Probabilistic Generating Circuits—Demystified*

Sanyam Agarwal [1] **Markus Bläser** [1]

Abstract

Zhang et al. (ICML 2021, PLMR 139, pp. 12447–12457) introduced probabilistic generating circuits (PGCs) as a probabilistic model to unify probabilistic circuits (PCs) and determinantal point processes (DPPs). At a first glance, PGCs store a distribution in a very different way, they compute the probability generating polynomial instead of the probability mass function and it seems that this is the main reason why PGCs are more powerful than PCs or DPPs. However, PGCs also allow for negative weights, whereas classical PCs assume that all weights are nonnegative. One main insight of this work is that the negative weights are the cause for the power of PGCs and not the different representation. PGCs are PCs in disguise: we show how to transform any PGC on binary variables into a PC with negative weights with only polynomial blowup. PGCs were defined by Zhang et al. only for binary random variables. As our second main result, we show that there is a good reason for this: we prove that PGCs for categorical variables with larger image size do not support tractable marginalization unless $\mathsf{NP} = \mathsf{P}$. On the other hand, we show that we can model categorical variables with larger image size as PC with negative weights computing set-multilinear polynomials. These allow for tractable marginalization. In this sense, PCs with negative weights strictly subsume PGCs.

*Broadrick et al. (2024) independently obtain some of the results presented in this paper. In particular, they prove that probabilistic circuits and probabilistic generating circuits for binary variables are equivalent (our Theorem 8.1) as well as the hardness of marginalization for probabilistic generating circuits with at least four categories (our Theorem 7.1). These results were obtained independently of ours and were submitted to a conference around the same time as ours.

[1]Saarland University, Saarland Informatics Campus, Saarbrücken, Germany. Correspondence to: Sanyam Agarwal <agarwal@cs.uni-saarland.de>, Markus Bläser <mblaeser@cs.uni-saarland.de>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

1. Introduction

Probabilistic modeling is a central task in machine learning. When the underlying models become large and complicated, however, probabilistic inference easily becomes intractable, see (Roth, 1996) for an explanation. Therefore, it is important to develop probabilistic models that are tractable (TPMs for short), that is, they allow for efficient probabilistic inference. On the other hand, the probabilistic models should be as expressive efficient as possible (in the sense of Martens & Medabalimi (2014)), which means that they are able to represent as many different distributions as possible. The more classes we can represent, the broader the spectrum of applications of the model. There is typically a tradeoff between expressiveness and tractability. The more expressive the model, the harder will be probabilistic inference.

Examples of tractable models are for instance bounded treewidth graphical models (Meila & Jordan, 2000; Koller & Friedman, 2009), the well-known determinantal point processes (Borodin & Rains, 2005; Kulesza & Taskar, 2012), or probabilistic circuits like for instance sum-product networks (Darwiche, 2009; Kisa et al., 2014; Poon & Domingos, 2012). These models represent probability distributions by computing probability mass functions: the input is an assignment to the random variables and the output is the corresponding probability of the event.

Probabilistic circuits (PCs), see also Section 2, compute (unnormalized) probability distributions for *syntactic*[1] reasons: the weights on the edges are required to be nonnegative. So-called *decomposable* PCs are tractable again for *syntactic* reasons: The scopes of each product gate are disjoint and therefore marginalization and multiplication commute.

Another popular tractable model are *determinantal point processes (DPPs)*. Zhang et al. (2020) show that PCs in general do not subsume DPPs. Shortly after, Zhang et al. (2021) propose a new model called *probabilistic generating circuits (PGCs)* that represent a distribution by a generating polynomial. PGCs allow for efficient marginalization and subsume both PCs and DPPs. It seems that the power of PGCs comes from the fact that they use a different rep-

[1]By a syntactic property, we mean a property that can be checked by inspecting the structure of the circuit, but we do not need to take the input-output behaviour of the circuit into account, in contrast to semantic properties.

resentation. However, there is another subtle difference between PGCs and PCs, namely, PGCs allow for negative constants. Zhang et al. (2021, page 12447) write: "Because of the presence of negative parameters, it is not guaranteed that the polynomials represented by a PGC is a probability generating polynomial: it might contain terms that are not multiaffine or have negative coefficients". Therefore it is a *semantic* property that a PGC represents a probability distribution, the designer of the PGC has to ensure that it computes a probability generating function.

One of the key insights of this paper is that the important property of PGCs is that they allow for negative constants and not the fact that they compute a probability generating function instead of a probability distribution itself. More precisely, we show how to turn any PGC on binary variables into a PC with negative weights that computes the probability distribution represented by the PGC and this PC computes a *set-multilinear* polynomial (see also Definition 9.1). The syntactic property of nonnegative weights is replaced by the semantic property of computing a probability distribution and the syntactic property of being decomposable (also called "syntactically set-multilinear") is replaced by the semantic property of computing a set-multilinear polynomial (but intermediate results might not be set-multilinear). So PGCs are nothing but PCs in disguise.

2. Probabilistic circuits

An *arithmetic* circuit is an acyclic directed graph. Nodes of indegree 0 are called *input nodes*. Internal nodes are either addition nodes or multiplication nodes. Addition nodes have edge weights on the incoming edges and compute the weighted sum of the inputs. Product nodes compute the unweighted product of its inputs.

Probabilistic circuits (PCs) are representations of probability distributions that allow tractable inference. See (Choi et al., 2020) for an in-depth description. They are arithmetic circuits such that the input nodes correspond to probability distributions on random variables X_i. In the case of binary random variables, we can for instance assume that each distribution is given as $px_i + (1-p)\overline{x}_i$ and therefore the input nodes are labelled by variables x_i and $\overline{x}_i$. PCs compute probability mass functions. Figure 1 shows an example.

Example 2.1. If we set $x_1 = 1$ and $\overline{x}_1 = 0$ as well as $x_2 = 0$ and $\overline{x}_2 = 1$ in the PC in Figure 1, we get $\frac{1}{6}$ as a result, which is $\Pr[X_1 = 1, X_2 = 0]$.

The edge weights of a PC are typically assumed to be nonnegative, therefore, PCs compute (unnormalized) probability distributions by design. We will later also consider PCs with potentially negative weights. We will call these *nonmonotone* PCs. Here, the designer of the PC has to ensure that it computes a probability distribution.

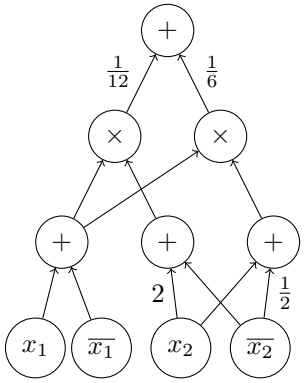


Figure 1. A PC over binary random variables, computing

Pr	$X_1 = 0$	$X_1 = 1$
$X_2 = 0$	$\frac{1}{6}$	$\frac{1}{6}$
$X_2 = 1$	$\frac{1}{3}$	$\frac{1}{3}$

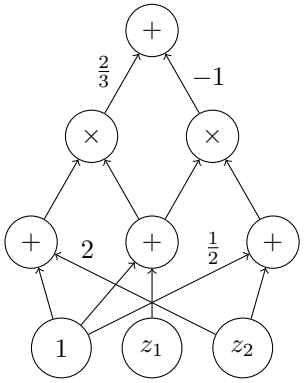


Figure 2. Example of a PGC computing the probability generating function of the distribution on the left.

PCs allow for tractable marginalization provided that the PC is *decomposable* and *smooth*. Decomposable means that for each multiplication gate, the scopes of the children (i.e., variables they depend on) are disjoint. A PC is smooth, if for each addition gate the scopes of the children are the same. While decomposability is a crucial property, smoothness is typically easy to ensure, see (Shih et al., 2019).

Example 2.2. The PC in Figure 1 is decomposable and smooth. If we set $x_1 = 1$ and $\overline{x}_1 = 0$ as well as $x_2 = \overline{x}_2 = 1$, we get $\Pr[X_1 = 1] = \frac{1}{2}$.

3. Probabilistic generating circuits

Given categorical variables $X_1, \ldots, X_n$ with image $\{0, \ldots, d-1\}$ and joint distribution $p(a_1, \ldots, a_n) = \Pr[X_1 = a_1, \ldots, X_n = a_n]$, the *probability generating function* is a formal polynomial in formal variables $z_1, \ldots, z_n$ defined by

$$G(z) = \sum_{j_1=0}^{d-1} \cdots \sum_{j_n=0}^{d-1} p(j_1, \ldots, j_n) z_1^{j_1} \cdots z_n^{j_n}. \quad (1)$$

A *probabilistic generating circuit (PGC)* for a probability distribution p is an arithmetic circuit that computes G. While PCs compute probability mass functions, PGCs store probability distributions as formal objects. We can feed some particular elementary event as an input into a PC and the corresponding output is the probability of this event. In PGCs, the probabilities are stored as coefficients. Figure 2 shows an example. While PCs can also model continuous distributions, PGCs can only model distributions over categorical variables due to the way they store distributions.

