Proceedings of Machine Learning Research vol TBD:1–26, 2023 2nd Conference on Causal Learning and Reasoning

Causal Inference Despite Limited Global Confounding via Mixture Models

Spencer Gordon SLGORDON@CALTECH.EDU
California Institute of Technology
Bijan Mazaheri BMAZAHER@CALTECH.EDU
California Institute of Technology
Yuval Rabani YRABANI@CS.HUJI.AC.IL
Hebrew University in Jerusalem
Leonard Schulman SCHULMAN@CALTECH.EDU
California Institute of Technology

Editors: Mihaela van der Schaar, Dominik Janzing and Cheng Zhang

Abstract

A Bayesian Network is a directed acyclic graph (DAG) on a set of n random variables (the vertices); a Bayesian Network Distribution (BND) is a probability distribution on the random variables that is Markovian on the graph. A finite k-mixture of such models is graphically represented by a larger graph which has an additional "hidden" (or "latent") random variable U, ranging in $\{1, \ldots, k\}$, and a directed edge from U to every other vertex. Models of this type are fundamental to causal inference, where U models an unobserved confounding effect of multiple populations, obscuring the causal relationships in the observable DAG. By solving the mixture problem and recovering the joint probability distribution on U, traditionally unidentifiable causal relationships become identifiable. Using a reduction to the more well-studied "product" case on empty graphs, we give the first algorithm to learn mixtures of non-empty DAGs.

Keywords: mixture models, Bayesian networks, causal DAGs, hidden confounder, population confounder, clobal confounding, causal identifiability

1. Introduction

A Bayesian network is a directed acyclic graph $\mathcal{G} = (\boldsymbol{V}, \boldsymbol{E})$, on a set of $|\boldsymbol{V}| = n$ random variables. A corresponding Bayesian network distribution (BND) is a probability distribution on the random variables that is Markovian on the graph. That is to say, the joint distribution on the variables can be factored as $\prod_{i=1}^{n} \mathcal{P}[V_i = v_i \mid \mathbf{pa}(V_i)]$ where $\mathbf{pa}(V_i)$ is the assignment to the parents of V_i. A k-MixBND on $\mathcal{G}$ is a convex combination, or "mixture", of k BNDs. We represent this situation graphically with a single unobservable random variable U with edges to each of the variable $V \in G$. Here, U is referred to as a "confounding" variable with range $1, \ldots, k$ and the variables in $\mathcal{G}$ are referred to as the "observables." The main complexity parameter of the problem is k, representing the number of mixture constituents or "sources."

One extremely special case has been of longstanding interest in the theory literature: where $\mathcal{G}$ is empty. Such a distribution is a mixture of k product distributions or k-MixProd. See Fig. 1.

In this paper we study the *identification* problem for k-MixBNDs. Specifically, given the graph $\mathcal{G}$, and given a joint distribution $\mathcal{P}$ on the variables (vertices), recover up to small statistical error (a) the mixture weights, up to a permutation of the constituents, and (b) for every mixture constituent and for every vertex V, its conditional distribution given each possible setting to its parents. The primary algorithm will be developed by reducing the k-MixBND problem into a series of calls to a k-MixProd oracle. k-MixBND models are not always identifiable, as further discussed in *Assumptions* below. Thus, another contribution of our paper is to establish a sufficient setting to guarantee identifiability.

© 2023 S. Gordon, B. Mazaheri, Y. Rabani & L. Schulman.

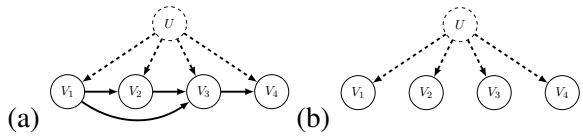

Figure 1: (a) A small Bayesian Network, with latent variable U. (b) In the empty graph, a k-MixBND is a k-MixProd (mixture of product distributions).

Assumptions The following assumptions are used throughout this paper.

1. *We have access to a k-MixProd oracle requiring $O(k)$ variables that are independent within each source. We will refer to this quantity as N_{mp}.* The most efficient published algorithm for this problem is given in Gordon et al. (2021), which requires $N_{\text{mp}} = 3k - 3$ variables and time complexity $\exp(k^2)$. Recent unpublished work improves the complexity bound to $\exp(k \log k)$.
2. *The observable variables in our BND are binary and discrete.* While a number of papers have focused on continuous or large-alphabet settings, we restrict our focus to the simplest setting of binary, discrete variables. Appendix C.4 gives a reduction from alphabets of any size d to the binary case, incurring a mild cost in complexity.
3. *The mixture is supported on $\leq k$ sources.* If the hidden variable U has unrestricted range (Specifically, range $k = 2^n$ would be enough), the BND can be *any* probability distribution, making identification impossible. The question is therefore one of trading k against the sample and computational complexity of an algorithm (and the degree of the network).
4. *The underlying Bayesian DAG is sufficiently sparse.* In order to reduce k-MixBND to k-MixProd we need sufficiently many variables that can be separated from each other by conditioning on disjoint *Markov boundaries* (example in Fig. 2, definition in Sec. 1.3). As a result, the complexity of the algorithm is exponential in the size of a Markov boundary. Both for complexity and in order to keep n small, a bound on the maximum degree Δ is required. The algorithm works if $n \geq (\Delta + 1)^4 N_{\text{mp}}$.[1]

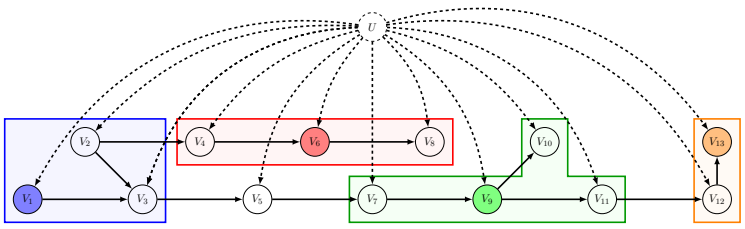

Figure 2: A Bayesian network. Four vertices V_1, V_6, V_9, V_{13} with their corresponding disjoint Markov boundaries are indicated.

5. *The resulting product mixtures are non-degenerate.* Even in mixtures of graphs with sparse structure (in particular the empty graph—the k-MixProd problem), the k-MixBND can be unidentifiable if the mixture components are insufficiently distinct. (E.g., trivially, a mixture of identical sources generates the same statistics as a single source.) Past work has used conditions such as ζ-separation Gordon et al. (2021). These ensure that matrices representing the parameters for each

1. If the skeleton of $\mathcal{G}$ happens to be a path, then we only need a milder condition that $n \geq 2N_{\text{mp}}$. For details see Appendix C.3.

source are well-conditioned. These are not always necessary conditions; characterizing necessary conditions is a difficult question tackled in part in Gordon and Schulman (2022).

6. *The DAG structure representing conditional independence properties within each source, or a common supergraph of these structures, is known.* The primary purpose of this assumption is to focus our paper on the causal identification problem itself. It is often the case that domain knowledge provides an understanding of the causal DAG. If the causal DAG is unknown, we are faced with a different problem, commonly known as "Causal Discovery;" this is an entirely distinct, and extensively developed area (see Glymour et al. (2019a) for a recent survey.) Causal discovery in the presence of a universal confounder has been suggested in Anandkumar et al. (2012a) by substituting independence tests with rank tests. We should stress that our algorithm does not depend on knowing the exact graph but can operate on any supergraph of it, which is why our algorithm works even if the different components of the k-MixBND use slightly different causal graphs, and why some uncertainty in knowledge of the graph can be tolerated.

1.1. Background

The primary application of Bayesian networks (a term coined in Pearl (1985)) is to model complex causal relations, with concrete applications in mining data on diverse populations, multi-purpose usage and bioinformatics, amoung others. In a wide range of complex natural and artificial systems causal relations are probabilistic, and moreover inquiry using controlled experiments is impossible or prohibitively expensive. In such settings researchers often have to resort to collecting samples of an assortment of measurements from a joint distribution governed by causal relations.

In causal identification, we are given observable and unobservable variables with a corresponding Bayesian network structure Pearl (2009). The *graphical*-conditions under which causal relationships are identifiable are well studied Shpitser and Pearl (2006); Huang and Valtorta (2008); Pearl (2009); Spirtes et al. (2000a); Peters et al. (2017); Koller and Friedman (2009). There has been little exploration into expanding the class of identifiable causal relationships using *numerical* conditions.

The mixture setting addressed by our paper corresponds to modeling an unobserved multiplicity of populations. Examples include private attributes like health status or artificially constructed classes such as individual price sensitivity. Such a setting satisfies neither observability of the "backdoor adjustment set" nor the sparsity requirements on the connection of U for the "frontdoor criterion" (Pearl, 2009). Hence, the causal relationships within $\mathcal{G}$ are considered unidentifiable in the traditional framework. However, mild additional assumptions in conjunction with a known DAG structure allow identification of the joint probability distribution with the previously unobservable U, a sufficient condition for applying backdoor adjustment.

There are two problems within mixture models: (1) *Learning* the model, namely, producing any model consistent with (or close to) the observations; (2) *Identifying* the model, namely, producing the true model (or one close to it) up to permutations in the source label. These are incomparable problems, as the second goal is stronger but not always possible. Our paper is concerned with identification, because the underlying motivation is to be able to predict the effect of interventions in a system (represented by the graph). Such predictions will not be valid when the model is not identifiable.

The idea of using mixture models to identify parameters in latent variables dates back to E. S. Allman (2009), who use algebraic methods from Kruskal (1976, 1977) to exploit within-source independence. Anandkumar et al. (2012b) follows a similar strategy using tensor decomposition. Both

of these works rely on within-source independence between three variables with support on large alphabets to achieve identifiablility. Specifically, the alphabet size of the independent variables must scale linearly with k.

A different line of work in the theory community involves scaling the number of variables n with k while leaving the alphabet size of the visible variables constant. This problem, referred to as k-MixProd, and has been studied for nearly 30 years Kearns et al. (1994); Cryan et al. (2001); Freund and Mansour (1999); Feldman et al. (2008); Chaudhuri and Rao (2008); Tahmasebi et al. (2018); Chen and Moitra (2019); Gordon et al. (2021). [2] In Feldman et al. (2008) a seminal algorithm for k-MixProd was given with running time $n^{O(k^3)}$ for mixtures on n binary variables (n sufficiently large). This was improved in Chen and Moitra (2019) to $k^{O(k^3)}n^{O(k^2)}$. The most recent algorithm Gordon et al. (2021) identifies a mixture of k product distributions on at least $3k-3$ variables in time $2^{O(k^2)}n^{O(k)}$, under a mild "separation" condition that excludes unidentifiable instances (see below). Under somewhat stricter separation, the time complexity improves to $2^{O(k^2)}n$.

In principle, combining binary variables can create the larger-alphabet variables, though this strategy reduces the sparsity of the new graph. In this sense the same complexity bottlenecks appear in both strategies. We continue the k-MixProd approach, but this paper is relevant to any work using mixture models to learn unobserved variables..

To our knowledge, the only other attempt at detailing a multiple-run reduction to k-MixProd is Anandkumar et al. (2012a), which gives an algorithm for mixtures of Markov random fields–i.e., undirected graphical models. While this setting is fundamentally different from ours, both papers make use of boundary conditioning to induce independence –even though "Markov boundaries" in DAGs are different from "separating sets" in undirected graphs. Additional complications arise for directed graphs because the outputs of the k-MixProd subroutine are conditioned on their Markov boundaries while the desired parameters are only conditioned on their *parents*. In addition, we note that Anandkumar et al. (2012a) only guarantees identification of second order marginal probabilities, which is insufficient for causal identification. [3]

Other related work Kivva et al. (2021) contains as a special case a reduction to the k-MixProd problem. Their goal is to learn a causal graphical model with latent variables, but with a very different structure on the visible and latent variables. They allow for a DAG of latent variables with visible children (which is learned as part of their algorithm); on the other hand, they require that there be no causal relations between visible variables. In our work, the structure on the latent variables is trivial (since there is a single latent variable), but the structure on the visible variables is arbitrary. Characterizing identifiability in the generalization of both these settings in which we allow structure on both the visible and latent portion of the graph is a nice problem beyond the scope of this paper.

