# Supplementary Material

## A Algorithm Details and Proofs

This section provides additional details about the algorithm used to compute the conditional probabilities $F_\pi(\boldsymbol{x})$ (i.e., Alg. 1) and the full proof of the theorems stated in the main paper.

### A.1 Details of Alg. 1

This section provides additional technical details of Alg. 1. Specifically, we demonstrate (i) how to select the set of PC units $\mathrm{eval}_i$ (cf. Alg. 1 line 5) and (ii) how to compute $p(x_1, \ldots, x_i)$ as a weighted mixture of $P_i$ (cf. Alg. 1 line 7). Using the example in Fig. 5, we aim to provide an intuitive illustration to both problems. As an extension to Alg. 1, rigorous and executable pseudocode for the proposed algorithm can be found in Alg. 2 and 3.

The key to speeding up the naive marginalization algorithm is the observation that we only need to evaluate a small fraction of PC units to compute each of the $D$ marginals in $F_\pi(\boldsymbol{x})$. Suppose we want to compute $F_\pi(\boldsymbol{x})$ given the structured-decomposable PC shown in Fig. 5(a), where $\oplus$, $\otimes$, and $\circledcirc$ denote sum, product, and input units, respectively. Model parameters are omitted for simplicity. Consider using the variable order $\pi = (X_1, X_2, X_3)$ (Fig. 5(b)). We ask the following question: what is the minimum set of PC units that need to be evaluated in order to compute $p(X_1 = x_1)$ (the first term in $F_\pi(\boldsymbol{x})$)? First, every PC unit with scope $\{X_1\}$ (i.e., the two nodes colored blue) has to be evaluated. Next, every PC unit $n$ that is not an ancestor of the two blue units (i.e., "non-ancestor units" in Fig. 5(b)) must have probability 1 since (i) leaf units correspond to $X_2$ and $X_3$ have probability 1 while computing $p(X_1 = x_1)$, and (ii) for any sum or product unit, if all its children have probability 1, it also has probability 1 following Eq. (2). Therefore, we do not need to evaluate these non-ancestor units. Another way to identify these non-ancestor units is by inspecting their variable scopes — if the variable scope of a PC unit $n$ does not contain $X_1$, it must has probability 1 while computing $p(X_1 = x_1)$. Finally, following all ancestors of the two blue units (i.e., "ancestor units" in Fig. 5(b)), we can compute the probability of the root unit, which is the target quantity $p(X_1 = x_1)$. At a first glance, this seems to suggest that we need to evaluate these ancestor units explicitly. Fortunately, as we will proceed to show, the root unit's probability can be equivalently computed using the blue units' probabilities weighted by a set of cached *top-down probabilities*.

For ease of presentation, denote the two blue input units as $n_1$ and $n_2$, respectively. A key observation is that the probability of every ancestor unit of $\{n_1, n_2\}$ (including the root unit) can be represented as a weighted mixture over $p_{n_1}(\boldsymbol{x})$ and $p_{n_2}(\boldsymbol{x})$, the probabilities assigned to $n_1$ and $n_2$, respectively. The reason is that for each decomposable product node $m$, only distributions defined on disjoint variables shall be multiplied. Since $n_1$ and $n_2$ have the same variable scope, their distributions will not be multiplied by any product node. Following the above intuition, the top-down probability $p_{\mathrm{down}}(n)$ of PC unit $n$ is designed to represent the "weight" of $n$ w.r.t. the probability of the root unit. Formally, $p_{\mathrm{down}}(n)$ is defined as the sum of the probabilities of every path from $n$ to the root unit $n_r$, where the probability of a path is the product of all edge parameters traversed by it. Back to our example, using the top-down probabilities, we can compute $p(X_1 = x_1) = \sum_{i=1}^{2} p_{\mathrm{down}}(n_i) \cdot p_{n_i}(x_1)$ without explicitly evaluating the ancestors of $n_1$ and $n_2$. The quantity $p_{\mathrm{down}}(n)$ of all PC units $n$ can be computed by Alg. 2 in $\mathcal{O}(|p|)$ time. Specifically, the algorithm performs a top-down traversal over all PC units $n$, and updates the top-down probabilities of their children $\mathsf{in}(n)$ along the process.

