Journal of Artificial Intelligence Research 80 (2024) 209-242 Submitted 11/2022; published 05/2024

Block Domain Knowledge-Driven Learning of Chain Graphs Structure

Shujing Yang YSJZUIBANG@126.COM
Fuyuan Cao (*Corresponding author*) CFY@SXU.EDU.CN
School of Computer and Information Technology, Shanxi University, Taiyuan, 030006, Chian

Abstract

As the interdependence between arbitrary objects in the real world grows, it becomes gradually important to use chain graphs containing directed and undirected edges to learn the structure among objects. However, independence among some variables corresponds to multiple structures and the direction of edges among variables cannot be uniquely determined. This limitation restricts existing chain graphs structure learning algorithms to only learning their Markov equivalence class. To alleviate this limitation, we define the block domain knowledge and propose a block domain **k**nowledge-**d**riven **l**earning **c**hain **g**raphs structure algorithm (KDLCG). The KDLCG algorithm learns the adjacencies and spouses of all variables, which are utilized to directly construct the skeleton and orient the edges of the complexes, thereby learning the Markov equivalence class of the chain graphs. Subsequently, the KDLCG algorithm then updates some edges with Meek rules, guided by block domain knowledge. Finally, the KDLCG algorithm directs some edges by estimating causal effects between two variables, driven by block domain knowledge. Meanwhile, we conduct theoretical analysis to prove the correctness of our algorithm and compare it with the LCD algorithm and MBLWF algorithm on synthetic and real-world datasets. The experimental results validate the effectiveness of our algorithm.

1. Introduction

A standard assumption in causal inference is the absence of unit interference, which asserts that giving treatment to a particular unit only affects the response of that unit. While the assumption in many statistical applications is sensible, there are settings where this assumption is not reasonable. For example, social media data exhibits homogeneity (friends are similar due to the fact that they are friends) and contagiousness (friends may causally influence each other) (Sherman & Shpitser, 2018). Similarly, vaccinating some subset of a population may confer immunity to the entire population, which is known as herd immunity in infectious disease epidemiology. This implies that the research subjects do not exist in isolation but in the interacting network, and the interactions among research subjects can lead to dependencies in the data.

In the context of causal inference, the dependence data are mostly modeled using Bayesian Networks (BNs), which are the most well-known subclasses of probabilistic graphical models. BNs represent the relationships among variables using directed acyclic graphs (DAGs), where the directed edges indicate causal relationships. However, Bayesian Networks have certain undesirable limitations when representing independent information for an actual problem domain (Schäfer & Strimmer, 2005). In order to better model the dependency data, chain graphs (CGs) are the better choice, which is one of the probabilistic graphical models.

Chain graphs (CGs) are a class of hybrid graphs that admit both undirected and directed edges but not partially directed cycles. The three following interpretations are the best known in the literature. The first interpretation (LWF), introduced by Lauritzen, Wermuth, and Frydenberg (1989), combines directed acyclic graphs and undirected graphs. The second interpretation (AMP), introduced by Andersson, Madigan, and Perlman (1996), combines directed acyclic graphs and undirected graphs but with a Markov equivalence criterion that more closely resembles the one of directed acyclic graphs. The third multivariate regression interpretation (MVR), introduced by Cox and Wermuth (2014), combines directed acyclic graphs and bidirected (covariance) graphs. Sonntag and Peña (2015a) describe the relationship between the three interpretations in detail. This paper deals with CGs under the LWF interpretation (LWF CGs), which is the generalization of graphical models based on Markov networks (MNs, undirected graphs) and Bayesian networks (BNs, directed acyclic graphs) and have been widely studied (Lauritzen & Wermuth, 1989). The directed edges of LWF CGs represent causal relationships. The undirected edges of LWF CGs represent symmetric relationships due to interference (Shpitser, Tchetgen, & Andrews, 2017; Ogburn, Shpitser, & Lee, 2020; Bhattacharya, Malinsky, & Shpitser, 2020). The subsequent references to CGs in the paper refer to LWF CGs. BNs and MNs are subclasses of LWF CGs. Sonntag et al. (2015) have studied that only a small portion of LWF CGs models can be represented by BNs and MNs. LWF CGs have received increasing attention as modeling tools for statistical applications, and CGs models have been applied to some fields, such as reasoning about correlations among diseases (Lappenschaar, Hommersom, & Lucas, 2014), recommender systems (Chen, Chang, Li, & Zheng, 2018), causal spillover effect estimation (Vazquez-Bare, 2023; Yu, Airoldi, Borgs, & Chayes, 2022; Tchetgen Tchetgen, Fulcher, & Shpitser, 2021; Bhattacharya et al., 2020), social networks (Ogburn et al., 2020), neural networks (Shen & Cremers, 2020) and time series(Xu, Fard, & Fang, 2020).

One important and challenging task is the structure learning of models directly from sampled data. LWF CGs structure learning methods fall into two main categories: the constraint-based method and the score-based method. Due to the difficulty of finding valid scoring functions, CGs structure learning uses the constraint-based method, which employs conditional independence tests to infer the relationship among variables. Similarly, most of the structure learning methods of AMP CGs and MVR CGs also use constraint-based methods (Javidian, Valtorta, & Jamshidi, 2020a; Wang & Bhattacharyya, 2022; Javidian, 2019).

Currently, many constraint-based algorithms have been proposed for LWF CGs structure learning. The largest CG recovery algorithm LCG, presented by Studenỳ (1997), was the earliest work that learned the structure learning of CGs. Based on the LCG algorithm, the order-dependent algorithm was proposed by Javidian et al. (2020c). They worked by first recovering the skeleton of CGs and then orienting the edges. Learning CGs structure algorithm via decomposition, named LCD, adopted a divide-and-conquer method for recovering the structure of CGs (Ma, Xie, & Geng, 2008). The LCD algorithm first constructed the separation trees, which may be utilized to decompose the knowledge represented by the CGs into local subsets, and then learned the local skeleton of each subset. The LCD algorithm determined all complex arrows after acquiring the skeleton. The inclusion optimal algorithm (CKES) (Peña, Sonntag, & Nielsen, 2014) introduced for learning CGs structure is to find an inclusion optimal CG that satisfies the probability distribution of composition attributes. The answer set programming method (ASP), presented by Sonntag et al. (2015) for learning an optimal CG, was based on encoding the learning problem using the answer set programming paradigm without making assumptions on the existing probability distribution. The method was proposed by Wang et al. (2019) for the local structure learning of CGs with false discovery rate control. The MbLWF algorithm was presented by Javidian et al. (2020b). The authors proved that GSMB (Margaritis & Thrun, 1999), IAMB (Tsamardinos, Aliferis, Statnikov, & Statnikov, 2003), and its variants (Yaramakala & Margaritis, 2005) were still sound for MB discovery in LWF CGs under faithfulness and causal sufficiency assumptions.

However, the above algorithms are limited to learning only the Markov equivalence classes of the chain graphs. This limitation arises since the independence among certain variables corresponds to multiple structures in chain graphs structure learning, making it impossible to determine the direction of edges among these variables uniquely. Fortunately, domain knowledge or background knowledge can further refine this structure, thereby increasing the number of identifiable causal relationships. Some of the current literature on causal DAGs learning has been enriched by introducing different types of background knowledge, such as specifying one variable as the cause of another, giving the order of variables, or giving a tiered ordering of variables (Meek, 1995; Scheines, Spirtes, Glymour, Meek, & Richardson, 1998; Eigenmann, Nandy, & Maathuis, 2017; Rothenhäusler, Ernest, & Bühlmann, 2018; Perkovic, 2020; Andrews, Spirtes, & Cooper, 2020), to enrich its causal relationships identification in DAGs learning.

To identify more causal relationships in CGs, we define block domain knowledge and propose a novel block domain **k**nowledge-**d**riven algorithm for **l**earning **c**hain **g**raphs structure from causally sufficient data, called KDLCG. The KDLCG algorithm first learns the adjacencies and spouses of all variables, directly constructs the CG skeleton based on the adjacencies, and orients the edges to obtain the Markov equivalence class of CG by using the adjacencies and spouses. Subsequently, the KDLCG algorithm directs some edges of the Markov equivalence class by using Meek rules (Meek, 1995) to obtain the new CG, driven by block domain knowledge. The KDLCG algorithm, finally driven by block domain knowledge, orients some undirected edges in the updated CG by estimating the causal effects between variables to obtain the final global CG structure. In parallel, we theoretically prove the correctness of our proposed LWF CGs structure learning algorithm, evaluate the performance of our proposed algorithm on synthetic and real-world datasets, and show the competitive performance of our algorithm against the state-of-the-art LCD algorithm and MbLWF algorithm.

The rest of the paper is organized as follows. In Section 2, we give the preliminaries. Section 3 proposes a novel KDLCG algorithm, uses an example to track the KDLCG algorithm, and theoretically analyzes the correctness of the KDLCG algorithm. Section 4 reports experimental results to illustrate the performance of the KDLCG algorithm and Section 5 concludes the paper and presents future work.

