Restructuring Tractable Probabilistic Circuits

Anonymous submission

Abstract

Probabilistic circuits (PCs) serve as a unifying representation for probabilistic models that support tractable inference. Numerous applications of PCs, such as controllable text generation, depend on the ability to efficiently *multiply* two circuits. Existing multiplication algorithms require that the circuits respect the same *structure*, i.e., variable scopes decompose according to the same *vtree*. In this work, we propose and study the task of *restructuring* structured(-decomposable) PCs, that is, transforming a structured PC such that it conforms to a target vtree. We propose a generic approach for this problem and show that it leads to novel polynomial-time algorithms for multiplying circuits respecting *different* vtrees, as well as a practical depth-reduction algorithm that preserves structured decomposability. Our work opens up new avenues for tractable PC inference, suggesting the possibility of training with less restrictive PC structures while enabling efficient inference by changing their structures at inference time.

1 Introduction

Probabilistic circuits represent distributions as *computation graphs* of sums and products. A crucial aspect to the design of PCs is the *structure* of the computation graph, that is, how distributions are factorized into (conditionally) independent components. The structure of PCs affects their tractability, modeling performance and computational efficiency. In this work, we consider the problem of *restructuring* PCs: constructing a new PC that follows a target structure while representing the same distribution. We present a general algorithm for restructuring structured-decomposable circuits by considering their graphical model representations. Specifically, we leverage the graphical models to reason about conditional independencies and recursively construct a new PC conforming to the desired structure.

We then investigate two key applications of PC restructuring: circuit multiplication and depth reduction. Circuit multiplication is a fundamental operation used for various inference queries (Vergari et al. 2021), such as conditioning on logical constraints (Choi, Van den Broeck, and Darwiche 2015; Ahmed et al. 2022; Liu, Niepert, and Van den Broeck 2024; Zhang et al. 2023, 2024), computing expected predictions of classifiers (Khosravi et al. 2019) and causal backdoor adjustment (Wang and Kwiatkowska 2023). Though the problem of multiplying circuits of different structures is in general #P-hard (Vergari et al. 2021), we identify a new class of PCs, which we call *contiguous* circuits, where it is possible to multiply circuits of different structures in polynomial (or quasi-polynomial) time using our algorithm. We also consider depth reduction, a well-established theoretical tool for reducing the depth of a circuit (Valiant et al. 1983; Raz and Yehudayoff 2008). We show that our restructuring algorithm can be used to transform a structured-decomposable circuit to an equivalent log-depth circuit, with much tighter upper bounds than given by prior work.

2 Probabilistic Circuits

Notation We use uppercase to denote variables (e.g. X) and lowercase to denote values of variables (e.g. x). We use boldface to denote sets of variables/values (e.g. $\mathbf{X}, \mathbf{x}$).

Definition 2.1 (Probabilistic Circuit)**.** A probabilistic circuit (PC) $\mathscr{A} = (\mathcal{G}, \mathbf{w})$ represents a joint probability distribution over random variables $\boldsymbol{X}$ through a rooted directed acyclic (computation) graph (DAG), consisting of sum ($\oplus$), product ($\otimes$), and leaf nodes (L), parameterized by $\mathbf{w}$. Each node t represents a probability distribution $p_t(\boldsymbol{X})$, defined recursively by:

$$p_t(\boldsymbol{x}) = \begin{cases} f_t(\boldsymbol{x}) & \text{if } t \text{ is a leaf node} \\ \prod_{c \in \mathrm{ch}(t)} p_c(\boldsymbol{x}) & \text{if } t \text{ is a product node} \\ \sum_{c \in \mathrm{ch}(t)} w_{t,c} p_c(\boldsymbol{x}) & \text{if } t \text{ is a sum node} \end{cases}$$

where $f_t(\boldsymbol{x})$ is a univariate input distribution function (e.g. Gaussian, Categorical), we use $\mathrm{ch}(t)$ to denote the set of children of a node t, and $w_{t,c}$ is the non-negative weight associated with the edge (t, c) in the DAG, which satisfy the constraint that $\sum_{c \in \mathrm{ch}(t)} w_{t,c} = 1$ for every sum node t. We define the *scope* of a node t to be the variables it depends on. The function represented by a PC, denoted $p_{\mathscr{A}}(\mathbf{x})$, is the function represented by its root node; and the size of a PC, denoted $|\mathscr{A}|$, is the number of edges in its graph.

Intuitively, product nodes represent a factorization of their child distributions, while sum nodes represent a weighted mixture of their child distributions. For simplicity, in the rest of this paper we assume that sum/leaf and product nodes alternate (i.e., child of a sum is a product, and child of a product is a leaf or sum), and that each product has exactly two children. The key feature of PCs is their *tractability*, i.e., the

ability to answer queries about the distributions they represent exactly and in polynomial time. Two commonly assumed properties known as smoothness and decomposability ensure efficient marginalization:

Definition 2.2 (Smoothness and Decomposability)**.** A sum node is *smooth* if all of its children have the same scope. A product node is *decomposable* if its children have disjoint scope. A PC is smooth (resp. decomposable) if all of its sum (resp. product) nodes are smooth (resp. decomposable).

Intuitively, decomposability requires that a product node partitions its scope among its children. For many other important queries, it is useful to enforce a stronger form of decomposability, known as *structured-decomposability*, which requires that product nodes with the same scope decompose in the same way.

Definition 2.3 (Vtree)**.** A vtree V over variables $\mathbf{X}$ is a rooted binary tree, where each $X \in \boldsymbol{X}$ is associated with a unique leaf node v (we write X_v for the variable associated with node v). Each inner node v covers a set of variables $\mathbf{X}_v$, satisfying $\mathbf{X}_v = \mathbf{X}_l \cup \mathbf{X}_r$ where l, r are the children of v. We write V_v to denote the subtree rooted at v.

Definition 2.4 (Structured Decomposability)**.** A PC $\mathscr{A}$ is structured-decomposable (w.r.t. a vtree V) if every product node $t \in \mathscr{A}$ partitions its scope according to some inner vtree node $v \in V$.

The main advantage of structured decomposability is that it enables tractable multiplication of two circuits respecting the same vtree, which is a core subroutine for many applications. However, structured-decomposable circuits can be less expressive efficient in general (de Colnet and Mengel 2021).

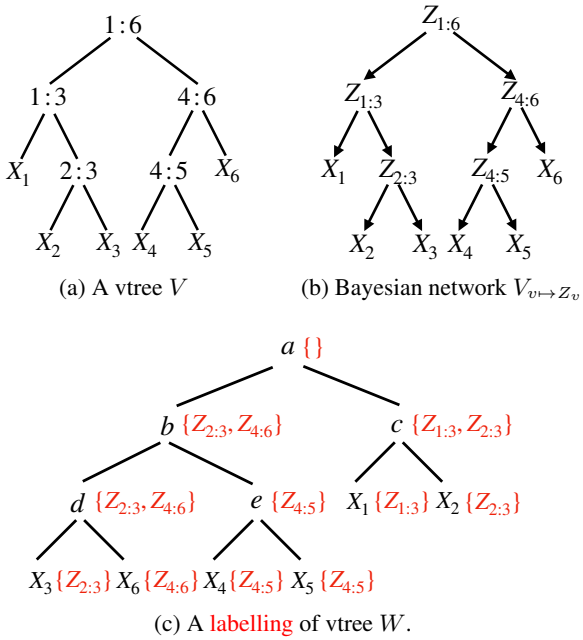
3 PC Restructuring

In this section, we describe a generic approach that restructures any structured-decomposable PC respecting a target vtree. The approach consists of three steps: (1) construct a Bayesian network representation of the PC; (2) find sets of latent variables in the Bayesian network that induce conditional independencies required by the target vtree; (3) construct a new structured PC recursively leveraging the conditional independence derived in (2).