Another paper with kindred motivation to ours is Kumar and Sinha (2021), which studies inference of a certain kind of MixBND, in which the structure of the Bayesian network is known, but the data collected is a mixture over some m unknown interventional distributions. The authors give sufficient conditions for identifiability of the network and of the intervention distributions. At a technical level, the papers are not closely related. k is not a parameter in their work, and instead what is essential is an "exclusion" assumption which says that each variable has some value to which it is not assigned by any of the interventions.

2. We do not even try to list the extensive analogous literature for parametrized distributions over $\mathbb{R}$
3. We also mention that Anandkumar et al. (2010) and Anandkumar et al. (2012a) introduce the idea of a sparse local separator; if this can be adapted to the directed-graph case one might be able to somewhat relax assumption 4. We do not attempt this in this paper.

Some other loosely related work includes learning hidden Markov models Hsu et al. (2012); Anandkumar et al. (2012b); Sharan et al. (2017), an incomparable line of work to our question, but with somewhat similar motivation. In the same vein, some papers study learning mixtures of Markov chains from observations of random paths through the state space Batu et al. (2004); Gupta et al. (2016). These models, too, differ substantially from the models addressed in this paper, and pose very different challenges. Literature on causal structure learning Spirtes et al. (2000b); Glymour et al. (2019b) answers the question of identifying the *presence* of hidden confounders. Fast Causal Inference (FCI) harnesses observed conditional independence to learn causal structure, which can detect the presence of unobserved variables when the known variables are insufficient to explain the observed behavior. This literature includes the MDAG problem in which the DAG structure may depend upon the hidden variable; see Thiesson et al. (1998) for heuristic approaches to this problem. Other related works study causal inference in the presence of visible "proxy" variables which are influenced by a latent confounder Miao et al. (2018); Kuroki and Pearl (2014). This has more recently given rise to multiple causal inference, which assumes multiple causes which are independent when conditioned on a confounder Heckerman (2018); Ranganath and Perotte (2018); Wang and Blei (2019). These settings are given with respect to a known causal inference task and significantly less general assumptions on the behavior of the proxy variables. Finite mixture models have been the focus of intense research for well over a century, since pioneering work in the late 1800s Newcomb (1886); Pearson (1894), and doing justice to the vast literature that emanated from this endeavor is impossible within the scope of this paper. See, e.g., the surveys Everitt and Hand (1981); Titterington et al. (1985); Lindsay (1995); McLachlan et al. (2019).

1.2. Summary of contributions

Theorem 1 *Our algorithm identifies a k-MixBND distribution with on a graph of maximum degree Δ and of size $n \geq \Omega(k\Delta^4)$, using $O(n2^{\Delta^2})$ calls to an oracle for the k-MixProd problem.*

(For an exact statement see Theorem 17.) The first key insight involves conditioning on the Markov boundaries of a linear number of variables via post-selection. This induces *within-source* independence which can be harnessed by running a k-MixProd oracle as a subroutine. Gordon et al. (2021), the best published algorithm for this subroutine requires $3k - 3$ conditionally independent variables. (Or $2k - 1$ for a weaker complexity bound.)

We execute a set of $O(n2^{\Delta^2})$ "runs" of a k-MixProd oracle on sets of these variables for all the possible restrictions (assignments of values) to their Markov boundaries. The first challenge is to align the outcomes of these runs, a procedure which we call "alignment." Anandkumar et al. (2012a) developed a primitive version of our "alignment" algorithm for a different but related problem: mixtures of undirected Markov random fields. While they require a single variable that is independent from the rest of the structure, our algorithm develops the notion of "good collections of runs" to eliminate this restriction – a contribution which may have implications in the Markov random field setting as well.

Our final challenge is to propagate the conditional probabilities through the entire network and recover the model – a procedure we call "Bayesian unzipping." Our key contributions can be summarized by these "alignment" and "unzipping" procedures, as well as the notion of a "good collection of runs."

Organization The rest of the paper is organized as follows. In Section 1.3 we outline some Bayesian network notation. In Section 2 we formally develop the notion of a "run," which calls a

k-MixProd oracle. In Section 3 we explain how the output of the "runs" is synthesized to get the desired mixture parameters. This section details the processes of alignment and Bayesian unzipping. Section 4 explains what is necessary in a group of runs in order for the algorithm to succeed, which provides a framework for defining algorithms in terms of sets of runs.

In Appendix A we define the k-MixBND algorithm in pseudo-code. In Appendix B we analyze the k-MixBND algorithm. In Appendix C we prove the existence of a good collection of runs. In Appendix C.4 we generalize our algorithms to handle non-binary observations.

1.3. Notation

A Bayesian network consists of a directed acyclic graph (DAG) $\mathcal{G} = (\boldsymbol{V}, \boldsymbol{E})$ and a probability distribution $\mathcal{P}$ over $\boldsymbol{V} = \{V_1, \ldots, V_n\}$ factoring according to $\mathcal{G}$, i.e.,

$$\mathcal{P}(V_1, V_2, \ldots, V_n) = \prod_{i=1}^{n} \mathcal{P}(V_i \mid \mathbf{PA}(V_i)),$$

where $\mathbf{PA}(V)$ is the set of parents of $V \in \boldsymbol{V}$. (Similarly, let $\mathbf{CH}(V)$ denote the children of V.) Equivalently, $\mathcal{P}$ must satisfy that for any $V \in \boldsymbol{V}$, V is independent of its non-descendants, given its parents.

Generally, we will use bold letters to denote sets of variables. For a matrix $\boldsymbol{M}$, we will use M_{ij} to identify entries. We use $\boldsymbol{M}_{i,-}$ denote the vector $(M_{i,1}, M_{i,2}, \ldots M_{i,k})$ and $M_{i,\sigma}$ to denote its permutation $(M_{i,\sigma(1)}, M_{i,\sigma(2)}, \ldots M_{i,\sigma(k)})$.

Conditioning on the "Markov boundary" of a vertex $\mathbf{MB}(V)$ makes V conditionally independent from everything else in the graph (Pearl, 2014).

Definition 2 (Markov Boundary) *For a vertex Y in a DAG $\mathcal{G} = (\boldsymbol{V}, \boldsymbol{E})$, the **Markov boundary** of Y, denoted $\boldsymbol{MB}(Y)$, is defined by*

$$\boldsymbol{MB}(Y) \coloneqq \boldsymbol{PA}(Y) \cup \boldsymbol{CH}(Y) \cup \boldsymbol{PA}(\boldsymbol{CH}(Y)) \setminus \{Y\}.$$

Lemma 3 (See Pearl (2014)) *For any vertex $V \in \boldsymbol{V}$ and subset $S \subseteq V \setminus (\boldsymbol{MB}(V) \cup \{V\})$, $\mathcal{P}(V \mid \boldsymbol{MB}(V), S) = \mathcal{P}(V \mid \boldsymbol{MB}(V))$.*

Observation 4 *For any $X, Y \in \boldsymbol{V}$, $X \in \boldsymbol{MB}(Y) \iff Y \in \boldsymbol{MB}(X)$*

Uppercase/lowercase conventions Following notation in causal inference literature, we will use lowercase letters to denote assignments. For example $\mathcal{P}(v \mid u) = \mathcal{P}(V = v \mid U = u)$. Following this convention, we will write $\mathbf{pa}(V)$, $\mathbf{ch}(V)$, and $\mathbf{mb}(X)$ to denote assignment to the parents $\mathbf{PA}(V)$, children $\mathbf{CH}(V)$, and Markov boundary $\mathbf{MB}(V)$.

Within-source probabilities It will be easier to write $\mathcal{P}_u(v) = \mathcal{P}(v \mid u)$ to give the probability distribution within a source.

Finally, here are a few more definitions that will make the upcoming sections simpler.

Definition 5 (Top) *We will use $\boldsymbol{TOP}(V)$ to denote $\boldsymbol{MB}(V) \setminus \boldsymbol{CH}(V)$.*

Definition 6 (Depth of a vertex) *Given a DAG $\mathcal{G} = (\boldsymbol{V}, \boldsymbol{E})$ and any vertex $V \in \boldsymbol{V}$, let $d_{\mathcal{G}}(V)$ be the **depth** of V in $\mathcal{G}$, i.e. the length of the shortest path from a degree-0 vertex in $\mathcal{G}$. When $\mathcal{G}$ is clear from context, we'll omit the subscript.*

Definition 7 *We'll introduce a parameter $\gamma(G)$ which will appear in the complexity of the identification procedure, which is defined by $\gamma(G) \coloneqq \max_{V \in \boldsymbol{V}} |\boldsymbol{MB}(V)|$.*

2. Applying a k-MixProd run

Modern k-MixProd algorithms require $\mathcal{O}(k)$ independent variables (when conditioned on the source U), which our algorithm will induce through post-selected conditioning. We will then run a k-MixProd oracle on this post-selected subset of the data. A significant portion of this paper will be accounting for multiple calls (or "runs") of this k-MixProd oracle and explaining how their results can be synthesized.

2.1. Describing runs

We will need to keep track of two crucial elements of each "run" of a k-MixProd oracle.

1. Which variables $\in \boldsymbol{V}$ are passed to our k-MixProd oracle as independent variables (the **independent set**).
2. Which variables $\in \boldsymbol{V}$ we have conditioned on (the **conditioning set**) and what values we have post-selected these variables to take.

A sufficient conditioning set to induce within-source independence among the independent set is the union of their Markov boundaries. This will be further refined in Subsection 4.1.

Definition 8 (Run) *A **run** over a graph $\mathcal{G} = (\boldsymbol{V}, \boldsymbol{E})$ is a tuple $a = (\boldsymbol{I}^a, f^a)$ where $\boldsymbol{I}^a \subseteq \boldsymbol{V}$ are variables that we will d-separate (within each source) by conditioning on assignments to the set*

$$\boldsymbol{COND}^a := \bigcup_{I \in \boldsymbol{I}^a} \boldsymbol{MB}(I)$$

The value of the assignment is given by $f^a : \boldsymbol{COND}^a \to \{0, 1\}$. We will restrict our attention to well-formed runs*, i.e. runs for which*

$$\boldsymbol{I}^a \cap \boldsymbol{COND}^a = \emptyset.$$

*We'll call $\boldsymbol{I}^a$ the **independent set** for a, and $\boldsymbol{COND}^a$ the **conditioning set**.*

Definition 9 *An individual run $a = (\boldsymbol{I}^a, f^a)$ is N-**independent** if $|\boldsymbol{I}^a| \geq N$.*

Superscript notation We'll write $\mathbf{mb}^a(V)$, $\mathbf{pa}^a(V)$, $\mathbf{ch}^a(V)$ to refer to the assignment to the Markov boundary of V, parents of V, and children of V as set by run a.[4] In a similar spirit, we'll occasionally write v^0 to denote the assignment $V = 0$.

Definition 10 (Distribution induced by a run) *For any well-formed run a, the induced distribution on the variables in $\boldsymbol{I}^a$ is denoted by*

$$\mathcal{P}^a(\cdot) = \mathcal{P}(\cdot \mid \boldsymbol{cond}^a),$$

where $\boldsymbol{cond}^a$ is the assignment to $\boldsymbol{COND}^a$ in keeping with our conventions.

