Estimating Causal Effects from Learned Causal Networks

Anna Raichev[a,*], **Jin Tian**[b], **Alexander Ihler**[a] **and Rina Dechter**[a]

[a]University of California, Irvine
[b]Iowa State University

Abstract. The standard approach to answering an identifiable causal-effect query (e.g., $P(Y|do(X))$) given a causal diagram and observational data is to first generate an estimand, or probabilistic expression over the observable variables, which is then evaluated using the observational data. In this paper, we propose an alternative paradigm for answering causal-effect queries over discrete observable variables. We propose to instead learn the causal Bayesian network and its confounding latent variables directly from the observational data. Then, efficient probabilistic graphical model (PGM) algorithms can be applied to the learned model to answer queries. Perhaps surprisingly, we show that this *model completion* learning approach can be more effective than estimand approaches, particularly for larger models in which the estimand expressions become computationally difficult. We illustrate our method's potential using a benchmark collection of Bayesian networks and synthetically generated causal models.

1 Introduction

Structural Causal Models (SCMs) are a formal framework for reasoning about causal knowledge in the presence of uncertainty [25]. When the full SCM is available, it is possible to use standard probabilistic inference [6, 9] to directly answer causal queries that evaluate how forcing some variable's assignment $X = x$ affects another variable Y, written as $P(Y|do(X = x))$. However, in practice the full causal model is rarely available; instead, only a causal diagram – a directed graph capturing the causal relationships of the underlying SCM – is assumed to be known. Causal diagrams typically include both *observable variables*, which can be measured from data, and *latent variables*, which are unobservable and for which data cannot be collected. A linchpin of causal reasoning is that given a causal diagram, many causal queries can be uniquely answered using only the observable variables [32, 28, 29, 12] and consequently estimated using observational data.

The main approach developed in the past two decades for answering causal queries under such assumptions is a two-step process which we call *estimand-based causal inference*. The first step is to determine if the causal query is *identifiable* – i.e., uniquely answerable from the model's observational distribution – and if so, construct an expression ("estimand") that captures the answer symbolically using probabilities over only observed variables. Then, one can use the observed data to estimate the probabilities involved in the estimand expression. Over the past few years a variety of estimand-based strategies have been developed using different statistical estimation methods [15, 3].

However, many of these approaches focus on specific, small, causal models, and do not examine the scalability of their approaches or provide significant empirical comparisons. In larger models, estimand expressions can become large and unwieldy, making them either computationally difficult, hard to estimate accurately, or both.

In this paper, we propose a straightforward yet under-explored alternative: to learn the causal model, including latent dependencies, directly from the observed data. Although the domains and distributional forms of these latent variables are unknown, recent work [38] has shown that, if the visible variables are discrete, any SCM is equivalent to one with discrete latent variables, and gives an upper bound on their domain sizes. This allows us to apply well-known techniques for learning latent variable models, such as the Expectation-Maximization (EM) algorithm, to build a *Causal Bayesian Network (CBN)* over the observed and latent variables. We can further apply model selection techniques such as the Bayesian Information Criterion (BIC) [18] to select appropriate domain sizes. Then, given our learned model, we can use efficient algorithms for reasoning in probabilistic graphical models (PGMs) to answer one, or even many causal queries [6, 18].

Perhaps surprisingly, we show that this approach is often significantly more effective than the estimand-based methodology. Furthermore, we provide structural conditions that help decide when each approach is likely to be more effective. The computational benefit of our approach is tied to the complexity of the causal graph: if the causal graph has low *treewidth*, then exact PGM algorithms (e.g., bucket elimination or join-tree scheme [8, 17]) are efficiently applicable. As an added benefit if there are multiple causal queries to perform on the same model, since the model is learned only once, the learning time can be amortized over all such queries.

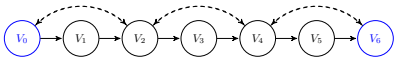
Our empirical evaluation incorporates a spectrum of models, including not only the small models common in causal effect literature, but also large models based on benchmark Bayesian networks and scalable classes of synthetic models; to the best of our knowledge this is the first such extensive empirical evaluation of causal effect queries.

Contributions. This paper presents a new path for answering causal effects queries by learning a Causal Bayesian Network that is consistent with a causal graph and observational data.

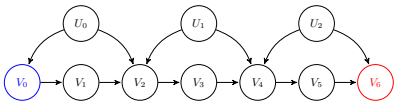
- We propose to answer causal queries by directly learning the full causal Bayesian network via EM, followed by query processing using traditional PGM algorithms.
- We provide a first of its kind, empirical evaluation of algorithms for causal effect queries on varied and large synthetic and real benchmarks.
- We show empirically that our proposed learning approach gives more accurate estimates than the estimand-based alternatives.

In settings with high-dimensional estimand expressions but low treewidth causal models, our approach allows more accurate estimates

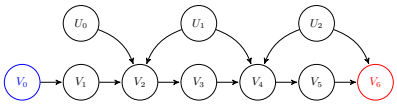
* Corresponding Author. Email: araichev@uci.edu.



(a) Causal diagram of a chain model with 7 observable and 3 latent variables. Dashed bidirected edges represent latent variables.



(b) The CBN for the model (a), showing latent variables explicitly.



(c) The truncated causal diagram after intervention $do(V_0)$.

Figure 1: Illustration of a causal diagram, CBN, and truncated CBN for evaluating a causal effect. Blue variables are intervened on and red variables are the outcome variables corresponding to the query $P(Y \mid do(X))$

by retaining more information from the causal graph, and allowing variance reduction without impacting computational tractability.

The rest of the paper is organized as follows. Section 2 provides background on related work and information and definitions, Section 3 outlines the main idea of our approach, Section 4 provides empirical evaluation, and we conclude in Section 5.

2 Background

We use capital letters X or V to represent variables, and lower case x or v for their values. Bold uppercase $\boldsymbol{X}$ denotes a collection of variables, with $|\boldsymbol{X}|$ its cardinality and $\mathcal{D}(\boldsymbol{X})$ their joint domain, while $\boldsymbol{x}$ indicates an assignment in that joint domain. We denote by $P(\boldsymbol{X})$ the joint probability distribution over $\boldsymbol{X}$ and $P(\boldsymbol{X} = \boldsymbol{x})$ the probability of $\boldsymbol{X}$ taking configuration $\boldsymbol{x}$ under this distribution, which we abbreviate $P(\boldsymbol{x})$. Similarly, $P(\boldsymbol{Y}|\boldsymbol{X})$ represents the set of conditional distributions $P(\boldsymbol{Y} \mid \boldsymbol{X} = \boldsymbol{x}), \forall \boldsymbol{x}$. In the definitions ahead we will use $\boldsymbol{V}$ and $\boldsymbol{U}$ to stand for arbitrary variables in models while use X and Y to refer specifically to variables that appear in queries.

Definition 1 (Structural Causal Model)**.** *A structural causal model (SCM) [25] is a 4-tuple $\mathcal{M} = \langle \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{F}, P(\boldsymbol{U}) \rangle$ where: (1) $\boldsymbol{U} = \{U_1, U_2, ..., U_k\}$ is a set of exogenous latent variables whose values are affected by factors outside the model; (2) $\boldsymbol{V} = \{V_1, V_2, ..., V_n\}$ is a set of endogenous, observable variables of interest whose values are determined by other variables in the model; (3) $\boldsymbol{F} = \{f_i : V_i \in \boldsymbol{V}\}$ is a set of functions f_i such that each f_i determines the value v_i of V_i as a function of V_i's causal parents $PA_i \subseteq \boldsymbol{U} \cup (\boldsymbol{V} \setminus V_i)$ so that $f_i : \mathcal{D}(PA_i) \to \mathcal{D}(V_i)$ and $v_i \leftarrow f_i(pa_i)$; and (4) $P(\boldsymbol{U})$ is a probability distribution over the latent variables. The latent variables are assumed to be mutually independent, i.e., $P(\boldsymbol{U}) = \prod_{U \in \boldsymbol{U}} P(U)$.*

Causal diagrams. An SCM $\mathcal{M} = \langle \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{F}, P(\boldsymbol{U}) \rangle$ induces a *causal diagram* $\mathcal{G} = \langle \boldsymbol{V} \cup \boldsymbol{U}, E \rangle$, where each node in the graph maps to a variable in the SCM, and there is an arc from every observed or latent node X in the graph to V_i iff $X \in PA_i$. Thus PA_i are the parents nodes of node X in $\mathcal{G}$. An SCM whose latent variables connect to at most a single observable variable is referred to as *Markovian*, and one whose latent variables connect to at most two observable variables is called *Semi-Markovian*. It is known that any SCM can be transformed into an equivalent Semi-Markovian one such that answers to causal queries are preserved [31]. In the semi-Markovian case it is common to use a simplified causal diagram called an *Acyclic Directed Mixed Graph* (ADMG), which omits any latent variables with a single child, and replaces any latent variable with two children with a bidirectional dashed arc between the children, so that latent variables are no longer explicitly shown (see, e.g., Figure 1).

The SCM $\mathcal{M}$ also induces a **Causal Bayesian Network (CBN)** $\mathcal{B} = \langle \mathcal{G}, \mathcal{P} \rangle$ specified by $\mathcal{M}$'s causal diagram $\mathcal{G} = \langle \boldsymbol{V} \cup \boldsymbol{U}, E \rangle$ along with its associated conditional probability distributions $\mathcal{P} = \{P(V_i|PA_i), P(U_j)\}$. The distribution $P(\boldsymbol{V}, \boldsymbol{U})$ factors according to the causal diagram:

$$P(\boldsymbol{V}, \boldsymbol{U}) = \prod_{V_i \in \boldsymbol{V}} P(V_i|PA_i) \cdot \prod_{U_j \in \boldsymbol{U}} P(U_j). \tag{1}$$

The *observational distribution*, $P(\boldsymbol{V})$, is given by

$$P(\boldsymbol{V}) = \sum_{\boldsymbol{U}} P(\boldsymbol{V}, \boldsymbol{U}). \tag{2}$$

Causal effect and the truncation formula. An external intervention, forcing variables $\boldsymbol{X} \subseteq \boldsymbol{V}$ to take on value $\boldsymbol{x}$, is modeled by replacing the mechanism for each $X \in \boldsymbol{X}$ with the function $X = x$. This is formally represented by the do-operator $do(\boldsymbol{X} = \boldsymbol{x})$. Thus the interventional distribution of an SCM $\mathcal{M}$ when applying $do(\boldsymbol{X})$ is defined by,

$$P(\boldsymbol{V} \setminus \boldsymbol{X}, \boldsymbol{U} \mid do(\boldsymbol{X} = \boldsymbol{x})) = \prod_{V_j \notin \boldsymbol{X}} P(V_j|PA_j) \cdot P(\boldsymbol{U}) \Big|_{\boldsymbol{X} = \boldsymbol{x}} \tag{3}$$

i.e., it is obtained from Eq. (1) by truncating from $\mathcal{M}$ the factors corresponding to the variables in $\boldsymbol{X}$ and setting $\boldsymbol{X} = \boldsymbol{x}$. The effect of $do(\boldsymbol{X})$ on a variable Y, denoted $P(Y|do(\boldsymbol{X}))$, is defined by marginalizing all the variables other than Y.

