On the Definition and Computation of Causal Treewidth

Abstract

Causal treewidth is a recent notion which allows one to speed up Bayesian network inference and to bound its complexity in the presence of functional dependencies (causal mechanisms) whose identities are unknown. Causal treewidth is no greater than treewidth and can be bounded even when treewidth is unbounded. The utility of causal treewidth has been illustrated recently in the context of causal inference and model-based supervised learning. However, the current definition of causal treewidth is descriptive rather than perspective, therefore limiting its full exploitation in a practical setting. We provide an extensive study of causal treewidth in this paper which moves us closer to realizing the full computational potential of this notion both theoretically and practically.

1 INTRODUCTION

Treewidth is one of the most influential notions for parameterizing the complexity of probabilistic inference. This notion originated in the graph theory literature and can be viewed as a measure of graph connectivity [Robertson and Seymour, 1986]. It has also been used to parameterize the complexity of many algorithmic tasks that transcend probabilistic inference; see, e.g., [Bodlaender, 2006, Dechter, 2003]. For Bayesian networks, the time and space complexity of computing marginals is bounded by $O(n \cdot \exp(w))$ where n is the number of nodes in the network and w is its treewidth. For example, tree-structured networks have a treewidth of 1 so treewidth allows us to show that inference on such networks can be done in linear time and space.

Treewidth captures the structural aspects of a model and is independent of its parameters. Hence, one can use treewidth to provide guarantees on the complexity of inference without needing to know the model parameters. In the first few decades of research on Bayesian network inference, the perception was that high treewidth is a barrier since all influential algorithms at that time, particularly the jointree and variable elimination algorithms [Jensen et al., 1990, Zhang and Poole, 1996, Dechter, 1996], had a complexity which was also lower bounded exponentially by treewidth. Later developments showed that exploiting the parametric structure of Bayesian networks can lead to tractable inference in some situations where the treewidth can be very high; see, e.g., [Larkin and Dechter, 2003, Chavira and Darwiche, 2005, Chavira et al., 2006, Chavira and Darwiche, 2008]. The parametric structure exploited was particularly in the form of context-specific independence [Boutilier et al., 1996] and logical constrains (i.e., parameters in $\{0, 1\}$).[1]

More recently, a new and more abstract type of parametric structure has been identified and exploited computationally: functional dependencies, also known as causal mechanisms, which identities are *unknown* [Darwiche, 2020]. In a Bayesian network, a node is functionally determined by its parents if fixing the state of these parents also fixes the state of the node (that is, the node distribution is deterministic given any state of its parents). We often know that a node is functionally determined by its parents but without knowing the identity of the underlying function. This is prominent, for example, in causal inference where one typically has a (non-parametric) causal graph in which every internal node is functionally determined by its parents yet without knowing the specific causal mechanisms of nodes [Pearl, 2000]. Classical techniques for exploiting parametric structure are not applicable in this case since these methods

[1] Among the most effective approaches for exploiting parametric structure are the ones based on compiling Bayesian networks into tractable circuits [Darwiche, 2003]. These approaches allow one to conduct inference in time linear in the circuit size while yielding circuits whose size is not necessarily exponential in treewidth—see [Darwiche, 2021a] for a recent survey on circuit representations and [Agrawal et al., 2021] for a recent empirical evaluation in which methods based on circuits ranked at the forefront in terms of efficiency.

Submitted to the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2022). **To be used for reviewing only**.

require knowledge of the specific model parameters which imply knowledge of the specific causal mechanisms. This also arises when learning the parameters of a Bayesian network from data where we may have background knowledge to the effect that some nodes are functionally determined by their parents but without knowing the specific functions as we are trying to learn them; see, e.g., [Chen et al., 2020]. Interestingly enough, a recent finding showed that one can exploit unknown causal mechanisms computationally, leading to potentially exponential reduction in complexity [Darwiche, 2020]. This finding was based on two new theorems and cast in the context of model-based supervised learning. It particularly took the form of an algorithm that compiles the *structure* of a Bayesian network into a tractable circuit which size is not necessarily exponential in treewidth. This approach managed to efficiently compile circuits for networks with treewidth over 100 without needing to know the network parameters, only that some nodes are functionally determined by their parents. More recently, this finding was cast in the context of causal inference while hinting that it can lead to a new parameter for bounding complexity that was called *causal treewidth* [Darwiche, 2021b].

Treewidth is classically defined for an undirected graph but it can be extended to directed acyclic graphs (DAGs) by computing the treewidth of the *moralized* DAG. This is an undirected graph obtained from the DAG by connecting every pair of parents by an edge and then removing the directionality of edges; see [Darwiche, 2009, Ch 9]. Causal treewidth applies only to DAGs in which some nodes are declared as being *functional.* If no nodes are functional, then the causal treewidth reduces to treewidth. While [Darwiche, 2021b] suggested this more refined notion of causal treewidth, it did not provide an operational definition of causal treewidth and therefore it did not specify a method for computing it. Moreover, while [Darwiche, 2020] showed that inference can be sped up, exponentially in some cases, by exploiting unknown causal mechanisms, it did not show that the proposed computational techniques fully exploit the two new, enabling theorems that inspired these techniques.

Our goal in this paper is to first review the two key theorems in [Darwiche, 2020] that enabled the computational exploitation of unknown causal mechanisms, and to then use them as basis for formally defining the notion of causal treewidth and how it can be computed. In the process of doing so, we will prove some results about the algorithmic techniques proposed in [Darwiche, 2020], showing that some are optimal while others are not. In other words, we will show that the algorithmic techniques proposed in [Darwiche, 2020] do not fully exploit the two enabling theorems identified in that work. Hence, the main contribution of this work is that it brings us closer, both theoretically and practically, towards the full exploitation of unknown causal mechanisms during inference. At a more cognitive level, our contribution may provide further hints as to why causal knowledge is so central to human reasoning [Pearl and Mackenzie, 2018] as we provide a formal account of how causal knowledge, even in this abstract form, can be quite useful computationally.

This paper is structured as follows. We start with some technical preliminaries and a review of the key results in [Darwiche, 2020]. We then study two key ingredients which are needed to formally define the causal treewidth: jointree thinning and machanism replication. We finally define causal treewidth and present some experimental results that shed more light on this notion and its underlying ingredients. Proofs of all results can be found in the appendix.

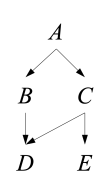
2 TECHNICAL PRELIMINARIES

Variables are discrete and denoted by uppercase letters (e.g., X) and their values are denoted by lowercase letters (e.g., x). Sets of variables are denoted by boldface, uppercase letters (e.g., $\mathbf{X}$) and their instantiations are denoted by boldface, lowercase letters (e.g., $\mathbf{x}$). A Bayesian network is specified by a DAG G and one conditional probability table (CPT) $f(X, \mathbf{P})$ for each node X and its parents $\mathbf{P}$ where $f(x, \mathbf{p}) = \Pr(x|\mathbf{p})$ (hence, $\sum_x f(x, \mathbf{p}) = 1$). A CPT is an instance of a *factor* which is a function $f(\mathbf{X})$ that maps instantiations $\mathbf{x}$ to non-negative numbers. We are particularly interested in factors that specify functional dependencies.

Definition 1. *A factor $f(X, \mathbf{P})$ is said to be a* <u>*mechanism*</u> *for variable X iff $f(x, \mathbf{p}) \in \{0, 1\}$ and $\sum_x f(x, \mathbf{p}) = 1$.*

Intuitively, a mechanism for variable X fixes the value of this variable once the values of its parents are fixed.

There are two operations on factors, multiplication and sum-out, which allow us to define the semantics of Bayesian networks and the corresponding inference problem whose complexity we wish to bound using causal treewidth. The *product* of factors $f(\mathbf{X})$ and $g(\mathbf{Y})$ is another factor $h(\mathbf{Z})$, where $\mathbf{Z} = \mathbf{X} \cup \mathbf{Y}$ and $h(\mathbf{z}) = f(\mathbf{x})g(\mathbf{y})$ for the unique instantiations $\mathbf{x}$ and $\mathbf{y}$ that are compatible with instantiation $\mathbf{z}$. *Summing-out* variables $\mathbf{Y} \subseteq \mathbf{X}$ from factor $f(\mathbf{X})$ yields another factor $g(\mathbf{Z})$, where $\mathbf{Z} = \mathbf{X} \setminus \mathbf{Y}$ and $g(\mathbf{z}) = \sum_{\mathbf{y}} f(\mathbf{yz})$. We will use $\sum_{\mathbf{Y}} f$ to denote the resulting factor g. We will also use $\sum_{\mathbf{Z}}^{=} f$ to denote summing out of all variables from factor f except for variables $\mathbf{Z}$. That is, for a factor $f(\mathbf{X})$, we will write $\sum_{\mathbf{Z}}^{=} f$ to mean $\sum_{\mathbf{Y}} f$ where $\mathbf{Y} = \mathbf{X} \setminus \mathbf{Z}$.



The *joint distribution* of a Bayesian network is the product of its CPTs. The network on the right has CPTs $f_A(A)$, $f_B(AB)$, $f_C(AC)$, $f_D(BCD)$ and $f_E(CE)$. Its joint distribution is $\Pr(ABCDE) = f_A f_B f_C f_D f_E$. We can now compute the marginal over any variable by suming out all other variables. For example, the marginal over variable D is the factor $\Pr(D) = \sum_{ABCE} f_A f_B f_C f_D f_E = \sum_{\bar{D}}^{=} f_A f_B f_C f_D f_E$. It is this computation of node marginals that we will be bounding

using causal treewidth. Even though we will be operating in a Bayesian network context, for concreteness, this computational problem can be defined more abstractly as follows.

Definition 2. *Consider a DAG G with nodes $X_1, \ldots, X_n$ and let $\mathbf{P}_i$ be the parents of node X_i. Given a set of factors $f(X_i, \mathbf{P}_i)$ for $i = 1, \ldots, n$, the marginals problem is that of computing the factor $\sum_{\overline{X_i}} \prod_{i=1}^{n} f(X_i, \mathbf{P}_i)$ for each X_i.*

A node X_i and its parents $\mathbf{P}_i$ in a DAG will be called a *family*. As such, a factor $f(X_i, \mathbf{P}_i)$ will be called a *family factor*. The marginals problem does not place any restrictions on family factors so it is quite general. When these factors are CPTs, the marginals problem corresponds to the computation of marginals in a Bayesian network.

