Restructuring Tractable Probabilistic Circuits

Honghua Zhang*
University of California, Los Angeles

Benjie Wang*
University of California, Los Angeles

Marcelo Arenas
Pontificia Universidad Católica de Chile

Guy Van den Broeck
University of California, Los Angeles

Abstract

Probabilistic circuits (PCs) are a unifying representation for probabilistic models that support tractable inference. Numerous applications of PCs like 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 decomposes 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 decomposibility. 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

A key challenge in deep generative modeling is the intractability of probabilistic reasoning (Roth, 1996; Geh et al., 2024). To address this challenge, probabilistic circuits (PCs) (Darwiche, 2003; Poon and Domingos, 2011; Choi et al., 2020) has emerged as a uni-

fying representation of *tractable* generative models for high-dimensional probability distributions. In particular, PCs support efficient and exact evaluation of various inference queries such as marginalization. The tractability of PCs has proven crucial in various applications, such as causal inference (Zečević et al., 2021; Wang and Kwiatkowska, 2023; Busch et al., 2024), knowledge graph learning (Loconte et al., 2023) and ensuring fairness in decision making (Choi et al., 2021).

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 particular (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 answering various inference queries (Vergari et al., 2021), such as conditioning on logical constraints (Choi et al., 2015; Ahmed et al., 2022; Liu et al., 2024b; Zhang et al., 2023, 2024), computing expected predictions of classifiers (Khosravi et al., 2019) and causal backdoor adjustment (Wang and Kwiatkowska, 2023), as well as in improving the expressive power of circuits through squaring (Loconte et al., 2024c,b; Wang and Van den Broeck, 2024). 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). Recent PC implementations have focused on layer-wise parallelization of PC inference via modern GPUs, and depth reduction enables greater parallelization (Peharz et al., 2020; Dang et al., 2021; Liu et al., 2024a; Loconte et al., 2024a). In this work, 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. This opens up new possibilities of practically implementing depth reduction techniques to speed up PC inference.

In summary, our key contributions are: (1) we identify the *restructuring* problem, which concerns the computational complexity of converting between different tractable PC structures; (2) we present a general algorithm for mapping between any two structures (Section 3); (3) we show that restructuring is possible in (quasi-)polynomial complexity for a class of *contiguous* structures (Section 4); and (4) we illustrate the application of restructuring to two important problems: *multiplying* probability distributions (Section 4) and reducing the *depth* of the model structure (Section 5), where we derive novel results on tractability.

2 PROBABILISTIC CIRCUITS

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

Definition 2.1 (Probabilistic Circuit). A probabilistic circuit (PC) $\mathscr{A} = (\mathcal{G}, \boldsymbol{w})$ 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 $\boldsymbol{w}$. Each node t represents a probability distribution $p_t(\boldsymbol{X})$, defined recursively by:

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

where $f_t(\boldsymbol{x})$ is a univariate input distribution function (e.g. Gaussian, Categorical), we use $\text{ch}(t)$ 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 $\sum_{c \in \text{ch}(t)} w_{t,c} = 1$ 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 $p_{\mathscr{A}}(\boldsymbol{x})$, 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 factorized product of its child distributions, while sum nodes represent a weighted mixture of its 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*, that requires that product nodes with the same scope decompose in the same way.

Definition 2.3 (Vtree). A vtree V over variables $\boldsymbol{X}$ 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 $\boldsymbol{X}_v$, satisfying $\boldsymbol{X}_v = \boldsymbol{X}_l \cup \boldsymbol{X}_r$ 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 $t \in \mathscr{A}$ decomposes its scope according to some inner vtree node $v \in V$.

The main advantage of structured decomposability is that it enables tractable circuit 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 to respect 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).

3.1 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 et al., 2020; Liu and Van den Broeck, 2021). In this subsection, we describe how to go in the opposite direction, i.e. 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 $\boldsymbol{X}$ respecting vtree V. Given a vtree node $v \in V$, we write $\mathrm{prod}(v)$ to denote the set of all product nodes with scope $\boldsymbol{X}_v$. We define the *hidden state size* h of the circuit to be $\max_{v \in V} |\mathrm{prod}(v)|$. Writing n for the number of variables, the size of the circuit is then $O(nh^2)$.[1]

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 $t \in \mathrm{prod}(v)$ with a unique index $\mathrm{idx}(t) \in \{0, ..., |\mathrm{prod}(v)|-1\}$, also writing $t_{v,i}$ to refer to the product node with index i in $\mathrm{prod}(v)$. Then we can introduce a categorical latent variable Z_v whose value corresponds to a particular product node in $\mathrm{prod}(v)$:

Definition 3.1 (Augmented PC). Given a structured-decomposable and smooth PC $\mathscr{A}$ over variables $\boldsymbol{X}$ respecting vtree V, we define the augmented PC $\mathscr{A}_{\mathrm{aug}}$ 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 $t \in \mathrm{prod}(v)$ that is a leaf node with scope Z_v and leaf function $f_{t_{\mathrm{aug}}}(Z_v) = \mathbb{1}_{Z_v = \mathrm{idx}(t)}$.

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

Proposition 3.2. $p_{\mathscr{A}}(\boldsymbol{X}) = \sum_{\boldsymbol{z}} p_{\mathscr{A}_{aug}}(\boldsymbol{X}, \boldsymbol{z})$

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

---

[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.
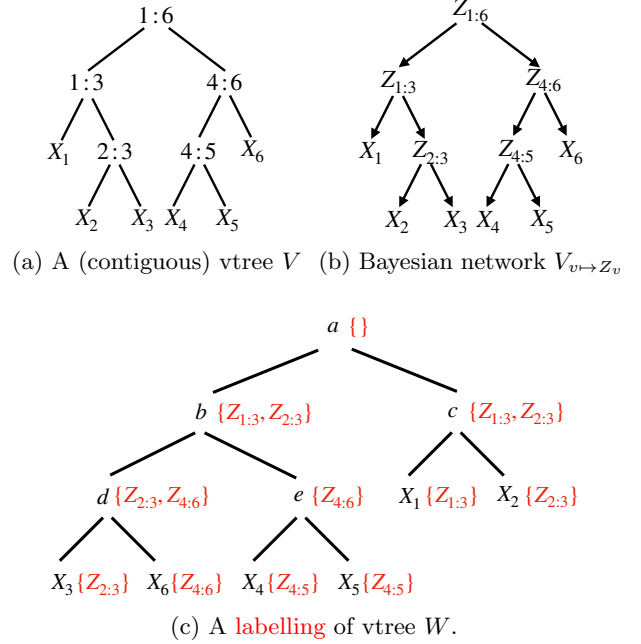

(a) A (contiguous) vtree V (b) Bayesian network $V_{v \mapsto Z_v}$

(c) A labelling of vtree W.

Figure 1: Fig. 1a shows a vtree V for some contiguous 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 $G_{\mathscr{A}}$.

$p^*(\boldsymbol{X}, \boldsymbol{Z})$, based on the augmented PC, that factorizes as required by the Bayesian network structure. There are three cases to consider: (i) the root node $p^*(Z_{\mathrm{ROOT}(V)})$, (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. $p_{\mathscr{A}_{\mathrm{aug}}} = p^*$. The intuitive idea is that each value of $\boldsymbol{Z}$ corresponds to a subtree of $\mathscr{A}_{\mathrm{aug}}$, whose value is precisely given by the product of weights and leaf functions specified by the Bayesian network; we refer readers to the Appendix for the complete proof. We thus have the following mapping from structured PCs to tree-shaped Bayesian networks:

Theorem 3.3. *Let $\mathscr{A}$ be a structured-decomposable and smooth PC over variables $\boldsymbol{X}$ respecting vtree V.*

Then there exists a Bayesian network $G_{\mathscr{A}}$ over variables $\boldsymbol{X}$ and $\boldsymbol{Z} = \{Z_v | v \in V\}$ with graph $V_{v \mapsto Z_v}$ such that $\sum_z p_G(\boldsymbol{X}, \boldsymbol{z}) = p_{\mathscr{A}}(\boldsymbol{X})$.

Since we have shown that $p_{\mathscr{A}}$ and p_G represents the same distribution over the observed variables $\boldsymbol{X}$, we will drop the subscripts when there is no ambiguity.