Therefore, we only need to compute the two PC units with scope $\{X_1\}$ in order to calculate $p(X_1 = x_1)$. Next, when computing the second term $p(X_1 = x_1, X_2 = x_2)$, as illustrated in Fig. 5(b), we can reuse the evaluated probabilities of $n_1$ and $n_2$, and similarly only need to evaluate the PC units with scope $\{X_2\}$, $\{X_2, X_3\}$, or $\{X_1, X_2, X_3\}$ (i.e., nodes colored purple). The same scheme can be used when computing the third term, and we only evaluate PC units with scope $\{X_3\}$, $\{X_2, X_3\}$, or $\{X_1, X_2, X_3\}$ (i.e., all red nodes). As a result, we only evaluate 20 PC units in total, compared to $3 \cdot |p| = 39$ units required by the naive approach.

This procedure is formalized in Alg. 3, which adds additional technical details compared to Alg. 1. In the main loop (lines 5-9), the $D$ terms in $F_\pi(\boldsymbol{x})$ are computed one-by-one. While computing each term, we first find the PC units that need to be evaluated (line 6).[7] After computing their probabilities

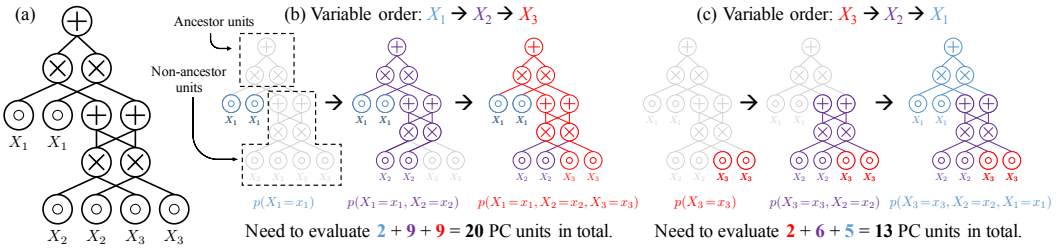


Figure 5: Good variable orders lead to more efficient computation of $F_\pi(\boldsymbol{x})$. Consider the PC $p$ shown in (a). (b): If variable order $X_1, X_2, X_3$ is used, we need to evaluate 20 PC units in total. (c): The optimal variable order $X_3, X_2, X_1$ allows us to compute $F_\pi(\boldsymbol{x})$ by only evaluating 13 PC units.

**Algorithm 2** PC Top-down Probabilities

1: **Input:** A smooth and structured-decomposable PC $p$
2: **Output:** The top-down probabilities $p_{\text{down}}(n)$ of all PC units $n$
3: For every PC unit $n$ in $p$, initialize $p_{\text{down}}(n) \leftarrow 0$
4: **foreach** unit $n$ traversed in preorder (parent before children) **do**
5: | **if** $n$ is the root node of $p$ **then** $p_{\text{down}}(n) \leftarrow 1$
6: | **elif** $n$ is a sum unit **then foreach** $c \in \mathsf{in}(n)$ **do** $p_{\text{down}}(c) \leftarrow p_{\text{down}}(c) + p_{\text{down}}(n) \cdot \theta_{n,c}$
7: | **elif** $n$ is a product unit **then foreach** $c \in \mathsf{in}(n)$ **do** $p_{\text{down}}(c) \leftarrow p_{\text{down}}(c) + p_{\text{down}}(n)$

in a bottom-up manner (line 7), we additionally use the pre-computed top-down probabilities to obtain the target marginal probability (lines 8-9).

The previous example demonstrates that even without a careful choice of variable order, we can significantly lower the computation cost by only evaluating the necessary PC units. We now show that with an *optimal* choice of variable order (denoted $\pi^*$), the cost can be further reduced. Consider using order $\pi^* = (X_3, X_2, X_1)$, as shown in Fig. 5(c), we only need to evaluate $2+6+5=13$ PC units in total when running Alg. 3. This optimal variable order is the key to guaranteeing $\mathcal{O}(\log(D)\cdot|p|)$ computation time. In the following, we first give a technical assumption and then proceed to justify the *correctness* and *efficiency* of Alg. 3 when using the optimal variable order $\pi^*$.

### A.2 Proof of Theorem 1

As hinted by the proof sketch given in the main text, this proof consists of three main parts — (i) construction of the optimal variable order $\pi^*$ given a smooth and structured-decomposable PC, (ii) justify the correctness of Alg. 3, and (iii) prove that $F_{\pi^*}(\boldsymbol{x})$ can be computed by evaluating no more than $\mathcal{O}(\log(K)\cdot|p|)$ PC units (i.e., analyze the time complexity of Alg. 3).