Zhang et al. (2021) introduce PGCs only for binary random variables and showed how to perform tractable marginal-

ization in this case. While binary variables are the most important categorical random variable, categorical variables with a larger number of outputs are important, too. Applications of ternary variables for example are excess losses or labelling with abstention (Wu & Seldin, 2022; Mhammedi et al., 2019; Thulasidasan et al., 2019).

The work by Zhang et al. (2021) led to further work on PGCs: Harviainen et al. (2023, Theorem 3) designed a faster marginalization procedure for PGCs, while Bläser (2023) constructed a strongly Rayleigh distribution that cannot be represented by small PGCs.

4. Our results

Zhang et al. (2021) show that given a PGC over binary random variables, marginalization is tractable. This raises the natural question whether this is also possible for categorical variables attaining more than two values. We give a negative answer to this question (under standard complexity theoretic assumptions) in Section 7. If marginalization for PGCs over quarternary random variables can be done in polynomial time, then $\mathsf{NP} = \mathsf{P}$. For ternary variables, we get a similar result, however, we need to be able to marginalize over subsets of the image. For the result on quaternary variables, it is sufficient to marginalize over the whole image set.

Zhang et al. (2021) show that *determinantal point processes* (DPPs) can be represented by PGCs. On the other hand, it is not clear whether decomposable and smooth PCs (which support tractable marginalization) can represent DPPs. This question is related to proving set-multilinear lower bounds for the determinant in algebraic complexity, see (Saptharishi, R. et al., 2021) for an overview. See also (Zhang et al., 2020) for further limitations of representing DPPs by PCs. As our second main result, we here prove that the additional power of the PGC by Zhang et al. (2021) does not come from the fact that they use a different representation but from the fact that one allows for negative constants. In particular we prove that every PGC over binary random variables can be transformed into a nonmonotone PC, which computes the corresponding probability mass function and allows for tractable marginalization (Section 8).

Third, we prove in Section 9 that nonmonotone PCs computing set-multilinear polynomials are more general than PGCs in the sense that they support tractable marginalization over categorical variables of an arbitrary image size. Since computing a set-multilinear polynomial and computing a probability distribution are semantic properties (i.e. they cannot be directly inferred from the structure of the PC), we also ask the question whether checking these properties is hard? It turns out that the first one can be checked in randomized polynomial time, while the second property is hard to decide.

In Section 10, we present basic compositional operations, which preserve the property that a nonmonotone PC computes a probability distribution. The first two are the well-known weighted sum and multiplication (when the domains are disjoint). The third one is an analogue of the hierarchical composition introduced by Zhang et al. (2021) for PGCs.

Finally, in Section 11, we discuss the relation between nonmonotone PCs and DPPs. PGCs and nonmonotone PCs were designed to subsume monotone PCs and DPPs. It is well known that nonmonotone PCs are strictly stronger than monotone ones. DPPs can only compute distributions with negative correlations. In this sense, nonmonotone PCs are strictly stronger than DPPs. However, once we allow to combine DPPs with simple compositional operations like affine projections, we show that the question whether nonmonotone PCs are more powerful than DPPs will be very hard to answer (Theorem 11.2). It will imply a separation between algebraic formulas and circuits, a question that has been open for decades, see (Bürgisser et al., 1997, Problem 21.2). While it is well-known that we can write a polynomial computed by a formula as a determinant, see e.g. (Bürgisser et al., 1997, Theorem 21.27), the crucial point here is that the variables of a DPP only appear on the diagonal. We here prove that every polynomial computed by a formula can be written as an affine projection of a DPP of size linear in the formula size. That means separating DPPs and nonmonotone PCs implies a separation of algebraic formulas and circuits. This result can even be strengthened to so-called algebraic branching programs instead of formulas (Theorem D.1).

5. Graphs and matchings

In the next two sections, we briefly review definitions and results from graph theory and computational complexity, which will be needed for our hardness results.

A graph G is called *bipartite*, if we can partition its nodes into two set U and V such that all edges have one node in U and the other node in V. When we write $G = (U \cup V, E)$ we mean that G is a bipartite graph with bipartition (U, V). We will typically call these nodes $u_1, \dots, u_m$ and $v_1, \dots, v_n$. The *degree* of a node is the number of edges that it is incident to. A graph is called *regular* if every node has the same degree. It is *d-regular* if it is regular and the degree of every node is d. The *neighbours* of a node u are the nodes that share an edge with u.

Example 5.1. Figure 3 shows a 3-regular bipartite graph with four nodes on each side. The neighbours of u_1 are v_1, v_2, v_3 but not v_4.

$M \subseteq E$ is a *matching* if each node of U and V appears in at most one edge of M. M is called *perfect*, if every node is in exactly one edge. If a bipartite graph has a perfect matching,

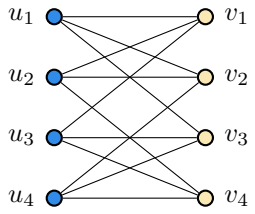


Figure 3. A 3-regular bipartite graph with bipartition $U = \{u_1, u_2, u_3, u_4\}$ and $V = \{v_1, v_2, v_3, v_4\}$.

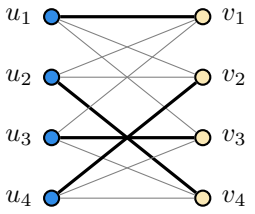


Figure 4. The thick edges form a perfect matching. Any subset of it forms a matching.

then necessarily $|U| = |V|$. The size $|M|$ of a matching is the number of edges in it. If M is perfect, then $|M| = |U| = |V|$. The set of all matchings of G is denoted by $\mathcal{M}(G)$ and the set of all perfect matchings by $\mathcal{PM}(G)$. The number of all matchings and perfect matchings is denoted by $\#\mathcal{M}(G)$ and $\#\mathcal{PM}(G)$, respectively.

Example 5.2. Figure 4 shows a perfect matching in the graph from Figure 3.

6. Complexity theory basics

We give some background information on the complexity classes and results used in this paper. We refer to (Papadimitriou, 1994; Arora & Barak, 2009) for further explanations and proofs of the well-known definitions and theorems in this section. The important fact for this paper is that counting perfect matchings in bipartite graphs is hard.

Deciding whether a formula ϕ in conjunctive normal form (CNF) has a satisfying assignment is the defining problem of the famous class NP. If we instead want to count the number of satisfying assignments, we get a problem which is complete for the class #P defined by Valiant (1979). Obviously, when you can count the number of satisfying assignments, then you can decide whether there is at least one, therefore, #P is a "harder" class than NP. It turns out that some problems become #P-hard when considered as a counting problem while their decision versions are easy. Perfect matchings in bipartite graphs is such an example: There are efficient algorithms for the decision problem, but the counting version is hard.

Theorem 6.1 (Valiant (1979))**.** *Counting perfect matchings in bipartite graphs is* #P*-complete under Turing reductions.*

Above, a problem A is Turing reducible to a problem B if there is a polynomial time deterministic Turing machine that solves A having oracle access to B. This means, that an efficient algorithm for B would yield an efficient algorithm for A and in this sense, A is easier than B.

Note that #P is a class of functions, not of languages. The class FP is the class of all functions computable in polynomial time. It relates to #P like P relates to NP. If #P = FP, then NP = P.

7. PGCs do not support efficient marginalization beyond binary variables

Theorem 7.1. *Efficient marginalization over PGCs involving quaternary random variables implies that* #P = FP*, and in particular,* NP = P.

Proof. Counting perfect matchings in 3-regular bipartite graphs is a #P-hard problem, as proved by Dagum & Luby (1992, Theorem 6.2). Given a 3-regular bipartite graph $G = (U \cup V, E)$ with $|U| = |V| = n$, we will construct a PGC C of size polynomial in n (in fact linear in n) over quaternary random variables such that a certain marginal probability is the number of perfect matchings in G.

For every vertex $v_i \in V$, we define a formal variable V_i. Similarly, for every edge $e_{i,j}$ representing an edge between $u_i \in U$ and $v_j \in V$, we define a binary random variable $E_{i,j}$. For every vertex $u_i \in U$, let $N(i)$ denote the set of indices of its neighbours. Since G is 3-regular, $|N(i)| = 3$. In particular, let $N(i,1), N(i,2), N(i,3)$ denote the indices of the three neighbours of u_i. We now define the polynomial f as

$$f(V_1, ..., V_n, E_{1,N(1,1)}, ..., E_{n,N(n,3)}) = \prod_{i=1}^{n} \sum_{j \in N(i)} E_{i,j} V_j \tag{2}$$

The right hand side of the equation is a depth-2 circuit of size $s = O(n)$, as each sum has only three terms due to 3-regularity. Thus f has a linear sized circuit.