Causal-effect queries. If we have access to the full causal model we can answer causal-effect queries from the full model using traditional probabilistic inference. However, normally we have no access to the full model $\mathcal{M}$. Yet, it was shown that assuming only a causal graph it is often possible to evaluate the effect of an intervention from the observational distribution $P(\boldsymbol{V})$ only. This occurs if the answer is unique for any full model that is consistent with the graph and $P(\boldsymbol{V})$ [25]. More formally:

Definition 2 (Identifiability, causal-effect)**.** *Given a causal diagram $\mathcal{G}$, the* **causal-effect** *query $P(\boldsymbol{Y} \mid do(\boldsymbol{X}))$ is* **identifiable** *if any two SCMs having the same $\mathcal{G}$ that agree on the observational distribution $P(\boldsymbol{V})$ also agree on $P(\boldsymbol{Y} \mid do(\boldsymbol{X}))$.*

Identifiability makes the causal-effect query well-posed.

Estimand-based approaches. The now-standard methodology for answering causal-effect queries is to break the task into two steps. The first is the *identifiability step*: given a causal diagram and a query $P(\boldsymbol{Y} \mid do(\boldsymbol{X}))$, determine if the query is identifiable and if so, generate an *estimand*, or an algebraic expression in terms of the observational distribution $P(\boldsymbol{V})$, that answers the query. A complete polynomial algorithm called ID has been developed for this task [31, 28]. The second step is *estimation*: use samples from the observational distribution $P(\boldsymbol{V})$ to estimate the value of the estimand expression.

A number of approaches have been applied to the estimation in the second step. A simple approach, called the *plug-in estimator*, replaces each term in the estimand with its empirical probability value in the observed data. More sophisticated approaches include the line of work by Jung et al., which apply ideas from data weighting and empirical risk minimization [13, 14] and double or debiased machine learning [15, 16]. Evaluating the estimand's value from a PAC perspective is analyzed in [4].

In practice, the estimand expression typically consists of multiple products or ratios of partially marginalized distributions. We illustrate with an example:

Example 1. *Consider the model in Figure 1a. To evaluate the query $P(V_6 \mid do(V_0))$, the ID algorithm [31, 28] gives the expression:*

$$P(V_6 \mid do(V_0)) = \sum_{V_1,V_2,V_3,V_4,V_5} P(V_5|V_0,V_1,V_2,V_3,V_4) \times P(V_3|V_0,V_1,V_2)\, P(V_1|V_0) \sum_{v_0} P(V_6|v_0,V_1,V_2,V_3,V_4,V_5) \times P(V_4|v_0,V_1,V_2,V_3)\, P(V_2|v_0,V_1)\, P(v_0). \quad (4)$$

Unfortunately, in large models the expression elements quickly become unwieldy; Example 1 involves estimating conditional distribution terms over the entire joint configuration space of the model. In terms of scalability, this presents several practical problems as model sizes increase. The required distributions have exponentially many configurations, suggesting they may require a significant amount of data to estimate accurately. Moreover, the required marginalizations are over high-dimensional spaces, potentially making them computationally intractable. These issues make it difficult to apply statistically sophisticated or machine learning-based estimators [13, 14, 15, 16] in such settings.

On the other hand, we can often maintain tractability by using the simple *plug-in* estimator, in which each term is estimated only on the configurations seen in the observed data. This reduces computation, since each term is non-zero on only a subset of configurations, and summations can also be performed over only the non-zero configurations. However, this approach also limits the quality of our estimates – our plug-in estimates may have high variance, especially in settings with very large probability tables, and it is difficult to apply regularization or other variance-reduction techniques without destroying the sparsity property that makes it computationally feasible.

Treewidth. The treewidth of a graph (also known as induced-width) characterizes how close is the graph to a tree structure. If the treewidth of a graph is w, the graph can be embedded in a tree (or a hypertree) of clusters whose sizes are bounded by w. Exact PGM inference is exponential in the treewidth of their graphs. For more formal definition and related properties see [8, 17, 9].

The above described issues on the estimand's computational and estimation challenges, motivate us to explore the effectiveness of a *model completion* approach, in which we directly learn the Causal Bayesan Network model, including its latent variables, and then use the resulting learned model to evaluate the query. A common technique for learning latent variable models is the EM algorithm.

Expectation Maximization (EM). EM is a well-studied scheme to learn Bayesian network parameters when the graph itself is known but there are some latent variables, for which data are missing [10, 18]. EM is an iterative maximum likelihood approach that alternates between an Expectation, or E-step, and a Maximization, or M-step; see [21] for details. The E-step entails inference in the Bayesian network, which can be computationally hard in general; but when the graph has bounded *treewidth*, the E-step can be performed using join tree or bucket elimination algorithms [21, 9]. Otherwise, approximate algorithms such as belief propagation can be used [24, 22]. While EM has also been applied to structure learning [11], in our setting the structure is assumed known via the provided causal diagram.

Other related work. Motivated by similar model completion ideas, [7, 5] first convert the causal diagram to a circuit, exploiting the functional form of SCM mechanisms to yield compact circuits, then learn the circuit parameters via EM, and answer the causal queries. However, their work is restricted to binary-valued variables and, to the best of our knowledge, no empirical evaluation is provided. Another related line of work explores neural network approaches for causal inference [34, 35]. In that work the authors focus on *counterfactual* queries and propose learning a Neural Causal Model (NCM) whose functions are neural networks, a task which requires learning the mechanisms of the underlying SCM, which is significantly more challenging and unnecessary for our task of answering causal-effect queries. Moreover, while the authors provide a theoretical analysis, empirical validation is only performed over very small synthetic networks. EM has also been applied to causality in [30, 37]; in the latter, the authors assume knowledge of the functional mechanisms in the SCM, with only the distributions of the discrete latent variables unknown. In contrast, our method assumes knowledge of the graph only.

3 Learning-Based Causal Inference

The approach we propose to answer the causal-effect query is to first learn a full CBN $\mathcal{B} = \langle \mathcal{G}, \mathcal{P} \rangle$ from the given causal diagram $\mathcal{G} = \langle \boldsymbol{V} \cup \boldsymbol{U}, E \rangle$ and from samples from the observational distribution $P(\boldsymbol{V})$. Then, we can answer causal-effect queries from the full model, employing the appropriate truncation Eq. (3), via probabilistic inference algorithms over the learned CBN. However, there could be many parameterizations $\mathcal{P}$ that are consistent with the same observational distribution $P(\boldsymbol{V})$. Luckily, as long as the query is identifiable, any of these alternative parameterizations $\mathcal{P}$ will generate the same answer:

Theorem 1. *Assume a given SCM $\mathcal{M} = \langle \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{F}, P(\boldsymbol{U}) \rangle$ having causal diagram $\mathcal{G}$ and observational distribution $P(\boldsymbol{V})$. Any CBN $\mathcal{B} = (\mathcal{G}, \mathcal{P})$ inducing the same observational distribution $P(\boldsymbol{V})$ via Eq. (2) will agree with $\mathcal{M}$ on any identifiable causal-effect query $P(\boldsymbol{Y} \mid do(\boldsymbol{X} = \boldsymbol{x}))$.*

Proof. Follows directly from the definition of identifiability. □

Consequently, our problem reduces to the well-studied task of estimating the parameters of a Bayesian network from the network structure when given data on a subset of latent variables is not available. A widely used algorithm for this purpose is Expectation-Maximization (EM), which aims to maximize the likelihood of the data. To apply EM, however, we also require knowledge of the latent variables' distributional forms.

Usefully, in the case of discrete visible variables $\boldsymbol{V}$, prior work has shown that any SCM can be transformed into an equivalent SCM in which the latent variables $\boldsymbol{U}$ take on discrete, finite domains [38] and it provides an upper bound on the required latent domain sizes. We will therefore assume a discrete distribution for the U_j.

However, the upper bound in [38] is conservative, and often unnecessarily large. So, we decided to use the discrete latent domain sizes as a complexity control mechanism, to impose an additional degree of regularity on the probability distribution over the visible variables. For regularization, we treat the domain sizes, $\boldsymbol{k} = \{k_{U_i}\}$, as hyperparameters and select their values by minimizing the model's BIC score, which penalizes models with larger domain sizes for their increased flexibility and potential over-fitting [6]:

$$BIC_{\mathcal{B},\mathcal{D}} = -2 \cdot LL_{\mathcal{B},\mathcal{D}} + p \cdot \log(|\mathcal{D}|) \quad (5)$$

where $\mathcal{D}$ are the data samples from $P(\boldsymbol{V})$, $LL_{\mathcal{B},\mathcal{D}}$ is the log likelihood of the CBN model $\mathcal{B}$ learned via EM, and p is the number of free probability parameters, denoted $\boldsymbol{\theta}$, in the Bayesian network $\mathcal{B}$.