**Construction of an optimal variable order** For ease of illustration, we first transform the original smooth and structured-decomposable PC into an equivalent PC where every product node has two children. Fig. 6 illustrates this transformation on any product node with more than two children. Note that this operation will not change the number of parameters in a PC, and will only incur at most $2\cdot|p|$ edges.

We are now ready to define the variable tree (*vtree*) (Kisa et al., 2014) of a smooth and structured-decomposable PC. Specifically, a *vtree* is a binary tree structure whose leaf nodes are labeled with a PC's input features/variables $\mathbf{X}$ (every leaf node is labeled with one variable). A PC conforms to a vtree if for every product unit $n$, there is a corresponding vtree node $v$ such that children of $n$ split the variable scope $\phi(n)$ in the same way as the children of the vtree node $v$. According to its definition, every smooth and structured-decomposable PC whose product units all have two children must conform to a vtree (Kisa et al., 2014). For example, the PC shown in Fig. 7(a) conforms to the vtree illustrated in Fig. 7(b). Similar to PCs, we define the scope $\phi(v)$ of a vtree node $v$ as the set of all descendent leaf variables of $v$.

We say that a unit $n$ in a smooth and structured-decomposable PC conforms to a node $v$ in the PC's corresponding vtree if their scopes are identical. For ease of presentation, define $\varphi(p, v)$ as the set of PC units that conform to vtree node $v$. Additionally, we define $\varphi_{\text{sum}}(p, v)$ and $\varphi_{\text{prod}}(p, v)$ as the set of sum and product units in $\varphi(p, v)$, respectively.

**Algorithm 3** Compute $F_\pi(\boldsymbol{x})$

1: **Input:** A smooth and structured-decomposable PC $p$, variable order $\pi$, variable instantiation $\boldsymbol{x}$
2: **Output:** $F_\pi(\boldsymbol{x}) = \{p(x_{\pi_1}, \ldots, x_{\pi_i})\}_{i=1}^{D}$
3: **Initialize:** The probability $p(n)$ of every unit $n$ is initially set to 1
4: $p_{\text{down}} \leftarrow$ the top-down probability of every PC unit$n$ (i.e., Alg. 2)
5: **for** $i = 1$ **to** $D$ **do** # Compute the $i$th term in $F_\pi(\boldsymbol{x})$: $p(x_{\pi_1}, \ldots, x_{\pi_i})$
6: $\quad \text{eval}_i \leftarrow$ the set of PC units $n$ with scopes $\phi(n)$ that satisfy at least one of the following conditions: (i) $\phi(n) = \{X_{\pi_i}\}$; (ii) $n$ is a sum unit and at least one child $c$ of $n$ needs evaluation, i.e., $c \in \text{eval}_i$; (iii) $n$ is a product unit and $X_{\pi_i} \in \phi(n)$ and $\nexists c \in \mathsf{in}(n)$ such that $\{X_{\pi_j}\}_{j=1}^{i} \in \phi(c)$
7: $\quad$ Evaluate PC units in $\text{eval}_i$ in a bottom-up manner to compute $\{p_n(\boldsymbol{x}) : n \in \text{eval}_i\}$
8: $\quad \text{head}_i \leftarrow$ the set of PC units in $\text{eval}_i$ such that none of their parents are in $\text{eval}_i$
9: $\quad p(x_{\pi_1}, \ldots, x_{\pi_i}) \leftarrow \sum_{n \in \text{head}_i} p_{\text{down}}(n) \cdot p_n(\boldsymbol{x})$

Next, we define an operation that changes a vtree into an *ordered vtree*, where for each inner node $v$, its left child has more descendent leaf nodes than its right child. See Fig. 7(c-d) as an example. The vtree in Fig. 7(b) is transformed into an ordered vtree illustrated in Fig. 7(c); the corresponding PC (Fig. 7(a)) is converted into an *ordered PC* (Fig. 7(d)). This transformation can be performed by all smooth and structured-decomposable PCs.

We are ready to define the optimal variable order. For a pair of ordered PC and ordered vtree, the optimal variable order $\pi^*$ is defined as the order the leaf vtree nodes (each corresponds to a variable) are accessed following an inorder traverse of the vtree (left child accessed before right child).

**Correctness of Algorithm 3** Assume we have access to a smooth, structured-decomposable, and ordered PC $p$ and its corresponding vtree. Recall from the above construction, the optimal variable order $\pi^*$ is the order following an inorder traverse of the vtree.