Structured PCs as Bayesian Networks

It is known that one can efficiently compile a tree-shaped Bayesian network to an equivalent probabilistic circuit (Darwiche 2003; Poon and Domingos 2011; Dang, Vergari, and Broeck 2020; Liu and Van den Broeck 2021). It is also known how to convert a (structured) PC into a Bayesian network, but requiring either exponentially many latent variables (Zhao, Melibari, and Poupart 2015) or a compilation assumption (Butz, Oliveira, and Peharz 2020; Papantonis and Belle 2023). Here, we describe a simple construction for converting an arbitrary structured-decomposable PC to a tree-shaped Bayesian network with linearly many variables.

Let $\mathscr{A}$ be a structured PC over variables $\mathbf{X}$ respecting vtree V. Given a vtree node $v \in V$, we write $\text{prod}(v)$ to denote the set of all product nodes with scope $\mathbf{X}_v$. We define



(a) A vtree V

(b) Bayesian network $V_{v \mapsto Z_v}$

(c) A labelling of vtree W.

Figure 1: Fig. 1a shows a vtree V for some PC $\mathscr{A}$; Fig. 1b shows a Bayesian network representation $G_{\mathscr{A}}$ for $\mathscr{A}$; Fig. 1c shows a valid labelling of vtree W with respect to $G_{\mathscr{A}}$.

the *hidden state size* h of the circuit to be $\max_{v \in V} |\text{prod}(v)|$. Writing n for the number of variables, the size of the circuit is then $O(nh^2)$.[1] Given some vtree node v, let us associate each $t \in \text{prod}(v)$ with a unique index $\text{idx}(t) \in \{0, ..., |\text{prod}(v)| - 1\}$, also writing $t_{v,i}$ to refer to the product node with index i in $\text{prod}(v)$. Then we can introduce a categorical latent variable Z_v whose value corresponds to a particular product node in $\text{prod}(v)$.

Let $G_{\mathscr{A}} := V_{v \to Z_v}$ be the rooted DAG obtained by replacing all inner nodes v in vtree V with variable Z_v (cf. Fig. 1). Now, it can be shown that the PC (with latent variables) has the same distribution as a Bayesian network with graph structure $V_{v \to Z_v}$ (with conditional probability distributions given by the PC's corresponding conditional distributions). We provide full details and a proof in Appendix A.

Recursive PC Restructuring

Suppose we have a PC $\mathscr{A}$ with its Bayesian network representation $G_{\mathscr{A}}$ and vtree V, and let W be some other vtree. We now show how to construct a new PC respecting W that encodes the same distribution as $\mathscr{A}$. The rough idea is to label each vtree node $w \in W$ with a subset of latent variables $\mathbf{C}_w \subseteq G_{\mathscr{A}}$ such that $\mathbf{X}_w$ is conditionally independent from $\mathbf{X} \setminus \mathbf{X}_w$ given $\mathbf{C}_w$. To characterize such properties, we introduce *covers*:

Definition 3.1 (Cover)**.** Given a tree-shaped Bayesian network $G_{\mathscr{A}}$ as constructed in Sec. 3, we say that $\mathbf{C} \subseteq \mathbf{Z}$ *covers*

[1] The number of active sum nodes per vtree node is at most h, as each such node must have a different product node parent corresponding to the parent vtree node scope. This leads to $O(h^2)$ edges per vtree node.

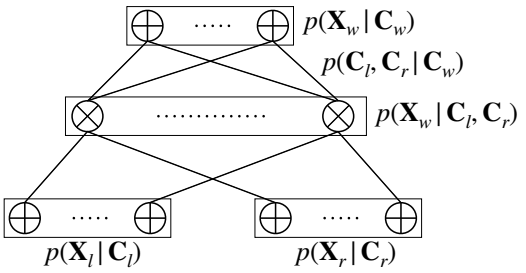


Figure 2: Recursive construction of vectors of sum nodes representing $p(\mathbf{X}_w \mid \mathbf{C}_w)$

$\mathbf{S} \subseteq \mathbf{X}$ if $\mathbf{C}$ blocks[2] all paths between $\mathbf{S}$ and $\mathbf{X} \setminus \mathbf{S}$ in $G_{\mathscr{A}}$[3].

Our goal is to recursively construct vectors of sum nodes $\oplus_i$ representing the probability distributions $p(\mathbf{X}_w | \mathbf{C}_w = i)$. Letting l and r be the children of w, we will establish a recurrence relation between $p(\mathbf{X}_w \mid \mathbf{C}_w)$, $p(\mathbf{X}_l \mid \mathbf{C}_l)$ and $p(\mathbf{X}_r \mid \mathbf{C}_r)$. This requires the vtree labels to satisfy the following properties:

Definition 3.2 (Valid Vtree Labelling)**.** Given the Bayesian network $G_{\mathscr{A}}$ and target vtree W, a valid *labelling* of W with respect to $G_{\mathscr{A}}$ associates each node $w \in W$ with a subset of latent variables $\mathbf{C}_w \subseteq G_V$ s.t.

1. $\mathbf{C}_w$ covers $\mathbf{X}_w$ in $G_{\mathscr{A}}$.
2. $\mathbf{C}_l$ blocks all paths between $\mathbf{X}_l$ and $\mathbf{C}_r \cup \mathbf{C}_w$.
3. $\mathbf{C}_r$ blocks all paths between $\mathbf{X}_r$ and $\mathbf{C}_l \cup \mathbf{C}_w$.

Furthermore, w.l.o.g., we set $\mathbf{C}_{\text{root of } W} := \varnothing$ and $\mathbf{C}_{X_j} :=$ parent of X_j in $G_{\mathscr{A}}$ for the leaf nodes $X_j \in W$. See Figure 1c for an example.

Assuming that we have computed a valid labelling for W, we can then proceed to construct the desired PC by a bottom-up recursion on W. For the base case, if w is a leaf node representing some random variable X_j, $p(X_j | \mathbf{C}_{X_j}) = p(X_j \mid \text{parent of } X_j \text{ in } G_{\mathscr{A}})$, which is directly given by the conditional probability table of $G_{\mathscr{A}}$. For the induction step, when w is a inner node with children l and r, we have the recurrence relation:

$$
\begin{aligned}
&p(\mathbf{X}_w \mid \mathbf{C}_w) \\
&= \sum_{(\mathbf{C}_l \cup \mathbf{C}_r) \setminus \mathbf{C}_w} p(\mathbf{X}_l, \mathbf{X}_r \mid \mathbf{C}_l, \mathbf{C}_r) \cdot p(\mathbf{C}_l, \mathbf{C}_r \mid \mathbf{C}_w) \\
&= \sum_{(\mathbf{C}_l \cup \mathbf{C}_r) \setminus \mathbf{C}_w} p(\mathbf{X}_l \mid \mathbf{C}_l) \cdot p(\mathbf{X}_r \mid \mathbf{C}_r) \cdot p(\mathbf{C}_l, \mathbf{C}_r \mid \mathbf{C}_w)
\end{aligned}
$$

Here the first step follows from Property 2 and 3, and the second step follows from all properties in Defintion 3.2. The circuit materialization of the recurrence relation is shown in Figure 2. Note that if w is the root, then $p(\mathbf{X}_w | \mathbf{C}_w)$ becomes $p(\mathbf{X})$, which is a single sum node representing the distribution of $\mathscr{A}$. The complete recursion is given by Algorithm 1. The following Theorem bounds the size of the resulting PC:

[2]a path P is blocked by a set $\mathbf{S}$ if $P \cap \mathbf{S} \neq \varnothing$.