2. Preliminaries

Below, we briefly list some of the central concepts covered in the paper, and Table 1 summarizes the notations used in this paper.

Table 1: Summary of Notation.

Symbol	**Meaning**
V	a variable set
E	an edge set
G	a chain graph on V
P	a joint probability distribution on V
X, Y, T	a node or a variable
W, R, S, Z	a set on V
B	all blocks in G
B_i	a block in G
B_{iX}	variables contained in a block in G in G
Pa_X	parents of X
Ch_X	children of X
Ne_X	neighbors of X
Bd_X	boundary of X
De_X	descendants of X
An_X	ancestors of X
Sd_X	strict descendants of X
Adj_X	adjacencies of X
$CAdj_X$	a candidate set of Adj_X
SP_X	spouse of X
CSP_X	a candidate set of SP_X
$CSP_{X,Y}$	a candidate set of SP_X regard to Y
MB_X	markov blanket of$X$
Adj	adjacencies of all variables on V
SP	spouses of all variables on V
$SepSet_{X,Y}$	separation set of X and Y
$SepSet$	separation set of between two variables on V
$X \perp\!\!\!\perp Y \mid S$	X is conditionally independent of Y given S
$X \not\!\perp\!\!\!\perp Y \mid S$	X is conditionally dependent of Y given S

2.1 Graphical Terminology

In this paper, we mainly use the terminology proposed by Lauritzen in (Lauritzen, 1996), and the reader can also find further details in (Lauritzen, 1996).

Definition 1 (Chain graph) A CG is a pair (G, P). $G = (V, E)$ consists of a nonempty finite set of variables V and an edge set E, where V represents the random variables. P is a joint probability distribution on V.

In a CG G, for variable $X, Y \in V$, if $(X, Y) \in E$ and $(Y, X) \in E$, we say that there is an undirected edge between X and Y denoted by $X - Y$. If $(X, Y) \in E$ and $(Y, X) \notin E$, we say that there exists a directed edge from X to Y denoted by $X \to Y$. We define the set of parents, children, neighbors, boundary and adjacencies of a variable X in G as follows, respectively: $Pa_X = \{Y|Y \to X \in E\}$, $Ch_X = \{Y|X \to Y \in E\}$, $Ne_X = \{Y|X - Y \in E\}$, $Bd_X = Pa_X \cup Ne_X$, $Adj_X = Pa_X \cup Ch_X \cup Ne_X$.

Definition 2 (Block) The blocks B of a CG are the connected components of the undirected graphs obtained by removing all directed edges from the CG, where an undirected graph is a block. A CG consists of k blocks, then $B = \{B_1, B_2, ..., B_k\}$. Each block is also called a chain component. In a DAG, all blocks are singletons.

A *path* in a CG G is a sequence of distinct variables $X_1,\cdots,X_n (n \geq 1)$, such that X_i and X_{i+1} are adjacent in G for each $i = \{1,\cdots,n-1\}$. A *path* $X_1,\cdots,X_n (n \geq 1)$ is descending if $X_i \to X_{i+1}$ or $X_i - X_{i+1}$ in G for all $1 \leq i \leq n-1$. If there exists a *descending path* from X_1 to X_n in G, we say that X_n is a descendant of X_1, or X_1 is an ancestor of X_n. The descendants of a variable X in G is denoted by De_X. The ancestors of a variable X in G is denoted by An_X. We say that X_n is a strict descendant of X_1, if X_n is a descendant of X_1, but X_1 is not a descendant of X_n. The strict descendants a variable X in G is denoted by Sd_X.

Definition 3 (Complex) A *complex* in a CG G is a path $X_1,\cdots, X_k (k \geq 3)$, such that $X_1 \to X_2$, $X_i - X_{i+1} (2 \leq i \leq k-2)$, $X_{k-1} \leftarrow X_k$ in G, and no additional edges exist among the variables of $X_1, \cdots,X_k$ in G. The variables X_1 and X_k are said to be the parents of the complex and we say that X_1 (X_k) is a spouse of X_k (X_1) in G. In a DAG, the complex is a *V-structure.*

Defination 3 states that the spouses of a variable X is denoted by SP_X. Then, the Markov blanket (MB) of a variable X is the set of parents, children, neighbors and spouses (Javidian et al., 2020b), $MB_X = Pa_X \cup Ch_X \cup Ne_X \cup SP_X$.

The *skeleton* of a CG G is obtained from G by changing all directed edges of G into undirected edges. The *moral graph* of G denoted by G^m is obtained by first connecting the parents of every complex in G with an undirected edge, and then taking the skeleton of the resulting graph. We say that two CGs have the same *pattern iff* they share the same skeleton and complexes. Two CGs are *Markov equivalent iff* they have the same *pattern.*

Studenỳ and Bouckaert introduced the notion of $c - separation$ for chain graphs G (Studenỳ & Bouckaert, 1998). We say that a path ρ on G is intervented by a subset S of V if and only if there exists a section σ of ρ such that:

1. either σ is a head-to-head section with respect to ρ, and σ is outside S; or
2. σ is a non head-to-head section with respect to ρ, and σ is hit by S.

P satisfies the *Global Markov property*, for any disjoint subsets (W, R, S) of V such that S separates W from R in $(G_{An_{W \cup R \cup S}})^m$, the moral graph of the smallest ancestral set containing, indicated as $W \perp\!\!\!\perp R|S$ (read: S *c-separates* W from R, or W is independent of R geiven S, where S is said to be *c-separation set*). P satisfies the *Local Markov property iff* $X \perp\!\!\!\perp V\backslash\{X\}\backslash Sd_X\backslash Bd_X|Bd_X$ for all $X \in V$ hold.

Assumption 1 (Faithfulness Assumption) Given a CG (G, P), P and G are faithful to each other *iff* the conditional independence between variables in G are captured by P.

Assumption 1 states that in a faithful CG, if X and Y are *c-separates* by S in G, then they will be conditionally independent conditioned on S in P.

Example 1. We consider a CG named as Toy graph in Figure 1 (a) presented in Cowell et al. (Cowell, Dawid, Lauritzen, & Spiegelhalter, 2007). $B - D$ is an undirected edge, $D \to F$ is a directed edge, and $D \to F - E \leftarrow C$ and $F \to K \leftarrow G$ are two Complexs. We obtain the six blocks B (chain components) in Figure 1 (b) after removing all directed edges in Figure 1 (a), the obtained six blocks are $B_1 = \{C-A-B-D\}, B_2 = \{E-F\}, B_3 = \{G-H\}, B_4 = \{K\}, B_5 = \{I\}$ and $B_6 = \{J\}$, see Figure 1 (b).

We consider other CG in Figure 1 (d). $D \to F - E \leftarrow C$ and $F \to K \leftarrow G$ are two Complexs. We obtain the three blocks B (chain components) in Figure 1 (b) after removing all directed edges in Figure 1 (a), the obtained three blocks are $B_1 = \{C - A - B - D - G - H\}, B_2 = \{E - F - I - J\}, B_3 = \{K\}$, see Figure 1 (e).

Let us take the variable F as an example, we can see from Figure 1 (a) and Figure 1 (d) that $Pa_F = \{D\}$, $Ch_F = \{K\}$, $Ne_F = \{E\}$, $Adj_F = \{D, E, K\}$, $SP_F = \{G\}$, $MB_F = \{D, E, G, K\}$. According to the definition of Markov equivalent, it is clear that Figure 1 (a) and Figure 1 (d) belong to the same Markov equivalent because they have the same pattern, where Figure 1 (d) is their pattern, Figure 1 (e) shows their skeleton and Figure 1 (f) shows their moral graph.

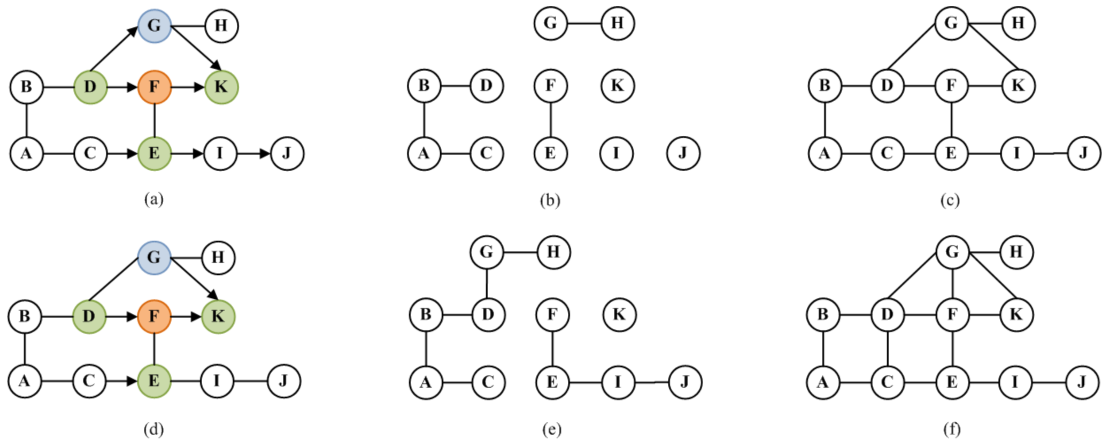


Figure 1: An example of Toy graph related concepts.