Algorithm 1: EM4CI

input : A causal diagram $\mathcal{G} = \langle \boldsymbol{U} \cup \boldsymbol{V}, E \rangle$, $\boldsymbol{U}$ latent and $\boldsymbol{V}$ observable; samples $\mathcal{D}$ from $P(\boldsymbol{V})$;
Identifiable Query $Q = P(\boldsymbol{Y} \mid do(\boldsymbol{X} = \boldsymbol{x}))$.

output : Estimated $P(\boldsymbol{Y} \mid do(\boldsymbol{X} = \boldsymbol{x}))$

// k= latent domain size, $BIC_{\mathcal{B}} \equiv BIC_{\mathcal{B},\mathcal{D}}$ the BIC score of
// CBN $\mathcal{B}$ & $\mathcal{D}$, $LL_{\mathcal{B}} \equiv LL_{\mathcal{B},\mathcal{D}}$ the log-likelihood of $\mathcal{B}$ & $\mathcal{D}$

1. Initialize: $BIC_{\mathcal{B}} \leftarrow \infty$
2. **for** $k = 2, ...,$ to upper bound, **do**
3. Initialize: $LL_{\mathcal{B}'} \leftarrow 0$
4. **for** $i = 0 \ldots 9$ **do** (optimize LL of EM)
5. Initialize: θ_i to random parameters of $\mathcal{G}$
6. $(LL_{\mathcal{B}_i}, \mathcal{B}_i) \leftarrow \text{EM}(\mathcal{G}, \mathcal{D}, k, \theta_i)$
7. **if** $LL_{\mathcal{B}_i} > LL_{\mathcal{B}'}$
8. $LL_{\mathcal{B}'} \leftarrow LL_{\mathcal{B}_i}, \mathcal{B}' \leftarrow \mathcal{B}_i$
9. **end for**
10. Calculate $BIC_{\mathcal{B}'}$ from $LL_{\mathcal{B}'}$
11. **if** $BIC_{\mathcal{B}'} \leq BIC_{\mathcal{B}}$,
12. $\mathcal{B} \leftarrow \mathcal{B}', BIC_{\mathcal{B}} \leftarrow BIC_{\mathcal{B}'}$
13. **else**, break.
14. **endfor**
15. $\mathcal{B}_{\boldsymbol{X}=\boldsymbol{x}} \leftarrow$ truncated CBN from $\mathcal{B}$.
16. **return** $P_{\mathcal{B}_{\boldsymbol{X}=\boldsymbol{x}}}(\boldsymbol{Y})$ computed by any PGM algorithm

To optimize our CBN over both the domain sizes $\boldsymbol{k}$ and probability parameters $\boldsymbol{\theta}$, we propose a practical algorithm that searches greedily in the space of $\boldsymbol{k}$ while optimizing $\boldsymbol{\theta}$ by EM. We prioritize searching the model space starting from smaller domain sizes since simpler models are preferable (i.e., Occam Razor). We adopt a simple strategy of keeping all latent domain sizes equal, i.e., set all $k_{U_i} = k$, and gradually increase the value k.

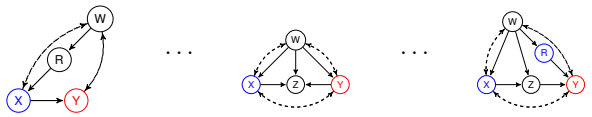
Our learning strategy benefits from two sources of variance reduction compared to the simple plug-in estimates: first, from the graph structure, which imposes some regularity on $P(\boldsymbol{V})$; and second, by preferring smaller latent domain sizes when possible, which encourages learning lower-rank distributions when supported by the data.

Model-completion rationale. Given any causal effect query $P(\boldsymbol{Y} \mid do(\boldsymbol{X} = \boldsymbol{x}))$ defined relative to causal graph $\mathcal{G}$ and given a full CBN $\mathcal{B} = \langle \mathcal{G}, \mathcal{P} \rangle$ learned from observational data, Theorem 1 suggests that $P(\boldsymbol{Y} \mid do(\boldsymbol{X} = \boldsymbol{x}))$ can be estimated as $P(\boldsymbol{Y})$ in the truncated learned CBN $\mathcal{B}_{\boldsymbol{X}=\boldsymbol{x}}$, i.e.,

$$P(\boldsymbol{Y} \mid do(\boldsymbol{X} = \boldsymbol{x})) \approx P_{\mathcal{B}}(\boldsymbol{Y} \mid do(\boldsymbol{X} = \boldsymbol{x})) = P_{B_{\boldsymbol{X}=\boldsymbol{x}}}(\boldsymbol{Y}). \quad (6)$$

where $\mathcal{B}_{\boldsymbol{X}=\boldsymbol{x}} = \langle \mathcal{G}_{\overline{\boldsymbol{X}}}, \mathcal{P}_{\boldsymbol{X}=\boldsymbol{x}} \rangle$ is the truncated CBN and $\mathcal{G}_{\overline{\boldsymbol{X}}}$ is the graph obtained by deleting all incoming arrows to $\boldsymbol{X}$ in $\mathcal{G}$.

The iterative learning EM4CI algorithm (Algorithm 1). Since we use EM for learning we call our algorithm *EM for Causal Inference (EM4CI)*, described in Algorithm 1. Its input is a causal diagram $\mathcal{G}$, samples $\mathcal{D}$ from $P(\boldsymbol{V})$, and an identifiable query $Q = P(\boldsymbol{Y} \mid do(\boldsymbol{X} = \boldsymbol{x}))$. *Steps 2-14* provide the iterative learning scheme. Given samples $\mathcal{D}$ from $P(\boldsymbol{V})$, we use EM to learn a CBN $\mathcal{B}$ over the graph $\mathcal{G}$ with latent domain size k. EM outputs the parameters for a network denoted $\mathcal{B}_i$, and its corresponding log-likelihood, $LL_{\mathcal{B}_i}$. Since EM's performance is known to be sensitive to initialization, we perform ten runs of EM starting from randomly generated initial parameters, and retain the model, $\mathcal{B}'$, with the highest likelihood, $LL_{\mathcal{B}'}$, as the candidate for the current latent domain size k (*steps 4-9*), with which



(a) Model 1 **(b)** Model 3 **(c)** Model 8

Figure 2: A subset of the small causal diagrams for models used in our experiments.

Table 1: Estimand Expressions for Models 1-8 in Figure 5 & Table 3.

Model	Estimate of $P(Y \mid do(X))$
1	$\frac{\sum_W P(X,Y\|R,W)P(W)}{\sum_W P(X\|R,W)P(W)}$
2	$\sum_R P(R\|X_1) \sum_{x_1,Z} P(Y\|R,x_1,X_2,Z)P(Z\|R,x_1)P(x_1)$
3	$P(Y)$
4	$\sum_{R,S} P(S\|X_1,X_2,X_3,Z)P(R\|X_1) \sum_{x_1,x_3,Z} P(Y\|R,x_1,X_2,x_3,Z)P(x_3\|x_1,X_2,Z)P(x_1,Z)$
5	$\sum_{Z_1,Z_2,Z_3} P(Z_3\|Z_2)P(Z_1\|X,Z_2) \frac{\sum_x P(Y,Z_3\|x,Z_1,Z_2)P(x,Z_2)}{\sum_x P(Z_3\|x,Z_1,Z_2)P(x,Z_2)} P(Z_2)$
6	$\sum_{Z_2} P(Y\|X_1,X_2,Z_2)P(Z_2)$
7	$\sum_{Z_2,Z_3} P(Y\|X_1,X_2,Z_2,Z_3)P(Z_2,Z_3)$
8	$\sum_{R,W,Z} P(Z\|R,W,X)P(R\|W) \sum_x P(Y\|R,W,x,Z)P(x\|R,W)P(W)$

we calculate the $BIC_{\mathcal{B}'}$ score (*step 10*). If the BIC score decreases (*steps 11-13*), we adopt the new candidate network as $\mathcal{B}$, and the process continues with increased k; otherwise the current $\mathcal{B}$ is selected as the final learned network. *Step 15* generates the truncated CBN $\mathcal{B}_{\boldsymbol{X}=\boldsymbol{x}}$, and *step 16* employs a standard *PGM* inference algorithm, like join-tree decomposition [6, 9], to the CBN $\mathcal{B}_{\boldsymbol{X}=\boldsymbol{x}}$ to compute $P(\boldsymbol{Y})$.

Complexity of EM4CI. The complexity of EM (*step 6*) depends on the sample size $|D|$, the variables' domain sizes, and the number of iterations needed for convergence which is hard to predict, but complexity-wise provides only a constant factor and can be ignored. *Steps 2-14* repeat the EM algorithm, which again adds only a constant factor. The most extensive computation in each iteration of EM is the Expectation step, which requires probabilistic inference and can be accomplished in time and memory exponential in the treewidth of the graph. Otherwise, approximation algorithms such as belief propagation may be used. In our experiments, exact inference was always possible. Thus, each iteration of EM takes $O(|D| \cdot nl^w)$, where $n = |V \cup U|$ is the number of variables, l bounds the variable domain sizes, and w is the treewidth. Thus for T iterations we have $O(T \cdot |D| \cdot n \cdot l^w)$. *Step 16* of probabilistic inference is $O(n \cdot l^w)$ with exact inference. In summary, the complexity of algorithm *EM4CI* is $O(T \cdot |D| \cdot n \cdot l^w)$. Thus, the algorithm is most effective for models with bounded treewidth. Note that the cost of the EM learning process can be amortized over multiple queries. In summary,

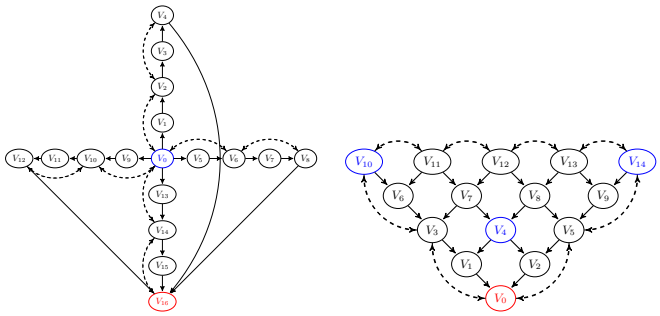
Theorem 2 (complexity of EM4CI)**.** *The complexity of algorithm* EM4CI *is* $O(T \cdot m \cdot n \cdot l^w)$*, where* m *is the number of samples,* T *is the number of iterations,* k *bounds the domain size,* n *is the total number of variables, and* w *is the treewidth of the causal diagram.*

4 Empirical Evaluation

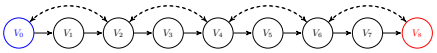
We perform an extensive empirical evaluation of EM4CI and compare against estimand-based approaches such as the brute-force empirical *plug-in* method and a state-of-the-art scheme called weighted empirical risk minimization (WERM) [14]. All experiments were run on a 2.66 GHz processor with 8 GB of memory. All experimental code can be found here [26].

4.1 Benchmarks.

The inputs to a causal inference algorithm are a triplet: 1) a causal diagram (of an underlying SCM), 2) data from the model's observational



(a) Diamond, $n = 17$ (17-D) **(b)** Cone Cloud, $n = 15$ (15-CC)



(c) Chain, $n = 9$ (9-CH)

Figure 3: Larger synthetic models.

distribution, and 3) a query $Q = P(\boldsymbol{Y} \mid do(\boldsymbol{X}))$. We use two sources for our benchmarks: synthetically generated scalable model classes, and real domains from the academic literature of various fields with causal interpretations [1].

Synthetic networks. We generate each triplet benchmark instance by first choosing a causal diagram and a query. Then, picking domain sizes of observed and latent variables, we generate the conditional probability tables (CPTs), one per variable and its parents, to yield the full CBN. From this full model we generate samples by forward sampling [6] and then discard the values of the latent variables, yielding samples from the observed distribution. We compute the exact answer to the target query on the full CBN via an exact algorithm (e.g., join-tree [6, 9]).