The outputs of applying a k-MixProd oracle to $\mathcal{P}^a(\boldsymbol{I}^a)$ are a matrix $\boldsymbol{M}^a \in [0,1]^{|\boldsymbol{I}^a| \times k}$ and a vector of mixture weights, $\pi^a \in [0,1]^k$ (satisfying $\sum_u \pi^a_u = 1$) given by

$$M^a_{i,u} := \mathcal{P}^a(X^a_i = 1 \mid U_a = u) = \mathcal{P}(X_i = 1 \mid U^a = u, \mathbf{COND}^a), \qquad \forall X_i \in \boldsymbol{I}^a, u \in [k] \quad (1)$$

$$\pi^a_u := \mathcal{P}^a(U^a = u) = \mathcal{P}(U_a = u \mid \mathbf{COND}^a), \qquad \forall j \in [k] \quad (2)$$

4. Any quantities parameterized by a run will take the parameter as a superscript.

where U^a is the mixture source distributed over $[k]$ according to $\mathcal{P}(U \mid \mathbf{COND}^a)$. Note that because a mixture is invariant to a permutation of the labels of mixture components, we cannot guarantee correspondence between the labels for the source variables from different runs. Hence the labels of U^a to an unknown permutation of the labels in U. Alignment of these labels is handled in Section 3.1.

3. Combining Runs

A single run of the k-MixProd oracle will not contain sufficient information to learn the parameters of the k-MixBND problem. Instead we must synthesize information across *multiple* runs.

3.1. Aligning source labels across different runs

Each run of the k-MixProd oracle will return $\mathcal{P}^a(V \mid U_a = u)$ for some arbitrary permutation U_a of the variable. We need to align all of the outputs to the same permutation of the source, U. If the runs overlap on at least one variable with the same mixture probabilities, we can use that "alignment variable" to identify which source corresponds to which set of parameters. In our setup, we will guarantee these alignment variable's exist by ensuring that runs have shared vertices in their independent sets whose Markov boundaries are identically conditioned on.

Definition 11 (Separated variable) *$X \in \boldsymbol{V}$ is **separated** if for all $u_i \neq u_j \in [k]$, $\mathcal{P}_{u_i}(x) \neq \mathcal{P}_{u_j}(x)$.*

Definition 12 (Aligned runs) *A pair of runs a, b over independent sets $\boldsymbol{I}^{(a)}, \boldsymbol{I}^{(b)}$ is **alignable** if there exist separated $V^{(ab)} \in \boldsymbol{I}^{(a)} \cap \boldsymbol{I}^{(b)}$ such that $\mathcal{P}^a_u(V^{(ab)}) = \mathcal{P}^b_u(V^{(ab)})$ for all $u \in [k]$. We'll call any such random variable $V^{(ab)}$ an **alignment variable**, and use $\boldsymbol{AV}(a, b)$ to denote the set of all alignment variables for $\mathcal{P}^{(a)}$ and $\mathcal{P}^{(b)}$. We sometimes say a and b are "aligned at" $V^{(ab)}$.*

Definition 13 (Alignment spanning tree) *We say a set of ℓ runs is alignable if there exists an undirected spanning tree over the graph with vertices $a_1, \ldots a_\ell$ and an edge $\{a_i, a_j\}$ whenever $\boldsymbol{AV}(a_i, a_j) \neq \emptyset$. We call this the **alignment spanning tree**.*

The alignment step will take the output from alignable runs and permute the mixture labels until the parameters match along each alignment variable. Pseudocode for this procedure is given in Algorithm **??**.

3.2. Bayesian unzipping: recovering parameters per source

Recall that our algorithm conditions on Markov boundaries to induce independent variables. Hence, after aligning the sources in runs of the k-MixProd oracle we will have access to $\mathcal{P}_u(Y \mid \mathbf{MB}(Y))$ for each $Y \in V$. Our goal is to obtain $\mathcal{P}_u(Y \mid \mathbf{PA}(Y))$. Note that

$$\mathcal{P}_u(y^1 \mid \mathbf{mb}(Y)) = \frac{\mathcal{P}_u(y^1, \mathbf{mb}(Y))}{\mathcal{P}_u(y^1, \mathbf{mb}(Y)) + \mathcal{P}_u(y^0, \mathbf{mb}(Y))}. \tag{3}$$

The terms in this fraction are all of the same form and can be factored according to the DAG into

$$\mathcal{P}_u(y, \mathbf{mb}^a(Y)) = \mathcal{P}_u(\mathbf{top}(Y))\mathcal{P}_u(y \mid \mathbf{pa}^a(Y)) \underbrace{\prod_{V \in \mathbf{CH}(Y)} \mathcal{P}_u(v^a \mid f^a(\mathbf{PA}(V) \setminus \{Y\}), y)}_{\mathcal{P}_u(\mathbf{ch}^a(Y) \mid \mathbf{top}^a(Y), y)}.$$

See Figure 3 for a concrete example of this decomposition. After substituting this factorization into Equation (3) we see that $\mathcal{P}_u(\textbf{top}(Y))$ appears in both the numerator and denominator because it is independent of the assignment to Y. Simplification leaves only the following terms:

1. $\mathcal{P}_u(y^0 \mid \textbf{pa}^a(Y))$ and $\mathcal{P}_u(y^1 \mid \textbf{pa}^a(Y))$, which must sum to 1.
2. $\mathcal{P}_u(\textbf{ch}^a(Y) \mid \textbf{top}^a(Y), y^0)$ and $\mathcal{P}_u(\textbf{ch}^a(Y) \mid \textbf{top}^a(Y), y^1)$ which are both the product of the desired parameters of variables later in the topological ordering. We can ensure we have access to these terms by solving for the parameters of $V \in \boldsymbol{V}$ in a reverse-topological ordering.[5]

We can substitute $1 - \mathcal{P}_u(y^1 \mid \textbf{pa}^a(Y))$ for $\mathcal{P}_u(y^0 \mid \textbf{pa}^a(Y))$ in the expanded version of Equation (3) to obtain a single equation with only $\mathcal{P}_u(y^1 \mid \textbf{pa}^a(Y))$ as an unknown, which we can then solve. The pseudocode for this process is given in Algorithm 1.

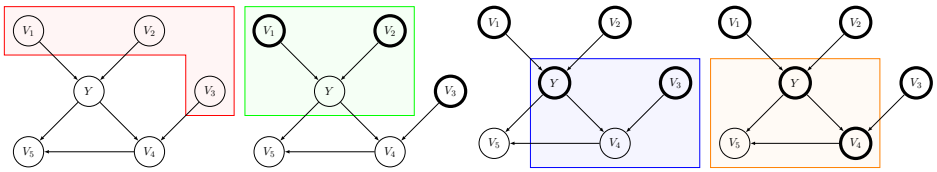

Figure 3: We can decompose $\mathcal{P}_u(v_1, v_2, v_3, y, v_4, v_5) = \mathcal{P}_u(v_1, v_2, v_3)\mathcal{P}_u(y \mid v_1, v_2)\mathcal{P}_u(v_4 \mid y, v_3)\mathcal{P}_u(v_5 \mid y, v_4)$. U and any other variables in the graph are omitted for clarity.

3.2.1. RECOVERING THE DISTRIBUTION ON SOURCES

Now consider some arbitrary run a with conditioning $\textbf{cond}^a$. Since $\mathcal{P}_u(\boldsymbol{V}) = \prod_{V \in \boldsymbol{V}} \mathcal{P}_u(V \mid \textbf{PA}(V))$, knowing $\mathcal{P}_u(V \mid \textbf{PA}(V))$ grants us full access to the within-source probability distribution $\mathcal{P}_u(\boldsymbol{V})$ after Bayesian unzipping. From this we can we obtain $\mathcal{P}_u(\textbf{cond}^a) = \mathcal{P}(\textbf{cond}^a \mid u)$. The k-MixProd oracle will also return $\mathcal{P}^a(U) = \mathcal{P}(U \mid \textbf{cond}^a)$ when run on a (after source alignment). Finally, $\mathcal{P}(\textbf{cond}^a)$ is directly observable. Combining these terms in Bayes' rule lets us compute the distribution on U (under the assumption of positivity),

$$\mathcal{P}(u) = \frac{\mathcal{P}(u \mid \textbf{cond}^a)\mathcal{P}(\textbf{cond}^a)}{\mathcal{P}(\textbf{cond}^a \mid u)}.$$

4. A good collection of runs

With the main concepts of source alignment and Bayesian unzipping now defined, our algorithm will primarily consist of finding a good collection of these runs so that these subroutines can be successfully applied to recover the k-MixBND mixture.

Observation 14 *Two runs a, b are aligned at $X \in \boldsymbol{V}$ if and only if*

1. $X \in \boldsymbol{I}^a \cap \boldsymbol{I}^b$,
2. $\boldsymbol{mb}^a(X) = \boldsymbol{mb}^b(X)$, *i.e,* $f^a(\boldsymbol{MB}(X)) = f^b(\boldsymbol{MB}(X))$, *and*
3. X *is separated given* $\boldsymbol{mb}^a(X)$ *(equivalently, given* $\boldsymbol{mb}^b(X)$*).*

5. We will want to ensure that we only need to unzip parameters from vertices of a bounded depth, which bounds the iterations of this step. Details on how this is done appear in Section 4.1.

Definition 15 *A collection of runs $\mathcal{A}$ **covers** $X \in \boldsymbol{V}$ if for every assignment $\boldsymbol{pa}(X)$ to $\boldsymbol{PA}(X)$ there exists a run $a \in \mathcal{A}$ with $X \in \boldsymbol{I}^a$ and $\boldsymbol{pa}(X) = \boldsymbol{pa}^a(X)$.*

Definition 16 (A good collection of runs) *A collection of well-formed runs $\mathcal{A}$ is* good *if it is (i) alignable, (ii) every run is N_{mp}-independent, and (iii) the collection covers every vertex in $\boldsymbol{V}$.*

The following is our main result on good collections of runs:

Theorem 17 *Given a graph with max degree Δ satisfying $n \geq N_{mp} \times O(\Delta^4)$, we can find a set of centers $\boldsymbol{X} = \{X_1, \ldots, X_{N_{mp}}\} \subseteq \boldsymbol{V}$ of size N_{mp} and depth at most $3N_{mp}$, such that Algorithm 3 succeeds in finding a good collection of runs $\mathcal{A}$ of size $O(2^{\Delta^2} n)$.*

While any good collection of runs will suffice for our algorithm, Theorem 17 represents conditions under which we can provably obtain such a collection of runs. In Appendix C.3, we give a good collection of runs for mixture of paths which is more efficient than the one found by Algorithm 3.

We will sketch the collection of good runs, leaving some details to Appendix C. To ensure alignment is possible, we will construct a set of *central runs*, $\mathcal{A}_C$ which we can align to each other and which all other runs will be alignable to.

Definition 18 (Centers, Central Runs) *A set of vertices $\boldsymbol{X} = \{X_1, \ldots, X_{N_{mp}}\} \subseteq \boldsymbol{V}$ will be called **centers** if the Markov boundaries of the vertices in $\boldsymbol{X}$ are disjoint. Given a set of centers $\boldsymbol{X}$, a run a is called a **central run** if $\boldsymbol{I}^a = \boldsymbol{X}$.*

To build these central runs, we will start with a set of N_{mp} vertices $\boldsymbol{X} = \{X_1, X_2, \ldots X_{N_{\text{mp}}}\}$ with *disjoint* Markov boundaries and a maximum depth of $3N_{\text{mb}}$, whose existence is implied by our degree bounds (see Appendix C). An example of four such vertices was given in Figure 2.

First, we fix a run a_0 with $\boldsymbol{I}_{a_0} = \boldsymbol{X}$ and $\mathbf{mb}^{a_0}(\boldsymbol{X})$ being chosen arbitrarily where $\mathbf{MB}(\boldsymbol{X}) := \cup_{X_i \in \boldsymbol{X}} \mathbf{MB}(X_i)$. We will refer to this assignment $\mathbf{mb}^{a_0}(\boldsymbol{X})$ as the *default* assignment. Each run in $a \in \mathcal{A}_C$ will have the same independent set $\boldsymbol{I}^a = \boldsymbol{X}$ and will agree with a_0 on the assignment to all of the conditioning set other than the Markov boundary of some $X_i \in \boldsymbol{X}$, i.e.

$$f^a(V) = f^{a_0}(V) \;\; \forall V \in \mathbf{MB}(\boldsymbol{X}) \setminus \mathbf{MB}(X_i). \tag{4}$$

The central runs will span over all assignments $\mathbf{mb}(X_i)$ to $\mathbf{MB}(X_i)$ for each $X_i \in \boldsymbol{X}$. We'll write each such run as $a_0[\mathbf{mb}(X_i)]$.

