Reliable and Efficient Anytime Skeleton Learning

Rui Ding,[1] Yanzhi Liu,[2*] Jingjing Tian,[3*] Zhouyu Fu,[4] Shi Han,[1] Dongmei Zhang[1]**

[1]Microsoft Research [2]Beijing University of Posts and Telecommunications [3]Peking University [4]Alibaba Group

[1]{juding, shihan, dongmeiz}@microsoft.com, [2]lyztyj@bupt.edu.cn, [3]tianjj97@pku.edu.cn, [4]zhouyu.fz@alibaba-inc.com

Abstract

Skeleton Learning (SL) is the task for learning an undirected graph from the input data that captures their dependency relations. SL plays a pivotal role in causal learning and has attracted growing attention in the research community lately. Due to the high time complexity, anytime SL has emerged which learns a skeleton incrementally and improves it overtime. In this paper, we first propose and advocate the *reliability* requirement for anytime SL to be practically useful. Reliability requires the intermediately learned skeleton to have precision and persistency. We also present REAL, a novel Reliable and Efficient Anytime Learning algorithm of skeleton. Specifically, we point out that the commonly existing Functional Dependency (FD) among variables could make the learned skeleton violate faithfulness assumption, thus we propose a theory to resolve such incompatibility. Based on this, REAL conducts SL on a reduced set of variables with guaranteed correctness thus drastically improves efficiency. Furthermore, it employs a novel edge-insertion and best-first strategy in anytime fashion for skeleton growing to achieve high reliability and efficiency. We prove that the skeleton learned by REAL converges to the correct skeleton under standard assumptions. Thorough experiments were conducted on both benchmark and real-world datasets demonstrate that REAL significantly outperforms the other state-of-the-art algorithms.

Introduction

Skeleton Learning (SL) plays a pivotal role in causal learning and knowledge discovery and has attracted growing attention in the research community lately (Shanmugam et. al. 2015; Peters, Janzing and Schölkopf 2017, Bühlmann and Meinshausen 2016). It is the task for learning an undirected graph from the input data that captures the variables' dependency relations. Under causal sufficiency assumption (i.e., the set of observed variables $\mathcal{V}$ include all the common causes of pairs in $\mathcal{V}$), each edge A–B in a skeleton has causal semantic: there exists direct cause-effect relationship between A and B but the direction is unknown. Thus for causal learning, SL is the primary step for performing interventions over directly linked variables in skeleton to discover cause effects (Kocaoglu, Dimakis and Vishwanath 2017; Lindgren et. al. 2018). In addition, SL has great potential in knowledge discovery (Chen et al., 2012). For example, an engineer tries to diagnose the causes of service degradation and how its impact propagates.

SL has a high computational complexity (Chickering 1996). The cost of waiting till the final solution is often unacceptable for practical tasks that require decisions to be made within given time budget, e.g., 30 minutes for investigating a cloud service degradation (Lou et al., 2017). On the other hand, a partially learned skeleton can also provide important clues for obtaining local causal relations (Pearl 2009) or for dependency analysis (Daly et al., 2006). This motivates the development of *anytime* SL techniques to incrementally learn a skeleton and improve it over time.

Two challenges need to be addressed to make anytime SL practically useful. One is *reliability* and the other is *efficiency*. The reliability challenge entails two requirements – *precision* and *persistency*. Precision means that the partially learned skeleton $\mathrm{S}(t)$ at any time t should contain only trustable edges that correctly encode the dependencies between variables. Persistency means that the skeleton $S(t+1)$ updated at time $t+1$ should remove as few as possible edges from previously learned skeleton $S(t)$. This enables users to reuse their analysis models obtained from the previous step over newly added edges rather than start from scratch. To the best of our knowledge, the reliability issue was not addressed in the existing work.

Another major source of complication for SL stems from the inter-variable Functional Dependency (Vardi 1987) in real-world data. For FD (i.e., short for FD relationship) between two variables, the value of one variable can be fully determined by the other. For instance, *Country* determines *Continent*, or *Date* determines *Year*. Using terms from relational database domain, FDs typically exist between the primary key and other non-key variables within a table. Thus, FD is prevalent in tabular datasets composed by joining multiple tables. We notice that certain FDs

* The contributions by Yanzhi Liu and Jingjing Tian have been conducted and completed during their internships at Microsoft Research Asia.

** The contributions by Zhouyu Fu have been conducted and completed during his employment at Microsoft Research Asia.

could make the learned skeleton violate faithfulness assumption. Faithfulness (Pearl, 2009) is an important assumption that has been widely adopted in many independence-based approaches for BNSL (Spirtes et. al. 2001; Tsamardinos, Brown and Aliferis 2006). In our problem, to deal with such incompatibility, we suggest a weaker assumption called *harmony* (i.e., the data distribution should satisfy Global Markov Property with respect to the learned graph *G* and *G* should be minimum) instead of faithfulness for SL, thus a theory for harmonious SL must be developed.

In this paper, we present REAL, a novel Reliable and Efficient Anytime skeleton Learning algorithm to achieve high reliability and high efficiency. Given FDs as input, we first provide a theory applied to FDs to create an FD induced graph G_{FD}, and identify a reduced set of variables for subsequent skeleton learning algorithm. We prove that the learned skeleton following this way is guaranteed to be harmonious, thus both reliability and efficiency are significantly improved. In contrary, existing SL algorithms do not take FD into consideration, which is more time consuming and is also prone to generate spurious edges in the learned skeleton.

To further achieve high reliability, REAL exploits a bottom-up skeleton growing strategy for anytime SL. Specifically, REAL starts from an empty graph, and incrementally adds edges into the skeleton by a robust local evaluation procedure called RobustPCFinder. RobustPCFinder aims to find candidates of neighbors for each target variable T with low false positives. Furthermore, due to the property of RobustPCFinder, the growing of local neighbors for each variable is irrelevant to the order of variables prioritized, we thus propose a best-first search strategy to effectively utilize the time budget by selecting variable with lowest overhead (i.e., time cost for local evaluation) first. When timeout or user interrupts, skeleton growing is stopped, and a lightweight post-processing is conducted to further remove false positives. In summary, we have made the following contributions.

1. To the best of our knowledge, we are the first to propose *reliability* (i.e., *precision* and *persistency*) as an essential requirement for building anytime SL algorithms to be practically useful.
2. We provide a theory to resolve incompatibility between FD and faithfulness, which significantly improves the reliability and efficiency of the skeleton to be learned for datasets with FDs.
3. We propose a novel anytime skeleton growing algorithm by edge-insertion and best-first strategy to achieve high reliability and efficiency. We prove that the skeleton learned by REAL converges to the correct skeleton under standard assumptions.
4. We conduct thorough evaluations on benchmark and real-world datasets to demonstrate that REAL significantly outperforms the other state-of-the-art algorithms.