Furthermore, f is a polynomial of $\deg(f) = 2n$. Each $E_{i,j}$ appears with degree 1 in f, since $E_{i,j}$ only appears in the ith factor of the outer product in (2). Each V_j appears with degree 3 in f, since each V_j appears in three factors in (2) due to the 3-regularity of G. The total number of variables appearing in f is $n + 3n = 4n$.

Since each coefficient of the product in the inner sums is 1, the coefficients of f are all nonnegative. Thus f can be viewed as the probability generating function of the (unnormalized) joint probability distribution of $4n$ quaternary random variables. The normalization constant is the sum of all coefficients of f, which is just $f(1, \ldots, 1)$. Hence, the normalization constant can be efficiently computed using the circuit for f. In this particular case, it is even easier. By looking at the righthand side of (2), we see that the normalization constant is 3^n. Thus, $\hat{f} := f/3^n$ is a (normalized) probability distribution over quaternary variables and is computed by a linear sized PGC.

We now view $\hat{f}$ as a polynomial in $V_1, \ldots, V_n$ with co-

efficients being polynomials in $E_{i,N(i,j)}$, $i = 1, \ldots, n$, $j = 1, 2, 3$. Let $h(E_{1,N(1,1)}, ..., E_{n,N(n,3)})$ be the coefficient of $V_1 V_2 \ldots V_n$. We claim that the number of monomials in h gives us the number of perfect matchings in G. Any perfect matching in G would be of the form (u_1, v_{k_1}), (u_2, v_{k_2}), ..., (u_n, v_{k_n}), such that $k_i \neq k_j$ for $i \neq j$, and $\{k_1, ..., k_n\} = [n]$. But then the monomial $E_{1,k_1} \cdot ... \cdot E_{n,k_n}$ would be present in h. For the converse, let $E_{1,a_1} \cdot ... \cdot E_{n,a_n}$ be some monomial in h. Clearly, $a_i \neq a_j$ for $i \neq j$, as each V_i only occurs in one factor in (2). Further, the coefficient of $E_{1,a_1} \cdot ... \cdot E_{n,a_n}$ in h is nonzero only if all the edges $(u_1, v_{a_1}), (u_2, v_{a_2}), \ldots, (u_n, v_{a_n})$ were present in G. Thus, any such monomial would denote a perfect matching in G. Hence, the number of monomials in h and the number of perfect matchings in G are equal.

Suppose we can efficiently marginalize in PGCs over quaternary random variables. Then we can compute $\Pr[V_1 = 1, V_2 = 1, \ldots, V_n = 1]$ efficiently. (By abuse of notation V_i also denotes the random variable corresponding to the formal variable V_i.) However, this probability is nothing but $h(1, \ldots, 1)$. Each monomial in h has the coefficient $1/3^n$. Thus

$$\Pr[V_1 = 1, \ldots, V_n = 1] = h(1, \ldots, 1) = \frac{\#\mathcal{PM}(G)}{3^n}.$$

Using marginalization we can efficiently get the number of perfect matchings in a 3-regular bipartite graph G. However, as computing the number of perfect matching in 3-regular bipartite graphs is a #P-complete problem, we get #P = FP. This implies NP = P. □

Zhang et al. (2021) show that we can marginalize efficiently over PGCs on binary random variables. Combined with the above result, it naturally raises the question whether there exists efficient marginalization for PGC involving ternary random variables. While we are not able to answer the question fully here, we do show that "selective" marginalization over such distributions should not be possible under standard complexity theoretic assumptions. In order to prove this result, we will first introduce some definitions.

Definition 7.2 (Selective Marginalization)**.** Let $X_1, \ldots, X_n$ be k-nary random variables. Let $V_i \subseteq \{0, 1, \ldots, k-1\}$, $1 \leq i \leq n$. $\Pr[X_1 \in V_1, \ldots, X_n \in V_n]$ is called a selective marginal probability.

Let $f(z_1, \ldots, z_n)$ be the corresponding probability generating polynomial:

$$f(z_1, ..., z_n) = \sum_{s=(s_1,...,s_n) \in \{0,1,...,k-1\}^n} c_s \cdot \prod_{i=1}^{n} z_i^{s_i}.$$

Then we want to compute $\sum_{s_1 \in V_1, ..., s_n \in V_n} c_s$.

Theorem 7.3. *Efficient selective marginalization over PGCs involving ternary random variables implies* #P = FP*, and in particular,* NP = P.

The proof is deferred to Appendix A.

8. PGCs over binary variables can be simulated by nonmonotone PCs

Theorem 8.1. *PGCs over binary variables can be simulated by nonmonotone PCs with only polynomial overhead in size.*

Proof. Let $f(z_1, ..., z_n)$ be a probability generating function of n binary random variables computed by a PGC of size s. We define the polynomial g as

$$g(x_1, \overline{x_1}, ..., x_n, \overline{x_n}) = f(\frac{x_1}{\overline{x_1}}, \frac{x_2}{\overline{x_2}}, ..., \frac{x_n}{\overline{x_n}}) \cdot \prod_{i=1}^{n} \overline{x_i}.$$

g is indeed a polynomial, since f is multilinear. To get a circuit for g, first we need to substitute all instances of z_i with $\frac{x_i}{\overline{x_i}}$ in the circuit for f, using the help of division gates, which we will remove later. Clearly, this gives us a circuit for $f(\frac{x_1}{\overline{x_1}}, \frac{x_2}{\overline{x_2}}, ..., \frac{x_n}{\overline{x_n}})$ with size $O(s + n)$. Furthermore, we can compute $\prod_{i=1}^{n} \overline{x_i}$ in size $O(n)$, and thus we get a circuit for g of size $O(s + n)$.

Let m be any monomial of $f(z_1, z_2, ..., z_n)$. Then, $m = c_S \cdot \prod_{i \in S} z_i$ for some $S \subseteq [n]$. Looking at the corresponding monomial m' in $g(x_1, \overline{x_1}, ..., x_n, \overline{x_n})$, we see that

$$m' = c_S \cdot (\prod_{i \in S} \frac{x_i}{\overline{x_i}}) \cdot (\prod_{j=1}^{n} \overline{x_j}) = c_S (\prod_{i \in S} x_i) \cdot (\prod_{i \notin S} \overline{x_i})$$

Thus, if z_i occurs in m, then x_i occurs in m'. Similarly, if z_i does not occur in m, then $\overline{x_i}$ occurs in m'. Thus, in m' exactly one of x_i or $\overline{x_i}$ occurs for all i. Hence, m' is multilinear with $\deg(m') = n$. Since, m' was an arbitrary monomial of g, this implies g is both multilinear and homogenous with $\deg(g) = n$. Thus, we obtained a nonmonotone PC, cf. (Zhang et al., 2021, Proposition 1).

Right now the circuit of g has division gates. This is a problem, since we might want to set $\overline{x}_i = 0$ when computing a probability. However using the famous result of Strassen (1973), we can eliminate all division gates to get a circuit for g of size $\text{poly}(s + n, \deg(g)) = \text{poly}(s, n)$. Furthermore, g computes a probability distribution since f represented a probability distribution. Thus, starting with a PGC of size s, we get a nonmonotone PC computing a set-multilinear polynomial g of size $\text{poly}(s, n)$. □

Remark 8.2. In the proof by Strassen, every arithmetic operation in the original circuit is replaced by a corresponding operation on polynomials of degree n. Therefore, the exact upper bound on the size of the new circuit is

$O(s \cdot n \log n \log\log n)$ if we use the fast polynomial multiplication by Cantor & Kaltofen (1991) or $O(s \cdot n)$ if we use interpolation.

Example 8.3. The following small example explains Strassen's construction: Suppose we have a PGC that computes the following polynomial: $f(z_1, z_2) = 0.6z_1z_2 + 0.4z_1$. We will demonstrate the conversion of the polynomial into a polynomial computed by the corresponding nonmonotone PC.

