions (Mariet et al., 2018) and strongly log-concave (SLC) distributions (Robinson et al., 2019). Specifically, most of these distributions are naturally characterized by their generating polynomials rather than probability mass functions. This poses a major barrier to linking them to other probabilistic models. Thus by showing that PGCs can tractably represent certain subclasses of SR distributions, we present PGCs as a prospective avenue for bridging this gap.

Although we conjecture that not all SR distributions can be represented by polynomial-size PGCs, we believe that the subclasses of the above distributions that have concise parameterizations should be representable by PGCs. Establishing this for various families is a direction for future work.

7. Conclusion and Perspectives

We conclude by summarizing our contributions and highlighting future research directions. In this paper, we study the use of probability generating polynomials as a data structure for representing probability distributions. We showed that their representation as circuits are a TPM, and are strictly more expressive efficient than existing families of TPMs. Indeed, even a simple example family of distributions that can be represented by PGCs but not PCs or DPPs obtains state-of-the-art performance as a probabilistic model on some datasets.

To facilitate the general use of PGCs for probabilistic modeling, a facinating direction for future work is to build efficient structure learning or 'architecture search' algorithms for PGCs. Theoretically, the main mathematical advantage of generating polynomials was the variety of properties they reveal about a distribution. This raises the question of whether there are other kinds of useful queries we can support efficiently with PGCs, and where truly lies the boundary of tractable probabilistic inference.

Acknowledgements

We thank the reviewers for their detailed and thoughtful feedback and efforts towards improving this paper. We thank Steven Holtzen, Antonio Vergari and Zhe Zeng for helpful feedback and discussion. This work is partially supported by NSF grants #IIS-1943641, #IIS-1633857, #CCF-1837129, #CCF-1718380, #IIS-1908287, and #IIS-1939677, DARPA XAI grant #N66001-17-2-4032, Sloan and UCLA Samueli Fellowships, and gifts from Intel and Facebook Research.

References

Affandi, R. H., Fox, E., Adams, R., and Taskar, B. Learning the parameters of determinantal point process kernels. In *ICML*, pp. 1224–1232, 2014.

Berkowitz, S. J. On computing the determinant in small parallel time using a small number of processors. *Information processing letters*, 18(3):147–150, 1984.

Bird, R. S. A simple division-free algorithm for computing determinants. *Information Processing Letters*, 111(21-22):1072–1074, 2011.

Borcea, J., Brändén, P., and Liggett, T. Negative dependence and the geometry of polynomials. *Journal of the American Mathematical Society*, 22(2):521–567, 2009.

Borodin, A. and Rains, E. M. Eynard–mehta theorem, schur process, and their pfaffian analogs. *Journal of statistical physics*, 121(3-4):291–317, 2005.

Cantor, D. G. and Kaltofen, E. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28(7):693–701, 1991.

Chaiken, S. and Kleitman, D. J. Matrix tree theorems. *Journal of combinatorial theory, Series A*, 24(3):377–381, 1978.

Chao, W.-L., Gong, B., Grauman, K., and Sha, F. Large-margin determinantal point processes. In *UAI*, pp. 191–200, 2015.

Choi, Y., Vergari, A., and Van den Broeck, G. Probabilistic circuits: A unifying framework for tractable probabilistic modeling. 2020.

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

Dang, M., Vergari, A., and Van den Broeck, G. Strudel: Learning structured-decomposable probabilistic circuits. *arXiv e-prints*, pp. arXiv–2007, 2020.

Darwiche, A. 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, 2001.

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

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

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

Gartrell, M., Brunel, V.-E., Dohmatob, E., and Krichene, S. Learning nonsymmetric determinantal point processes. In *Advances in Neural Information Processing Systems 32*, 2019.

Gillenwater, J., Kulesza, A., Fox, E., and Taskar, B. Expectation-maximization for learning determinantal point processes. In *Advances in Neural Information Processing Systems 27*, pp. 3149–3157, 2014.

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

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kisa, D., Van den Broeck, G., Choi, A., and Darwiche, A. Probabilistic sentential decision diagrams. In *KR*, 2014.

Kulesza, A. and Taskar, B. Determinantal point processes for machine learning. *Foundations and Trends R in Machine Learning*, 5(2–3):123–286, 2012.

Li, C., Jegelka, S., and Sra, S. Fast mixing Markov chains for strongly rayleigh measures, DPPs, and constrained sampling. In *Advances In Neural Information Processing Systems 29*, pp. 4188–4196, 2016.