Approach

Preliminaries

A dataset D consists of N records and d categorical columns, which represents N instances drawn i.i.d. from d discrete variables $\mathcal{V} = \{V_1, V_2, \cdots, V_d\}$ by a joint probability distribution $P_{\mathcal{V}}$. Denote the cardinality of V_i (i.e., number of distinct values in the *i*-th column of *D*) as c_i, and $C = \{c_1, c_2, \cdots, c_d\}$ is the cardinality set. To avoid degeneracy, we restrict $c_i > 1, \forall i$.

Markov Factorization Property. Given a distribution $P_{\mathcal{V}}$ and a Directed Acyclic Graph (DAG) G, $P_{\mathcal{V}}$ is said to satisfy Markov factorization property or *Markovian* (with respect) to G if $P_{\mathcal{V}} := P(V_1, V_2, \cdots, V_d) = \prod_{i=1}^{d} P(V_i|pa_i^G)$, where pa_i^G is the parent set of V_i in G.

Markov factorization property is the basis of graphical models, it encodes conditional independences in the distribution that we can exploit for learning graph structure.

Global Markov Property (GMP). A distribution $P_{\mathcal{V}}$ is said to satisfy Global Markov Property or GMP (with respect) to a DAG G if $X \perp_G Y|Z \Rightarrow X \perp Y|Z$. Here $\perp_G$ denotes d-separation, and $\perp$ denotes statistical independence.

GMP indicates that any d-separation in graph *G* implies conditional independence in distribution $P_{\mathcal{V}}$. GMP is equivalent to Markov factorization property (Lauritzen 1996).

Minimality. A distribution $P_{\mathcal{V}}$ satisfies minimality (with respect) to DAG *G* if it is Markovian to *G*, but not proper to any subgraph of *G*.

Minimality coincides with Occam's Razor principle (Pearl 2009): when distribution $P_{\mathcal{V}}$ is Markovian to both two graphs *G* and *G'*, we prefer to use the simpler one to interpret the data. In fact, minimality is a necessary constraint for DAG learning. Without minimality constraint, we can always construct a trivial graph G' by setting $pa_i^{G'} = \{V_1, \dots V_{i-1}\}$, so that $P_{\mathcal{V}}$ is always Markovian to G' because $P(V_1, V_2, \cdots, V_d) \equiv \prod_{i=1}^{d} P(V_i|V_1 \sim V_{i-1}) = \prod_{i=1}^{d} P\left(V_i|pa_i^{G'}\right)$.

Faithfulness. a distribution $P_{\mathcal{V}}$ is faithful (with respect) to a DAG *G* if $X \perp Y|Z \Leftrightarrow X \perp_G Y|Z$.

Faithfulness implies minimality (Peters, Janzing and Schölkopf 2017). In practice faithfulness is a strong assumption that could be violated in different ways. In our problem, faithfulness can be violated when data contains FDs.

Approach Overview

As depicted in Figure 1, the raw data is first fed into the module called FD Solver, which outputs a simplified FD graph G'_{FD} and a subset of variables $\mathcal{V}'$ for skeleton learning module. FD Solver first identifies FDs and then resolves the incompatibility between FDs and faithfulness assumption by outputting G'_{FD} and $\mathcal{V}'$. $\mathcal{V}'$ is ensured to contain no FDs. Then SL is conducted on $\mathcal{V}'$ by incremental skeleton-growing which is designed in an anytime fashion that whenever time is expired (i.e., either by user interruption or timeout), the partially learned skeleton is output to

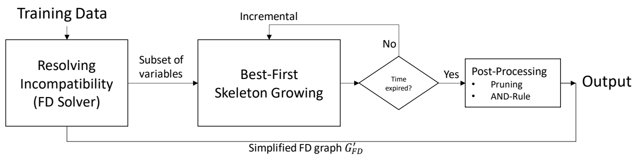

Figure 1. Overview of REAL's workflow

the post-processing module. Post-processing module further removes false-positives and then return the best-so-far skeleton to user. FD Solver runs fast with time complexity $O(Nd^2)$. Skeleton growing module is time consuming, thus it is designed in an incremental manner. Starting from an empty graph, it iteratively selects a variable and tries to insert an edge on it. The variable selection is performed in a best-first way, so that the time budget can be effectively utilized. Post-processing is lightweight. Below we illustrate the details of each part.

FD Solver

Definition 1. A functional dependency $X \xrightarrow{FD} Y$ means that the value of Y is determined by the value of X. The specific mapping is denoted as $Y = f(X)$, where Y is a discrete variable and X is a set of discrete variables.

FD (Vardi 1987) typically exists between the primary key and other non-key variables within a table. Thus, FD is prevalent in tabular datasets composed by joining multiple tables, which reflects consistent or deterministic inter-variable relationship, such as Country$\xrightarrow{FD}$ Continent. In our problem, we restrict FD $X \xrightarrow{FD} Y$ to the case where $|X| = 1$ (i.e., X is a single variable), because it is the most common FD relationship (i.e., primary key is a single column) and detecting such kind of FDs is lightweight. FDs with $|X| > 1$ are also useful for SL, but they are rarer and time cost for detecting such cases is exponential to $|X|$, which is unsuitable for our problem.

Intuitively, given $X \xrightarrow{FD} Y$, X and Y naturally exhibit strong dependency. Furthermore, conditioning on X would screen-off Y with other variables since Y becomes single-valued. Thus FD has inherent connection to SL. These intuitions can be described in the following two lemmas (NOTE: considering page limited, ALL proofs for lemmas and theorems are available at our website[1]):

Lemma 1. If $X \xrightarrow{FD} Y$, then $Y \nparallel X$; for any other variable set Z, $Z \perp Y|X$. Here symbol $\nparallel$ denotes dependent (i.e., two variables are not statistically independent). ■

Lemma 2. If $X \xrightarrow{FD} Y$ and $Z \perp X|W$, then $Z \perp Y|W$. Where Z and W are disjoint sets of variables other than X and Y. ■

FDs have the following properties:

Transitivity: if $X \xrightarrow{FD} Y$ and $Y \xrightarrow{FD} Z$, then $X \xrightarrow{FD} Z$

Equivalence: If $X \xrightarrow{FD} Y$ and $Y \xrightarrow{FD} X$, then X is one-to-one correspondence of Y, denoted as $X \leftrightarrow Y$.