3.2 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 $\boldsymbol{C}_w \subseteq G_{\mathscr{A}}$ such that $\boldsymbol{X}_w$ is conditionally independent from $\boldsymbol{X} \setminus \boldsymbol{X}_w$ given $\boldsymbol{C}_w$. To characterize such properties, we introduce *covers*:

Definition 3.4 (Blocked Path). Given a directed rooted tree, we say that a path P is blocked by a set of nodes $\boldsymbol{S}$ if $P \cap \boldsymbol{S} \neq \varnothing$.

Definition 3.5 (Cover). Given a tree-shaped Bayesian network $G_{\mathscr{A}}$ as constructed in Sec. 3.1, we say that $\boldsymbol{C} \subseteq \boldsymbol{Z}$ *covers* $\boldsymbol{S} \subseteq \boldsymbol{X}$ if $\boldsymbol{C}$ blocks all paths between $\boldsymbol{S}$ and $\boldsymbol{X} \setminus \boldsymbol{S}$ in $G_{\mathscr{A}}$.

Our definition of cover is a special case of *d-separation* (Geiger et al., 1990), which characterizes conditional independence for Bayesian networks:

Proposition 3.6 (Geiger et al. (1990)). *$\boldsymbol{A}, \boldsymbol{B} \subseteq G_{\mathscr{A}}$ are conditionally independent given $\boldsymbol{C} \subseteq G_{\mathscr{A}}$ if and only if $\boldsymbol{C}$ blocks all paths between $\boldsymbol{A}$ and $\boldsymbol{B}$.[2] In particular, if $\boldsymbol{C}$ covers $\boldsymbol{S}$ then $\boldsymbol{S}$ and $\boldsymbol{X} \setminus \boldsymbol{S}$ are conditionally independent given $\boldsymbol{C}$.*

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

Definition 3.7 (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 $\boldsymbol{C}_w \subseteq G_V$ s.t.

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

Furthermore, w.l.o.g., we set $\boldsymbol{C}_{\text{root of } W} := \varnothing$ and $\boldsymbol{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.

[2] Defining blocked paths for DAGs requires considering *colliders*, which do not occur in directed rooted trees.
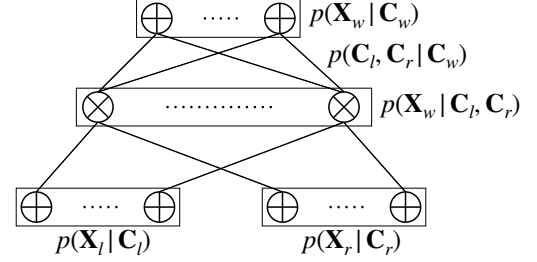

Figure 2: Recursive construction of vectors of sum nodes representing $p(\boldsymbol{X}_w \,|\, \boldsymbol{C}_w)$

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, $p(X_j \,|\, \boldsymbol{C}_{X_j}) = p(X_j \,|\, \text{parent of } X_j \text{ in } G_{\mathscr{A}})$, 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:

$$p(\boldsymbol{X}_w \,|\, \boldsymbol{C}_w)$$
$$= \sum_{(\boldsymbol{C}_l \cup \boldsymbol{C}_r) \setminus \boldsymbol{C}_w} p(\boldsymbol{X}_l, \boldsymbol{X}_r \,|\, \boldsymbol{C}_l, \boldsymbol{C}_r) \cdot p(\boldsymbol{C}_l, \boldsymbol{C}_r \,|\, \boldsymbol{C}_w)$$
$$= \sum_{(\boldsymbol{C}_l \cup \boldsymbol{C}_r) \setminus \boldsymbol{C}_w} p(\boldsymbol{X}_l \,|\, \boldsymbol{C}_l) \cdot p(\boldsymbol{X}_r \,|\, \boldsymbol{C}_r) \cdot p(\boldsymbol{C}_l, \boldsymbol{C}_r \,|\, \boldsymbol{C}_w)$$

Here the first step follows from Property 2 and 3, and the second step follows from all properties in Defintion 3.7. The circuit materialization of the recurrence relation is shown in Figure 2. Note that if w is the root, then $p(\boldsymbol{X}_w \,|\, \boldsymbol{C}_w)$ becomes $p(\boldsymbol{X})$, which is a single sum node representing the distribution of $\mathscr{A}$. The complete recursion is given by Algorithm 1.

Algorithm 1 Construct PC with respect to W

---

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

Theorem 3.8. *Let h be the number of hidden states of the original PC $\mathscr{A}$ and n the number of random variables. The number of hidden states of the restructured PC is given by $O(h^M)$ where $M = \max_{w \in W} |\boldsymbol{C}_l \cup \boldsymbol{C}_r|$ and the size of the restructured PC is bounded by $O(nh^{M'})$ where $M' = \max_{w \in W} |\boldsymbol{C}_l \cup \boldsymbol{C}_r \cup \boldsymbol{C}_w| \leq 2M$. We refer to M' as the cardinality of the labelling $\boldsymbol{C}_w$.*

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 $O(h^{|\boldsymbol{C}_l \cup \boldsymbol{C}_r|})$; we set $M := \max_{w \in W} |\boldsymbol{C}_l \cup \boldsymbol{C}_r|$ 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 $O(h^{|\boldsymbol{C}_l \cup \boldsymbol{C}_r \cup \boldsymbol{C}_w|})$ and the number of product edges is given by $O(h^{|\boldsymbol{C}_l \cup \boldsymbol{C}_r|})$; since there are $O(n)$ vtree nodes in total, the total number of edges in $\mathscr{B}$ is given by $O(nh^{M'})$, with $M' = \max_{w \in W} |\boldsymbol{C}_l \cup \boldsymbol{C}_r \cup \boldsymbol{C}_w|$. $\square$

Remark 3.9. By Theorem 3.8, the restructured PC $\mathscr{A}'$ has hidden state size $O(h^M)$, which gives a circuit of size $\Theta(nh^{2M})$ only if $\mathscr{A}'$ is densely connected. In fact, we will show in Section 4 and 5 that the restructured PCs are often sparsely connected, resulting in sizes much smaller than $O(nh^{2M})$. Thus, while the graphical model representation is useful for reasoning about conditional independencies, the circuit representation allows us to visualize and exploit the sparsity for efficient inference (Dang et al., 2022a; Liu et al., 2024a).

3.3 Computing Vtree Labelling

The next question that immediately arises is how to compute a valid labelling for W with respect to $G_{\mathscr{A}}$. One naive solution is to set $\boldsymbol{C}_w$ to be $\boldsymbol{Z}$, the set of all latent variables in $G_{\mathscr{A}}$. However, this is not desirable as $M' = \max_{w \in W} |\boldsymbol{C}_l \cup \boldsymbol{C}_r \cup \boldsymbol{C}_w| = |\boldsymbol{Z}| = n-1$, resulting in the restructured circuit having exponential size $O(nh^{n-1})$. Hence we present a greedy approach that computes a labelling while trying to minimize M'.

The algorithm proceeds top-down on W. For the base case where w is the root, we set $\boldsymbol{C}_w := \varnothing$. For the inductive step, let l and r be the children of w and assume that we have computed $\boldsymbol{C}_w$ as a cover for $\boldsymbol{X}_w$ in $G_{\mathscr{A}}$: we (1) split $G_{\mathscr{A}}$ into connected components $\{G_i\}$ via $\boldsymbol{C}_w$; then (2) within each connected component G_i, we compute a minimum d-separator $\boldsymbol{C}_i$ that blocks all paths between $\boldsymbol{X}_l \cap G_i$ and $\boldsymbol{X}_r \cap G_i$ by calling the sub-routine MINIMUMSEPARATOR. We set $\boldsymbol{D}_w := (\bigcup_i \boldsymbol{C}_i) \cup \boldsymbol{C}_w$ and observe that $\boldsymbol{D}_w$ covers both $\boldsymbol{X}_l$ and $\boldsymbol{X}_r$ in $G_{\mathscr{A}}$. To compute $\boldsymbol{C}_l$, similarly for $\boldsymbol{C}_r$, we consider all paths starting from $\boldsymbol{X}_l$ and stopping immediately when reaching some $Z_j \in \boldsymbol{D}_w$, and we let $\boldsymbol{C}_l$ to be the set containing all such Z_js. The pseudo code is shown in Algorithm 2. Note that the MINIMUMSEPARATOR procedure called in Algorithm 2 computes a minimum d-separator that blocks all paths between $\boldsymbol{X}_l$ and $\boldsymbol{X}_r$ in the subgraph G_i. Even though polytime algorithms for computing minimum d-separators exist in literature (Tian et al., 1998), we

---

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

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

---

derive a linear-time algorithm that is easy to implement for our use case, where G_i is a rooted tree with leaves in $\boldsymbol{X}_l$, $\boldsymbol{X}_r$ and $\boldsymbol{C}_w$. We refer readers to the Appendix for details.