Definition 19 $\mathcal{A}_C := \{a_0\} \cup \left\{a_0[\boldsymbol{mb}(X_i)] : i \in [3k-3], \boldsymbol{mb}(X_i) \in \{0,1\}^{\boldsymbol{MB}(X_i)}\right\}$.

See Figure 4 for an example of a set of central runs and a visualization of how they are alignable.

The central runs provide a backbone for easily guaranteeing alignment. The runs in $\mathcal{A}_Y$ made up of the following two types of perturbations to the independent set (with $\mathbf{COND}^a$ always defined as the union of the Markov boundaries of the independent set, as in Definition 8):

1. For each $Y \in \mathbf{MB}(X_i)$ for some $X_i \in \boldsymbol{X}$, we exclude X_i from the independent set to form $\boldsymbol{I}^{a_Y} = \boldsymbol{X} \cup \{Y\} \setminus \{X_i\}$.[6]
2. For each $Y \notin \mathbf{MB}(\boldsymbol{X}) \cap \boldsymbol{X}$, $\boldsymbol{I}^a = \boldsymbol{X} \cup \{Y\}$.

For either independence set we will form $2^{|\mathbf{PA}(Y)|}$ runs each associated with a single assignment to $\mathbf{pa}^a(Y)$, with the remaining variables in $\mathbf{COND}^a \cap \mathbf{COND}^{a_0}$ conditioned on their defaults given by f^{a_0}. Any other assignments to variables in $\mathbf{COND}^a$ can be chosen arbitrarily. The pseudocode for this construction is in Appendix C, as well as justification for why our degree bounds imply that N_{mp} disjoint Markov boundaries can be found.

6. This is a well-formed run since $Y \in \mathbf{MB}(X_i) \implies X_j \notin \mathbf{MB}(Y)$ for any $j \neq i$ by Observation 4.

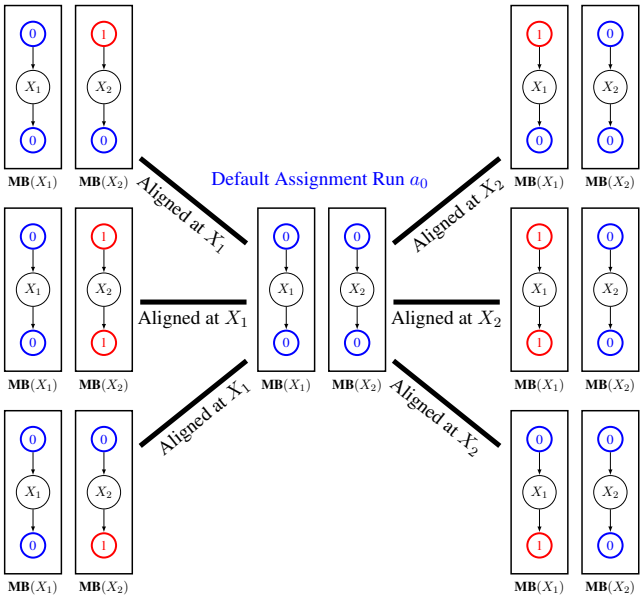

Figure 4: An alignment spanning tree of the default assignment a_0 ($\mathbf{COND}^{a_0}$ arbitrarily assigns all Markov boundaries to 0) and six other central runs. The runs on the left cover all possible assignments to $\mathbf{MB}(X_2) \in \{(0,0),(0,1),(1,0)\}$, while maintaining the default assignment to $\mathbf{MB}(X_1)$ to allow alignment with a_0. The right runs similarly cover all possible assignments to $\mathbf{MB}(X_1)$, aligned at X_2.

4.1. Limiting the depth of unzipping

As currently given, our algorithm may require iteratively Bayesian unzipping parameters up to the depth of the graph. By limiting the depth of the vertices that need to be unzipped, we can ensure that we have an upper bound to the number of consecutive Bayesian unzipping, limiting accumulated errors from the process.

Recall that the goal of the conditioning set of each run is to d-separate each of the vertices in the independent set. For a topological ordering on the independence set, notice that we need not condition on the descendants of the deepest vertices in order to d-separate them from the others. Conveniently, avoiding conditioning on these vertices descendants leaves the output of the k-MixProd oracle in the desired form. We call these deepest vertices "bottom" vertices.

Definition 20 (Bottom vertices in a run) *Given vertices $\boldsymbol{I}$, we'll define the set of bottom vertices of the run to be the subset $\boldsymbol{BOT}(\boldsymbol{I}) \subseteq \boldsymbol{I}$ of vertices with maximal depth among the vertices in $\boldsymbol{I}$. That is $d(B) = \max_{I \in \boldsymbol{I}} d(I)$ for all $B \in \boldsymbol{BOT}(\boldsymbol{I})$.*

We can now update the conditioning sets for our definition of runs:

$$\mathbf{COND}^a := \bigcup_{I \in \boldsymbol{I}^a \setminus \mathbf{BOT}(\boldsymbol{I}^a)} \mathbf{MB}(I) \bigcup_{B \in \mathbf{BOT}(\boldsymbol{I}^a)} \mathbf{PA}(B) \tag{5}$$

We append two additional requirements for a good collection of runs (Definition 16).

- no vertex appears both as a bottom vertex and a non-bottom vertex, and
- every non-bottom vertex has depth at most $3N_{\text{mp}}$.

Note that because we only need independent sets of size N_{mp}, it is trivial to limit the depth of our non-bottom vertices to $3N_{\text{mp}}$.

4.2. Outline of the algorithm

Our algorithm has four steps:

1. Use a k-MixProd oracle on $\mathcal{P}(\boldsymbol{I}^a \mid \mathbf{COND}^a)$ for each run $a \in \mathcal{A}$ to compute $P(Y \mid \mathbf{MB}(Y), U_a = u)$ for all variables $Y \in \boldsymbol{V}$.
2. Align the parameters obtained from the previous step to ensure that U means the same thing across different runs, giving $\mathcal{P}_u(Y \mid \mathbf{MB}(Y))$.
3. Work backwards in topological order to recover $\mathcal{P}_u(Y \mid \mathbf{PA}(Y))$ for each vertex $Y \in \boldsymbol{V}$ via Bayesian unzipping.
4. Compute $\mathcal{P}(U)$ by applying Bayes' law.

The full procedure appears as Algorithm 2.

5. Conclusion

We have developed the first algorithm for identifying the parameters of k-MixBND mixtures. This algorithm allows us to access the probability distributions within each source - equivalent to the probability distribution conditioned on a universal and unobserved confounder. With access to this conditional distribution, the confounding of U can be adjusted for, opening up the opportunity for causal inference despite universal confounding.

The algorithm presented here is not intended for immediate practical use. Instead, we hope this algorithm and framework can serve as a springboard for understanding how solutions to the k-MixProd and k-MixBND problems are intimately related.

The alignment process is highly nontrivial and a likely reason why papers such as Anandkumar et al. (2012a) made crude assumptions (such as a conveniently independent variable) to simplify this process. The formal development of a notion of a run, while tedious, will allow for further improvements to give better "good sets of runs." Graph-specific sets of runs can be optimized further, as demonstrated in Appendix C.3.

References

A. Anandkumar, V. Tan, and A. Willsky. High dimensional structure learning of ising models on sparse random graphs. 2010.

A. Anandkumar, D. Hsu, F. Huang, and S. M. Kakade. Learning high-dimensional mixtures of graphical models. *arXiv preprint arXiv:1203.0697*, 2012a.

A. Anandkumar, D. J. Hsu, and S. M. Kakade. A method of moments for mixture models and hidden Markov models. In *Proc. 25th Ann. Conf. on Learning Theory - COLT*, volume 23 of *JMLR Proceedings*, pages 33.1–33.34, 2012b. URL http://proceedings.mlr.press/v23/anandkumar12/anandkumar12.pdf.

T. Batu, S. Guha, and S. Kannan. Inferring mixtures of Markov chains. In *Proc. 17th Conf. on Learning Theory*, pages 186–199, 2004. doi: 10.1007/978-3-540-27819-1_13.

K. Chaudhuri and S. Rao. Learning mixtures of product distributions using correlations and independence. In *Proc. 21st Ann. Conf. on Learning Theory - COLT*, pages 9–20. Omnipress, 2008. URL http://colt2008.cs.helsinki.fi/papers/7-Chaudhuri.pdf.

S. Chen and A. Moitra. Beyond the low-degree algorithm: mixtures of subcubes and their applications. In *Proc. 51st Ann. ACM Symp. on Theory of Computing*, pages 869–880, 2019. doi: 10.1145/3313276.3316375.

M. Cryan, L. Goldberg, and P. Goldberg. Evolutionary trees can be learned in polynomial time in the two state general Markov model. *SIAM J. Comput.*, 31(2):375–397, 2001. doi: 10.1137/S0097539798342496.

J. A. Rhodes E. S. Allman, C. Matias. Identifiability of parameters in latent structure models with many observed variables. *Ann. Statist.*, 37(6A):3099–3132, 2009. doi: 10.1214/09-AOS689.

B. S. Everitt and D. J. Hand. Mixtures of discrete distributions. In *Finite Mixture Distributions*, pages 89–105. Springer Netherlands, Dordrecht, 1981.

J. Feldman, R. O'Donnell, and R. A. Servedio. Learning mixtures of product distributions over discrete domains. *SIAM J. Comput.*, 37(5):1536–1564, 2008. doi: 10.1137/060670705.

Y. Freund and Y. Mansour. Estimating a mixture of two product distributions. In *Proc. 12th Ann. Conf. on Computational Learning Theory*, pages 53–62, July 1999. doi: 10.1145/307400.307412.

C. Glymour, K. Zhang, and P. Spirtes. Review of causal discovery methods based on graphical models. *Frontiers in Genetics*, 10, 2019a. ISSN 1664-8021. doi: 10.3389/fgene.2019.00524. URL https://www.frontiersin.org/article/10.3389/fgene.2019.00524.

C. Glymour, K. Zhang, and P. Spirtes. Review of causal discovery methods based on graphical models. *Frontiers in Genetics*, 10:524, 2019b. doi: 10.3389/fgene.2019.00524.

S. L. Gordon and L. J. Schulman. Hadamard extensions and the identification of mixtures of product distributions. *IEEE Transactions on Information Theory*, 2022. to appear.

S. L. Gordon, B. Mazaheri, Y. Rabani, and L. J. Schulman. Source identification for mixtures of product distributions. In *Proc. 34th Ann. Conf. on Learning Theory - COLT*, volume 134 of *Proc. Machine Learning Research*, pages 2193–2216. PMLR, 2021. URL http://proceedings.mlr.press/v134/gordon21a.html.

R. Gupta, R. Kumar, and S. Vassilvitskii. On mixtures of Markov chains. In *Advances in Neural Information Processing Systems*, volume 29, 2016. URL https://proceedings.neurips.cc/paper/2016/file/8b5700012be65c9da25f49408d959ca0-Paper.pdf.

D. Heckerman. Accounting for hidden common causes when inferring cause and effect from observational data. (NIPS 2017 causal inference workshop), 2018. URL https://arxiv.org/abs/1801.00727.

D. Hsu, S. M. Kakade, and T. Zhang. A spectral algorithm for learning hidden Markov models. *J. Comput. Syst. Sci.*, 78(5):1460–1480, September 2012. doi: 10.1016/j.jcss.2011.12.025.