[3]This is a special case of *d-separation* (Geiger, Verma, and Pearl 1990).

Algorithm 1: Construct PC with respect to W

procedure CONSTRUCTCIRCUIT(w)
  if w is a leaf node X_i **then**
    return $p(X_i \mid \mathbf{C}_{X_i})$
  end if
  $l, r \leftarrow$ CHILDREN(w)
  $\bigoplus_{\mathbf{X}_l, \mathbf{C}_l} \leftarrow$ CONSTRUCTCIRCUIT(l)
  $\bigoplus_{\mathbf{X}_r, \mathbf{C}_r} \leftarrow$ CONSTRUCTCIRCUIT(r)
  $\bigoplus_{\mathbf{X}_w, \mathbf{C}_w} \leftarrow \sum_{(\mathbf{C}_l \cup \mathbf{C}_r) \setminus \mathbf{C}_w} \bigoplus_{\mathbf{C}_l} \cdot \bigoplus_{\mathbf{C}_r} \cdot p(\mathbf{C}_l, \mathbf{C}_r \mid \mathbf{C}_w)$
  return $\bigoplus_{\mathbf{C}_w}$
end procedure

Algorithm 2: Computing $\mathbf{C}_w$ for $w \in W$

procedure COMPUTELABEL($w, \mathbf{C}_w$)
  $\{G_i\} \leftarrow$ CONNECTEDCOMPONENTS($G_{\mathscr{A}}, \mathbf{C}_w$)
  $\mathbf{C}_i \leftarrow$ MINIMUMSEPARATOR($G_i, \mathbf{X}_l \cap G_i, \mathbf{X}_r \cap G_i$)
  $\mathbf{D}_w \leftarrow (\bigcup_i \mathbf{C}_i) \cup \mathbf{C}_w$
  $\mathbf{C}_l \leftarrow \{Z_j \in \mathbf{D}_w :$ PATHS$(\mathbf{X}_l, Z_j) \cap \mathbf{D}_w = \{Z_j\}\}$
  $\mathbf{C}_r \leftarrow \{Z_j \in \mathbf{D}_w :$ PATHS$(\mathbf{X}_r, Z_j) \cap \mathbf{D}_w = \{Z_j\}\}$
  COMPUTELABEL($l, \mathbf{C}_l$)
  COMPUTELABEL($r, \mathbf{C}_r$)
end procedure

Theorem 3.3. *The number of hidden states of the restructured PC is given by $O(h^M)$ where $M = \max_{w \in W} |\mathbf{C}_l \cup \mathbf{C}_r|$ and the size of the restructured PC is bounded by $O(nh^{M'})$ where $M' = \max_{w \in W} |\mathbf{C}_l \cup \mathbf{C}_r \cup \mathbf{C}_w| \leq 2M$. We refer to M' as the cardinality of the labelling $\mathbf{C}_w$.*

Computing Vtree Labelling

The next question that arises is how to compute a (compact) valid labelling for W with respect to $G_{\mathscr{A}}$. Here we present a greedy approach that computes a labelling while trying to minimize M'. The algorithm (Algorithm 2) proceeds top-down on W. For the base case where w is the root, we set $\mathbf{C}_w := \varnothing$. For the inductive step, let l and r be the children of w and assume that we have computed $\mathbf{C}_w$ as a cover for $\mathbf{X}_w$ in $G_{\mathscr{A}}$: we (1) split $G_{\mathscr{A}}$ into connected components $\{G_i\}$ via $\mathbf{C}_w$; then (2) within each connected component G_i, compute a minimum d-separator $\mathbf{C}_i$ that blocks all paths between $\mathbf{X}_l \cap G_i$ and $\mathbf{X}_r \cap G_i$ by calling the sub-routine MINIMUMSEPARATOR. We set $\mathbf{D}_w := (\bigcup_i \mathbf{C}_i) \cup \mathbf{C}_w$ and observe that $\mathbf{D}_w$ covers both $\mathbf{X}_l$ and $\mathbf{X}_r$ in $G_{\mathscr{A}}$. To compute $\mathbf{C}_l$, similarly for $\mathbf{C}_r$, we consider all paths starting from $\mathbf{X}_l$ and stopping immediately when reaching some $Z_j \in \mathbf{D}_w$, and we let $\mathbf{C}_l$ to be the set containing all such Z_js.

Proposition 3.4. *Algorithm 2 computes a valid labelling with respect to $G_{\mathscr{A}}$.*

We leave open the problem of whether $M' = \max_{w \in W} |\mathbf{C}_l \cup \mathbf{C}_r \cup \mathbf{C}_w|$ is globally minimized.

4 Applications

Circuit Multiplication One important application of restructuring PCs is circuit multiplication: given two PCs $\mathscr{A}$

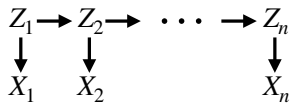


Figure 3: $G_{\mathscr{A}}$ for $\mathscr{A}$ with a linear vtree

and $\mathscr{B}$, the goal is to construct a tractable PC $\mathscr{C}$ such that $p_{\mathscr{C}}(\mathbf{x}) \propto p_{\mathscr{A}}(\mathbf{x}) \cdot p_{\mathscr{B}}(\mathbf{x})$. PC multiplication was previously only addressed for structured PCs respecting the *same* vtree (Shen, Choi, and Darwiche 2016; Vergari et al. 2021). Circuit restructuring immediately gives us a means of multiplying two structured circuits respecting *different* vtrees, as we can simply restructure one of them to be compatible with the other. Though the restructured PC will in general have exponential size, in this section, we consider practical cases where circuit multiplications with respect to *different* vtrees is tractable. We start by introducing a new structural property of tractable PCs called *contiguity*.

Definition 4.1 (Contiguity)**.** Given the canonical ordering of variables $X_1, X_2, \ldots, X_n$, a PC node is *contiguous* if its scope is of the form $X_a, X_{a+1}, \ldots, X_b$ for some $1 \leq a \leq b \leq n$. A smooth and decomposable PC is contiguous if all of its nodes are contiguous.

We start by considering the case when $\mathscr{A}$ is a contiguous structured PC respecting the linear vtree V and $\mathscr{B}$ is a contiguous structured PC respecting an arbitrary vtree W. It follows from Section 3 that the Bayesian network representation for $\mathscr{A}$ is a hidden Markov model (Rabiner 1989), as shown in Figure 3. By the definition of contiguity, each node $w \in W$ has a scope of the form $\mathbf{X}_{a:b} := \{X_a, \ldots, X_b\}$ and we can label it with $\mathbf{C}_{a:b} := \{Z_a, Z_{b+1}\}$; in particular, we drop Z_a if $a = 1$ and drop Z_{b+1} if $b = n$.

Claim 4.2. *$\mathbf{C}_{a:b}$ is a valid vtree labelling of W respecting $G_{\mathscr{A}}$ with cardinality $M' = 3$.*

It follows from Theorem 3.3 that the size of $\mathscr{A}'$, i.e., the PC obtained by restructuring $\mathscr{A}$ respecting W, is bounded by $O(nh^3)$, with $O(|\mathscr{A}|^2)$ being a looser bound. Further, we can compute the product of $\mathscr{A}'$ with $\mathscr{B}$ tractably by the existing algorithm for multiplying circuits respecting the same vtree (Shen, Choi, and Darwiche 2016; Vergari et al. 2021).