Liang, Y., Bekker, J., and Van den Broeck, G. Learning the structure of probabilistic sentential decision diagrams. In *UAI*, 2017.

Lin, H. and Bilmes, J. A. Learning mixtures of submodular shells with application to document summarization. In *UAI*, pp. 479–490, 2012.

Lowd, D. and Rooshenas, A. The libra toolkit for probabilistic models. *Journal of Machine Learning Research*, 16:2459–2463, 2015.

Mahajan, M. and Vinay, V. A combinatorial algorithm for the determinant. In *SODA*, pp. 730–738, 1997.

Mariet, Z. and Sra, S. Fixed-point algorithms for learning determinantal point processes. In *ICML*, pp. 2389–2397. PMLR, 2015.

Mariet, Z., Sra, S., and Jegelka, S. Exponentiated strongly rayleigh distributions. *Advances in neural information processing systems 31*, 2018.

Mariet, Z. E. and Sra, S. Kronecker determinantal point processes. In *Advances in Neural Information Processing Systems 29*, pp. 2694–2702, 2016.

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

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

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

Peharz, R., Vergari, A., Stelzner, K., Molina, A., Shao, X., Trapp, M., Kersting, K., and Ghahramani, Z. Random sum-product networks: A simple but effective approach to probabilistic deep learning. In *UAI*, 2019.

Peharz, R., Lang, S., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Van den Broeck, G., Kersting, K., and Ghahramani, Z. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *ICML*, pp. 7563–7574. PMLR, 2020.

Poon, H. and Domingos, P. Sum-product networks: A new deep architecture. In *ICCV Workshops*, pp. 689–690, 2011.

Rabiner, L. and Juang, B. An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16, 1986.

Rahman, T. and Gogate, V. Learning ensembles of cutset networks. In *AAAI*, 2016.

Robinson, J., Sra, S., and Jegelka, S. Flexible modeling of diversity with strongly log-concave distributions. In *Advances in Neural Information Processing Systems 32*, pp. 15225–15235, 2019.

Rooshenas, A. and Lowd, D. Learning sum-product networks with direct and indirect variable interactions. In *ICML*, pp. 710–718. PMLR, 2014.

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

Samuelson, P. A. A method of determining explicitly the coefficients of the characteristic equation. *The Annals of Mathematical Statistics*, 13(4):424–429, 1942.

Schönhage, A. and Strassen, V. Schnelle multiplikation grosser zahlen. *Computing*, 7(3):281–292, 1971.

Shih, A., Van den Broeck, G., Beame, P., and Amarilli, A. Smoothing structured decomposable circuits. In *Advances in Neural Information Processing Systems 32*, pp. 11412–11422, 2019.

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

Vergari, A., Choi, Y., Peharz, R., and Van den Broeck, G. Probabilistic circuits: Representations, inference, learning and applications. Tutorial at the The 34th AAAI Conference on Artificial Intelligence, 2020.

Zhang, H., Holtzen, S., and Van den Broeck, G. On the relationship between probabilistic circuits and determinantal point processes. In *UAI*, volume 124 of *PMLR*, pp. 1188–1197, 2020.

Zhou, T., Kuscsik, Z., Liu, J.-G., Medo, M., Wakeling, J. R., and Zhang, Y.-C. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010.

A. Proofs

Proof for Lemma 1. We write the generating polynomial for $\Pr$ as $g(z_1,\dots,z_n) = \sum_{S \subseteq [n]} \alpha_S z^S$; then,

$$\Pr(\{X_i = 1 \mid i \in A\}, \{X_i = 0 \mid i \in B\}) = \sum_{A \subseteq S, B \setminus S = \emptyset} \alpha_S$$

Besides, we also have

$$\mathrm{coef}_{j A j}(g|_{\mathrm{asgn}}) = \sum_{S} \alpha_S \mathrm{coef}_{j A j}(z^S|_{\mathrm{asgn}});$$

given the assignment $\mathrm{asgn} := \{z_i = t \mid i \in A\} \cup \{z_i = 0 \mid i \in B\} \cup \{z_i = 1 \mid i \in \overline{A \cup B}\}$; that is, to prove the Proposition, we only need to show that $\mathrm{coef}_{j A j}(z^S|_{\mathrm{asgn}}) = 1$ for $S \subseteq [n]$ where $A \subseteq S$ and $B \setminus S = \emptyset$; and $\mathrm{coef}_{j A j}(z^S|_{\mathrm{asgn}}) = 0$ otherwise.