2.2 Causal Effect Terminology

A statistical model associated with a directed acyclic graph is a set of distributions that factorize as: $f(x_1, ..., x_n) = \prod_{i=1}^{n} f(x_i|Pa(x_i, G))$ (Pearl, 2003). In order to obtain the effect of an intervention on a target variable, Pearl (Pearl, 1995) employed the *do* operator to formulate the post-intervention distribution as follow:

$$f(x_1,...,x_n|do(X_j = x_j)) = \begin{cases} \prod\limits_{i=1,i\neq j}^{n} f(x_i|Pa_{x_i})|_{x_j=x_j}, & \text{if } x_j = x_j, \\ 0, & otherwise. \end{cases} \tag{1}$$

Here, $f(x_1,...,x_n|do(X_j = x_j))$ is the post-intervention distribution over $V = X_1,\ldots,X_n$ after intervening on X_j, by forcing X_j to equal x_j.

The distribution of $Y = X_n$ after an intervention $do(X_i = x_i)$ can be found by integrating out $x_1,\ldots,x_{n-1}$ in (1). It can be shown that this simplifies to the following:

$$f(y|do(X_i = x_i)) = \begin{cases} f(y), & \text{if } Y \in Pa_{x_i}, \\ \int f(y|X_i = x_i, Pa_{x_i})f(Pa_{x_i})d(Pa_{x_i}), & \text{if } Y \notin Pa_{x_i}. \end{cases} \tag{2}$$

In fact, the expression in Equation 2 for $Y \notin Pa_{x_i}$ is a special case of so-called $back-door$ $adjustment$, and Pa_{x_i} is a $back-door$ $adjustment$ set (Neuberg, 2003).

It is common to summarize the distribution generated by an intervention by its mean (Neuberg, 2003), i.e., the mean of Y w.r.t. $f(y|do(X = x))$, which is denoted by $E(Y|do(X = x))$.

$$E(Y|do(X = x_i)) = \begin{cases} E(Y), & \text{if } Y \in Pa_{x_i}, \\ \int E(Y|x_i, Pa_{x_i})f(Pa_{x_i})d(Pa_{x_i}), & \text{if } Y \notin Pa_{x_i}. \end{cases} \tag{3}$$

And we can define the **average causal effect (ACE)** of $do(X = x_i)$ on Y, i.e., $ACE(Y|do(X = x_i))$, by

$$ACE(Y|do(X = x_i)) = \frac{\partial E(Y|do(X = x_i))}{\partial x_i}. \tag{4}$$

In the remainder of paper, we consider the case that $X_1,\ldots,X_{n-1},Y$ are jointly Gaussian, it is very simple to compute the causal effects (CE) as defined in Equation 4, since Gaussianity implies that $E(Y|x_i, Pa_{x_i})$ is linear in x_i and Pa_{x_i}, $E(Y|x_i, Pa_{x_i}) = \gamma_0 + \gamma_i x_i + \gamma_{Pa_{x_i}}^T Pa_{x_i}$ for some values $\gamma_0, \gamma_i \in \mathbb{R}$ and $\gamma_{Pa_{x_i}} \in \mathbb{R}^{|Pa_{x_i}|}$, where $|Pa_{x_i}|$ is the cardinality of the set Pa_{x_i} (Maathuis, Kalisch, & Bühlmann, 2009). Hence, $\int E(Y|x_i, Pa_{x_i})f(Pa_{x_i})d(Pa_{x_i}) = \gamma_i x_i + \int \gamma_{Pa_{x_i}}^T Pa_{x_i} f(Pa_{x_i})d(Pa_{x_i})$ is linear in x_i. Combining this with Equation 4, it follows that the causal effect of x_i on Y with $Y \in Pa_{x_i}$ is given by γ_i, which is simply the regression coefficient of X_i in the regression of Y on X_i and Pa_{x_i}. In general, the causal effect of X_i on Y as defined in Equation 4 is given by $\boldsymbol{\beta_{i|Pa_{x_i}}}$, where, for any set $S \subset X_q,\ldots,X_{n-1},Y \backslash X_i$,

$$\beta_{i|S} = \begin{cases} 0, & \text{if } Y \in S, \\ \text{coefficient of } X_i \text{ in } Y \sim X_i + S, & \text{if } Y \notin S. \end{cases} \tag{5}$$

3. The Proposed Algorithm

In this section, we present a novel KDLCG algorithm in Section 3.1 and use examples to track the KDLCG algorithm to increase its readability. Additionally, we theoretically analyze the correctness of the KDLCG algorithm in detail in Section 3.2.

3.1 The KDLCG Algorithm

We detail the proposed KDLCG algorithm, dividing it into learning the Adj and SP and the LWF CG structure. Algorithm 1 and Algorithm 2 give the implementation details of KDLCG.

3.1.1 Learn the Adj and SP

In this section, we introduce our proposed KDLCG algorithm for learning the Adj and SP of variables, named learn-AS (Algorithm 1). In the following, we describe the algorithm in detail in three steps.

Step 1: Learn $CAdj_T$ and CSP_T (lines 3-24). The learn-AS algorithm adds $X \in V\backslash\{T\}$ that is conditionally dependent on $T \in V$ into $CanAdj_T$ (lines 4-7). Then, $CanAdj_T$ is sorted in increasing value of $pvalue$[1] to select the variable with the highest association with T to join $CAdj_T$ as early as possible (line 8). Starting from line 9, when Y is added to $CAdj_T$ (line 10) and is removed from $CanAdj_T$ (line 11), the learn-AS algorithm checks each variable in $CAdj_T$ by conditioning on Z for removing false positives (lines 12-14) (the first pruning operation on $CAdj_T$). Then, the learn-AS algorithm looks for a variable $X \in CAdj_T$ that unblocks a path from T to some variable $Y \in NonAdj$ (lines 18-24). If such X exists, Y could be a spouse (line 24), while X could be an adjacency or non-adjacency variable. After checking at line 21, line 22 removes the found non-adjacencies (the second pruning operation on $CAdj_T$).

Step 2: Remove false positives from CSP_T (lines 26-30). The learn-AS algorithm tests whether X in $SP_{T,Y}$ is conditionally independent on T given $Z \cup \{Y\}$ (line 29). If the subset Z exists and X and T are independent, X is removed (line 30).

Step 3: Remove false positives from $CAdj_T$ (lines 32-35). The learn-AS algorithm tests whether X in $SP_{T,Y}$ is conditionally independent on T given a subset of $CAdj_T \cup SP_T\backslash\{X\}$ (line 34). If X and T are independent, X is removed (line 35) (the third pruning operation on $CAdj_T$).

Highlights of Algorithm 1: (1) The learn-AS algorithm uses a forward method to select the variable with the highest dependency on target variable T from V to join the $CAdj_T$ while learning the $CAdj_T$. As long as a variable is added to the $CAdj_T$, the learn-AS algorithm checks the false positives in the $CAdj_T$ and deletes them to reduce the cascading errors caused by the existence of false positives later. **(2)** The learn-AS algorithm simultaneously prunes the false positives from $CAdj_T$ and learns spouses, learning true spouse variables as possible instead of learning some pseudo-spouse variables (false positive spouse variables) that depend on non-adjacent variables. **(3)** The learn-AS algorithm learns the separated Adj and SP, which is very convenient for learning LWF CGs structure.

Example 2. We use the example with F in the Toy graph (Figure 1 (a)) (Cowell et al., 2007) as the target variable to trace the execution of the learn-AS algorithm, which enhances its readability when learning Adj_F and SP_F, see Table 2.

1. The conditional independence tests calculate the $pvalue$ of between variables, which a larger value indicates more independence between the two variables and vice versa.

