PAC Learning of Causal Trees with Latent Variables

Prasad Tadepalli,[1] **Stuart J. Russell**[2]
[1] School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, Oregon, 97331
[2] Division of Computer Science, University of California, Berkeley, California, 94720
tadepall@oregonstate.edu, russell@cs.berkeley.edu

Abstract

Learning causal probabilistic models with latent variables from observational and experimental data is an important problem. In this paper we present a polynomial-time algorithm that PAC-learns the structure and parameters of a rooted, tree-structured causal network of bounded degree where the internal nodes of the tree cannot be observed or manipulated. Our algorithm is the first of its kind to provably learn the structure and parameters of tree-structured causal models with latent internal variables from random examples and active experiments.

Introduction

Structural Causal Models (SCMs) offer a probabilistic model of causality with intuitive semantics (Pearl and Mackenzie 2018; Pearl 2009). They support sound inference algorithms to answer a variety of causal queries (Pearl 2019). In this paper, we study the problem of efficiently learning a causal model by interacting with a stochastic black-box function through interventions and observations.

It is well-known that interventional experiments are often needed to identify the true causal structure from its Markov equivalence class (Reichenbach 1991). There has been a lot of empirical research on learning causal models by combining observational and interventional data acquired offline (Cooper and Yoo 1999; Peters, Bühlmann, and Meinshausen 2016; Silander and Myllymaki 2012; Eaton and Murphy 2007; He and Geng 2008). Previous work in active learning of Bayesian networks also explored the use of greedy heuristics such as structure entropy and expected posterior loss over the query distribution (Li and Leong 2009; Tong and Koller 2001; He and Geng 2008). More theoretically grounded approaches such as IC*, FCI, and RFCI are applicable in the presence of latent variables (Spirtes, Glymour, and Scheines 2000; Colombo et al. 2012) and rely on conditional independence queries over sets of variables. There are also other algorithmic approaches that actively design optimal experiments to learn the structure and parameters of causal network (He and Geng 2008). In recent work, optimal active learning strategies for identifying causal network structures with minimal numbers of single-node and multi-node interventions have been studied (Hauser and Bühlmann 2014). Other methods have explored the use of path queries (Bello and Honorio 2017), experimental design (Kocaoglu, Shanmugam, and Bareinboim 2017), and constraint satisfaction to learn the structure (Hyttinen et al. 2013).

In contrast with most prior work, we allow the learning algorithm to intervene only at the pre-specified *input* variables and observe the conditional probability of the *target* variable. The remaining variables can neither be observed nor manipulated. More specifically, we address the problem of learning a tree-structured causal network with bounded in-degree k. The root of the tree represents the target or output variable, and the leaves represent the input variables, where the causality flows from the inputs to the target. The objective is to model the probability of the target given the inputs.

We approach the problem of learning causal trees of bounded degree from the perspective of *probably approximately correct* (PAC) learning of probabilistic concepts or p-concepts (Haussler 1992; Kearns and Schapire 1994). More specifically, our objective is to efficiently learn, with high probability, an approximate conditional distribution for the target variable given the inputs drawn from an unknown but fixed natural distribution. In the more relaxed setting of PAC-prediction, the learner is allowed to express the target function in any form that can be evaluated in polynomial time. When all variables are Boolean and the model is deterministic, an SCM can encode a Boolean circuit. It is known that PAC-prediction of Boolean circuits from random examples is cryptographically hard, i.e., as hard as solving cryptographic problems such as integer factoring (Pitt and Valiant 1988). Since probabilistic Boolean SCMs, where only the inputs and the final output of the SCM are observed, generalize Boolean circuits, it follows that learning them from random examples is as hard as factoring. As with Boolean circuits, the hardness remains even when the SCM has a known tree structure (Pitt and Warmuth 1990).

Despite the above negative results, it has been shown that the functional formulas at the latent nodes of a deterministic Boolean circuit with known tree structure can be learned from random examples and membership queries (Tadepalli 1993; Tadepalli and Russell 1998). A membership query asks the output of the target function for any input desired by the learning algorithm (Angluin 1987). In previous work, we showed that the deterministic algorithm of Tadepalli (1993) can be extended to causal trees of known structure (Tadepalli, Barrie, and Russell 2019). Here, we present a new algorithm to learn both the structure and the parameters of probabilistic causal trees from observational and experimental data. Our work can be viewed as extending the work of Bshouty, Hancock, and Hellerstein (1995) to *probabilistic* read-once formulas and that of Schapire (1994) to general rather than product distributions over the input variables.

Our PAC-learning algorithm has access to a source of random examples drawn from a fixed but unknown natural distribution as well as an experimentation oracle that reveals the conditional probability of the target variable for any desired input. To simplify the analysis, we assume that this probability is exact. The key challenge is to infer the tree structure and the conditional distributions at the intermediate variables, which are neither observed nor manipulated, with only polynomial time and polynomially many queries.

Our algorithm proceeds in a bottom-up (causal) direction testing for a set of at most k subtrees from the current set that forms a valid next-level supertree. It then finds the "control" and "context" inputs for the root of the new supertree. The control inputs of a latent variable set it to a desired value. The context input of a latent variable allows the inference of its conditional distribution from the conditional distribution of the target variable given the input. Together, the context input of a latent variable and the control inputs of its causal parents allow us to infer its conditional probability table. To the best of our knowledge our algorithm is the first of its kind to efficiently learn the structure and parameters of a non-trivial class of causal models with latent variables from random examples and active experiments.

Problem Setup

A Structural Causal Model (SCM) is a 4-tuple (V, D, G, P), where V is a set of random variables over the domain D, G is a directed acyclic graph over V, and P is the conditional distribution of variables $V_i \in V$ given the values of their parents (causal antecedents) $Pa(V_i)$ in G (Pearl and Mackenzie 2018). A tree-structured causal model or *causal tree* is an SCM in the form of a rooted tree, where V is partitioned into the input variables I which are at the leaves of the tree, the single root node R which is the target of prediction, and the latent variables L which are internal to the tree. The direction of causality is from the leaf nodes that represent the input variables to the root. (Note that this reverses the usual convention of parent–child relationship in trees to be consistent with the terminology of causal networks.) In particular we assume that each node (variable) has a single child and has multiple parents which are its causal antecedents.

We often use the word 'node' to mean the variable represented at that node. We assume that the domain D is Boolean and that the probability model is represented as conditional probability tables (CPTs) at each node. The CPT has one entry for every possible value tuple of the parents of each latent variable $V_n \in L$. We write $P(n|\mathbf{x})$ to represent the conditional probability $P(V_n = 1|I = \mathbf{x})$, where $\mathbf{x}$ is a Boolean vector assigned to the variables in I. We use $\mathbf{x}_n$ to refer to a partial assignment to the input variables that are causal ancestors of n and $\mathbf{x}_{\overline{n}}$ to represent a partial assignment to the remaining input variables. Because of the tree structure and the rules of d-separation, $P(n|\mathbf{x}) = P(n|\mathbf{x_n})$. From the rules of do-calculus, it also follows that $P(n|\mathbf{x}) = P(n|do(\mathbf{x})) = P(n|do(\mathbf{x_n}))$ in Pearl's notation. The SCM induces a conditional distribution $P(R{=}1|do(\mathbf{x})) = P(r|\mathbf{x})$, which we seek to learn.