Case 1. Assume $A \subseteq S$ and $B \setminus S = \emptyset$; then $z^S|_{\mathrm{asgn}} = t^{|A|}$; hence $\mathrm{coef}_{j A j}(z^S|_{\mathrm{asgn}}) = 1$.

Case 2. Assume $A \not\subseteq S$ or $B \setminus S \neq \emptyset$; if $B \setminus S \neq \emptyset$, then $z^S|_{\mathrm{asgn}} = 0$; done. Now we assume $A \not\subseteq S$. In this case, $z^S|_{\mathrm{asgn}} = t^{|S \setminus A|}$. It follows from $S \setminus A \subsetneq A$ that $|S \setminus A| < |A|$, which implies that $\mathrm{coef}_{j A j}(z^S|_{\mathrm{asgn}}) = 0$. □

Proof for Proposition 1. Let A be a decomposable and smooth PC. Without loss of generality we assume A is normalized. For each node u in A, we define $I_u = \{i : X_i \text{ or } \overline{X_i} \in \mathrm{scope}(u)\}$ and denote the polynomial that u represents by m_u. We first prove the following intermediate result by a bottom-up induction on A:

$$m_u = \sum_{S \subseteq I_u} \alpha_S \prod_{i \in S} X_i \prod_{i \in \overline{S}} \overline{X_i};$$

where the α_S are some non-negative numbers depending on the node u.

Case 1. If u is a leaf node X_i or $\overline{X_i}$, then m_u is X_i or $\overline{X_i}$; done.

Case 2. If u is a sum node with children $\{v_i\}_{1 \le i \le k}$ and weights $\{w_i\}_{1 \le i \le k}$. $m_u = \sum_{1 \le i \le k} w_i m_{v_i}$. By smoothness, $I_{v_i} = I_u$ for all i. Then, by the induction hypothesis:

$$\begin{aligned} m_u &= \sum_{1 \le i \le k} w_i \sum_{S \subseteq I_{v_i}} \alpha_{i S} \prod_{j \in S} X_j \prod_{j \in \overline{S}} \overline{X_j} \\ &= \sum_{1 \le i \le k} w_i \sum_{S \subseteq I_u} \alpha_{i S} \prod_{j \in S} X_j \prod_{j \in \overline{S}} \overline{X_j} \\ &= \sum_{S \subseteq I_u} \Big(\sum_{1 \le i \le k} w_i \, \alpha_{i S} \Big) \prod_{j \in S} X_j \prod_{j \in \overline{S}} \overline{X_j} \end{aligned}$$

Case 3. If u is a product node with children $\{v_i\}_{1 \le i \le k}$. Then, by decomposability, $I_{v_1}, \dots, I_{v_k}$ are pairwise disjoint; in particular, for each $S \subseteq I_u$, S can be uniquely decomposed into $S_1 \subseteq I_{v_1}, \dots, S_k \subseteq I_{v_k}$. Thus,

$$\begin{aligned} m_u &= \prod_{1 \le i \le k} m_{v_i} \\ &= \prod_{1 \le i \le k} \sum_{S_i \subseteq I_{v_i}} \alpha_{i S} \prod_{j \in S_i} X_j \prod_{j \in \overline{S_i}} \overline{X_j} \\ &= \sum_{S_1 \subseteq I_{v_1}, \dots, S_k \subseteq I_{v_k}} \left(\prod_{1 \le i \le k} \alpha_{i S_i} \right) \prod_{1 \le i \le k} \left(\prod_{j \in S_i} X_j \prod_{j \in \overline{S_i}} \overline{X_j} \right) \\ &= \sum_{S \subseteq I_u} \left(\prod_{1 \le i \le k} \alpha_{i S_i} \right) \prod_{j \in S} X_j \prod_{j \in \overline{S}} \overline{X_j}; \quad \text{with } I_u = I_{v_1} \sqcup \cdots \sqcup I_{v_n} \text{ a disjoint union.} \end{aligned}$$

Hence the mass polynomial represented by A is given by:

$$m(X_1,\dots,X_n,\overline{X_1},\dots,\overline{X_n}) = \sum_{S \subseteq \{1,\dots,n\}} \alpha_S \prod_{i\in S} X_i \prod_{i\notin S} \overline{X_i}$$

By plugging in $\{X_i = 1\}_{i\in S}, \{X_i = 0\}_{i\notin S}$, it immediately follows that $\alpha_S = \Pr(\{X_i = 1\}_{i\in S}, \{X_i = 0\}_{i\notin S})$. □

Proof for Proposition 2. Let $A, B \subseteq \mathbb{N}^+$; let $f = \sum_{S\subseteq A} \alpha_S z^S$ and $g = \sum_{S\subseteq B} \beta_S z^S$ be the normalized probability generating polynomials for distributions $\Pr_f(\mathbf{X}_A)$ and $\Pr_g(\mathbf{X}_B)$, respectively.