As mentioned earlier, if the Bayesian network has n nodes and treewidth w, marginals can be computed in $O(n \cdot \exp(w))$ time and space. The simplest proof of this result is based on the algorithm of variable elimination (VE) which applies more generally to the problem in Definition 2 [Zhang and Poole, 1996, Dechter, 1996]. VE is based on two theorems, the first allows us to sum out variables in any order.

Theorem 1. $\sum_{\mathbf{XY}} f = \sum_{\mathbf{X}} \sum_{\mathbf{Y}} f = \sum_{\mathbf{Y}} \sum_{\mathbf{X}} f$.

The second theorem allows us to pull out factors from sums.

Theorem 2. *If variables $\mathbf{X}$ appear in factor f but not in factor g, then $\sum_{\mathbf{X}} f \cdot g = g \sum_{\mathbf{X}} f$.*

Consider the factor $\sum_{ABDE} f(ACE)g(BCD)$. A direct computation of this factor multiplies factors f and g to yield the factor $h(ABCDE)$ and then sums out variables $ABDE$ from g. Using Theorem 1, we can arrange the above sum into $\sum_{AE} \sum_{BD} f(ACE)g(BCD)$. Using Theorem 2, we can arrange it further into $\sum_{AE} f(ACE) \sum_{BD} f(BCD)$. This is more efficient to compute as the largest factor constructed in the process will be over 3 instead of 5 variables.

Suppose we eliminate variables according to order π when computing a marginal and let $w + 1$ be the size of largest factor constructed in the process (the size of a factor is the number of its variables). The time and space complexity of VE can then be bounded by $O(n \cdot \exp(w))$ where n is the number of variables. The number w is called the *width* of order π. If the DAG has treewidth w then there must exist an elimination order of width w. Moreover, no elimination order can have a width less than w; see [Darwiche, 2009, Ch 6 & 9] for a detailed exposition of these results.

3 EXPLOITING UNKNOWN MECHANISMS

Two new theorems were added to VE by Darwiche [2020] which enabled the exploitation of unknown causal mechanisms. We will use $\mathcal{F}$, $\mathcal{G}$, $\mathcal{H}$ next to denote sets of factors, where each set is interpreted as a product of its factors.

Algorithm 1 Complete Replication

```
procedure REPLICATE(DAG G, Family factors Σ)
    for each functional node X in G (bottom-up traversal) do
        if X is a leaf then continue
        n ← number of X-feeding factors in Σ
        Σ ← Σ ∪ {n − 1 replicas of the mechanism for X}
    return Σ
```

Theorem 3 ([Darwiche, 2020])**.** *Let f be a mechanism for variable X. If $f \in \mathcal{G}$ and $f \in \mathcal{H}$, then $\mathcal{G} \cdot \mathcal{H} = \mathcal{G} \sum_X \mathcal{H}$.*

According to this result, if a mechanism for X appears in both parts of a product, then variable X can be summed out from one part without changing the value of the product.

Corollary 1 ([Darwiche, 2020])**.** *If f is a mechanism for X, $f \in \mathcal{G}$ and $f \in \mathcal{H}$, then $\sum_X \mathcal{G} \cdot \mathcal{H} = (\sum_X \mathcal{G})(\sum_X \mathcal{H})$.*

That is, if a mechanism for X appears in both parts of a product, we can sum out variable X from the product by independently summing it out from each part. Corollary 1 may appear unusable as it is predicated on multiple occurrences of a mechanism whereas the factors of a Bayesian network contain a single mechanism for each variable. This is where the second theorem comes in: *replicating* mechanisms in a product does not change the product value.

Theorem 4 ([Darwiche, 2020])**.** *For mechanism f, if $f \in \mathcal{G}$, then $f \cdot \mathcal{G} = \mathcal{G}$.*

Consider the factor $\alpha = \sum_X f(XY)g(XZ)h(XW)$. VE has to multiply factors f, g and h before summing out variable X, therefore constructing a factor over four variables $XYZW$. However, if factor f is a mechanism for variable X, then we can replicate it by Theorem 4: $\alpha = f(XY)g(XZ)f(XY)h(XW)$. Corollary 1 then gives $\alpha = \sum_X f(XY)g(XZ) \sum_X f(XY)h(XW)$. Hence, we can now compute factor α without having to construct any factor over more than three variables. As shown in [Darwiche, 2020], this technique can lead to exponential savings that are attained without needing to know the identity of mechanisms which is a major departure from earlier techniques.

As the above example shows, the exploitation of unknown mechanisms requires their replication. A specific replication strategy was mentioned briefly and informally in [Darwiche, 2020] and referred to as a "heuristic." We shall call it the *complete replication strategy* for a reason that will become apparent later. This strategy is described formally in Algorithm 1 and uses the following definition.

Definition 3. *A family factor $f(X, \mathbf{P})$ is said to be Y-feeding iff $Y \in \mathbf{P}$.*

Algorithm 1 works with a *multi-set* of factors Σ instead of a *set* since Σ may contain multiple instances of the same factor.

It starts with Σ containing all family factors and traverses the DAG G bottom up. When visiting a functional node X, it adds replicas of the mechanism for X to Σ. Algorithm 1 returns what is called a *replication* of family factors.

Definition 4. *A replication of factors $\mathcal{F}$ is a multi-set $\mathcal{F}' \supseteq \mathcal{F}$ obtained by replicating some of the mechanisms in $\mathcal{F}$.*

Even though a replication is technically a multi-set, we will simply refer to it as set for convenience. We will study (complete) mechanism replication extensively later.

A popular mechanization of VE is based on the notion of a *jointree*. We will review jointrees next as we shall use them to mechanize the exploitation of Theorems 3 and 4 and to formally define the notion of causal treewidth.

Definition 5. *A jointree for factors $\mathcal{F}$ is a tree in which every leaf node i is assigned a non-empty set of factors $\mathcal{F}_i$ where the sets $\{\mathcal{F}_i\}_i$ form a partition of factors $\mathcal{F}$.* [2]

When a factor appears in $\mathcal{F}_i$, we will say that leaf node i *hosts* the factor. The definition of a jointree ensures that every factor in $\mathcal{F}$ is hosted by exactly one leaf node. Figure 2 depicts a jointree for 11 factors, some of which are replicas.

A jointree induces edge and node labels as follows.

Definition 6. *The separator $\mathbf{S}_{ij}$ of edge (i, j) is the set of variables shared between factors on the i-side of the edge and factors on the j-side of the edge. The cluster $\mathbf{C}_i$ of node i is $\texttt{vars}(\mathcal{F}_i)$ if i is a leaf node and $\bigcup_j \mathbf{S}_{ij}$ otherwise. The width of a jointree is the size of its largest cluster minus one.*

Jointrees play at least two key roles. First, their structure provides a specific recipe for when to multiply factors and when to sum out variables when applying VE. Second, their separators and clusters explicate the factors constructed by VE so the sizes of these separators and clusters can be used to precisely determine the complexity of VE. We explain both roles next, starting with the following theorem which slightly deviates from how jointrees are traditionally used.

Theorem 5. *Consider a jointree for factors $f_1, \ldots, f_n$. Define the message from jointree node i to its neighbor j as:*

$$\mathcal{M}_{ij} = \begin{cases} \sum_{\overline{\mathbf{S}}_{ij}} \mathcal{F}_i & \text{for leaf node } i \\ \sum_{\overline{\mathbf{S}}_{ij}} \prod_{k \neq j} \mathcal{M}_{ki} & \text{for internal node } i \end{cases}$$

For jointree edge (i, j), $\mathcal{M}_{ij}\mathcal{M}_{ji} = \sum_{\overline{\mathbf{S}}_{ij}} f_1, \ldots, f_n$. [3]

[2] Standard jointrees allow factors to be assigned to any node. Assigning factors to leaves, even one factor per leaf, does not preclude jointrees with optimal width; see [Darwiche, 2009, Ch 9].

[3] If $f_1, \ldots, f_n$ are the family factors of a DAG G and if each jointree leaf is assigned exactly one factor, then every non-leaf in G will appear in some separator. To ensure that every variable appears in some separator, add an auxiliary child C_i for each leaf X_i in G with factor $g_i(C_i, X_i)$ such that $g_i(c_i, x_i) = 1$. Any jointree for factors $f_1, \ldots, f_n$ can be easily turned into a jointree of the same width for factors $f_1, \ldots, f_n, g_1, \ldots, g_m = f_1, \ldots, f_n$.

A message $\mathcal{M}_{ij}$ can be computed in $O(\exp(|\mathbf{C}_i|)$ time and space given messages $\mathcal{M}_{ki}$ for $k \neq j$. Since $|\mathbf{C}_i| \leq w + 1$, where w is the jointree width, all messages can be computed in $O(n \cdot \exp(w))$ time and space where n is the number of jointree factors. Given an elimination order of width w, one can always construct a jointree of width $\leq w$; see [Darwiche, 2009, Ch 9]. Hence, the mechanization of VE using jointrees preserves the treewidth complexity bound.

In our context, jointrees play a third key role as they provide a direct method for exploiting Theorems 3 and 4 as shown in [Darwiche, 2020]. Instead of computing a jointree for the original set of factors $\mathcal{F}$, one computes a jointree for a replication $\mathcal{F}' \supseteq \mathcal{F}$ as licensed by Theorem 4. One can then remove variables from separators and clusters in the expanded jointree based on Theorem 3 while preserving the soundness of the message passing algorithm. This reduces the jointree width and can therefore lead to an exponential reduction in complexity. As in [Darwiche, 2021b], we shall refer to the process of removing variables from separators and clusters as the process of *thinning a jointree*.[4] We will show in the next section that the thinning procedure in [Darwiche, 2020] is not complete though as it can miss opportunities that are licensed by Theorem 3. We will also provide a complete thinning procedure which will pave the way for the formal definition of causal treewidth.

4 JOINTREE THINNING

Suppose we have a replication $\mathcal{F}' \supseteq \mathcal{F}$ of some factors $\mathcal{F}$. Given a jointree for the replication $\mathcal{F}'$, we will next define the notion of a *jointree thinning* and show that our definition is optimal (i.e., no further thinning can be done while preserving the soundness of the message passing algorithm). The replication $\mathcal{F}'$ may not be optimal though. Constructing optimal replications will be discussed in the next section.

