On Inference and Learning With Probabilistic Generating Circuits

Juha Harviainen[1] **Vaidyanathan Peruvemba Ramaswamy**[2] **Mikko Koivisto**[1]

[1]Department of Computer Science, University of Helsinki, Helsinki, Finland
[2]Faculty of Informatics, TU Wien, Vienna, Austria

Abstract

Probabilistic generating circuits (PGCs) are economical representations of multivariate probability generating polynomials (PGPs). They unify and extend decomposable probabilistic circuits and determinantal point processes, admitting tractable computation of marginal probabilities. However, the need for addition and multiplication of high-degree polynomials incurs a significant additional factor in the complexity of inference. Here, we give a new inference algorithm that eliminates this extra factor. Specifically, we show that it suffices to keep track of the highest degree coefficients of the computed polynomials, rendering the algorithm linear in the circuit size. In addition, we show that determinant-based circuits need not be expanded to division-free circuits, but can be handled by division-based fast algorithms. While these advances enhance the appeal of PGCs, we also discover an obstacle to learning them from data: it is NP-hard to recognize whether a given PGC encodes a PGP. We discuss the implications of our ambivalent findings and sketch a method, in which learning is restricted to PGCs that are composed of moderate-size subcircuits.

1 INTRODUCTION

Much recent research on probabilistic modeling has focused on the computational tractability of inference. A class of inference queries—such as marginal probabilities or most probable explanations—is said to be *tractable* in a model family if any query can be computed in time polynomial in the model size. For this concept to be useful, one would require succinct yet expressive models.

To this end, two significant model families supporting tractable marginal probability queries are (decomposable) *probabilistic circuits* (PCs) [Choi et al., 2020] and *determinantal point processes* (DPPs) [Borodin and Rains, 2005, Kulesza and Taskar, 2012]. PCs unify a range of models composed of conditional independences and mixture distributions, including arithmetic circuits [Darwiche, 2003], sum-product networks [Peharz et al., 2019, Poon and Domingos, 2011], probabilistic sentential decision diagrams [Kisa et al., 2014], and cutset networks [Rahman and Gogate, 2016]. DPPs, on the other hand, represent global negative correlations among the variables through the determinant of a kernel matrix. The two families are distinct: there are PCs whose representations as DPPs are exponentially larger, and vice versa [Zhang et al., 2020]. Another recent model family is the mixture of all trees that supports tractable likelihood computation and estimation of marginals [Selvam et al., 2023].

Both decomposable PCs and DPPs can be succinctly represented by *probabilistic generating circuits* (PGCs) [Zhang et al., 2021]. A PGC encodes a probability distribution into a *probability generating polynomial* (PGP) using a circuit that consists of four types of nodes: sum nodes, product nodes, indeterminates, and constants. The key difference to PCs is that the sum nodes in PGCs can also take negative weights. This significantly boosts the expressiveness of the model family: a circuit with negation can be exponentially smaller than a monotone circuit representing the same polynomial [Valiant, 1980]. The ability to economically represent DPPs is (just) one remarkable incarnation of this power of PGCs.

Marginal probabilities remain tractable in PGCs: they can be computed by a single pass over the circuit, using arithmetic over univariate polynomials [Zhang et al., 2021, Thm 1]. However, since the degree of the polynomials can be as large as the number of variables—say, hundreds or thousands in practice—the computational overhead can be significant in comparison to the highly efficient inference algorithms available for decomposable PCs [Peharz et al., 2020]. While the expensive polynomial arithmetic can be avoided in special, somewhat basic PGC architectures (called SimplePGCs) [Zhang et al., 2021], slow inference hampers the deploy-

Proceedings of the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023), PMLR 216:829–838.

ment of more general PGCs.

As the first contribution of this paper, we give a new inference algorithm. Our algorithm is linear in the circuit size, thus removing the extra factor due to polynomial arithmetic. While this complexity is the best possible for PGCs given as explicit circuits, one might hope for even faster algorithms if the circuit admits a smaller, implicit representation. The determinant of an $n \times n$ matrix is a prime example: it can be evaluated in time $O(n^3)$ using division, whereas division-free circuits, practical for moderate n, have size $\Omega(n^4)$; see, e.g., Bird's [2011] algorithm.[1] As a second contribution, we show that PGCs that involve determinants, indeed, need not be compiled into explicit division-free circuits—we can employ the fastest division-based algorithms!

The latter half of this paper investigates the task of learning a PGC from data. We consider the setting in which the structure of the circuit is given but model parameters—the weights of the addition nodes—are to be fitted to the data. Here a standard method would be to (locally) maximize a penalized likelihood function using (stochastic) gradient descent; implementing this method is straightforward [Zhang et al., 2021, Sect. 5] when all points in the "natural" parameter space are feasible solutions, i.e., the PGC encodes a PGP. Unfortunately, allowing arbitrary negative weights turns out to jeopardize that method: as we will show, not only there exist infeasible PGCs, but recognizing whether a given PGC is feasible is NP-hard. As a remedy, we end our paper by sketching a framework for learning restricted classes of PGCs: we argue that optimization or sampling in the parameter space can be made computationally feasible for PGCs that are composed of moderate-size subcircuits using the sum, product, and hierarchical composition operations [Zhang et al., 2021, Prop. 2 and 3].

In this paper, we restrict ourselves to analytic, theoretical considerations. While we can expect our inference algorithms, no doubt, bring orders-of-magnitude speedups in practice, there remain numerous intriguing implementation issues related to, e.g., numerical stability and good use of modern hardware; these are best addressed by devoted engineering research (cf. Peharz et al. [2020]). Likewise, we expect that implementing and experimenting with the sketched approach to learn PGCs will require a significant dedicated effort, which is beyond the present work.