Case 1 (**Sum**). First, we view f and g as polynomials over $\{z_i\}_{i\in A\cup B}$ by setting $\alpha_S = 0\ \forall S \not\subseteq A$, and $\beta_S = 0\ \forall S \not\subseteq B$, which is equivalent to, from the perspective of probability distributions, viewing $\Pr_f$ and $\Pr_g$ as distributions over $\mathbf{X}_{A\cup B}$ such that

$$\Pr_f(\mathbf{X}_A = \mathbf{a}, \mathbf{X}_B = \mathbf{b}) = \begin{cases} \Pr_f(\mathbf{X}_A = \mathbf{a}), & \text{if } b_i = 0 \text{ for } i \in (A\cup B)\cap A \\ 0, & \text{otherwise} \end{cases}$$

and

$$\Pr_g(\mathbf{X}_A = \mathbf{a}, \mathbf{X}_B = \mathbf{b}) = \begin{cases} \Pr_g(\mathbf{X}_B = \mathbf{b}), & \text{if } a_i = 0 \text{ for } i \in (A\cup B)\cap B \\ 0, & \text{otherwise} \end{cases}$$

Then,

$$\begin{aligned} f + (1-\lambda)g &= \sum_{S\subseteq A} \alpha_S z^S + (1-\lambda)\sum_{S\subseteq B} \beta_S z^S \\ &= \sum_{S\subseteq A\cup B} (\lambda\alpha_S + (1-\lambda)\beta_S) z^S, \end{aligned}$$

where $\lambda\alpha_S + (1-\lambda)\beta_S$ are clearly non-negative, and $\sum_{S\subseteq A\cup B} \lambda\alpha_S + (1-\lambda)\beta_S = \lambda\sum_{S\subseteq A\cup B}\alpha_S + (1-\lambda)\sum_{S\subseteq A\cup B}\beta_S = \lambda + (1-\lambda) = 1$. That is, $f + (1-\lambda)g$ is a valid probability generating polynomial for a distribution, $\Pr_{sum}$.

For assignments $\mathbf{X}_A = \mathbf{a}$ and $\mathbf{X}_B = \mathbf{b}$ with no conflict (A and B are not necessarily disjoint), let $S = \{i \in A : a_i = 1\} \cup \{i \in B : b_i = 1\}$. By definition, $\Pr_{sum}(\mathbf{X}_A = \mathbf{a}, \mathbf{X}_B = \mathbf{b})$ is given by the coefficient of the term z^S, which is

$$\begin{aligned} &\lambda\alpha_S + (1-\lambda)\beta_S \\ =\ &\Pr_f(\mathbf{X}_A = \mathbf{a}, \mathbf{X}_B = \mathbf{b}) + (1-\lambda)\Pr_g(\mathbf{X}_A = \mathbf{a}, \mathbf{X}_B = \mathbf{b}) \\ =\ &\Pr_f(\mathbf{X}_A = \mathbf{a}) + (1-\lambda)\Pr_g(\mathbf{X}_B = \mathbf{b}) \text{ for short.} \end{aligned}$$

Case 2 (**Product**). We assume $A \setminus B = \emptyset$. Then,

$$\begin{aligned} fg &= \left(\sum_{S\subseteq A} \alpha_S z^S\right)\left(\sum_{T\subseteq B} \beta_T z^T\right) \\ &= \sum_{S\subseteq A, T\subseteq B} \alpha_S\beta_T z^S z^T \end{aligned}$$

As A and B are disjoint, $z^S z^T$ are multiaffine. On top of that, $\alpha_S\beta_T \geq 0$ and $\sum_{S\subseteq A, T\subseteq B} \alpha_S\beta_T = \sum_{S\subseteq A}\alpha_S\sum_{T\subseteq B}\beta_T = 1$. Thus, fg is a valid probability generating polynomial for a distribution, $\Pr_{prod}$.