A PAC-learning model for probabilistic concepts was introduced by Kearns and Schapire (1994) and has been used to show that Bayesian networks with known structure and bounded degree are learnable from polynomial-sized random samples (Dasgupta 1997; Haussler 1992; Pollard 1984). Unfortunately, empirical loss minimization for Bayesian networks even with known structure reduces to predicting arbitrary Boolean functions, which is cryptographically hard (Pitt and Valiant 1988; Pitt and Warmuth 1990). In the deterministic case, this computational barrier is broken when the network is tree-structured and the learning algorithm has access to a membership oracle (Tadepalli 1993; Tadepalli and Russell 1998).

The goal of the current paper is to learn both the structure and parameters of probabilistic tree-structured causal models. The algorithm has access to a source of random examples $\mathbf{x}$ drawn from a natural distribution, which we also denote by P with some abuse of notation.[1] It can also make experimental queries EXP$(\mathbf{y})$ on any arbitrary instance $\mathbf{y}$ and receive as output the exact conditional probability $P(r|\mathbf{y})$. The goal of the learning algorithm, given parameters ϵ and δ, is to output with a probability at least $1-\delta$ an approximate causal model that is correct with a probability at least $1-\epsilon$ on test examples drawn from P. Note that the correctness here means that the conditional probability $P(r|\mathbf{x})$ should be identical to that of the true model. The number of random examples, the number of experimental queries, and the time complexity of the algorithm are required to be polynomial functions of $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, and the size of the target model, i.e., the total number of CPT entries for all its variables.

Control Normal Form of Causal Trees

In this section, we prove some important properties of causal trees that are exploited by our algorithm.

Definition 1 *Let R be the root of a tree whose leaf inputs are partitioned into sets A and B. The* causal effect *on R of changing the value of A from $\mathbf{a}$ to $\mathbf{a}'$ while the other inputs B remain fixed at $\mathbf{b}$, denoted by, $\Delta(r|A = \mathbf{a} \to \mathbf{a}', \mathbf{b}) = \Delta(r|\mathbf{a} \to \mathbf{a}', \mathbf{b})$ is defined as $P(r|\mathbf{a}', \mathbf{b}) - P(r|\mathbf{a}, \mathbf{b})$.*

We now state and prove the *Chain Rule of Causal Effects*, which says that the causal effect on a variable under a fixed context can be decomposed into a product of causal effects.

Theorem 1 (Chain Rule of Causal Effects) *Let T be a causal tree rooted at R and n be an internal variable. Let A be the set of input variables under n, i.e., n's causal ancestors, and B be the remaining input variables. Then $\Delta(r|\mathbf{a} \to \mathbf{a}', \mathbf{b}) = \Delta(n|\mathbf{a} \to \mathbf{a}')\Delta(r|n{=}0 \to 1, \mathbf{b})$.*

[1]Unlike in Bayesian networks, we do not assume that the different components of $\mathbf{x}$ are independent of each other, i.e., P need not be a product distribution.

Proof: Thanks to the tree structure, n d-separates nodes in A from the root R. Hence,
$P(r|\mathbf{a},\mathbf{b}) = \sum_{x\in\{0,1\}} P(n{=}x|\mathbf{a},\mathbf{b})P(r|n{=}x,\mathbf{a},\mathbf{b})$
$= \sum_{x\in\{0,1\}} P(n{=}x|\mathbf{a})P(r|n{=}x,\mathbf{b})$ (d-separation)
$= P(n|\mathbf{a})P(r|n{=}1,\mathbf{b}) + (1-P(n|\mathbf{a}))P(r|n{=}0,\mathbf{b})$
$= P(n|\mathbf{a})(P(r|n{=}1,\mathbf{b})-P(r|n{=}0,\mathbf{b}))+P(r|n{=}0,\mathbf{b})$
$\Delta(r|\mathbf{a}\to\mathbf{a}',\mathbf{b}) = P(r|\mathbf{a}',\mathbf{b}) - P(r|\mathbf{a},\mathbf{b})$
$= P(n|\mathbf{a}')(P(r|n{=}1,\mathbf{b}) - P(r|n{=}0,\mathbf{b}))$
$+P(r|n{=}0,\mathbf{b})-$
$(P(n|\mathbf{a})(P(r|n{=}1,\mathbf{b})-P(r|n{=}0|\mathbf{b}))+P(r|n{=}0,\mathbf{b}))$
$= (P(n|\mathbf{a}') - P(n|\mathbf{a}))(P(r|n{=}1,\mathbf{b}) - P(r|n{=}\,0,\mathbf{b}))$
$= \Delta(n|\mathbf{a}\to\mathbf{a}')\Delta(r|n = 0\to 1,\mathbf{b})$ □

The Chain Rule of Causal Effects is useful to show that the conditional probability tables of causal trees with latent internal nodes can be transformed into a special normal form that preserves the conditional distribution of the output given the inputs.

Definition 2 *A causal tree is in control normal form (CNF) if for every latent node $n \in L$ there are some vectors $\mathbf{d}$ and $\mathbf{z}$ such that $P(n|Pa(n){=}\mathbf{d}){=}1$ and $P(n|Pa(n){=}\mathbf{z}){=}0$.*

If a causal tree is in CNF, there are inputs that can deterministically set any internal variable to 0 or 1. We can find these inputs by choosing appropriate values for each parent of the internal node to set it to 0 or 1 and recursing over the parents to do the same.

We now describe the algorithm Normalize, whose sole purpose is to constructively show that any rooted causal tree can be converted to CNF. It is not part of the learning algorithm. In Algorithm 1 and elsewhere, we assume n_i is the i^{th} parent of n, and n_i' is the set of all other parents of n. Normalize processes the latent nodes in the causal (topological) order (line 2). Lines 3 and 4 compute lo and hi, the lowest and the highest entries in the CPT of n_i. Note that if $lo = hi$, then n_i's value does not depend on its parents, and hence on any of its causal ancestors. In this case, the node n_i itself can be removed from the tree and be marginalized in its child's CPT. Hence, w.l.o.g. we assume that $lo \neq hi$ for any internal node. This allows us to scale the old CPT entries, where lo maps to 0 and hi maps to 1 (lines 5-7). Lines 9–12 compensate for this change at node n by adjusting its CPT with respect to node n_i. Lemma 1 shows that the compensation preserves the distribution of n for any values of parents of n_i and the other parents of n. Lemma 1 and Theorem 2 justify the normalization algorithm.