We show that it is sufficient to only evaluate the set of PC units stated in line 6 of Alg. 3. Using our new definition of vtrees, we state line 6 in the following equivalent way. At iteration $i$ (i.e., we want to compute the $i$th term in $F_\pi(\boldsymbol{x})$: $p(x_{\pi_1}, \ldots, x_{\pi_i})$), we need to evaluate all PC units that conform to any vtree node in the set $T_{p,i}$. Here $T_{p,i}$ is defined as the set of vtree nodes $v$ that satisfy the following condition: $X_{\pi_i} \in \phi(v)$ and there does not exist a child $c$ of $v$ such that $\{X_{\pi_j}\}_{j=1}^{i} \in \phi(c)$. For ease of presentation, we refer to evaluate PC units $\varphi(p, v)$ when we say "evaluate a vtree node $v$".

First, we don't need to evaluate vtree units $v$ where $X_{\pi_i} \notin \phi(v)$ because the probability of these PC units will be identical to that at iteration $i-1$ (i.e., when computing $p(x_{\pi_1}, \ldots, x_{\pi_{i-1}})$). Therefore, we only need to cache these probabilities computed in previous iterations.

Second, we don't need to evaluate vtree units $v$ where at least one of its children $c$ satisfy $\{X_{\pi_j}\}_{j=1}^{i-1} \in \phi(c)$ because we can obtain the target marginal probability $p(x_{\pi_1}, \ldots, x_{\pi_i})$ following lines 7-9 of Alg. 3. We proceed to show how this is done in the following.

Denote the "highest" in $T_{p,i}$ as $v_{r,i}$ (i.e., the parent of $v_{r,i}$ is not in $T_{p,i}$). According to the variable order $\pi^*$, $v_{r,i}$ uniquely exist for any $i \in [D]$. According to Alg. 2, the top-down probabilities of PC units is defined as follows

• $p_{\text{down}}(n_r) = 1$, where $n_r$ is the PC's root unit.

• For any product unit $n$, $p_{\text{down}}(n) = \sum_{m \in \text{par}(n)} p_{\text{down}}(m) \cdot \theta_{m,n}$, where $\text{par}(n)$ is the set of parent (sum) units of $n$.

• For any sum unit $n$, $p_{\text{down}}(n) = \sum_{m \in \text{par}(n)} p_{\text{down}}(m)$, where $\text{par}(n)$ is the set of parent (product) units of $n$.

We now prove that

$$p(x_{\pi_1}, \ldots, x_{\pi_i}) = \sum_{n \in \varphi_{\text{sum}}(p,v)} p_{\text{down}}(n) \cdot p_n(\boldsymbol{x}) \tag{3}$$

holds when $v = v_{r,i}$.

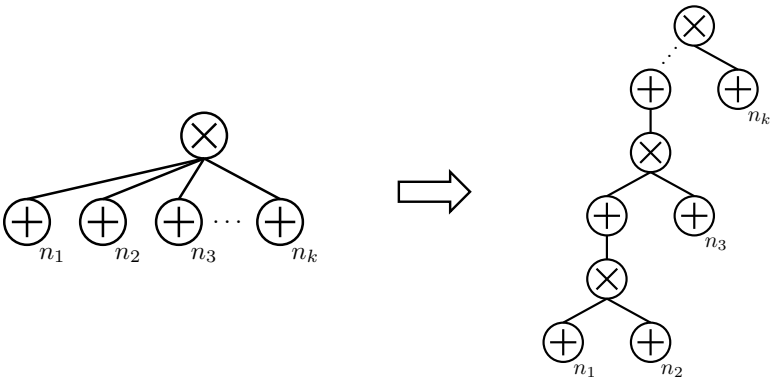


Figure 6: Convert a product unit with $k$ children into an equivalent PC where every product node has two children.

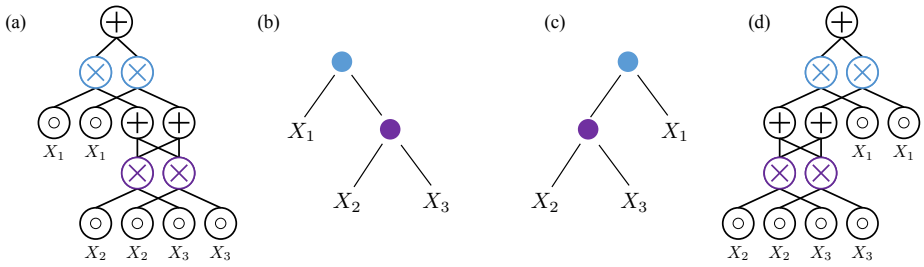


Figure 7: (a-b): An example structured-decomposable PC and a corresponding *vtree*. (c-d): Converting (b) into an ordered vtree. (d) The converted ordered PC that is equivalent to (a).