Definition 7. *A jointree node i is said to be X-connected to a factor f iff i hosts f or X appears in every separator on the path between i and some node j that hosts f.*

Definition 8. *A jointree thinning maps every edge (i, j) in the jointree to a set of variables $\mathbf{S}^\star_{ij} \subseteq \mathbf{S}_{ij}$, called a thinned separator, and satisfies two properties. First, for each functional variable $X \in \mathbf{S}^\star_{ij}$, we have:*

(a) Node i is not X-connected to an X-mechanism on the i-side of edge (i, j), or node j is not X-connected to an X-mechanism on the j-side of the edge.

(b) If node i is not a leaf, then $X \in \mathbf{S}^\star_{ik}$ for some $k \neq j$.

(c) If node j is not a leaf, then $X \in \mathbf{S}^\star_{jk}$ for some $k \neq i$.

Second, no other mapping from edges to supersets of $\mathbf{S}^\star_{ij}$ satisfies the above property.

[4] The term "thin jointree" was used earlier in the context of approximate inference [Bach and Jordan, 2001].

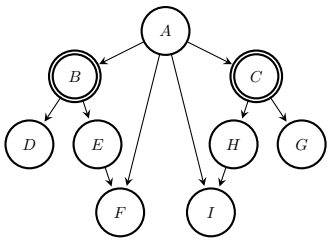
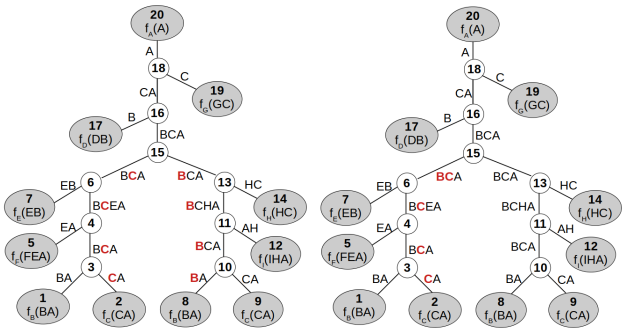


Figure 1: A Bayesian network with functional nodes B and C.



(a) jointree thinning of width 2 (b) jointree thinning of width 3

Figure 2: Two jointree thinnings. Each edge (i, j) is marked by separator $\mathbf{S}_{ij}$. Red variables are not in the thinned separator $\mathbf{S}^{\star}_{ij}$.

The separators $\mathbf{S}_{ij}$ of a jointree are determined by the locations of factors (the leaf nodes they are hosted at). Hence, the separators of a jointree are unique. However, a thinned separator $\mathbf{S}^{\star}_{ij}$ depends on both the locations of factors and other thinned separators. Hence, a jointree may have multiple thinnings. We define next the quality of a jointree thinning.

Definition 9. *A jointree thinning induces a* <u>*thinned cluster*</u> $\mathbf{C}^{\star}_i$ *for each jointree node i: If i is a leaf, $\mathbf{C}^{\star}_i = \texttt{vars}(\mathcal{F}_i)$; otherwise, $\mathbf{C}^{\star}_i = \bigcup_j \mathbf{S}^{\star}_{ij}$. The* <u>*width*</u> *of a jointree thinning is the size of its largest thinned cluster minus one.*

A jointree thinning leads to the notion of a causal jointree.

Definition 10. *A* <u>*causal jointree*</u> *is a jointree in which edges are annotated with thinned separators and nodes are annotated with thinned clusters. The* <u>*causal width*</u> *of a jointree is the smallest width attained by any of its causal jointrees.*

The width of a jointree can be determined by examining its cluster sizes. However, determining the causal width of a jointree is more involved as, in principle, it requires examining all thinnings of the jointree (causal jointrees).

Theorem 6. *The width of a jointree thinning and the causal width of a jointree are no greater than the jointree width.*

Figure 1 depicts a Bayesian network with two functional nodes (B, C) and nine factors $\mathcal{F} = f_A, f_B, \ldots, f_I$. Consider now the replication $\mathcal{F}' \supseteq \mathcal{F}$ which results from replicating mechanisms f_B and f_C once (that is, $\mathcal{F}'$ has 11 factors). Figure 2 depicts a jointree for the replication $\mathcal{F}'$ and two of its thinnings according to Definition 8. The one in Figure 2(a) has width 2. The one in Figure 2(b) has width 3.

Definition 8 of jointree thinning preserves the correctness of the message passing algorithm. Moreover, it cannot be improved on using only Corollary 1 and Theorem 4.

Theorem 7. *Theorem 5 continues to hold if we use thinned separators $\mathbf{S}^{\star}_{ij}$ (Definition 8) instead of classical separators $\mathbf{S}_{ij}$ (Definition 6). Moreover, if we assume that the statements in Corollary 1 and Theorem 4 are equivalences, then Theorem 5 does not hold if we remove any functional variable from a thinned separator.*

Definition 8 tells us what a thinning is but it does not tell us how to obtain one. We next provide a set of *thinning rules* that will generate every thinning admitted by Definition 8.

Theorem 8. *We can obtain a jointree thinning by starting with $\mathbf{S}^{\star}_{ij} = \mathbf{S}_{ij}$ and then removing variables from $\mathbf{S}^{\star}_{ij}$ according to the following rules, until no rules can be applied. Remove variable X from $\mathbf{S}^{\star}_{ij}$ if either*

(a) Node i is X-connected to an X-mechanism on the i-side of edge (i, j) and node j is X-connected to an X-mechanism on the j-side of the edge; or

(b) $X \notin \mathbf{S}^{\star}_{ki}$ for all $k \neq j$ when node i is not a leaf; or

(c) $X \notin \mathbf{S}^{\star}_{jk}$ for all $k \neq i$ when node j is not a leaf.

We will use $R_a(i, j, X)$ to mean that Rule (a) is applicable to variable X and edge (i, j) and call it a *rule application*. Similarly for $R_b(i, j, X)$ and $R_c(i, j, X)$. A jointree thinning can now be specified using a sequence of rule applications. The thinning in Figure 2(b) corresponds to $R_a(6, 15, C)$, $R_c(4, 6, C)$, $R_c(3, 4, C)$, $R_c(2, 3, C)$, $R_a(6, 15, B)$. The one in Figure 2(a) corresponds to $R_a(6, 15, C)$, $R_c(4, 6, C)$, $R_c(3, 4, C)$, $R_c(2, 3, C)$, $R_a(13, 15, B)$, $R_c(11, 13, B)$, $R_c(10, 11, B)$, $R_c(8, 10, B)$.

Definition 11. *A* <u>*thinning sequence*</u> *is list of rule applications $R^1, \ldots, R^n$ where each rule is valid when it is applied and no rules are applicable after the sequence terminates.*

Theorem 8 says that the thinning rules are sound. The following result says they are also complete.

Theorem 9. *Every causal jointree can be obtained using some thinning sequence.*

Two distinct thinning sequences may yield the same jointree thinning since the order of applying rules may not matter in some cases. The following result suggests a restriction on thinning sequences that does not compromise their ability to discover every possible jointree thinning.

Theorem 10. *Every jointree thinning can be obtained by a thinning sequence in which all applications of Rule (a) come before the applications of Rules (b,c).*

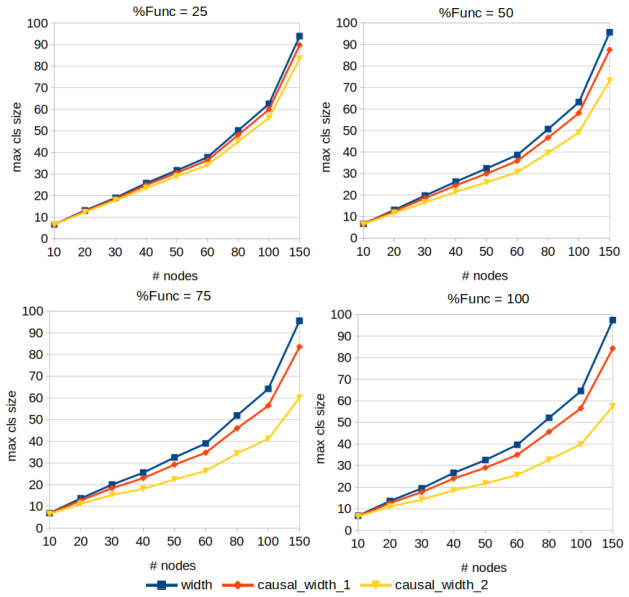


Figure 3: Comparing the complete thinning rules in Theorem 8 with the incomplete ones in [Darwiche, 2020].

That is, we can first exhaust all applications of Rule (a) and then apply Rules (b,c). In fact, once we exhaust all applications of Rule (a), applying Rules (b,c) becomes deterministic. In other words, the jointree thinning obtained by a thinning sequence is fully determined by its Rules (a).

Thinning sequences mechanize the thinning process but finding an optimal thinning sequence remains a computationally challenging task given the large number of such sequences (even under the above restriction). Hence, one needs either sophisticated search algorithms or a heuristic to decide which thinning rule to apply and when. One heuristic that we found effective is to prefer $R_a(i, j, X)$ with the largest $\mathbf{S}^\star_{ij}$, followed by X that is contained in the fewest neighboring separators, followed by minimizing the number of X-connected X-mechanisms on either side of edge (i, j).

[Darwiche, 2020] proposed three thinning rules that apply only to binary jointrees in which each node has one or three neighbors [Shenoy, 1996]. The rules are not complete though as they can miss thinnings admitted by Definition 8. As in the rules we defined above, one starts by setting thinned separators $\mathbf{S}^\star_{ij}$ to classical separators $\mathbf{S}_{ij}$ and then tries to remove variables from $\mathbf{S}^\star_{ij}$ using the rules. However, these rules can only be applied when visiting the jointree nodes in a particular order. A leaf node h is identified first and then nodes are visited based on their distance from h, where the closer nodes are visited first. Suppose we are visiting a non-leaf node i. Let p be its neighbor that is closest to leaf h and let c_1 and c_2 be its two other neighbors. The first two rules require the following conditions: $X \in \mathbf{S}^\star_{ic_1}$, $X \in \mathbf{S}^\star_{ic_2}$, an X-mechanism is hosted on the c_1-side of edge (c_1, i) and an X-mechanism is hosted on the c_2-side of edge (c_2, i). If we further have $X \in \mathbf{S}^\star_{ip}$, the first rule licenses the removal of X from either $\mathbf{S}^\star_{ic_1}$ or $\mathbf{S}^\star_{ic_2}$. If $X \notin \mathbf{S}^\star_{ip}$, the second rule licenses the removal of X from both $\mathbf{S}^\star_{ic_1}$ and $\mathbf{S}^\star_{ic_2}$. The final rule applies to the single neighnor r of leaf h, allowing us to remove variable X from $\mathbf{S}^\star_{hr}$ when an X-mechanism is hosted at leaf h and also at some other leaf in the jointree. The first rule involves a choice which is made using a heuristic described in [Darwiche, 2020].