Lemma 1 *Consider a causal tree for which* Normalize *has been applied at node n_i. Then for any child n of n_i and vectors $\mathbf{x}$ and $\mathbf{y}$ of suitable dimensions,*
$P(n|Pa(n_i) = \mathbf{x}, n_i' = \mathbf{y}) = \hat{P}(n|Pa(n_i) = \mathbf{x}, n_i' = \mathbf{y})$, *where P represents the probabilities based on the original CPTs and $\hat{P}$ represents the probabilities based on the new CPTs.*

Proof: Consider a subtree of the causal tree which is rooted at n and includes all causal parents of n and n_i and no other nodes. Let $X = Pa(n_i)$ and $Y = Pa(n) - \{n_i\}$. Let $\hat{\Delta}$ represent the Δ after normalization.

Algorithm 1 Normalizing a Causal Tree

```
1:  Procedure Normalize($V, G, P$)
2:  for each latent node $n_i \in L$ in topological order do
3:      $lo \leftarrow \min_x P(n_i|Pa(n_i) = \mathbf{x})$ in the CPT of $n_i$
4:      $hi \leftarrow \max_x P(n_i|Pa(n_i) = \mathbf{x})$ in the CPT of $n_i$
5:      for each assignment vector $\mathbf{z}$ to $Pa(n_i)$ do
6:          $P(n_i|Pa(n_i) = \mathbf{z}) \leftarrow \frac{P(n_i|pa(n_i)=\mathbf{z})-lo}{hi-lo}$
7:      end for
8:      $n_i' \leftarrow Pa(n) - n_i$
9:      for each assignment vector $\mathbf{y}$ to nodes in $n_i'$ do
10:         $P(n|n_i = 0, n_i' = \mathbf{y}) \leftarrow (1-lo)P(n|n_i = 0, n_i' = \mathbf{y}) + loP(n|n_i{=}1, n_i'{=}\mathbf{y}))$
11:         $P(n|n_i = 1, n_i' = \mathbf{y}) \leftarrow (1-hi)P(n|n_i = 0, n_i' = \mathbf{y}) + hiP(n|n_i{=}1, n_i'{=}\mathbf{y}))$
12:     end for
13: end for
14: end Normalize
```

We show that the causal effects at node n are preserved by Normalize when $lo \neq hi$. From line 6, we have

$$\begin{aligned}\hat{\Delta}(n_i|\mathbf{x}\to\mathbf{x}') &= \hat{P}(n_i|\mathbf{x}') - \hat{P}(n_i|\mathbf{x})\\ &= \tfrac{P(n_i|\mathbf{x}')-P(n_i|\mathbf{x})}{hi-lo} = \tfrac{\Delta(n_i|\mathbf{x}\to\mathbf{x}')}{hi-lo}\ .\end{aligned}$$

From lines 10-11 of Normalize, we have

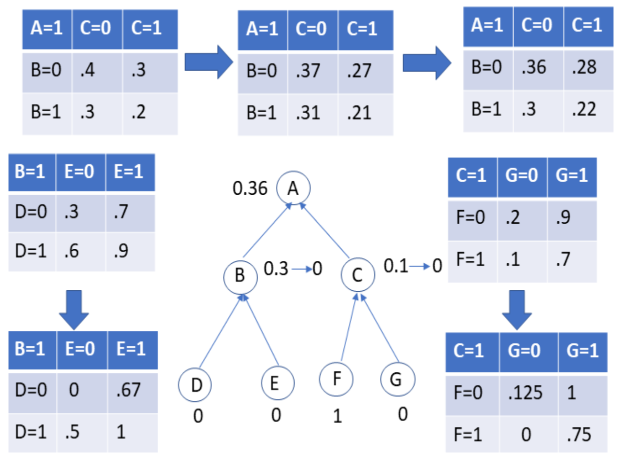
$$\begin{aligned}&\hat{\Delta}(n|n_i = 0\to 1,\mathbf{y})\\ &= \hat{P}(n|n_i{=}1, n_i'{=}\mathbf{y}) - \hat{P}(n|n_i{=}0, n_i'{=}\mathbf{y})\\ &= (hi-lo)(P(n|n_i{=}1, n_i'{=}\mathbf{y}) - P(n|n_i{=}0, n_i'{=}\mathbf{y}))\\ &= (hi-lo)\Delta(n|n_i = 0\to 1,\mathbf{y})\ .\end{aligned}$$

From the Chain Rule of Causal Effects (Theorem 1),

$$\begin{aligned}&\hat{\Delta}(n|\mathbf{x}\to\mathbf{x}',\mathbf{y}) = \hat{\Delta}(n_i|\mathbf{x}\to\mathbf{x}')\hat{\Delta}(n|n_i = 0\to 1,\mathbf{y})\\ &= \tfrac{\Delta(n_i|\mathbf{x}\to\mathbf{x}')}{hi-lo}(hi-lo)\Delta(n|n_i = 0\to 1,\mathbf{y})\\ &= \Delta(n|\mathbf{x}\to\mathbf{x}',\mathbf{y})\ .\end{aligned}$$

We now set $\mathbf{x}' = \mathtt{argmin}_{\mathbf{w}} P(n_i|Pa(n_i) = \mathbf{w})$, so that $P(n_i|Pa(n_i) = \mathbf{x}') = lo$ per line 3 and $\hat{P}(n_i|Pa(n_i) = \mathbf{x}') = 0$ per line 6 of Normalize.
$\hat{P}(n|Pa(n_i) = \mathbf{x}', n_i' = \mathbf{y}) = \hat{P}(n|n_i = 0, n_i' = \mathbf{y})$
$= (1-lo)P(n|n_i{=}0, n_i'{=}\mathbf{y}) + loP(n|n_i{=}1, n_i'{=}\mathbf{y}))$
$= P(n_i{=}0|Pa(n_i) = \mathbf{x}')P(n|n_i{=}0, n_i'{=}\mathbf{y})$
$+P(n_i{=}1|Pa(n_i) = \mathbf{x}')P(n|n_i{=}1, n_i'{=}\mathbf{y}))$
$= P(n|Pa(n_i) = \mathbf{x}', n_i' = \mathbf{y})$.
Since the probability is preserved for $\mathbf{x}', \mathbf{y}$ and the causal effects are preserved for all $\mathbf{x}, \mathbf{x}'$ and $\mathbf{y}$, the Lemma follows.
$\hat{P}(n|Pa(n_i) = \mathbf{x}, n_i' = \mathbf{y})$
$= \hat{P}(n|Pa(n_i) = \mathbf{x}', n_i' = \mathbf{y}) - \hat{\Delta}(n|\mathbf{x}\to\mathbf{x}',\mathbf{y})$
$= P(n|Pa(n_i) = \mathbf{x}', n_i' = \mathbf{y}) - \Delta(n|\mathbf{x}\to\mathbf{x}',\mathbf{y})$
$= P(n|Pa(n_i) = \mathbf{x}, n_i' = \mathbf{y})$ □

Since each transformation at n_i preserves the conditional distribution $P(n|Pa(n_i), n_i')$, a sequence of such transformations at each n_i preserves it as well. Repeating this argument for each internal node n shows that the probability of the outputs given any input is preserved. The following theorem formalizes the argument.