• **Base case:** If $v$ is the vtree node correspond to $n_r$, then $\varphi_{\text{sum}}(p, v) = \{n_r\}$ and it is easy to verify that

$$p(x_{\pi_1}, \ldots, x_{\pi_i}) = p_{\text{down}}(n_r) \cdot p_{n_r}(\boldsymbol{x}) = \sum_{n \in \varphi_{\text{sum}}(p,v)} p_{\text{down}}(n) \cdot p_n(\boldsymbol{x})$$

• **Inductive case:** Suppose $v$ is an ancestor of $v_{r,i}$ and the parent vtree node $v_p$ of $v$ satisfy Eq. (3). We have

$$\begin{aligned}
p(x_{\pi_1}, \ldots, x_{\pi_i}) &= \sum_{m \in \varphi_{\text{sum}}(p,v_p)} p_{\text{down}}(m) \cdot p_m(\boldsymbol{x}) \\
&= \sum_{m \in \varphi_{\text{sum}}(p,v_p)} \sum_{n \in \mathsf{in}(m)} p_{\text{down}}(m) \cdot \theta_{m,n} \cdot p_n(\boldsymbol{x}) \\
&\overset{(a)}{=} \sum_{n \in \varphi_{\text{prod}}(p,v_p)} \underbrace{\sum_{m \in \text{par}(n)} p_{\text{down}}(m) \cdot \theta_{m,n}}_{p_{\text{down}}(n)} \cdot p_n(\boldsymbol{x}) \\
&= \sum_{n \in \varphi_{\text{prod}}(p,v_p)} p_{\text{down}}(n) \cdot p_n(\boldsymbol{x}) \\
&\overset{(b)}{=} \sum_{n \in \varphi_{\text{prod}}(p,v_p)} \sum_{o \in \{o : o \in \mathsf{in}(n), \{X_j\}_{j=1}^{i} \in \phi(o)\}} p_{\text{down}}(n) \cdot p_o(\boldsymbol{x}) \\
&\overset{(c)}{=} \sum_{o \in \varphi_{\text{sum}}(p,v)} \underbrace{\sum_{n \in \text{par}(o)} p_{\text{down}}(n) \cdot p_o(\boldsymbol{x})}_{p_{\text{down}}(o)}
\end{aligned}$$

$$= \sum_{o \in \varphi_{\text{sum}}(p,v)} p_{\text{down}}(o) \cdot p_o(\boldsymbol{x})$$

where $(a)$ reorders the terms for summation; $(b)$ holds since $\forall n \in \varphi_{\text{prod}}(p, v_p)$, $p_n(\boldsymbol{x}) = \prod_{o \in \mathsf{in}(n)} p_o(\boldsymbol{x})$ and $\forall o \in \mathsf{in}(n)$ such that $\{X_j\}_{j=1}^{i} \cap \phi(o) = \varnothing$, $p_o(\boldsymbol{x}) = 1$;[8] $(c)$ holds because

$$\bigcup_{n \in \varphi_{\text{prod}}(p,v_p)} \{o : o \in \mathsf{in}(n), \{X_j\}_{j=1}^{i} \in \phi(o)\} = \varphi_{\text{sum}}(p, v).$$

Thus, we have prove that Eq. (3) holds for $v = v_{r,i}$, and hence the probability $p(x_{\pi_1}, \ldots, x_{\pi_i})$ can be computed by weighting the probability of PC units $\varphi_{\text{sum}}(p, v_{r,i})$ (line 8 in Alg. 3) with the corresponding top-down probabilities (line 9 in Alg. 3).

**Efficiency of following the optimal variable order** We proceed to show that when using the optimal variable order $\pi^*$, Alg. 3 evaluates no more than $\mathcal{O}(\log(D) \cdot |p|)$ PC units.

According to the previous paragraphs, whenever Alg. 3 evaluates a PC unit $n$ w.r.t. vtree node $v$, it will evaluate all PC units in $\varphi(p, v)$. Therefore, we instead count the total number of vtree nodes need to be evaluated by Alg. 3. Since the PC is assumed to be balanced Def. 4, for every $v$, we have $\varphi(p, v) = \mathcal{O}(|p|/D)$. Therefore, we only need to show that Alg. 3 evaluates $\mathcal{O}(D \cdot \log(D))$ vtree nodes in total.