Consider now the Bayesian network in Figure 1 and its thinned jointree in Figure 2(a) which has width 2. The best thinning that can be obtained by the rules in [Darwiche, 2020] has width 3, regardless of which leaf node h we choose and regardless of what choices we make when applying the first rule. Figure 3 depicts a comparison between these incomplete rules and the complete ones in Theorem 8 on random (binary) jointrees, for the factors of complete replications generated by Algorithm 1. The plots in this figure vary the number of Bayesian network nodes from 10 to 150 and consider different percentages of functional nodes (25, 50, 75 and 100). They report the mean of maximal cluster size (width+1) over 10 jointrees for each data point. The plots are for the cluster sizes of (1) a classical jointree (blue), (2) a causal jointree obtained by the incomplete rules (red) and (2) a causal jointree obtained by the proposed complete rules (yellow). Four patterns are clear: more thinning takes place as we increase the number of functional nodes; the complete thinning rules are much more effective; the gap between the two sets of rules grows as we increase the number of Bayesian network nodes and the number of functional nodes; the exploitation of unknown mechanisms can lead to significant reduction in inference complexity.

5 MECHANISM REPLICATION

The sound and complete definition of thinning that we developed in the previous section was with respect to a particular replication $\mathcal{F}$ and a particular jointree for the factors in $\mathcal{F}$. Some replications are better than others in that they lead to causal jointrees of smaller width. We formalize this next.

Definition 12. *The width of a replication $\mathcal{F}$ is defined as the minimum width attained by any causal jointree for $\mathcal{F}$.*

Given a replication $\mathcal{F}$, we need to examine two search spaces before we can determine its width. First, we must choose a jointree for the factors in $\mathcal{F}$. Second, we must choose a particular thinning of the jointree. Hence, determining the width of a replication is not a straightforward task. Moreover, the width of a replication is not the only measure of its quality as we need to also consider its size. This is a critical issue that was not discussed in [Darwiche, 2020, 2021b] and that we need to explore carefully before we are ready to provide the formal definition of causal treewidth.

The size of a replication is the number of factors it contains (replicas are counted individually). To highlight the importance of a replication size, consider two replications $\mathcal{F}_1$ and

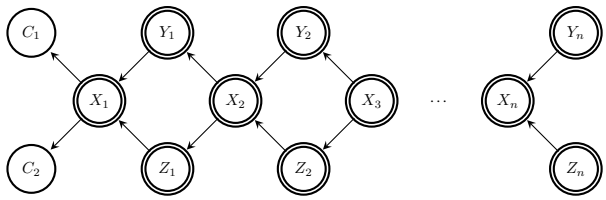


Figure 4: A DAG with an exponential complete replication.

$\mathcal{F}_2$ with respective sizes n_1 and n_2. Suppose now that replication $\mathcal{F}_1$ has width w_1 and replication $\mathcal{F}_2$ has width w_2. This means that there exists a causal jointree for replication $\mathcal{F}_1$ of width w_1 and no other causal jointree can have a smaller width (and similarly for replication $\mathcal{F}_2$). If we use these optimal causal jointrees, inference using these replications can be done in $O(n_1 \cdot \exp(w_1))$ and $O(n_2 \cdot \exp(w_2))$ time and space, respectively. One may be tempted to choose the replication with smaller width since complexity is exponential in width but linear in size. However, the size of a replication may also be exponential as we show next. In fact, the key result of this section is that the replication strategy proposed in [Darwiche, 2020], shown in Algorithm 1, satisfies two interesting properties. First, it is optimal: no other replication strategy will have a smaller width. Second, it can lead to replications of exponential size. We will in fact provide a bound on the size of replications produced by this strategy and suggest how it can be improved to avoid a blow up in replication size.

We start with the following result which shows that excessive replication can never hurt width.

Theorem 11. *Consider two replications $\mathcal{F}_1$ and $\mathcal{F}_2$ of some factors $\mathcal{F}$ where $\mathcal{F} \subseteq \mathcal{F}_1 \subseteq \mathcal{F}_2$. If the width of replication $\mathcal{F}_1$ is w, then the width of replication $\mathcal{F}_2$ is $\leq w$.*

We next identify a class of replications that possess some significant properties.

Definition 13. *A replication $\mathcal{F}$ is complete iff it satisfies the following property for each functional variable X and its mechanism f_X. If n is the number of X-feeding factors in $\mathcal{F}$ and $n > 0$, then $\mathcal{F}$ contains n replicas of mechanism f_X. Otherwise, $\mathcal{F}$ contains only one replica of f_X.*

The first property of complete replications is uniqueness.

Theorem 12. *The family factors of any DAG have a unique complete replication. Moreover, the replications generated by Algorithm 1 are complete.*

The second property of complete replications is optimality.

Theorem 13. *Let $\mathcal{F}$ and $\mathcal{F}'$ be two replications of the family factors of a DAG. If replication $\mathcal{F}$ is complete, then its width is no greater than the width of replication $\mathcal{F}'$.*

The third property of complete replications is that their size can be exponential. Consider the family of DAGs in Figure 4 which have $3n + 2$ nodes for $n \geq 1$. Algorithm 1, which generates complete replications, starts with a set Σ containing the $3n+2$ family factors. It then visits functional nodes bottom-up and replicates their mechanisms. One can easily show that after visiting functional node X_i, the set Σ will contain 2^i replicas of the mechanism for X_i.

We next provide a bound on the number of factors in a complete replication which suggests a method for controlling the blow up in its size.

Definition 14. *A functional chain of length k in a DAG is a set of functional nodes $n_1, \ldots, n_k$ where node n_i is a parent of node n_{i+1} for $i = 1, \ldots, k-1$.*

Theorem 14. *Consider a DAG with n nodes. Let c be the largest number of children for any node and let k be the length of longest functional chain. The (unique) complete replication of this DAG will contains at most nc^k factors.*

This bound immediately provides a method for controlling the number of replicas in a complete replication. If we treat variable $X_{n/2}$ in Figure 4 as a non-functional variable, the length of the largest functional chain will be cut by half. Hence, by selectively ignoring some functional variables we can bound the size of functional chains and therefore ensure that Algorithm 1 will produce replications with size that is polynomial in the number of DAG nodes.

We are now ready to define causal treewidth formally.

Definition 15. *Consider a DAG G with n nodes, some of which are declared as functional. The causal treewidth of DAG G is the smallest width attained by any replication $\mathcal{F}$ for G where the size of $\mathcal{F}$ is polynomial in n.*

For the class of DAGs with bounded functional chains, one can use the complete replication $\mathcal{F}$ to determine the treewidth of the DAG. That is, determining treewidth becomes a matter of finding an optimal causal jointree for the factors in $\mathcal{F}$. The situation is more intricate for DAGs with unbounded functional chains. The complete replication cannot be used in this case and one must search among replications of polynomial size. It remains to be seen whether the space of replications to be explored can be restricted to subsets of the complete replication as suggested earlier. This is a subject of future work.

We will finally report an experiment that reveals the importance of replication strategies and how such strategies interact with jointree construction methods.

We first note that classical methods for constructing jointrees do not directly apply to the construction of jointrees for replications. To see why, consider a set of factors $\mathcal{F}$ with no replicas. The classical method for constructing a jointree for such factors is to first construct a *primal graph.* This is a

% func	replication type							
	c	$5c$	$10c$	$15c$	$20c$	$25c$	$30c$	complete
25%	126.2	334.1	594.6	854.3	1114.1	1373.6	1634.2	181.1
50%	151.2	561.4	1076.0	1588.9	2104.1	2618.3	3132.3	513.4
75%	178.7	799.3	1575.6	2352.9	3128.2	3907.0	4682.3	2564.3
100%	187.0	867.7	1721.0	2573.5	3429.6	4280.1	5132.8	4448.0

Table 1: Average size of replications. Replication nc means: the number of replicas for a node X is between $(n-1)c$ and nc where c is the number of children for node X.

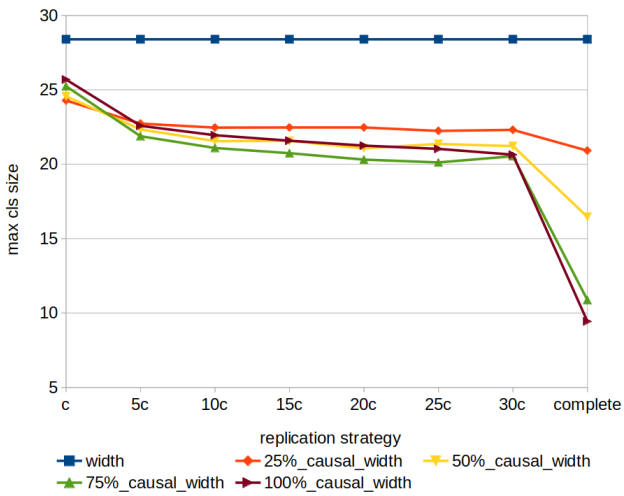


Figure 5: Illustrating the impact of replication strategies.

graph with nodes corresponding to the variables in factors $\mathcal{F}$ and which includes an edge between two variables iff they appear in the same factor.[5] Consider now a replication $\mathcal{F}'$ of factors $\mathcal{F}$. It follows immediately that the primal graph of $\mathcal{F}'$ is precisely the primal graph of $\mathcal{F}$. Hence, a classical jointree construction method will produce the same jointree for factors $\mathcal{F}$ and for all their replications. [Darwiche, 2020] proposed a jointree construction method that targets complete replications. For a non-leaf functional variable X, the method uses a distinct name for X in each of its n replicas and these distinct names are also used in the X-feeding factors whose count is also n. A jointree is then constructed using a classical technique followed by a reversal of the renaming process. While this method proved generally effective, it applies only to complete replications.