Proposition 3.10. *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 $\boldsymbol{C}_w$ computed by Algorithm 2 is valid. Assume that $\boldsymbol{C}_w$ covers $\boldsymbol{X}_w$ in $G_{\mathscr{A}}$, we want to show that $\boldsymbol{C}_l$ and $\boldsymbol{C}_r$ satisfy the properties from Definition 3.7. To prove that $\boldsymbol{C}_l$ covers $\boldsymbol{X}_l$, we consider a path from $X_a \in \boldsymbol{X}_l$ to $X_b \in \boldsymbol{X} \setminus \boldsymbol{X}_l$. (1) If X_a and X_b are in the same G_i, then the path is blocked by $\boldsymbol{C}_i$. (2) If X_a and X_b are in different G_is, then the path contains some node $Z \in \boldsymbol{C}_w$, and we can choose from the path the first $Z \in \boldsymbol{C}_w$. Then $Z \in \boldsymbol{C}_l$ by construction, implying that the path is blocked by $\boldsymbol{C}_l$. Hence we conclude that $\boldsymbol{C}_l$ is a cover for $\boldsymbol{X}_l$, satisfying Property 1. To prove that $\boldsymbol{C}_l$ satisfies Property 2, we argue that because $\boldsymbol{C}_r$ and $\boldsymbol{C}_w$ are both subsets of $\boldsymbol{D}_w$, all paths from $\boldsymbol{X}_l$ to $\boldsymbol{C}_r \cup \boldsymbol{C}_w$ will be blocked by $\boldsymbol{C}_l$ by the way that $\boldsymbol{C}_l$ is constructed. We can show that $\boldsymbol{C}_r$ satisfies Property 1 and 3 by the same argument. $\square$

As an example, we (partially) illustrate the application of Algorithm 2 to the target vtree in Figure 1c. For node a and its children b, c, the only connected component is $G_1 := G_{\mathscr{A}}$. A minimum separator of $\boldsymbol{X}_b = \{X_3, X_6, X_4, X_5\}$ and $\boldsymbol{X}_c = \{X_1, X_2\}$ is then given by $\boldsymbol{C}_1 := \{Z_{1:3}, Z_{2:3}\}$. Then $\boldsymbol{D}_1 = \boldsymbol{C}_1 \cup \boldsymbol{C}_a = \{Z_{1:3}, Z_{2:3}\}$ covers both $\boldsymbol{X}_b, \boldsymbol{X}_c$. Tracing the paths from each X variable to $\boldsymbol{D}_1$ finally gives $\boldsymbol{C}_b = \boldsymbol{C}_c = \{Z_{1:3}, Z_{2:3}\}$.

Though Algorithm 2 computes a valid labelling while greedily minimizing $|\boldsymbol{C}_l \cup \boldsymbol{C}_r \cup \boldsymbol{C}_w|$, we do not know whether $M' = \max_{w \in W} |\boldsymbol{C}_l \cup \boldsymbol{C}_r \cup \boldsymbol{C}_w|$ is globally minimized or not. In addition, we hypothesize that if we can find a minimum vtree labelling, then the size of the PC constructed by Algorithm 1 is optimal. We leave it as an open problem to design an algorithm that computes *minimum* labellings and prove the optimality of Algorithm 1 given a minimum labelling.

Nonetheless we show that Algorithm 1 yields novel polynomial-time algorithms for the tasks of PC multiplication and depth-reduction. Specifically, we show that for important subclasses of PCs, we *can* compute vtree labellings of constant or $O(\log n)$ cardinality. We refer readers to Section 4 and Section 5 for details.

3.4 Corollaries

With our restructuring algorithm in hand, we now examine the restructuring of two other types of circuits: namely, deterministic PCs, and logical circuits.

Definition 3.11 (Determinism)**.** A sum node is deterministic if for every value $\boldsymbol{x}$ of $\boldsymbol{X}$, at most one child c returns a non-zero value (i.e. $p_c(\boldsymbol{x}) > 0$). 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 et al., 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 3.12. *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[3] (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 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).

---

[3]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).
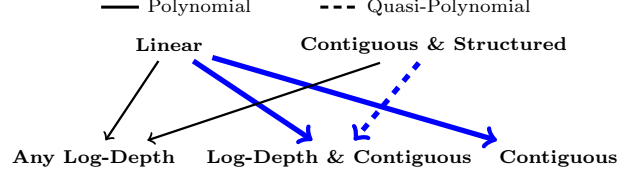

Figure 3: Summary of restructuring results; the top/bottom rows indicate the source/target structures. **Blue and bold** arrows indicate novel complexity results. For the two other arrows that were known to be polynomial-time, our approach yields more efficient algorithms amenable to practical implementation.

4 PC MULTIPLICATION

One important application of restructuring PCs is circuit multiplication: given two PCs $\mathscr{A}$ and $\mathscr{B}$, the goal is to construct a tractable PC $\mathscr{C}$ such that $p_{\mathscr{C}}(\boldsymbol{x}) \propto p_{\mathscr{A}}(\boldsymbol{x}) \cdot p_{\mathscr{B}}(\boldsymbol{x})$. PC multiplication was previously only addressed for structured PCs respecting the *same* vtree (Shen et al., 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. Further, we will show how "on-the-fly" restructuring can enable circuit multiplication even when one of the circuits is not structured-decomposable.

We start by introducing a new structural property of tractable PCs called *contiguity*.[4]

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 $1 \le a \le b \le n$. A smooth and decomposable PC is contiguous if all of its nodes are contiguous.

Contiguous PCs subsume many widely-used subclasses of PCs such as Hidden Markov Models (HMMs). Note that a contiguous circuit is not necessarily structured-decomposable and $0.5 \otimes p(X_1) \otimes p(X_2, X_3) \oplus 0.5 \otimes p(X_1, X_2) \otimes p(X_3)$ is such an example. The fact that contiguous PCs need not be structured-decomposable also suggests new designs of circuits that can be tractably multiplied. One important example is probabilistic context-free grammars (PCFGs).

Theorem 4.2. *Let G be a PCFG consisting of m production rules. For sequences of length n, G can be rep-*

---

[4]Amarilli et al. (2017) defined a similar concept called *compatible order* in the context of logical circuits.

$$Z_1 \rightarrow Z_2 \rightarrow \cdots \rightarrow Z_n$$
$$\downarrow \quad \downarrow \qquad\qquad \downarrow$$
$$X_1 \quad X_2 \qquad\qquad X_n$$
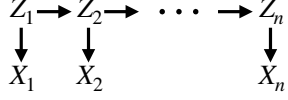
Figure 4: $G_{\mathscr{A}}$ for $\mathscr{A}$ with a linear vtree

resented as a contiguous yet non-structured PC $\mathscr{A}$ of size $O(mn^3)$; i.e. $p_{\mathscr{A}}(\boldsymbol{x})$ computes the probability of $\boldsymbol{x}$ being derived from G, for all $\boldsymbol{x}$ of length n.

The algorithm for constructing the PC representation for a PCFG is in spirit similar to the *CYK algorithm* (Kasami, 1966; Younger, 1967; Cocke, 1969): given a PCFG, for each contiguous subsegment $[a, b]$ of $1, 2, \ldots, n$, and each non-terminal N in the PCFG, we can construct a sum node u (recursively) such that the sub-circuit rooted at u represents the distribution over the set of strings on the segment $[a, b]$ that can be derived from N.

In the following, we consider the problem of multiplying two contiguous PCs $\mathscr{A}$ and $\mathscr{B}$. Intuitively, random variables forming contiguous scopes are more likely to be covered by vtree labellings of small cardinalities, which would give efficient restructuring by Section 3. We start by considering the simpler case where both circuits are structured. We then generalize our results by allowing one of them to be non-structured. Figure 3 summarizes our main results (Theorems 4.4, 4.6, 4.9).