Theorem 4.3. *Let $\mathscr{A}$ and $\mathscr{B}$ be contiguous structured PCs. If $\mathscr{A}$ has a linear vtree, then $\mathscr{A}$ and $\mathscr{B}$ can be multiplied in polynomial time and the size of the product PC is bounded by $O(|\mathscr{A}|^2|\mathscr{B}|)$.*

We can derive a similar result for arbitrary contiguous structured PCs, essentially by application of Algorithm 2; we provide details in Appendix D.

Depth Reduction Depth reduction of a probabilistic circuit refers to the construction of an equivalent circuit with reduced depth, e.g. to a depth logarithmic in the number of variables. A depth reduction algorithm for general circuits is known (Valiant et al. 1983; Raz and Yehudayoff 2008; Yin and Zhao 2024) but does not take advantage of structuredness. We show how to reduce a structured-decomposable circuit to an equivalent log-depth circuit by restructuring.

Algorithm 3: Depth Reduction Vtree

```
1:  procedure BalancedVtree(V, S_l = ∅, S_r = ∅)
2:      if |V| = 1 then
3:          return Leaf(V; S_l ∪ S_r)
4:      end if
5:      v ← Root(V)
6:      l, r ← Children(v)
7:      while |V_r| > (2/3)|V| do
8:          v ← r
9:          l, r ← Children(v)          ▷ assume |V_l| ≤ |V_r|
10:     end while
11:     V'_l ← BalancedVtree(V_[v↦l], S_l, {Z_v})
12:     V'_r ← BalancedVtree(V_r, {Z_v}, S_r)
13:     return Join(V'_l, V'_r; S_l ∪ S_r)
14: end procedure
```

Algorithm 3 constructs a log-depth vtree labelling of constant cardinality. Intuitively, each step of the algorithm breaks a vtree down into two connected components, which are then depth-reduced recursively. One selects a single vtree node by traversing the vtree top-down, until the split would be balanced in the sense that the two connected components have size between $\frac{1}{3}$ and $\frac{2}{3}$ of the input vtree (Lines 7-10). The algorithm simultaneously constructs a valid label for the vtree node. The JOIN routine then returns a labelled vtree that consists of a single root node with the aforementioned label, connected to the depth-reduced component vtrees. It can be easily verified that the vtree labelling produced is valid and has cardinality 3. This gives the following result:

Theorem 4.4. *Any structured PC over n variables and with hidden state size h can be restructured to a structured PC of depth $O(\log n)$ and size $O(nh^3)$ that represents the same distribution.*

Thus, we have shown that one can depth-reduce a structured PC while (i) retaining structured decomposability and (ii) having a sub-quadratic bound on the size of the depth-reduced PC (as compared with a cubic bound for the general algorithm (Raz and Yehudayoff 2008)).

5 Discussion

Our work takes a first step to understanding when and how the space of structured circuits -- hierarchical tensor factorizations (Loconte et al. 2024) -- can be "connected" through tractable transformations. Many interesting theoretical problems remain open: for instance, the optimality of our algorithm, and circuit lower bounds for particular structures. We also envisage that our restructuring algorithm will enable new methodology both within the PC community and broadly in structured low-rank representations. For example, multiplication for contiguous PCs enables one to combine hidden Markov models with more flexible logical constraints for constrained language generation (Zhang et al. 2024). Additionally, our sub-quadratic depth reduction algorithm enables one to practically trade off model size for greater parallelization, potentially improving processing throughput on modern hardware.

References

Ahmed, K.; Teso, S.; Chang, K.-W.; Van den Broeck, G.; and Vergari, A. 2022. Semantic probabilistic layers for neuro-symbolic learning. *Advances in Neural Information Processing Systems*, 35: 29944–29959.

Amarilli, A.; Arenas, M.; Choi, Y.; Monet, M.; Broeck, G. V. d.; and Wang, B. 2024. A Circus of Circuits: Connections Between Decision Diagrams, Circuits, and Automata. *arXiv preprint arXiv:2404.09674*.

Bova, S. 2016. SDDs are exponentially more succinct than OBDDs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Butz, C.; Oliveira, J. S.; and Peharz, R. 2020. Sum-product network decompilation. In *International Conference on Probabilistic Graphical Models*, 53–64. PMLR.

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

Choi, A.; Van den Broeck, G.; and Darwiche, A. 2015. Tractable Learning for Structured Probability Spaces: A Case Study in Learning Preference Distributions. In *Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI)*.

Choi, Y.; Vergari, A.; and Van den Broeck, G. 2020. Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models.

Conaty, D.; Maua, D. D.; and de Campos, C. P. 2017. Approximation Complexity of Maximum A Posteriori Inference in Sum-Product Networks. In *The 33rd Conference on Uncertainty in Artificial Intelligence (UAI)*. AUAI.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2022. *Introduction to algorithms*. MIT press.

Dang, M.; Liu, A.; and Van den Broeck, G. 2022. Sparse Probabilistic Circuits via Pruning and Growing. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*.

Dang, M.; Liu, A.; Wei, X.; Sankararaman, S.; and Van den Broeck, G. 2022. Tractable and Expressive Generative Models of Genetic Variation Data. In *Proceedings of the International Conference on Research in Computational Molecular Biology (RECOMB)*.

Dang, M.; Vergari, A.; and Broeck, G. 2020. Strudel: Learning structured-decomposable probabilistic circuits. In *International Conference on Probabilistic Graphical Models*, 137–148. PMLR.

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

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

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

de Colnet, A.; and Mengel, S. 2021. A Compilation of Succinctness Results for Arithmetic Circuits. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 18, 205–215.

Geiger, D.; Verma, T.; and Pearl, J. 1990. d-separation: From theorems to algorithms. In *Machine intelligence and pattern recognition*, volume 10, 139–148. Elsevier.

Khosravi, P.; Choi, Y.; Liang, Y.; Vergari, A.; and Van den Broeck, G. 2019. On Tractable Computation of Expected Predictions. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*.

Kisa, D.; Van den Broeck, G.; Choi, A.; and Darwiche, A. 2014. Probabilistic sentential decision diagrams. In *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

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

Liu, A.; Niepert, M.; and Van den Broeck, G. 2024. Image Inpainting via Tractable Steering of Diffusion Models. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*.

Liu, A.; and Van den Broeck, G. 2021. Tractable Regularization of Probabilistic Circuits. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*.

Loconte, L.; Mari, A.; Gala, G.; Peharz, R.; de Campos, C.; Quaeghebeur, E.; Vessio, G.; and Vergari, A. 2024. What is the Relationship between Tensor Factorizations and Circuits (and How Can We Exploit it)? *arXiv preprint arXiv:2409.07953*.

Papantonis, I.; and Belle, V. 2023. Transparency in Sum-Product Network Decompilation. In *European Conference on Artificial Intelligence*, 1827–1834. IOS Press.

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

Pipatsrisawat, K.; and Darwiche, A. 2008. New Compilation Languages Based on Structured Decomposability. In *AAAI*, volume 8, 517–522.

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

Rabiner, L. R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2): 257–286.

Rahman, T.; Kothalkar, P.; and Gogate, V. 2014. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part II 14*, 630–645. Springer.