The experiment we conducted compared the complete replication strategy with random replications of increasing size, while varying the percentage of functional nodes in a Bayesian network (25, 50, 75, 100). The comparison was based on constructing jointrees using the minfill heuristic (see Footnote 5). We used the method of [Darwiche, 2020] for complete replications, and adapted it somewhat arbitrarily for random replications. In particular, when the number of X-feeding factors did not match the number of X-mechanisms, we renamed variables in the X-feeding factors distinctly to the extent possible and randomly thereafter.

Table 1 shows the size of random and complete replications, with some random replications being larger than complete replications. Figure 5 shows the mean maximal cluster size (width+1) for jointrees and causal jointrees where each data point is an average over 100 random Bayesian networks, each containing 100 nodes. A few patterns are clear. First, the causal width is always smaller than the width, and quite substantially smaller, even when using random replications. Second, complete replications always produced a smaller causal width compared to random replications, particularly when the number of functional nodes is largest (100%). Third, increasing the size of a random replication almost always correlated with decreasing the causal width but up to a certain point after which increasing the size of a replication did not help. The few exceptions to this pattern highlight the suboptimality of the jointree construction method we used (see Theorem 11) and the suboptimality of the heuristic for applying thinning rules. Beyond emphasizing some of the theoretical results we presented earlier, this experiment further highlights the practical significance of causal treewidth and causal jointrees as they can lead to an exponential reduction in inference complexity. The experiment also highlights the need for developing principled jointree construction methods that target replications which are not complete, and highlight the need for further heuristics to guide the application of thinning rules.

6 CONCLUSION

We provided a formal definition of the notion of causal treewidth, which dominates the classical and influential notion of treewidth. We also studied the three ingredients needed to define causal treewidth: mechanism replication, jointree construction and jointree thinning which yields causal jointrees. On the first front, we presented a number of results about a replication strategy that we called complete replication, showing that it is optimal while providing a bound on the size of replications it produces and suggesting a technique for controlling their size. On the second front, we highlighted the relevance (and irrelevance) of classical jointree construction methods to the construction of jointrees for replications. On the third front, we provided a complete characterization of causal jointrees and provided three thinning rules that are sound and complete for generating causal jointrees. We also proved some properties of these rules which can be of practical significance. We finally presented some experimental results to shed further light on the developments in this paper, which also showed that causal jointrees can lead to exponential improvements in the complexity of inference in comparison to jointrees.

[5] There are various methods for constructing a jointree based on a primal graph; see [Darwiche, 2009, Ch 9]. One of the popular methods is to construct a low-width elimination order for the primal graph using the *minfill* heuristic and to then convert the order into a jointree of no greater width. This is what we used.

References

Durgesh Agrawal, Yash Pote, and Kuldeep S. Meel. Partition function estimation: A quantitative study. In *IJCAI*, pages 4276–4285. ijcai.org, 2021.

Francis R. Bach and Michael I. Jordan. Thin junction trees. In *NIPS*, pages 569–576. MIT Press, 2001.

Hans L. Bodlaender. Treewidth: Characterizations, applications, and computations. In *WG*, volume 4271 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006.

Craig Boutilier, Nir Friedman, Moisés Goldszmidt, and Daphne Koller. Context-specific independence in bayesian networks. In *UAI*, pages 115–123. Morgan Kaufmann, 1996.

Mark Chavira and Adnan Darwiche. Compiling Bayesian networks with local structure. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 1306–1312. Professional Book Center, 2005. URL `http://ijcai.org/Proceedings/05/Papers/0931.pdf`.

Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artif. Intell.*, 172(6-7):772–799, 2008. doi: 10.1016/j.artint.2007.11.002. URL `https://doi.org/10.1016/j.artint.2007.11.002`.

Mark Chavira, Adnan Darwiche, and Manfred Jaeger. Compiling relational Bayesian networks for exact inference. *Int. J. Approx. Reason.*, 42(1-2):4–20, 2006.

Yizuo Chen, Arthur Choi, and Adnan Darwiche. Supervised learning with background knowledge. In *PGM*, volume 138 of *Proceedings of Machine Learning Research*, pages 89–100. PMLR, 2020.

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

Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

Adnan Darwiche. An advance on variable elimination with applications to tensor-based computation. In *ECAI*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 2559–2568. IOS Press, 2020.

Adnan Darwiche. Tractable Boolean and arithmetic circuits. In Pascal Hitzler and Md Kamruzzaman Sarker, editors, *Neuro-Symbolic Artificial Intelligence: The State of the Art*, volume 342 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021a. https://arxiv.org/abs/2202.02942.

Adnan Darwiche. Causal inference with tractable circuits. In *Why-21 Workshop, NeurIPS*, 2021b. https://arxiv.org/abs/2202.02891.

Rina Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 211–219, 1996.

Rina Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003.

F. V. Jensen, S. Lauritzen, and K. Olesen. Bayesian updating in recursive graphical models by local computation. *Computational Statistics Quarterly*, 4:269–282, 1990.

David Larkin and Rina Dechter. Bayesian inference in the presence of determinism. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics (AISTATS)*, 2003.

Judea Pearl. *Causality*. Cambridge University Press, 2000.

Judea Pearl and Dana Mackenzie. *The Book of Why: The New Science of Cause and Effect*. Basic Books, 2018.

Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3): 309–322, 1986.

Prakash P. Shenoy. Binary join trees. In *UAI*, pages 492–499. Morgan Kaufmann, 1996.

Nevin Lianwen Zhang and David Poole. Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research*, 5:301–328, 1996.

A PROOFS

Our proofs are ordered sightly differently than the corresponding results in the paper as we need some results when proving others. In the upcoming proofs, we will also use f_X to denote a mechanism for variable X. We will also say that a leaf node i in a jointree *contains* variable X iff X appears in a factor that is hosted at leaf node i.

A.1 PROOF OF THEOREM 5

See [Darwiche, 2009, Ch 7].

A.2 PROOF OF THEOREM 6

Follows directly from Definitions 9 and 10.

A.3 PROOF OF THEOREM 8

In this proof, we will assume we have a mapping from jointree edges (i, j) to sets $\mathbf{S}'_{ij} \subseteq \mathbf{S}_{ij}$ that satisfy conditions (a,b,c) of Definition 8 (that is, these sets satisfy only the first part of this definition but not the second part so they are not necessarily thinned separators). We will also say that jointree nodes i and j are $(X, \mathbf{S}')$-connected iff $i = j$ or variable X appears in every set $\mathbf{S}'$ that is attached to an edge on the path between i and j. We will first state and prove two lemmas which we need for the proof of this theorem.

Lemma 1. *If $X \in \mathbf{S}'_{ij}$, then node i is $(X, \mathbf{S}')$-connected to some leaf node on the i-side of edge (i, j) which contains variable X and node j is $(X, \mathbf{S}')$-connected to some leaf node on the j-side of the edge which also contains X.*

Proof. Suppose node i is not $(X, \mathbf{S}')$-connected to some leaf node on the i-side of edge (i, j) which contains X. Consider a longest path $i, \ldots, r, l$ on the i-side of the edge such that X appears in $\mathbf{S}'$ for each edge on the path. If l is a leaf node, then it must contain variable X by definition of separators, which is a contradiction. Suppose now that l is not a leaf node. Then $X \notin \mathbf{S}'_{lk}$ for $k \neq r$, otherwise the path would not be longest. This contradicts the assumed condition (b) of Definition 8 on sets $\mathbf{S}'$. Hence, node i must be $(X, \mathbf{S}')$-connected to some leaf node that contains X on the i-side of edge (i, j). We can similarly show the second part of the lemma. □

Lemma 2. *Suppose sets $\mathbf{S}'_{ij}$ were obtained by exhausting thinning rules on separators $\mathbf{S}_{ij}$. If node k is $(X, \mathbf{S}')$-connected to some leaf node that contains X, then node k is $(X, \mathbf{S}')$-connected to exactly one leaf node that hosts f_X.*

Proof. Suppose node k is $(X, \mathbf{S}')$-connected to some leaf node that contains X. Then node k cannot be $(X, \mathbf{S}')$-connected to more than one leaf node hosting f_X; otherwise, Rule (a) will apply. We next show that node k must be $(X, \mathbf{S}')$-connected to at least one mechanism f_X. We will show this by induction on the number of rule applications. We will use $\mathbf{S}^n$ to denote the state of separators after the nth rule application. For the base case (before any thinning rules are applied), node k must be (X, S^0)-connected to some f_X by the definition of separators in a jointree. For the inductive step, suppose node k is $(X, \mathbf{S}^n)$-connected to some leaf node that contains X only if node k is $(X, \mathbf{S}^n)$-connected to at least one leaf node that hosts f_X. Suppose now that node k is $(X, \mathbf{S}^{n+1})$-connected to some leaf node that contains X. We will next show that node k must be $(X, \mathbf{S}^{n+1})$-connected to at least one mechanism f_X.

First, node k must be $(X, \mathbf{S}^n)$-connected to some leaf node that contains X. Hence, by the induction hypothesis, node k must be $(X, \mathbf{S}^n)$-connected to at least one mechanism f_X. We will consider an edge (i, j) such that $X \in \mathbf{S}^n_{ij}$ and $X \notin \mathbf{S}^{n+1}_{ij}$ (such an edge must exist) and do a case analysis on which rule applied to this edge.

Case: Rule (a). We will show next that node k must be $(X, \mathbf{S}^{n+1})$-connected to at least one mechanism f_X while assuming that node k is on the i-side of edge (i, j). A similar argument will show the same if node k is on the j-side of the edge. Suppose node k is on the i-side of edge (i, j). If node k is $(X, \mathbf{S}^n)$-connected to some f_X on the i-side, then node k is $(X, \mathbf{S}^{n+1})$-connected to the same f_X on the i-side. If node k is $(X, \mathbf{S}^n)$-connected to some f_X on the j-side, then node k is $(X, \mathbf{S}^{n+1})$-connected to node i. By the definition of Rule (a), node i must be $(X, \mathbf{S}^{n+1})$-connected to some f_X on the i-side. Therefore, node k must be $(X, \mathbf{S}^{n+1})$-connected to the same f_X on the i-side.