FD equivalent variables are indistinguishable for skeleton learning tasks. If a dataset contains FD equivalent groups, we will only keep one variable from each FD equivalent group by pre-processing. We assume no variables are FD equivalent in subsequent discussion.

Definition 2. (FD induced graph). Given a dataset D with n variables $X_1{\sim}X_n$, an FD induced graph G_{FD} is a directed graph with n variables $X_1{\sim}X_n$, by assigning a directed edge from X_i to X_j whenever $X_i \xrightarrow{FD} X_j$ in D.

G_{FD} is a set of connected components, where each component is a DAG. This is ensured since we have eliminated FD equivalent variables at preprocessing stage.

Violation of Faithfulness Induced by FDs

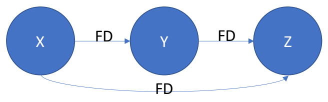

Figure 2. Violation of faithfulness by simple FD structure

When a dataset contains FDs, the faithfulness assumption can be violated: Figure 2 shows a G_{FD} of 3 variables X, Y and Z. According to lemma 1, $P_{\mathcal{V}}$ implies two conditional independencies: $Z \perp Y|X$ and $Z \perp X|Y$. If faithfulness assumption is hold, then $\{Z \perp Y|X;\ Z \perp X|Y\} \Rightarrow \{Z \perp_G Y|X;\ Z \perp_G X|Y\} \Rightarrow$\{no edge between Z and Y; no edge between Z and X\}$\Rightarrow Z \perp_G Y \Rightarrow Z \perp Y \Rightarrow$ contradiction: because $Y \xrightarrow{FD} Z \Rightarrow Y \nparallel Z$ by lemma 1. Therefore, if we take faithfulness assumption to deal with variables have FDs, we could fail to get a DAG that is Markovian to $P_{\mathcal{V}}$.

Skeleton Learning with FDs

As illustrated above, the issue is due to faithfulness can be invalid when data contains FDs. On the other hand, GMP and minimality are weaker assumptions without such issue. We thus define *harmonious*:

Definition 3 (harmonious). A DAG G is said to be harmonious with respect to a distribution $P_{\mathcal{V}}$ if $P_{\mathcal{V}}$ satisfies both GMP and minimality to G.

Now we formulate our graph structure learning as follows:

[1] https://www.microsoft.com/en-us/research/project/real/

FD-DAG Problem (DAG Learning with FDs). Given a dataset D, corresponding distribution $P_{\mathcal{V}}$, and the FD induced graph G_{FD} of D, given that faithfulness assumption is only valid for variables containing no FDs. Find a DAG G such that G is harmonious to $P_{\mathcal{V}}$.

Solving FD-DAG problem also solves the problem of skeleton learning since skeleton is the undirected version of G. Now we propose a theory to solve FD-DAG problem.

Solving FD-DAG Problem

We analyze how to obtain edges in the learned DAG without conducting learning when data contains FDs. Our analysis starts from two typical structures of G_{FD}.

Single-Root Structure

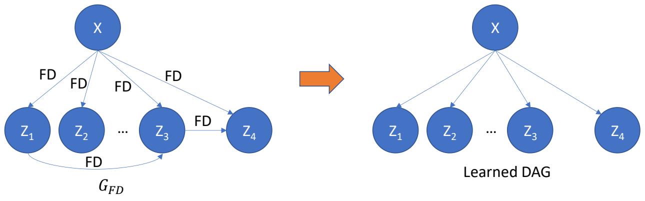

Figure 3. Example of single-root structure

If G_{FD} has only one root variable (i.e., variable with no parents), we say G_{FD} is a single-root structure. An example is shown at left-hand side of Figure 3, where X is the single root variable.

Theorem 1. If G_{FD} of variables $\{X, Z_1, \dots, Z_n\}$ is a single-root structure with root X, then a harmonious DAG G is a single-root structure with root X, all the other variables are directly linked from X, but no other edges exist in G. ■

Single-Sink Structure

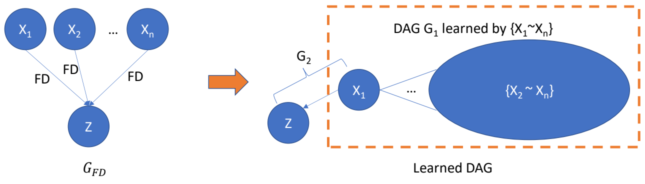

Figure 4. Example of single-sink structure

If G_{FD} of variables $\{X_1, \dots, X_n, Z\}$ has only one sink variable Z (i.e., variable with no children) and all the other variables X_i are root variables linked to Z, we say G_{FD} is a single-sink structure. An example is shown at left-hand side of Figure 4, where Z is the single sink variable.

Lemma 3. Suppose $X_1 \to X_2$ in a DAG G, and no more edges link X_2 with other variables. If G is Markovian to a distribution $P_{\mathcal{V}}$, then $X_2 \perp_G Y | U \Rightarrow X_1 \perp_G Y | U, \forall Y \in \{X_3, \dots X_n\}, \forall U \subset \{X_3, \dots X_n\}, Y \notin U$. ■

Theorem 2. If G_{FD} of variables $\{X_1, \dots, X_n, Z\}$ is a single-sink structure with sink Z, then a harmonious DAG G is union of two graphs G_1 and G_2, where G_1 is a DAG learned by any sound and complete DAG-learn algorithm over $\{X_1, \dots, X_n\}$, and G_2 is $X_1 \to Z$. ■

The right part of Figure 4 is an example of theorem 2.

General Structure

The analysis on single-root and single-sink structures suggest two ways to simplify the FD-DAG problem in general case. In G_{FD}, when a set of variables share a common root X (i.e., single-root structure), theorem 1 suggests these variables are directly linked from X in the learned DAG G; When a variable Z can be determined by multiple roots (i.e., single-sink structure), theorem 2 suggests we can just pick an arbitrary root X_1, links Z from X_1, and disconnect Z to any other variables in the learned DAG G. Below we show these suggestions are indeed correct in general. We first define "simplified FD graph":

Definition 4 (simplified FD graph). In G_{FD} over $n + m$ variables, denote the root variables as $\{X_1, \dots X_n\}$, and the other variables as $\{Z_1, \dots, Z_m\}$. A simplified FD graph G'_{FD} is a subgraph of G_{FD} that each Z_i only preserves one edge from one root that links to it, and removes all the other edges. Figure 5 shows an example.