2 PRELIMINARIES

This section reviews the main concepts of PGCs, following Zhang et al. [2021].

[1]The asymptotically fastest known division-free algorithms reduce the additional factor from n to around $n^{0.3}$ [Kaltofen and Villard, 2005].

2.1 REPRESENTATION

Let $X_1, \ldots, X_n$ be binary random variables and $\Pr(\cdot)$ a probability distribution over them. Then, the *probability generating polynomial* (PGP) of the distribution with indeterminates $z_1, \ldots, z_n$ is

$$g(z_1, \ldots, z_n) \coloneqq \sum_{S \subseteq \{1,\ldots,n\}} c_S z^S,$$

where $z^S = \prod_{i \in S} z_i$ and

$$c_S = \Pr\left(\{X_i = 1\}_{i \in S}, \{X_i = 0\}_{i \notin S}\right).$$

Probability generating circuits (PGCs) are one way for representing these polynomials. They are directed acyclic graphs (DAGs) with edges oriented towards a single sink node, and each node has four options for its type:

- $(+)$: Sum of two real-weighted inputs;
- $(\times)$: Product of two inputs;
- (c): Constant real value c;
- (z_i): Indeterminate z_i.

The *scope* of a node in a PGC is the set of indeterminates on which the node depends (i.e., the descendants in the graph). The *scope of a PGC* is the scope of its sink node. We may refer to a scope by the index set of the indeterminates. The *size* of a PGC is the number of edges in the graph.

As an example, Figure 1 contains the joint probability table and one possible PGC for the generating polynomial

$$\begin{aligned} &0.15z_1z_2z_3 + 0.025z_2z_3 + 0.15z_1z_3 \\ &\quad + 0.025z_3 + 0.25z_1z_2 + 0.05z_2 + 0.3z_1 + 0.05 \end{aligned} \tag{1}$$

that stems from rewriting the polynomial in the form

$$0.025(1 + 6z_1)(1 + z_2)(2 + z_3) - 0.05z_1z_2\,.$$

2.2 COMPOSITION

Let f_k be a PGP with scope S_k, for $k = 1, \ldots, \ell$. We obtain a new PGP with scope $S_1 \cup \cdots \cup S_\ell$ by any of the following composition operations [Zhang et al., 2021]:

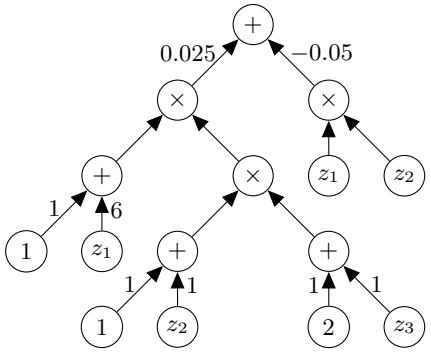
- *Sum:* $w_1 f_1 + \cdots + w_\ell f_\ell$, where the weights w_k are non-negative and sum up to 1.
- *Product:* $f_1 \cdots f_\ell$ for pairwise disjoint scopes S_k.
- *Hierarchical:* $h(f_1, \ldots, f_\ell)$ for pairwise disjoint scopes S_k and some ℓ-variate PGP h.

As an example of a hierarchical composition, consider the generating polynomial

$$h(y_1, \ldots, y_\ell) = \det\left(I + L\,\mathrm{diag}(y_1, \ldots, y_\ell)\right). \tag{2}$$

X_1	X_2	X_3	$\Pr(\cdot)$
0	0	0	0.050
1	0	0	0.300
0	1	0	0.050
1	1	0	0.250
0	0	1	0.025
1	0	1	0.150
0	1	1	0.025
1	1	1	0.150

(a) Joint probability table



(b) One possible PGC

Figure 1: The joint probability table and one possible PGC for the polynomial of Eq. (1).

of a DPP with kernel L. Now, let $f_1, \ldots, f_\ell$ be PGPs with disjoint scopes that partition the set of indeterminates $z_1, \ldots, z_n$. By the substitutions $y_k := f_k$ we obtain an n-variate PGP. Referring to the representations of the involved polynomials as circuits, Zhang et al. [2021] call the compound model a *determinantal PGC* (DetPGC).

2.3 INFERENCE

PGCs support polynomial-time marginal probability queries. The idea is to extract a particular coefficient of a univariate polynomial obtained by assigning appropriate values to the indeterminates of the circuit. For a univariate polynomial $p(t) = b_0 + b_1 t + \cdots + b_n t^n$, we write $\operatorname{coef}_d p(t)$ for the coefficient b_d of the term t^d.

Lemma 1 (Zhang et al. [2021])**.** *Let $\Pr(\cdot)$ be a probability distribution with a PGP $g(z_1, \ldots, z_n)$. For disjoint index sets $A, B \subseteq \{1, \ldots, n\}$, define a univariate polynomial in an indeterminate t by*

$$g_{A,B}(t) := g\big(\{z_i = t\}_{i\in A}, \{z_i = 0\}_{i\in B}, \{z_i = 1\}_{i\notin A\cup B}\big)\,.$$

Then, the marginal probabilities are given by

$$\Pr\big(\{X_i = 1\}_{i\in A}, \{X_i = 0\}_{i\in B}\big) = \operatorname{coef}_{|A|} g_{A,B}(t)\,.$$

We can compute the marginal probability for a given query (A, B) by a bottom-up pass through the PGC: first we set each indeterminate (in source nodes) to t, 0, or 1; then in sum and product nodes, respectively, we add and multiply the two polynomials represented by the child subcircuits. Since two degree-n polynomials can be added in time $O(n)$ and multiplied in time $O(n \log n \log\log n)$ [Cantor and Kaltofen, 1991, Schönhage and Strassen, 1971], the polynomial $g_{A,B}(t)$ can be computed in time $O(mn \log n \log\log n)$ in a circuit of size m. Note that any terms of degree higher than n can be safely ignored as they get cancelled out eventually by the definition of a PGP.