The causal diagrams were selected in two ways. First, we use a set of 8 small diagrams from the causality literature [25, 15]; see Figure 5 and the Supplemental [27]. Second, we examine three scalable classes of graphs whose treewidths vary but can be controlled. These are: chain networks (treewidth 3; Figure 3c), diamond networks (treewidth 5; Figure 3a), and cone-cloud networks (treewidth $O(\sqrt{n})$, Figure 3b), abbreviated CH, D, CC respectively.

In the small diagrams (Figure 5), we set the domain size of the observed variables to $d = 2$, and of the latent variables to $k = 10$. For the chain, diamond, and cone-cloud models, we use $(d, k) = (4, 10)$. The parameters of each CPT were generated by sampling from a Dirichlet distribution [6] with parameters $\alpha = [\alpha_1, \ldots, \alpha_n]$, where $\alpha_i \in [0, 1)$ selected uniformly at random. The aim is to generate conditional probabilities near the edges of the simplex (i.e., far from uniform).

Use-case benchmarks. We also experimented with four networks created for real-world domains. The "Alarm" network was developed for on-line monitoring of patients in intensive care units [2]; the "A" network, from the UAI literature, is synthetic but known to be daunting to exact algorithms [19]; the "Barley" network was built for a decision support system for growing barley without pesticides [20]; and the "Win95" model is an expert system for printer troubleshooting in Windows 95. Since no variables in these networks are specified as latent, we designated source vertices with at least two children as latent confounding variables. As in the synthetic networks case, we generated data by forward sampling, subsequently discarding the latent variables' values.

Queries. We hand selected multiple identifiable queries per model, aiming for non-trivial ones and prioritizing those that appear to be complex. We provide the queries and their corresponding estimands in the Supplemental [27].

Table 2: Algorithms and Packages

Algorithm	Use	Software Package	Programming Language
ID Algorithm	Plug-In & EM4CI	*causaleffect*	R
Join-tree Decomposition	EM4CI inference	SMILE	C++
EM Algorithm	EM4CI learning	SMILE	C++
Sparse Plug-In Evaluation	Plug-In		Python
WERM			R

Table 3: Results of EM4CI & Plug-In on $P(Y|do(\mathbf{X}))$ $(d, k) = (2, 10)$

	100 Samples					1,000 Samples				
	EM4CI			Plug-In		EM4CI			Plug-In	
Model	k_{lrn}	mad	time(s)	mad	time(s)	k_{lrn}	mad	time(s)	mad	times(s)
1	2	0.0037	0.48	0.0104	1.9	2	0.0032	3.1	0.0025	2.3
2	2	0.1832	1.86	0.1436	2.3	2	0.0490	8.4	0.0867	2.0
3	2	0.1288	0.93	0.0569	1.1	2	0.0040	3.6	0.0039	0.7
4	2	0.1819	1.82	0.1469	2.3	2	0.1438	12.0	0.0704	2.1
5	2	0.4910	1.65	0.5000	2.0	2	0.0044	17.3	0.0058	2.2
6	2	0.2663	0.30	0.3930	2.0	2	0.1263	0.5	0.1319	2.1
7	2	0.2520	0.78	0.2509	1.9	2	0.0891	7.1	0.0238	2.0
8	2	0.1372	0.63	0.1579	2.0	2	0.2340	4.7	0.1303	1.9

4.2 *Algorithms and performance measures.*

EM4CI. In our EM4CI algorithm we used the EM implementation of SMILE: Structural Modeling, Inference, and Learning Engine package [1], written in C++. In SMILE, inference is carried out via join trees [24, 21], and used both in the E step of the EM algorithm [21] and for answering queries over the learned model.

Estimand-based algorithms: Plug-In and WERM. We used the ID algorithm in the causal effect package [33] to compute the estimands. Subsequently, the plug-in method evaluates the estimand from the observational data, by plugging in the empirical conditional probabilities. Our plug-in implementation uses sparse table representations in Python, to ensure that the representation size of each estimated probability term is linear in the data size, rather than exponential in the number of variables. We also compare against a state-of-the-art scheme called WERM [14] using the author-provided implementation in R. A summary of the algorithms and implementations is given in Table 2.

Measures of performance. We report the results of EM4CI along the two phases of the algorithm; the *learning phase* (steps 2-14) done via EM and the *inference phase* (steps 15-16) done by the join tree algorithm. For the learning process, we report the selected latent domain sizes and the total time at termination. For the inference phase we report the *mean absolute deviation (mad)* between the true answer and the estimated answer. The measure mad for a query $P(Y \mid do(\boldsymbol{X}))$ is computed by averaging the absolute error over all instantiations of the intervened and queried variables, $P(Y = y \mid do(\boldsymbol{X} = \boldsymbol{x}))$ for $\boldsymbol{x}, y \in \mathcal{D}(\boldsymbol{X}) \times \mathcal{D}(Y)$. We also report inference time for each query. We incremented k by steps of two ($k = 2, 4, 6, 8, 10 \ldots$) in the EM4CI algorithm.

4.3 *Results*

Results on small synthetic models. Results on models 1-8 (Figure 5) are presented in Table 3. Since these models are quite small we report the total time (learning plus inference) for EM4CI. We compare against the *Plug-In* estimand method (see estimand expressions in Table 1), which is guaranteed to converge to the exact answer. Therefore, we expect *Plug-In* to produce fairly accurate results on these small models if given enough samples. We observe that the accuracy of both methods are similar at both 100 and 1,000 samples, with EM4CI being more accurate on some cases, and *Plug-In* on others. EM4CI

Table 4: Results of absolute error on Query $P(Y = 1|do(\mathbf{X} = 1))$ on Models 1, 8, & 3' by WERM & EM4CI. k_{lrn} is the learned domain sizes of latent variables.

	1,000 Samples					10,000 Samples				
	WERM		EM4CI			WERM		EM4CI		
Model	error	time(s)	error	time(s)	k_{lrn}	error	time(s)	error	time(s)	k_{lrn}
1	0.0071	18.7	0.0059	8.8	2	0.0031	32.6	0.0046	63.5	2
8	0.1082	25.8	0.1566	7.6	2	0.11	47.7	0.0001	81.4	2
3'	0.027	27.2	0.0004	3.5	2	0.001	44.1	0.0009	53.1	2

Table 5: Results for EM4CI and Plug-In on $P(Y|do(\mathbf{X}))$ $(d, k) = (4, 10)$

(a) 1,000 samples

			EM4CI			Plug-In	
Model	Query	k_{lrn}	mad	learn-time(s)	inf-time(s)	mad	time(s)
5-CH	$P(V_4\|do(V_0))$	4	0.0902	3.5	0.0001	0.1509	2.3
9-CH	$P(V_8\|do(V_0))$	4	0.1204	11.5	0.0002	0.1516	2.4
25-CH	$P(V_{24}\|do(V_0))$	2	0.0070	77.7	0.0003	0.0959	6.1
49-CH	$P(V_{48}\|do(V_0))$	4	0.0005	161.2	0.0007	0.0319	17.8
99-CH	$P(V_{98}\|do(V_0))$	6	0.0093	413.4	0.0023	0.0611	88.1
9-D	$P(V_8\|do(V_0))$	2	0.0719	24.6	0.0002	0.1832	3.4
17-D	$P(V_{16}\|do(V_0))$	6	0.0542	202.3	0.0006	0.0700	4.5
65-D	$P(V_{64}\|do(V_0))$	4	0.0074	432.4	0.0012	0.1716	232.5
6-CC	$P(V_0\|do(V_5))$	4	0.0088	23.5	0.0001	0.0156	2.3
15-CC	$P(V_0\|do(V_{14}))$	4	0.0147	60.8	0.0001	0.0659	4.5
45-CC	$P(V_0\|do(V_{14}, V_{36}, V_{44}))$	6	0.0097	199.2	2.7429	0.1509	18.6

(b) 10,000 samples

			EM4CI			Plug-In	
Model	Query	k_{lrn}	mad	learn-time(s)	inf-time(s)	mad	time(s)
5-CH	$P(V_4\|do(V_0))$	4	0.0508	17.3	0.0001	0.0537	2.5
9-CH	$P(V_8\|do(V_0))$	4	0.0236	150.0	0.0002	0.1074	3.1
25-CH	$P(V_{24}\|do(V_0))$	6	0.0068	697.1	0.0005	0.0714	26.4
49-CH	$P(V_{48}\|do(V_0))$	10	0.0017	2412.6	0.0036	0.0160	133.7
99-CH	$P(V_{98}\|do(V_0))$	6	0.0028	3887.9	0.0022	0.0433	850.6
9-D	$P(V_8\|do(V_0))$	4	0.0611	390.7	0.0002	0.1481	3.0
17-D	$P(V_{16}\|do(V_0))$	6	0.0360	1849.6	0.0007	0.0582	8.4
65-D	$P(V_{64}\|do(V_0))$	4	0.0022	4787.2	0.0013	0.1376	2258.5
6-CC	$P(V_0\|do(V_5))$	6	0.0138	116.9	0.0003	0.0136	2.7
15-CC	$P(V_0\|do(V_{14}))$	4	0.0022	489.5	0.0043	0.0321	10.9
45-CC	$P(V_0\|do(V_{14}, V_{36}, V_{44}))$	6	0.0026	1833.7	2.757	0.1561	105.8

had better time performance for 100 samples, but for 1,000 samples the *Plug-In* due to the learning time of EM4CI.

WERM comparison. The results for Models 1, 8, and 3' [1] comparing WERM [14] to EM4CI are given in Table 4. We use domain size $d = 2$, and 1,000 and 10,000 samples. Again, EM4CI produced more accurate results in several instances, though the disparities are smaller than with the Plug-In method. EM4CI was faster than WERM with 1,000 samples but slower with 10,000 samples. Unfortunately, the available code for WERM is specific to these models, so we were unable to compare against it more generally.

Results on large synthetic models. In Tables 5 and 6 we show results on larger models of chains, diamonds, and cone-cloud graphs. The first two tables compare EM4CI to the plug-in method for a single query over all the models. Specifically, Table 5a presents time and accuracy results for 1,000 samples. We see that EM4CI was consistently more accurate, and in many cases significantly better (e.g., in 45-cone-cloud). Table 5b shows the same trend for 10,000 samples. The superior accuracy of learning for model completion over *Plug-In* is clearly visible in Figure 4, which shows the accuracy trends for each class as a function of model sizes, for both 1,000 and 10,000 samples. We see that generally, EM4CI using only 1,000 samples is even more accurate than *Plug-In* with 10,000 samples.