Raz, R.; and Yehudayoff, A. 2008. Balancing syntactically multilinear arithmetic circuits. *Computational Complexity*, 17: 515–535.

Raz, R.; and Yehudayoff, A. 2009. Lower bounds and separations for constant depth multilinear circuits. *Computational Complexity*, 18: 171–207.

Rooshenas, A.; and Lowd, D. 2014. Learning sum-product networks with direct and indirect variable interactions. In *International Conference on Machine Learning*, 710–718. PMLR.

Shen, Y.; Choi, A.; and Darwiche, A. 2016. Tractable operations for arithmetic circuits of probabilistic models. *Advances in Neural Information Processing Systems*, 29.

Shih, A.; and Ermon, S. 2020. Probabilistic circuits for variational inference in discrete graphical models. *Advances in neural information processing systems*, 33: 4635–4646.

Sidheekh, S.; and Natarajan, S. 2024. Building Expressive and Tractable Probabilistic Generative Models: A Review. *arXiv preprint arXiv:2402.00759*.

Valiant, L.; Skyum, S.; Berkowitz, S.; and Rackoff, C. 1983. Fast Parallel Computation of Polynomials Using Few Processors. *SIAM Journal on Computing*, 12(4): 641–644.

Vergari, A.; Choi, Y.; Liu, A.; Teso, S.; and Van den Broeck, G. 2021. A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*.

Wang, B.; and Kwiatkowska, M. 2023. Compositional probabilistic and causal inference using tractable circuit models. In *International Conference on Artificial Intelligence and Statistics*, 9488–9498. PMLR.

Yang, Y.; Gala, G.; and Peharz, R. 2023. Bayesian structure scores for probabilistic circuits. In *International Conference on Artificial Intelligence and Statistics*, 563–575. PMLR.

Yin, L.; and Zhao, H. 2024. On the Expressive Power of Tree-Structured Probabilistic Circuits. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*.

Zhang, H.; Dang, M.; Peng, N.; and Van den Broeck, G. 2023. Tractable Control for Autoregressive Language Generation. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*.

Zhang, H.; Kung, P.-N.; Yoshida, M.; Broeck, G. V. d.; and Peng, N. 2024. Adaptable Logical Control for Large Language Models. *arXiv preprint arXiv:2406.13892*.

Zhao, H.; Melibari, M.; and Poupart, P. 2015. On the relationship between sum-product networks and Bayesian networks. In *International Conference on Machine Learning*, 116–124. PMLR.

Zhao, H.; Poupart, P.; and Gordon, G. J. 2016. A unified approach for learning the parameters of sum-product networks. *Advances in neural information processing systems*, 29.

A Structured PC to BN Construction

We begin by providing a latent variable interpretation of structured PCs. Specifically, we define an augmented PC which explicitly associates latent variables with product nodes for each variable scope. Given some vtree node v, let us associate each $t \in \text{prod}(v)$ with a unique index $\text{idx}(t) \in \{0, ..., |\text{prod}(v)|-1\}$, also writing $t_{v,i}$ to refer to the product node with index i in $\text{prod}(v)$. Then we can introduce a categorical latent variable Z_v whose value corresponds to a particular product node in $\text{prod}(v)$:

Definition A.1 (Augmented PC)**.** Given a structured-decomposable and smooth PC $\mathscr{A}$ over variables $\mathbf{X}$ respecting vtree V, we define the augmented PC $\mathscr{A}_{\text{aug}}$ to be a copy of $\mathscr{A}$ where for each vtree node $v \in V$, we add an additional child t_{aug} to each product node $t \in \text{prod}(v)$ that is a leaf node with scope Z_v and leaf function $f_{t_{\text{aug}}}(Z_v) = \mathbb{1}_{Z_v=\text{idx}(t)}$.

It is not hard to see that the augmented PC $\mathscr{A}_{\text{aug}}$ is a PC over variables $\mathbf{X}, \mathbf{Z}$ and retains structured decomposability and smoothness. Further, the standard marginalization algorithm for PCs ensures that the augmented PC has the correct distribution:

Proposition A.2. $p_{\mathscr{A}}(\mathbf{X}) = \sum_{\mathbf{z}} p_{\mathscr{A}_{aug}}(\mathbf{X}, \mathbf{z})$

Proof. Suppose that $\mathscr{A}$ is a structured decomposable and smooth PC respecting vtree V. Write $\text{prod}(v)$ and $\text{sum}(v)$ for the set of product and sum nodes with scope $\mathbf{X}_v$. The augmented PC $\mathscr{A}_{\text{aug}}$ is decomposable as if leaves with scope $\{Z_v\}$ were contained in (the subcircuits rooted at) two different children t_1, t_2 of a product node, then their parents (nodes in $\text{prod}(v)$) would be contained in t_1, t_2, which is a contradiction of decomposability of $\mathscr{A}$. It is also smooth as for any sum node, if one sum node contains some leaf with scope $\{Z_v\}$, then it contains some node in $\text{sum}(v)$, hence by smoothness of $\mathscr{A}$ all sum nodes contain some node in $\text{sum}(v)$ and thus some leaf with scope $\{Z_v\}$.

Consider the standard marginalization algorithm for PCs (Darwiche 2003; Choi, Vergari, and Van den Broeck 2020), where one replaces each leaf whose scope is contained within the variables being marginalized out with the constant 1. This correctly marginalizes the function represented by the PC if the PC is decomposable and smooth. If we marginalize over all newly introduced latents $\mathbf{Z}$, it is immediate that the resulting PC represents the same function as $\mathscr{A}$. □

Let $V_{v \to Z_v}$ be the rooted DAG obtained by replacing all inner nodes v in vtree V with variable Z_v (cf. Fig. 1). Now, we claim that the augmented PC can be interpreted as a Bayesian network with graph structure $V_{v \to Z_v}$. To do this, we construct a distribution $p^*(\mathbf{X}, \mathbf{Z})$, based on the augmented PC, that factorizes as required by the Bayesian network structure.

There are three cases to consider: (i) the root node $p^*(Z_{\text{ROOT}(V)})$, (ii) the leaf nodes $p^*(X_v|Z_p)$, and (iii) other nodes $p^*(Z_v|Z_p)$ (where we write p for the parent of v in V). In case (i), we set $p^*(Z_v = i) := w_i$ where w_i is the weight of the edge from the root sum node to the product node $t_{v,i}$. In case (ii), we set $p^*(X_v|Z_p = j) = p_t(X_v)$, where t is the leaf node child (with scope X_v) of the product node $t_{p,j}$. Finally, in case (iii) we note that due to alternating sums and products, $t_{p,j}$ must have a sum node child, which may or may not have a weighted edge to $t_{v,i}$ (whose weight we denote by w_{ij} if it exists). We thus define:

$$p^*(Z_v = i|Z_p = j) = \begin{cases} w_{ij} & \exists \text{ path from } t_{p,j} \text{ to } t_{v,i} \\ 0 & \text{otherwise} \end{cases}$$

It remains to show that this distribution faithfully represents the distribution of the augmented PC, i.e. $p_{\mathscr{A}_{\text{aug}}} = p^*$. The intuitive idea is that each value of $\mathbf{Z}$ corresponds to a subtree of $\mathscr{A}_{\text{aug}}$, whose value is precisely given by the product of weights and leaf functions specified by the Bayesian network.