Case: Rule (b). Then none of the neighboring separators except $\mathbf{S}^n_{ij}$ contains X. If node k is on the i-side, then node k must be $(X, \mathbf{S}^n)$ to some f_X on the i-side and therefore must be $(X, \mathbf{S}^{n+1})$ to the same f_X on the i-side. The same argument applies if node k is on the j-side.

Case: Rule (c). Symmetric to the previous case. □

Proof of Theorem 8. First note that conditions (a,b,c) of Definition 8 hold when we exhaust thinning rules. Suppose sets $\mathbf{S}'_{ij}$ were obtained by exhausting thinning rules on separators $\mathbf{S}_{ij}$ (and hence satisfy the three conditions). We just need to prove that no supersets of $\mathbf{S}'_{ij}$ satisfy these conditions. Suppose by contradiction such supersets $\mathbf{S}''_{ij}$ exist and consider an edge (i, j) such that $X \in \mathbf{S}''_{ij}$ and $X \notin \mathbf{S}'_{ij}$. By Lemma 1 since sets $\mathbf{S}''_{ij}$ satisfy the three conditions, node i must $(X, \mathbf{S}'')$-connect to some leaf node k on the i-side which contains X. Since leaf node k contains X, it $(X, \mathbf{S}')$-connects to itself. By Lemma 2, the leaf node k is $(X, \mathbf{S}')$-connected to some f_X on the i-side since $X \notin \mathbf{S}'_{ij}$. Hence, leaf node k is also $(X, \mathbf{S}'')$-connected to the same f_X on the i-side since sets $\mathbf{S}''_{ij}$ are supersets of $\mathbf{S}'_{ij}$. We

have shown that node i is $(X, \mathbf{S}'')$-connected to leaf node k which is $(X, \mathbf{S}'')$-connected to some f_X on the i-side, therefore node i is $(X, \mathbf{S}'')$-connected to some f_X on the i-side. By a similar argument, node j is $(X, \mathbf{S}'')$-connected to some f_X on the j-side. However, this implies that the supersets $\mathbf{S}''_{ij}$ violate condition (a), which is a contradiction. Hence, the supersets $\mathbf{S}''_{ij}$ cannot exist. □

A.4 PROOF OF THEOREM 9

We need the following lemma which states the same property of Lemma 2 except under different conditions. The proof of this lemma uses the notion of "a closest leaf k to node i which hosts mechanism f_X." This is a leaf node that hosts f_X where the path $i = p_0, p_1, \ldots, p_{n-1}, p_n = k$ has a minimal number of sets $\mathbf{S}^\star_{p_i p_{i+1}}$ that do not contain X.

Lemma 3. *Consider thinned separators $\mathbf{S}^\star$ according to Definition 8 and let X be a functional variable. If node i is $(X, \mathbf{S}^\star)$-connected to some leaf node that contains X, then node i is $(X, \mathbf{S}^\star)$-connected to exactly one leaf node that hosts mechanism f_X.*

Proof. Suppose node i is $(X, \mathbf{S}^\star)$-connected to some leaf node that contains X. Node i cannot be $(X, \mathbf{S}^\star)$-connected to two different leaves that host f_X as this would violate condition (a) of Definition 8. Suppose now that node i is not $(X, \mathbf{S}^\star)$-connected to any mechanism f_X. We will next show a contradiction. Consider the path $i = p_0, p_1, \ldots, p_{n-1}, p_n = k$ where k is a closest leaf to node i which hosts mechanism f_X. We claim that adding X to all sets $\mathbf{S}^\star_{p_i p_{i+1}}$ which do not contain X on the path results in a jointree thinning that still satisfies conditions (a,b,c) of Definition 8. This would be a contradiction as it implies there is a superset of thinning $\mathbf{S}^\star$ that satisfies these conditions.

To show the above claim, note that conditions (b,c) will immediately continue to be satisfied if we add variables to sets $\mathbf{S}^\star_{p_i p_{i+1}}$. We next show that condition (a) will continue to be satisfied as well.

Suppose we are adding X to sets $\mathbf{S}^\star_{p_i p_{i+1}}$ which do not contain variable X using the given order of these sets. Let $\mathbf{S}'$ be the separators after adding X to $\mathbf{S}^\star_{p_i p_{i+1}}$ when the first violation to condition (a) takes place. Then node p_i will be $(X, \mathbf{S}')$-connected to a leaf node p'_i that hosts mechanism f_X on the p_i-side of edge (p_i, p_{i+1}). This contradicts with the definition of k as p'_i will be closer to i than k. □

Proof of Theorem 9. Given a jointree thinning $\mathbf{S}^\star$ that satisfies Definition 8, we will next show how to construct a thinning sequence that produces it.

A key observation is that rule applications for a variable X are independent of rule applications for a variable $Y \neq X$; that is, we can always rearrange a thinning sequence so rules that apply to the same variable are consecutive. Hence, we will construct a thinning sequence that produces $\mathbf{S}^\star$ by constructing a set of rule applications for each variable and then paste them together.

To construct the rule applications for variable X, we start with some leaf node l which hosts mechanism f_X and then traverse nodes away from l. Suppose we are visiting node i now which has a neighbor j that has not been visited. If $X \notin \mathbf{S}^\star_{ij}$ and $X \in \mathbf{S}_{ij}$, we consider two cases. If $X \notin \mathbf{S}^\star_{jk}$ for all k, we add $R_c(i, j, X)$. Otherwise, we add $R_a(i, j, X)$. Every variable that has been thinned will now be accounted for by a rule. Moreover, these rules will be applicable in the reverse order in which they have been constructed. Each $R_a(i, j, X)$ will be applicable by Lemmas 1 and 3. Each $R_c(i, j, X)$ will be applicable by definition. □

A.5 PROOF OF THEOREM 10

Lemma 4. *Consider a thinning sequence $\mathbf{R} = \{\mathbf{X}, R_k, \mathbf{Y}, R_t, \mathbf{Z}\}$ where $\mathbf{X} = \{R_1, \ldots, R_{k-1}\}$, $\mathbf{Y} = \{R_{k+1}, \ldots, R_{t-1}\}$ and $\mathbf{Z} = \{R_{t+1}, \ldots, R_n\}$. Suppose R_k and R_t are applications of Rule (a) and no member of $\mathbf{Y}$ is an application of Rule (a). Then the following is a valid thinning sequence $\mathbf{R}' = \{\mathbf{X}, R_k, R_t, \mathbf{Y}, \mathbf{Z}\}$.*

Proof. When applying the thinning sequence $\mathbf{R}$, we start with $\mathbf{S}^\star_{ij} = \mathbf{S}_{ij}$ and reduce a set $\mathbf{S}^\star_{ij}$ after each rule application. The key observation here is that if an application of Rule (a) is valid at some state of the thinned separators, it will be valid at any earlier state of these separators (because no thinned separator can be smaller at an earlier state). Moreover, if an application of Rules (b,c) is valid at some state of the thinned separators, it will be valid at any later state of these separators. □

Proof of Theorem 10. Consider a thinning sequence $\mathbf{R}$. We can apply Lemma 4 repeatedly to obtain a valid thinning sequence $\mathbf{R}'$ that has the same rule applications as $\mathbf{R}$ and in which Rules (a) appear before Rules (b,c). The sequences $\mathbf{R}$ and $\mathbf{R}'$ generate the same jointree thinning since they contain the exact same rule applications. □

A.6 PROOF OF THEOREM 7

We will first prove the first part of the theorem (soundness) and then prove the second part (completeness).

Soundness. Our proof of soundness will start with the correctness of the message passing algorithm using classical separators and then show that the algorithm continues to be sound after we apply a thinning rule to remove a variable from some separator (recall that every jointree thinning can be obtained by a sequence of thinning rules). We will use

$\mathbf{S}'$ to indicate the state of separators after some rule applications. We will also say that a jointree node i is *strongly* $(X, \mathbf{S}')$-connected to a factor f iff i hosts f or $\texttt{vars}(f)$ appears in every separator $\mathbf{S}'_{kl}$ on the path between node i and some leaf node j that hosts f. Similarly, we will say that jointree nodes i and j are *strongly* $(X, \mathbf{S}')$-connected iff $i = j$ or variables $\texttt{vars}(f_X)$ appear in every separator $\mathbf{S}'_{kl}$ on the path between i and j. The thinning rules applied to some variable X do not interact with the thinning rules applied to another variable Y. Hence, in this proof and the next lemma, we will assume that all thinning rules are applied according to a reverse topological ordering π of the variables in the underlying DAG.

Lemma 5. *Suppose separators $\mathbf{S}'$ are obtained by applying some thinning rules to classical separators (using ordering π). For jointree edge (i, j), node i is $(X, \mathbf{S}')$-connected to some f_X on the i-side of the edge and node j is $(X, \mathbf{S}')$-connected to some f_X on the j-side of the edge iff i is strongly $(X, \mathbf{S}')$-connected to some f_X on the i-side and j is strongly $(X, \mathbf{S}')$-connected to some f_X on the j-side.*

Proof. The if part follows from the fact that $X \in \texttt{vars}(f_X)$. We next show the only-if part. Suppose node i is $(X, \mathbf{S}')$-connected to node l that hosts f_X on the i-side and node j is $(X, \mathbf{S}')$-connected to node r that hosts f_X on the j-side. By the property of classical separators, $\texttt{vars}(f_X) \subseteq \mathbf{S}_{xy}$ for all edges (x, y) on the path between l and r. Since the thinning rules are applied according to a reverse topological ordering of the variables, none of the variables in $\texttt{vars}(f_X)$ except for X are thinned from the separators. Therefore, $\texttt{vars}(f_X) \setminus \{X\} \subseteq \mathbf{S}'_{xy}$ for all edges (x, y) on the path between l and r. Together with the fact that node i is $(X, \mathbf{S}')$-connected to l and node j is $(X, \mathbf{S}')$-connected to r, we conclude that node i is strongly $(X, \mathbf{S}')$-connected to l and node j is strongly $(X, \mathbf{S}')$-connected to r. □

Given Lemma 5, X-connection and strong X-connection are now equivalent as we will be applying thinning rules according to order π as indicated above. We are now ready to proceed with the proof.