- We first replace each z_i with $\frac{x_i}{\overline{x_i}}$, and then get $g = f(\frac{x_1}{\overline{x_1}}, \frac{x_2}{\overline{x_2}}) \cdot \overline{x_1}\,\overline{x_2} = 0.6x_1x_2 + 0.4x_1\overline{x_2}$. However, the new circuit contains the two divisions $\frac{x_1}{\overline{x_1}}$ and $\frac{x_2}{\overline{x_2}}$, which we have to remove.
- The idea by Strassen (1973) is to expand $\frac{1}{\overline{x_1}}$ as a formal power series. Since we are computing a polynomial in the end, it is enough to work with finite approximations. We can only invert a power series if it has a nonzero constant term. Therefore, we first have to perform a Taylor shift, which will be reverted in the end. In our case, the shift $\overline{x_i}$ to $1 - \overline{x_i}$ works. We get $f(\frac{x_1}{1-\overline{x_1}}, \frac{x_2}{1-\overline{x_2}}) \cdot (1 - \overline{x_1})(1 - \overline{x_2}) = 0.6x_1x_2 + 0.4x_1(1 - \overline{x_2})$.
- The inverse of $1 - x$ as a formal power series is the geometric series $\frac{1}{1-x} = \sum_{i=0}^{\infty} x^i$. Since in this example, we compute a multilinear function of degree two, it is enough to work with order-one approximations and replace $\frac{1}{1-\overline{x_i}}$ by $1 + \overline{x_i}$. We get

$$\begin{aligned} &f(x_1(1+\overline{x_1}), x_2(1+\overline{x_2})) \cdot (1-\overline{x_1})(1-\overline{x_2}) \\ &= (0.6x_1x_2(1+\overline{x_1})(1+\overline{x_2}) \\ &\quad + 0.4x_1(1+\overline{x_1}))(1-\overline{x_1})(1-\overline{x_2}) \\ &= 0.6x_1x_2 + 0.4x_1(1-\overline{x_2}) \;+\; \text{higher degree terms.} \end{aligned} \tag{3}$$

- To compute only the terms up to degree two and remove the unwanted higher degree terms, we compute the homogeneous parts separately. This is the part where we incur the blowup in the circuit size.
- Now, finally to get back to g, we need to invert the substitutions that we made earlier. Hence, replacing $\overline{x_i}$ with $1 - \overline{x_i}$ for $i \in \{1, 2\}$ in (3) after removing the higher degree terms gives us the polynomial $0.6x_1x_2 + 0.4x_1\overline{x_2}$ which is in fact g.

9. Nonmonotone PCs computing set-multilinear polynomials support tractable marginalization

We consider categorical random variables $X_1, \ldots, X_n$, w.l.o.g. taking values in $I = \{0, 1, \ldots, d-1\}$. With each random variable X_i, we associate d indeterminates $z_{i,0}, \ldots, z_{i,d-1}$. A probability distribution for X_i given by $\Pr[X_i = \delta] = \alpha_\delta, 0 \le \delta < d$, can be modelled by the linear polynomial $\ell(z_i) = \sum_{\delta=0}^{d-1} \alpha_\delta z_{i,\delta}$. By setting $z_{i,\delta}$ to 1 and all other $z_{i,\delta'}$ for $\delta' \neq \delta$ to 0, that is, evaluating p_i at the unit vector e_δ, we get $\ell(e_\delta) = \alpha_\delta$. More generally, if we want to compute $\Pr[X_i \in S]$ for some set $S \subseteq \{0, \ldots, d-1\}$, we can compute this by evaluating ℓ at v, where $v_i = 1$ if $i \in S$ and $v_i = 0$ otherwise.

Definition 9.1. Let X be a set of variables and $Y_1, \ldots, Y_t$ be a partition of X. A polynomial p in variables X is called *set-multilinear* with respect to the above partition, if every monomial of p contains exactly one variable from each set Y_τ, $1 \le \tau \le t$, with degree 1. In particular, p is homogeneous of degree t.

Example 9.2. A decomposable and smooth PC over binary variables computes a set-multilinear polynomial with the parts given by $\{x_i, \overline{x}_i\}$, $1 \le i \le n$.

Given a decomposable and smooth PC over categorical random variables $X_1, \ldots, X_n$, we can model every input distribution by a linear form as described above. The corresponding polynomial will be set-multilinear with the parts of the partition being $\{z_{i,0}, \ldots, z_{i,d-1}\}$, $1 \le i \le n$. However, when the PC is nonmonotone, that is, we allow for negative weights, then the PC can compute a set-multilinear polynomial, even if it is not decomposable. It turns out that this is sufficient for performing marginalization.

Theorem 9.3. *Let C be a nonmonotone PC of size s computing a probability distribution over categorical random variables $X_1, \ldots, X_n$ such that the polynomial P computed by C is set-multilinear with respect to the partition $\{z_{i,0}, \ldots, z_{i,d-1}\}$, $1 \le i \le n$. Let $A_1, \ldots, A_n \subseteq \{0, \ldots, d-1\}$. Then we can compute $\Pr[X_1 \in A_1, \ldots, X_n \in A_n]$ in time $O(s)$.*

Proof. Note that

$$P = \sum_{j_1=0}^{d-1} \cdots \sum_{j_n=0}^{d-1} \Pr[X_1 = j_1, \ldots, X_n = j_n] z_{1,j_1} \cdots z_{n,j_n}. \tag{4}$$

Consider the elementary event $X_1 = a_1, \ldots, X_n = a_n$. Define an input vector e for P by

$$e_{i,j} = \begin{cases} 1 & \text{if } j = a_i, \\ 0 & \text{otherwise} \end{cases}$$

for $1 \le i \le n$, $0 \le j \le d-1$. By (4), $P(e) = \Pr[X_1 = a_1, \ldots, X_n = a_n]$. From all monomials in P, only $z_{1,a_1} \ldots z_{n,a_n}$ evaluates to 1 under e and all others evaluate to 0. Now to compute $\Pr[X_1 \in A_1, \ldots, X_n \in A_n]$, we simply evaluate at the point

$$v_{i,j} = \begin{cases} 1 & \text{if } j \in A_i, \\ 0 & \text{otherwise.} \end{cases}$$

We claim that $P(v) = \Pr[X_1 \in A_1, \ldots, X_n \in A_n]$: A monomial $z_{1,j_1} \cdots z_{n,j_n}$ evaluated at v becomes 1 iff $j_i \in A_i$ for all $1 \leq i \leq n$. Otherwise, it evaluates to 0. Thus

$$\begin{aligned} P(v) &= \sum_{j_1 \in A_1} \cdots \sum_{j_n \in A_n} \Pr[X_1 = j_1, \ldots, X_n = j_n] \cdot 1 \\ &= \Pr[X_1 \in A_1, \ldots, X_n \in A_n], \end{aligned}$$

which proves the claim. □

Remark 9.4. This gives also an alternative marginalization algorithm for PGCs over n binary variables. We convert it into a equivalent nonmonotone PC computing a set-multilinear polynomial. The total running time will be $O(s \cdot n)$. The extra factor n comes from the conversion to a PC. The running time matches the one by Harviainen et al. (2023).

While being decomposable or being smooth is a *syntactic* property, that is, it is a property of the circuit and can be checked efficiently, computing a set-multilinear polynomial and computing a probability distribution are *semantic* properties[2], that is, properties of the polynomial computed by the circuit. We can of course compute the coefficients of the polynomial to check whether it is set-multilinear or evaluate the circuit at all inputs to see whether it is a probability distribution, but this requires exponential time. Can we check these properties nevertheless efficiently? It turns out that the first property is efficiently checkable while the second is most likely not.

Proposition 9.5. *Testing whether a nonmonotone PC computes a set-multilinear polynomial with respect to a given partition can be done in randomized polynomial time.*

Proposition 9.6. *Testing whether a nonmonotone PC computes a probability distribution is* NP*-hard.*

The proof of the first proposition can be found in Appendix B. The proof of the second proposition essentially follows from the hardness result by Harviainen et al. (2023, Theorem 5) combined with our transformation of PGCs to PCs (Theorem 8.1).

10. Compositional operations for nonmonotone PCs

Since it is hard to determine whether a nonmonotone PC computes a probability distribution, we here present three compositional operations for nonmonotone PCs that preserve the property that the PC computes a probability distribution and computes a set-multilinear polynomial, analogous to (Zhang et al., 2021). The first two are the well-known multiplication and mixing operations. The third one is more interesting, it is an analogue of the hierarchical composition for PGC introduced by Zhang et al. (2021) for nonmonotone PCs computing set-multilinear polynomials. The proofs of the results of this section are in Appendix C.

Let $X_1, \ldots, X_n$ be categorical random variables with image $\Delta = \{0, \ldots, d-1\}$. Let $A, B \subseteq \{1, \ldots, n\}$. Let C, D be two nonmonotone PCs computing joint probability distributions f and g for $X_A = (X_i)_{i \in A}$ and $X_B = (X_j)_{j \in B}$, resp., as set-multilinear polynomials in the variables $z_{i,j}$, $1 \leq i \leq n$, $0 \leq j \leq d-1$. That is

$$f(a) = \sum_{a \in \Delta^{|A|}} \alpha_a \prod_{i \in A} z_{i,a_i}, \quad g(b) = \sum_{b \in \Delta^{|B|}} \beta_b \prod_{j \in B} z_{j,b_j},$$

where $\alpha_a = \Pr[X_A = a]$ and $\beta_b = \Pr[X_b = b]$. Let s and t be the sizes of C and D respectively.