A=1	C=0	C=1
B=0	.4	.3
B=1	.3	.2

A=1	C=0	C=1
B=0	.37	.27
B=1	.31	.21

A=1	C=0	C=1
B=0	.36	.28
B=1	.3	.22

B=1	E=0	E=1
D=0	.3	.7
D=1	.6	.9

B=1	E=0	E=1
D=0	0	.67
D=1	.5	1

C=1	G=0	G=1
F=0	.2	.9
F=1	.1	.7

C=1	G=0	G=1
F=0	.125	1
F=1	0	.75

Figure 1: The CPTs show the probability that a variable = 1, given its parents. Normalization at node B linearly transforms B's CPT and changes A's CPT to the intermediate table at the top. Normalization at node C linearly transforms its CPT and changes A's CPT to its final form. After normalization both B and C have a 0 and a 1 in their CPTs. The input 0010 sets both nodes B and C to 0 in the normalized network, but preserves the probability of the root at 0.36.

Theorem 2 *Every causal tree over binary variables can be transformed into control normal form to preserve the conditional distribution of the root given the inputs.*

Proof: We show that for all input vectors $\mathbf{z}$,the probability $P(r|\mathbf{z})$ is preserved by Normalize. Let n be a node which has a directed path to the root R and n_i be a parent of n.

$$P(r|\mathbf{z}) = \sum_{w\in\{0,1\}} P(r|n{=}w,\mathbf{z})P(w|\mathbf{z})$$
$$= \sum_{w\in\{0,1\}} P(r|n{=}w,\mathbf{z}) \sum_{\mathbf{x},\mathbf{y}} P(n'_i{=}\mathbf{y}|\mathbf{z})$$
$$P(Pa(n_i){=}\mathbf{x}|\mathbf{z})P(n{=}w|Pa(n_i){=}\mathbf{x}, n'_i{=}\mathbf{y})$$

Note that $P(r|n{=}w,\mathbf{z})$ is preserved by Normalize which only changes the CPTs of n and its parent n_i. Given that the nodes are normalized in causal order, $P(Pa(n_i){=}\mathbf{x}|\mathbf{z})$ and $P(n'_i{=}\mathbf{y}|\mathbf{z})$ do not change either. By Lemma 1, the changes at nodes n and n_i are such that $P(n{=}w|Pa(n_i){=}\mathbf{x}, n'_i{=}\mathbf{y})$ is preserved for $w{=}1$ or 0. Hence the value of the whole expression is preserved. □

For example, consider the 4-input tree-structured causal network in Figure 1. We assume all variables are Boolean valued. Only the input variables D, E, F, G at the leaf nodes and the target A at the root are observable. B and C are latent. The figure shows how the CPTs of A, B, and C are transformed to normalize the causal network and preserve the distribution $P(A|D, E, F, G)$. For example, the notation $0.3 \rightarrow 0$ on node B indicates that the $P(B{=}1|D{=}0, E{=}0, F{=}1, G{=}0) = 0.3$ before normalization and 0.0 after normalization. The conditional probability of node C similarly changes from 0.1 to 0. The conditional probability of A on the other hand remains the same as before at 0.36.

The Learning Algorithm

We now introduce the notion of distinguishability of a latent node, which allows us to infer its distribution given the inputs. We then describe our algorithm in multiple stages.

Distinguishability

Given that any target causal tree can be put in CNF, we assume that the target causal tree is in CNF and seek to learn it. Recall that for trees in CNF, every internal node n has control inputs $\mathbf{d}(\mathbf{n})$ and $\mathbf{z}(\mathbf{n})$ to its leaves that set it to 1 and 0 respectively. In addition, if there is also an input context, i.e., an assignment to the input variables other than the leaves of n, that makes this internal node have an impact on the distribution of the target variable, we call the node distinguishable. Distinguishability allows us to infer the distribution of the latent internal node from the distribution of the target variable when the node's context is set appropriately. If a node is not distinguishable by any context, it has little impact on the target distribution for any input, and can be dropped from the tree. Hence we will only be interested in distinguishable internal nodes for learning.

Definition 3 *A node n is* distinguishable *if there is an assignment $\mathbf{c}(\overline{\mathbf{n}})$ to the inputs other than the leaves of n, called the* distinguishing context *of n such that $|P(r|\mathbf{d}(\mathbf{n}), \mathbf{c}(\overline{\mathbf{n}})) - P(r|\mathbf{z}(\mathbf{n}), \mathbf{c}(\overline{\mathbf{n}}))| > 0$.*

When a node n is distinguishable, we assume $P(r|\mathbf{c}(\overline{\mathbf{n}}), n = 1) > P(r|\mathbf{c}(\overline{\mathbf{n}}), n = 0)$ so that $P(r|\mathbf{c}(\overline{\mathbf{n}}), \mathbf{d}(\mathbf{n})) > P(r|\mathbf{c}(\overline{\mathbf{n}}), \mathbf{z}(\mathbf{n}))$. This is w.l.o.g. since the internal nodes are latent, which allows redefining 0 and 1 values for them arbitrarily.

Suppose n is a distinguishable node with control inputs $\mathbf{d}(\mathbf{n})$ and $\mathbf{z}(\mathbf{n})$ and context input $\mathbf{c}(\overline{\mathbf{n}})$. Let $\mathbf{y_n}$ represent some assignment to the leaf (input) nodes under the subtree rooted at node n. The following shows how to infer the conditional distribution of n given $\mathbf{y_n}$ from the conditional distribution of the root node given $\mathbf{y_n}$ and $\mathbf{c}(\overline{\mathbf{n}})$.

$$\begin{aligned} P(r|\mathbf{y_n}, \mathbf{c}(\overline{\mathbf{n}})) &= P(n = 1|\mathbf{y_n})P(r|\mathbf{y_n}, \mathbf{c}(\overline{\mathbf{n}}), n{=}1) \\ &\quad +(P(n = 0|\mathbf{y_n}))P(r|\mathbf{y_n}, \mathbf{c}(\overline{\mathbf{n}}), n{=}0) \\ &= P(n{=}1|\mathbf{y_n})P(r|\mathbf{d}(\mathbf{n}), \mathbf{c}(\overline{\mathbf{n}})) \\ &\quad +(1 - P(n = 1|\mathbf{y_n}))P(r|\mathbf{z}(\mathbf{n}), \mathbf{c}(\overline{\mathbf{n}})) \end{aligned}$$