3 FASTER INFERENCE

We present three improved algorithms for answering marginal probability queries (A, B) in PGCs. Our algorithms make use of Lemma 1, however avoiding the per-node arithmetic on univariate polynomials.

Our first algorithm is based point evaluations and applies to arbitrary PGCs. Evaluating a PGC of size m in a given point takes time $O(m)$. Since, the corresponding univariate polynomial (after the substitution) is of degree at most $|A| \le n$, it is uniquely determined by $n+1$ evaluations over different values of the variable t.

Theorem 2. *Any marginal probability in a PGC of size m and n variables can be computed in time $O(mn)$.*

Proof. Evaluate the circuit at the points $t = 0, 1, \ldots, |A|$. This takes $O(mn)$ time. Finally, apply fast polynomial interpolation [Horowitz, 1972] in time $O(n \log^2 n)$. Since the circuit mentions each indeterminate at least once, we have $m \ge n$. The asymptotic bound follows. □

Remark 1. *The algorithm only needs a way to evaluate the circuit in a given point—the evaluation method need not restrict its operations to additions and multiplications. Supposing one evaluation takes time T, any marginal probabilty can be computed in time $O(Tn)$.*

3.1 LINEAR-TIME INFERENCE

We next further expedite the computations by a factor of n, yielding a truly linear running time $O(m)$. Our algorithm requires that the PGC is decomposable:

Definition 1. A product node in a PGC is *decomposable* if its children have disjoint scopes. A PGC is *decomposable* if all its product nodes are decomposable.

Decomposable PGCs are a significant subclass of PGCs. All example PGCs considered in this and previous work Zhang

Algorithm 1: Fast inference in decomposable PGCs

Input : A PGC with a sink node s after the substitution.
Output : A pair (v, d).
if s has children **then**
  Call Algorithm 1 with the children as sink nodes;
  Save outputs as (v_1, d_1) and (v_2, d_2);
  if s is a product node **then**
    return $(v_1, d_1) \times (v_2, d_2)$;
  else
    return $w_1 \cdot (v_1, d_1) + w_2 \cdot (v_2, d_2)$ with w_1, w_2 being the corresponding weights for addition;
if s has value c **then**
  return $(c, 0)$;
return $(1, 1)$;

et al. [2021] are decomposable. Note that the composition operations (Section 2.2) preserve decomposability.

The key observation is that for every node in the circuit, the degree of the corresponding univariate polynomial is at most $|A|$, as the scopes are disjoint.[2] This observation allows us to only track the coefficients of the highest degree terms (with a slight abuse of terminology allowing zero-valued coefficients—we consider $t^2 - t^2$ to have degree 2 in a sense, with a coefficient 0).

The algorithm operates on a semiring $\langle \mathbb{R} \times (\mathbb{N} \cup \{0\}), +, \times \rangle$ where the elements (v, d) describe the coefficient v of the highest degree term t^d that has been seen. Thus, the operators are defined as follows: Let (v_1, d_1) and (v_2, d_2) be elements of this semiring. Then,

$$(v_1, d_1) + (v_2, d_2) \coloneqq \begin{cases} (v_1, d_1), & \text{if } d_1 > d_2, \\ (v_2, d_2), & \text{if } d_1 < d_2, \\ (v_1 + v_2, d_1), & \text{if } d_1 = d_2 \end{cases}$$

and

$$(v_1, d_1) \times (v_2, d_2) \coloneqq (v_1 v_2, d_1 + d_2)\,.$$

We also define scalar multiplication by $w \in \mathbb{R}$ for the weighted addition: $w \cdot (v, d) \coloneqq (wv, d)$.

Similar structures have been used before, e.g., for counting optimal variable assignments in graphical models [Marinescu and Dechter, 2019] and with infinitesimal numbers in probabilistic programming [Martires et al., 2023].

The algorithm proceeds by replacing t by $(1, 1)$ and any constant c by $(c, 0)$. Then, the sum and product nodes are evaluated using the operations of the semiring as described in Algorithm 1. Clearly, the whole process takes $O(m)$ time.

We claim that the desired coefficient is obtained from the output (v, d) by the function

$$\varphi_{|A|}(v, d) \coloneqq \begin{cases} v, & \text{if } d = |A|, \\ 0, & \text{otherwise.} \end{cases}$$

The results are proven formally in the following theorem:

Theorem 3. *Any marginal probability in a decomposable PGC of size m can be computed in time $O(m)$.*

Proof. We prove that Algorithm 1 computes the marginal probability correctly by induction on the size of the circuit. Consider the four types the sink node can have:

- Constant c: The polynomial is c and the corresponding semiring element $(c, 0)$. In both cases the output will be c if and only if $|A| = 0$ and 0 otherwise.
- Indeterminate z_i: Similar to above with semiring element $(1, 1)$ and $|A| = 1$.
- Addition: Let p and q be the univariate polynomials that are the inputs of the node and w_1 and w_2 the corresponding weights for addition. The equality

  $$\operatorname{coef}_{|A|}(w_1 \cdot p + w_2 \cdot q) = w_1 \cdot \operatorname{coef}_{|A|} p + w_2 \cdot \operatorname{coef}_{|A|} q$$

  implies that it suffices to compute the sum of the coefficients from two smaller circuits. A straightforward case analysis for the outputs of the subcircuits proves the result.
- Multiplication: Similar to addition, let p and q be the input polynomials and (v_1, d_1) and (v_2, d_2) the outputs for the subcircuits, respectively. Additionally, let A_p be the set of indeterminates with $z_i = t$ in the subcircuit of p, and A_q this set for q. The circuit is decomposable, and thus A_p and A_q are disjoint. If $A_p \cup A_q = A$, then

  $$\operatorname{coef}_{|A|}(p \cdot q) = (\operatorname{coef}_{|A_p|} p)(\operatorname{coef}_{|A_q|} q)\,,$$

  and by induction,

  $$\begin{aligned}(\operatorname{coef}_{|A_p|} p)(\operatorname{coef}_{|A_q|} q) &= \varphi_{|A_p|}(v_1, d_1) \cdot \varphi_{|A_q|}(v_2, d_2) \\ &= \varphi_{|A_p|+|A_q|}(v_1 v_2, d_1 + d_2) \\ &= \varphi_{|A_p \cup A_q|}(v_1 v_2, d_1 + d_2) \\ &= \varphi_{|A|}(v_1 v_2, d_1 + d_2)\,.\end{aligned}$$

  Otherwise

  $$\operatorname{coef}_{|A|}(p \cdot q) = \varphi_{|A|}(v_1 v_2, d_1 + d_2) = 0$$

  because $d_1 + d_2 < |A|$, completing the proof.

□

To see why the linear-time algorithm can fail in a non-decomposable PGC, consider the PGP

$$\tfrac{1}{3}(z_1 z_2 + z_1 z_3 + z_2 z_3) = \tfrac{1}{3}\big((z_1 + z_2)(z_1 + z_3) - z_1^2\big)\,.$$

[2] A non-decomposable PGC may contain a product node that corresponds to a multivariate polynomial where some indeterminate has degree larger than 1. Such terms cancel out at the end.

Assigning $z_1 = t$ and $z_2 = z_3 = 1$ yields a term $z_1^2 = t^2$ on the right-hand side. This disrupts the computation because z_1^2 is always cancelled out when the formula is expanded, but the algorithm still finds a semiring element $(0, 2)$.

However, it is possible to modify the algorithm to allow overlap between the scopes. Let (v, d) be the output from evaluating Algorithm 1 with $z_i = t$ for all i on a PGC. Then, $\mu := \max\{0, d - n\}$ measures the *degree of disjointness* of the PGC in a sense, higher values meaning more overlap. It is also an upper bound for the degree of disjointness of all its subcircuits. This suggests that maintaining $\mu + 1$ largest coefficients suffices for evaluating marginal probabilities in general PGCs in time $O(m\mu \log \mu \log\log \mu)$ with fast polynomial multiplication — nearly linear time in m for moderately small overlap. In the extreme, this reduces to the algorithm of Zhang et al. [2021] as it needs to maintain all terms of the polynomial.

3.2 INFERENCE FOR DETERMINANTAL FORMS

We next present faster inference for PGCs containing determinants. We first define the form we require the circuit to have:

Definition 2. Let $P = (p_{ij})$ be a matrix where each entry is a polynomial over indeterminates $y_1, \ldots, y_\ell$ of degree at most one, that is,

$$p_{ij} = p_{ij0} + \sum_{k \in S_{ij}} p_{ijk} y_k \,,$$

with some scopes $S_{ij} \subseteq \{1, \ldots, \ell\}$ and real coefficients p_{ijk}. We say that P is *scope-disjoint* if each indeterminate y_k appears in at most one row or column, that is, $k \in S_{ij}$ implies that $k \notin S_{rs}$ if $r \neq i$ and $s \neq j$.

Remark 2. *The degree restriction is here for convenience of exposition. Definition 2 and Algorithm 2 can be extended in a straightforward manner to matrices where the entries are multilinear polynomials, the running time scaling with the number of terms in the polynomials.*

For example, in the DPP representation in Eq. (2) the matrix $I + L\,\mathrm{diag}(y_1, \ldots, y_\ell)$ with a kernel $L = (l_{ij})$ is scope-disjoint, since its diagonal and off-diagonal entries are, respectively, of the form $1 + l_{ii} y_i$ and $l_{ij} y_j$.

A more complex example is provided by the matrix

$$\frac{1}{13} \begin{pmatrix} y_1 + 2y_2 & y_1 & y_1 & -y_1 \\ 1 + y_2 & -y_3 & y_5 & 0 \\ y_2 & 1 & 2 & y_4 \\ -y_2 & 0 & y_5 & 0 \end{pmatrix},$$

demonstrating that scope-disjoint matrices are more expressive than those of DPPs: Here, the probability that $y_k = 0$ for all k is zero, whereas for DPPs this has to be nonzero. (Verifying that the determinant of the matrix is a PGP is left as an exercise to the reader.)

More generally, the properties of a scope-disjoint matrix guarantee that its determinant is a multilinear polynomial, thus holding potential to encode a probability distribution.

Definition 3. A PGC is a *determinantal PGC* if it is a hierarchical composition $h(f_1, \ldots, f_\ell)$, where $h = \det P$ with a scope-disjoint square matrix P over ℓ indeterminates and $f_1, \ldots, f_\ell$ are PGPs with disjoint scopes over $z_1, \ldots, z_n$. We call P the *parent* matrix and each f_k a *child* PGP.