Y. Huang and M. Valtorta. On the completeness of an identifiability algorithm for semi-Markovian models. *Ann. Math. Artif. Intell.*, 54(4):363–408, December 2008. doi: /10.1007/s10472-008-9101-x.

M. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. Schapire, and L. Sellie. On the learnability of discrete distributions. In *Proc. 26th Ann. ACM Symp. on Theory of Computing*, pages 273–282, 1994. doi: 10.1145/195058.195155.

Bohdan Kivva, Goutham Rajendran, Pradeep Kumar Ravikumar, and Bryon Aragam. Learning latent causal graphs via mixture oracles. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=f9mSLa07Ncc.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

Joseph B Kruskal. More factors than subjects, tests and treatments: an indeterminacy theorem for canonical decomposition and individual differences scaling. *Psychometrika*, 41(3):281–293, 1976.

Joseph B Kruskal. Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear algebra and its applications*, 18(2):95–138, 1977.

A. Kumar and G. Sinha. Disentangling mixtures of unknown causal interventions. In C. de Campos and M. H. Maathuis, editors, *Proc. Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161 of *Proc. Machine Learning Research*, pages 2093–2102. PMLR, 27–30 Jul 2021. URL https://proceedings.mlr.press/v161/kumar21a.html.

M. Kuroki and J. Pearl. Measurement bias and effect restoration in causal inference. *Biometrika*, 101(2):423–437, 2014. doi: 10.1093/biomet/ast066.

B. G. Lindsay. *Mixture models: theory, geometry and applications*. 1995.

G. J. McLachlan, S. X. Lee, and S. I. Rathnayake. Finite mixture models. *Annual Review of Statistics and Its Application*, 6(1):355–378, 2019. doi: 10.1146/annurev-statistics-031017-100325.

W. Miao, Z. Geng, and E. T. Tchetgen. Identifying causal effects with proxy variables of an unmeasured confounder. *Biometrika*, 105(4):987–993, 2018. doi: 10.1093/biomet/asy038.

S. Newcomb. A generalized theory of the combination of observations so as to obtain the best result. *American Journal of Mathematics*, 8(4):343–366, 1886.

J. Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. Technical Report CSD-850021, R-43, UCLA Computer Science Department, June 1985.

J. Pearl. *Causality*. Cambridge, 2nd edition, 2009.

J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

K. Pearson. Contributions to the mathematical theory of evolution III. *Philosophical Transactions of the Royal Society of London (A.)*, 185:71–110, 1894.

J. Peters, D. Janzing, and B. Schölkopf. *Elements of Causal Inference*. MIT Press, 2017.

R. Ranganath and A. Perotte. Multiple causal inference with latent confounding. 2018. URL https://arxiv.org/abs/1805.08273.

V. Sharan, S. M. Kakade, P. Liang, and G. Valiant. Learning overcomplete HMMs. In *Advances in Neural Information Processing Systems*, pages 940–949, 2017. URL https://arxiv.org/abs/1711.02309.

I. Shpitser and J. Pearl. Identification of joint interventional distributions in recursive semi-Markovian causal models. In *Proc. 20th AAAI Conference on Artifical Intelligence*, pages 1219–1226, 2006. doi: 10.5555/1597348.1597382.

P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction and Search*. MIT Press, second edition, 2000a.

P. Spirtes, C. Glymour, R. Scheines, S. Kauffman, V. Aimale, and F. Wimberly. Constructing Bayesian network models of gene expression networks from microarray data. 2000b.

B. Tahmasebi, S. A. Motahari, and M. A. Maddah-Ali. On the identifiability of finite mixtures of finite product measures. (Also in "On the identifiability of parameters in the population stratification problem: A worst-case analysis," Proc. ISIT'18 pp. 1051-1055), 2018. URL https://arxiv.org/abs/1807.05444.

B. Thiesson, C. Meek, D. M. Chickering, and D. Heckerman. Learning mixtures of DAG models. In *Proc. 14th Conf. on Uncertainty in Artificial Intelligence*, page 504–513, 1998. doi: 10.5555/2074094.2074154.

D. M. Titterington, A. F. M. Smith, and U. E. Makov. *Statistical Analysis of Finite Mixture Distributions*. John Wiley and Sons, Inc., 1985.

Y. Wang and D. M. Blei. The blessings of multiple causes. *Journal of the American Statistical Association*, 114(528):1574–1596, 2019. doi: 10.1080/01621459.2019.1686987.

Appendix A. k-MixBND algorithm pseudocode

Algorithm 1: Bayesian Unzipping

Input: A collection of runs $\mathcal{A}$ of size at most $2^{O(\Delta^2)}$ and their aligned output. For each V_i and assignment to its parents $\mathbf{pa}(V_i)$ there must be some run with V_i in its independent set with parents conditioned to $\mathbf{pa}(V_i)$.

Output: $\widetilde{\mathcal{P}}_u(Y \mid \mathbf{PA}(Y))$

Fix a topological ordering on the vertices in $\boldsymbol{V}$, $\langle X_1, X_2, \ldots, X_n \rangle$; **for** $i = n, n-1, \ldots, 1$ **do**
  for *each assignment* $\boldsymbol{pa}(X_i)$ *to* $\boldsymbol{PA}(X_i)$ **do**
    Let a be a run in $\mathcal{A}$ with $\mathbf{pa}^a(X_i) = \mathbf{pa}(X_i)$. **for** $u = 1, \ldots, k$ **do**
      for $b = 0, 1$ **do**
        if X_i *is a bottom vertex* **then**
          Set $\widetilde{\mathcal{P}}_u(x_i^b \mid \mathbf{pa}^a(X_i)) \leftarrow \widetilde{\mathcal{P}}_u^a(x_i^b)$.
        else
          Set $\widetilde{\mathcal{P}}_u(x_i^b \mid \mathbf{pa}(X_i)) \leftarrow \frac{\widetilde{\mathcal{P}}_u^a(x_i^b)\widetilde{\mathcal{P}}_u(\mathbf{ch}^a(X_i)|\mathbf{top}^a(X_i),x_i^{1-b})}{\widetilde{\mathcal{P}}_u^a(x_i^b)\widetilde{\mathcal{P}}_u(\mathbf{ch}^a(X_i)|\mathbf{top}^a(X_i),x_i^{1-b})+\widetilde{\mathcal{P}}_u^a(x_i^{1-b})\widetilde{\mathcal{P}}_u(\mathbf{ch}^a(X_i)|\mathbf{top}^a(X_i),x_i^b)}$;
        end
      end
    end
  end
end

Algorithm 2: The k-MixBND algorithm

Input: A good collection of runs $\mathcal{A}$ of size at most $2^{O(\Delta^2)}$; a target accuracy ε.

Output: $\widetilde{\mathcal{P}}_u(Y \mid \mathbf{PA}(Y))$ and $\widetilde{\mathcal{P}}(U)$.

Estimate $\widetilde{\mathcal{P}}^a(\boldsymbol{I}^a)$ for all runs $a \in \mathcal{A}$ using

$$n \log n \cdot \varepsilon^{-2}(\Delta+1)^{O(k)} 2^{O(\Delta^2+\Delta k)} (\min \pi_i)^{-O(\log k)} (1/\zeta)^{O(k \log k + \Delta^2 k)}$$

samples from $\mathcal{P}$. Set $\varepsilon' \leftarrow \varepsilon/(6(\Delta+1))^{18k}$. **for** *each run* $a \in \mathcal{A}$ **do**
  Let $Q := \mathcal{P}^a(\boldsymbol{I}^a)$. Set $\left([m_{iu}^a]_{i \in \boldsymbol{I}^a, u \in [k]}, (\pi_u^a)_{u \in [k]}\right) \leftarrow$ LEARNPRODUCTMIXTURE(Q, ε'), where $m_{ij}^a = Q_u(x_i^1) = P_u(x_i^1 \mid \mathbf{cond}^a)$ and $\pi_u^a = P(U = u \mid \mathbf{cond}^a)$.
end

Run Algorithm **??** to align the sources in the output of all the runs in $\mathcal{A}$. Run Algorithm 1 to unzip the parameters. Fix any run $a \in \mathcal{A}$. **for** $j = 1, 2, \ldots, k$ **do**
  Set $\widetilde{\mathcal{P}}(u^j) \leftarrow \frac{\widetilde{\mathcal{P}}(u^j|\mathbf{cond}^a)\widetilde{\mathcal{P}}(\mathbf{cond}^a)}{\widetilde{\mathcal{P}}(\mathbf{cond}^a|u^j)}$.
end

Appendix B. Analyzing Bayesian Unzipping

Lemma 21 *Let a be a run with $Y \in \boldsymbol{I}^a$. Then we can compute $\mathcal{P}_u(y^b \mid \boldsymbol{pa}^a(Y))$ for $b \in \{0,1\}$ as follows:*

1. *If $Y \in \boldsymbol{BOT}(\boldsymbol{I}^a)$, then*
$$\mathcal{P}_u(y^b \mid \boldsymbol{pa}^a(Y)) = \mathcal{P}_u^a(y^b).$$
2. *If $Y \notin \boldsymbol{BOT}(\boldsymbol{I}^a)$, then*
$$\mathcal{P}_u(y^b \mid \boldsymbol{pa}^a(y)) = \frac{\mathcal{P}_u^a(y^b)\mathcal{P}_u(\boldsymbol{ch}^a(y) \mid \boldsymbol{top}^a(y), y^{1-b})}{\mathcal{P}_u^a(y^b)\mathcal{P}_u(\boldsymbol{ch}^a(y) \mid \boldsymbol{top}^a(y), y^{1-b}) + \mathcal{P}_u^a(y^{1-b})\mathcal{P}_u(\boldsymbol{ch}^a(y) \mid \boldsymbol{top}^a(y), y^b)} \tag{6}$$

Proof In the following we'll fix $\mathbf{ch}^a := \mathbf{ch}^a(Y)$, $\mathbf{mb}^a := \mathbf{mb}^a(Y)$, $\mathbf{pa}^a := \mathbf{pa}^a(Y)$, and $\mathbf{top}^a := \mathbf{top}^a(Y)$. If $Y \in \mathbf{BOT}(\boldsymbol{I}^a)$, then $\mathcal{P}_u^a(y^b) = \mathcal{P}_u(y^b \mid \mathbf{pa}^a(Y))$, so the claim is trivially true. If $Y \notin \mathbf{BOT}(\boldsymbol{I}^a)$, then

$$\mathcal{P}_u^a(y^b) = \mathcal{P}_u^a(y^b \mid \mathbf{mb}^a) = \frac{\mathcal{P}_u(y^b, \mathbf{mb}^a)}{\mathcal{P}_u(\mathbf{mb}^a)} = \frac{\mathcal{P}_u(y^b, \mathbf{mb}^a)}{\sum_{b'=0,1} \widetilde{\mathcal{P}}_u(y^{b'}, \mathbf{mb}^a)}$$

and we can expand $\mathcal{P}_u(y^b, \mathbf{mb}^a)$ as

$$\mathcal{P}_u(y^b, \mathbf{mb}^a) = \mathcal{P}_u(y^b \mid \mathbf{pa}^a)\mathcal{P}_u(\mathbf{ch}^a \mid \mathbf{top}^a, y^b)\mathcal{P}_u(\mathbf{top}^a).$$

Substituting this back into the preceding equation, we obtain

$$\begin{aligned}\mathcal{P}_u^a(y^b) &= \frac{\mathcal{P}_u(y^b \mid \mathbf{pa}^a)\mathcal{P}_u(\mathbf{ch}^a \mid \mathbf{top}^a, y^b)\mathcal{P}_u(\mathbf{top}^a)}{\sum_{b' \in \{0,1\}} \mathcal{P}_u(y^{b'} \mid \mathbf{pa}^a(Y))\mathcal{P}_u(\mathbf{ch}^a \mid \mathbf{top}^a, y^{b'})\mathcal{P}_u(\mathbf{top}^a)} \\ &= \frac{\mathcal{P}_u(y^b \mid \mathbf{pa}^a)\mathcal{P}(\mathbf{ch}^a \mid \mathbf{top}^a, y^b)}{\sum_{b' \in \{0,1\}} \mathcal{P}_u(y^{b'} \mid \mathbf{pa}^a)\mathcal{P}_u(\mathbf{ch}^a \mid \mathbf{top}^a\, y^{b'})}.\end{aligned}$$