Algorithm 1: learn the *Adj* and *SP* (learn-AS)

```
input  : $D$:data
output: $Adj, SP$
1  for $T \in V$ do
2      /* Step 1: learn $CAdj_T$ and $CSP_T$ */
3      $CAdj_T \leftarrow \phi$ ;
4      for $X \in V \backslash X$ do
5          if $T \not\perp\!\!\!\perp X|\phi$ then
6              $CanAdj_T \leftarrow CanAdj_T \cup \{X\}$;
7      Sort $CanAdj_T$ in increasing order of $pvalue$;
8      for $Y \in CanAdj_T$ do
9          $CAdj_T \leftarrow CAdj_T \cup \{Y\}$;
10         $CanAdj_T \leftarrow CanAdj_T \backslash \{Y\}$;
11         for $X \in CAdj_T$ do
12             if $T \perp\!\!\!\perp X|Z, \exists Z \subseteq CAdj_T \backslash \{X\}$ then
13                 $CAdj_T \leftarrow CAdj_T \backslash \{X\}$;
14                 $SepSet_{T,X} \leftarrow Z$ ;
15     $remove \leftarrow \phi$;
16     $NonAdj_T \leftarrow V \backslash CAdj_T \backslash \{T\}$;
17     for $X \in CAdj_T$ do
18         for $Y \in NonAdj_T$ do
19             if $T \not\perp\!\!\!\perp Y|SepSet_{T,Y} \cup \{X\}$ then
20                 if $T \perp\!\!\!\perp X|CAdj_T \cup \{Y\} \backslash \{X\}$ then
21                     $remove \leftarrow remove \cup \{X\}$;
22                 else
23                     $CSP_{T,X} \leftarrow CSP_{T,X} \cup \{Y\}$;
24     $CAdj_T \leftarrow CAdj_T \backslash remove$;
25     /* Step 2: remove false positive from $CSP_T$ */
26     for $Y \in CAdj_T$ do
27         $SP_{T,Y} \leftarrow CSP_{T,Y}$;
28         for $X \in SP_{T,Y}$ do
29             if $T \perp\!\!\!\perp X|Z \cup Y, \exists Z \subseteq CAdj_T \cup SP_{T,Y} \backslash \{X\}$ then
30                 $SP_{T,Y} \leftarrow SP_{T,Y} \backslash \{X\}$;
31     /* Step 3: remove false positive from $CAdj_T$ */
32     $Adj_T \leftarrow CAdj_T$;
33     for $X \in CAdj_T$ do
34         if $T \perp\!\!\!\perp X|Z, \exists Z \subseteq Adj_T \cup SP_T \backslash \{X\}$ then
35             $Adj_T \leftarrow Adj_T \backslash \{X\}$;
36     return $Adj, SP$;
```

(1) Step 1: The learn-AS algorithm adds the variables in $\{B, C, D, E, G, H, I, J, K\}$ to $CanAdj_F$ given the empty set, because these variables are conditionally dependent of F given the empty set. Then, $CanAdj_F$ is sorted in increasing value of $pvalue$, $CanAdj_F = \{K, J, I, E, D, C, G, B, H\}$. The learn-AS algorithm sequentially adds

Table 2: Tracking of the learn-AS algorithm.

Step	Iteration	result of learnAS algorithm
Step 1	1	**Condition set:** ϕ
		$F \perp\!\!\!\perp A \vert \phi$
		$F \not\perp\!\!\!\perp B\vert\phi, F \not\perp\!\!\!\perp C\vert\phi, F \not\perp\!\!\!\perp D\vert\phi, F \not\perp\!\!\!\perp E\vert\phi, F \not\perp\!\!\!\perp G\vert\phi$
		$F \not\perp\!\!\!\perp H\vert\phi, F \not\perp\!\!\!\perp I\vert\phi, F \not\perp\!\!\!\perp J\vert\phi, F \not\perp\!\!\!\perp K\vert\phi$
		Conclusion: $\{B, C, D, E, G, H, I, J, K\}$ added into $CanAdj_F$
		Sort $CanAdj_F$ in increasing order of $pvalue$ as $\{K, J, I, E, D, C, G, B, H\}$
	2	**Condition set: the subsets of** $CAdj_F$
		$F \perp\!\!\!\perp J\vert I, F \perp\!\!\!\perp I\vert E, F \perp\!\!\!\perp C\vert\{D, E\}, F \perp\!\!\!\perp G\vert D, F \perp\!\!\!\perp B\vert D, F \perp\!\!\!\perp H\vert D$
		Conclusion: $\{J, I, C, G, B, H\}$ are removed from $CAdj_F$
		$F \not\perp\!\!\!\perp K\vert Z, F \not\perp\!\!\!\perp E\vert Z, F \not\perp\!\!\!\perp D\vert Z, \forall Z \subseteq CAdj_F$
		Conclusion: $\{K, E, D\}$ are remained in $CAdj_F$
	3	**Condition set: union of the separated set of variable pairs and the subsets of** $CAdj_F$
		$F \not\perp\!\!\!\perp H\vert\{D, K\}$, $F \not\perp\!\!\!\perp G\vert\{D, K\}$
		$F \not\perp\!\!\!\perp K\vert Z, \forall Z \subseteq CAdj_F$
		Conclusion: $\{H, G\}$ are added into CSP_F
	Step 1: the learnAS algorithm outputs $CAdj_F = \{K, E, D\}$ and $CSP_F = \{H, G\}$.	
Step 2	1	**Condition set: the subsets of the union of** $CAdj_F$ **and** CSP_F
		$F \perp\!\!\!\perp H\vert G$, H is removed from SP_F
		$F \not\perp\!\!\!\perp G\vert Z$, $\forall Z \subseteq CAdj_F \cup CSP_F$, G is added into SP_F
	Step 2: the learnAS algorithm outputs $SP_F = \{G\}$.	
Step 3	1	**Condition set: the subsets of the union of** $CAdj_F$ **and** CSP_F
		$F \not\perp\!\!\!\perp K\vert Z, F \not\perp\!\!\!\perp E\vert Z, F \not\perp\!\!\!\perp D\vert Z, \forall Z \subseteq CAdj_F \cup CSP_F$, $\{K, E, D\}$ is added into Adj_F
	Step 3: the learnAS algorithm outputs $Adj_F = \{K, E, D\}$.	
Conclusion: the learnAS algorithm outputs $Adj_F = \{K, E, D\}$ **and** $SP_F = \{G\}$**.**		

the variables in $\{K, J, I, E, D, C, G, B, H\}$ to $CAdj_F$, and the learn-AS algorithm prunes the set for the presence of false positives. Since $F \perp\!\!\!\perp J|I, F \perp\!\!\!\perp I|E, F \perp\!\!\!\perp C|\{D, E\}, F \perp\!\!\!\perp G|D, F \perp\!\!\!\perp B|D, F \perp\!\!\!\perp H|D$, $\{J, I, C, G, B, H\}$ are removed from $CAdj_F$. $\{K, E, D\}$ are always conditionally dependent of F given any set and are remained in $CAdj_F$. We know that $NonAdj = \{A, B, C, G, H, I, J\}$. The algorithm iterates through the variables in $NonAdj$ to find the spouse variables given $CAdj_F$. The algorithm operates to remove false positives in $CAdj_F$ while finding the spouse variables. Since $F \not\perp\!\!\!\perp H|\{D, K\}$, $F \not\perp\!\!\!\perp G|\{D, K\}$, $\{H, G\}$ is added into CSP_F. Finally, $CAdj_F = \{K, E, D\}, CSP_F = \{G, H\}$.

(2) Step 2: The learn-AS algorithm prunes $CSP_F = \{G, H\}$ to delete the false positives. Since $F \perp\!\!\!\perp H|G$, H is not added into SP_F. $SP_F = \{G\}$.

(3) Step 3: Since $\{K, E, D\}$ are conditionally dependent of F given any set, $Adj_F = \{K, E, D\}$.

The learn-AS algorithm iterates over all variables and finally outputs the set of adjacencies and spouses of all variables, the Adj and SP.

3.1.2 Learn LWF CG Structure

In this section, we introduce our proposed KDLCG algorithm for learning the LWF CG structure (Algorithm 2). In the following, we first introduce our proposed concepts of block domain knowledge and then describe the KDLCG algorithm in detail in three steps.

Algorithm 2: learn LWF CG structure (KDLCG)

```
input  : $Adj, SP$, block domain knowledge: $K$
output: The structure of CG
1  /* Step 1: learn the Markov equivalence class of CG $G$ */
2  for $T \in V$ do
3      for $X \in Adj_T$ do
4          $T - X$; /* add undirected edge */
5      for $Y \in SP_T$ do
6          if $T \not\perp\!\!\!\perp Y|Z \cup X, \exists X \in Adj_T, \forall Z \subseteq Adj_T \backslash \{X\}$ then
7              $T \to X$; /* add directed edge */
8  /* Step 2: learn the structure of CG using Meek rules driven by $K$*/
9  Extraction of the block structure $B_G^{'}$ from the Markov equivalence class of CG
     driven by $K$;
10 Using Meek rules to update the block structure $B_G^{'}$ driven by $K$;
11 Use the valid orientation rule to update the CG structure $G$;
12 /* Step 3: learn the structure of CG by estimating causal effects between variables
   driven by $K$ */
13 Record the information $und_{info}$ of the variable pairs with undirected edges in $B_G^{'}$;
14 for $K_i, K_j \in und_{info}$ do
15     $ce_{value} = CE - based(K_i, K_j)$; /* Estimate the causal effects between $K_i$ and
         $K_j$ */
16     if $ce_{value} == 0$, $X \in K_i$, $Y \in K_j$ and $X - Y$ then
17         $X \to Y$; /* add directed edge */
18     if $ce_{value}$ contains 0, $X \in K_i$, $Y \in K_j$, $i < j$ and $X - Y$ then
19         $X \to Y$; /* add directed edge */
20 return CG $G$;
```

It is known from the preliminaries in Section 2 that removing all directed edges in the chain graph gives k undirected graphs. An undirected graph is called a block in the chain graph, and each undirected graph is composed of one or more variables. Below, we define two new concepts: block variables and block domain knowledge.