Consider the problem of computing the marginal probability in a determinantal PGC for a given query (A, B). So far, we have considered two different approaches. One is to apply Theorem 2 and Remark 1: first evaluate the PGC in $O(n)$ points, and then perform one univariate polynomial interpolation at the end. The other approach is to represent the determinant as an explicit PGC which incurs overhead due to the size of division-free circuits. Could we evaluate the determinant with fast algorithms without the need for multiple evalutations, getting the best of both worlds?

We achieve this by an approach similar to the fast inference of PGCs: we only track the highest-degree terms. However, the fast algorithms for the determinant are designed for real matrices, so we need to find a way of incorporating the information about the degrees of the terms. Note that because the matrix is scope-disjoint, each f_k must appear in P in at most one row (column). Thus, we can safely ignore entries on that row (column) that do not contain f_k as a term, as degree $|A|$ cannot be achieved without picking an entry containing f_k from that row (column). It turns out this is sufficient for the fast computation of marginal probabilities. This is more formally represented in Algorithm 2 and proven in Theorem 4.

As an example of the execution of Algorithm 2, consider the earlier matrix where each y_k is substituted by $f_k(z_k) = z_k$. Then, the transformation of P into R with $A = \{1, 2, 3\}$ and $B = \{4\}$ goes as follows:

$$\begin{pmatrix} z_1 + 2z_2 & z_1 & z_1 & -z_1 \\ 1 + z_2 & -z_3 & z_5 & 0 \\ z_2 & 1 & 2 & z_4 \\ -z_2 & 0 & z_5 & 0 \end{pmatrix} \mapsto \begin{pmatrix} 0 & 0 & 1 & -1 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ -1 & 0 & 1 & 0 \end{pmatrix}$$

Now, the only permutations with a nonzero value are $(4, 2, 1, 3)$ and $(4, 2, 3, 1)$, encoding terms $z_1 z_2 z_3 z_5$ and $2 z_1 z_2 z_3$ in the determinant of P.

Theorem 4. *Any marginal probability in a determinantal PGC can be computed in time $O(m^3 + m^2\ell + T\ell)$, assuming the parent matrix is of size $m \times m$ and in each child PGP any marginal probability can be computed in time T.*

Proof. Consider Algorithm 2. All for-loops can clearly be

Algorithm 2: Fast inference in determinantal PGCs

Input : A determinantal PGC over $X_1, \ldots, X_n$, with an $m \times m$ parent matrix (p_{ij}) and child PGPs $f_1, \ldots, f_\ell$; disjoint subsets $A, B \subseteq \{1, \ldots, n\}$.
Output : The probability of $\{X_i = 1\}_{i \in A}, \{X_i = 0\}_{i \in B}$.

```
for k = 1, ..., ℓ do
    A_k ← the intersection of A and the scope of f_k;
    B_k ← the intersection of B and the scope of f_k;
    c_k ← the marginal probability for (A_k, B_k) in f_k;
for i = 1, ..., m do
    row_i ← {k : f_k appears only on row i and A_k ≠ ∅};
for j = 1, ..., m do
    col_j ← {k : f_k appears only on column j and A_k ≠ ∅};
c_0 ← 1;
for i = 1, ..., m do
    for j = 1, ..., m do
        r_ij ← 0;
        for each k ∈ {0} ∪ S_ij do
            if row_i, col_j ⊆ {k} then
                r_ij ← r_ij + p_ijk c_k;
return the determinant of the matrix (r_ij)
```

implemented to run in the claimed time. It remains to show that the algorithm is correct.

By Lemma 1, it is sufficient to show that for the matrix $R = (r_{ij})$ we have

$$\det R = \operatorname{coef}_{|A|} \det P(t)\,, \tag{3}$$

where the entries of $P(t)$ are given by

$$p_{ij}(t) = p_{ij}\big(\{z_i = t\}_{i \in A}, \{z_i = 0\}_{i \in B}, \{z_i = 1\}_{i \notin A \cup B}\big)\,;$$

here we view each entry of P as a multivariate polynomial in the indeterminates $z_1, \ldots, z_n$. For convenience, we omit showing the dependence on the sets A and B in the notation.

Recall that the determinant can be written as the sum

$$\det R = \sum_{\sigma} \operatorname{sgn}(\sigma) \prod_{i=1}^{m} r_{i,\sigma(i)}$$

over the permutations σ on $\{1, \ldots, m\}$, where $\operatorname{sgn}(\sigma)$ is the sign of the permutation. Fix an arbitrary permutation σ. It suffices to show that

$$\prod_{i=1}^{m} r_{i,\sigma(i)} = \operatorname{coef}_{|A|} \prod_{i=1}^{m} p_{i,\sigma(i)}(t)\,. \tag{4}$$

Let $A_{ij} \coloneqq \bigcup_{k \in S_{ij}} A_k$. Say that a permutation σ *partitions* A if $|A| = \sum_{i=1}^{m} |A_{i,\sigma(i)}|$.

Assume first that σ does not partition A. Then the right-hand side of Eq. (4) is zero, as the degree of the polynomial cannot reach $|A|$. Accordingly, we can select an f_k such that $A_k \neq \emptyset$ and $k \notin S_{i,\sigma(i)}$ for all i. Because P is scope-disjoint, we can further select an i such that f_k appears only on row i or column $j \coloneqq \sigma(i)$. Now, as $k \notin S_{ij}$ but $k \in \mathit{row}_i$ or $k \in \mathit{col}_j$, the algorithm assigns r_{ij} to zero, thus rendering the left-hand side of Eq. (4) zero, as desired.