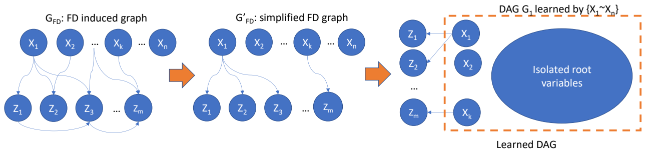

Figure 5. Example of general structure and simplified FD graph

Theorem 3. In G_{FD} over variables $\{X_1, \dots, X_n, Z_1, \dots, Z_m\}$, denote the root variables as $\{X_1, \dots, X_n\}$, denote the simplified FD graph as G'_{FD}. Then a harmonious DAG G is union of two graphs G_1 and G'_{FD}, where G_1 is a DAG learned by any sound and complete DAG-learn algorithm over root variables $\{X_1, \dots, X_n\}$. ■

Uniqueness

According to theorem 2, when there are multiple roots share same sink in G_{FD}, the learned harmonious DAG is not unique. Below we show that when a variable Z has only one parent X in G_{FD}, then the edge between X and Z exists in every harmonious DAG.

Theorem 4 (Uniqueness). If a variable Z can only be functionally determined by another variable X, then the edge from X to Z exists in every harmonious DAG G. ■

The uniqueness property provides convenience for experimental evaluation. We use these unique edges to check if a skeleton-learn algorithm can identify them or not.

Variables Reduction

Theorem 3 shows that when data contains FDs, we can use a subset of variables for subsequent SL algorithm, without loss of correctness, thus reduce the computational cost.

Corollary 1. Given G_{FD}, the set of variables used for subsequent SL is the set of root variables in G_{FD}, these variables do not have FDs with each other. ■

It is worthy to mention that, FD aided SL generates more stable skeleton than flat approach, because CI (conditional independence) tests based or score-based DAG-learn algorithm cannot guarantee to find correct FD edges (e.g., FD edges with uniqueness property), but FDs typically reflect stable and consistent inter-variable knowledge.

Best-First Skeleton Learning

REAL's skeleton learning strategy is in a growing manner which starts from an empty graph and incrementally inserting edges. The newly added edges should contain few false positives. To accommodate these considerations, REAL maintains a variable set $RawPC[T]$ to store each variable T's raw PCs (i.e., set of parents and children before post-processing). Once the PC growing for T is complete, $RawPC[T]$ is pruned to get $PrunedPC[T]$. The output $PC[T]$ is based on $PrunedPC[T]$ by applying AND-Rule to appropriate variable pairs.

```
Procedure RobustPCFinder
Input: $T$, and its $RawPC$
Output: A new variable most likely to be a $PC$ of $T$
1: $\mathcal{V}' = \mathcal{V} \backslash \{RawPC \cup \{T\}\}$
2: $r^* \leftarrow \max_{V \in \mathcal{V}'} RobustCorr(V, T, RawPC)$
3: if $r^*$ is null
4:     return null
5: $V^* \leftarrow$ argmax$_{V \in \mathcal{V}'}$ $RobustCorr(V, T, RawPC)$
6: return $\{V^*, r^*\}$
```

```
Sub-Procedure RobustCorr
Input: $T$, and its current $RawPC$, $V$
1: $R \leftarrow \emptyset$
2: foreach $S \subseteq RawPC$ do
3:     $r \leftarrow Corr(V, T|S)$
4:     if $r < threshold$
5:         return null
6:     $R \leftarrow R \cup \{r\}$
7: return kth percentile of $R$
```

Figure 6. Procedure of RobustPCFinder

Adding an edge. Once a variable T is selected, $RawPC[T]$ is grown by iteratively adding a new, most-likely PC candidate by calling RobustPCFinder (see left part of Figure 6). A sub-procedure RobustCorr is used to measure the "likelihood" of a variable V to be a PC candidate of T (right part of Figure 6). If there exists independence between V and T by conditioning on a subset of current $RawPC[T]$, which indicates V cannot be a PC of T (an implication if we assume faithfulness), thus V is discarded (line 2~5 in RobustCorr); otherwise, RobustCorr returns the kth percentile among the collection of correlations between V and T by conditioning on all possible subsets of current PC as a measure of the "likelihood" (line 7). RobustPCFinder picks the variable with highest likelihood and insert it into $RawPC[T]$. Practically, kth percentile is a robust statistic rather than Max/Mean which is more tolerant to the unstable CI tests when sample size goes smaller. In this way, the learned RawPC achieves high precision.

Best-First Skeleton Growing. RobustPCFinder only takes variable T and its current RawPC[T] as input, thus its next picked variable V^* is *irrelevant* to the status of PCs of other variables. In other words, how RawPC[T] grows is irrelevant to the order of variables prioritized for calling RobustPCFinder. Also note that, as the size of RawPC[T] becomes larger, adding a new PC to T is more time consuming. Therefore, we could dynamically prioritize the variables (i.e., arrange the tasks to call RobustPCFinder), to let the easier tasks be executed earlier. In this way, given a limited time budget, we could output as many edges as possible by using a best-first skeleton growing strategy. Below we show how to prioritize variables.

Lemma 4. Denote N_{upper} as the upper bound of time complexity of procedure $RobustPCFinder(T, RawPC[T])$, then $N_{upper} = N_{RawPC[T]} c_T \sum_{j: V_j \neq T, V_j \notin RawPC[T]} c_j$ where $N_{RawPC[T]} = \prod_{l: V_l \in RawPC[T]} (c_l + 1)$. ■

Thus, time complexity of RobustPCFinder mainly depends on cardinalities of T and its *PC*. N_{upper} can be computed or updated incrementally and quickly.