We start with the base case, which is PCs correspond to a single vtree leaf node $v$. In this case, $F_{\pi^*}(\boldsymbol{x})$ boils down to computing a single marginal probability $p(x_{\pi_1^*})$, which needs to evaluate PC units $\varphi(p, v)$ once.

Define $f(x)$ as the number of vtree nodes need to be evaluated given a PC corresponds to a vtree node with $x$ descendent leaf nodes. From the base case we know that $f(1) = 1$.

Next, consider the inductive case where $v$ is an inner node that has $x$ descendent leaf nodes. Define the left and right child node of $v$ as $c_1$ and $c_2$, respectively. Let $c_1$ and $c_2$ have $y$ and $z$ descendent leaf nodes, respectively. We want to compute $F_{\pi^*}(\boldsymbol{x})$, which can be broken down into computing two following sets of marginals:

$$\text{Set 1: } \left\{p(x_{\pi_1^*}, \cdots, x_{\pi_i^*})\right\}_{i=1}^{y}, \qquad \text{Set 2: } \left\{p(x_{\pi_1^*}, \cdots, x_{\pi_i^*})\right\}_{i=y+1}^{y+z}.$$

Since $\pi^*$ follows the in-order traverse of $v$, to compute the first term, we only need to evaluate $c_1$ and its descendents, that is, we need to evaluate $f(y)$ vtree nodes. This is because the marginal probabilities in set 1 are only defined on variables in $\phi(c_1)$. To compute the second term, in addition to evaluating PC units corresponding to $c_2$ (that is $f(z)$ vtree nodes in total),[9] we also need to re-evaluate the PC units $\varphi(p, v)$ every time, which means we need to evaluate $z$ more vtree nodes. In summary, we need to evaluate

$$f(x) = f(y) + f(z) + z \quad (y \geq z, y + z = x)$$

vtree nodes.

To complete the proof, we upper bound the number of vtree nodes need to be evaluated. Define $g(\cdot)$ as follows:

$$g(x) = \max_{y \in \{1, \ldots, \lfloor \frac{x}{2} \rfloor\}} y + g(y) + g(x - y).$$

It is not hard to verify that $\forall x \in \mathbb{Z}, g(x) \geq f(x)$. Next, we prove that

$$\forall x \in \mathbb{Z}\,(x \geq 2),\ g(x) \leq 3x \log x.$$

First, we can directly verify that $g(2) \leq 3 \cdot 2 \log_2 2 \approx 4.1$. Next, for $x \geq 3$,

$$g(x) = \max_{y \in \{1, \ldots, \lfloor \frac{x}{2} \rfloor\}} y + g(y) + g(x - y)$$

[8] This is because the scope of these PC units does not contain any of the variables in $\{X_{\pi_j}\}_{j=1}^{i}$.

[9] As justified in the second part of this proof, all probabilities of PC units that conform to descendents of $c_1$ will be unchanged when computing the marginals in set 2. Hence we only need to cache these probabilities.

$$
\begin{aligned}
&\leq \max_{y\in\{1,\dots,\lfloor \frac{x}{2} \rfloor\}} \underbrace{y + 3y\log y + 3(x-y)\log(x-y)}_{h(y)} \\
&\overset{(a)}{\leq} \max\left(1 + 3(x-1)\log(x-1), \left\lfloor\frac{x}{2}\right\rfloor + 3\left\lfloor\frac{x}{2}\right\rfloor \log\left\lfloor\frac{x}{2}\right\rfloor + 3\left(x - \left\lfloor\frac{x}{2}\right\rfloor\right)\log\left(x - \left\lfloor\frac{x}{2}\right\rfloor\right)\right) \\
&\leq \max\left(1 + 3(x-1)\log(x-1), \left\lfloor\frac{x}{2}\right\rfloor + 3(x+1)\log\frac{x+1}{2}\right) \\
&\leq 3x\log x,
\end{aligned}
$$

where $(a)$ holds since according to its derivative, $h(y)$ obtains its maximum value at either $y = 1$ or $y = \lfloor \frac{x}{2} \rfloor$.

For a structured-decomposable PC with $D$ variables, $g(D) \leq 3D\log D$ vtree nodes need to be evaluated. Since each vtree node corresponds to $\mathcal{O}(\frac{|p|}{D})$ PC units, we need to evaluate $\mathcal{O}(\log(D)\cdot|p|)$ PC units to compute $F_{\pi^*}(\boldsymbol{x})$.