Definition 5 (Block variable) Let $X_{B_1}, \cdots, X_{B_k}$ be a partition of the variable set V, where each X_{B_i} $(i = 1, \cdots, k)$ is referred to as a block variable consisting of the sequence of variables contained in block B_i. We denote the collection of all block variables as the block variable set, represented as $X_B = \{X_{B_1}, \cdots, X_{B_k}\}, k \in [1, |V|]$.

Definition 5 states that if a CG has k blocks, then the CG contains k block variables. We can consider each block variable as a random variable. Based on the properties of the blocks in the CG and the directed acyclicity of the CG, we can deduce that the structure composed of block variables forms a directed acyclic graph, denoted as the block structure B_G corresponding to the CG.

Example 3. Consider the CG in Figure 2(a) and the six blocks of the CG after removing the directed edges, as shown in Figure 2(b). Then, the CG contains six block variables, namely $X_{B_1} = \{A, B, C, D\}, X_{B_2} = \{E, F\}, X_{B_3} = \{G, H\}, X_{B_4} = \{I\}, X_{B_5} = \{J\}, X_{B_6} = \{K\}$. The block structure B_G composed of six block variables is shown in Figure 2(c).

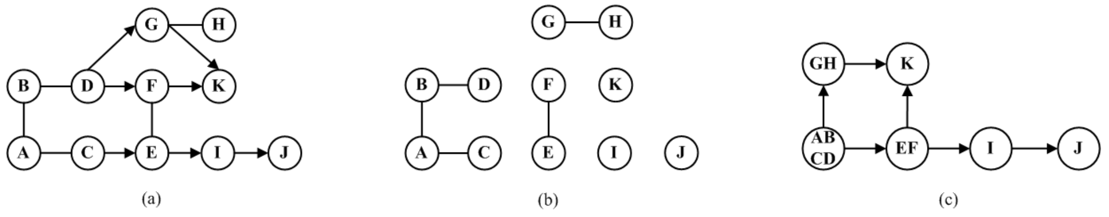


Figure 2: An example of block variable and block structure.

Definition 6 (Block domain Knowledge) Let $K = \{K_1, \cdots, K_k\}$ represent block domain knowledge, where $k \in [1, |V|]$, and $K_1, \cdots, K_k$ are the partition of variable set V ($|V|$ is the number of variables contained in V). Each K_i represents a block domain knowledge consisting of the variables in the block B_i. Therefore, block domain knowledge K can also be expressed as $K = \{X_{B_1}, \cdots, X_{B_k}\}$.

Definition 6 states that a CG has k block variables, which means that a CG contains k block domain knowledge, and $K = \{K_1, \cdots, K_k\} = \{B_{X_1}, \cdots, B_{X_k}\} = B_X, K_i = B_{X_i} = \{X_j, \cdots, X_m\}, 1 \leq i \leq k, 1 \leq j \neq m \leq |V|$. Based on the properties of chain graphs, we know that variables within a block domain knowledge K_i (i.e., within a block) are connected by undirected edges, while variables between different block domain knowledge (i.e., between different blocks) are connected by directed edges, representing causal relationships. We assume no selection bias and that the block domain knowledge is correct and ordering. Correctness implies the existence of a CG that is consistent with the data and block domain knowledge. The correctness of block domain knowledge is a prerequisite for learning a reasonable chain graphs structure and an important guarantee for ensuring the accuracy of causal inference results.

Block domain knowledge arises in various situations, including but not limited to (i) relationships between units in social networks (Bhattacharya et al., 2020), (ii) the effect of vaccination of units on other units in epidemiology (Tchetgen Tchetgen et al., 2021; Vazquez-Bare, 2023), and (iii) relationships between users or between items in recommender systems (Chen et al., 2018). For instance, when studying the impact of vaccinating specific diseases among family members on disease transmission, we have come to understand that the vaccination status of different members influences one another, and the outcomes after vaccination or non-vaccination also interact with each other, with each member's vaccination affecting their own outcome causally. Based on this research background, we can extract block domain knowledge, grouping the vaccination statuses of different members into one block and categorizing the outcomes of members into another block. By utilizing block domain knowledge, we can establish a more precise causal model between vaccination among family members and the transmission of diseases, thereby aiding us in accurately assessing the impact of vaccination on disease spread. The specific details of the KDLCG algorithm are described below (Algorithm 2).

Step 1: Learn the Markov equivalence class of CG (lines 2-7). The KDLCG algorithm firstly connects $T \in V$ and $X \in Adj_T$ by undirected edges to construct the local skeleton of CG (lines 3-4), and then the KDLCG algorithm orients between T and X by conditional independence tests to obtain the local structure of target variable T, and finally the KDLCG algorithm integrates the local structure of all variables into the global structure (lines 5-7). The global structure obtained at this point is the Markov equivalence class of CG.

Step 2: Learning the structure of CG using Meek rules (lines 9-11). The KDLCG algorithm first extracts the block structure $B_G^{'}$ from the Markov equivalence class of CG driven by the block domain knowledge K (line 9). The definitions of block variables and block domain knowledge show that the true CG corresponds to the block structure B_G as a directed acyclic graph, and the $B_G^{'}$ is the Markov equivalent of B_G. Next, the KDLCG algorithm uses Meek rule R1 (citation (Meek, 1995)) to update the undirected edges in $B_G^{'}$ (line 10). Finally, the KDLCG algorithm utilizes the valid orientation rule proposed in this paper to refine the CG, which is to determine the direction of edges based on the fact that no additional complexes are created in CG.

The Meek rule **R1** is as follows: orient $X_j - X_k$ into $X_j \leftarrow X_k$ whenever there is a directed edge $X_i \leftarrow X_j$ such that X_i and X_k are not adjacent (otherwise a new v-structure is created).

The valid orientation rule: orient $X - Y$ into $X \rightarrow Y$ whenever $K_i - K_j(X_{Bi} - X_{Bj})$ in $B_G^{'}$ is oriented as $K_i \rightarrow K_j, X \in K_i, Y \in K_j$.

Step 3: Learning the structure of CG by estimating causal effects between variables (lines 13-19). The KDLCG algorithm first records the information of variable pairs und_{info} that are undirected edges between two variables in the updated $B_G^{'}$ (line 13), then it iterates through variable pairs K_i and K_j in und_{info} and estimates the causal effect value ce_{value} of K_i and K_j by using Equation 5 (lines 14-15). If $ce_{value} = 0$ (K_i is a parent of K_j), the KDLCG algorithm is concluded that K_i points to K_j in $B_G^{'}$. If $X \in K_i$, $Y \in K_j$ and $X - Y$, then X points to Y in G (lines 16-17). If ce_{value} contains 0 and $i < j$, then K_i points to K_j in $B_G^{'}$. If $X \in K_i$, $Y \in K_j$ and $X - Y$, then X points to Y in G (lines 18-19). Finally, the KDLCG algorithm proposed in this paper outputs a global CG structure (line 20).

Highlights of Algorithm 2: (1) The Markov equivalence classes of CGs are learned using the separated Adj and SP obtained by Algorithm 1. **(2)** To identify more causal relationships in the Markov equivalence classes of CGs, we define the block domain knowledge to drive CGs structure learning. First, the introduction of block domain knowledge is justified, as in sociological or epidemiological studies, where such knowledge is usually available from experts in the field of study to know which units (variables) belong to a group (block variables, block domain knowledge). The introduction of block domain knowledge provides a large amount of information for the learning of CGs structure. Second, the block domain knowledge reduces the search space for structure learning and improves the efficiency of structure learning. **(3)** In addition to this, we also refine the CGs by utilizing the Meek rule and estimating causal effects between the two variables, driven by the block domain knowledge.

Example 4. We use Toy graph as the example to trace the KDLCG algorithm, which can increase the readability of algorithm when learning the CG structure, see Figure 3 (F is orange, adjacencies are green and spouses are blue).