We want to construct a mixture of the two distributions. To this aim, we can extend f to a probability distribution on $X_{A \cup B}$ by

$$f(a, b') = \left(\sum_{a \in \Delta^{|A|}} \alpha_a \prod_{i \in A} z_{i,a_i} \right) \prod_{j \in B \setminus A} \frac{1}{d}(z_{j,0} + \cdots + z_{j,d-1}),$$

where b' corresponds to the variables $X_{B \setminus A}$. In the same way, we can extend g. This is a kind of "smoothening" operation, which ensures that both distributions have the same domain. For each variable that is not in the scope of f, we add a uniform distribution.

Proposition 10.1. *There is a nonmonotone PC of size $O(s + t + nd)$ computing the mixture $\alpha f + (1-\alpha)g$ on $X_{A \cup B}$ for any $0 \leq \alpha \leq 1$ as a set-multilinear polynomial.*

Proposition 10.2. *If A and B are disjoint, then there is a nonmonotone PC of size $O(s + t)$ computing the product distribution $f(a) \cdot g(b)$ as a set-multilinear polynomial.*

Finally, we can also define a hierarchical composition. Let f be a distribution on binary random variables given as a set-multilinear polynomials in variables $z_1, \ldots, z_n, \overline{z}_1, \ldots, \overline{z}_n$. Let $g_1, \ldots, g_n$ be distributions on m disjoint d-ary random variables each given as set-multilinar polynomials in variables $y_{i,j,k}$, $1 \leq i \leq n$, $1 \leq j \leq m$, $0 \leq k \leq d-1$.

Definition 10.3. The hierarchical composition of f and $g_1, \ldots, g_n$ is defined by the set-multilinear polynomial obtained by replacing z_i by g_i and $\overline{z}_i$ by $\prod_{j=1}^{m} \sum_{k=1}^{d} \frac{1}{d} y_{i,j,k}$, which is the uniform distribution on the domain of g_i.

Proposition 10.4. *The hierarchical composition is indeed a probability distribution. It can be computed by a nonmonotone PC whose size is linear in the sum of the sizes of the nonmonotone PCs for f and $g_1, \ldots, g_n$.*

[2] This is only true for nonmonotone PCs, for monotone PCs these conditions are equivalent, see (Vergari et al., 2020).

11. Nonmonotone PCs computing set-multilinear polynomials versus DPPs

Determinantal point processes (DPPs) are stochastic point processes whose probability distribution is characterized by a determinant of some matrix. DPPs are important because they are able to express negative dependencies. For the purposes of modeling real data, the class of DPPs is restricted to *L-ensembles* (Kulesza & Taskar, 2012), which have the interesting property:

Remark 11.1. Let $X_1, X_2, ..., X_n$ be binary random variables. For any $Y \subseteq [n]$, the marginal probability is given by: $\Pr(X_i = 1 \mid i \in Y) = \det(L + I_{\overline{Y}})$ where L is the *L-ensemble* matrix of the DPP, and $I_{\overline{Y}}$ denotes the diagonal matrix with all entries indexed by elements of Y as 0, and the rest as 1.

As a polynomial, a DPP computes a polynomial $\det(L+\mathbf{X})$, where $\mathbf{X}$ is a diagonal matrix with variables on the diagonal.

PGCs were designed by Zhang et al. (2021) to subsume decomposable PCs and DPPs. It is natural to ask whether PGCs strictly subsume DPPs. The obvious answer is "yes", since DPPs can only model negative dependencies. However, what happens if we allow simple pre- or postprocessing? We prove that this question will be hard to answer:

Theorem 11.2 (Formulas as DPPs)**.** *Any arithmetic formula can be represented as an affine projection of a* DPP.

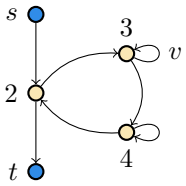
An arithmetic formula is an arithmetic circuit whose underlying structure is a tree. An affine projection is a mapping that maps the variables to affine linear forms in (a subset of) the variables.

The interpretation of the Theorem 11.2 is the following. Assume there is a PGC that we cannot write as a projection of a DPP. Then, since every arithmetic formula is a projection of a DPP, we found an arithmetic circuit (the PGC) that cannot be written as an arithmetic formula. This problem has been open in algebraic complexity theory for decades, see (Bürgisser et al., 1997). It is well-known that every formula is the projection of a determinant, see (Bürgisser et al., 1997). However, this is not sufficient to answer our question, since all known constructions place the variables in off-diagonal entries.

Before we start with the proof, we need some combinatorial interpretation of the determinant: The determinant of an $n \times n$-matrix $A = (a_{i,j})$ is defined by

$$\det A = \sum_{\pi \in S_n} \operatorname{sgn}(\pi) a_{1,\pi(1)} \dots a_{n,\pi(n)}.$$

One can think of A being the weighted adjacency matrix of a directed graph. A permutation π then corresponds to a cycle cover in the graph: A *cycle cover* is a collection of node-disjoint directed cycles such that each node appears in exactly one cycle. This is nothing but the cycle decomposition of the permutation. The sign can be written as $\operatorname{sgn}(\pi) = (-1)^{n+\#\text{cycles}}$. The weight $w(C)$ of a cycle cover C is the product of the weights of the edges in it multiplied with the sign. In this way, we can write the determinant as the sum of the weights of all cycle covers. If G is the directed graph corresponding to the matrix A, we denote this sum by $w(G)$. Thus $\det A = w(G)$.



$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & v & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 5. The graph in the base case and the corresponding adjacency matrix. The source s and target t are drawn in blue, internal nodes are drawn in yellow. v only appears on the diagonal. Edges without label have weight 1.

Proof. (of Theorem 11.2) The proof will be by induction on the structure of the formula. For every subformula, we will create a corresponding directed graph $G = (V, E)$ with self-loops and a unique start vertex s and end vertex t. Each edge of the graph will have a nonnegative weight assigned to it. An s,t-cover of such a graph is a set of edges consisting of a directed path from s to t and directed cycles such that each nodes is either in the path or in exactly one of the cycles. The weight of such an s,t-cover is the weight of the cycle cover that we get when we add the back edge (t, s) with weight 1.

For a formula computing a polynomial $p(x_1, \dots, x_n)$, we construct a graph G such that it has the following properties:

1. $w(G \setminus \{s,t\}) = 1$ and there is exactly one cover.
2. $w(G \setminus \{s\}) = w(G \setminus \{t\}) = 0$ and in both cases there are no covers.
3. $\sum_{C \in \operatorname{Cov}(G)} w(C) = p(x_1, ..., x_n)$ where $\operatorname{Cov}(G)$ are all valid s,t-coverings of G.

Furthermore the variables $x_1, \dots, x_n$ only appear on self-loops. The proof is by structural induction. The base case is when the formula is a constant or a variable and in the induction step, we combine two smaller formulas using an addition or multiplication gate.

Base Case: $p(\mathbf{x}) = v$ where v can be either a field constant or a single variable. The corresponding graph G is shown in Figure 5. Note that, since v could be a variable, it occurs only on a self loop.

1. $G' = G \setminus \{s,t\}$ leaves us with a graph G' containing only a 3-cycle of nodes $2, 3$ and 4. The only way to

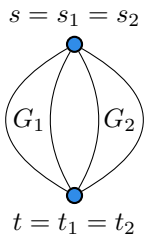


Figure 6. The construction in the case of addition. The source nodes and the target nodes of both graph are identified with each other.

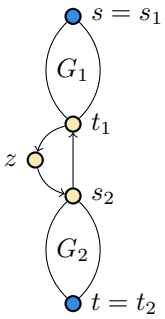


Figure 7. The construction in the case of multiplication. The source s_1 of the G_1 becomes the new source s and the target t_2 of G_2 becomes the new target t. t_1 and s_2 become internal nodes.

cover node 2 is to include it in a cycle with nodes 3 and 4, and hence the only possible cycle cover with nonzero weight, in fact weight 1, is (234).

2. Consider $G' = G \setminus \{s\}$. Clearly, t in G' is not part of any cycle. Hence, any cycle cover would need to include t with a self loop. But t has no self loop. Hence, there is no cover and $w(G') = 0$. Similarly, $w(G \setminus \{t\}) = 0$.
3. Looking at the covers C of G with an s, t-path and disjoint cycles, we notice that there is just one possible cover: the path $s, 2, t$ and two self loops on vertices 3 and 4. The sign is $(-1)^{5+3} = 1$ and the weight is v.