Theorem A.3. *Let $\mathscr{A}$ be a structured-decomposable and smooth PC over variables $\mathbf{X}$ respecting vtree V. Then there exists a Bayesian network $G_{\mathscr{A}}$ over variables $\mathbf{X}$ and $\mathbf{Z} = \{Z_v | v \in V\}$ with graph $V_{v \mapsto Z_v}$ such that $\sum_{\boldsymbol{z}} p_G(\mathbf{X}, \boldsymbol{z}) = p_{\mathscr{A}}(\mathbf{X})$.*

Proof. In Section 3 we described a Bayesian network $p_G = p^*$ with the required graph. It remains to show that this network represents the same distribution as $\mathscr{A}$. We will do this by showing that the Bayesian network has the same distribution as the augmented PC, i.e. $p_G(\mathbf{X}, \mathbf{Z}) = p_{\mathscr{A}_{\text{aug}}}(\mathbf{X}, \boldsymbol{Z})$.

The key observation is to consider the *induced trees* of the augmented PC (Zhao, Poupart, and Gordon 2016):

Definition A.4 (Induced Trees)**.** Given a decomposable and smooth circuit $\mathscr{A}$, let T be a subgraph of $\mathscr{A}$. We say that T is an induced tree of $\mathscr{A}$ if (1) $\text{ROOT}(\mathscr{A}) \in T$; (2) If $t \in T$ is a sum node, then exactly one child of t (and the corresponding edge) is in T; and (3) If $t \in T$ is a product node, then all children of t (and the corresponding edges) are in T.

It is easy to see that an induced tree T is indeed a tree, as otherwise decomposability would be violated. Let $\mathcal{T}$ be the set of all induced trees of $\mathscr{A}_{\text{aug}}$. Each induced tree defines a function:

$$p_{\mathscr{A}_{\text{aug}},T}(\mathbf{X}, \mathbf{Z}) := \prod_{(t_i,t_j) \in \text{SUMEDGES}(T)} w_{t_i,t_j} \prod_{t \in \text{LEAVES}_{\mathbf{X}}(T)} f_t(X_{\text{sc}(t)}) \prod_{t \in \text{LEAVES}_{\mathbf{Z}}(T)} f_t(Z_{\text{sc}(t)})$$

where $\text{SUMEDGES}(T)$ denotes the set of outgoing edges from sum nodes in T, and $\text{LEAVES}_{\mathbf{X}}(T), \text{LEAVES}_{\mathbf{Z}}(T)$ denote the set of leaf nodes in T with scope corresponding to a variable in $\mathbf{X}, \mathbf{Z}$ respectively. The distribution of the augmented PC is then in fact given by the sum of these functions over all induced trees:

Proposition A.5 (Zhao, Poupart, and Gordon (2016))**.** $p_{\mathscr{A}_{aug}}(\mathbf{X}, \mathbf{Z}) = \sum_{T \in \mathcal{T}} p_{\mathscr{A}_{aug},T}(\boldsymbol{X}, \boldsymbol{Z})$.

Now, let path(v, i, j) be a predicate indicating whether there is a path between $t_{p,j}$ and $t_{v,i}$ (where we use p to denote the parent vtree node of v, and as before e.g. $t_{v,i}$ indicates the product node with scope $\mathbf{X}_v$ and corresponding to $Z_v = i$). We will consider two cases depending on the value of the latents. Specifically, we will say that an assignment $\boldsymbol{z}$ is *consistent* if path(v, z_v, z_p) holds for all non-root inner nodes in the vtree, and *inconsistent* otherwise.

If an assignment $\boldsymbol{z}$ is inconsistent, then for any assignment $\mathbf{x}$ of the observed variables, we have that $p_G(\mathbf{x}, \mathbf{z}) = 0$ by definition of the Bayesian network distribution. Now consider any induced subtree $T \in \mathcal{T}$. Each T must contain one product node for every variable scope. In particular, T must contain some product node $t_{v,i}$ such that $z_v \neq i$ (otherwise, since $\mathbf{z}$ is inconsistent, a (connected) tree would be impossible). We then have $p_{\mathscr{A}_{\text{aug}},T}(\boldsymbol{x}, \boldsymbol{z}) = 0$ for all $\boldsymbol{x}$, as $p_{\mathscr{A}_{\text{aug}},T}(\boldsymbol{x}, \boldsymbol{z})$ then contains a leaf function $f_t(z_v) = \mathbb{1}_{z_v=i} = 0$. Thus $p_{\mathscr{A}_{\text{aug}}}(\mathbf{x}, \mathbf{z}) = \sum_{T \in \mathcal{T}} p_{\mathscr{A}_{\text{aug}},T}(\boldsymbol{x}, \boldsymbol{z}) = 0$ for any $\mathbf{x}$.

If an assignment $\mathbf{z}$ is consistent, note that, by our assumption of alternating sums and products, there can be exactly one path from $t_{p,j}$ to $t_{v,j}$, as $t_{p,j}$ has a unique sum node child with scope containing $\mathbf{X}_v$, and this sum node must immediately have $t_{v,j}$ as a child. Thus there is exactly one induced tree T containing t_{v,z_v} for all (non-root) inner vtree nodes v. Further, examining the definition of the Bayesian network distribution $p_G(\boldsymbol{X}, \boldsymbol{z})$, this exactly matches the definition of $p_{\mathscr{A}_{\text{aug}},T}(\boldsymbol{X}, \boldsymbol{z})$: each sum node edge weight in the tree corresponds to a sum node edge weight along a path from some t_{p,z_p} to t_{v,z_v} and thus the CPT of Z_v given Z_p (the root sum node edge weight corresponds to the CPT for $Z_{\text{ROOT}(V)}$), and each leaf node distribution for observed variables corresponds to the CPT for that variable given its parent.

Thus we have shown that $p_{\mathscr{A}_{\text{aug}}}(\boldsymbol{X}, \boldsymbol{Z}) = p_G(\boldsymbol{X}, \boldsymbol{Z})$, as required. □

B PC Restructuring: Proofs

In this section we provide the proofs for results in Section 3.

Theorem 3.3. *The number of hidden states of the restructured PC is given by $O(h^M)$ where $M = \max_{w \in W} |\mathbf{C}_l \cup \mathbf{C}_r|$ and the size of the restructured PC is bounded by $O(nh^{M'})$ where $M' = \max_{w \in W} |\mathbf{C}_l \cup \mathbf{C}_r \cup \mathbf{C}_w| \leq 2M$. We refer to M' as the cardinality of the labelling $\mathbf{C}_w$.*

Proof. Let $\mathscr{A}'$ be the restructured circuit respecting W. As described in Algorithm 1, for each inner node $w \in W$, we construct two layers of nodes as shown in Figure 2. By construction, the product layer contains all product nodes respecting the vtree node w and its cardinality is given by $O(h^{|\mathbf{C}_l \cup \mathbf{C}_r|})$; we set $M := \max_{w \in W} |\mathbf{C}_l \cup \mathbf{C}_r|$ and it follows that the hidden states size of $\mathscr{B}$ is given by $O(h^M)$. Similarly, the number of edges in the sum layer is given by $O(h^{|\mathbf{C}_l \cup \mathbf{C}_r \cup \mathbf{C}_w|})$ and the number of product edges is given by $O(h^{|\mathbf{C}_l \cup \mathbf{C}_r|})$; since there are $O(n)$ vtree nodes in total, the total number of edges in $\mathscr{B}$ is given by $O(nh^{M'})$, with $M' = \max_{w \in W} |\mathbf{C}_l \cup \mathbf{C}_r \cup \mathbf{C}_w|$. □

Proposition 3.4. *Algorithm 2 computes a valid labelling with respect to $G_{\mathscr{A}}$.*