For jointree edge (i, j), let $\mathcal{M}'_{ij}$ and $\mathcal{M}'_{ji}$ denote the messages between i and j under separators $\mathbf{S}'$. Let $Pr(\mathbf{S}'_{ij}) = \sum^{=}_{\mathbf{S}'_{ij}} f_1, \ldots, f_n$. We will show $Pr(\mathbf{S}'_{ij}) = \mathcal{M}'_{ij}\mathcal{M}'_{ji}$ for all edges (i, j) by induction on rule applications.

We start with $Pr(\mathbf{S}_{ij}) = \mathcal{M}_{ij}\mathcal{M}_{ji}$ for all edges (i, j) by Theorem 5. We next show that the equivalence holds after each rule application. For each rule application, we will use $\mathbf{S}$ to denote the separators before the thinning and $\mathbf{S}'$ to denote the separators after the thinning. For simplicity, we use $f_X \in \mathcal{G}$ to denote that $\mathcal{G} = f_X \cdot \mathcal{F}$ for some factor $\mathcal{F}$.

—Suppose Rule (a) is applied so that $X \in \mathbf{S}_{ij}$ and $X \notin \mathbf{S}'_{ij}$. By definition, node i is strongly $(X, \mathbf{S})$-connected to some f_X hosted at l on the i-side and j is strongly $(X, \mathbf{S})$-connected to some f_X hosted at r on the j-side. Case $(k, z) = (i, j)$, then $\mathcal{M}'_{kz} = \sum_X \mathcal{M}_{ij}$, $\mathcal{M}'_{zk} = \sum_X \mathcal{M}_{ji}$, $\mathbf{S}'_{kz} = \mathbf{S}_{ij} \setminus \{X\}$. Since node i is strongly $(X, \mathbf{S})$-connected to f_X at l and node j is strongly $(X, \mathbf{S})$-connected to f_X at r, $f_X \in \mathcal{M}_{ij}$ and $f_X \in \mathcal{M}_{ji}$. Thus,

$$\begin{aligned} Pr(\mathbf{S}'_{ij}) &= \sum_X Pr(\mathbf{S}_{ij}) = \sum_X \mathcal{M}_{ij} \cdot \mathcal{M}_{ji} \\ &\overset{Cor.\ 1}{=} (\sum_X \mathcal{M}_{ij}) \cdot (\sum_X \mathcal{M}_{ji}) \\ &= \mathcal{M}'_{ij} \cdot \mathcal{M}'_{ji} \end{aligned}$$

Case (k, z) is an edge on the path between l and r but $(k, z) \neq (i, j)$. W.l.o.g., suppose (k, z) is on the i-side of the path and k is between l and z, then $\mathbf{S}'_{kz} = \mathbf{S}_{kz}$, $\mathcal{M}'_{kz} = \mathcal{M}_{kz} \ni f_X$. We expand message $\mathcal{M}_{zk}$ as follows:

$$\begin{aligned} \mathcal{M}_{zk} = &\sum^{=}_{\mathbf{S}_{z,k}} \xi_{n+1} \cdot (\sum^{=}_{\mathbf{S}_{p_n,z}} \xi_n \cdots \\ &(\sum^{=}_{\mathbf{S}_{p_1,p_2}} \xi_1 \cdot (\sum^{=}_{\mathbf{S}_{i,p_1}} \xi_0 \cdot \mathcal{M}_{ji}) \end{aligned}$$

where $(k, z, p_n, \ldots, p_1, i, j)$ is a path from k to j, and $\xi_0, \ldots, \xi_{n+1}$ are factors that represent the product of messages passed by other neighbors (that are not on the path between l and r) at each edge.

Since k is strongly $(X, \mathbf{S})$-connected to j, we have $\texttt{var}(f_X) \subseteq \mathbf{S}_{z,k}, \ldots, \texttt{var}(f_X) \subseteq \mathbf{S}_{i,p_1}$. Moreover, since j is strongly $(X, \mathbf{S})$-connected to r, we have $f_X \in \mathcal{M}_{ji}$. It follows that

$$\begin{aligned} &\mathcal{M}_{kz} \cdot \mathcal{M}_{zk} = (f_X \cdot \mathcal{F}) \cdot \mathcal{M}_{zk} \\ &= \mathcal{F} \cdot (\sum^{=}_{\mathbf{S}_{zk}} f_X \cdot \xi_{n+1} \cdots (\sum^{=}_{\mathbf{S}_{i,p_1}} \xi_0 \cdot \mathcal{M}_{ji})) \\ &= \mathcal{F} \cdot (\sum^{=}_{\mathbf{S}_{zk}} \xi_{n+1} \cdots (\sum^{=}_{\mathbf{S}_{i,p_1}} f_X \cdot \xi_0 \cdot \mathcal{M}_{ji})) \\ &= \mathcal{F} \cdot (\sum^{=}_{\mathbf{S}_{zk}} \xi_{n+1} \cdots (\sum^{=}_{\mathbf{S}_{i,p_1}} f_X \cdot \xi_0 \cdot (\sum_X \mathcal{M}_{ji}))) \\ &= \mathcal{F} \cdot (\sum^{=}_{\mathbf{S}_{zk}} \xi_{n+1} \cdots (\sum^{=}_{\mathbf{S}_{i,p_1}} f_X \cdot \xi_0 \cdot \mathcal{M}'_{ji})) \\ &= (f_X \cdot \mathcal{F}) \cdot (\sum^{=}_{\mathbf{S}_{zk}} \xi_{n+1} \cdots (\sum^{=}_{\mathbf{S}_{i,p_1}} \xi_0 \cdot \mathcal{M}'_{ji})) \\ &= \mathcal{M}'_{zk} \cdot \mathcal{M}'_{kz} = Pr(\mathbf{S}'_{zk}) \end{aligned} \tag{1}$$

Case (k, z) is not on the path between l and r, then the path between l and r must be either on the k-side or on the z-side. W.l.o.g., suppose the path is on the k-side, then $\mathbf{S}'_{kz} = \mathbf{S}_{kz}$, $\mathcal{M}'_{zk} = \mathcal{M}_{zk}$, we next show that $\mathcal{M}'_{kz} = \mathcal{M}_{kz}$. Let (t, h) be the edge where the path between k and i intersects with the path between l and r, where t is on the path between l and r but h is not. If we can show $\mathcal{M}'_{th} = \mathcal{M}_{th}$, then

$\mathcal{M}'_{kz} = \mathcal{M}_{kz}$ since other messages entering k are invariant. W.l.o.g. assume that t is on the i-side and we show $\mathcal{M}'_{th} = \mathcal{M}_{th}$. Let (c_l, t) and (c_r, t) be the two edges adjacent to t where c_l is between l and t and c_r is between r and t, then $\mathcal{M}_{th} = \xi \cdot \mathcal{M}_{c_l,t} \cdot \mathcal{M}_{c_r,t}$, where ξ denotes the product of messages entering t other than $\mathcal{M}_{c_l,t}$ and $\mathcal{M}_{c_r,t}$. Note that the thinning at (i, j) does not change ξ. Therefore, it suffices to show that $\mathcal{M}'_{c_l,t} \cdot \mathcal{M}'_{c_r,t} = \mathcal{M}_{c_l,t} \cdot \mathcal{M}_{c_r,t}$. Note that $\mathcal{M}'_{c_l,t} = \mathcal{M}_{c_l,t}$, $f_X \in \mathcal{M}_{c_l,t}$ and $f_X \in \mathcal{M}_{c_r,t}$. Similar to Equation 1, we expand $\mathcal{M}_{c_r,t}$ as follows

$$
\begin{aligned}
&\mathcal{M}_{c_l,t} \cdot \mathcal{M}_{c_r,t} = (f_X \cdot \mathcal{F}) \cdot \mathcal{M}_{c_r,t} \\
&= \mathcal{F} \cdot (\overset{=}{\sum}_{\mathbf{S}_{zk}} f_X \cdot \xi_{n+1} \cdots (\overset{=}{\sum}_{\mathbf{S}_{i,p_1}} \xi_0 \cdot \mathcal{M}_{ji})) \\
&= \mathcal{F} \cdot (\overset{=}{\sum}_{\mathbf{S}_{zk}} \xi_{n+1} \cdots (\overset{=}{\sum}_{\mathbf{S}_{i,p_1}} f_X \cdot \xi_0 \cdot \mathcal{M}_{ji})) \\
&= \mathcal{F} \cdot (\overset{=}{\sum}_{\mathbf{S}_{zk}} \xi_{n+1} \cdots (\overset{=}{\sum}_{\mathbf{S}_{i,p_1}} f_X \cdot \xi_0 \cdot (\sum_X \mathcal{M}_{ji}))) \\
&= \mathcal{F} \cdot (\overset{=}{\sum}_{\mathbf{S}_{zk}} \xi_{n+1} \cdots (\overset{=}{\sum}_{\mathbf{S}_{i,p_1}} f_X \cdot \xi_0 \cdot \mathcal{M}'_{ji})) \\
&= (f_X \cdot \mathcal{F}) \cdot (\overset{=}{\sum}_{\mathbf{S}_{zk}} \xi_{n+1} \cdots (\overset{=}{\sum}_{\mathbf{S}_{i,p_1}} \xi_0 \cdot \mathcal{M}'_{ji})) \\
&= \mathcal{M}'_{c_l,t} \cdot \mathcal{M}'_{c_r,t}
\end{aligned}
$$

where $(t, c_r, p_n, \ldots, i, j)$ is a path from t to j, and $\xi_0, \ldots, \xi_{n+1}$ are messages passed from other neighbors (that are not on the path between l and r) at each edge. Again, the "shift" of f_X is valid since $\texttt{var}(f_t) \subseteq \mathbf{S}_{c_r,t}, \ldots, \texttt{var}(f_t) \subseteq \mathbf{S}_{i,p_1}$.

—Suppose X is thinned from $\mathbf{S}_{ij}$ by Rule (b), then i is not a leaf and $X \notin \mathbf{S}_{ik}$ for all $k \neq j$. We first show that the message passing equation holds at $\mathbf{S}'_{ij}$. We have $\mathcal{M}'_{ij} = \sum_X \mathcal{M}_{ij} = \mathcal{M}_{ij}$ since no message entering i contains X. Hence, $Pr(\mathbf{S}'_{ij}) = \sum_X Pr(\mathbf{S}_{ij}) = \mathcal{M}_{ij} \cdot (\sum_X \mathcal{M}_{ji}) = \mathcal{M}'_{ij} \cdot \mathcal{M}'_{ji}$.