4.1 Multiplying Contiguous Structured PCs

<u>Case 1.</u> For the multiplication of contiguous PCs $\mathscr{A}$ and $\mathscr{B}$, 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 3.1 that the Bayesian network representation for $\mathscr{A}$ is a hidden Markov model (Rabiner, 1989), as shown in Figure 4. By the definition of contiguity, each node $w \in W$ has a scope of the form $\boldsymbol{X}_{a:b} := \{X_a, \ldots, X_b\}$ and we can label it with $\boldsymbol{C}_{a:b} := \{Z_a, Z_{b+1}\}$; in particular, we drop Z_a if $a = 1$ and drop Z_{b+1} if $b = n$.

Claim 4.3. $\boldsymbol{C}_{a:b}$ *is a valid vtree labelling of* W *respecting* $G_{\mathscr{A}}$ *with cardinality* $M' = 3$.

Then it follows from Theorem 3.8 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. Eventually we can compute the product of $\mathscr{A}'$ of $\mathscr{B}$ tractably by the existing algorithm for multiplying two circuits respecting the same vtree (Shen et al., 2016; Vergari et al., 2021).

Theorem 4.4. *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 $O(|\mathscr{A}|^2|\mathscr{B}|)$.

<u>Case 2.</u> Then we 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 3, which is adapted from the segment tree querying algorithm, computes a cover $\boldsymbol{C}_{a:b} \subseteq G_{\mathscr{A}}$ for each contiguous segment $\boldsymbol{X}_{a:b}$. For each $w \in W$, $\boldsymbol{X}_w = \boldsymbol{X}_{a:b}$ for some $1 \leq a \leq b \leq n$ and we set $\boldsymbol{C}_w = \boldsymbol{C}_{a:b} = \text{SEGMENTCOVER}(V, \boldsymbol{X}_{a:b})$.

Algorithm 3 Compute Cover for Segment $\boldsymbol{X}_{a:b}$

---

  procedure SEGMENTCOVER$(v, \boldsymbol{X}_{a:b})$
    if $\boldsymbol{X}_{a:b} = \varnothing$ **then**
      return $\varnothing$
    end if
    if $\boldsymbol{X}_{a:b} = \boldsymbol{X}_v$ **then**
      return $\{Z_v\}$
    end if
    $l, r \leftarrow$ CHILDREN(v)
    $\boldsymbol{L} \leftarrow$ SEGMENTCOVER$(l, \boldsymbol{X}_l \cap \boldsymbol{X}_{a:b})$
    $\boldsymbol{R} \leftarrow$ SEGMENTCOVER$(r, \boldsymbol{X}_r \cap \boldsymbol{X}_{a:b})$
    return $\boldsymbol{L} \cup \boldsymbol{R}$
  end procedure

Proposition 4.5. $\boldsymbol{C}_w$ *is a valid vtree labelling with respect to* $G_{\mathscr{A}}$, *and* $|\boldsymbol{C}_w| \leq 4d$ *for all* w.

In addition to the fact that $\boldsymbol{C}_w$ is a valid labelling, $|\boldsymbol{C}_w| \leq 4d$ follows from the fact that the number of nodes visited at each level of V is at most 4 (see Appendix); hence the cardinality of $\boldsymbol{C}_w$ is bounded by $12d$. Then following a similar analysis, we have the following results for multiplying two contiguous PCs.

Theorem 4.6. *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 4.7. *If* $\mathscr{A}$ *is of depth* $O(\log n)$ *then* $\mathscr{A}$ *and* $\mathscr{B}$ *can be multiplied in quasi-polynomial time.*

Remark 4.8. 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 3.8 can be adapted to such generalized cases thus giving a polynomial-time algorithm for multiplying contiguous structured circuits of constant depths.

4.2 Generalization to Non-structured PCs

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 4.4 and 4.6 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 $\boldsymbol{X}_{a:b}$ that appears in $\mathscr{B}$, we recursively construct circuit representations for the functions $p_q(\boldsymbol{X}_{a:b}) \cdot p_{\mathscr{A}}(\boldsymbol{X}_{a:b} \mid Z_a = i, Z_b = j)$ for i, j and $q \in \mathscr{B}$ with scope $\boldsymbol{X}_{a:b}$. The recurrence relation is similar to that of Algorithm 1 and we refer readers to the Appendix for details.

Theorem 4.9. *Let $\mathscr{A}$ and $\mathscr{B}$ be contiguous PCs with $\mathscr{B}$ not necessarily structured. If $\mathscr{A}$ is structured respecting the linear vtree, then $\mathscr{A}$ and $\mathscr{B}$ can be multiplied in polynomial time and the size of the product PC is bounded by $O(|\mathscr{A}|^2|\mathscr{B}|)$.*

As an explicit application of circuit multiplication, let us consider constrained text generation, in which linear PCs (HMMs) are multiplied with deterministic finite automata (DFAs) representing the logical constraint (Zhang et al., 2024). Our results imply that one can also multiply HMMs with contiguous logical circuits such as a sentential decision diagrams (SDDs) (Darwiche, 2011), which have been shown to be exponentially more expressive efficient (Bova, 2016). It also follows from Theorem 4.2 that we can tractably multiply HMMs with (P)CFGs, subsuming a recent result that HMMs can be tractably marginalized over unambiguous CFGs (Marzouk and de La Higuera, 2022). Our results on multiplying contiguous PCs open up new possibilities for generalizing (Zhang et al., 2024) to much broader families of circuits and constraints.

5 PC 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. Unlike in multiplication, we are not concerned with restructuring to a par-

---

Algorithm 4 Depth Reduction Vtree

1: **procedure** BALANCEDVTREE($V, \boldsymbol{S}_l = \emptyset, \boldsymbol{S}_r = \emptyset$)
2:      **if** $|V| = 1$ **then**
3:          **return** LEAF($V; \boldsymbol{S}_l \cup \boldsymbol{S}_r$)
4:      **end if**
5:      $v \leftarrow$ ROOT(V)
6:      $l, r \leftarrow$ CHILDREN(v)
7:      **while** $|V_r| > \frac{2}{3}|V|$ **do**
8:          $v \leftarrow r$
9:          $l, r \leftarrow$ CHILDREN(v)   ▷ assume $|V_l| \leq |V_r|$
10:     **end while**
11:     $V_l' \leftarrow$ BALANCEDVTREE($V_{[v \mapsto l]}, \boldsymbol{S}_l, \{Z_v\}$)
12:     $V_r' \leftarrow$ BALANCEDVTREE($V_r, \{Z_v\}, \boldsymbol{S}_r$)
13:     **return** JOIN($V_l', V_r'; \boldsymbol{S}_l \cup \boldsymbol{S}_r$)
14: **end procedure**

---

ticular vtree, but rather *any* log-depth vtree. The key step is thus to identify a log-depth vtree such that restructuring to that vtree using Algorithm 1 (and some valid choice of labels) results in at most a polynomial increase in size.

Algorithm 4 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 vtrees for the components. Note that the algorithm produces exactly one vtree node for each vtree node in the original vtree; we can thus write $v(w)$ for the node in V corresponding to w. Then we have the following result:

Theorem 5.1 (Depth Reduction). *Given any vtree V, Algorithm 4 returns a vtree W of depth $O(\log |V|)$ with a valid labelling of cardinality M' at most 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 $\boldsymbol{S}_l$ and $\boldsymbol{S}_r$ are either empty or singleton sets, and that the algorithm produces $\boldsymbol{C}_w = \boldsymbol{S}_l \cup \boldsymbol{S}_r$, $\boldsymbol{C}_l = \boldsymbol{S}_l \cup \{Z_{v(w)}\}$, and $\boldsymbol{C}_r = \{Z_{v(w)}\} \cup \boldsymbol{S}_l$ where $Z_{v(w)}$ for each inner node $w \in W$. $\qquad\square$

Remark 5.2. Firstly, the depth-reduced PC retains structuredness, which is not guaranteed by the existing depth-reduction algorithms. Secondly, exploiting structuredness and tracking the hidden state size enables a more fine-grained analysis of the size of the depth-reduced circuit. 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 $O(n^3h^6)$. However, by Theorem 5.1, we see that $M' = \max_{w \in W} |\boldsymbol{C}_l, \boldsymbol{C}_r, \boldsymbol{C}_W| \leq 3$ and so by Theorem 3.8 we immediately obtain a much tighter bound of $O(nh^3)$ for the size of the resulting circuit.

Corollary 5.3. *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.*

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., 2022b). In such cases, depth reduction can be a practical strategy where the improved parallelism outweighs the increased circuit size.