We now multiply both sides by the denominator (which is non-zero since all terms are strictly positive by assumption) and then simplify to obtain

$$\mathcal{P}_u(y^b \mid \mathbf{pa}^a)\left(\mathcal{P}_u^a(y^{1-b})\mathcal{P}(\mathbf{ch}^a \mid \mathbf{top}^a, y^b)\right) + \mathcal{P}_u(y^b \mid \mathbf{pa}^a)\left(-\mathcal{P}_u^a(y^{1-b})\mathcal{P}_u(\mathbf{ch}^a \mid \mathbf{top}^a, y^{1-b})\right) = 0.$$

When augmented with the equation

$$\mathcal{P}_u(y^b \mid \mathbf{pa}^a) + \mathcal{P}_u(y^{1-b} \mid \mathbf{pa}^a) = 1$$

we have a system of two equations in $\mathcal{P}_u(y^b \mid \mathbf{pa}^a), \mathcal{P}_u(y^{1-b} \mid \mathbf{pa}^a)$ which is non-singular whenever

$$\left(\mathcal{P}_u^a(y^{1-b})\mathcal{P}(\mathbf{ch}^a \mid \mathbf{top}^a, y^b)\right) \neq \left(-\mathcal{P}_u^a(y^{1-b})\mathcal{P}_u(\mathbf{ch}^a \mid \mathbf{top}^a, y^{1-b})\right).$$

Since all probabilities occurring are strictly positive, this will always be the case and we can solve the system for $\mathcal{P}_u(y^b \mid \mathbf{pa}^a)$. The resulting equation is

$$\mathcal{P}_u(y^b \mid \mathbf{pa}^a) = \frac{\mathcal{P}_u^a(y^b)\mathcal{P}(\mathbf{ch}^a \mid \mathbf{top}^a, y^{1-b})}{\mathcal{P}_u^a(y^b)\mathcal{P}(\mathbf{ch}^a \mid \mathbf{top}^a, y^{1-b}) + \mathcal{P}_u^a(y^{1-b})\mathcal{P}(\mathbf{ch}^a \mid \mathbf{top}^a, y^b)},$$

proving the claim. ■

The following standard inequalities will be useful throughout the analysis:

Observation 22

$$\frac{1+x}{1-x} \le 1+3x \quad \textit{and} \qquad 1-3x \le \frac{1-x}{1+x} \qquad \textit{for } x \in (0,1/4);$$
$$1-2rx \le (1-x)^r \quad \textit{and} \qquad (1+x)^r \le 1+2rx \qquad \textit{for } x \in (0,1), r \ge 1, rx \le 1.$$

Lemma 23 *Given access to* $\left\{\widetilde{\mathcal{P}}_u^a(Y)\right\}_{a\in\mathcal{A},j\in[k]}$ *for a good collection of runs* $\mathcal{A}$ *from a distribution* $\mathcal{P}$ *satisfying* $\left|\widetilde{\mathcal{P}}_u^a(y^b) - \mathcal{P}_u^a(y^b)\right| / \mathcal{P}_u^a(y^b) \le \varepsilon$ *for all* $Y \in \boldsymbol{V}$, $b \in \{0,1\}$, $u \in [k]$ *for* ε *sufficiently small (and the estimated probabilities used as input to* k*-MixProd oracles), the procedure in Algorithm 1 will output* $\widetilde{\mathcal{P}}_u(y^b \mid \boldsymbol{PA}(Y))$ *and* $\widetilde{\mathcal{P}}(u)$ *for all* $Y \in \boldsymbol{V}$, $b \in \{0,1\}$, $u \in [k]$ *satisfying*

$$\frac{\left|\widetilde{\mathcal{P}}_u(y^b \mid \boldsymbol{pa}(Y)) - \mathcal{P}_u(y^b \mid \boldsymbol{pa}(Y))\right|}{\mathcal{P}_u(y^b \mid \boldsymbol{pa}(Y))} \le (6(\Delta+1))^\ell \varepsilon$$

where ℓ *is the distance from* Y *to the nearest bottom vertex if* Y *doesn't appear as a bottom vertex and is* 0 *if* Y *is a bottom vertex.*

Proof We fix a topological ordering on $\mathcal{V}$, let's say $X_1, X_2, \ldots, X_n$. Now starting with the last vertex in topological order, X_n, and proceeding in decreasing order, we'll compute $\widetilde{\mathcal{P}}_u(X_i \mid \mathbf{PA}(X_i))$. For bottom vertices, we set $\widetilde{\mathcal{P}}_u(x_i^b \mid \mathbf{pa}(X_i)) = \widetilde{\mathcal{P}}_u^a(x_i^b)$ for some run a with $\mathbf{pa}(X_i) = \mathbf{pa}^a(X_i)$. It immediately follows that $P_u(X_i \mid \mathbf{PA}(X_i))$ satisfies the desired relative error bound. Inductively, assume we've already computed $\widetilde{\mathcal{P}}_u(V_m \mid \mathbf{PA}(V_m))$ for all $m > i$ satisfying the stated bounds, and that we also have access to $\widetilde{\mathcal{P}}_u^a(V_i) = \widetilde{\mathcal{P}}_u(V_i \mid \mathbf{COND}^a)$ for a subset of the runs $a \in \mathcal{A}$ that cover V_i. To streamline notation, let $Y := V_i$. Now fix a run $a \in \mathcal{A}$ with $Y \in \boldsymbol{I}^a$.

Now we can bound each term above and below using the inductive hypothesis to get the desired result. In particular, we know that

$$(1-\varepsilon)\mathcal{P}_u^a(y^b) \le \widetilde{\mathcal{P}}_u^a(y^b) \le (1+\varepsilon)\mathcal{P}_u^a(y^b)$$

for $b \in \{0,1\}$ and

$$(1-(6(\Delta+1))^{\ell-1}\varepsilon)\mathcal{P}_u(v^b \mid \mathbf{pa}^a(v^b)) \le \widetilde{\mathcal{P}}_u(v^b \mid \mathbf{pa}^a(v^b)) \le (1+(6(\Delta+1))^{\ell-1}\varepsilon)\mathcal{P}_u(v^b \mid \mathbf{pa}^a(v^b))$$

for all $V \in \mathbf{CH}(Y)$ and $b \in \{0,1\}$ which implies that both the numerator and denominator of (6) are within a factor of $(1 \pm \varepsilon)(1 \pm (6(\Delta+1))^{\ell-1}\varepsilon)^\Delta$ of the correct value for those terms. It immediately follows that $\widetilde{\mathcal{P}}_u(y^b \mid \mathbf{pa}^a(Y))$ is bounded by

$$\mathcal{P}_u(y^b \mid \mathbf{pa}^a(Y))\frac{1-\varepsilon}{1+\varepsilon}\left(\frac{1-(6(\Delta+1))^{\ell-1}\varepsilon}{1+(6(\Delta+1))^{\ell-1}\varepsilon}\right) \le \widetilde{\mathcal{P}}_u(y^1 \mid \mathbf{pa}^a(Y))$$
$$\le \mathcal{P}_u(y^1 \mid \mathbf{pa}^a(Y))\frac{1+\varepsilon}{1-\varepsilon}\left(\frac{1+(6(\Delta+1))^{\ell-1}\varepsilon}{1-(6(\Delta+1))^{\ell-1}\varepsilon}\right).$$

We now use the inequalities from Observation 22 to simplify the bounds as follows:

$$\begin{aligned}\widetilde{\mathcal{P}}_u(y^b \mid \mathbf{pa}^a(Y)) &\leq \mathcal{P}_u(y^b \mid \mathbf{pa}^a(Y))\frac{1+\varepsilon}{1-\varepsilon}\left(\frac{1+(6(\Delta+1))^{\ell-1}\varepsilon}{1-(6(\Delta+1))^{\ell-1}\varepsilon}\right)^{\Delta} \\ &\leq \mathcal{P}_u(y^b \mid \mathbf{pa}^a(Y))(1+3\varepsilon)(1+3(6(\Delta+1))^{\ell-1}\varepsilon)^{\Delta} \\ &\leq \mathcal{P}_u(y^b \mid \mathbf{pa}^a(Y))(1+3(6(\Delta+1))^{\ell-1}\varepsilon)^{\Delta+1} \\ &\leq \mathcal{P}_u(y^b \mid \mathbf{pa}^a(Y))(1+2\cdot 3(\Delta+1)(6(\Delta+1))^{\ell-1}\varepsilon) \\ &\leq \mathcal{P}_u(y^b \mid \mathbf{pa}^a(Y))(1+(6(\Delta+1))^{\ell}\varepsilon).\end{aligned}$$

The lower bound is analogous. ■

Lemma 24 *Given access to* $\left\{\widetilde{\mathcal{P}}_u^a(Y)\right\}_{a\in\mathcal{A},j\in[k]}$ *for a good collection of runs* $\mathcal{A}$ *from a distribution* $\mathcal{P}$ *satisfying* $\left|\widetilde{\mathcal{P}}_u^a(y^b)-\mathcal{P}_u^a(y^b)\right|/\mathcal{P}_u^a(y^b)\leq\varepsilon$ *for all* $Y\in\boldsymbol{I}^a$, $b\in\{0,1\}$, $u\in[k]$ *for* ε *sufficiently small, the procedure in Algorithm 1 will output* $\widetilde{\mathcal{P}}_u(y^b \mid \boldsymbol{PA}(Y))$ *and* $\widetilde{\mathcal{P}}(u)$ *for all* $Y\in\boldsymbol{V}$, $b\in\{0,1\}$, $u\in[k]$ *satisfying*

$$\frac{\left|\widetilde{\mathcal{P}}_u(y^b \mid \boldsymbol{pa}(Y))-\mathcal{P}_u(y^b \mid \boldsymbol{pa}(Y))\right|}{\mathcal{P}_u(y^b \mid \boldsymbol{pa}(Y))}\leq(6(\Delta+1))^{3N_{mp}}\varepsilon$$

and

$$\frac{\left|\widetilde{\mathcal{P}}(u^j)-\mathcal{P}(u^j)\right|}{\mathcal{P}(u^j)}\leq 5\Delta^2\varepsilon.$$

Since $\mathcal{P}_u(y^b \mid \boldsymbol{pa}(Y)),\mathcal{P}(u^j)\leq 1$, *we also have that*

$$\left|\widetilde{\mathcal{P}}_u(y^b \mid \boldsymbol{pa}(Y))-\mathcal{P}_u(y^b \mid \boldsymbol{pa}(Y))\right|\leq(6(\Delta+1))^{9k}\varepsilon \quad \textit{and} \quad \left|\widetilde{\mathcal{P}}(u^j)-\mathcal{P}(u^j)\right|\leq 5\Delta^2\varepsilon.$$

Proof The error bound on $\widetilde{\mathcal{P}}_u(y^b \mid \mathbf{pa}(Y))$ follows from Lemma 23 above and the depth bound of $9k$ on non-bottom vertices.