Definition 5 (priority). We define the priority of a variable as the reciprocal of its N_{upper}: $priority = 1/N_{upper}$

```
Algorithm 1 BestFirstGrowing
Input: priority queue $\mathbb{Q}$, variable set $\mathcal{V}$
1:  while $\mathbb{Q}$ is not empty and no interrupt do
2:      $T \leftarrow$ variable with highest priority in $\mathbb{Q}$
3:      $V \leftarrow RobustPCFinder(T, RawPC[T])$
4:      if $V$ is not null
5:          $RawPC[T] \leftarrow RawPC[T] \cup \{V\}$
6:          $Priority[T] \leftarrow Update(T, RawPC[T])$
7:      else
8:          remove $T$ from $\mathbb{Q}$
9:          $PrunedPC[T] \leftarrow Prune(T, RawPC[T])$
10:         $hasPruned[T] \leftarrow true$
11: end while
12: foreach pair $\langle X, Y \rangle$ in $\mathcal{V}$ do
13:     if $hasPruned[X]$ is true, $hasPruned[Y]$ is true
14:         $PC[X], PC[Y] \leftarrow ANDRule(X, Y)$
15: if interrupt occurred
16:     foreach $X$ in $\mathcal{V}$ do
17:     if $hasPruned[X]$ is false
18:         $PC[X] \leftarrow Prune(X, RawPC[X])$
19: return $PC$
```

```
Procedure Prune
Input: $T, RawPC[T]$
1: $PrunedPC[T] \leftarrow RawPC[T]$
2: foreach $X$ in $PrunedPC[T]$
3:     $Other \leftarrow PrunedPC[T] \backslash \{X\}$
4:     if $RobustCorr(X, T, Other)$ is null
5:         $PrunedPC[T] \leftarrow Other$
6: return $PrunedPC[T]$
```

```
Procedure ANDRule
Input: $X, Y$
1: $PC[X] \leftarrow PrunedPC[X], PC[Y] \leftarrow PrunedPC[Y]$
2: if $X \notin PrunedPC[Y]$
3:     $PC[X] \leftarrow PrunedPC[X] \backslash \{Y\}$
4: if $Y \notin PrunedPC[X]$
5:     $PC[Y] \leftarrow PrunedPC[Y] \backslash \{X\}$
6: return $PC[X], PC[Y]$
```

Figure 7. Algorithm for Best-First Skeleton Learning

Best-first strategy. As shown in algorithm 1 (left part of Figure 7), a priority queue $\mathbb{Q}$ is maintained, with each item is a $\langle variable, priority \rangle$ pair and priority is calculated by definition 5. Iteratively, REAL picks the variable with highest priority, executes RobustPCFinder on it, and then update priority and put it back into the queue (line 2~6).

Post Processing. When timeout, skeleton growing is stopped, and a relatively lightweight post-processing is conducted to further remove false positives. The post processing consists of two parts, *Prune* for each variable (line 12~15) and AND-Rule checking for those variable pairs whose RawPCs are both completely acquired (line 17~19).

The procedure *Prune* is shown on the right top of Figure 7. It eliminates false positives in RawPC[T]: if any variable V in RawPC[T] is independent of T by conditioning on a subset of RawPC[T]$\backslash\{V\}$, then V is not a PC of T. This is valid even when the RawPC[T] has not been fully acquired due to timeout. In summary, the variables eliminated by *Prune* are guaranteed to be false positives, thus the precision is further improved. Note that pruning will also be conducted during skeleton growing once all the raw PCs for a specific variable are acquired (line 8~10), to reduce the workload on post-processing. AND-Rule is a standard rule applied by many independence-based approach (Margaritis & Thrun, 2000; Tsamardinos, Brown and Aliferis 2006), which is a symmetry constraint, saying that if X has an edge to T in skeleton G, then X must be in the PC of T and T must be in the PC of X. AND-Rule is only applicable when the RawPC[X] and RawPC[T] are completely acquired and after pruning is conducted (line 17~19). Otherwise, some true positives would be wrongly eliminated.

The post-processing is lightweight thus REAL responses to user interruption quickly. Although procedure *Prune* conducts CI tests among subsets of RawPC[T], many of the CI tests can be saved by using cache cumulated by calling RobustPCFinder. The size of RawPC[T] is closer to the size of true PC set due to good performance of RobustPCFinder.

Lemma 5 (soundness and completeness of skeleton learning). Under faithfulness, correct CI tests, and infinite data assumptions, given sufficient time or without user interruption, BestFirstGrowing identifies correct skeleton. ■

REAL. Algorithm 2 in Figure 8 shows the overall procedure of REAL. REAL first resolves incompatibility between faithfulness and FDs (line 1~3), and then BestFirstGrowing is conducted on a subset of variables $\mathcal{V}'$ (line 4~6). The final output skeleton is the union of the learned skeleton G' and the skeleton obtained by simplified FD induced graph G'_{FD} (line 7).

```
Algorithm 2 REAL
Input: data $D$, variable set $\mathcal{V}$, cardinalities $C$
Output: skeleton $G$:
1: $G_{FD}$ ← obtain FD induced graph from $D, \mathcal{V}$
2: $G'_{FD}$ ← obtain simplified FD graph from $G_{FD}$
3: $\mathcal{V}'$ ← get root nodes in $G'_{FD}$
4: $\mathbb{Q}$ ← construct priority queue from $\mathcal{V}', C$
5: $G^0$ ← empty graph
   /* skeleton growing allows timeout or user interrupt */
6: $G'$ ← BestFirstGrowing($G^0$, $\mathbb{Q}$)
7: $G \leftarrow G' \cup Skeleton(G'_{FD})$
8: return $G$
```

Figure 8. Algorithm of REAL

Theorem 5 (consistency). Given faithfulness assumption is only valid for variables with no FDs, under the correct CI tests and infinite data assumptions, the skeleton learned by REAL converges to the correct skeleton. ■

Experiments

We conduct evaluations to compare REAL with state-of-the-art SL algorithms on a) 19 large-scale benchmark datasets with ground-truth skeletons, and b) 3 real-world datasets, with feedback from domain experts.

Evaluation on Benchmark Datasets

Dataset. We use available benchmark datasets from Bayesian Network Repository (Scutari 2012) for evaluation. In these datasets, we focus on large (~100 variables) and very large (~1000 variables) networks, since they require and would benefit from anytime analysis due to the large size. So we choose 19 large-scale networks. We sample 20,000 records from each network. Among these datasets, 'pathfinder' and 'diabetes' are having unique FD edges (see theorem 4 for definition), so we use 'pathfinder' and 'diabetes' to evaluate effectiveness and efficiency of our FD Solver.

Comparison algorithms. Due to the absence of standalone anytime SL algorithm, we choose three most relevant and competitive algorithms: MINOBS (Lee and van Beek 2017), BLIP (Scanagatta, et al. 2015) and PC (Spirtes, et al., 2001) as our baselines. MINOBS and BLIP are the state-of-the-art score-based algorithm for BNSL, while PC is an independence-based algorithm used to learn skeleton for causal discovery. We only call the skeleton learning part in PC for comparison, which is denoted as PC*. MINOBS and BLIP do not directly learn skeleton, thus we use the skeleton from their resultant DAG for comparison. Configurations of MINOBS and BLIP are optimized on these datasets: because all networks are with max-in-degree≤ 6 except 'win95pts' is with max-in-degree=7. We set max-in-degree threshold for BLIP and MINOBS to 6 (change it to 7 would significantly degrade their efficiency).

Implementation. All experiments are conducted on a machine with 3.2GHz Intel i7-8700 processor and 16 GB RAM. We use the C++ implementation of MINOBS provided by the author, the Java implementation of BLIP provided by the author. Both are latest versions. For PC* and REAL, we implemented them by C#. All code are executed in single thread.