Inductive case 1: $p(\mathbf{x}) = p_1(\mathbf{x}) + p_2(\mathbf{x})$. Let G_i be the graph for the formula $p_i(\mathbf{x})$ with start and end vertices being s_i and t_i respectively. The corresponding graph G will be constructed by identifying s_1 with s_2 and t_1 and t_2, see Figure 6 for a schematic drawing.

1. $G' = G \setminus \{s, t\}$. Since $G'_1 = G_1 \setminus \{s, t\}$ and $G'_2 = G_2 \setminus \{s, t\}$ are not connected, the only possible cycle covers of G' are the disjoint union of cycle covers of G'_1 and G'_2. By, induction their weights will be equal to 1 and there is only one cover in each graph. Hence, $w(G') = 1$ and there is only one cover of G'.
2. Consider $G' = G \setminus \{s\}$. Now, there are two possible ways to cover G'. Either t is covered in G'_1 or in G'_2. Suppose, t is covered in G'_1. But G'_1 has no cycle cover containing t. The same is true for when t is covered in G'_2. Thus there is no cover of G'. The case for $G' = G \setminus \{t\}$ is analogous.
3. Looking at the covers C of G with an s, t-path and disjoint cycles, we notice that either G_1 or G_2 contains the s, t-path. In the first case, the s, t-cover of G will be a disjoint union of an s, t-cover of G_1 and of a cycle cover of $G_2 \setminus \{s, t\}$. There is only one cycle cover of $G_2 \setminus \{s, t\}$ and it has weight 1. Thus the sum of all such s, t-covers is $p_1(\mathbf{x})$. Similarly, when the s, t-path is contained in G_2, we get the weight $p_2(\mathbf{x})$. Hence, $w(G) = p_1(\mathbf{x}) \cdot 1 + 1 \cdot p_2(\mathbf{x}) = p(\mathbf{x})$.

Inductive case 2: $p(\mathbf{x}) = p_1(\mathbf{x}) \cdot p_2(\mathbf{x})$. Let G_i be the graph for the formula $p_i(\mathbf{x})$ with start and end vertices being s_i and t_i. The corresponding graph G will be constructed as shown in Figure 7: The start vertex is $s = s_1$ and the end vertex is $t = t_2$. t_1 and s_2 are connected via a 3-cycle.

1. $G' = G \setminus \{s, t\}$. Now, any cycle cover of G' will have z covered in the (z, s_2, t_1)-cycle. This means that t_1 will not be a part of G_1 and s_2 will not be a part of G_2. Hence $G'_1 = G_1 \setminus \{s_1, t_1\}$ and $G'_2 = G_2 \setminus \{s_2, t_2\}$. These two graph have only one cover each and their weight is 1. Since we add one cycle (z, s_2, t_1) and an odd number of nodes, the overall weight of the whole cycle cover is 1, again.
2. $G' = G \setminus \{s\}$. Like in the previous case, z can only be covered in the (z, s_2, t_1)-cycle. This means that t_1 will not be a part of G_1 and s_2 will not be a part of G_2. Let $G'_1 = G_1 \setminus \{s_1, t_1\}$ and $G'_2 = G_2 \setminus \{s_2\}$. By the induction hypothesis, G'_2 has no cycle covers and thus G' has neither. The case for $G' = G \setminus \{t\}$ is analogous.
3. Looking at the covers C of G with an s, t-path and disjoint cycles, we notice that there we can only go from s to t using z. Any such cover C will include a path $P = s = s_1 \rightsquigarrow t_1 \to z \to s_2 \rightsquigarrow t = t_2$. Let $P_i = s_i \rightsquigarrow t_i$, $i = 1, 2$. Any cycle cover formed by joining t to s with a weight 1 edge will include the path P as a cycle and further covers C_1 and C_2 of disjoint odd cycles from $G_1 \setminus P$ and $G_2 \setminus P$. This cover has one cycle less than the two corresponding covers in G_1 and G_2, since the two path P_1 and P_2 in G_1 and G_2 are merged into P. On the other hand, we also have one more node, z. Therefore, the sign of C is the product of the signs of $P_1 \cup C_1$ and $P_2 \cup C_2$ and the weight $w(C)$ is $w(P_1 \cup C_1) \cdot w(P_2 \cup C_2)$. Since we can take any combination of s, t-covers of G_1 and G_2, $w(G) = p_1(\mathbf{x}) \cdot p_2(\mathbf{x}) = p(\mathbf{x})$.

Now given a formula, we get an equivalent DPP by applying our inductive construction and adding the edge (t, s). Lastly, the L-ensemble matrix of the DPP should be positive semi-definite. However, our construction does not ensure this. But this can be easily achieved by adding a large value M to the diagonal elements of the matrix constructed and make it diagonally dominant (and hence positive definite). M can be subtracted again by the affine projection. □

In fact, we can further generalize this idea to a more powerful model, the so-called *algebraic branching programs (ABPs)*. We defer the construction to Appendix D.

Acknowledgements

We thank the anonymous referees for their helpful comments.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Arora, S. and Barak, B. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

Bläser, M. Not all strongly rayleigh distributions have small probabilistic generating circuits. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 2592–2602. PMLR, 2023. URL https://proceedings.mlr.press/v202/blaser23a.html.

Bläser, M. and Curticapean, R. The complexity of the cover polynomials for planar graphs of bounded degree. In Murlak, F. and Sankowski, P. (eds.), *Mathematical Foundations of Computer Science 2011*, pp. 96–107, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-22993-0.

Borodin, A. and Rains, E. M. Eynard–Mehta theorem, Schur process, and their Pfaffian analogs. *Journal of Statistical Physics*, 121:291–317, 2005.

Broadrick, O., Zhang, H., and den Broeck, G. V. Polynomial semantics of tractable probabilistic circuits. *arXiv 2402.09085*, 2024.

Bürgisser, P., Clausen, M., and Shokrollahi, M. A. *Algebraic complexity theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften*. Springer, 1997. ISBN 3-540-60582-7.

Cantor, D. G. and Kaltofen, E. L. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28(7):693–701, 1991. doi: 10.1007/BF01178683. URL https://doi.org/10.1007/BF01178683.

Choi, Y., Vergari, A., and Van den Broeck, G. Probabilistic circuits: A unifying framework for tractable probabilistic models. Technical report, UCLA, Oct 2020. URL http://starai.cs.ucla.edu/papers/ProbCirc20.pdf.

Dagum, P. and Luby, M. Approximating the permanent of graphs with large factors. *Theoretical Computer Science*, 102(2):283–305, 1992. ISSN 0304-3975. doi: https://doi.org/10.1016/0304-3975(92)90234-7. URL https://www.sciencedirect.com/science/article/pii/0304397592902347.

Darwiche, A. *Modeling and reasoning with Bayesian networks*. Cambridge University Press, 2009.

Harviainen, J., Ramaswamy, V. P., and Koivisto, M. On inference and learning with probabilistic generating circuits. In Evans, R. J. and Shpitser, I. (eds.), *Uncertainty in Artificial Intelligence, UAI 2023, July 31 - 4 August 2023, Pittsburgh, PA, USA*, volume 216 of *Proceedings of Machine Learning Research*, pp. 829–838. PMLR, 2023. URL https://proceedings.mlr.press/v216/harviainen23b.html.

Kisa, D., Van den Broeck, G., Choi, A., and Darwiche, A. Probabilistic sentential decision diagrams. In Baral, C., Giacomo, G. D., and Eiter, T. (eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. AAAI Press, 2014. URL http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/8005.

Koller, D. and Friedman, N. *Probabilistic Graphical Models Principles and Techniques*. MIT Press, 2009.

Kulesza, A. and Taskar, B. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286, 2012. ISSN 1935-8237. doi: 10.1561/2200000044. URL http://dx.doi.org/10.1561/2200000044.

Martens, J. and Medabalimi, V. On the expressive efficiency of sum product networks. *CoRR*, abs/1411.7717, 2014. URL http://arxiv.org/abs/1411.7717.

Meila, M. and Jordan, M. I. Learning with mixtures of trees. *J. Mach. Learn. Res.*, 1:1–48, 2000. URL http://jmlr.org/papers/v1/meila00a.html.

Mhammedi, Z., Grünwald, P., and Guedj, B. Pac-bayes un-expected bernstein inequality. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 12180–12191, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/3dea6b598a16b334a53145e78701fa87-Abstract.html.

Papadimitriou, C. *Computational Complexity*. Addison Welsey, 1994.

Poon, H. and Domingos, P. M. Sum-product networks: A new deep architecture. *CoRR*, abs/1202.3732, 2012. URL http://arxiv.org/abs/1202.3732.

Roth, D. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1):273–302, 1996. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(94)00092-1. URL https://www.sciencedirect.com/science/article/pii/0004370294000921.