Proof. We prove by a top-down induction on W that the labelling $\mathbf{C}_w$ computed by Algorithm 2 is valid. Assume that $\mathbf{C}_w$ covers $\mathbf{X}_w$ in $G_{\mathscr{A}}$, we want to show that $\mathbf{C}_l$ and $\mathbf{C}_r$ satisfy the properties from Definition 3.2. To prove that $\mathbf{C}_l$ covers $\mathbf{X}_l$, we consider a path from $X_a \in \mathbf{X}_l$ to $X_b \in \mathbf{X} \setminus \mathbf{X}_l$. (1) If X_a and X_b are in the same G_i, then the path is blocked by $\mathbf{C}_i$. (2) If X_a and X_b are in different G_is, then the path contains some node $Z \in \mathbf{C}_w$, and we can choose from the path the first $Z \in \mathbf{C}_w$. Then $Z \in \mathbf{C}_l$ by construction, implying that the path is blocked by $\mathbf{C}_l$. Hence we conclude that $\mathbf{C}_l$ is a cover for $\mathbf{X}_l$, satisfying Property 1. To prove that $\mathbf{C}_l$ satisfies Property 2, we argue that because $\mathbf{C}_r$ and $\mathbf{C}_w$ are both subsets of $\mathbf{D}_w$, all paths from $\mathbf{X}_l$ to $\mathbf{C}_r \cup \mathbf{C}_w$ will be blocked by $\mathbf{C}_l$ by the way that $\mathbf{C}_l$ is constructed. We can show that $\mathbf{C}_r$ satisfies Property 1 and Property 3 by the same argument. □

C Determinism and Logical Circuits

In this section, we examine the restructuring of two other types of circuits: namely, deterministic PCs, and logical circuits.

Definition C.1 (Determinism)**.** A sum node is deterministic if for every value $\mathbf{x}$ of $\mathbf{X}$, at most one child c returns a non-zero value (i.e. $p_c(\mathbf{x}) > 0$). A PC is determinstic if all of its sum nodes are deterministic.

Determinism is crucial for tractability of various inference queries such as computing the most likely state (MAP) (Peharz et al. 2016; Conaty, Maua, and de Campos 2017) or computing the entropy of the PC's distribution (Shih and Ermon 2020; Vergari et al. 2021). It is thus of interest to ask whether applying our restructuring algorithm maintains determinism.

Claim C.2. *Algorithm 1 preserves determinism.*

Proof. If the original circuit is deterministic, then each assignment to the observed variables fully determines the values of all latent variables (and thus the latents being conditioned on for the restructuring). Hence the constructed sum nodes are deterministic. □

Although we have focused on probabilistic circuits up to this point, our restructuring algorithm also applies to logical circuits - in particular, structured-decomposable negation normal form (SDNNF) circuits[4] (Pipatsrisawat and Darwiche 2008). To see this, we use a simple trick: (1) convert the logical circuit into a probabilistic circuit by replacing $\vee$ with $\oplus$ and $\wedge$ with $\otimes$, and assigning positive weights to $\oplus$ edges; (2) restructure the PC; (3) convert the PC back to a logical circuit by replacing $\oplus$ with $\vee$ and $\otimes$ with $\wedge$, and removing the weights. It is immediate that the logical circuits

[4]Many other representations, such as the ordered binary decision diagram (OBDD) and deterministic finite automaton (DFA), can be converted efficiently to (deterministic) SDNNFs (Amarilli et al. 2024).

and the corresponding PCs have the same support throughout the process.

It is also not hard to see that this procedure for logical circuits retains determinism, so, e.g, an ordered binary decision diagram (OBDD) can be efficiently restructured into a deterministic SDNNF with the reverse order while maintaining the ability to perform model counting (Darwiche and Marquis 2002).

D Circuit Multiplication

In the main paper, we described a restructuring that enables tractable multiplication of contiguous structured circuits where one respects a linear vtree. Here we prove the correctness of the proposed labelling:

Claim 4.2. *$\mathbf{C}_{a:b}$ is a valid vtree labelling of W respecting $G_{\mathscr{A}}$ with cardinality $M' = 3$.*

Proof. We first show that $\mathbf{C}_w$ satisfies the following properties:

1. $X_{a:b} = \bigcup_{Z_i \in \mathbf{C}_{a:b}} \textsc{Leaves}(Z_i)$ is a *disjoint* union.
2. If $a \leq c \leq d \leq b$, then for $Z \in \mathbf{C}_{c:d}$, there exists $Z' \in \mathbf{C}_{a:b}$ such that Z' is an ancestor of Z in $G_{\mathscr{A}}$.

Property 1 follows from the proof of correctness of the segment tree querying algorithm. Property 2 follows from Property 1 together with the key observation that we can compute $\mathbf{C}_{c:d}$ via $\bigcup_{Z_i \in \mathbf{C}_{a:b}} \textsc{SegmentCover}(Z_i, \mathbf{X}_{c:d} \cap \textsc{Leaves}(Z_i))$. Let $w = a : b$ be a node in W with children $l = a : c$ and $r = c + 1 : b$; it follows from Property 1 that $\mathbf{C}_{a:b}$ covers $\mathbf{X}_{a:b}$ and $\mathbf{C}_{a:c}$ blocks all paths from $\mathbf{X}_{a:c}$ to $\mathbf{C}_{c+1:b}$; it follows from Property 2 that $\mathbf{C}_{a:c}$ blocks all paths from $\mathbf{X}_{a:c}$ to $\mathbf{X}_{a:b}$. Hence we conclude that $\mathbf{C}_w$ is a valid labelling. A minor catch is that $\mathbf{C}_w$ may contain variables in $\mathbf{X}$, but we can replace them by their parent in $G_{\mathscr{A}}$ without affecting the validity of $\mathbf{C}_w$. □

We now consider the more general case where $\mathscr{A}$ is a contiguous structured PC of depth d respecting vtree V and $\mathscr{B}$ is a contiguous structured PC with an arbitrary vtree W. Similarly to the previous case, our goal is to come up with a small labelling of W with respect to $G_{\mathscr{A}}$. Since $\mathscr{A}$ is contiguous, its vtree V can be viewed as a *segment tree* (Cormen et al. 2022). Algorithm 4, which is adapted from the segment tree querying algorithm, computes a cover $\mathbf{C}_{a:b} \subseteq G_{\mathscr{A}}$ for each contiguous segment $\mathbf{X}_{a:b}$. For each $w \in W$, $\mathbf{X}_w = X_{a:b}$ for some $1 \leq a \leq b \leq n$ and we set $\mathbf{C}_w = \mathbf{C}_{a:b} = \textsc{SegmentCover}(V, X_{a:b})$.

Proposition D.1. *$\mathbf{C}_w$ is a valid vtree labelling with respect to $G_{\mathscr{A}}$.*

In addition to the fact that $\mathbf{C}_w$ is a valid labelling, by the runtime analysis of the original segment tree querying algorithm, we know that the number of nodes visited at each level of V is at most 4 and it follows that $|\mathbf{C}_w| \leq 4d$; hence the cardinality of $\mathbf{C}_w$ is bounded by $12d$. Then following a similar analysis, we have the following results for multiplying two contiguous PCs.