Measures. We evaluate the quality of learned skeleton by precision and $F_{0.5}$ against the ground-truth skeleton. For anytime setting, precision is more important than recall due to the requirement of reliability, so we choose $F_{0.5}$ measure, a commonly used measure that weighs precision higher than recall (Sasaki 2007). For an anytime SL algorithm, we record the $F_{0.5}$ and precision of its output given different time budgets.

Results. Figure 9 depicts precision and $F_{0.5}$ for REAL/PC*/BLIP/MINOBS on top-12 largest benchmark datasets (results for all 19 datasets are available in supplementary materials in our website due to page limit). REAL significantly outperforms the other algorithms both from precision and $F_{0.5}$ on 10 datasets except hepar2 and diabetes.

dataset		Precision						F0.5					
hepar2 70/123	time budget(s) →	1	2	4	8	15	30	1	2	4	8	15	30
	REAL	0.905	0.941	0.977	0.977	0.977	**0.977**	0.855	0.904	0.902	0.902	0.902	0.902
	PC*	0.280	0.334	0.839	0.839	0.839	0.832	0.323	0.379	0.823	0.823	0.823	0.808
	BLIP	1.000	1.000	0.967	0.978	0.978	0.949	0.876	0.868	0.901	0.911	0.914	0.903
	MINOBS	1.000	1.000	0.978	0.957	0.957	0.949	0.865	0.865	0.908	0.896	0.896	**0.903**
win95pts 76/112	REAL	0.872	0.867	0.938	0.938	0.938	**0.938**	0.802	0.826	0.868	0.868	0.868	**0.868**
	PC*	0.220	0.252	0.521	0.736	0.718	0.716	0.259	0.295	0.558	0.734	0.706	0.702
	BLIP	0.821	0.775	0.802	0.746	0.744	0.727	0.793	0.737	0.800	0.753	0.755	0.745
	MINOBS	0.904	0.800	0.761	0.754	0.711	0.727	0.817	0.763	0.762	0.762	0.721	0.745
pathfinder 109/195	time budget(s) →	10	20	40	80	150	300	10	20	40	80	150	300
	REAL	0.879	0.976	0.976	0.971	0.971	**0.971**	0.817	0.878	0.878	0.827	0.827	**0.827**
	PC*	0.414	0.417	0.411	0.411	0.407	0.391	0.448	0.450	0.444	0.444	0.439	0.421
	BLIP	0.515	0.469	0.463	0.442	0.442	0.429	0.469	0.439	0.436	0.421	0.421	0.413
	MINOBS	0.485	0.476	0.463	0.442	0.442	0.429	0.445	0.445	0.436	0.421	0.421	0.413
munin1 186/273	REAL	0.605	0.601	0.894	0.953	0.953	**0.953**	0.560	0.572	0.753	0.779	0.779	**0.779**
	PC*	0.029	0.031	0.036	0.054	0.153	0.141	0.036	0.038	0.045	0.067	0.179	0.164
	BLIP	0.757	0.757	0.710	0.662	0.662	0.647	0.653	0.653	0.656	0.620	0.624	0.613
	MINOBS	0.737	0.784	0.699	0.686	0.667	0.664	0.636	0.678	0.646	0.643	0.631	0.629
andes 223/338	REAL	0.930	0.979	0.979	0.979	0.979	**0.979**	0.916	0.942	0.942	0.942	0.942	**0.942**
	PC*	0.257	0.372	0.745	0.745	0.745	0.744	0.300	0.423	0.766	0.766	0.766	0.765
	BLIP	0.963	0.972	0.865	0.868	0.847	0.880	0.864	0.875	0.869	0.876	0.860	0.890
	MINOBS	0.972	0.972	0.870	0.855	0.867	0.918	0.875	0.875	0.873	0.871	0.881	0.926
diabetes 413/602	time budget(s) →	100	200	400	800	1500	3000	100	200	400	800	1500	3000
	REAL	0.620	0.581	0.815	0.846	0.846	**0.846**	0.611	0.595	0.700	0.718	0.718	0.718
	PC*	0.011	0.011	0.014	0.020	0.045	0.066	0.013	0.014	0.017	0.025	0.055	0.081
	BLIP	0.835	0.837	0.853	0.704	0.764	0.789	0.764	0.767	0.796	0.695	0.755	**0.786**
	MINOBS	0.833	0.842	0.877	0.716	0.714	0.780	0.762	0.771	0.816	0.707	0.713	0.781
pigs 441/592	REAL	1.000	1.000	1.000	1.000	1.000	**1.000**	1.000	1.000	1.000	1.000	1.000	**1.000**
	PC*	0.046	0.058	0.100	0.524	0.524	0.526	0.057	0.071	0.122	0.579	0.579	0.581
	BLIP	1.000	1.000	0.944	0.953	0.953	0.952	0.935	0.935	0.955	0.962	0.962	0.961
	MINOBS	1.000	1.000	0.949	0.956	0.953	0.950	0.935	0.935	0.959	0.965	0.962	0.960
link 724/1125	REAL	0.558	0.774	0.998	0.998	0.998	**0.998**	0.503	0.699	0.841	0.841	0.841	**0.841**
	PC*	0.041	0.050	0.072	0.201	0.273	0.274	0.050	0.061	0.087	0.229	0.301	0.302
	BLIP	0.556	0.562	0.563	0.561	0.577	0.623	0.495	0.500	0.524	0.544	0.558	0.600
	MINOBS	0.557	0.563	0.574	0.566	0.588	0.630	0.496	0.502	0.533	0.549	0.569	0.606
munin2 1003/1244	REAL	0.646	0.607	0.951	0.951	0.951	**0.951**	0.169	0.542	0.675	0.675	0.675	**0.675**
	PC*	0.006	0.014	0.016	0.020	0.032	0.273	0.007	0.017	0.020	0.025	0.040	0.288
	BLIP	0.755	0.755	0.767	0.713	0.700	0.716	0.645	0.645	0.661	0.641	0.636	0.651
	MINOBS	0.758	0.753	0.743	0.681	0.679	0.684	0.648	0.644	0.641	0.615	0.618	0.622
munin4 1038/1388	REAL	0.481	0.660	0.572	0.941	0.941	**0.941**	0.081	0.535	0.523	0.677	0.677	**0.677**
	PC*	0.004	0.007	0.007	0.008	0.009	0.015	0.005	0.009	0.009	0.011	0.011	0.019
	BLIP	0.713	0.721	0.729	0.654	0.647	0.668	0.600	0.606	0.613	0.575	0.578	0.600
	MINOBS	0.715	0.699	0.709	0.661	0.650	0.661	0.601	0.588	0.596	0.582	0.581	0.594
munin3 1041/1306	REAL	0.561	0.731	0.598	0.930	0.930	**0.930**	0.104	0.589	0.534	0.669	0.669	**0.669**
	PC*	0.004	0.006	0.006	0.007	0.007	0.011	0.005	0.007	0.008	0.008	0.009	0.014
	BLIP	0.793	0.772	0.788	0.720	0.687	0.677	0.679	0.660	0.674	0.645	0.624	0.618
	MINOBS	0.765	0.768	0.772	0.689	0.670	0.666	0.654	0.657	0.660	0.620	0.609	0.607
munin 1041/1397	REAL	0.444	0.624	0.595	0.950	0.950	**0.950**	0.085	0.507	0.541	0.676	0.676	**0.676**
	PC*	0.005	0.006	0.007	0.008	0.009	0.019	0.006	0.008	0.008	0.010	0.012	0.023
	BLIP	0.690	0.703	0.695	0.654	0.641	0.658	0.579	0.591	0.584	0.575	0.570	0.588
	MINOBS	0.695	0.706	0.691	0.644	0.653	0.654	0.584	0.593	0.580	0.567	0.582	0.585