For the rest of the proof, assume that σ partitions A. Since P is scope-disjoint, we care only about the highest-degree terms in the factors of the product on the right-hand side of Eq. (4). Therefore, it can be rewritten as

$$\prod_{i=1}^{m} \operatorname{coef}_{|A_{i,\sigma(i)}|} p_{i,\sigma(i)}(t).$$

Consider a single factor of this product. By Lemma 1 and the definition of p_{ij}, the coefficient corresponds to a weighted sum over the precomputed marginal probabilities c_k:

$$\operatorname{coef}_{|A_{ij}|} p_{ij}(t) = \sum_{\substack{k \in \{0\} \cup S_{ij} \\ A_k = A_{ij}}} p_{ijk} \cdot c_k,$$

where $A_0 \coloneqq \emptyset$.

We separate two cases: If $A_{ij} = \emptyset$, then all indeterminates appear only in other rows and columns. Thus, $\mathit{row}_i = \mathit{col}_j = \emptyset$ and

$$r_{ij} = \sum_{k \in \{0\} \cup S_{ij}} p_{ijk} \cdot c_k\,,$$

as desired.

Otherwise $A_{ij} \neq \emptyset$. Then we can have $A_k = A_{ij}$ for at most one $k \in S_{ij}$, since the scopes of f_k are disjoint.

Now, if such a k exists, then

$$\operatorname{coef}_{|A_{ij}|} p_{ij}(t) = p_{ijk} \cdot c_k\,.$$

But because we assumed that σ partitions A, and because P is scope-disjoint, the indeterminates in $A \setminus A_{ij}$ must appear in other columns and rows. Thus, $\mathit{row}_i, \mathit{col}_j \subseteq \{k\}$ and, thereby, the algorithm correctly assigns r_{ij} to $p_{ijk} \cdot c_k$.

Otherwise, no such k exists and there is an $l \in S_{ij}$ with a nonempty set $A_l \subseteq A_{ij} \setminus A_k$. Now, $l \in \mathit{row}_i$ or $l \in \mathit{col}_j$, meaning that

$$r_{ij} = 0 = \operatorname{coef}_{|A_{ij}|} p_{ij}(t),$$

concluding the proof. □

4 LEARNING

Let us now turn to the task of finding parameter values for a given PGC structure so as to (locally) maximize a penalized likelihood function. Or, equally, consider the task of drawing a sample of parameter values (approximately) from a posterior distribution obtained by multiplying the likelihood by a prior. If the parameters were subject to only simple

constraints, such as nonnegativity, then one could apply standard gradient-based optimization methods or Markov chain Monte Carlo methods, such as those implemented in Adam [Kingma and Ba, 2015] and Stan [Carpenter et al., 2017]. Unfortunately, in our case the constraints are complicated: as we will show next, checking whether a given PGC encodes a probability distribution is NP-hard. Motivated by this obstacle, we end this section with a sketch of how to learn PGCs that composed of moderate-size subcircuits.

4.1 RECOGNIZING FEASIBLE PGCS IS HARD

We call a PGC *feasible* if it encodes a PGP, that is, all the coefficients in the generating polynomial are nonnegative. Thus, for recognizing that a given PGC is *infeasible*, it would be sufficient to exhibit a negative coefficient. While a single coefficient can be efficiently extracted using inference algorithms, it turns out to be hard to figure out, which of the exponentially many coefficients one should compute:

Theorem 5. *Recognizing whether a given PGC is infeasible is an NP-complete problem.*

Proof. We prove the statement with a reduction from k-CNF-SAT by showing that efficient recognition would yield an efficient algorithm for k-CNF-SAT. The main idea is to encode the SAT instance into a PGC that is infeasible exactly when it has a solution. We will ignore normalizing the distribution here: it is easy to do by dividing by the value obtained from assigning each indeterminate to 1. See Figure 2 for a visualization of the construction given in the following paragraphs.

Essentially, we are building a probability mass function that expresses the number of unsatisfied clauses for each truth assignment. We start by associating each truth assignment

$$\{x_i = 1\}_{i \in S}, \{x_j = 0\}_{j \notin S}$$

with the monomial $\prod_{i \in S} z_i = z^S$. A truth assignment is unsatisfying if any of the clauses is unsatisfied. This leads to the following idea: Add weight -1 to all assignments and then, for each clause, increment the weight of every assignment that does not satisfy the clause. Thus, only a satisfying assignment has weight -1.

Initializing all assignments with -1 corresponds to the product

$$-(1 + z_1)(1 + z_2) \cdots (1 + z_n)\,. \tag{5}$$

Next, encode the clauses into polynomials such that the set of terms in it matches the set of unsatisfying truth assignments for it: A positive literal x_i has to be false (1) and a negative literal $\bar{x}_j$ true (z_j). For example, clause $x_1 \vee x_2 \vee \bar{x}_3$ corresponds to