6 RELATED WORK

Probabilistic circuits have emerged as a unifying representation of tractable probabilistic models (Choi et al., 2020; Sidheekh and Natarajan, 2024), such as sum-product networks (Poon and Domingos, 2011), cutset networks (Rahman et al., 2014), probabilistic sentential decision diagrams (Kisa et al., 2014) and probabilistic generating circuits (Zhang et al., 2021; Harviainen et al., 2023; Agarwal and Bläser, 2024; Broadrick et al., 2024). Significant effort has been devoted to learning PC structures to fit data (Liang et al., 2017; Dang et al., 2020; Yang et al., 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 et al., 2020; Liu and

Van den Broeck, 2021; Dang et al., 2022a). 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 et al., 2015). Our conversion from structured PC to Bayesian network is most closely related to the *decompilation* methods of Butz et al. (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.

7 DISCUSSION

We introduce the problem of *restructuring* probabilistic circuits, and develop a general algorithm for restructuring a structured-decomposable circuit to any target vtree structure. Our method exploits an interpretation of structured-decomposable circuits as latent tree Bayesian networks, which enables recursive construction of a circuit respecting the target vtree using probabilistic semantics of the Bayesian network. As concrete applications of restructuring, we show how to tractably multiply two circuits which do not necessarily share the same structure but satisfy a *contiguity* property, and show how to restructure a circuit to log-depth with a sub-quadratic increase in size.

Our work takes a step towards understanding when and how the space of structured circuits -- hierarchical tensor factorizations (Loconte et al., 2024a) -- 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 representations of high-dimensional tensors and probability distributions.

Acknowledgements

References

Agarwal, S. and Bläser, M. (2024). Probabilistic generating circuits-demystified. In *International Conference on Machine Learning*, pages 329–342. PMLR.

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*.

Amarilli, A., Bourhis, P., Jachiet, L., and Mengel, S. (2017). A Circuit-Based Approach to Efficient Enumeration. In Chatzigiannakis, I., Indyk, P., Kuhn, F., and Muscholl, A., editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 111:1–111:15, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Bova, S. (2016). Sdds are exponentially more succinct than obdds. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Broadrick, O., Zhang, H., and Van den Broeck, G. (2024). Polynomial semantics of tractable probabilistic circuits. In *Proceedings of the 40th Conference on Uncertainty in Artificial Intelligence (UAI)*.

Busch, F. P., Willig, M., Seng, J., Kersting, K., and Dhami, D. S. (2024). Ψnet: Efficient causal modeling at scale. In *International Conference on Probabilistic Graphical Models*, pages 452–469. PMLR.

Butz, C., Oliveira, J. S., and Peharz, R. (2020). Sum-product network decompilation. In *International Conference on Probabilistic Graphical Models*, pages 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., Dang, M., and Van den Broeck, G. (2021). Group fairness by probabilistic modeling with latent fair decisions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12051–12059.

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

Cocke, J. (1969). *Programming languages and their compilers: Preliminary notes*. New York University.

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., Khosravi, P., Liang, Y., Vergari, A., and Van den Broeck, G. (2021). Juice: A julia package for logic and probabilistic circuits. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (Demo Track)*.

Dang, M., Liu, A., and Van den Broeck, G. (2022a). 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. (2022b). 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*, pages 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, pages 205–215.

Geh, R. L., Zhang, H., Ahmed, K., Wang, B., and Van den Broeck, G. (2024). Where is the signal in tokenization space? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

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

Harviainen, J., Ramaswamy, V. P., and Koivisto, M. (2023). On inference and learning with probabilistic generating circuits. In *Uncertainty in Artificial Intelligence*, pages 829–838. PMLR.

Kasami, T. (1966). An efficient recognition and syntax-analysis algorithm for context-free languages. *Coordinated Science Laboratory Report no. R-257*.

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., Ahmed, K., and Van den Broeck, G. (2024a). Scaling tractable probabilistic circuits: A systems perspective. In *Proceedings of the 41th International Conference on Machine Learning (ICML)*.

Liu, A., Niepert, M., and Van den Broeck, G. (2024b). 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., Di Mauro, N., Peharz, R., and Vergari, A. (2023). How to turn your knowledge graph embeddings into generative models. *Advances in Neural Information Processing Systems*, 36.

Loconte, L., Mari, A., Gala, G., Peharz, R., de Campos, C., Quaeghebeur, E., Vessio, G., and Vergari, A. (2024a). What is the relationship between tensor factorizations and circuits (and how can we exploit it)? *arXiv preprint arXiv:2409.07953*.

Loconte, L., Mengel, S., and Vergari, A. (2024b). Sum of squares circuits. *arXiv preprint arXiv:2408.11778*.

Loconte, L., Sladek, A., Mengel, S., Trapp, M., Solin, A., Gillis, N., and Vergari, A. (2024c). Subtractive mixture models via squaring: Representation and learning. In *International Conference on Learning Representations (ICLR)*.

Marzouk, R. and de La Higuera, C. (2022). Marginal inference queries in hidden markov models under context-free grammar constraints. *arXiv preprint arXiv:2206.12862*.

Papantonis, I. and Belle, V. (2023). Transparency in sum-product network decompilation. In *European Conference on Artificial Intelligence*, pages 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.

Peharz, R., Lang, S., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Van den Broeck, G., Kersting, K., and Ghahramani, Z. (2020). Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*.

Pipatsrisawat, K. and Darwiche, A. (2008). New compilation languages based on structured decomposability. In *AAAI*, volume 8, pages 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)*, pages 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*, pages 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*, pages 710–718. PMLR.

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

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*.

Tian, J., Paz, A., and Pearl, J. (1998). *Finding minimal d-separators*. Citeseer.

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*, pages 9488–9498. PMLR.

Wang, B. and Van den Broeck, G. (2024). On the relationship between monotone and squared probabilistic circuits. In *Proceedings of the UAI Workshop on Tractable Probabilistic Modeling (TPM)*.

Yang, Y., Gala, G., and Peharz, R. (2023). Bayesian structure scores for probabilistic circuits. In *International Conference on Artificial Intelligence and Statistics*, pages 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)*.

Younger, D. H. (1967). Recognition and parsing of context-free languages in time n3. *Information and control*, 10(2):189–208.

Zečević, M., Dhami, D., Karanam, A., Natarajan, S., and Kersting, K. (2021). Interventional sum-product networks: Causal inference with tractable probabilistic models. *Advances in neural information processing systems*, 34:15019–15031.

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., Juba, B., and Van den Broeck, G. (2021). Probabilistic generating circuits. In *Proceedings of the 38th 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*, pages 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.

Supplementary Materials

A ADDITIONAL PROOFS

Proposition 3.2. $p_{\mathscr{A}}(\boldsymbol{X}) = \sum_{\boldsymbol{z}} p_{\mathscr{A}_{aug}}(\boldsymbol{X}, \boldsymbol{z})$

Proof. Suppose that $\mathscr{A}$ is a structured decomposable and smooth PC respecting vtree V. Write prod(v) and sum(v) for the set of product and sum nodes with scope $\boldsymbol{X}_v$. The augmented PC $\mathscr{A}_{\mathrm{aug}}$ 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 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 sum(v), hence by smoothness of $\mathscr{A}$ all sum nodes contain some node in sum(v) and thus some leaf with scope $\{Z_v\}$.

Consider the standard marginalization algorithm for PCs (Darwiche, 2003; Choi et al., 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 $\boldsymbol{Z}$, it is immediate that the resulting PC represents the same function as $\mathscr{A}$. □

Theorem 3.3. *Let $\mathscr{A}$ be a structured-decomposable and smooth PC over variables $\boldsymbol{X}$ respecting vtree V. Then there exists a Bayesian network $G_{\mathscr{A}}$ over variables $\boldsymbol{X}$ and $\boldsymbol{Z} = \{Z_v | v \in V\}$ with graph $V_{v \mapsto Z_v}$ such that $\sum_{\boldsymbol{z}} p_G(\boldsymbol{X}, \boldsymbol{z}) = p_{\mathscr{A}}(\boldsymbol{X})$.*

Proof. In Section 3.1 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. $p_G(\boldsymbol{X}, \boldsymbol{Z}) = p_{\mathscr{A}_{\mathrm{aug}}}(\boldsymbol{X}, \boldsymbol{Z})$.