For the error bound on $\widetilde{\mathcal{P}}(u^j)$ we write

$$\widetilde{\mathcal{P}}(u^j)=\frac{\widetilde{\mathcal{P}}(u^j \mid \mathbf{cond}^a)\widetilde{\mathcal{P}}(\mathbf{cond}^a)}{\widetilde{\mathcal{P}}(\mathbf{cond}^a \mid u^j)}$$

and analyze each term separately. First, define $Z\coloneqq\mathbf{AN}(\mathbf{COND}^a)\setminus\mathbf{COND}^a$ to be all ancestors of vertices conditioned upon in a minus $\mathbf{COND}^a$. Fix a topological order on the vertices in $Z\cup\mathbf{COND}^a$,

$\langle X_1, \ldots, X_m \rangle$. We can write

$$\begin{aligned}
\widetilde{\mathcal{P}}(\mathbf{cond}^a \mid u^j) &= \sum_{z \in \{0,1\}^Z} \prod_{i=1}^{m} \widetilde{\mathcal{P}}(z(X_i) \mid z(X_1, \ldots, X_{i-1}), \mathbf{cond}^a) \\
&= \sum_{z \in \{0,1\}^Z} \prod_{X_i \in Z \cup \mathbf{COND}^a} \widetilde{\mathcal{P}}(z(X_i) \mid z(\mathbf{PA}(X_i)), \mathbf{pa}^a(X_i)) \\
&= \sum_{z \in \{0,1\}^Z} \left(\prod_{X_i \in Z} \widetilde{\mathcal{P}}(z(X_i) \mid z(\mathbf{PA}(X_i)), \mathbf{pa}^a(X_i)) \right) \\
&\quad \left(\prod_{X_i \in \mathbf{COND}^a} \widetilde{\mathcal{P}}(z(X_i) \mid z(\mathbf{PA}(X_i)), \mathbf{pa}^a(X_i)) \right).
\end{aligned}$$

Now let C_z denote the value in the first grouped product for a given $z \in \{0,1\}^Z$. Since $\sum_{z \in \{0,1\}^Z} C_z = 1$, we can bound the error in the result by the error in the second grouped product:

$$(1-\varepsilon)^{|\mathbf{COND}^a|} \mathcal{P}(\mathbf{cond}^a \mid u^j) \leq \widetilde{\mathcal{P}}(\mathbf{cond}^a \mid u^j) \leq (1+\varepsilon)^{|\mathbf{COND}^a|} \mathcal{P}(\mathbf{cond}^a \mid u^j).$$

Finally using the bound $|\mathbf{COND}^a| < 2\Delta^2$ and the inequalities from Observation 22 we obtain

$$(1 - 2\Delta^2\varepsilon)\mathcal{P}(\mathbf{cond}^a \mid u^j) \leq \widetilde{\mathcal{P}}(\mathbf{cond}^a \mid u^j) \leq (1 + 2\Delta^2\varepsilon)\mathcal{P}(\mathbf{cond}^a \mid u^j).$$

By assumption $\widetilde{\mathcal{P}}(u^j \mid \mathbf{cond}^a) \in (1 \pm \varepsilon)\mathcal{P}(u^j \mid \mathbf{cond}^a)$; $\widetilde{\mathcal{P}}(\mathbf{cond}^a)$ is also known up to multiplicative accuracy ε since the sampling needed to estimate the distributions that are input to k-MixProd oracles far exceed that needed to get an ε estimate of this quantity. Thus, the resulting bound on $\widetilde{\mathcal{P}}(u^j)$ is

$$(1 - 2\Delta^2\varepsilon)(1 - 3\varepsilon)\mathcal{P}(u^j) \leq \widetilde{\mathcal{P}}(\mathbf{cond}^a \mid u^j) \leq (1 + 2\Delta^2\varepsilon)(1 + 3\varepsilon)\mathcal{P}(\mathbf{cond}^a \mid u^j)$$

which can be simplified to

$$(1 - 2\Delta^2\varepsilon - 3\varepsilon)\mathcal{P}(u^j) \leq \widetilde{\mathcal{P}}(\mathbf{cond}^a \mid u^j) \leq (1 + 2\Delta^2\varepsilon + 3\varepsilon)\mathcal{P}(\mathbf{cond}^a \mid u^j)$$

using $(1-x)(1-y) \geq (1-x-y)$ for $x, y \in [0,1)$ and $2\Delta^2\varepsilon + 3\varepsilon \leq 5\Delta^2\varepsilon$ for $\Delta \geq 1$. ■

Appendix C. Finding good collections of runs

In this section we prove Theorem 17 and give a few different sufficient conditions for the existence of a good collection of runs.

C.1. Constructing a good collection of runs from $N_{\mathbf{mp}}$ centers

One sufficient condition for the existence of a good collection of runs is the presence of N_{mp} variables with disjoint Markov boundaries. We will now present a good collection of runs for this case.

Lemma 25 *Given a set $\boldsymbol{X} = \{X_1, \ldots, X_{N_{mp}}\} \subseteq \boldsymbol{V}$ of N_{mp} variables with depth at most $3N_{mp}$ and disjoint Markov boundaries, Algorithm 3 finds a good collection of runs $\mathcal{A}$, satisfying $|\mathcal{A}| = O(2^\gamma n)$ where $\gamma = \max_{V \in \boldsymbol{V}} |\boldsymbol{MB}(V)|$,.*

Algorithm 3: Building a good collection of runs

Input: Vertices $\boldsymbol{X} = \{X_1, \ldots, X_{3k-3}\} \subseteq \boldsymbol{V}$ having disjoint Markov boundaries with maximum depth $3N_{\text{mp}}$.

Output: A good collection of runs $\mathcal{A}$.

Let a_0 be a run with $\boldsymbol{I}_{a_0} = \{X_1, \ldots, X_{3k-3}\}$ and $\mathbf{COND}^{a_0}$ chosen arbitrarily.

Set $\mathcal{A} \leftarrow \{a_0\}$.

for $i = 1, \ldots, 3k-3$ **do**

  $\mathcal{A} \leftarrow \mathcal{A} \cup \left\{a_0[\mathbf{mb}(X_i)] : \mathbf{mb}(X_i) \in \{0,1\}^{\mathbf{MB}(X_i)}\right\}$. **for** $Y \in \boldsymbol{V} \setminus \boldsymbol{X}$ **do**

    if $Y \in \boldsymbol{MB}(X)$ *for some* $X \in \mathcal{X}$ **then**

      $\boldsymbol{I}^a = \boldsymbol{X} - X_i + Y$

    end

    else

      $\boldsymbol{I}^a = \boldsymbol{X} + Y$

    end

    for $\boldsymbol{pa}(Y) \in \{0,1\}^{|\boldsymbol{PA}(Y)|}$ **do**

      if $Y \in \boldsymbol{AN}(\boldsymbol{I}^a - Y)$ **then**

        $\mathbf{COND}^a = \mathbf{MB}(\boldsymbol{I}^a)$

      end

      else

        $\mathbf{COND}^a = \mathbf{MB}(\boldsymbol{I}^a - Y) \cup \mathbf{PA}(Y)$

      end

      $\mathbf{COND}^a = \mathbf{MB}(\boldsymbol{I}^a)$

      $f^a(\mathbf{PA}(Y)) = \mathbf{pa}(Y)$

      $\mathsf{Defaults} = \mathbf{COND}^{a_0} \cap \mathbf{COND}^a \setminus \mathbf{PA}(Y)$ $f^a(\mathsf{Defaults}) = f^{a_0}(\mathsf{Defaults})$ $f^a(\mathbf{COND}^a \setminus \mathbf{COND}^{a_0} \setminus \mathbf{PA}(Y))$ are chosen arbitrarily. $\mathcal{A} \leftarrow \mathcal{A} \cup \{a\}$, where a is given by $a = (\boldsymbol{I}^a, f^a)$.

    end

  end

end

Claim 26 *By construction, $\mathcal{A}_C$ covers $\boldsymbol{X}$ and is alignable.*

Claim 27 *A_Y covers $\boldsymbol{V} \setminus \boldsymbol{X}$ and each run in $\mathcal{A}_Y$ can be aligned with some run in $\mathcal{A}_C$.*

Proof The fact that $\mathcal{A}_Y$ covers $\boldsymbol{V} \setminus \boldsymbol{X}$ follows immediately from $\mathcal{A}_Y$ containing a run assigning for independent variable $Y \in \boldsymbol{V} \setminus \boldsymbol{X}$ each possible assignment $\mathbf{pa}(Y)$. Fix any run $a \in \mathcal{A}_Y$ with $\boldsymbol{I}^a = \boldsymbol{X} - X_i + Y$ or $\boldsymbol{I}^a = \boldsymbol{X} + Y$ depending on whether Y overlaps with $\mathbf{MB}(\mathcal{X})$. Now if $\mathbf{MB}(Y) \cap \mathbf{MB}(X_j) = \emptyset$ for any $j \neq i$, a and a_0 are aligned at X_j. If instead $\mathbf{MB}(Y) \cap \mathbf{MB}(X_j) \neq \emptyset$ for all $j \neq i$, pick any j and consider the central run $a_0[\mathbf{mb}^a(X_j)] \in \mathcal{A}_C$. Clearly, a and $a_0[\mathbf{mb}^a(X_j)]$ are aligned at X_j. In either case, we've aligned a to a run in $\mathcal{A}_C$. ■

Claim 28 *Every run $a \in \mathcal{A}$ is at least N_{mp}-independent.*

Proof [Proof of Lemma 25] This follows immediately from Claims 26, 27, and 28 ■

C.2. Degree bounds

We can ensure that N_{mp} centers can be found on certain degree-bounded graphs, which in turn bound γ. Let Δ_{in} upper bound on the in-degree of any vertex in $\mathcal{G}$ and let Δ_{out} upper bound the out-degree. Then

$$\gamma \leq \Delta_{\text{in}} + \Delta_{\text{out}} + \Delta_{\text{out}}(\Delta_{\text{in}} - 1) = \Delta_{\text{in}} + \Delta_{\text{out}}\Delta_{\text{in}}.$$

If we have a bound Δ on the degree of the undirected skeleton of $\mathcal{G}$, we get that

$$\gamma \leq \Delta(\Delta - 1) = \Delta^2 - \Delta.$$

Corollary 29 *If either of the following conditions hold, we can find N_{mp} centers for $\mathcal{G}$ with depth at most $3N_{mp}$:*

1. $n \geq N_{mp}(\Delta_{in}^2 + 2\Delta_{out}\Delta_{in} + \Delta_{out}^2\Delta_{in}^2 - \Delta_{in} - \Delta_{out} + 1) = N_{mp} \cdot O(\Delta_{out}^2\Delta_{in}^2)$.
2. $n \geq N_{mp}(\Delta^4 - 2\Delta^3 + \Delta + 1) = N_{mp} \cdot O(\Delta^4)$.

Proof [Proof of Lemma 17] This follows immediately from Corollary 29 and Lemma 25. ■

C.3. Mixtures of paths

Special cases do not require the condition of N_{mp} disjoint Markov boundaries. One of these special cases is a mixture of paths $v_1 \to v_2 \to v_3 \to \cdots v_n$ over k. We will will give a good set of runs for $n \geq 2N_{\text{mp}}$.

First, we will specify three default runs, which we will call ODD, EVEN, and LINK.

Definition 30 (ODD default run for Markov chains) *Run ODD is specified by an independence set of vertices with odd indices $\boldsymbol{I}^{ODD} = \{V_1, V_3, V_5, \ldots, V_{2N_{mp}-1}\}$. The conditioning set is given by the evenly indexed vertices $\boldsymbol{COND}^{ODD} = \{V_2, V_4, V_6, \ldots, V_{2N_{mp}-2}\}$. f^{ODD} may be chosen arbitrarily, but for simplicity we will give $f^{ODD}(V_i) = 0$ for all $V_i \in \boldsymbol{COND}^{ODD}$.*

Definition 31 (EVEN default run for Markov chains) *Run EVEN is defined the same way for evenly indexed vertices. That is,* $\boldsymbol{I}^{\mathsf{EVEN}} = \{V_2, V_4, V_6, \ldots, V_{2N_{mp}}\}$, $\boldsymbol{COND}^{\mathsf{EVEN}} = \{V_1, V_3, V_5, \ldots, V_{2N_{mp}-1}\}$, *and* $f^{\mathsf{EVEN}}(v_i) = 0$ *for all* $V_i \in \boldsymbol{COND}^{\mathsf{EVEN}}$.