The last step follows because n d-separates $\mathbf{y_n}$ from the root, and $P(n = 1|\mathbf{d}(\mathbf{n})) = P(n = 0|\mathbf{z}(\mathbf{n})) = 1$. The last equation can be solved for $P(n = 1|\mathbf{y_n})$ yielding:

$$\begin{aligned} P(n{=}1|\mathbf{y_n}) &= \frac{P(r|\mathbf{y_n}, \mathbf{c}(\overline{\mathbf{n}})) - P(r|\mathbf{z}(\mathbf{n}), \mathbf{c}(\overline{\mathbf{n}}))}{P(r|\mathbf{d}(\mathbf{n}), \mathbf{c}(\overline{\mathbf{n}})) - P(r|\mathbf{z}(\mathbf{n}), \mathbf{c}(\overline{\mathbf{n}}))} \\ &= \frac{\mathrm{EXP}(\mathbf{y_n}, \mathbf{c}(\overline{\mathbf{n}})) - \mathrm{EXP}(\mathbf{z}(\mathbf{n}), \mathbf{c}(\overline{\mathbf{n}}))}{\mathrm{EXP}(\mathbf{d}(\mathbf{n}), \mathbf{c}(\overline{\mathbf{n}})) - \mathrm{EXP}(\mathbf{z}(\mathbf{n}), \mathbf{c}(\overline{\mathbf{n}}))} \end{aligned} \quad (1)$$

Equation 1 shows that the context and control inputs of any node n would make it effectively observable via EXP queries.

Algorithm 2 The Bottom-up Tree-Building Algorithm

```
 1: Procedure BuildTree (Sample $S$, Degree $k$, Inputs $V_1, \dots V_p$, Root label $R$)
 2: Forest $F = \{V_1 \dots V_p\}$
 3: repeat
 4:   $C$ = All possible trees over $1 < j \leq k$ subtrees in $F$
 5:   while $C$ is non-empty do
 6:     Select and remove largest tree $T \in C$ with root $n$
 7:     if $\forall \mathbf{x} \in S$,EXP$(\mathbf{x})$ =EXP$(\mathbf{x_n}, \mathbf{0_{\overline{n}}})$ then
 8:       Relabel its root $n$ as $R$
 9:       FillCPT$(R)$
10:     else $\mathbf{y}$ = argmax$_{x \in S}$| EXP$(\mathbf{x})-$EXP$(\mathbf{0_n}, \mathbf{x_{\overline{n}}})$|
11:       if EXP$(\mathbf{y}) \neq$EXP$(\mathbf{0_n}, \mathbf{y_{\overline{n}}})$ then
12:         $\mathbf{c}(\overline{\mathbf{n}}) = \mathbf{y_{\overline{n}}}$
13:         if $f(T, S) < \theta$ then
14:           FillCPT$(n)$
15:           $C \leftarrow C - \{T' | T'$ overlaps with $T\}$
16:           $F \leftarrow F \bigcup \{T\} -$Subtrees$(T)$
17:   end while
18: until $|F| = 1$ or some tree in $F$ has its root labeled $R$
19: if $\exists$ a tree $T \in F$ rooted at $R$ return $T$ else Fail
20: end BuildTree
```

The Tree-Building Algorithm

Algorithm 2 describes the top level pseudocode of our approach. The algorithm BuildTree proceeds bottom-up starting with a forest F initialized to the set of the input nodes which can be viewed as trivial trees (line 2). In each iteration of the Repeat loop (lines 3–18), the algorithm constructs a candidate set of trees C, where each tree has a new distinct root node and a unique subset of 2 to k trees in F as its subtrees (line 4).

The While loop of the algorithm iterates over the candidates in C and selects them to add to F (lines 5–17). It selects a tree $T \in C$ with the largest number of leafs and removes it from C (line 6). The selection criterion is a heuristic that helps move quickly towards the root and does not affect the correctness. It checks to see if EXP$(\mathbf{x})$ changes by replacing the inputs which are not in T with 0's for any sample $x \in S$ (line 7). If it does not, it means that the inputs not in T are not relevant for the sample at hand, and hence it replaces its root with the root label R and calls FillCPT to fill its conditional probability table (lines 8–9).

Otherwise, the algorithm tries to find a distinguishing context $c(\overline{n})$ for the root n, by searching for an input $\mathbf{x} \in S$ which changes EXP$(\mathbf{x})$ the most when the leafs under node n are replaced by the zero vector 0_n (line 10). If the amount of change is non-zero, it sets the distinguishing context $c(\overline{n})$ to the corresponding input $\mathbf{y_{\overline{n}}}$ (lines 11–12). It evaluates the resulting tree T using the scoring function f-score (line 13). If the score is less than a small threshold θ, it builds a CPT for the root node of T by calling FillCPT (lines 13–14). It removes from C all trees which overlap with the leafs of T (line 15). It adds T to the forest F and removes the subtrees used in creating T from F so that they will not be reused (line 16). The While loop terminates when the candidate set C is empty (line 5).

Algorithm 3 Filling in the CPT of a Node

```
 1: Procedure FillCPT$(n)$
 2: for each tuple of values $\mathbf{x} \in (0,1)^k$ for parents of $n$ do
 3:   $\mathbf{y_n}[\mathbf{x}] \leftarrow$ concatenation of $\mathbf{y_{n_1}}, \dots, \mathbf{y_{n_k}}$, where, $\forall i$,
 4:     if $n_i$ is an input variable, then $\mathbf{y_{n_i}} = \mathbf{x_i}$
 5:     elseif $x_i = 0$ then $\mathbf{y}_{n_i} = \mathbf{z}(n_i)$
 6:     else $\mathbf{y}_{n_i} = \mathbf{d}(\mathbf{n_i})$
 7:   $P(n|Pa(n) = \mathbf{x})$ = EXP $(\mathbf{y_n}[\mathbf{x}], \mathbf{c}(\overline{\mathbf{n}}))$
 8: end for
 9: if $n = R$ then return
10: $\mathbf{d}(\mathbf{n}) \leftarrow \mathbf{y_n}[\text{argmax}_{\mathbf{x} \in (0,1)^k} \text{EXP}(\mathbf{y_n}[\mathbf{x}], \mathbf{c}(\overline{\mathbf{n}}))]$
11: $\mathbf{z}(\mathbf{n}) \leftarrow \mathbf{y_n}[\text{argmin}_{\mathbf{x} \in (0,1)^k} \text{EXP}(\mathbf{y_n}[\mathbf{x}], \mathbf{c}(\overline{\mathbf{n}}))]$
12: for each tuple of values $\mathbf{x} \in (0,1)^k$ for parents of $n$ do
13:   $P(n|Pa(n) = \mathbf{x}) \leftarrow \frac{P(n|Pa(n)=\mathbf{x}) - \text{EXP}(\mathbf{z}(\mathbf{n}), \mathbf{c}(\overline{\mathbf{n}}))}{\text{EXP}(\mathbf{d}(\mathbf{n}), \mathbf{c}(\overline{\mathbf{n}})) - \text{EXP}(\mathbf{z}(\mathbf{n})), \mathbf{c}(\overline{\mathbf{n}}))}$
14: end for
15: end FillCPT
```

The Repeat loop terminates when there is a single tree in F or there is a tree $T \in F$ which is rooted at R, in which case, it is returned (lines 18-19). The next 2 sections describe the components that learn the CPTs and score the tree structure in more detail.