Again, we expect that this improvement is due to the variance reduction of the estimation process. By its nature, model completion exploits more information from the causal graph than is apparent in the estimand expression alone, implicitly capturing the known conditional independence structure in its estimates of $P(\boldsymbol{V})$. We can also include simple and easy-to-impose complexity control, in the form of the latent domain sizes, to further reduce variance. In contrast, it is difficult to impose meaningful regularity or variance reduction on the plug-in estimates of $P(\boldsymbol{V})$ without creating significant computational burdens. Taking it all together, our results suggest that whenever the causal graph's treewidth is bounded and the estimand expression has large scope functions, we should prefer using model completion. Other settings may require more study to determine the best approach.

[1] model 3' is the same as Model 3 in Figure 5, but with edge $Y \to Z$ reversed to match the hard-coded model in WERM.

Table 6: EM4CI & Plug-In on the 45 Cone Cloud
$|\boldsymbol{V}| = 45$; $|\boldsymbol{U}| = 16$; $d = 4$; $k = 10$; treewidth ≈ 11

(a) Plug-In

	1,000 Samples		*10,000 Samples*	
Query	mad	time(s)	mad	time(s)
$P(V_0\|do(V_{14}, V_{36}, V_{44}))$	0.1510	18.6	0.1561	105.8
$P(V_0\|do(V_{12}, V_{29}, V_{34}))$	0.1744	19.0	0.1043	101.0
$P(V_0\|do(V_{10}, V_{14}, V_{40}))$	0.1906	18.4	0.1201	131.4
$P(V_4\|do(V_{15}, V_{20}, V_{40}))$	0.2080	26.2	0.1282	121.6

(b) EM4CI

	1,000 Samples		*10,000 Samples*	
Learning:	time(s) = 199	$k_{lrn} = 6$	time(s)= 1833	$k_{lrn} = 6$
Inference: Query	mad	time(s)	mad	time(s)
$P(V_0\|do(V_{14}, V_{36}, V_{44}))$	0.0097	2.77	0.0026	2.76
$P(V_0\|do(V_{12}, V_{29}, V_{34}))$	0.0118	18.99	0.0024	18.99
$P(V_0\|do(V_{10}, V_{14}, V_{40}))$	0.0097	0.06	0.0029	0.06
$P(V_4\|do(V_{15}, V_{20}, V_{40}))$	0.0082	0.02	0.0013	0.02

Focusing on time, we see that the time of EM4CI is significantly larger than *Plug-In*, with EM learning being the most time consuming part. Interestingly, while time grows with model size for both schemes, inference time remains efficient, likely due to the low treewidth of some of the models (e.g., the chains and the diamonds). In the 45-cone case, inference is impacted more by model size, since its treewidth increases with the square root of the number of variables. Generally, for a single query, we find that the *Plug-In* is superior time-wise, and its time increases at a slower pace with model size. Finally, in both methods, time increases with sample size as expected.

Results on real-world data sets. Table 7 compares EM4CI against *Plug-In* on a single query for all 4 networks. We observe the same pattern of performance as in the synthetic networks: EM4CI provides far more accurate results on all real-world models but with some time cost, attributed to learning. Evaluation for multiple queries are given in Table 8 for the **A network**. We find that the same latent domain size ($k_{lrn} = 4$) is selected for both sample sizes. As before, learning time grows with sample size and accuracy results are excellent and are improving with increased sample sizes. Note, however, that increased sample size does not impact inference time. Results for **Alarm**, **Win95**, and **Barley** networks are similar and can be found in the supplemental [27].

Answering multiple queries. Tables 6b and 8b highlight how learning time can be amortized effectively over multiple queries. The table

Table 7: Results on EM4CI & Plug-In on Real Networks

(a) 1,000 Samples

			EM4CI			Plug-In	
Model	Query	k_{lrn}	mad	learn-time(s)	inf-time(s)	mad	time(s)
A	$P(V_{51}\|do(V_{10}))$	4	0.0139	71	0.0012	0.0584	8
Alarm	$P(HRBP\|Shunt))$	2	0.0076	15.7	0.0002	0.0190	2.9
Barley	$P(protein\|do(expYield))$	14	0.0066	199	0.9038	0.0687	9.9
Win95	$P(Output\|do(NnPsGrphc))$	2	0.0113	109.1	0.0002	0.2674	1.5

(b) 10,000 samples

			EM4CI			Plug-In	
Model	Query	k_{lrn}	mad	learn-time(s)	inf-time(s)	mad	time(s)
A	$P(V_{51}\|do(V_{10}))$	4	0.0083	540.6	0.0012	0.0114	55.7
Alarm	$P(HRBP\|Shunt))$	4	0.0043	181.3	0.0002	0.0075	5.7
Barley	$P(protein\|do(expYield))$	10	0.0031	819.5	0.8882	0.0655	63.4
Win95	$P(Output\|do(NnPsGrphc))$	4	0.0008	1080.8	0.0002	0.2894	2.0

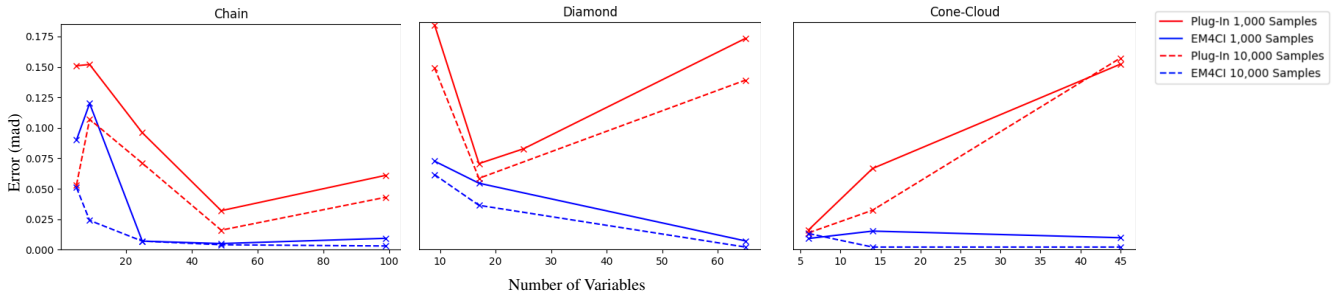


Figure 4: Comparing the accuracy trends of EM4CI and Plug-In as a function of the model size. While both methods improve with more samples (solid to dashed lines), the error (mad) of EM4CI is smaller, even when compared to Plug-In with more samples.

presents EM4CI's learning and inference time separately on the multiple queries. In contrast, the *Plug-In* method requires estimation of each new query from scratch, even if on the same model; see Table 6a and 8a. Thus, for multiple queries, EM4CI may take far less time per query, while providing superior quality estimates. (See additional results on multiple queries in the Supplemental [27].)

Summary. Our experiments illustrate that model-completion by learning implemented in EM4CI yields highly accurate estimates of causal effect queries on a variety of benchmarks, including both synthetic and real-world networks of varying sizes. Moreover EM4CI shows clear superiority to the Plug-In estimands approach. In terms of time efficiency however, EM4CI was consistently slower due to learning time overhead. Yet, when answering multiple queries is desired the time overhead can be amortized over multiple queries.

5 Conclusion

In causal inference, the estimand-based approach has become standard: generating a potentially complex expression in terms of the observable distribution, and then estimating the required probabilities quantities from the data and evaluating the resulting expression. While mathematically correct, this approach tends to ignore the difficulties inherent in estimating the complex, conditional probabilities required, and the computationally challenge in evaluating the resulting expression. These difficulties become increasingly apparent as model size increases.

Table 8: Plug-In & EM4CI results on the **A Network**
$|\boldsymbol{V}| = 46;\ |\boldsymbol{U}| = 8;\ d = 2;\ k = 2$, treewidth ≈ 16

(a) Plug-In

	1,000 Samples		*10,000 Samples*	
Query	mad	time (s)	mad	time (s)
$P(V_{51}\|\text{do}(V_{10}))$	0.0584	8.0	0.0114	55.7
$P(V_{51}\|\text{do}(V_{14}))$	0.0319	8.3	0.0056	51.3
$P(V_{51}\|\text{do}(V_{41}))$	0.0255	13.9	0.0092	48.3
$P(V_{51}\|\text{do}(V_{45}))$	0.0496	9.8	0.0206	49.1

(b) EM4CI

	1,000 Samples		*10,000 Samples*	
Learning	$time(s) = 71$	$k_{lrn} = 4$	$time(s) = 541$	$k_{lrn} = 4$
Inference: Query	mad	time (s)	mad	time (s)
$P(V_{51}\|\text{do}(V_{10}))$	0.0139	0.0012	0.0083	0.0012
$P(V_{51}\|\text{do}(V_{14}))$	0.0143	0.0047	0.0086	0.0046
$P(V_{51}\|\text{do}(V_{41}))$	0.0147	0.0042	0.0079	0.0041
$P(V_{51}\|\text{do}(V_{45}))$	0.0140	0.0031	0.0082	0.0030

An alternative path, relatively less explored, is to leverage the causal model structure more directly via learning. By performing model completion – i.e., learning a Causal Belief Network, including its latent confounding variables, from the observed data – we exploit additional information about the co-dependence structure in the distribution, and can more easily apply complexity control and variance reduction strategies to the parameter estimation process. Then, once an approximate full model is available, traditional computationally efficient PGM algorithms can be applied to answer the query, either exactly or approximately.

Our algorithm EM4CI uses the well-known EM algorithm to learn the model and its latent variable distributions, then uses tree-decomposition algorithms to answer the queries.

We carried out an extensive empirical evaluation, the first of its scale in the causal community, over a collection incorporating both synthetic networks and real world distributions. We evaluated and compared EM4CI's performance to the Plug-In estimand approach, in terms of both accuracy and time.

Our results appear conclusive: we show clearly that model completion using EM4CI yields consistently superior results compared to the estimand Plug-In. This benefit does come with a cost in time from the learning phase, which grows more quickly with network size and number of samples compared to the Plug-In approach. However, learning time can be amortized over multiple queries on the same model, making a collection of queries significantly more efficient.

Our EM4CI approach relies in part on the efficiency of inference; when the treewidth of the graph is bounded, both learning and inference using EM4CI are likely to be effective. Additionally, since the structure of the graph is better exploited by model completion, we reduce the variance of our estimators, resulting in better performance for a given dataset size. Although in some cases, the estimand approach may generate a simple and easy-to-estimate expression, in larger models with many latent confounders the expression is often complex. In such settings, the estimand expression loses structural information, forcing us to estimate high-dimensional probabilities and making it difficult to apply variance reduction strategies without creating a computationally difficult evaluation problem. Overall this suggests that model completion should be considered a competitive strategy for causal estimation.