Definition 32 (LINK default run for Markov chains) *Run LINK is part evenly indexed vertices and part oddly indexed vertices.* $\boldsymbol{I}^{\mathsf{LINK}} = \boldsymbol{I}^{\mathsf{EVEN}} \cup \{V_1\} \setminus \{V_2\}$. *Similarly, we condition on the complement* $\boldsymbol{COND}^{\mathsf{LINK}} = \{V_2, V_3, V_5, \ldots, V_{2N_{mp}-1}\}$ *being* 0.

Claim 33 *If* $n \geq 2N_{mp}$, *then ODD, EVEN, and LINK are well-formed runs.*

Claim 34 *ODD and LINK are aligned at* V_1. *EVEN and LINK are aligned at evenly indexed vertices beyond* V_2.

Now, we enumerate a set of runs that cover all possible entries to parents of vertices.

Definition 35 *Let* $\mathsf{ODD}[v_i]$ *denote a run on* $\boldsymbol{I}^{\mathsf{ODD}[V_i]} = \boldsymbol{I}^{\mathsf{ODD}}$, $\boldsymbol{COND}^{\mathsf{ODD}[V_i]} = \boldsymbol{COND}^{\mathsf{ODD}}$ *with* $f^{\mathsf{ODD}[V_i]}(V_j) = 1$ *if* $i = j$ *and* 0 *if* $i \neq j$. *Similarly define* $\mathsf{EVEN}[V_i]$ *to be a run on* $\boldsymbol{I}^{\mathsf{EVEN}[V_i]} = \boldsymbol{I}^{\mathsf{EVEN}}$, $\boldsymbol{COND}^{\mathsf{EVEN}[V_i]} = \boldsymbol{COND}^{\mathsf{EVEN}}$ *with* $f^{\mathsf{EVEN}[V_i]}(v_j) = 1$ *if* $i = j$ *and* 0 *if* $i \neq j$.

Claim 36 *Any run* $\mathsf{ODD}[V_i]$ *is aligned with ODD at* V_j *for* $j < i - 1$ *or* $j > i + 1$. *Similarly, any run* $\mathsf{EVEN}[V_i]$ *is aligned with EVEN at* V_j *for* $j < i - 1$ *or* $j > i + 1$.

Definition 37 *Let* $\mathsf{TAIL}^b[V_i]$ *for* $i > 2N_{mp}$ *give runs which have*

$$\boldsymbol{I}^{\mathsf{TAIL}[V_i]} = \{V_1, V_3, V_5, \ldots, V_{2N_{mp}-1}, V_i\}$$
$$\boldsymbol{COND}^{\mathsf{TAIL}[V_i]} = \{V_2, V_4, \ldots, V_{2N_{mp}-2}, V_{i-1}\}$$

with

$$f^{\mathsf{TAIL}[V_i]}(V_j) = \begin{cases} 0 & \text{if } i-1 \neq j \\ b & \text{if } i-1 = j \end{cases}$$

Claim 38 *Any run* $\mathsf{LINK}[V_i]$ *is aligned with ODD at* V_j *for* $j < 2N_{mp}$ *with even* j.

Claim 39 *The set*

$$\left\{\mathsf{ODD}[V_i] \ : \ v_i \in \boldsymbol{I}^{\mathsf{ODD}}\right\} \cup \left\{\mathsf{EVEN}[V_j] \ : \ V_j \in \boldsymbol{I}^{\mathsf{EVEN}}\right\} \cup \left\{\mathsf{TAIL}^0[V_j], \mathsf{TAIL}^1[V_j] \ : \ j > 6k - 6\right\}$$

covers $\boldsymbol{V}$.

Proof For all V_i with $i > 6k - 6$ we have $\mathsf{LINK}^0[V_i]$ and $\mathsf{LINK}^1[V_i]$ to cover both possible assignments to V_{i-1}.

Consider V_i with $i \leq 2N_{\text{mp}}$. If $i > 1$ is odd, then $f^{\mathsf{EVEN}[V_i]}(V_{i-1}) = 0$ and $f^{\mathsf{EVEN}}(V_{i-1}) = 1$, so all possible assignments to the parent of V_i are covered. Similarly, if i is even, then $f^{\mathsf{ODD}[V_i]}(V_{i-1}) = 0$ and $f^{\mathsf{ODD}}(V_{i-1}) = 1$, so all possible assignments to the parent of V_i are covered. Finally, if $i = 1$, then V_i has no parents. ■

We give the following example to illustrate this construction. Consider $k = 2$ and $n = 8$. We will express a run a as sequences of values $\texttt{0}$, $\texttt{1}$ or $\star$. A $\star$ in the ith location indicates $V_i \in \boldsymbol{I}^a$ and a $\texttt{0}$ or $\texttt{1}$ indicates $f^a(V_i) = 0$ or $f^a(V_i) = 1$, respectively. Finally, $-$ indicates that the variable is not conditioned on and not in the independent set (not in **COND** and not in $\boldsymbol{I}$). The default runs are:

$$\begin{aligned}
\mathsf{EVEN} &= \texttt{0}\star\texttt{0}\star\texttt{0}\star\texttt{--} \\
\mathsf{ODD} &= \star\texttt{0}\star\texttt{0}\star\star\texttt{--} \\
\mathsf{LINK} &= \star\texttt{00}\star\texttt{0}\star\texttt{--}
\end{aligned}$$

In addition to these runs, we have:

$$\begin{aligned}
\mathsf{ODD}[V_1] &= \texttt{1}\star\texttt{0}\star\texttt{0}\star\texttt{--} \\
\mathsf{ODD}[V_3] &= \texttt{0}\star\texttt{1}\star\texttt{0}\star\texttt{--} \\
\mathsf{ODD}[V_5] &= \texttt{0}\star\texttt{0}\star\texttt{1}\star\texttt{--} \\
\mathsf{EVEN}[V_2] &= \star\texttt{1}\star\texttt{0}\star\texttt{---} \\
\mathsf{EVEN}[V_4] &= \star\texttt{0}\star\texttt{1}\star\texttt{---} \\
\mathsf{EVEN}[V_6] &= \star\texttt{0}\star\texttt{0}\star\texttt{---} \\
\mathsf{TAIL}^0[V_7] &= \star\texttt{0}\star\texttt{0-0}\star\texttt{-} \\
\mathsf{TAIL}^1[V_7] &= \star\texttt{0}\star\texttt{0-1}\star\texttt{-} \\
\mathsf{TAIL}^0[V_8] &= \star\texttt{0}\star\texttt{0--0}\star \\
\mathsf{TAIL}^1[V_8] &= \star\texttt{0}\star\texttt{0--1}\star
\end{aligned}$$

The construction of a good set of runs given does not use disjoint Markov boundaries, yielding greater efficiency than our more general algorithm for degree bounded graphs.

While similar in name and structure, this setting is different from that of hidden Markov models. Hidden Markov models are Bayesian networks consisting of a chain of unobserved variables, each affecting a unique observed variable, with no causal relations among the observed variables. For instance, in Gupta et al. (2016), all chains in the mixture have the same state space. The sampling process selects a chain and a starting state from the mixture distribution, then generates a short observable path through the chain. In their model, under some conditions, a sufficiently large collection of paths of length 3 suffices to recover the parameters of the mixture, using spectral methods. A different model is considered in Batu et al. (2004). There, the constituents of the mixture have disjoint state spaces and the observation is a long sequence of interleaved paths in the separate chains. The primary challenge is to cluster the observable states into the k constituents. This model can be viewed as a special case of the hidden Markov model problem.

The natural extension of EVEN and ODD vertices is a two-coloring on a tree which denotes sets that are of even or odd distance from the root. One can construct a good set of runs for learning mixtures of trees of size $n \geq 2N_{\text{mp}}$ that is very similar to the one given for mixtures of paths.

C.4. Larger alphabets for mixtures of DAGs

The simplest reduction for larger alphabets is to replace each vertex with a clique of d binary vertices which represent the value of the nonbinary vertex. The pseudocode for this process is given in Algorithm 4.

Algorithm 4: Reduction for larger alphabets
Input: A DAG $(\boldsymbol{V}, \mathcal{E})$ on n variables $V_1, \ldots, V_n \in [d]$.
Output: A DAG $(\mathcal{W}, \mathcal{E}_W)$ on dn binary variables $W_1^1, \ldots W_1^d, \ldots W_n^1, \ldots W_n^d \in \{0, 1\}$.
Start with $\mathcal{E}_W \leftarrow \emptyset$ **for** *each vertex* $i \in [n]$ **do**
  Form a clique among $W_i^1, \ldots W_i^d$ by adding directed edges (W_i^a, W_i^b) to $\mathcal{E}_W$ for all pairs $a < b$ with $a, b \in [d]$.
end
for *each directed edge* $(V_i, V_j) \in \mathcal{E}$ **do**
  Add $\{W_i^1, \ldots, W_i^d\} \times \{W_j^1, \ldots, W_j^d\}$ to $\mathcal{E}_W$.
end

Observation 40 *The maximum degree of* $(\mathcal{W}, \mathcal{E}_W)$ *outputted by Algorithm 4 is now at least* $\Delta \geq d$.

Observation 41 *The number of vertices* $|\mathcal{W}| = nd$.

We now give the function that translates data from the original graph to the new graph.

Definition 42 (One-hot encoding) *We define* $\chi(v) = (\mathbb{1}_0(v), \ldots, \mathbb{1}_{d-1}(v))$ *to give the one-hot encoding of the value of a variable* V.

This allows us to give the full algorithm.

Algorithm 5: DAG reduction for larger alphabets
Input: A DAG $(\boldsymbol{V}, \mathcal{E})$ on n variables $V_1, \ldots, V_n \in [d]$. And data with entries of the form $(v_1, v_2, \ldots, v_n)$.
Output: Parameters $\mathcal{P}_u(v_i \mid \mathbf{PA}(V_i))$ for $i \in [n]$.
Use Algorithm 4 to create a larger DAG on binary variables, called $\mathcal{G}'$.
One-hot encode data on each V_i into binary variables $\xi(V) = (W_i^1, \ldots, W_i^d)$.
Run the Mixture of DAGs algorithm for $\mathcal{G}'$ on the one-hot encoded data.
for *each parameter* $\mathcal{P}_u(v_i \mid \boldsymbol{PA}(V_i))$ **do**
  Let Z indicate zero assignments to W_i^b for $b \neq v_i$.
  One-hot encode each parent $V_j \in \mathbf{PA}(V_i)$ to obtain assignments to $\mathbf{PA}(W_i^{v_i}) \setminus \{w^b \ : \ b \neq v_i\}$, denoted $\mathbf{pa}^{\mathbf{OH}}(W_i^{v_i})$.
  Assign $\mathcal{P}_u(v_i \mid \mathbf{PA}(V_i)) = \mathcal{P}_u(W_i^{v_i} = 1 \mid Z, \mathbf{pa}^{\mathbf{OH}}(W_i^{v_i}))$.
end

Theorem 43 *If* $\mathcal{P}(\boldsymbol{V}) = \sum_u \mathcal{P}_u(\boldsymbol{V})\mathcal{P}(u)$ *is a mixture of* k *distributions over a DAG* $\mathcal{G} = (\boldsymbol{V}, \boldsymbol{E})$ *with* $\boldsymbol{V} \in \{0, \ldots, d-1\}$ *and there exists a set of* N_{mp} *centers, then there is an algorithm to compute estimates of all parameters to accuracy* ε. *If* $D = \max(d, \Delta)$ *then the algorithm uses* $O(nd2^{D^2})$ *calls to the* k*-MixProd oracle.*

Proof The algorithm uses the reduction in Algorithm 5. This involves running the algorithm on a larger graph with nd vertices and $\Delta \geq d$, which modifies the run-time and sample complexity as given. ■