Saptharishi, R. et al. A selection of lower bounds in arithmetic circuit complexity, 2021. URL https://github.com/dasarpmar/lowerbounds-survey/releases/tag/v9.0.3. Version 2021-07-27.

Shih, A., Van den Broeck, G., Beame, P., and Amarilli, A. Smoothing structured decomposable circuits. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 11412–11422, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/940392f5f32a7ade1cc201767cf83e31-Abstract.html.

Strassen, V. Vermeidung von Divisionen. *Journal für die reine und angewandte Mathematik*, 264:184–202, 1973. URL http://eudml.org/doc/151394.

Thulasidasan, S., Bhattacharya, T., Bilmes, J. A., Chennupati, G., and Mohd-Yusof, J. Combating label noise in deep learning using abstention. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6234–6243. PMLR, 2019. URL http://proceedings.mlr.press/v97/thulasidasan19a.html.

Valiant, L. G. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979. doi: 10.1137/0208032. URL https://doi.org/10.1137/0208032.

Vergari, A., Choi, Y., Peharz, R., and Van den Broeck, G. Probabilistic circuits: Representations, inference, learning and applications. *AAAI Tutorial*, 2020.

Wu, Y. and Seldin, Y. Split-kl and pac-bayes-split-kl inequalities for ternary random variables. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/49ffa271264808cf500ea528ed8ec9b3-Abstract-Conference.html.

Zhang, H., Holtzen, S., and Van den Broeck, G. On the relationship between probabilistic circuits and determinantal point processes. In Adams, R. P. and Gogate, V. (eds.), *Proceedings of the Thirty-Sixth Conference on Uncertainty in Artificial Intelligence, UAI 2020, virtual online, August 3-6, 2020*, volume 124 of *Proceedings of Machine Learning Research*, pp. 1188–1197. AUAI Press, 2020. URL http://proceedings.mlr.press/v124/zhang20c.html.

Zhang, H., Juba, B., and Van Den Broeck, G. Probabilistic generating circuits. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 12447–12457. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/zhang21i.html.

A. Proof of Theorem 7.3

Definition A.1 ($(2,3)$-regular bipartite graph)**.** Let $G = (L \cup R, E)$ be a bipartite graph such that $|L| = \frac{3}{2}n$ and $|R| = n$. G is a $(2,3)$-regular bipartite graph, if $\forall u \in L$, $\deg(u) = 2$ and $\forall v \in R$, $\deg(v) = 3$.

n has to be even in a $(2,3)$-regular graph and the left-hand size is necessarily $\frac{3}{2}n$ because of the degree constraints.

Definition A.2. Let $G = (L \cup R, E)$ be a $(2,3)$-regular bipartite graph. Let $\mathcal{M}(G)$ be the set of matchings in G. Let M be any matching in $\mathcal{M}(G)$ and $\text{unsat}_R(M)$ denote the set of unmatched R-vertices in M. Define $\text{RMatch}(G, x)$ as follows:

$$\text{RMatch}(G, x) = \sum_{M \in \mathcal{M}(G)} x^{|\text{ unsat}_R(M)|}$$

For each fixed $\lambda \in \mathbb{Q}$, the graph polynomial $\text{RMatch}(G, x)$ defines a mapping $G \mapsto \text{RMatch}(G, \lambda)$, which takes graphs to rational numbers. Bläser & Curticapean (2011, Lemma 2) show that the evaluation of the polynomial $\text{RMatch}(G, \lambda)$ is #P-hard for all $\lambda \in \mathbb{Q} \setminus \{-3, -2, -1, 0\}$.

Proof. (of Theorem 7.3) Consider any $(2,3)$-regular bipartite graph $G = (L \cup R, E)$ with $|L| = m = \frac{3}{2}n, |R| = n$. For every vertex $u_i \in L$, we define a ternary random variable U_i. For every vertex $v_j \in R$, let $N(j)$ denote the set of indices of its neighbours. In particular, $N(j,1), N(j,2), N(j,3)$ denote the indices of the three neighbours of v_j. We now define the polynomial f as

$$f(U_1, ..., U_m, \lambda) = \prod_{j=1}^{n} (\lambda + \sum_{i \in N(j)} U_i) \tag{5}$$

where λ is an arbitrary positive number, in particular it is not in $\{-3, -2, -1, 0\}$. Further, notice that every vertex $u_i \in L$ has degree $= 2$, hence the degree of any U_i in any monomial of f cannot exceed 2. We also point out that the coefficient of any monomial of f is nonnegative.

The right-hand side of (5) is a depth-2 circuit of size $s = O(n + m) = O(n)$. Furthermore, f is a polynomial of $\deg(f) = n$. Using proof ideas similar to the proof of Theorem 7.1, we transform f into a probability generating polynomial represented by a PGC by computing $f(1, 1, ..., \lambda) = (\lambda + 3)^n$. Thus, we set

$$\hat{f}(U_1, ..., U_m, \lambda) = \frac{f(U_1, ..., U_m, \lambda)}{(\lambda + 3)^n}$$

which has a circuit of size $O(n)$ and represents a probability generating polynomial.

We now selective marginalise $\hat{f}$ over $V_i = \{0, 1\}$ for each i, as per Definition 7.2. This means that we sum the coefficients of all monomials of $\hat{f}$ that are multilinear. We now claim that the multilinear monomials of $\hat{f}$ stand in one-to-one correspondance with matchings in G: Any matching in G would match all v_j for $j \in S \subseteq [n]$ for some S, and leave all v_j for $j \in [n] \setminus S$ unmatched. Let the corresponding matching be $(u_{t_1}, v_{s_1}), ..., (u_{t_k}, v_{s_k})$. Clearly, $u_{t_i} \neq u_{t_j}$ for $i \neq j$. Thus, in the polynomial f, this would correspond to the monomial $\lambda^{n-|S|} \cdot \prod_{i \in S} U_{t_i}$. Since all the exponents of U_i are in $\{0, 1\}$ for all $i \in [m]$, we see that this term would also be present. Now consider any set of edges $A \subseteq E$ which do not correspond to a matching in G. This implies that either some u_i for $i \in [m]$, or some v_j for $j \in [n]$ has been matched more than once. By definition of f, for any $j \in [n]$, every monomial of f either takes one neighbour of v_j or leaves v_j unmatched. Further, since we are looking at selective marginalisation with U_i being either 0 or 1, this implies none of the u_i's can be matched twice either. Thus there cannot be any monomial that corresponds to A. Therefore, the result of the selective marginalization is

$$\frac{\sum_{S \in \mathcal{M}(G)} \lambda^{n-|S|}}{(\lambda + 3)^n} = \frac{\text{RMatch}(G, \lambda)}{(\lambda + 3)^n}.$$

Thus, using *selective* marginalisation we can efficiently determine the RMatch polynomial for a given λ in a $(2,3)$-regular bipartite graph G. However, computing this is a #P-complete problem. Recall that the PGC for $\hat{f}$ has size $O(m)$. Thus, efficient selective marginalization over ternary PGCs would imply that #P = FP. □

B. Proofs of Proposition 9.5

Proof. (of Proposition 9.5) Let P be the polynomial computed by a given circuit C and let $\{z_{i,0}, \ldots, z_{i,d-1}\}$, $1 \leq i \leq n$, be the parts of the partition.

First we check whether each monomial of P depends on at least one variable of each part. To this aim we iterate over all i and set $z_{i,0} = \cdots = z_{i,d-1} = 0$. If the resulting polynomial is nonzero, then there is at least one monomial in P that does not contain a variable from $\{z_{i,0}, \ldots, z_{i,d-1}\}$. Testing whether a polynomial is nonzero can be done in randomized polynomial time by the Schwartz-Zippel-Lemma, see (Arora & Barak, 2009).

Second we need to check that each monomial depends on at most one variable from each part and has degree ≤ 1 in this variable. For each pair $z_{i,j}, z_{i,j'}$ with $j \neq j'$, we replace all other variables by random values (from a polynomially large set) obtaining a polynomial $P_{i,j,j'}$. Then again by the Schwartz-Zippel-Lemma, if the polynomial P has a monomial containing $z_{i,j}$ and $z_{i,j'}$ or $z_{i,j}^2$ or $z_{i,j'}^2$, then also the new polynomial $P_{i,j,j'}$ has such a monomial with high probability. Since $P_{i,j,j'}$ has only two variables, we can expand it completely and check whether is has a monomial containing $z_{i,j}$ and $z_{i,j'}$ or $z_{i,j}^2$ or $z_{i,j'}^2$.