6 Acknowledgments

Thank you to the reviewers for their valuable comments and suggestions. This work was supported in part by NSF grants IIS-2008516 and CNS-2321786.

References

[1] BayesFusion LLC. *QGeNIe Modeler USER MANUAL*, 2022.

[2] I. Beinlich, H. Suermondt, R. Chavez, and G. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proceedings of the Second European Conference on Artificial Intelligence in Medical Care*, pages 247–256. Springer-Verlag, Berlin, 1989.

[3] R. Bhattacharya, R. Nabi, and I. Shpitser. Semiparametric inference for causal effects in graphical models with hidden variables. *Journal of Machine Learning Research*, 23:1–76, 2022.

[4] A. Bhattacharyya, S. Gayen, S. Kandasamy, V. Raval, and V. N. Variyam. Efficient interventional distribution learning in the pac framework. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, pages 7531–7549, 2022.

[5] Y. Chen and A. Darwiche. On the definition and computation of causal treewidth. In J. Cussens and K. Zhang, editors, *Uncertainty in Artificial Intelligence, UAI, 2022*, 2022.

[6] A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

[7] A. Darwiche. Causal inference using tractable circuits. *CoRR*, 2022.

[8] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artif. Intell.*, 113(1-2):41–85, 1999.

[9] R. Dechter. *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013.

[10] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B.*, 39:1–38, 1977.

[11] N. Friedman. The Bayesian structural EM algorithm. In *Uncertainty in Artificial Intelligence (UAI)*, pages 129–138, 1998.

[12] Y. Huang and M. Valtorta. On the completeness of an identifiability algorithm for semi-Markovian models. *Annals of Mathematics and Artificial Intelligence*, 54(4):363–408, 2008.

[13] Y. Jung, J. Tian, and E. Bareinboim. Estimating causal effects using weighting-based estimators. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, pages 10186–10193, 2020.

[14] Y. Jung, J. Tian, and E. Bareinboim. Learning causal effects via weighted empirical risk minimization. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33*, 2020.

[15] Y. Jung, J. Tian, and E. Bareinboim. Estimating identifiable causal effects through double machine learning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*, pages 12113–12122, 2021.

[16] Y. Jung, J. Tian, and E. Bareinboim. Estimating identifiable causal effects on markov equivalence class through double machine learning. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021*, 2021.

[17] K. Kask, R. Dechter, J. Larrosa, and A. Dechter. Unifying tree decompositions for reasoning in graphical models. *Artif. Intell.*, 166 (1-2):165–193, 2005. doi: 10.1016/J.ARTINT.2005.04.004. URL https://doi.org/10.1016/j.artint.2005.04.004.

[18] N. Koller, Daphne; Friedman. *Probabilistic Graphical Models*. The MIT Press, 2009.

[19] A. V. Kozlov and J. P. Singh. Parallel implementations of probabilistic inference. *IEEE Computer*, pages 33–40, 1996.

[20] K. Kristensen and I. Rasmussen. The use of a Bayesian network in the design of a decision support system for growing malting barley without use of pesticides. *Computers and Electronics in Agriculture*, 33:197–217, 2002.

[21] S. L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics & Data Analysis*, pages 191–201, 1995.

[22] R. Mateescu, K. Kask, V. Gogate, and R. Dechter. Join-graph propagation algorithms. *J. Artif. Intell. Res.*, 37:279–328, 2010.

[23] Microsoft, Inc. win95: an expert system for trouble shooting, 1995.

[24] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1989.

[25] J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2nd edition, 2009.

[26] A. Raichev, A. Ihler, J. Tian, and R. Dechter. Em4ci: Causal inference using a model learning approach, 2024. URL https://github.com/anniemeow/EM4CI. Accessed: 2024-08-26.

[27] A. Raichev, J. Tian, A. Ihler, and R. Dechter. Estimating causal effects from learned causal networks. *arXiv preprint arXiv:2408.14101*, 2024. URL https://arxiv.org/abs/2408.14101.

[28] I. Shpitser and J. Pearl. Identification of joint interventional distributions in recursive semi-Markovian causal models. In *Proceedings of the 21st AAAI Conference on Artificial Intelligence*, page 1219, 2006.

[29] I. Shpitser and J. Pearl. Identification of conditional interventional distributions. In *Uncertainty in Artificial Intelligence*, pages 442–449, 2006.

[30] I. Shpitser, T. S. Richardson, and J. M. Robins. An efficient algorithm for computing interventional distributions in latent variable causal models. *CoRR*, abs/1202.3763, 2012.

[31] J. Tian. *Studies in Causal reasoning and Learning*. PhD thesis, University of California, Los Angeles, 2002.

[32] J. Tian and J. Pearl. A general identification condition for causal effects. In R. Dechter, M. J. Kearns, and R. S. Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*, pages 567–573, 2002.

[33] S. Tikka. *Package 'causaleffect'*. CRAN, 2022. URL https://github.com/santikka/causaleffect/. Version 1.3.15, October 12, 2022.

[34] K. Xia, K. Lee, Y. Bengio, and E. Bareinboim. The causal-neural connection: Expressiveness, learnability, and inference. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, editors, *NeurIPS 34*, pages 10823–10836, 2021.

[35] K. Xia, Y. Pan, and E. Bareinboim. Neural causal models for counterfactual identification and estimation. *CoRR*, abs/2210.00035, 2022.

[36] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Generalized belief propagation. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *NIPS*, 2000.

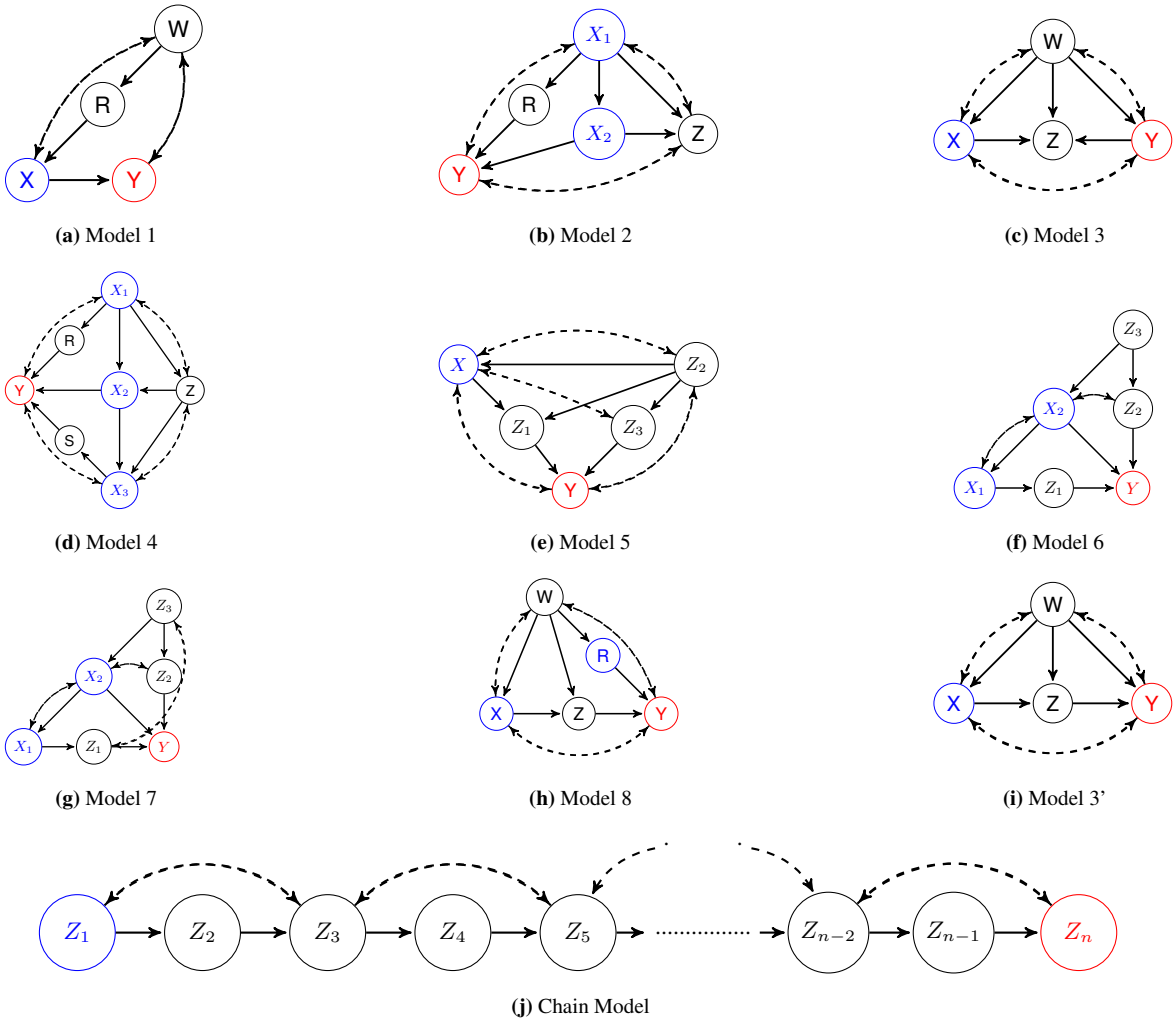
[37] M. Zaffalon, A. Antonucci, and R. Cabañas. EM based bounding of unidentifiable queries in structural causal models. *Why-21 workshop at NeurIPS*, 2021.

[38] J. Zhang, J. Tian, and E. Bareinboim. Partial counterfactual identification from observational and experimental data. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, editors, *International Conference on Machine Learning, ICML 2022*, volume 162 of *Proceedings of Machine Learning Research*, pages 26548–26558. PMLR, 2022.

A Supplementary Material

A.1 Models

Here we show the additional small models used in our experiments; see Figures 2 & 3 in the main text.



(a) Model 1 **(b)** Model 2 **(c)** Model 3

(d) Model 4 **(e)** Model 5 **(f)** Model 6

(g) Model 7 **(h)** Model 8 **(i)** Model 3'

(j) Chain Model

Figure 5: Causal diagrams for models used for experiments. Blue variables are intervened on and red variables are the outcome variables corresponding to the query $P(Y \mid do(X))$.

A.2 Additional Results

We provide several additional tables of results to complement those presented in the main text on larger synthetic and real networks. In Table 9a and 9b, we see the results of EM4CI on the **99-chain** and **65-diamond** on multiple queries. We see the same trend discussed in the paper, where learning time is longer than inference time, and grows with sample size. Again, the learned domain sizes are the same for both sample sizes, and close to the true domain size of the latent variables. We also see that inference is very fast and therefore we can amortize the learning time over multiple queries if desired. Finally, we see that EM4CI is very accurate.