Filling in the CPTs

If a node n and its parents are distinguishable with known context and control inputs, it is straightforward to fill in n's CPT using appropriate experimental queries. This is shown in Algorithm 3. The algorithm sets the parents of node n to each possible assignment vector $\mathbf{x}$ and infers and fills in $P(n|Pa(n) = \mathbf{x})$ (lines 2–8). The parents of node n are set to a desired bit vector $\mathbf{x} \in (0,1)^k$ by setting the input vector $\mathbf{y_n}[\mathbf{x}]$ to the concatenation of $\mathbf{y_{n_1}}, \dots, \mathbf{y_{n_k}}$ (line 3), which are defined in lines 4–6. If n_i is a leaf node, y_{n_i} is the same as x_i (line 4). Recall that we assume that the target tree is in CNF. Assuming that the control inputs $\mathbf{d}(\mathbf{n_i})$ and $\mathbf{z}(\mathbf{n_i})$ maximize $P(r|\mathbf{d}(\mathbf{n_i})) - P(r|\mathbf{z}(\mathbf{n_i}))$, we can set any non-leaf node n_i to $x_i = 0$ by setting its leafs y_{n_i} to $\mathbf{z}(\mathbf{n_i})$ (line 5), and to $x_i = 1$ by setting y_{n_i} to $\mathbf{d}(\mathbf{n_i})$ (line 6). After setting all inputs for the parent nodes, the algorithm sets the context of n according to its context input $\mathbf{c}(\overline{\mathbf{n}})$ and makes an EXP query on the result to infer $P(n|\mathbf{x})$ (line 7).

Unless the node n represents the root of the target (checked in line 9), its CPTs should be normalized. This is done by first finding the maximum and the minimum values in the CPT. The assignment that yields the maximum value is set to $\mathbf{d}(\mathbf{n})$ (line 10) and the assignment that yields the minimum value is set to $\mathbf{z}(\mathbf{n})$ (line 11). It then scales the distribution linearly using Equation 1 (lines 12–14).

Scoring the Tree Structures

We will now describe and justify the scoring function that tests subtree structures. Suppose that we are considering a candidate subtree rooted at node n, whose causal ancestors are the inputs A, where the other inputs are B. Recall that from the Chain Rule of Causal Effects (Theorem 1), the causal effect on R of changing A from $\mathbf{a}$ to $\mathbf{a}'$ in the context of $B = \mathbf{b}$, decomposes into the causal effect on n of changing $\mathbf{a}$ to $\mathbf{a}'$ and the causal effect on R of changing n from 0 to 1. Importantly, the first term does not depend on $\mathbf{b}$ and the second term does not depend on $\mathbf{a}$ or $\mathbf{a}'$. We now consider a similar equation for a different context $\mathbf{b}'$, and divide the former by the latter, yielding:

$$\begin{aligned}\frac{\Delta(r|\mathbf{a}\to\mathbf{a}',\mathbf{b})}{\Delta(r|\mathbf{a}\to\mathbf{a}',\mathbf{b}')} &= \frac{P(r|\mathbf{a}',\mathbf{b})-P(r|\mathbf{a},\mathbf{b})}{P(r|\mathbf{a}',\mathbf{b}')-P(r|\mathbf{a},\mathbf{b}')}\\ &= \frac{\Delta(r|n=0\to1,\mathbf{b})}{\Delta(r|n=0\to1,\mathbf{b}')} \qquad (2)\end{aligned}$$

Note that the left hand side of Equation 2 can be measured from observations of the target node and the right hand side suggests that it does not depend on $\mathbf{a}$ or $\mathbf{a}'$ if $P(r|\mathbf{a},\mathbf{b}') \neq P(r|\mathbf{a}',\mathbf{b}')$. Such input $\mathbf{b}'$ exactly corresponds to the context input $\mathbf{c}(\overline{n})$, while $\mathbf{a}$ and $\mathbf{a}'$ correspond to the control inputs $\mathbf{z}(n)$ and $\mathbf{d}(n)$. If such context and control inputs are found, then we can test if Equation 2 holds for any pairs $\mathbf{b},\mathbf{b}'$ by checking if the ratio is invariant under all pairs of $\mathbf{a},\mathbf{a}'$. In particular, we let $\mathbf{b}'=\mathbf{c}(\overline{\mathbf{n}})$, set $\mathbf{a}$ according to $\mathbf{z}(\mathbf{n})$ and vary $\mathbf{a}'$ to deterministically generate all possible assignments in $\{0,1\}^{\leq k}$ at the parents of node n using the control inputs for these nodes. Unfortunately it is not feasible to generate all possible assignments to variables in B because there could be too many of them. Hence we choose these inputs from a random sample S chosen according to the natural distribution P. We now present the scoring function of the candidate subtree T as a function of the sample S and the control and context inputs $\mathbf{d}(n)$, $\mathbf{z}(n)$ and $\mathbf{c}(\overline{n})$:

$$f(T,S) = \max_{\mathbf{b}\in S\neq\mathbf{c}(\overline{n})}\left\{\max_{\mathbf{a}'\neq\mathbf{z}(n)}\frac{\Delta(r|\mathbf{z}(\mathbf{n})\to\mathbf{a}',\mathbf{b})}{\Delta(r|\mathbf{z}(\mathbf{n})\to\mathbf{a}',\mathbf{c}(\overline{\mathbf{n}}))} - \min_{\mathbf{a}'\neq\mathbf{z}(n)}\frac{\Delta(r|\mathbf{z}(\mathbf{n})\to\mathbf{a}',\mathbf{b})}{\Delta(r|\mathbf{z}(\mathbf{n})\to\mathbf{a}',\mathbf{c}(\overline{\mathbf{n}}))}\right\}$$

Ideally the above f-score will be 0 for real subtrees of the target tree. We use a small upperbound $\theta > 0$ (set to 10^{-5} in our experiments) to allow for roundoff errors. If the test fails, it means that n does not d-separate A from the root and hence is not a valid subtree.

Sample Complexity Analysis

While the uniform convergence results of (Dasgupta 1997; Haussler 1992; Pollard 1984) can be used to derive the sample complexity of our algorithm, they are applicable to any algorithm that learns a consistent causal model, and are quite loose. Here we derive tighter bounds which are specific to our algorithm. We first need the following *faithfulness assumption* on the target causal network and the distribution of data generated by it.