Figure 9. Precision/F0.5 on 12 large-scale benchmark datasets (in first column, 70/123 means the network has 70 nodes and 123 edges)

For these two exceptional cases, REAL's precision is still significantly higher than the others while achieving similar $F_{0.5}$. Note that precision is more important than recall in anytime scenario.

dataset	#unique FD edges	config	performance of REAL				#edges missed by BLIP/MINOBS
			time(s)	Precision	Recall	F0.5	
pathfinder	7	FD	**176**	**0.971**	**0.518**	**0.827**	5
		NoFD	208	0.970	0.503	0.818	
diabetes	23	FD	**703**	**0.846**	**0.447**	**0.718**	8
		NoFD	760	0.834	0.409	0.690	

Figure 10. Effectiveness and Efficiency of FD theory

Figure 10 shows the evaluation result of FD Solver. Specifically, 'pathfinder' contains 7 unique FD edges and 'diabetes' contains 23 unique FD edges. We first find that 100% of these edges exist in ground-truth skeletons, which justifies the usefulness of theorem of uniqueness in practice. In addition, by comparing with running REAL without using FD solver (i.e., config=NoFD), it is shown that time is significantly saved while both precision and $F_{0.5}$ are increased. All these results strongly support the effectiveness and efficiency of FD theory. Note that in result of MINOBS, 5 out of 7 are missed on 'pathfinder' and 8 out of 23 are missed on 'diabetes' (last column of Figure 10). BLIP's result is the same as MINOBS's.

Evaluation on Real-World Datasets

We also evaluate REAL on three real-world, complex datasets (Table 1). All these datasets are obtained from domain experts on their regular analytical tasks. They are with large number of records and high cardinality. Table 1 also lists out the number of unique FD edges for each dataset. Since these datasets don't have ground-truth skeleton, we conduct evaluation by: a) the effectiveness of finding unique FD edges; b) in-depth study of the learned skeleton based on experts' feedback.

Table 1. Information of real datasets and results

Dataset	#var	#records	$\max_i c_i$	#unique FD edges	Missed by BLIP/MINOBS
School	9	27,611	1486	3	1
BirdStrikes	50	52,962	3377	2	2
FlightDelays	20	1,048,575	1060	9	3

Finding unique FD edges. The right-most column in Table 1 shows unique FD edges missed by BLIP and MINOBS. We omit results from PC* since Out-Of-Memory incurred when running on BirdStrikes and FlightDelays. BLIP and MINOBS produce identical results on identifying unique FD edges, i.e., at least one third of the unique FD edges are missed in their results, which indicates the learned DAG either violates global Markov property or is not minimal. E.g., in School dataset, there exists two FDs in three variables school_code, school_name and borough: school_code → school_name, and school_code → borough. This makes sense since school code is a unique ID for a specific school, which locates at a specific borough and with a fixed school name. But there exists different schools with same school name, thus these two FDs satisfy uniqueness property which implies a conditional independence: given school_code, school_name and borough are independent. However, BLIP and MINOBS don't find the edge from school_code to school name, thus their result is not Markovian.

Skeleton of BirdStrikes dataset. BirdStrikes dataset contains the record for aero planes be struck by birds. Besides the regular information of each record such as date, time, location etc., there are two important indicator variables "Effect" and "Damage Level". "Effect" records what actions taken when the bird strike occurred, such as "Aborted Take-off" / "Engine Shutdown" / "Precautionary Landing" / "…", and "Damage Level" records the level of the damage such as "High" / "Medium" / "Low" / "None". Besides, there also 14 Boolean variables indicate which specific part is struck by birds such as "Strike Engine" / "Strike Propeller" / etc., and 14 Boolean variables indicate which specific part is damaged by the strike such as "Damage Engine" / "Damage Propeller" / etc. According to the feedback from domain experts, "Effect" should be the direct effect of "StrikeX", because whenever a strike occurred, pilot or tower controllers must immediately take action regardless of whether damage is incurred or not, and the action is recorded in the "Effect" column. Therefore, in the learned skeleton, we would expect to see edges between "Effect" and "StrikeX". REAL identifies 7 out of 14 "StrikeX" variables as neighbors of "Effect", but BLIP or MINOBS only identifies 1. In comparison, REAL's result is more satisfactory according to the experts' feedback.

Related Work

The motivation for SL largely stems from the resurgence of interest in causality in recent years (Pearl 2009, Shanmugam et. al. 2015; Peters, Janzing and Schölkopf 2017). As a standard approach to building causal graphs for reasoning, a skeleton is first constructed, and human interventions are then applied to determine the edge directions. Whereas many recent studies focus on how to minimize the cost for human intervention (Shanmugam et. al. 2015; Kocaoglu, Dimakis and Vishwanath 2017; Lindgren et. al. 2018), the problem of how to reliably and efficiently build the skeleton for intervention has been largely overlooked.

Independence-based BNSL methods generally use statistical tests to determine whether two variables are independent or not. These include PC (Spirtes et. al. 2001), MMPC (Tsamardinos, Brown and Aliferis 2006), etc. Specifically, PC starts from a fully connected skeleton and iteratively removes edges, which violates reliability. The skeleton growing in MMPC is not conducted in a best-first way, and its procedure for inserting edges is not robust thus false positives are largely included.