In Table 10 we see results on the Plug-In method and EM4CI on the **Alarm network**. Again, we see that EM4CI is more accurate than the Plug-In, but the learning time is longer. However, for multiple queries, EM4CI may take less time per query, while providing superior quality estimates.

Lastly in Table 11 the results for EM4CI on the **Barley network** and the **Win95** are displayed in the context of multiple queries, illustrating a similar pattern. For the **Barley network** (Table 11a), the learned domain sizes are large ($k_{lrn} = 14, 10$) for each sample size, and accordingly learn time is also large for both settings. However, inference time remains very low.

Table 9: EM4CI on large synthetic models

(a) 99 chain:
$|\boldsymbol{V}| = 99;\ |\boldsymbol{U}| = 49; d = 4; k = 10$. treewidth=2

	1,000 Samples		*10,000 Samples*	
Learning	time(s)	k_{lrn}	time(s)	k_{lrn}
	413.4	6	3887.9	6
Inference				
Query	mad	time(s)	mad	time(s)
$P(V_{98}\|do(V_0))$	0.0093	0.0023	0.0028	0.0022
$P(V_{49}\|do(V_0))$	0.0113	0.0011	0.0041	0.0011
$P(V_{98}\|do(V_{49}))$	0.0093	0.0011	0.0028	0.0011
$P(V_{98}\|do(V_{90}))$	0.0152	0.0004	0.0063	0.0004

(b) 65 diamond:
$|\boldsymbol{V}| = 65;\ |\boldsymbol{U}| = 32; d = 4; k = 10$. treewidth=5

	1,000 Samples		*10,000 Samples*	
Learning	time(s)	k_{lrn}	time(s)	k_{lrn}
	432.3	4	4787.2	4
Inference				
Query	mad	time(s)	mad	time(s)
$P(V_{64}\|do(V_0))$	0.0074	0.0012	0.0022	0.0013
$P(V_{32}\|do(V_{16}))$	0.0193	0.0004	0.0046	0.0005
$P(V_{16}\|do(V_0))$	0.0283	0.0004	0.0150	0.0005
$P(V_{48}\|do(V_{32}))$	0.0086	0.0004	0.0093	0.0004

Table 10: Plug-In & EM4CI results on the **Alarm Network**
$|\boldsymbol{V}| = 32;\ |\boldsymbol{U}| = 5;\ 2 \leq d \leq 4;\ 2 \leq k \leq 3$. treewidth$\approx 3$

(a) Plug-In

	1,000 Samples		*10,000 Samples*	
Query	mad	time(s)	mad	time(s)
$P(HRBP$|do(Shunt))	0.0190	2.9	0.0075	5.7
$P(HRBP$|do(VentAlv))	0.0433	3.5	0.0212	5.7
$P(HR$|do(VentAlv))	0.04706	4	0.0184	5.53
$P(HR$|do(Shunt))	0.0229	3.0	0.00296	6.4

(b) EM4CI

	1,000 Samples		*10,000 Samples*	
Learning	time(s)	k_{lrn}	time(s)	k_{lrn}
	16	2	181	4
Inference				
Query	mad	time(s)	mad	time(s)
$P(HRBP$|do(Shunt))	0.0076	0.0002	0.0043	0.0002
$P(HRBP$|do(VentAlv))	0.0146	0.0003	0.0033	0.0003
$P(HR$|do(VentAlv))	0.0106	0.0002	0.0020	0.0003
$P(HR$|do(Shunt))	0.0101	0.0002	0.0027	0.0002

Table 11: EM4CI results on the Real Networks

(a) "**Barley**":
$|\boldsymbol{V}| = 42;\ |\boldsymbol{U}| = 6;\ 2 \leq d \leq 67; 2 \leq k \leq 9$. treewidth$\approx 7$

	1,000 Samples		*10,000 Samples*	
Learning	time(s)	k_{lrn}	time(s)	k_{lrn}
	199	14	820	10
Inference				
Query	mad	time(s)	mad	time(s)
P(Protein|do(expYield))	0.0066	0.9038	0.0031	0.8882
P(FieldCap|do(protein))	0.0108	0.0004	0.0032	0.0004
P(expYield|do(precipitation))	0.0284	0.0002	0.0064	0.0002
P(weight|do(precipitation))	0.0107	0.0002	0.0023	0.0002

(b) "**Win95**":
$|\boldsymbol{V}| = 59;\ |\boldsymbol{U}| = 17;\ d = 2; k = 2$ treewidth≈ 8

	1,000 Samples		*10,000 Samples*	
Learning	time(s)	k_{lrn}	time(s)	k_{lrn}
	109	2	1081	4
Inference				
Query	mad	time(s)	mad	time(s)
P(Ouput|do(NnPsGrphic))	0.0113	0.0002	0.0008	0.0002
P(PrintData|do(localOK))	0.0768	0.0002	0.0049	0.0002
P(PCtoPRT|do(netOK))	0.0167	0.0001	0.0141	0.0002
P(PrintData|do(lnetOK))	0.0116	0.0002	0.0016	0.0002

B Estimand Expressions

We also provide the estimand expressions for the queries presented in the paper. In some cases these expressions are impractically large to include directly, in which case we provide a table listing the number of variables in the largest summation and function (probability term). All expressions were derived using the Causal Effect package [33].

B.1 Diamond Model

First we present the expression on the diamond model when $|\boldsymbol{V}| = 17$. This expression is quite large, so for $|\boldsymbol{V}| = 65$ we summarize the estimand's properties in a table.

$$
\begin{aligned}
P(V_{16}|do(V_0)) = & \sum_{V_1,V_2,V_3,V_4,V_5,V_6,V_7,V_8,V_9,V_{10},V_{11},V_{12},V_{13},V_{14},,V_{15}} P(V_{15},|V_0,V_{13},V_{14})\times \\
& \frac{\sum_{V_0',V_2,V_6,V_{10}} numerator}{\sum_{V_0',V_2,V_6,V_{10}} denominator} \times P(V_{13}|V_0)P(V_{11}|V_0,V_9,V_{10})P(V_9|V_0)\times \\
& P(V_7|V_0,V_5,V_6)P(V_5|V_0)P(V_3|V_0,V_1,V_2)P(V_1|V_0)\times \\
& \sum_{V_0'}(V_{12}|V_0',V_9,V_{10},V_{11})(V_{10}|V_0',V_9)P(V_0')\sum_{V_0'}(V_8|V_0',V_5,V_6,V_7)(V_6|V_0',V_5)P(V_0') \\
& \times \sum_{V_0'}(V_4|V_0',V_1,V_2,V_3)(V_2|V_0',V_1)P(V_0') \qquad (7)
\end{aligned}
$$

$$
\begin{aligned}
numerator = & P(V_{16}|V_0',V_1,V_2,V_3,V_4,V_5,V_6,V_7,V_8,V_9,V_{10},V_{11},V_{12},V_{13},V_{14},,V_{15})\times \\
& P(V_{14},V_4|V_0',V_1,V_2,V_3,V_5,V_6,V_7,V_8,V_9,V_{10},V_{11},V_{12},V_{13})\times \\
& P(V_2|V_0',V_1,V_5,V_6,V_7,V_8,V_9,V_{10},V_{11},V_{12},V_{13})P(V_8|V_0',V_5,V_6,V_7,V_9,V_{10},V_{11},V_{12},V_{13}) \\
& \times P(V_6|V_0',V_5,V_9,V_{10},V_{11},V_{12},V_{13})P(V_{12}|V_0',V_9,V_{10},V_{11},V_{13})P(V_{10}|V_0',V_9,V_{13})P(V_0') \qquad (8)
\end{aligned}
$$

$$
\begin{aligned}
denominator = & P(V_4|V_0',V_1,V_2,V_3,V_5,V_6,V_7,V_8,V_9,V_{10},V_{11},V_{12},V_{13})\times \\
& P(V_2|V_0',V_1,V_5,V_6,V_7,V_8,V_9,V_{10},V_{11},V_{12},V_{13})P(V_8|V_0',V_5,V_6,V_7,V_9,V_{10},V_{11},V_{12},V_{13}) \\
& \times P(V_6|V_0',V_5,V_9,V_{10},V_{11},V_{12},V_{13})P(V_{12}|V_0',V_9,V_{10},V_{11},V_{13})P(V_{10}|V_0',V_9,V_{13})P(V_0') \qquad (9)
\end{aligned}
$$

Query	size of function	size of sum
$P(V_{64}\|do(V0))$	65	63
$P(V_{32}\|do(V0))$	17	16
$P(V_{16}\|do(V0))$	17	16
$P(V_8\|do(V0))$	17	16

Table 12: Size of largest function & sum in queries for Diamond model when $|\boldsymbol{V}| = 65$

B.2 Cone-cloud Model

Here we give the expression for a query the cone-cloud model when $|\boldsymbol{V}| = 15$. Again, a table is provided summarizing the estimand for $|\boldsymbol{V}| = 45$.

$$
\begin{aligned}
P(V_0|V_{14}, V_{10}, V_4) = \\
&\sum_{V_1,V_2,V_3,V_5,V_6,V_7,V_8,V_9,V_{11},V_{12},V_{13}} P(V_2|V_4, V_5, V_7, V_8, V_9, V_{11}, V_{12}, V_{13}, V_{14})\times \\
&P(V_9|V_{13}, V_{14})P(V_8|V_{12}, V_{13})P(V_1|V_3, V_4, V_6, V_7, V_8, V_{10}, V_{11}, V_{12}, V_{13})\times \\
&P(V_7|V_{11}, V_{12})P(V_6|V_{10}, V_{11})P(V_{11}, V_{12}, V_{13})\times \\
&\sum_{V'_{10},V_{11},V_{12},V_{13},V'_{14}} P(V_0|V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V'_{10}, V_{11}, V_{12}, V_{13}, V'_{14})\times \\
&P(V_5|V_1, V_3, V_4, V_6, V_7, V_8, V_9, V'_{10}, V_{11}, V_{12}, V_{13}, V'_{14})\times \\
&P(V'_{14}|V_1, V_3, V_4, V_6, V_7, V_8, V'_{10}, V_{11}, V_{12}, V_{13})\times \\
&P(V_3, V_{13}|V_6, V_7, V'_{10}, V_{12}, V_{13})P(V'_{10}|V_7, V_{11}, V_{12})P(V_{11}, V_{12})
\end{aligned} \tag{10}
$$

Query	size of function	size of sum
$P(V_0\|do(V_4, V_{36}, V_{44}))$	45	41
$P(V_0\|do(V_{15}, V_{20}, V_{38}, V_{42}))$	45	34
$P(V_0\|do(V_4, V_{21}, V_{27}))$	45	37
$P(V_0\|do(V_{10}, V_{14}, V_{40}))$	45	33