The overall procedure is polynomial time. □

C. Proofs from Section 10

Proof. (of Proposition 10.1) The extended function $f(a, b')$ is set-multilinear. The same is true for g. A linear combination of set-multilinear polynomials on the same partition is set-multilinear. The additional $O(nd)$ term comes from the smoothening operation, where we add the uniform distribution corresponding to the variables occurring in B but not in A. Since there can be at most n such variables, each with d categories, the maximum size needed is $O(nd)$. □

Proof. (of Proposition 10.2) Since A and B are disjoint,

$$f(a)g(b) = \left(\sum_{a\in\Delta^{|A|}} \alpha_a \prod_{i\in A} z_{i,a_i}\right)\left(\sum_{b\in\Delta^{|B|}} \beta_b \prod_{j\in B} z_{j,b_j}\right)$$
$$= \sum_{a\in\Delta^{|A|}, b\in\Delta^{|B|}} \alpha_a\beta_b \prod_{i\in A} z_{i,a_i} \prod_{j\in B} z_{j,b_j}.$$

□

Proof. (of Proposition 10.4) The hierarchical composition is a mixture of 2^n many n-fold products of distributions on disjoint variables. The proposition follows immediately by repeated application of Propositions 10.1 and 10.2 as the variables involved in any g_i and g_j for $i \neq j$ are disjoint.

The size of the circuit stays small, since we can replace the input variables in the circuit for f by circuits for the corresponding $g_1, \ldots, g_n$. □

D. Additional results from Section 11

A more general class than formulas are *algebraic branching programs (ABPs)*. An ABP is a acyclic graph with a source s and a sink t. Edges are labeled with constants or variables. The weight of an s, t-path is the product of the weights of the edge in the path. The polynomial computed by an ABP is sum of the weights of all s, t-path. ABPs can be efficiently simulated by arithmetic circuits, since they can be written as an iterated matrix multiplication. See (Saptharishi, R. et al., 2021) for more information.

Generalizing the construction of the graph in Theorem 11.2, we can even show that any ABP can be represented as a DPP.

Theorem D.1 (ABPs as DPPs)**.** *An ABP of size s can be represented as a DPP of size* $\mathrm{poly}(s)$.

Proof. Recall that any ABP is a projection of an iterated matrix multiplication (IMM). Hence, we will show a reduction from the iterated matrix multiplication polynomial $\mathrm{IMM}_{n,d}$, which is the $(1, 1)$-entry of d variable matrices of size $n \times n$, to a graph whose determinant represents a DPP. Our construction uses ideas similar to the ones used in proof of Theorem 11.2. We will modify the ABP in such a way that any s, t-path in our new graph will have an odd number of vertices. Further, any other cycle in the graph would either be a self-loop or a 3-cycle. Hence, on connecting t to s via an edge, any node in the graph will always be covered by an odd cycle. Hence, the sign of any cycle cover is always positive. We show our construction for the ABP corresponding to $\mathrm{IMM}_{2,3}$ in Figure 8.

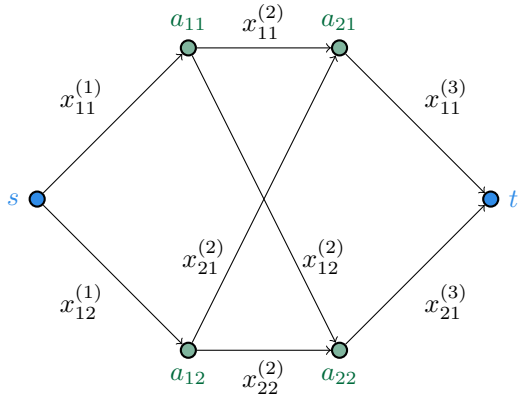


Figure 8. ABP computing $\mathrm{IMM}_{2,3}$

For an ABP corresponding to $\mathrm{IMM}_{n,d}$ we create the graph G with the following properties:

- We include all the nodes in the ABP in our graph G. We create an edge form t to s in G.
- Let $e = (u, v)$ be an edge in the ABP with weight w_e. In G, we add a node n_e corresponding to e. We join u and n_e with a directed edge of weight 1 going from u to n_e, and join n_e to v with a directed edge of weight 1. Furthermore, we create two auxiliary nodes $n_e^{(1)}$ and $n_e^{(2)}$ with self-loops of weight w_e and 1, respectively, and form a 3-cycle between $n_e, n_e^{(1)}, n_e^{(2)}$ by adding edges $(n_e, n_e^{(1)}), (n_e^{(1)}, n_e^{(2)})$ and $(n_e^{(2)}, n_e)$, each of weight 1.
- Furthermore, for every nodes u in the ABP except for the start and end nodes, we add two auxiliary nodes $u^{(1)}$ and $u^{(2)}$ with self-loops of weight 1. We finally create a 3-cycle between $u, u^{(1)}, u^{(2)}$ by adding directed edges $(u, u^{(1)}), (u^{(1)}, u^{(2)}), (u^{(2)}, u)$, each of weight 1.

The corresponding DPP graph for the ABP in Figure 8 is shown in Figure 9.

Now, let $P = s \to a_{1,k_1} \to a_{2,k_2} \to \ldots \to a_{d-1,k_{d-1}} \to t$ be any $s \rightsquigarrow t$ path in the original ABP. Then, the corresponding path P' in G will be of the form $s \to n_{(s,a_{1,k_1})} \to a_{1,k_1} \to n_{(a_{1,k_1},a_{2,k_2})} \to a_{2,k_2} \to \ldots \to a_{d-1,k_{d-1}} \to$

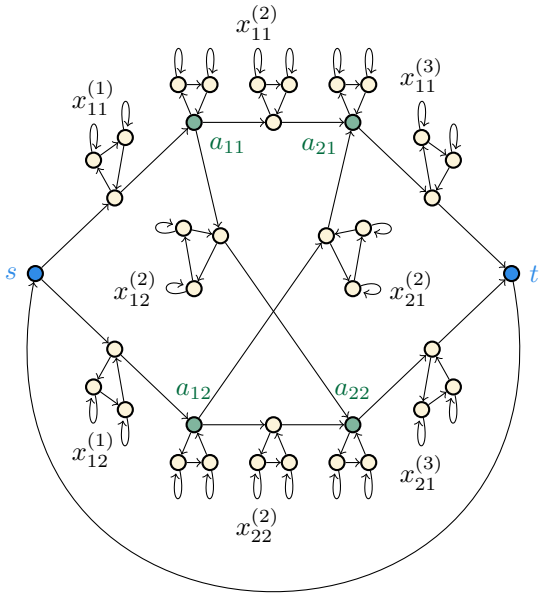


Figure 9. DPP graph G for $\mathrm{IMM}_{2,3}$

$n_{(a_{d-1,k_{d-1}},t)} \to t$. Hence, it is easy to see that any such path will have an odd number of vertices. Further, the weight of this path is 1 in G. Any cycle cover in G must cover s and t, which can only be done by a s,t-path in G along with the edge (t,s). Thus, any such cycle will have weight $= 1$, and an odd number of vertices implying a positive sign. Furthermore, for all the nodes u of the original ABP not occurring on the path P', they can only be covered by the corresponding 3-cycles $(u, u^{(1)}, u^{(2)})$, which have weight 1 and a positive sign. Moreover, for all the nodes that appear on the path P' and were present in the original ABP, the corresponding auxiliary nodes will have to be covered by self-loops of weight 1. Finally, for the nodes in P' that do not appear in the ABP (like $n_{(a_{i,k_i},a_{i+1,k_{i+1}})}$), the corresponding auxiliary nodes will have to be covered by self-loops of weight $w_{(a_{i,k_i},a_{i+1,k_{i+1}})}$ and 1, respectively. Hence, the sign of any such cycle cover will be positive, as it only has 3-cycles and self-loops, and the weight would just be

$$w_{(s,a_{1,k_1})} \cdot w_{(a_{1,k_1},a_{2,k_2})} \cdot \ldots \cdot w_{(a_{d-1,k_{d-1}},t)} = x^{(1)}_{1k_1} \cdot x^{(2)}_{k_1k_2} \cdot \ldots \cdot x^{(d)}_{k_{d-1}1}.$$

Thus,

$$\begin{aligned}\det(G) &= \sum_C \mathrm{sgn}(C) w(C) \\ &= \sum_C 1 \cdot x^{(1)}_{1k_1} \cdot x^{(2)}_{k_1k_2} \ldots x^{(d)}_{k_{d-1}1} \\ &= \sum_{s,\,t\text{-path}} x^{(1)}_{1k_1} \cdot x^{(2)}_{k_1k_2} \cdot \ldots \cdot x^{(d)}_{k_{d-1}1} \;=\; \mathrm{IMM}_{n,d},\end{aligned}$$

where the first two sums are over all cycle covers of G and the third sum is over all s,t-path in the ABP corresponding to $\mathrm{IMM}_{n,d}$. □