### A.3 HCLTs, EiNets, and RAT-SPNs are Balanced

Consider the compilation from a PGM to an HCLT (Sec. 4.1). We first note that each PGM node $g$ uniquely corresponds to a variable scope $\phi$ of the PC. That is, all PC units correspond to $g$ have the same variable scope. Please first refer to Appx. B.2 for details on how to generate a HCLT given its PGM representation.

In the main loop of Alg. 4 (lines 5-10), for each PGM node $g$ such that $\mathrm{var}(g) \in \mathbf{Z}$, the number of computed PC units are the same ($M$ product units compiled in line 9 and $M$ sum units compiled in line 10). Therefore, for any variable scopes $\phi_1$ and $\phi_2$ possessed by some PC units, we have $|\mathrm{nodes}(p, \phi(m))| \approx |\mathrm{nodes}(p, \phi(n))|$. Since there are in total $\Theta(D)$ different variable scopes in $p$, we have: for any scope $\phi'$ exists in an HCLT $p$, $\mathrm{nodes}(p, \phi') = \mathcal{O}(|p|/D)$.

EiNets and RAT-SPNs are also balanced since they also have an equivalent PGM representation of their PCs. The main difference between these models and HCLTs is the different variable splitting strategy in the product units.

## B Methods and Experiment Details

### B.1 Learning HCLTs

**Computing Mutual Information** As mentioned in the main text, computing the pairwise mutual information between variables $\mathbf{X}$ is the first step to compute the Chow-Liu Tree. Since we are dealing with categorical data (e.g., 0-255 for pixels), we compute mutual information by following its definition:

$$
I(X;Y) = \sum_{i=1}^{C_X}\sum_{j=1}^{C_Y} P(X=i, Y=j)\log_2\frac{P(X=i,Y=j)}{P(X=i)P(Y=j)},
$$

where $C_X$ and $C_Y$ are the number of categories for variables $X$ and $Y$, respectively. To lower the computation cost, for image data, we truncate the data by only using 3 most-significant bits. That is, we treat the variables as categorical variables with $2^3 = 8$ categories during the construction of the CLT. Note that we use the full data when constructing/learning the PC.

**Training pipeline** We adopt two types of EM updates — mini-batch and full-batch. In mini-batch EM, parameters are updated according to a step size $\eta$: $\boldsymbol{\theta}^{(k+1)} \leftarrow (1-\eta)\boldsymbol{\theta}^{(k)} + \eta\boldsymbol{\theta}^{(\text{new})}$, where $\boldsymbol{\theta}^{(\text{new})}$ is the EM target computed with a batch of samples; full-batch EM updates the parameters by the EM target computed using the whole dataset. In this paper, HCLTs are trained by first running mini-batch EM with batch size 1024 and $\eta$ changing linearly from $0.1$ to $0.05$; full-batch EM is then used to finetune the parameters.

**Algorithm 4** Compile the PGM representation of a HCLT into an equivalent PC

1: **Input:** A PGM representation of a HCLT $\mathcal{G}$ (e.g., Fig. 3(c)); hyperparameter $M$
2: **Output:** A smooth and SD PC $p$ equivalent to $\mathcal{G}$
3: **Initialize:** $\text{cache} \leftarrow \text{dict}()$ a dictionary storing intermediate PC units
4: **Sub-routines:** **PC_leaf**$(X_i)$ returns a PC input unit of variable $X_i$; **PC_prod**$(\{n_i\}_{i=1}^m)$ (resp. **PC_sum**$(\{n_i\}_{i=1}^m)$) returns a product (resp. sum) unit over child nodes $\{n_i\}_{i=1}^m$.
5: **foreach** node $g$ traversed in postorder (bottom-up) of $\mathcal{G}$ **do**
6: $\quad$ **if** $\text{var}(g) \in \mathbf{X}$ **then** $\text{cache}[g] \leftarrow \big[\textbf{PC\_leaf}\big(\text{var}(g)\big) \text{ for } i = 1 : M\big]$
7: $\quad$ **else** # That is, $\text{var}(g) \in \mathbf{Z}$
8: $\quad\quad$ $\text{chs\_cache} \leftarrow \big[\text{cache}[c] \text{ for } c \text{ in children}(g)\big]$ # children$(g)$ is the set of children of $g$
9: $\quad\quad$ $\text{prod\_nodes} \leftarrow \big[\textbf{PC\_prod}\big(\big[\text{nodes}[i] \text{ for nodes in chs\_cache}\big]\big) \text{ for } i = 1 : M\big]$
10: $\quad\quad$ $\text{cache}[g] \leftarrow \big[\textbf{PC\_sum}\big(\text{prod\_nodes}\big) \text{ for } i = 1 : M\big]$
11: **return** $\text{cache}[\text{root}(\mathcal{G})][0]$