Definition 4 *The target causal network is α-*faithful *to the natural distribution $D(\mathbf{x})$ and the conditional distribution $P(r|\mathbf{x})$ if there is a probability threshold $\alpha > 0$ such that for any latent subtree and a potential conditional independence relation $\mathcal{I}$ that is* not *entailed by the target causal network, the probability of P generating an instance $\mathbf{x}$ that violates $\mathcal{I}$ is at least α.*

The *faithfulness assumption* guarantees that with large enough data bad subtrees will be eventually discovered by violations of conditional independence relationships. Without this assumption, several bad subtrees may be learned at lower levels, which could lead to other bad subtrees at higher levels with potentially exponential blow up. We are now ready to state and prove our main result.

Theorem 3 *Let the true causal model be represented as a tree of p leaf nodes, q non-leaf nodes and degree k. Let the target causal model be α-faithful and the* EXP *oracle answers the experimental queries exactly. Let* BuildTree *be called on a random input sample of size $m > \max(\frac{q}{\epsilon}\ln\frac{2q}{\delta}, \frac{1}{\alpha}\ln\frac{2(q+k\ln p)}{\delta})$ chosen using an arbitrary but fixed natural distribution P. Then with probability at least $\geq 1-\delta$ it outputs a causal model whose probability of non-zero error on a random test example is at most ϵ. Its query and time complexity are polynomial in $q, p, \frac{1}{\alpha}, \frac{1}{\delta}$, and $\frac{1}{\epsilon}$.*

Proof: Note that the algorithm only uses the random sample to find the distinguishing contexts and to rule out bad subtrees. Hence, there are the following 2 types of imperfections in any tree learned by the algorithm.

1. **Decoy Subtrees**. The learned tree contains one or more subtrees that do not exist in the target tree.
2. **Non-distinguishability**. One or more subtrees of the target tree are absent in the learned tree due to non-distingushability of their root nodes.

Call a causal tree model "bad" if it contains a decoy subtree or the probability of error on a random example chosen using P due to absent subtrees is $\geq \epsilon$. Otherwise it is "good." Hence the probability of a good tree mispredicting on a random test example is $< \epsilon$. We should show that the probability of learning a bad causal tree $< \delta$ when trained on the above sample size. We consider each of the 2 types of errors in that order.

First, the only way to have a set of decoy subtrees in the learned output is when all training data are consistent with the corresponding conditional independence relationships. By α-faithfulness, the probability of a training example detecting a non-existent conditional independence relationship is $\geq \alpha$. Hence, the probability that m i.i.d. random examples are not able to detect a specific subtree is at most $(1-\alpha)^m$. At most p^k subtrees are considered at the lowest level of the tree by Algorithm 2. The number of subtrees considered reduces in each iteration for at most q iterations, giving an upper bound of qp^k subtrees. By union bound, the probability that some subtree goes undetected is upper bounded by $qp^k(1-\alpha)^m < qp^k e^{-\alpha m}$.

Now we will consider the second type of error due to non-distinguishability. For a random test example to fail with a probability $\geq \epsilon$ due to this reason, at least one of the q internal nodes, say node i should not have been distinguished with probability $\geq \frac{\epsilon}{q}$. Since the training and testing distributions are the same, this implies that the probability of a training example distinguishing i must be $\geq \frac{\epsilon}{q}$. The probability that an example not distinguishing an internal node $< 1-\frac{\epsilon}{q}$. The probability that m i.i.d. examples not distinguishing that node $< (1-\frac{\epsilon}{q})^m < e^{-\frac{m\epsilon}{q}}$. The probability that m i.i.d. examples not distinguishing *some* internal node $< qe^{-\frac{m\epsilon}{q}}$ due to union bound.

Hence the probability of learning a bad causal model due to either type of error $< qe^{-\frac{m\epsilon}{q}} + qp^k e^{-m\alpha}$. To keep it less than δ, we enforce $qe^{-(\frac{m\epsilon}{q})} < \frac{\delta}{2}$ and $qp^k e^{-m\alpha} < \frac{\delta}{2}$, which will be satisfied by $m > \max(\frac{q}{\epsilon}\ln\frac{2q}{\delta}, \frac{1}{\alpha}\ln\frac{2(q+k\ln p)}{\delta})$.

BuildTree makes at most $m + 2mqp^k + qp^k 2^k$ experimental queries in the worst case. The first term is for the EXP query on each sample which is pre-computed. The second term accounts for lines 7 and 10 of Algorithm 2. Each line loops over the m samples for p^k iterations of the While loop and q iterations of the Repeat loop. The last term is due to filling the CPTs of the roots of all subtrees generated (line 7 of Algorithm 3). Lines 10,11 and 13 do not generate additional queries since they can be folded into the For loop. The time complexity is bounded by $O(qp^{k+1}(2^k + m))$, since the time is dominated by the experimental queries and each query takes $O(p)$ time to setup. $\square$

Experimental Results

Unfortunately the worst-case PAC bounds tend to be very pessimistic in general and are not useful to predict the learning curves. To examine the empirical performance of our algorithm, we evaluated it on datasets generated from synthetic target causal trees. Each is a full binary tree with depth 4 and 16 leaf nodes. At each node we randomly selected a CPT, where each probability in the table is chosen independently from a uniform distribution in the union of the two intervals $(0.0, 0.1)$ and $(0.9, 1.0)$. The extreme probabilities are chosen to make the learning problem challenging. The input example distribution P is uniform. During the training of each batch, the algorithm also asks experimental queries, which are answered by referring to the target tree. When a query is answered, its response is stored so that it is never asked again. After each batch of training the learned tree was evaluated on 40 test examples which were chosen randomly from the pool of examples not queried during training.

We report in Table 1 the average results of 30 different training sets on 10 target trees, one per each row. The first two columns are based on an algorithm that knows the tree structure. It is identical to Algorithm 2 with two differences: (1) the candidate list of trees constructed in line 4 will only include the correct subtrees at the next level, and (2) the test "if $f(T, S) < \theta$ " in line 13 is removed and treated as true since we are only considering correct subtrees. The last 3 columns are based on our algorithm. Columns 1 and 3 report the number of average unique queries asked on each target tree in the two scenarios. To achieve comparable test error rates, the algorithm was run with 5 random examples when the tree is not given and 1 random example when the tree is given. The results show that approximately 10 times as many unique queries are asked when the tree is learned compared to when it is given.