The key observation is to consider the *induced trees* of the augmented PC (Zhao et al., 2016):

Definition A.1 (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) 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 $\mathscr{A}_{\mathrm{aug}}$. Each induced tree defines a function:

$$p_{\mathscr{A}_{\mathrm{aug}}, T}(\boldsymbol{X}, \boldsymbol{Z}) := \prod_{(t_i, t_j) \in \mathrm{SUMEDGES}(T)} w_{t_i, t_j} \prod_{t \in \mathrm{LEAVES}_{\boldsymbol{X}}(T)} f_t(X_{\mathrm{sc}(t)}) \prod_{t \in \mathrm{LEAVES}_{\boldsymbol{Z}}(T)} f_t(Z_{\mathrm{sc}(t)}) \qquad (1)$$

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

Proposition A.2 (Zhao et al. (2016)). $p_{\mathscr{A}_{aug}}(\boldsymbol{X}, \boldsymbol{Z}) = \sum_{T \in \mathcal{T}} p_{\mathscr{A}_{aug}, T}(\boldsymbol{X}, \boldsymbol{Z})$.

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 $\boldsymbol{X}_v$ 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 $\boldsymbol{x}$ of the observed variables, we have that $p_G(\boldsymbol{x}, \boldsymbol{z}) = 0$ 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 $\boldsymbol{z}$ 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 Equation 1 then contains a leaf function $f_t(z_v) = \mathbb{1}_{z_v = i} = 0$. Thus $p_{\mathscr{A}_{\text{aug}}}(\boldsymbol{x}, \boldsymbol{z}) = \sum_{T \in \mathcal{T}} p_{\mathscr{A}_{\text{aug}}, T}(\boldsymbol{x}, \boldsymbol{z}) = 0$ for any $\boldsymbol{x}$.

If an assignment $\boldsymbol{z}$ 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 $\boldsymbol{X}_v$, 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. $\qquad\square$

Proposition 4.5. $\boldsymbol{C}_w$ *is a valid vtree labelling with respect to* $G_{\mathscr{A}}$*, and* $|\boldsymbol{C}_w| \leq 4d$ *for all* w*.*

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

1. $X_{a:b} = \bigcup_{Z_i \in \boldsymbol{C}_{a:b}} \text{LEAVES}(Z_i)$ is a *disjoint* union.
2. If $a \leq c \leq d \leq b$, then for $Z \in \boldsymbol{C}_{c:d}$, there exists $Z' \in \boldsymbol{C}_{a:b}$ 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 $\boldsymbol{C}_{c:d}$ via $\bigcup_{Z_i \in \boldsymbol{C}_{a:b}} \text{SEGMENTCOVER}(Z_i, \boldsymbol{X}_{c:d} \cap \text{LEAVES}(Z_i))$. 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 $\boldsymbol{C}_{a:b}$ covers $\boldsymbol{X}_{a:b}$ and $\boldsymbol{C}_{a:c}$ blocks all paths from $\boldsymbol{X}_{a:c}$ to $\boldsymbol{C}_{c+1:b}$; it follows from Property 2 that $\boldsymbol{C}_{a:c}$ blocks all paths from $\boldsymbol{X}_{a:c}$ to $\boldsymbol{X}_{a:b}$. Hence we conclude that $\boldsymbol{C}_w$ is a valid labelling. A minor catch is that $\boldsymbol{C}_w$ may contain variables in $\boldsymbol{X}$, but we can replace them by their parent in $G_{\mathscr{A}}$ without affecting the validity of $\boldsymbol{C}_w$.

Now we show that $|\boldsymbol{C}_w| \leq 4d$ and it suffices to show that for each layer of the vtree V, the number of nodes v visited by Algorithm 3 is at most 4. We prove this by a top-down induction. For the base case, the root node is the only node in the first layer hence the number of nodes visited is 1. Now assume that for the k-th layer, the number of nodes visited is at most 4, and denote them by v_1, v_2, v_3, v_4, ordered from "left" to "right". Note that $\{v_i\}_{1 \leq i \leq 4}$ must form a contiguous segment with v_2 and v_3 completely covered by $\boldsymbol{X}_{a:b}$; hence the function call on v_2 and v_3 would return without further recursive calls. For the function calls on v_1 and v_2, suppose both of them trigger recursive calls, in the worse case scenario, each would trigger at most two recursive calls; hence, the number of nodes visited in the $(k+1)$th layer is at most 4. Since there are d layers in total, the total number of nodes returned is at most $4d$. $\qquad\square$

Corollary 5.3. *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. By Theorem 5.1, given a structured PC $\boldsymbol{X}$ 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 3.8 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. $\qquad\square$

B   Computing Minimum D-separators

Let G be a tree-shaped Bayesian network rooted at Z; in particular, assume that the leaves of $G \subseteq \boldsymbol{X} \cup \boldsymbol{Z}$ and the internal nodes of $G \subseteq \boldsymbol{Z}$ (e.g. Figure 1b). Then, we want to prove that Algorithm 5 computes a minimum d-separator $\boldsymbol{C} \subseteq G$ for $\boldsymbol{A}, \boldsymbol{B} \subseteq \boldsymbol{X}$.

Algorithm 5 Computing minimum d-separators for tree-shaped Bayesian network G rooted at Z

---

procedure MINIMUMSEPARATOR(Z, $\boldsymbol{A}$, $\boldsymbol{B}$)
    if $\boldsymbol{A} = \varnothing$ and $\boldsymbol{B} = \varnothing$ **then**
        return $\varnothing, \varnothing, \varnothing$
    end if
    if $\boldsymbol{B} = \varnothing$ **then**
        return $\{Z\}, \varnothing, \varnothing$
    end if
    if $\boldsymbol{A} = \varnothing$ **then**
        return $\varnothing, \{Z\}, \varnothing$
    end if
    for $Z_i \in$ CHILDREN(Z) **do**
        $\boldsymbol{C}_{i,\boldsymbol{A}}, \boldsymbol{C}_{i,\boldsymbol{B}}, \boldsymbol{C}_i \leftarrow$ MINIMUMSEPARATOR(
            $Z_i, \boldsymbol{A} \cap$ LEAVES(Z_i), $\boldsymbol{B} \cap$ LEAVES(Z_i))
    end for
    $\boldsymbol{C_A} \leftarrow$ MIN($\bigcup_i \boldsymbol{C}_i \cup \{Z\}, \bigcup_i \boldsymbol{C}_{i,\boldsymbol{A}}$)
    $\boldsymbol{C_B} \leftarrow$ MIN($\bigcup_i \boldsymbol{C}_i \cup \{Z\}, \bigcup_i \boldsymbol{C}_{i,\boldsymbol{B}}$)
    $\boldsymbol{C} \leftarrow$ MIN($\boldsymbol{C_A}, \boldsymbol{C_B}$)
    return $\boldsymbol{C_A}, \boldsymbol{C_B}, \boldsymbol{C}$
end procedure

As shown in Algorithm 5, given tree-shaped Bayesian network G rooted at Z, the procedure MINIMUMSEPARTOR computes three sets of latent variables $\boldsymbol{C_A}$, $\boldsymbol{C_B}$ and $\boldsymbol{C}$. Specifically, we shall prove the following properties:

1. $\boldsymbol{C_A}$ is a minimum d-separator between $\boldsymbol{A}$ and $\boldsymbol{B}$ that also blocks all paths from $\boldsymbol{A}$ to Z.
2. $\boldsymbol{C_B}$ is a minimum d-separator between $\boldsymbol{A}$ and $\boldsymbol{B}$ that also blocks all paths from $\boldsymbol{B}$ to Z.
3. Either $\boldsymbol{C_A}$ or $\boldsymbol{C_B}$ is a minimum d-separator between $\boldsymbol{A}$ and $\boldsymbol{B}$ in G (rooted at Z); hence $\boldsymbol{C}$ is a minimum d-separator between $\boldsymbol{A}$ and $\boldsymbol{B}$ in G.