$$1 \cdot 1 \cdot z_3 \cdot (1 + z_4)(1 + z_5) \cdots (1 + z_n)\,.$$

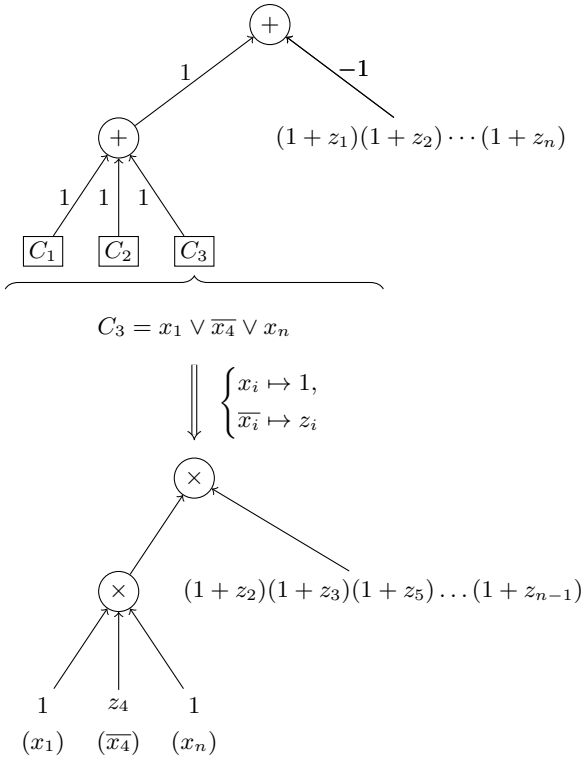


Figure 2: An example of our reduction from a k-CNF-SAT instance $C_1 \wedge C_2 \wedge C_3$ to recognizing whether a given PGC is infeasible.

Now, if any clause is unsatisfied for a truth assignment, the corresponding term is added at least once to Eq. (5), and thus it is no longer negative. However, the term of a truth assignment satisfying all clauses remains negative after taking the sum over all clauses. If we were able to find it in polynomial time, we would have solved k-CNF-SAT efficiently. □

Remark 3. *Equivalently, recognizing whether a given PGC is feasible is a co-NP-complete problem.*

Note that we actually showed a slightly stronger result: the constructed circuit is decomposable, so recognizing such infeasible PGCs is NP-complete as well.

The hardness result in itself would leave open the possibility of finding a (worst-case) moderately exponential algorithm running, e.g., in time $O(1.23^n m)$. However, the reduction to CNF-SAT is sufficiently tight to render that difficult. Indeed, such an algorithm would give an equally fast algorithm for CNF-SAT, thus refuting the conjectured *strong exponential time hypothesis* [Impagliazzo and Paturi, 2001]:

Corollary 6. *Assuming the strong exponential time hypothesis holds, no algorithm can recognize feasible PGCs in time $O(c^n)$ for any $c < 2$.*

4.2 LEARNING COMPOUND CIRCUITS

We next sketch an approach to learn *compound PGCs* that are composed of moderate-size subcircuits. The key observation is that in such models, the check for feasibility can be done separately for each subcircuit.

Suppose we are given a PGC that is composed of some PGCs with generating polynomials $g_1, \ldots, g_s$, recursively using the sum, product, and hierarchical composition. We view each g_j as a function of its weight parameters, which we denote by θ_j. We refer to $\theta = (\theta_1, \ldots, \theta_s)$ as a *solution*. Recall that *if* every g_k is a PGP, then also the compound PGC encodes a PGP.

Consider the following template algorithm, which can be instantiated either to local optimization or to approximate posterior sampling:

1. *Initialize.* Let θ be any feasible solution; e.g., assign all weights a nonnegative value from the range $[0, 1]$.
2. *Move.* Let θ' be a candidate solution in a local neighborhood of θ; e.g., take a small step in the direction of the gradient of the likelihood.
3. *Accept or reject.* If g_j is a PGP for all j, update θ to θ'.
4. *Iterate.* Go to step 2.

In the next paragraphs we discuss each step in detail.

Initialization with nonnegative weights guarantees that the PGC encodes a PGP (up to a normalizing constant). Ideally, there would be no need to select a "good" initial solution, as the iterations would quickly move the solution to regions of high likelihood or posterior. An implicit assumption here is that such good regions can be—at least in theory—reached also from poor initial solutions. Characterizing the connectiveness of the feasible region is a question for future research.

Efficient moves can be critical for the performance of the algorithm. A simple implementation would generate a candidate solution in a random walk manner, perturbing the values in the current solution only in one or a few dimensions. While this would be extremely fast, it would also miss the opportunity to generate better candidates by computationally more demanding algorithms. Namely, the accept–reject step that follows is expected to dominate the computational complexity. It would be beneficial to generate candidates that are feasible with good probability. For gradient-based moves it would be useful if the inference algorithms support automated differentiation; it is not immediate what the status of our present algorithms is with this respect.

Deciding whether to accept or reject the proposed candidate solution is the most (or only) nontrivial step of this algorithm. Here we assume that the feasibility of the candidate was not ensured in the move step, but was left to be checked for in this step. Clearly, feasibility needs to be checked only for those subcircuits whose weights were changed; the number of affected subcircuits can be small or large depending on the type of the move. The main challenge is that deciding whether a subcircuit encodes a PGP is hard problem: by Corollary 6, in the worst case, the running time of our algorithm would scale as 2^s for a subcircuit on s variables. On the positive side, the complexity is practical when s is small, say $s < 20$. In practice, it may be harder to ensure that a candiate is feasible (no polynomial-size certificate) than verifying that the candidate is infeasible—finding fast, practical algorithms for these tasks is an intriguing research question. It is worth noting that feasiblity checks need to be faster than likelihood computations, which also can be relatively expensive if there is a large number of data points.

In the present form, the algorithm template does not specify any stopping rule. One can apply any criteria commonly used in local optimization and Markov chain simulation (to approximate sampling from a posterior). However, since each iteration is expected to be computationally expensive, the number of iterations may have to be kept relatively small in practice. This setting favors parallel schemes, such as annealed importance sampling [Neal, 2001], supposing multiple processors are available.