Algorithm 4: Compute Cover for Segment $\mathbf{X}_{a:b}$

```
procedure SegmentCover(v, X_{a:b})
    if X_{a:b} = ∅ then
        return ∅
    end if
    if X_{a:b} = X_v then
        return {Z_v}
    end if
    l, r ← Children(v)
    L ← SegmentCover(l, X_l ∩ X_{a:b})
    R ← SegmentCover(r, X_r ∩ X_{a:b})
    return L ∪ R
end procedure
```

Theorem D.2. *Let $\mathscr{A}$ and $\mathscr{B}$ be contiguous structured PCs. Let d be the depth of the vtree for $\mathscr{A}$, then $\mathscr{A}$ and $\mathscr{B}$ can be multiplied in time $O(|\mathscr{A}|^{12d}|\mathscr{B}|)$.*

Corollary D.3. *If $\mathscr{A}$ is of depth $O(\log n)$ then $\mathscr{A}$ and $\mathscr{B}$ can be multiplied in quasi-polynomial time.*

Remark D.4. In this work, we assumed product nodes always have two children and binary vtrees. Hence the depths of PCs are lower-bounded by $\Omega(\log n)$ under such assumptions. However, if we allow product nodes to have arbitrarily many children, we can have PCs of smaller or even constant depths (Raz and Yehudayoff 2009) and we hypothesize that Theorem 3.3 can be adapted to such generalized cases thus giving a polynomial-time algorithm for multiplying contiguous structured circuits of constant depths.

Remark D.5. Thus far, we have assumed that both $\mathscr{A}$ and $\mathscr{B}$ are structured PCs. We claim that we can further generalize our results by removing the constraint that $\mathscr{B}$ has to be structured, and Theorems 4.3 and D.2 would still hold. We illustrate the idea by showing how to multiply a contiguous structured PC $\mathscr{A}$ respecting a linear vtree and an arbitrary contiguous PC $\mathscr{B}$. Since $\mathscr{B}$ is not structured decomposable, we cannot restructure $\mathscr{A}$ to the vtree of $\mathscr{B}$. However, we can use the same idea as Algorithm 1 to restructure $\mathscr{A}$ "on-the-fly" as we multiply it with $\mathscr{B}$ in a bottom-up way. Specifically, for each possible scope $\mathbf{X}_{a:b}$ that appears in $\mathscr{B}$, we recursively construct circuit representations for the functions $p_q(\mathbf{X}_{a:b}) \cdot p_{\mathscr{A}}(\mathbf{X}_{a:b} \mid Z_a = i, Z_b = j)$ for i, j and $q \in \mathscr{B}$ with scope $\mathbf{X}_{a:b}$. The recurrence relation is similar to that of Algorithm 1 and we refer readers to the Appendix for details.

As an explicit application of circuit multiplication, let us consider constrained text generation (Zhang et al. 2024), in which linear PCs (HMMs) are multiplied with deterministic finite automata (DFAs) representing the logical constraint. Our results imply that one can also multiply a HMM with a contiguous logical circuit such as a sentential decision diagram (SDD) (Darwiche 2011), which have been shown to be exponentially more expressive efficient (Bova 2016).

E Depth Reduction

We now prove Theorem 4.4.

Theorem 4.4. *Any structured PC over n variables and with hidden state size h can be restructured to a structured PC*

of depth $O(\log n)$ and size $O(nh^3)$ that represents the same distribution.

Proof. We begin by showing the following results regarding Algorithm 3 for constructing a labelled vtree to which the circuit is to be restructured:

Proposition E.1 (Depth Reduction Vtree)**.** *Given any vtree V, Algorithm 3 returns a vtree W of depth $O(\log |V|)$ with a valid labelling of cardinality 3.*

Proof. The depth reduction to $O(\log |V|)$ is achieved as the algorithm increases the depth by one in each recursive call, but reduces the vtree size by a multiplicative factor. The validity condition holds due to the separation into connected components (the labels can also be obtained from Algorithm 2). The value of M' follows by noting that $\mathbf{S}_l$ and $\mathbf{S}_r$ are either empty or singleton sets, and that the algorithm produces $\mathbf{C}_w = \mathbf{S}_l \cup \mathbf{S}_r$, $\mathbf{C}_l = \mathbf{S}_l \cup \{Z_{v(w)}\}$, and $\mathbf{C}_r = \{Z_{v(w)}\} \cup \mathbf{S}_l$ where $Z_{v(w)}$ for each inner node $w \in W$. □

By Proposition E.1, given a structured PC $\mathbf{X}$ over n variables with hidden state size h and respecting vtree V, we can generate a vtree W of depth $O(\log n)$ and with labelling cardinality $M' = 3$. Thus, by Theorem 3.3 we can construct a PC representing the same function and respecting vtree W of size $O(nh^3)$. The depth of the PC is then also $O(\log n)$ as we have assumed alternating sum and product nodes, so the depth of the circuit is at most double that of the vtree. □

Since the size of the original circuit is $O(nh^2)$, using the known cubic bound on the size of the depth-reduced circuit (Raz and Yehudayoff 2008) gives $O(n^3h^6)$, while our bound is sub-quadratic. While this result is of independent theoretical interest, the sub-quadratic complexity of $O(nh^3)$ also opens up practical applications of depth-reduction. Almost all PC inference and learning algorithms involve forward/backward passes through the computation graph, where computation is only parallelized across nodes of the same depth such that $O(\text{depth of PC})$ sequential computations are required. This is problematic when the number of variables n is large, as is often the case in applications such as computational biology (Dang et al. 2022). In such cases, depth reduction can be a practical strategy where the improved parallelism outweighs the increased circuit size.

F Related Work

Probabilistic circuits have emerged as a unifying representation of tractable probabilistic models (Choi, Vergari, and Van den Broeck 2020; Sidheekh and Natarajan 2024), such as sum-product networks (Poon and Domingos 2011), cutset networks (Rahman, Kothalkar, and Gogate 2014), and probabilistic sentential decision diagrams (Kisa et al. 2014). Significant effort has been devoted to learning PC structures to fit data (Liang, Bekker, and Van den Broeck 2017; Dang, Vergari, and Broeck 2020; Yang, Gala, and Peharz 2023), but the implications for the structure-dependent queries have been less studied. We bridge this gap by providing a general restructuring algorithm with specific cases of (quasi-)polynomial complexity.

As tractable representations of distributions, PCs have been employed extensively as a compilation target for inference in graphical models (Darwiche 2003; Chavira and Darwiche 2008; Rooshenas and Lowd 2014). Hidden tree-structured Bayesian networks have also been used as a starting point for the learning of a probabilistic circuit (Dang, Vergari, and Broeck 2020; Liu and Van den Broeck 2021; Dang, Liu, and Van den Broeck 2022). A particularly useful analysis technique for learning probabilistic circuits has been to interpret them as latent variable models (Peharz et al. 2016). Decomposable and smooth PCs can be interpreted as Bayesian networks by introducing a latent variable for each sum node in the PC (Zhao, Melibari, and Poupart 2015). Our conversion from structured PC to Bayesian network is most closely related to the *decompilation* methods of Butz, Oliveira, and Peharz (2020); Papantonis and Belle (2023), but we do not assume the PC has been compiled from a Bayesian network.

The seminal work of Valiant et al. (1983) showed that any poly-size arithmetic circuit can be transformed into an equivalent circuit of polylogarithmic depth. Raz and Yehudayoff (2008) show that this procedure maintains syntactic multilinearity (decomposability). Recently, Yin and Zhao (2024) showed a quasipolynomial upper bound on converting decomposable and smooth PCs to tree-shaped PCs via a depth-reduction procedure. Our application of restructuring focuses on structured-decomposable circuits and shows a tighter bound based on a graphical model interpretation.