Table 13: Size of largest function & sum in Queries for cone-cloud model when $|\boldsymbol{V}| = 45$

B.3 A Model

For illustration, we show one expression for the A model; the remainder are summarized via a table.

$$
\begin{aligned}
P(N_{51}|do(N_{45})) = \\
&\sum_{\substack{N_3,N_9,N_{10},N_{11},N_{12},N_{13},N_{14},N_{15},N_{17},N_{18},N_{22},N_{23},N_{24},N_{25},N_{26},\\ N_{27},N_{28},N_{29},N_{32},N_{34},N_{35},N_{36},N_{37},N_{39},N_{41},N_{43},N_{47},N_{49},N_{50}}} P(N_{50}|N_3, N_9, N_{10}, \dots \\
&N_{11}, N_{12}, N_{13}, N_{14}, N_{15}, N_{17}, N_{18}, N_{22}, N_{23}, N_{24}, N_{26}, N_{27}, N_{28}, N_{29}, N_{34}, N_{36}, \dots \\
&N_{37}, N_{41}, N_{43}, N_{45}, N_{47}, N_{49})\times \\
&P(N_{47}|N_3, N_{11}, N_{13}, N_{14}, N_{17}, N_{18}, N_{22}, N_{23}, N_{24}, N_{26}, N_{27}, N_{36}, N_{43}, N_{45})\times \\
&P(N_{43})P(N_{49}|N_9, N_{13}, N_{14}, N_{17}, N_{22}, N_{41})P(N_{34}|N_{10}, N_{12}, N_{14}, N_{17}, N_{18}, N_{23}, N_{29})\times \\
&P(N_{18})P(N_{32}|N_{10}, N_{11}, N_{12}, N_{14}, N_{15}, N_{17}, N_{25})P(N_{26}|N_{14})P(N_{25}|N_{11}, N_{14}, N_{17})\times \\
&P(N_{37}|N_3, N_{10}, N_{12}, N_{13}, N_{14}, N_{17}, N_{18}, N_{22}, N_{23}, N_{24}, N_{28}, N_{29}, N_{34}, N_{36})\times \\
&P(N_{22}|N_3, N_{10}, N_{12}, N_{13}, N_{14}, N_{17})P(N_{17}|N_{14})\times \\
&P(N_{35}|N_{10}, N_{12})P(N_{27}|N_{11})P(N_{12}|N_{10})P(N_9) \\
&\sum_{V'_{45}} P(N_{51}|N_3, N_9, N_{10}, N_{11}, N_{12}, N_{13}, N_{14}, N_{15}, N_{17}, N_{18}, N_{22}, N_{23}, N_{24}, N_{25}, \dots \\
&N_{26}, N_{27}, N_{28}, N_{29}, N_{32}, N_{34}, N_{35}, N_{36}, N_{37}, N_{39}, N_{41}, N_{43}, N'_{45}, N_{47}, N_{49}, N_{50})\times \\
&P(N_{41}|N_3, N_9, N_{10}, N_{11}, N_{12}, N_{13}, N_{14}, N_{15}, N_{17}, N_{18}, N_{22}, N_{23}, N_{24}, N_{25}, N_{26} \dots \\
&N_{27}, N_{28}, N_{29}, N_{32}, N_{34}, N_{35}, N_{36}, N_{37}, N_{39}, N'_{45})\times \\
&P(N'_{45}|N_3, N_9, N_{10}, N_{11}, N_{12}, N_{13}, N_{14}, N_{15}, N_{17}, N_{18}, N_{22}, N_{23}, N_{24}, N_{25}, N_{26} \dots \\
&N_{27}, N_{28}, N_{29}, N_{32}, N_{34}, N_{35}, N_{36}, N_{39})\times \\
&P(N_{23}, N_{29}|N_3, N_9, N_{10}, N_{11}, N_{12}, N_{13}, N_{14}, N_{15}, N_{17}, N_{18}, N_{22}, N_{24}, N_{25}, \dots \\
&N_{26}, N_{27}, N_{28}, N_{32}, N_{35}, N_{36}, N_{39})\times \\
&P(N_{24}, N_{28}, N_{36}, N_{39}|N_3, N_9, N_{10}, N_{11}, N_{12}, N_{13}, N_{14}, N_{15}, N_{17}, N_{22}, N_{25}, \dots \\
&N_{26}, N_{27}, N_{32}, N_{35})\times \\
&P(N_{13}, N_{14}|N_3, N_9, N_{10}, N_{11}, N_{12}, N_{15}, N_{27}, N_{35})P(N_{15}|N_3, N_9, N_{10}, N_{11}, N_{12}, N_{27})\times \\
&P(N_{11}|N_3, N_9, N_{10}, N_{12})P(N_{10}|N_3, N_9)P(N_3)
\end{aligned} \tag{11}
$$

Query	size of function	size of sum
$P(N_{51}\|do(N_{10}))$	31	29
$P(N_{51}\|do(N_{14}))$	31	29
$P(N_{51}\|do(N_{41}))$	31	29
$P(N_{51}\|do(N_{45}))$	31	29

Table 14: Size of largest function & sum in queries for A-network where $|\boldsymbol{V}| = 54$

B.4 Alarm Network

The Alarm network has the following estimand expressions for the four evaluated queries:

$$P(HRBP|do(Shunt)) = \sum_{ArtCO2} P(HRBP|Shunt, ArtCO2)P(ArtCO2)$$

$$P(HRBP|do(VentAlv)) = \sum_{SaO2} P(HRBP|VentAlv, SaO2)P(SaO2)$$

$$P(HR|do(VentAlv)) = \sum_{SaO2} P(HR|VentAlv, SaO2)P(SaO2)$$

$$P(HR|Shunt) = \sum_{ArtCO2} P(HR|Shunt, ArtCO2)P(ArtCO2)$$

B.5 Barley Network

The Barley network has the following estimand expressions for the four evaluated queries:

$$P(Protein|do(expYield)) = \sum_{n4protein} P(n4protein|protein)\times \sum_{expYield'} P(Protein|expYield', n4protein) \times P(expYield') \quad (12)$$

$$P(Protein|do(FieldCap)) = \sum_{AllwedFertN} P(protein|FieldCap, AllowedFertN) \times P(AllowedFertN) \quad (13)$$

$$P(yield|do(Precipitation)) = P(Yield)$$

$$P(weight|do(Precipitation)) = P(weight)$$

B.6 Win95pts Network

The Win95pts network has the following estimand expressions for the four evaluated queries:

$$P(Problem|do(NonPSGrphic)) = P(Problem)$$

$$P(Printdata|do(LocalOK)) = \sum_{\substack{DOSlocalOK,DOSnetOK,\\ GDIOutputOK,netOK}} P(printData|localOK, DOSlocalOK, DOSnetOK, GDIOutputOK, netOk) \times P(DOSlocalOK, DOSnetOK, GDIOutputOK, netOk) \quad (14)$$

$$P(PCtoPRT|do(netOk)) = \sum_{\substack{DOSlocalOK,DOSnetOK,\\ GDIOutputOK,localOK}} P(PCtoPRT|netOK, DOSlocalOK, DOSnetOK, GDIOutputOK, localOk) \times P(DOSlocalOK, DOSnetOK, GDIOutputOK, localOk) \quad (15)$$

$$P(printData|do(netOk)) = \sum_{\substack{DOSlocalOK,DOSnetOK,\\ GDIOutputOK,localOK}} P(printData|netOK, DOSlocalOK, DOSnetOK, GDIOutputOK, localOk) \times P(DOSlocalOK, DOSnetOK, GDIOutputOK, localOk) \quad (16)$$

C Code

To run EM4CI:
All code can be found here [26] In order to run the code for this paper, a license to use BayesFusion software package SMILE is required. EM4CI is written in C++. There is one source code file for the learning phase and one for inference. They are named:

```
learn_main.cpp   inf.cpp
```

In order to compile the source files, the command is:

```
g++ -O3 learn_main.cpp -o learn.out -I./smile
-L./smile -lsmile
g++ -O3 inf.cpp -o inf.out -I./smile
-L./smile -lsmile
```

Which will produce the executables:

```
learn.out         inf.out
```

The **learn.out** file expects command line arguments of the model file, the em-model file with a domain specified for the unobserved variables, a data csv file containing samples on the observed variables, and the number of samples. An example run is:

```
./learn.out models_xdsl/ex1_TD2_10.xdsl
models_xdsl/em_ex1_TD2_10_ED2_0.xdsl
data/100/ex1_TD2_10.csv  100
```

The **inf.out** file expects command line arguments of the model file, the learned model file that will be used to perform inference on, the query variable, Y, in $P(Y|do(\mathbf{X}))$, the do variables(s) $\mathbf{X}$, and the number of samples used in the learning phase. An example run is:

```
./inf.out models_xdsl/ex1_TD2_10.xdsl
learned_models/100/em_ex1_TD2_10_ED2_0.xdsl
Y X 100
```

The resulting log-likelihood, BIC score, time for learning, time for inference, and *mean absolute deviation (mad)* are output to csv files, LL.csv, BIC.csv, timesLearn.csv, timesInf.csv, and err.csv, respectively. These will all be output to a folder named after the model, and containing a different subfolder per sample size and assumed domain size in the learning phase. For example, if running for assumed domain size 2 and sample size 100, the output will be in folder:

```
ex1_TD2_10/100/em_ex1_TD2_10_ED2
```

The learned model files will be in the folder:

```
learned_models/ex1_TD2_10/100
```

The bash script **em4ci_wrapper.sh** was used to automate the learning process. This script will automatically iterate through increasing latent domain sizes, while running the EM algorithm 10 times for each latent domain size. It will stop when the BIC score stops decreasing, and will output the minimum BIC score with the latent domain size of the final learned model. To run this script you can pass in the model name and number of samples. For example:

```
./em4ci_wrapper.sh ex1_TD2_10 100
```

The wrapper assumes all model files are contained in a folder named **models_xdsl** and data files are contained in a folder called **data** with subfolders corresponding to the number of samples, like **data/100**.

The learned models files are of the form **em_ex1_TD2_10_ED2_0.xdsl** where the number after ED corresponds to the assumed domain size of the latent variables, and the last number corresponds to one of the runs $\{0, \ldots, 9\}$ that produced that model. You can perform inference on any learned model you like, but the **em4ci_wrapper.sh** outputs the run that correspond to the highest likelihood models with minimum BIC score, so we suggest using those.

All model files are in XDSL format, for more information see the Bayefusion Documentation [1]