Proof of Property 3. It suffices to show that $P :=$ {d-separators between $\boldsymbol{A}$ and $\boldsymbol{B}$} and $Q :=$ {d-separators between $\boldsymbol{A}$ and $\boldsymbol{B}$ that blocks all paths from $\boldsymbol{A}$ to Z} $\cup$ {d-separators between $\boldsymbol{A}$ and $\boldsymbol{B}$ that blocks all paths from $\boldsymbol{B}$ to Z} are the same set. It is obvious that $Q \subseteq P$ and let's prove that $P \subseteq Q$. Let $\boldsymbol{C}$ be a d-separator between $\boldsymbol{A}$ and $\boldsymbol{B}$ in G, then $\boldsymbol{C}$ either blocks all path from $\boldsymbol{A}$ to Z or blocks all path from $\boldsymbol{B}$ to Z; suppose not, then there is a path connecting $\boldsymbol{A}$ and $\boldsymbol{B}$ through Z; contradiction. $\square$

Proof of Property 1 and Property 2. We prove Property 1 (and Property 2) by a bottom-up induction on G. First of all it is easy to verify that the three base cases, i.e. $\boldsymbol{A} = \varnothing$ and $\boldsymbol{B} = \varnothing$, $\boldsymbol{A} = \varnothing$ and $\boldsymbol{B} = \varnothing$, are correct.

We now prove Property 1 (the proof for Property 2 is symmetric) by induction; first of all it is clear that both $\bigcup_i \boldsymbol{C}_i \cup \{Z\}$ and $\bigcup_i \boldsymbol{C}_{i,\boldsymbol{A}}$ form d-separators between $\boldsymbol{A}$ and $\boldsymbol{B}$, and it remains to show that the minimum of these two is a *minimum* d-separator between $\boldsymbol{A}$ and $\boldsymbol{B}$ that blocks all paths from $\boldsymbol{A}$ to Z. Suppose, towards a contradiction, let $\boldsymbol{C'_A}$ be such a d-separator of size $<$ MIN($|\bigcup_i \boldsymbol{C}_i \cup \{Z\}|, |\bigcup_i \boldsymbol{C}_{i,\boldsymbol{A}}|$). There are two cases:

- **If $\boldsymbol{C'_A}$ contains Z**: let G_i be the subtree rooted at Z_i and set $\boldsymbol{C}_i' = \boldsymbol{C'_A} \cap G_i$. It is immediate that $\boldsymbol{C'_i}$ is a d-separator between $\boldsymbol{A}$ and $\boldsymbol{B}$ in G_i and by assumption, there exists at least one one i such that $|\boldsymbol{C}_i'| < |\boldsymbol{C}_i|$; contradicting the induction hypothesis.

- **If $\boldsymbol{C'_A}$ does not contain Z**: let G_i be the subtree rooted at Z_i and set $\boldsymbol{C}_{i,\boldsymbol{A}}' = \boldsymbol{C'_A} \cap G_i$. Similarly, it is not hard to see that $\boldsymbol{C}_{i,\boldsymbol{A}}'$ is a d-separator between $\boldsymbol{A}$ and $\boldsymbol{B}$ in G_i that blocks all paths from $\boldsymbol{A} \cap G_i$ to Z_i. By assumption, there exists at least one i such that $|\boldsymbol{C}_{i,\boldsymbol{A}}'| < |\boldsymbol{C}_{i,\boldsymbol{A}}|$; contradicting the induction hypothesis.

$\square$

C   MULTIPLICATION WITH NON-STRUCTURED CIRCUITS

Given a contiguous structured PC $\mathscr{A}$ respecting a linear vtree and an arbitrary contiguous PC $\mathscr{B}$, which is not necessarily structured, we show a recursive algorithm that multiplies $\mathscr{A}$ and $\mathscr{B}$ in polynomial time. Specifically, for each possible scope $\boldsymbol{X}_{a:b}$ that appears in $\mathscr{B}$, we recursively construct circuit representations for the functions $p_q(\boldsymbol{X}_{a:b}) \cdot p_\mathscr{A}(\boldsymbol{X}_{a:b} \,|\, Z_a = i, Z_b = j)$ for $\oplus$ nodes $q \in \mathscr{B}$ with scope $\boldsymbol{X}_{a:b}$ and i, j hidden states of $\mathscr{A}$. In particular, from $p_\mathscr{A}(\boldsymbol{X}_{a:b} \,|\, Z_a = i, Z_b = j)$ we drop $Z_a = i$ if $a = 1$ and drop $Z_{b+1} = j$ if $b = n$, thus $p_\mathscr{A}(\boldsymbol{X}_{a:b} \,|\, Z_a = i, Z_b = j)$ corresponds to $p_\mathscr{A}(\boldsymbol{X}_{a:b} \,|\, \boldsymbol{C}_{a:b})$ as defined in Case 1. of Section 4.

The recurrence relation is similar to that of Algorithm 1. In the following derivation, we use the notations: (1) denote the children of $\oplus$ node q by $c \in \text{CH}(q)$; (2) denote the children of $\otimes$ node c by c_1 and c_2; (3) denote the weight of the edge connecting q and c by w_{qc}; (4) for each $\otimes$ node $c \in \text{CH}(q)$, c splits $\boldsymbol{X}_q = \{X_a, \dots, X_b\}$ into two contiguous segments, and denote them by $\boldsymbol{X}_{c_1} = \{X_a, \dots, X_{m_c}\}$ and $\boldsymbol{X}_{c_2} = \{X_{m_c+1}, \dots, X_b\}$ for some $a \leq m_c \leq b$.

$$\boxed{p_q(\boldsymbol{X}_{a:b}) \cdot p_\mathscr{A}(\boldsymbol{X}_{a:b+1} \,|\, Z_a = i, Z_{b+1} = j)}$$

$$= \sum_{c \in \text{CH}(q)} p_c(\boldsymbol{X}_{a:b}) \cdot w_{qc} \cdot p_\mathscr{A}(\boldsymbol{X}_{a:b+1} \,|\, Z_a = i, Z_{b+1} = j)$$

$$= \sum_{c \in \text{CH}(q)} p_{c_1}(\boldsymbol{X}_{a:m_c}) \cdot p_{c_2}(\boldsymbol{X}_{m_c+1:b+1}) \cdot w_{qc} \cdot p_\mathscr{A}(\boldsymbol{X}_{a:b+1} \,|\, Z_a = i, Z_{b+1} = j)$$

$$= \sum_{c \in \text{CH}(q)} p_{c_1}(\boldsymbol{X}_{a:m_c}) \cdot p_{c_2}(\boldsymbol{X}_{m_c+1:b+1}) \cdot w_{qc} \cdot \sum_k p_\mathscr{A}(\boldsymbol{X}_{a:b+1}, Z_{m_c} = k \,|\, Z_a = i, Z_{b+1} = j)$$

$$= \sum_{c \in \text{CH}(q)} \sum_k w_{qc} \cdot p_\mathscr{A}(Z_{m_c+1} = k \,|\, Z_a = i, Z_{b+1} = j)$$
$$\cdot p_{c_1}(\boldsymbol{X}_{a:m_c}) \cdot p_\mathscr{A}(\boldsymbol{X}_{a:m_c} \,|\, Z_{m_c+1} = k, Z_a = i, Z_{b+1} = j) \cdot p_{c_2}(\boldsymbol{X}_{m_c+1:b}) \cdot p_\mathscr{A}(\boldsymbol{X}_{a:b} \,|\, Z_{m_c+1} = k, Z_a = i, Z_{b+1} = j)$$

$$= \sum_{c \in \text{CH}(q)} \sum_k w_{qc} \cdot p_\mathscr{A}(Z_{m_c+1} = k \,|\, Z_a = i, Z_{b+1} = j)$$
$$\cdot \boxed{p_{c_1}(\boldsymbol{X}_{a:m_c}) \cdot p_\mathscr{A}(\boldsymbol{X}_{a:m_c} \,|\, Z_a = i, Z_{m_c+1} = k)} \cdot \boxed{p_{c_2}(\boldsymbol{X}_{m_c+1:b}) \cdot p_\mathscr{A}(\boldsymbol{X}_{a:b} \,|\, Z_{m_c+1} = k, Z_{b+1} = j)}$$

Now let's analyze the complexity of the constructed circuit, which we denote by $\mathscr{C}$. $\mathscr{C}$ has $O(mkh^2)$ $\oplus$ nodes in total, where m is the number of scopes in $\mathscr{B}$, $k := \max_{\boldsymbol{S} \text{ a scope in } \mathscr{B}} |\{\oplus \in \mathscr{B} \text{ with scope } \boldsymbol{S}\}|$, and h is the hidden states size of $\mathscr{A}$. Suppose that each $\oplus$ node in $\mathscr{B}$ has at most r children, then each $\oplus$ node in $\mathscr{C}$ has at most $O(rh)$ children. Hence the size of $\mathscr{C}$ is bounded by $O(mkh^2 \cdot rh) = O(mkr \cdot h^3)$. Note that $O(mkr)$ corresponds to $O(|\mathscr{B}|)$ and $O(h^3)$ is upper-bounded by $O(h^4)$, which is $O(|\mathscr{A}|^2)$; hence the size of $\mathscr{C}$ is bounded by $O(|\mathscr{A}|^2|\mathscr{B}|)$, which is the same complexity as stated in Theorem 4.4. Hence, we can remove the assumption that $\mathscr{B}$ has to be structured from Theorem 4.4.