### B.2 Generating PCs Following the HCLT Structure

After generating the PGM representation of a HCLT model, we are now left with the final step of compiling the PGM representation of the model into an equivalent PC. Recall that we define the latent variables $\{Z_i\}_{i=1}^4$ as categorical variables with $M$ categories, where $M$ is a hyperparameter. As demonstrated in Alg. 4, we incrementally compile every PGM node into an equivalent PC unit though a bottom-up traverse (line 5) of the PGM. Specifically, leaf PGM nodes corresponding to observed variables $X_i$ are compiled into PC input units of $X_i$ (line 6), and inner PGM nodes corresponding to latent variables are compiled by taking products and sums (implemented by product and sum units) of its child nodes' PC units (lines 8-10). Leaf units generated by $\mathbf{PC\_leaf}(X)$ can be any simple univariate distribution of $X$. We used categorical leaf units in our HCLT experiments. Fig. 3(d) demonstrates the result PC after running Alg. 4 with the PGM in Fig. 3(c) and $M = 2$.

### B.3 Implementation Details of the PC Learning Algorithm

We adopted the EM parameter learning algorithm introduced in Choi et al. (2021), which computes the EM update targets using *expected flows*. Following Liu & Van den Broeck (2021), we use a hybrid EM algorithm, which uses mini-batch EM updates to initiate the training process, and switch to full-batch EM updates afterwards.

• Mini-batch EM: denote $\boldsymbol{\theta}^{(\text{EM})}$ as the EM update target computed with a mini-batch of samples. An update with step-size $\eta$ is: $\boldsymbol{\theta}^{(k+1)} \leftarrow (1-\eta)\boldsymbol{\theta}^{(k)} + \eta\boldsymbol{\theta}^{(\text{EM})}$.

• Full-batch EM: denote $\boldsymbol{\theta}^{(\text{EM})}$ as the EM update target computed with the whole dataset. Full-batch EM updates the parameters with $\boldsymbol{\theta}^{(\text{EM})}$ at each iteration.

In our experiments, we trained the HCLTs with 100 mini-batch EM epochs and 20 full-batch EM epochs. During mini-batch EM updates, $\eta$ was annealed linearly from $0.15$ to $0.05$.

### B.4 Details of the Compression/Decompression Experiment

**Hardware specifications** All experiments are performed on a server with 72 GPUs, 512G Memory, and 2 TITAN RTX GPUs. In all experiments, we only use a single GPU on the server.

**IDF** We ran all experiments with the code in the GitHub repo provided by the authors. We adopted an IDF model with the following hyperparameters: 8 flow layers per level; 2 levels; densenets with depth 6 and 512 channels; base learning rate $0.001$; learning rate decay $0.999$. The algorithm adopts an CPU-based entropy coder rANS. For (de)compression, we used the following script: `https://github.com/jornpeters/integer_discrete_flows/blob/master/experiment_coding.py`.

**BitSwap** We trained all models using the following author-provided script: `https://github.com/fhkingma/bitswap/blob/master/model/mnist_train.py`. The al-

gorithm adopts an CPU-based entropy coder rANS. And we used the following code for (de)compression: `https://github.com/fhkingma/bitswap/blob/master/mnist_compress.py`.

**BB-ANS** All experiments were performed using the following official code: `https://github.com/bits-back/bits-back`.

### B.5 Details of the PC+IDF Model

The adopted IDF architecture follows the original paper (Hoogeboom et al., 2019). For the PCs, we adopted EiNets (Peharz et al., 2020a) with hyperparameters $K = 12$ and $R = 4$. Instead of using random binary trees to define the model architecture, we used binary trees where "closer" latent variables in $\boldsymbol{z}$ will be put closer in the binary tree.

Parameter learning was performed by the following steps. First, compute the average log-likelihood over a mini-batch of samples. The negative average log-likelihood is the loss we use. Second, compute the gradients w.r.t. all model parameters by backpropagating the loss. Finally, update the IDF and PCs using the gradients individually: for IDF, following Hoogeboom et al. (2019), the Adamax optimizer was used; for PCs, following Peharz et al. (2020a), we use the gradients to compute the EM target of the parameters and performed mini-batch EM updates.