For assignments $\mathbf{X}_A = \mathbf{a}, \mathbf{X}_B = \mathbf{b}$, we set $S_\mathbf{a} = \{i \in A : a_i = 1\}$ and $S_\mathbf{b} = \{i \in B : b_i = 1\}$. Then, by definition, $\Pr_{prod}(\mathbf{X}_A = \mathbf{a}, \mathbf{X}_B = \mathbf{b})$ is given by the coefficient of the term $z^{S_\mathbf{a}\cup S_\mathbf{b}}$, which is $\alpha_{S_\mathbf{a}}\beta_{S_\mathbf{b}} = \Pr_f(\mathbf{X}_A = \mathbf{a})\Pr_f(\mathbf{X}_B = \mathbf{b})$.

□

Proof for Proposition 3. Let $g(z_1,\dots,z_n)$ be a normalized probability generating polynomial. Let $A_1,\dots,A_n$ be disjoint subsets of $\mathbb{N}^+$ and $f_1(\mathbf{z}_{A_1}),\dots,f_n(\mathbf{z}_{A_n})$ be normalized generating polynomials. Write $g = \sum_{S\subseteq\{1,\dots,n\}} \alpha_S z^S$. Then,

$$g\big|_{z_i = f_i} = \sum_{S\subseteq\{1,\dots,n\}} \alpha_S \prod_{i\in S} f_i$$

It follows from Proposition 2 (product operation) that $\prod_{i \in S} f_i$ are valid generating polynomials for $S \subseteq \{1, \ldots, n\}$; again by Proposition 2 (sum operation), $g\big|_{z_i = f_i} = \sum_{S \subseteq \{1,\ldots,n\}} \alpha_S \prod_{i \in S} f_i$ is a valid generating polynomial. □

B. Experiments

B.1. The Construction of SimplePGC

A SimplePGC is a weighted sum over several DetPGCs, which are defined in Section 4.2. The structure of a SimplePGC is governed by two hyperparameters, the number of DetPGCs in the weighted sum (denoted by C), and the maximum number of the variables (i.e. k in Figure 2) in the leaf distributions of the DetPGCs (denoted by K).

Partitioning Variables To construct SimplePGC, we first partition the variables $X_1, \ldots, X_n$ into several groups. The idea is, as shown in Section 4.2, for a DetPGC, variables from different groups have to be negatively dependent, so we want to put pairs of variables that are positively dependent in the same group. Given some training examples $D_1, \ldots, D_I$, we estimate the probabilities $\Pr(X_i = 1)$ and $\Pr(X_i = 1, X_j = 1)$ by counting; in particular, we set

$$\Pr(\text{event}) = \frac{|D_i \text{ where event is true}|}{I}.$$

Then, inspired by the definition of *pairwise mutual information*, when $\Pr(X_i = 1, X_j = 1) > 0$, we use the quantity

$$w_{ij} = \Pr(X_i = 1, X_j = 1) \log \frac{\Pr(X_i = 1, X_j = 1)}{\Pr(X_i = 1)\Pr(X_j = 1)}, \tag{9}$$

to measure the degree of positive dependence between X_i and X_j. Note that X_i and X_j are positively dependent if $w_{ij} > 0$. Then we partition the variables into groups by the following greedy algorithm.

Algorithm 1 Partition Variables

Input: variables $\{X_i\}_{1 \le i \le n}$, weights $\{w_{ij}\}_{1 \le i < j \le n}$ as defined by Equation (9)
Output: function $group$ that maps X_i to the group that X_i belongs to
Initialization: $group(X_i) \leftarrow \{X_i\}, W \leftarrow \{w_{ij} > 0\}_{1 \le i < j \le n}$

sort W in descending order
for w_{ij} **in** W **do**
  $union \leftarrow group(X_i) \cup group(X_j)$
  if $|union| \le K$ **then**
    $group(X_i) \leftarrow union$
    $group(X_j) \leftarrow union$
  end if
end for

Leaf PGCs After we partition the variables in to groups, we feed them to the leaf PGCs $\Pr_i$, as shown in Figure 2. $\Pr_i$ can be any PGCs; for SimplePGCs, we make the simplest choice by setting them to be the fully general distributions. For the leaf distribution $\Pr_i$ over variables $\{X_{i1}, \ldots, X_{ik}\}$, we let

$$g_i = \frac{1}{Z_i} \sum_{z_{i1}, \ldots, z_{ik} \in \{0,1\}} \exp(\theta_{i,S}) z^S$$

be its generating polynomial with parameters $\theta_{i,S}$ and normalizing constant Z_i. $\Pr_i$s are fully general except for the constraint that all-zero assignment must have zero probability; this constraint is nothing but a trick that makes implementation easier.