Score-based methods is another family of BNSL that can be used for SL. They fit a Bayesian network to the data by optimizing certain metrics, such as the posterior probability of a network given data (Cooper and Herskovits 1992). These include GOBNILP (Cussens 2011), MINOBS (Lee and van Beek 2017), A* (Yuan, Malone and Wu 2011), BLIP (Scanagatta et. al. 2015), etc. Since the metrics used by score-based methods are designed for BNSL and not tailored for SL, they may produce sub-optimal skeletons.

Anytime algorithms for BNSL have also been studied recently in the literature (Malone and Yuan 2013, Lee and van Beek 2017), several anytime score-based methods for BNSL have been investigated by previous works (Campos and Ji 2011; Cussens 2011; Fan 2015; Lee and van Beek 2017; Jaakkola, Sontag, and Globerson 2010). However, these methods do not explicitly address reliability concerns for SL and may add spurious edges or delete true edges over the iterations. BLIP (Scanagatta et. al. 2015) also supports an anytime mode but it uses all time budgets allocated and cannot be interrupted to return an intermediate skeleton during the execution.

To deal with FD relationships for SL, Scheines et al. 1996 suggests to ignore the variables that can be functionally determined by others. As we pointed out, FDs exhibit strong dependency, which is useful for further tasks; Mabrouk et al. 2014 proposes an efficient algorithm for BNSL when data contains FDs, however, the algorithm assumes all Z are parents of X in learned BN when Z can determine X, thus the learned BN is not harmonious since minimality could be violated.

Conclusions

In this paper, we present REAL, a novel Reliable and Efficient Anytime skeleton Learning algorithm. REAL first resolves incompatibility between FD and faithfulness assumption. Based on this, REAL conducts SL on a reduced set of variables with guaranteed correctness thus drastically improves efficiency. Furthermore, it employs a novel edge-insertion and best-first strategy in anytime fashion for skeleton growing to achieve high reliability and efficiency. Experimental results on benchmark and real-world datasets demonstrate that REAL significantly outperforms the other state-of-the-art algorithms.

References

Chen, H.; Chiang, R.H.; and Storey, V.C. 2012. Business intelligence and analytics: From big data to big impact. MIS quarterly, pp.1165-1188.

Chickering, D. M. 1996. Learning Bayesian networks is NP-complete. In Learning from data (pp. 121-130). Springer, New York, NY.

Lee, C., and van Beek, P. 2017, September. An experimental analysis of anytime algorithms for Bayesian network structure learning. In Advanced Methodologies for Bayesian Networks (pp. 69-80).

Cooper, G. F., and Herskovits, E. 1992. A Bayesian method for the induction of probabilistic networks from data. Machine learning, 9(4), pp.309-347.

Cussens, J. 2012. Bayesian network learning with cutting planes. arXiv preprint arXiv:1202.3713.

Daly, R.; Shen, Q.; and Aitken, S. 2011. Learning Bayesian networks: approaches and issues. The knowledge engineering review, 26(2), pp.99-157.

Campos, C. P. D., and Ji, Q. 2011. Efficient structure learning of Bayesian networks using constraints. Journal of Machine Learning Research, 12(Mar), pp.663-689.

Fan, X., and Yuan, C. 2015, March. An improved lower bound for bayesian network structure learning. In Twenty-ninth AAAI conference on artificial intelligence.

Jaakkola, T.; Sontag, D.; Globerson, A.; and Meila, M. 2010, March. Learning Bayesian network structure using LP relaxations. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (pp. 358-365).

Lou, J. G.; Lin, Q.; Ding, R.; Fu, Q.; Zhang, D.; and Xie, T. 2017. Experience report on applying software analytics in incident management of online service. Automated Software Engineering, 24(4), pp.905-941.

Kocaoglu, M.; Dimakis, A.; and Vishwanath, S. 2017, August. Cost-optimal learning of causal graphs. In Proceedings of the 34th International Conference on Machine Learning-Volume 70 (pp. 1875-1884). JMLR. org.

Lee, C., and van Beek, P. 2017, May. Metaheuristics for score-and-search Bayesian network structure learning. In Canadian Conference on Artificial Intelligence (pp. 129-141). Springer, Cham.

Lindgren, E.; Kocaoglu, M.; Dimakis, A. G.; and Vishwanath, S. 2018. Experimental design for cost-aware learning of causal graphs. In Advances in Neural Information Processing Systems (pp. 5279-5289).

Mabrouk, A.; Gonzales, C.; Jabet-Chevalier, K.; and Chojnacki, E. 2014, August. An Efficient Bayesian Network Structure Learning Algorithm in the Presence of Deterministic Relations. In ECAI (Vol. 14, pp. 567-572).

Malone, B., and Yuan, C. 2013. Evaluating anytime algorithms for learning optimal Bayesian networks. arXiv preprint arXiv:1309.6844.

Margaritis, D., and Thrun, S. 2000. Bayesian network induction via local neighborhoods. In Advances in neural information processing systems (pp. 505-511).

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

Peters, J.; Janzing, D.; and Schölkopf, B. 2017. Elements of causal inference: foundations and learning algorithms. MIT press.

Sasaki, Y. 2007. The truth of the F-measure. Teach Tutor mater, 1(5), 1-5.

Scanagatta, M.; de Campos, C. P.; Corani, G.; and Zaffalon, M. 2015. Learning Bayesian networks with thousands of variables. In Advances in neural information processing systems (pp. 1864-1872).

Scheines, R.: Spirtes, P.; Glymour, C.; Meek, C.; and Richardson, T. 1996. TETRAD 3: Tools for Causal Modeling–User's Manual. CMU Philosophy.

Scutari, M. 2012. Bayesian Network Repository. http://www.bnlearn.com/bnrepository

Shanmugam, K.; Kocaoglu, M.; Dimakis, A. G.; and Vishwanath, S. 2015. Learning causal graphs with small interventions. In Advances in Neural Information Processing Systems (pp. 3195-3203).

Spirtes, P.; Glymour, C. N.; Scheines, R.; and Heckerman, D. 2000. Causation, prediction, and search. MIT press.

Tsamardinos, I.; Brown, L. E.; and Aliferis, C. F. 2006. The max-min hill-climbing Bayesian network structure learning algorithm. Machine learning, 65(1), pp.31-78.

Vardi, M. Y. 1985. Fundamentals of dependency theory. IBM Thomas J. Watson Research Division.

Yuan, C.; Malone, B.; and Wu, X. 2011, June. Learning optimal Bayesian networks using A* search. In Twenty-Second International Joint Conference on Artificial Intelligence.