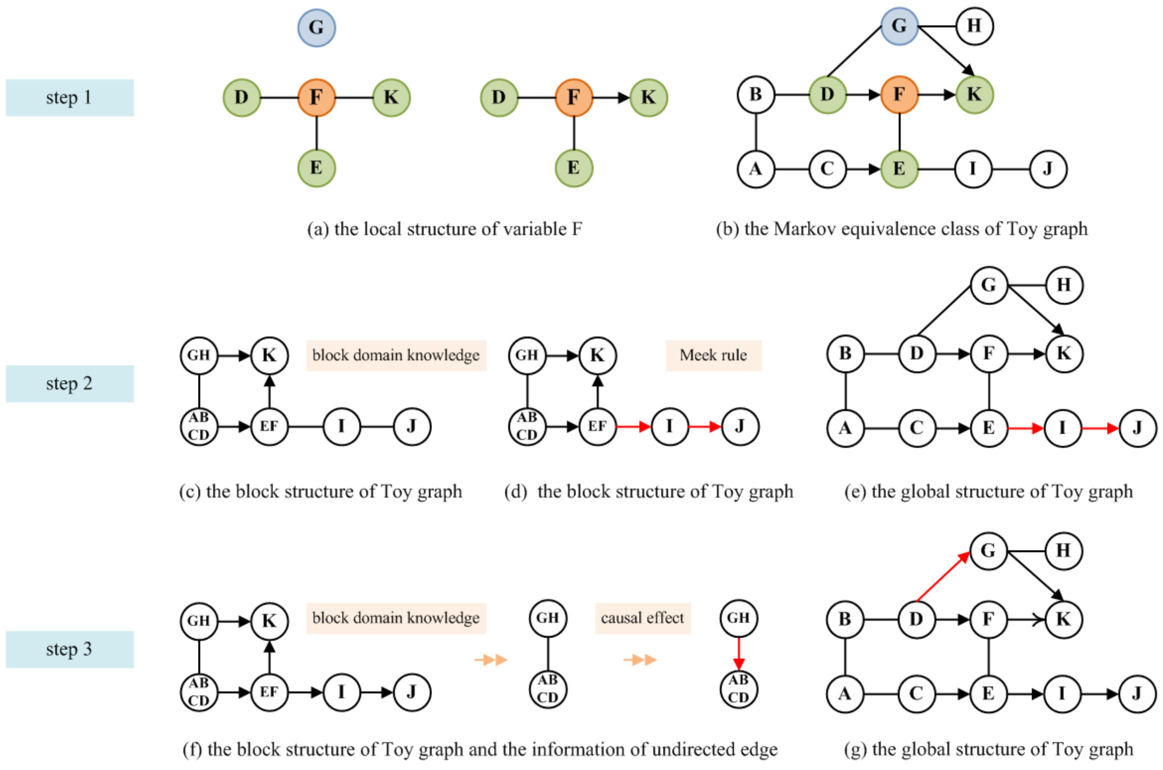

step 1
step 2
step 3
(a) the local structure of variable F
(b) the Markov equivalence class of Toy graph
block domain knowledge
Meek rule
(c) the block structure of Toy graph
(d) the block structure of Toy graph
(e) the global structure of Toy graph
causal effect
(f) the block structure of Toy graph and the information of undirected edge
(g) the global structure of Toy graph

Figure 3: An example of the execution process of KDLCG.

(1) Step 1: The KDLCG algorithm constructs the CG skeleton with the help of Adj of variables obtained above and learns the CG structure with the help of Adj and SP. Taking F as an example, the above obtained $Adj_F = \{K, E, D\}$, the KDLCG algorithm connects F to $\{K, E, D\}$ by undirected edges to construct the local skeleton of F, and then $F \not\perp\!\!\!\perp G|K$, which determines F to point to K, and finally obtains the local structure of F, see Figure 3(a). The KDLCG algorithm traverses all variables

and integrates the local structure of all variables into a global structure. At this point, the learned structure is the Markov equivalence class of Toy graphs, see Figure 3(b).

(2) Step 2: Guided by the block domain knowledge, we extract the block structure $B_G^{'}$ of the Toy graph from the Markov equivalence class of the Toy graph, see Figure 3(c). The KDLCG algorithm uses the Meek rule R1 (without creating a new V-structure) to determine $\{E, F\} \rightarrow I, I \rightarrow J$ in the block structure $B_G^{'}$, see Figure 3(d). Finally, the KDLCG algorithm uses the valid orientation rule to update the direction of undirected edges in CG, see Figure 3(e).

(3) Step 3: Guided by the block domain knowledge, we record the information of variable pairs $\{G, H\} - \{A, B, C, D\}$ that are undirected edges between variables in the updated block structure $B_G^{'}$, see Figure 3(f). And then the KDLCG algorithm estimates the value of causal effects of variable pairs $\{G, H\} - \{A, B, C, D\}$ using equation (5) (Maathuis et al., 2009), then $\{G, H\} \rightarrow \{A, B, C, D\}$, so $D \rightarrow G$. After the above learning, the KDLCG algorithm finally learns the Toy graph, see Figure 3(g).

3.2 Theoretical Analysis

The main focus of this paper is to identify more causal relationships in the Markov equivalence classes of CGs. In this section, we give the theoretical analysis of the KDLCG algorithm.

Theorem 1. Under the faithfulness assumption, if X and Y are adjacent in a LWF CG G, *iff* $\forall Z \subseteq V, X \not\perp\!\!\!\perp Y | Z$(Studenỳ, 1997).

Theorem 2. Under the faithfulness assumption, then for X in a LWF CG G, $X \perp\!\!\!\perp V \backslash \{X\} \backslash MB_X | MB_X$(Javidian et al., 2020b).

Theorem 3 (Correctness of the learn-AS algorithm) Under the faithfulness assumption, the learn-AS algorithm can correctly learn the *Adj* and *SP* from causally sufficient data.

Proof. According to Theorem 1, the learn-AS algorithm adds variables that are conditionally dependent of T into $CAdj_T$ and removes variables that are conditionally independent of T from $CAdj_T$. Since the true adjacencies is always dependent on T given any subsets in V, $CAdj_T$ contains all true adjacencies. After the finding of adjacencies, the learn-AS algorithm looks for the spouse variable with $X \in CAdj_T$. Before adding Y to CSPs, the learn-AS checks whether X is a true adjacencies. If X becomes conditionally independent on T (line 21), the learn-AS algorithm removes X from $CAdj_T$. If X becomes conditionally dependent on T (line 21), the learn-AS algorithm adds Y into CSP_T by Theorem 2. Steps 2 and 3 only remove false positive *Adj* and *SP* variables. After Step 1, $CAdj_T$ and CSP_T have included all true Adj_T and SP_T. The learn-AS algorithm uses Theorem 2 to remove false positives from $CAdj_T$ and CSP_T, and true Adj_T and SP_T will not be removed. Therefore, the learn-AS algorithm is correct. □

Theorem 4 (Correctness of the KDLCG algorithm) Under the faithfulness assumption, the KDLCG algorithm can correctly learn LWF CG structure from causally sufficient data.

Proof. After the learn-AS algorithm, Adj_T includes all adjacencies of T, and SP_T includes all spouses of T. Step 1 of the KDLCG algorithm is to learn the Markov equivalence class of CG. The KDLCG algorithm uses undirected edges to connect T variables to each variable in Adj_T to build the local skeleton. And then we know that every ordered variables triple $< T, X, Y >$ in G, if $T \to X$ is a complex arrow, there exists a variable $Y \in SP_T$. For the triple $< T, X, Y >$, the condition at line 6 is satisfied, $T \to X$ is returned on line 7, while $T \leftarrow X$ or $T - X$ is not returned. If $T \leftarrow X$ or $T - X$ in G, $X \in Bd_T \subseteq Adj_T$, $Y \in SP_T$, then Y is not a strict descendant or a adjacent node of T in G. Based on the local Markov property of CG, there exists a set $S \subseteq Adj_T$ including X, such that $T \perp\!\!\!\perp Y|S$. Consequently, the condition at line 6 is not satisfied, $T \to X$ is forbidden in the output. Hence, The KDLCG algorithm learns the local structure of the target variable. The KDLCG algorithm integrates the local structure of all variables into the global structure, and the global structure is the Markov equivalence class of CG at this time. Step 2 of the KDLCG algorithm refines the CG structure using the Meek rules. The KDLCG algorithm extracts the block structure $B_G^{'}$ from the Markov equivalence class of the chain graphs obtained in the previous step. At this time, the $B_G^{'}$ is also the Markov equivalence class of the block structure of true CG, i.e. there are undirected edges in $B_G^{'}$. Driven by the block domain knowledge K, the KDLCG algorithm uses Meek rule R1 to determine the direction of some edges in $B_G^{'}$ without creating a new V-structure and then updates the chain graphs structure by the valid orientation rules. Step 3 of the KDLCG algorithm further refines the CG structure driven by knowledge. The KDLCG algorithm uses Equation 5 (a causal effects-based approach) (Maathuis et al., 2009) to estimate the causal effects of variable pairs that have undirected edges between them in $B_G^{'}$. If the condition in line 16 and line 18 is satisfied, the relationship between the pairs of variables can be determined (line 17 and line 19). Finally, the KDLCG algorithm outputs the global chain graphs structure (line 20). Therefore, the KDLCG algorithm is correct. $\square$

4. Experimental Evaluation

In this section, we first list the experimental settings on synthetic data required for the comparison algorithms in Section 4.1, then analyze the experiment on the synthetic data in Section 4.2, and finally verify the effectiveness of the algorithms on the real-world data in Section 4.3.