We next show that the message passing equation holds for other separators in $\mathbf{S}'$. Since $\mathcal{M}'_{ij} = \mathcal{M}_{ij}$, all messages passed to the edges on the j-side are invariant. Therefore, the message passing equation still holds for all $\mathbf{S}'$ on the j-side. To see that the thinning does not affect messages passed to the edges on the i-side, we show that $\mathcal{M}'_{ik} = \mathcal{M}_{ik}$ for all neighbors of i where $k \neq j$. Let t be the neighbors of i where $t \neq j, k$, then $X \notin \mathcal{M}_{ti}$ since $X \notin \mathbf{S}_{ti}$ and

$$
\begin{aligned}
\mathcal{M}_{ik} &= \overset{=}{\sum}_{\mathbf{S}_{ik}} \mathcal{M}_{ji} \cdot \prod_{t \neq k, t \neq j} \mathcal{M}_{ti} \\
&\overset{X \notin \mathbf{S}_{ik}}{=} \overset{=}{\sum}_{\mathbf{S}_{ik}} \sum_X \mathcal{M}_{ji} \cdot \prod_{t \neq k, t \neq j} \mathcal{M}_{ti} \\
&= \overset{=}{\sum}_{\mathbf{S}_{ik}} (\sum_X \mathcal{M}_{ji}) \cdot \prod_{t \neq k, t \neq j} \mathcal{M}_{ti} \\
&= \overset{=}{\sum}_{\mathbf{S}_{ik}} \mathcal{M}'_{ji} \cdot \prod_{t \neq k, t \neq j} \mathcal{M}_{ti} = \mathcal{M}'_{ik}
\end{aligned}
$$

where t are the neighbors of i.

—Similar arguments can be applied to the case of Rule (c).

Completeness. We will next show that Definition 8 cannot be improved on under the following two assumptions, which predicate the reverse directions of Corollary 1 and Theorem 4. That is, the assumptions basically mean that one cannot thin separators further unless one invokes results that go beyond Corollary 1 and Theorem 4.

Assumption 1. *Let $\mathcal{G}$ and $\mathcal{H}$ be two sets of factors that contain functional variable X. If $f_X \notin \mathcal{G}$ or $f_X \notin \mathcal{H}$, then $\sum_X \mathcal{G} \cdot \mathcal{H} \neq (\sum_X \mathcal{G})(\sum_X \mathcal{H})$.*

Assumption 2. *Let f_X be a mechanism for X. If $f_X \notin \mathcal{G}$ and $f_X \notin \mathcal{H}$, then $f_X \notin \mathcal{G} \cdot \mathcal{H}$.*

The soundness proof showed that Theorem 5 holds for thinned separators $\mathbf{S}^\star$ (which can be obtained using thinning rules by Theorem 9). We will use $\mathcal{M}^\star_{ij}$ and $\mathcal{M}^\star_{ji}$ to denote messages for edge (i, j) under thinned separators $\mathbf{S}^\star$. We will now show that $Pr(\mathbf{S}'_{ij}) \neq \mathcal{M}'_{ij}\mathcal{M}'_{ji}$, where $\mathbf{S}'_{ij} = \mathbf{S}^\star_{ij} \setminus \{X\}$, $\mathcal{M}'_{ij} = \sum_X \mathcal{M}^\star_{ij}$, $\mathcal{M}'_{ji} = \sum_X \mathcal{M}^\star_{ji}$ and X is a functional variable.

By condition (a) of Definition 8, node i is not $(X, \mathbf{S}^\star)$-connected to any f_X on the i-side or node j is not $(X, \mathbf{S}^\star)$-connected to any f_X on the j-side. By conditions (b,c) of Definition 8, $X \in \mathcal{M}^\star_{ij}$ and $X \in \mathcal{M}^\star_{ji}$. W.l.o.g., suppose node i is not $(X, \mathbf{S}^\star)$-connected to any f_X on the i-side. We next show that $f_X \notin \mathcal{M}^\star_{ij}$. For each leaf node k that hosts f_X on the i-side, consider the path $p_0 = k, p_1, \cdots, p_n = i$ from k to i. Since i is not $(X, \mathbf{S}^\star)$-connected to k, $X \notin \mathbf{S}^\star_{p_t,p_{t+1}}$ for some edge (p_t, p_{t+1}) on the path, which implies that $f_X \notin \mathcal{M}_{p_t,p_{t+1}}$. A similar argument applies to other f_X's on the i-side. By Assumption 2, it follows that $f_X \notin \mathcal{M}^\star_{ij}$. By Assumption 1, we have

$$
\begin{aligned}
&\sum_X Pr(\mathbf{S}^\star_{ij}) = \sum_X \mathcal{M}^\star_{ij} \cdot \mathcal{M}^\star_{ji} \\
&\neq (\sum_X \mathcal{M}^\star_{ij})(\sum_X \mathcal{M}^\star_{ji}) = \mathcal{M}'_{ij} \cdot \mathcal{M}'_{ji}
\end{aligned}
$$

A.7 PROOF OF THEOREM 11

Proof. Suppose replication $\mathcal{F}_1$ has width w. Then it must have a causal jointree $\mathcal{T}_1$ of width w. We can turn $\mathcal{T}_1$ into a causal jointree $\mathcal{T}_2$ for replication $\mathcal{F}_2$ by assigning more factors to leaf nodes in $\mathcal{T}_1$. In particular, for each replica $f_X \in \mathcal{F}_2 \setminus \mathcal{F}_1$, assign this replica to a leaf node in $\mathcal{T}_1$ which hosts a mechanism f_X. This guarantees that $\mathcal{T}_2$ will also have width w. Since replication $\mathcal{F}_2$ has a causal jointree of width w, its width must be $\leq w$. □

A.8 PROOF OF THEOREM 12

Proof. The fact that Algorithm 1 computes a complete replication follows directly from the statement of the algorithm. Suppose there exists another complete replication $\mathcal{F}'$ that is different from the complete replication $\mathcal{F}$ computed by Algorithm 1. Then $\mathcal{F}'$ and $\mathcal{F}$ must differ on the number of mechanisms f_X for some functional variable X. Suppose X is the first variable visited by Algorithm 1 on which this disagreement takes place. Then $\mathcal{F}$ and $\mathcal{F}'$ must have the same number of X-feedings factors; otherwise, they will have a different number of mechanisms for some descendant of variable X. Since $\mathcal{F}$ and $\mathcal{F}'$ both satisfy Definition 13 and they have the same number of X-feeding factors, they must have the same number of mechanisms f_X. This contradicts the assumption that $\mathcal{F}'$ and $\mathcal{F}$ differ on the number of mechanisms for variable X. Hence, $\mathcal{F} = \mathcal{F}'$. □

A.9 PROOF OF THEOREM 13

Lemma 6. *Consider a replication $\mathcal{F}$ that contains $n > 1$ mechanisms for variable X and m X-feeding factors where $n > m$. Let $\mathcal{F}'$ be the result of removing one mechanism for X from $\mathcal{F}$ (hence, $|\mathcal{F}'| = |\mathcal{F}| - 1$). The width of $\mathcal{F}'$ is no greater than the width of $\mathcal{F}$.*

Proof. Let $\mathcal{T}$ be a causal jointree for $\mathcal{F}$. We can turn $\mathcal{T}$ into a causal jointree $\mathcal{T}'$ for $\mathcal{F}'$ with no greater width as follows. Suppose there exists a leaf node in $\mathcal{T}$ that hosts two mechanisms f_X. We can then remove one of these mechanisms from the leaf without increasing the width. Suppose now that each leaf node in $\mathcal{T}$ hosts at most one mechanism f_X. The edges of $\mathcal{T}$ which contain variable X in their (thinned) separators form a set of connected subtrees. Each one of these subtrees will contain at most one mechanism f_X (otherwise thinning Rule (a) will apply). By the pigeonhole principle, at least one of these subtrees must contain a mechanism f_X but no X-feeding factors. We can show that removing this mechanism f_X from the subtree would not increase the causal width of resulting tree. Hence, if $\mathcal{F}$ has a causal jointree of width w, then $\mathcal{F}'$ has a causal jointree of no greater width. This implies that the width of $\mathcal{F}'$ is no greater than the width of $\mathcal{F}$. □

Proof of Theorem 13. We first construct a new replication $\mathcal{G} = \mathcal{F} \cup \mathcal{F}'$. By Theorem 11, the width of $\mathcal{G}$ is no greater than the width of $\mathcal{F}'$. We then go through the functional variables in reverse-topological order (as visited by Algorithm 1). For each variable X, we compare if the X-mechanisms in $\mathcal{F}$ and $\mathcal{G}$ are equal. If so, we proceed to the next functional variable in the order. Otherwise, $\mathcal{G}$ contains more mechanisms than $\mathcal{F}$ and thus there are more X-mechanisms than X-feeding factors in $\mathcal{G}$. By Lemma 6, we are licensed to remove the excess X-mechanisms from $\mathcal{G}$ without increasing its width. By the end of this process, $\mathcal{G}$ will become equal to complete replication $\mathcal{F}$. Hence, the width of $\mathcal{F}$ is no greater than the width of $\mathcal{F}'$. □

A.10 PROOF OF THEOREM 14

Proof. In a complete replication, the number of mechanisms for a functional variable is upper-bounded by the total number of mechanisms of its children. Let X be a functional variable and $f(X)$ be the number of mechanisms in the complete replication, then $f(X) = \sum_{C_i} f(C_i) \leq c \cdot max_{C_i} f(C_i)$ where C_i are the children of X in the DAG. Hence, we can recursively bound the number of mechanisms for all the functional nodes in a functional chain. Since the longest functional chain has a length of k, we can recursively apply the above bound for at most k steps. When the recursion terminates, $max_{C_i} f(C_i) = 1$ since all $f(C_i)$ will be non-functional variables. Therefore, $f(X) \leq c^k$ for each functional variable X in the replication. The inequality also holds for non-functional variables, $f(X) = 1 = c^0 \leq c^k$. Given a total of n nodes in the DAG, we have at most $O(nc^k)$ factors in the complete replication. □