5 CONCLUDING REMARKS

Probabilistic generating circuits (PGCs) are a recently proposed model class that subsumes probabilistic circuits (PCs) and determinantal point processes (DPPs) [Zhang et al., 2021]. In this paper, we took a closer look at the computational complexity of inference and learning in PGCs.

We observed that in a PGC on n random variables, any marginal probability can be computed by $n + 1$ point evaluations of the circuit, followed by a single, relatively inexpensive polynomial interpolation. Compared to a previous algorithm, the saving in the asymptotic complexity is by a factor logarithmic in n. In practice, for n in hundreds or thousands, we may expect an orders-of-magnitude speedup given the simplicity of scalar arithmetic, as compared to multiplication of polynomials.

In determinantal PGCs, our inference algorithm would circumvent the requirement to represent determinants as division-free circuits, an idea suggested in a previous work. But, we gave an even faster algorithm that avoids multiple point evaluations and interpolation altogether. We showed that, in essence, it suffices to evaluate a single determinant of a matrix with real-valued entries.

Using similar ideas, we also gave a linear-time inference algorithm in what we call decomposable PGCs. This removes a factor of n from the time complexity of our evaluation–interpolation algorithm. Arguably, decomposable PGC are a significant subclass of PGCs: they are strictly more expressive than PCs and DPPs [Zhang et al., 2021, implied by

the proof of Thm 2], and the class is closed under the three composition operations.

As to the learning of PGCs from data, we pointed out a notable obstacle: it is NP-hard to recognize whether even a decomposable PGC encodes a probability distribution. On the other hand, we sketched an approach to learn PGCs composed of moderate-size subcircuits. Our inspection suggests that standard methods of (stochastic) local search may be practical, in particular if typical instances of the NP-hard recognition problem can be solved fast in practice.

Author Contributions

All authors collaborated on writing the paper. The faster inference algorithms for decomposable, determinantal, and general PGCs were devised by J. Harviainen, V. Peruvemba Ramaswamy, and M. Koivisto, respectively. J. Harviainen gave the NP-completeness proof, and the learning algorithm was outlined by M. Koivisto.

Acknowledgements

Research supported by grants from the Academy of Finland (projects 316771 and 351156) and the Austrian Science Fund (project W1255).

References

Richard S. Bird. A simple division-free algorithm for computing determinants. *Inf. Process. Lett.*, 111(21-22):1072–1074, 2011.

Alexei Borodin and Eric M. Rains. Eynard-Mehta theorem, Schur process, and their pfaffian analogs. *J. Stat. Phys.*, 121(3–4):291–317, 2005.

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

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

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. `http://starai.cs.ucla.edu/papers/ProbCirc20.pdf`, 2020. Accessed: 2023-01-30.

Adnan Darwiche. A differential approach to inference in Bayesian networks. *J. ACM*, 50(3):280–305, 2003.

Ellis Horowitz. A fast method for interpolation using preconditioning. *Inf. Process. Lett.*, 1(4):157–163, 1972.

Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

Erich Kaltofen and Gilles Villard. On the complexity of computing determinants. *Comput. Complex.*, 13(3-4): 91–130, 2005.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the Third International Conference on Learning Representations, ICLR 2015*, 2015.

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

Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. *Found. Trends Mach. Learn.*, 5(2-3):123–286, 2012.

Radu Marinescu and Rina Dechter. Counting the optimal solutions in graphical models. In *Advances in Neural Information Processing Systems 32, NeurIPS 2019*, pages 12091–12101, 2019.

Pedro Zuidberg Dos Martires, Luc De Raedt, and Angelika Kimmig. Declarative probabilistic logic programming in discrete-continuous domains. *CoRR*, abs/2302.10674, 2023.

Radford M. Neal. Annealed importance sampling. *Stat. Comput.*, 11(2):125–139, 2001.

Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Xiaoting Shao, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019*, volume 115 of *Proceedings of Machine Learning Research*, pages 334–344. AUAI Press, 2019.

Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*, volume 119 of *Proceedings of Machine Learning Research*, pages 7563–7574. PMLR, 2020.

Hoifung Poon and Pedro M. Domingos. Sum-product networks: A new deep architecture. In *Proceedings of the*

Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, UAI 2011, pages 337–346. AUAI Press, 2011.

Tahrima Rahman and Vibhav Gogate. Learning ensembles of cutset networks. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 3301–3307. AAAI Press, 2016.

Arnold Schönhage and Volker Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7(3-4):281–292, 1971.

Nikil Roashan Selvam, Honghua Zhang, and Guy Van den Broeck. Mixtures of all trees. In *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics, AISTATS 2023*, volume 206 of *Proceedings of Machine Learning Research*, pages 11043–11058. PMLR, 25–27 Apr 2023.

Leslie G. Valiant. Negation can be exponentially powerful. *Theor. Comput. Sci.*, 12:303–314, 1980.

Honghua Zhang, Steven Holtzen, and Guy Van den Broeck. On the relationship between probabilistic circuits and determinantal point processes. In *Proceedings of the Thirty-Sixth Conference on Uncertainty in Artificial Intelligence, UAI 2020*, volume 124 of *Proceedings of Machine Learning Research*, pages 1188–1197. AUAI Press, 2020.

Honghua Zhang, Brendan Juba, and Guy Van den Broeck. Probabilistic generating circuits. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021*, volume 139 of *Proceedings of Machine Learning Research*, pages 12447–12457. PMLR, 2021.