To evaluate the performance of the proposed algorithm, we perform extensive experiments to contrast our proposed KDLCG algorithm against the state-of-the-art LCD algorithm and MbLWF algorithm. We implemented all algorithms in R by extending the code from the *bnlearn* (Javidian et al., 2020b), *lcd* (Ma et al., 2008), and *pcalg* (Kalisch, Mächler, Colombo, Maathuis, & Bühlmann, 2012) packages to LWF CGs. We evaluate the performance of the proposed algorithm in terms of the following six measurements:

(1) The true positive rate (TPR, also known as recall) is the ratio of the number of correctly identified edges (TP) over total number of edges (Pos),

(2) The false positive rate (FPR) is the ratio of the number of incorrectly identified edges (FP) over total number of gaps (Neg),

(3) The true discovery rate (TDR, also known as precision) is the ratio of the number of correctly identified edges over total number of edges (both in estimated graph),

(4) Accuracy (ACC) is $\frac{TP+TN}{Pos+Neg}$,

(5) The structural Hamming distance (SHD) is the number of legitimate operations needed to change the current resulting graph to the true CG, where legitimate operations are: (a) add or delete an edge and (b) insert, delete or reverse an edge orientation. This is the metric described in (Tsamardinos, Brown, & Aliferis, 2006) to compare the structure of the learned graphs and the ground truth graphs, and

(6) Running time is the time required to execute the algorithms.

Note that we use TPR, FPR, TDR, and ACC for comparing the skeleton of a learned structure and a ground truth graph. In principle, a large TPR, TDR, and ACC, a small FPR, SHD, and Running time indicate good performance.

4.1 The Experimental Setting on Synthetic Data

In this section, we list the data generation procedure in Section 4.1.1 and the experimental setting in Section 4.1.2.

4.1.1 Data Generation Procedure

First, we discuss how the random chain graphs and the Gaussian distribution are generated (Ma et al., 2008). Given a variable set V, let $p = |V|$ and N denote the average degree of edges (including undirected and pointing out and pointing in) for each variable. We generate a random chain graph on V as follows:

(1) Order the p vertices and initialize a $p \times p$ adjacency matrix A with zeros;

(2) For each element in the lower triangle part of A, set it to be a random number generated from a Bernoulli distribution with probability of occurrence $s = N/(p-1)$;

(3) Symmetrize A according to its lower triangle;

(4) Select an integer k randomly from $\{1, ..., p\}$ as the number of chain components;

(5) Split the interval $[1, p]$ into k equal-length subintervals $I_1, ..., I_k$ so that the set of variables falling into each subinterval I_m forms a chain component C_m;

(6) Set $A_{ij} = 0$ for any (i, j) pair such that $i \in I_l, j \in I_m$ with $l > m$.

This procedure then yields an adjacency matrix A for a chain graph with ($A_{ij} = A_{ji} = 1$) representing an undirected edge between V_i and V_j and ($A_{ij} = 1, A_{ji} = 0$) representing a directed edge from V_i to V_j. Moreover, it is not hard to see that $E[\text{vertex degree}] = N$ where an adjacent vertex can be linked by either an undirected or a directed edge. Given a randomly generated chain graph G with ordered chain components $C_1, ..., C_m$, we generate the Gaussian distribution from *lcd* R package (Ma et al., 2008).

4.1.2 The Experimental Setting on Synthetic Data

In order to assess the performance of the proposed algorithm, we consider comparing the performance of algorithms from both low-dimensional settings and high-dimensional settings.

The low-dimensional experimental settings: We consider the Toy graph in Figure 1 (a) and random CGs with p ={10, 20, 40} and N ={2, 3}, in which p is the number of variables and N represents the average number of adjacent variables per variable (including undirected, pointing out, and pointing in). For each combination (p, N), we first randomly generate CGs. Additionally, we generate a random Gaussian probability distribution based on each CG and obtain training databases with the sample size of $n = 500$ or 5000 from this probability distribution. For each sample, the significance levels *alpha* of the LCD algorithm, the MbLWF algorithm, and our KDLCG algorithm are respectively set at the values of 0.005 or 0.05 to perform the hypothesis.

The high-dimensional experimental settings: We consider random CGs with p ={300, 500}, and $N = 2$, in which p is the number of variables and N represents the average number of adjacent variables per variable (including undirected, pointing out, and pointing in). For each combination (p, N), we first randomly generate CGs. Additionally, we generate a random Gaussian probability distribution based on each CG and obtain training databases with the sample size of $n = 50$ or 100 from this probability distribution. For each sample, the significance levels *alpha* of the LCD algorithm, the MbLWF algorithm, and our KDLCG algorithm are respectively set at the values of 0.005 or 0.05 to perform the hypothesis.

4.2 The Experimental Analysis on Synthetic Data

In this section, we evaluate the performance of the proposed algorithm from the low-dimensional and high-dimensional experimental analysis.

4.2.1 The Low-Dimensional Experimental Analysis

We check the performance in terms of the TPR, FPR, TDR, ACC, SHD, and running time. Tables $3 - 6$ and Figure 4 show the corresponding experimental results on different CGs. Each reported statistic is the average and standard deviation values obtained by independently running 100 times the proposed algorithm in Tables $3 - 6$. Each reported statistic is the average value obtained by independently running 100 times the proposed algorithm in Figure 4. The following conclusions can be obtained from experimental results.

(1) From the perspective of TPR, the TPR values returned by our proposed KDLCG algorithm, the LCD algorithm, and the MbLWF algorithm increase as n increases. The TPR values decrease as p increases when N is the same, and the TPR values give better results at $\alpha = 0.05$ than at $\alpha = 0.005$, and the TPR values give better results at $N = 2$ than at $N = 3$ when p is the same. The TPR values of the KDLCG algorithm are large for the sample size $n = 500$ and the significance level $\alpha = 0.005$, see the third column of Table 3. In other cases, the TPR values of the LCD algorithm are large, see the third column of Tables 4, 5, 6. This is because the KDLCG algorithm prunes them multiple times when learning the adjacencies and spouses to learn accurate CG structure, see Algorithm 1. The pruning operation may make the KDLCG algorithm exclude some ambiguous but true

Table 3: Performance of our proposed KDLCG algorithm, the LCD algorithm and the MbLWF algorithm for $alpha = 0.005$ and $n = 500$.

Chain graphs	Algorithm	TPR	FPR	TDR	ACC
Toy graph	LCD	**0.9450 ± 0.0548**	0.0023 ± 0.0070	0.9913 ± 0.0263	0.9862 ± 0.0150
	MbLWF	0.9350 ± 0.0636	**0.0005 ± 0.0033**	**0.9985 ± 0.0109**	0.9855 ± 0.0137
	KDLCG	0.9370 ± 0.0642	**0.0005 ± 0.0033**	**0.9985 ± 0.0109**	**0.9863 ± 0.0130**
p=10, N=2	LCD	0.9400 ± 0.0571	**0.0011 ± 0.0057**	0.9958 ± 0.0209	0.9858 ± 0.0147
	MbLWF	**0.9520 ± 0.0579**	**0.0011 ± 0.0057**	**0.9960 ± 0.0200**	**0.9884 ± 0.0144**
	KDLCG	**0.9520 ± 0.0579**	**0.0011 ± 0.0057**	**0.9960 ± 0.0200**	**0.9884 ± 0.0144**
p=10, N=3	LCD	0.7893 ± 0.0845	**0.0067 ± 0.0135**	**0.9829 ± 0.0348**	0.9253 ± 0.0329
	MbLWF	0.7773 ± 0.0708	0.0093 ± 0.0191	0.9775 ± 0.0454	0.9196 ± 0.0298
	KDLCG	**0.8093 ± 0.0646**	0.0120 ± 0.0199	0.9727 ± 0.0456	**0.9284 ± 0.0270**
p=20, N=2	LCD	0.9300 ± 0.0505	**0.0002 ± 0.0012**	**0.9978 ± 0.0107**	0.9924 ± 0.0056
	MbLWF	**0.9370 ± 0.0472**	0.0004 ± 0.0016	0.9959 ± 0.0140	0.9929 ± 0.0052
	KDLCG	**0.9370 ± 0.0472**	0.0004 ± 0.0016	0.9960 ± 0.0139	**0.9930 ± 0.0051**
p=20, N=3	LCD	0.8553 ± 0.0658	0.0011 ± 0.0030	0.9935 ± 0.0173	0.9762 ± 0.0104
	MbLWF	**0.8613 ± 0.0483**	0.0071 ± 0.0040	0.9583 ± 0.0222	0.9721 ± 0.0079
	KDLCG	0.8550 ± 0.0512	**0.0001 ± 0.0009**	**0.9993 ± 0.0052**	**0.9770 ± 0.0081**
p=40, N=2	LCD	0.7670 ± 0.0656	0.0015 ± 0.0016	0.9664 ± 0.0337	0.9866 ± 0.0034
	MbLWF	0.7925 ± 0.0639	0.0017 ± 0.0015	0.9625 ± 0.0310	0.9877 ± 0.0033
	KDLCG	**0.7925 ± 0.0637**	**0.0015 ± 0.0014**	**0.9683 ± 0.0298**	**0.9880 ± 0.0033**
p=40, N=3	LCD	0.7750 ± 0.0390	0.0011 ± 0.0013	0.9840 ± 0.0191	0.9817 ± 0.0035
	MbLWF	0.7873 ± 0.0399	0.0009 ± 0.0010	**0.9867 ± 0.0145**	0.9828 ± 0.0034
	KDLCG	**0.7877 ± 0.0399**	**0.0009 ± 0.0009**	**0.9867 ± 0.0145**	**0.9828 ± 0.0033**

positives. Thus, the TPR values of the KDLCG algorithm are lower than the values of the LCD algorithm in some cases.