B.2. Supplementary Experiment Results

In addition to the experiment results presented in Figure 3, we also conducted one-sample t-tests. The results of the statistical test are shown in Figure 4. For the Twenty Datasets benchmark, the log-likelihood of Strudel is only statistically better than SimplePGC on 6 out of 20 datasets; the log-likelihood of EiNets is only statistically better than SimplePGC on 2 out of 20

datasets; the log-likelihood of MT is statistically better than SimplePGC on 7 out of 20 datasets. For the Amazon Baby Registries benchmark, the log-likelihoods of Strudel, EiNets and MT are statistically better than SimplePGC on none of the datasets.

It would also help if we estimate the confidence intervals of the average test log-likelihoods via cross validation. However, since we have four baselines and not all of their learning algorithms were designed to be fast, the computation cost for estimating the confidence intervals would be infeasible.

	Strudel	EiNet	MT	SimplePGC	vs. Strudel	vs. EiNet	vs. MT
nltcs	6.07	6.02	**6.01**	6.05	=	=	=
msnbc	**6.04**	6.12	6.07	6.06	$<$	$>$	=
kdd	2.14	2.18	**2.13**	2.14	=	$>$	=
plants	13.22	13.68	**12.95**	13.52	$<$	=	$<$
audio	42.20	**39.88**	40.08	40.21	$>$	=	=
jester	54.24	**52.56**	53.08	53.54	$>$	$<$	$<$
netflix	57.93	**56.54**	56.74	57.42	$>$	$<$	$<$
accidents	**29.05**	35.59	29.63	30.46	$<$	$>$	$<$
retail	**10.83**	10.92	**10.83**	10.84	=	=	=
pumsb	24.39	31.95	**23.71**	29.56	$<$	$>$	$<$
dna	87.15	96.09	85.14	**80.82**	$>$	$>$	$>$
kosarek	10.70	11.03	**10.62**	10.72	=	=	=
msweb	**9.74**	10.03	9.85	9.98	$<$	=	=
book	34.49	34.74	34.63	**34.11**	=	=	=
movie	53.72	**51.71**	54.60	53.15	=	=	=
webkb	**154.83**	157.28	156.86	155.23	=	=	=
reuters	86.35	87.37	**85.90**	87.65	=	=	$<$
20ng	**153.87**	153.94	154.24	154.03	=	=	=
bbc	256.53	**248.33**	261.84	254.81	=	=	=
ad	16.52	26.27	**16.02**	21.65	$<$	$>$	$<$

(a) One-sided t-test results on the Twenty Datasets benchmark.

	Strudel	EiNet	MT	SimplePGC	vs. Strudel	vs. EiNet	vs. MT
apparel	9.51	9.24	9.31	**9.10**	$>$	=	$>$
bath	8.38	8.49	8.53	**8.29**	=	$>$	$>$
bedding	8.50	8.55	8.59	**8.41**	=	=	$>$
carseats	4.79	4.72	4.76	**4.64**	$>$	=	$>$
diaper	9.90	9.86	9.93	**9.72**	$>$	=	$>$
feeding	11.42	11.27	11.30	**11.17**	$>$	=	=
furniture	4.39	4.38	4.43	**4.34**	=	=	=
gear	9.15	9.18	9.23	**9.04**	=	$>$	$>$
gifts	**3.39**	3.42	3.48	3.47	=	=	=
health	7.37	7.47	7.49	**7.24**	$>$	$>$	$>$
media	**7.62**	7.82	7.93	7.69	=	=	=
moms	3.52	**3.48**	3.54	3.53	=	=	=
safety	4.43	4.39	4.36	**4.28**	$>$	$>$	=
strollers	5.07	5.07	5.14	**5.00**	=	=	$>$
toys	**7.61**	7.84	7.88	7.62	=	$>$	$>$

(b) One-sided t-test results on the Amazon Baby Registries benchmark.

Figure 4. Results for one-sample two-sided t-test on two benchmarks with $p = 0.1$. In the 5th and 6th columns of the tables, = means a statistical tie, $>$ means that the log-likelihood of SimplePGC is statistically better, and $<$ means that of SimplePGC is statistically worse. Note that a statistical tie does not necessarily mean there is no difference in terms of performance. Bold numbers indicate the best log-likelihood.