The last column reports the fraction of training sets in which there were one or more errors in the learned tree structure. In two of the target trees, 1 out of 30 training sets resulted in an error. In one target tree, 3 out of 30 training sets gave an error. Column 4 reports the average absolute error

Tree structure is given		Tree structure is learned		
Queries	Test Error	Queries	Test Error	Errors
45.47	4.705E-14	486.13	9.950E-17	0/30
45.40	3.791E-15	483.00	1.339E-16	0/30
44.87	4.876E-15	482.73	2.328E-16	0/30
46.03	2.194E-15	494.83	2.094E-16	0/30
44.33	3.301E-16	462.90	1.237E-16	0/30
45.73	2.218E-15	474.80	0.0199E0	1/30
44.77	3.746E-16	465.63	0.0151E0	1/30
44.53	1.485E-14	484.93	1.372E-16	0/30
44.47	4.408E-16	467.73	1.290E-16	0/30
45.03	7.252E-16	483.93	0.0112E0	3/30

Table 1: Mean number of unique queries asked and mean test error on 10 different target trees. The last column shows the fraction of trials where the learned tree was imperfect.

in the conditional probabilities of the target variable on the test set. The errors are near-zero except for the 3 target trees where the learned trees are imperfect. Column 2 reports the errors when the tree is given, which are always near-zero. The results show both the effectiveness of our learning algorithm in robustly learning causal trees, as well as the significant reduction in sample and query complexities when the tree structure is given. It also shows that a very small number of random examples is sufficient and the query complexity is dominated by structure learning.

Conclusions and Future Work

While the informational need for experiments to identify the causal structure is well-known, their role in the computational efficiency of learning is not as well-studied. We presented an efficient algorithm to PAC-learn causal trees with latent variables from examples and directed experiments. Without the experiments, learning the trees from random examples alone appears to be intractable. Random observational examples are also needed because without them, the trees can encode arbitrary passwords so that it takes exponentially many guesses to get useful information in the worst case. Our algorithm can be generalized to exactly identifying the causal tree from counterexamples and experiments by rerunning it whenever a new counterexample is discovered. This framework generalizes Angluin's model of exact learning from counterexamples and membership queries (Angluin 1987) to stochastic functions. Our algorithm assumes that the EXP oracle is exact and the true model is a tree. Generalizations to approximate EXP oracles and agnostic learning are important future directions. Extending the algorithm to more general causal structures and non-binary variables offers some more challenges. Another direction is to build a simpler causal tree that approximates the predictions of a more complicated network and can be said to *explain* its behavior (Shaughnessy et al. 2020).

Acknowledgments

The authors acknowledge the support of DARPA under contract N66001-17-2-4030 and NSF under grant IIS-1619433.

References

Angluin, D. 1987. Queries and Concept Learning. *Machine Learning* 2(4): 319–342.

Bello, K.; and Honorio, J. 2017. Learning Bayes networks using interventional path queries in polynomial time and sample complexity URL http://arxiv.org/abs/1706.00754.

Bshouty, N. H.; Hancock, T. R.; and Hellerstein, L. 1995. Learning Boolean read-once formulas over generalized bases. *Journal of Computer and System Sciences* 50(3): 521–542.

Colombo, D.; Maathuis, M. H.; Kalisch, M.; and Richardson, T. S. 2012. Learning High Dimensional Direct Acyclic Graphs with Latent and Selection Variables. *The Annals of Statistics* 40(1): 294–321.

Cooper, G. F.; and Yoo, C. 1999. Causal discovery from a mixture of experimental and observational data. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, 116–125. Morgan Kaufmann Publishers Inc.

Dasgupta, S. 1997. The Sample Complexity of Learning Fixed Structure Bayesian Networks. *Machine Learning* 29: 165–180.

Eaton, D.; and Murphy, K. 2007. Exact Bayesian structure learning from uncertain interventions. In *Artificial Intelligence and Statistics*, 107–114.

Hauser, A.; and Bühlmann, P. 2014. Two optimal strategies for active learning of causal models from interventional data. *International Journal of Approximate Reasoning* 55(4): 926–939.

Haussler, D. 1992. Decision Theoretic Generalizations of the PAC Model for Neural Net and Other Learning Applications. *Inf. Comput.* 100(1): 78–150.

He, Y.-B.; and Geng, Z. 2008. Active Learning of Causal Networks with Intervention Experiments and Optimal Designs. *JMLR* 9: 2523–2547.

Hyttinen, A.; Hoyer, P. O.; Eberhardt, F.; and Järvisalo, M. 2013. Discovering Cyclic Causal Models with Latent Variables: A General SAT-Based Procedure. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Kearns, M. J.; and Schapire, R. E. 1994. Efficient Distribution-Free Learning of Probabilistic Concepts. *J. Comput. Syst. Sci.* 48(3): 464–497.

Kocaoglu, M.; Shanmugam, K.; and Bareinboim, E. 2017. Experimental Design for Learning Causal Graphs with Latent Variables. In *Advances in Neural Information Processing Systems 30*, 7018–7028.

Li, G.; and Leong, T.-Y. 2009. Active Learning for Causal Bayesian Network Structure with Non-symmetrical Entropy. In *PAKDD*, 290–301. Springer-Verlog.

Pearl, J. 2009. *Causality*. Cambridge university press.

Pearl, J. 2019. The seven tools of causal inference, with reflections on machine learning. *Commun. ACM* 62(3): 54–60.

Pearl, J.; and Mackenzie, D. 2018. *The Book of Why: The New Science of Cause and Effect*. Basic Books.

Peters, J.; Bühlmann, P.; and Meinshausen, N. 2016. Causal inference by using invariant prediction: identification and confidence intervals. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 78(5): 947–1012.

Pitt, L.; and Valiant, L. G. 1988. Computational limitations on learning from examples. *J. ACM* 35(4): 965–984.

Pitt, L.; and Warmuth, M. K. 1990. Prediction-Preserving Reducibility. *J. Comput. Syst. Sci.* 41(3): 430–467.

Pollard, D. 1984. *Convergence of Stochastic Processes*. New York, Berlin: Springer Verlog.

Reichenbach, H. 1991. *The direction of time*, volume 65. Univ of California Press.

Schapire, R. E. 1994. Learning Probabilistic Read-once Formulas on Product Distributions. *Machine Learning* 47–81.

Shaughnessy, M.; Canal, G.; Connor, M.; Rozell, C.; and Davenport, M. 2020. Generative causal explanations of black-box classifiers. In *Advances in Neural Information Processing Systems*, volume 33, 5453–5467. Curran Associates, Inc.

Silander, T.; and Myllymaki, P. 2012. A simple approach for finding the globally optimal Bayesian network structure. *arXiv preprint arXiv:1206.6875* .

Spirtes, P.; Glymour, C.; and Scheines, R. 2000. Causation, prediction, and search. Adaptive computation and machine learning.

Tadepalli, P. 1993. Learning from Queries and Examples with Tree-structured Bias. In *Proceedings of the Tenth International Machine Learning Conference*, 322–329.

Tadepalli, P.; Barrie, C.; and Russell, S. J. 2019. Learning Causal Trees with Latent Variables via Controlled Experimentation. In *AAAI Spring Symposium on Beyond Curve Fitting: Causation, Counterfactuals, and Imagination-Based AI*.

Tadepalli, P.; and Russell, S. J. 1998. Learning from Examples and Membership Queries with Structured Determinations. *Machine Learning* 32(3): 245–295.

Tong, S.; and Koller, D. 2001. Active Learning for Structure in Bayesian Networks. In *IJCAI*, 863–869.