Theorem 4.9. *Let $\mathscr{A}$ and $\mathscr{B}$ be contiguous PCs with $\mathscr{B}$ not necessarily structured. If $\mathscr{A}$ is structured respecting the linear vtree, then $\mathscr{A}$ and $\mathscr{B}$ can be multiplied in polynomial time and the size of the product PC is bounded by $O(|\mathscr{A}|^2|\mathscr{B}|)$.*

By a similar recursive construction, we can also remove the assumption that $\mathscr{B}$ has to be structured from Theorem 4.6:

Theorem C.1. *Let $\mathscr{A}$ and $\mathscr{B}$ be contiguous PCs. If $\mathscr{A}$ is structured of depth d, then then we can construct a product circuit of $\mathscr{A}$ and $\mathscr{B}$ of size bounded by $O(|\mathscr{A}|^{12d}|\mathscr{B}|)$.*

D   COMPLETE EXAMPLE OF RESTRUCTURING

In the section, we show a simple example of our restructuring algorithm in its entirety. In particular, we will consider the restructuring of a 4-variable circuit following a linear vtree, to a contiguous log-depth vtree.

In Figure 5 we show the procedure of converting the linear vtree into a Bayesian network representation, the target contiguous vtree W, and the labeling constructed by Algorithm 2. In particular, this is a valid cover according

(a) Bayesian Network for linear vtree $V_{v \mapsto Z_v}$

(b) Target vtree W
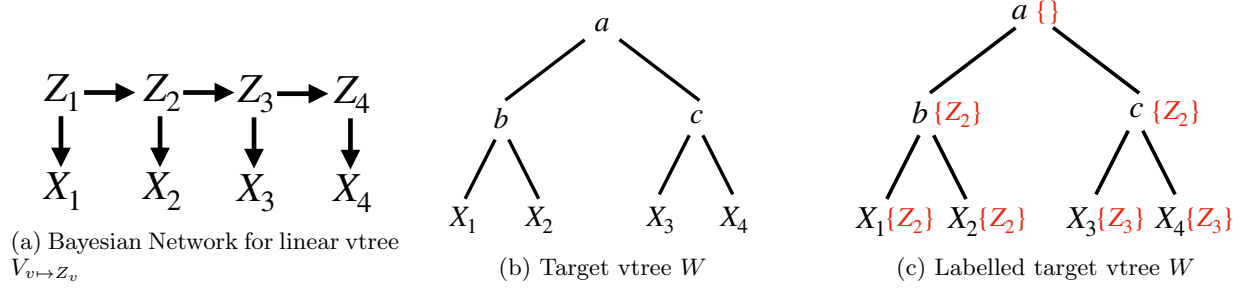
(c) Labelled target vtree W

Figure 5: Example of restructuring labeling.

to Definition 3.7 as $\{Z_2\}$ is a cover for both $\{X_1, X_2\}$ and $\{X_3, X_4\}$, and the second and third conditions in the definition are satisfied as e.g. $\{Z_2\}$ blocks paths between $\{X_1, X_2\}$ and $\{Z_2\}$ (trivially).

We further show an explicit example of the restructuring of a PC respecting the right-linear vtree in Figure 6. In the original PC, we have each $Z_i \in \{0, 1\}$, corresponding to one of the product nodes in each scope. In the restructuring, we follow the generic construction in Figure 2 for each triple of vtree nodes (w, l, r) given the labeling in Figure 5c, giving rise to the restructured PC. For example, note that the root probabilities $(0.66, 0.34)$ correspond to the marginal probability $p(Z_2)$ in the original PC. One subtlety is that the leaf nodes do not all directly correspond to the leaf nodes in the original PC; for example $p(X_1|Z_2) = \sum_{Z_1} p(Z_1|Z_2)p(X_1|Z_1)$ where $p(X_1|Z_1)$ are the leaf nodes with scope $\{X_1\}$ in the original circuit; but this is not problematic as we can simply mix the leaf nodes (either explicitly with a sum node, or by constructing new leaf nodes).
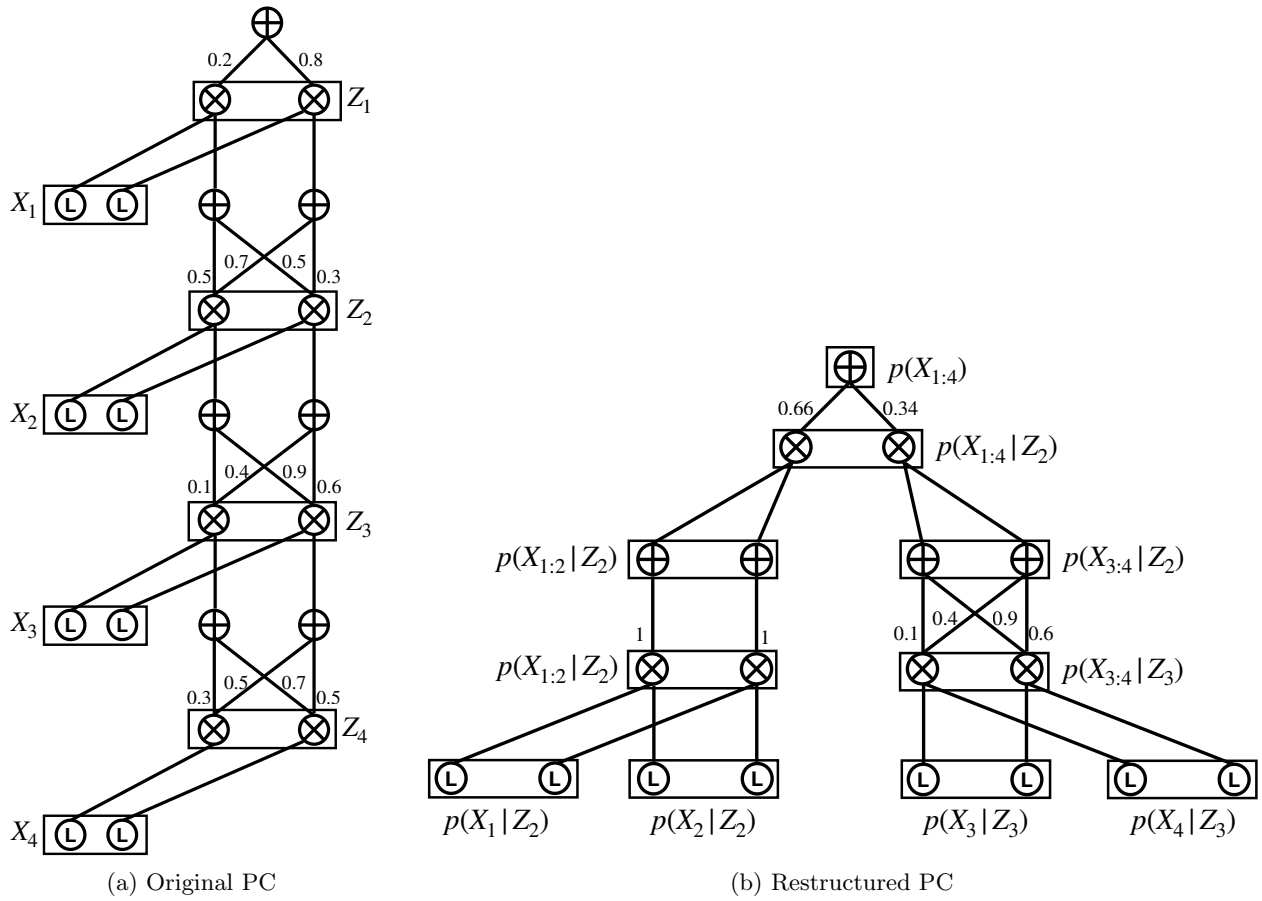
Figure 6: Example of original and restructured PC for the structures in Figure 5.