(2) From the perspective of FPR, the FPR values returned by our proposed KDLCG algorithm, the LCD algorithm, and the MbLWF algorithm roughly decrease as n increases. The FPR values get better results at $\alpha = 0.005$ than at $\alpha = 0.05$, and the FPR values get better results at $N = 2$ than at $N = 3$ when p is the same. The FPR values of the KDLCG algorithm are small in most settings, see the fourth column of Tables $3-6$. This is because the KDLCG algorithm prunes them multiple times when learning the adjacencies and spouses to learn accurate CG structure, see Algorithm 1. Thus, the FPR values obtained by the KDLCG algorithm are small.

(3) From the perspective of TDR, the TDR values returned by our proposed KDLCG algorithm, the LCD algorithm, and the MbLWF algorithm decrease as p increases when N are the same, and the TDR values give better results at $\alpha = 0.005$ than at $\alpha = 0.05$. The TDR values of the KDLCG algorithm are large for most cases, see the fifth column of Tables $3 - 6$. This is because the KDLCG algorithm prunes them multiple times when learning

Table 4: Performance of our proposed KDLCG algorithm, the LCD algorithm and the MbLWF algorithm for $alpha = 0.005$ and $n = 5000$.

Chain graphs	Algorithm	TPR	FPR	TDR	ACC
Toy graph	LCD	**0.9950 ± 0.0200**	0.0009 ± 0.0046	0.9969 ± 0.0152	**0.9980 ± 0.0060**
	MbLWF	0.9900 ± 0.0274	**0.0000 ± 0.0000**	**1.0000 ± 0.0000**	0.9978 ± 0.0060
	KDLCG	0.9900 ± 0.0274	**0.0000 ± 0.0000**	**1.0000 ± 0.0000**	**0.9980 ± 0.0060**
p=10, N=2	LCD	**1.0000 ± 0.0000**	0.0011 ± 0.0057	0.9964 ± 0.0180	0.9991 ± 0.0044
	MbLWF	0.9980 ± 0.0141	**0.0000 ± 0.0000**	**1.0000 ± 0.0000**	**0.9996 ± 0.0031**
	KDLCG	0.9980 ± 0.0141	**0.0000 ± 0.0000**	**1.0000 ± 0.0000**	**0.9996 ± 0.0031**
p=10, N=3	LCD	**0.9360 ± 0.0538**	0.0127 ± 0.0177	0.9743 ± 0.0356	0.9702 ± 0.0240
	MbLWF	0.9173 ± 0.0596	0.0240 ± 0.0278	0.9517 ± 0.0536	0.9564 ± 0.0327
	KDLCG	0.9280 ± 0.0644	**0.0073 ± 0.0139**	**0.9853 ± 0.0282**	**0.9711 ± 0.0230**
p=20, N=2	LCD	**1.0000 ± 0.0000**	0.0009 ± 0.0025	0.9925 ± 0.0197	0.9992 ± 0.0022
	MbLWF	**1.0000 ± 0.0000**	**0.0006 ± 0.0018**	**0.9952 ± 0.0143**	0.9994 ± 0.0020
	KDLCG	**1.0000 ± 0.0000**	0.0007 ± 0.0019	0.9943 ± 0.0156	**0.9994 ± 0.0017**
p=20, N=3	LCD	**0.9940 ± 0.0129**	**0.0000 ± 0.0000**	**1.0000 ± 0.0000**	0.9991 ± 0.0020
	MbLWF	0.9900 ± 0.0168	0.0133 ± 0.0059	0.9342 ± 0.0273	0.9873 ± 0.0052
	KDLCG	0.9780 ± 0.0257	0.0002 ± 0.0012	0.9990 ± 0.0063	**0.9996 ± 0.0040**
p=40, N=2	LCD	**0.9995 ± 0.0035**	0.0015 ± 0.0015	0.9730 ± 0.0258	0.9985 ± 0.0014
	MbLWF	**0.9995 ± 0.0035**	**0.0013 ± 0.0013**	**0.9770 ± 0.0216**	**0.9987 ± 0.0012**
	KDLCG	**0.9995 ± 0.0035**	**0.0013 ± 0.0013**	**0.9770 ± 0.0216**	**0.9987 ± 0.0012**
p=40, N=3	LCD	**0.9970 ± 0.0065**	0.0007 ± 0.0009	0.9922 ± 0.0109	**0.9992 ± 0.0009**
	MbLWF	0.9953 ± 0.0089	0.0010 ± 0.0012	0.9880 ± 0.0130	0.9987 ± 0.0012
	KDLCG	0.9953 ± 0.0089	**0.0004 ± 0.0009**	**0.9945 ± 0.0101**	**0.9992 ± 0.0009**

the adjacencies and spouses to learn accurate CG structure, see Algorithm 1. The Pruning operation also means removing as many false positives as possible from the adjacencies and spouses and ensuring that the adjacencies and spouses contain as many true positives as possible. Due to the three pruning of the adjacencies and spouses, the TDR values obtained by the KDLCG algorithm are large.

(4) From the perspective of ACC, the ACC values returned by our proposed KDLCG algorithm, the LCD algorithm, and the MbLWF algorithm increase as n increases, and the ACC values give better results at $\alpha = 0.05$ than at $\alpha = 0.005$ when the sample size is 5000 ($n = 5000$). The ACC values of the KDLCG algorithm are large in most settings, see the sixth column of Tables $3 - 6$.

(5) From the perspective of SHD, the SHD values returned by our proposed KDLCG algorithm, the LCD algorithm, and the MbLWF algorithm decrease as n increases. The SHD values increase as p increases when N is the same, and the SHD values increase as N increases when p is the same, and the SHD values give better results at $\alpha = 0.005$ than at $\alpha = 0.05$, see Figure 4. The SHD values of the KDLCG algorithm are relatively small in all

Table 5: Performance of our proposed KDLCG algorithm, the LCD algorithm and the MbLWF algorithm for $alpha = 0.05$ and $n = 500$.

Chain graphs	Algorithm	TPR	FPR	TDR	ACC
Toy graph	LCD	**0.9617 ± 0.0511**	0.0088 ± 0.0140	0.9695 ± 0.0478	0.9847 ± 0.0185
	MbLWF	0.9600 ± 0.0564	0.0042 ± 0.0102	0.9853 ± 0.0353	0.9880 ± 0.0163
	KDLCG	0.9600 ± 0.0564	**0.0033 ± 0.0082**	**0.9883 ± 0.0294**	**0.9887 ± 0.0155**
p=10, N=2	LCD	**0.9760 ± 0.0431**	0.0080 ± 0.0153	0.9745 ± 0.0483	0.9884 ± 0.0157
	MbLWF	0.9740 ± 0.0443	**0.0051 ± 0.0138**	**0.9835 ± 0.0434**	**0.9902 ± 0.0156**
	KDLCG	0.9740 ± 0.0443	**0.0051 ± 0.0138**	**0.9835 ± 0.0434**	**0.9902 ± 0.0156**
p=10, N=3	LCD	0.8573 ± 0.0632	**0.0127 ± 0.0163**	**0.9722 ± 0.0361**	**0.9440 ± 0.0247**
	MbLWF	0.8387 ± 0.0647	0.0233 ± 0.0303	0.9510 ± 0.0590	0.9307 ± 0.0297
	KDLCG	**0.8600 ± 0.0508**	0.0150 ± 0.0220	0.9673 ± 0.0459	0.9431 ± 0.0242
p=20, N=2	LCD	**0.9810 ± 0.0301**	0.0094 ± 0.0074	0.9276 ± 0.0547	0.9896 ± 0.0075
	MbLWF	0.9780 ± 0.0310	0.0082 ± 0.0072	0.9362 ± 0.0528	0.9903 ± 0.0071
	KDLCG	0.9780 ± 0.0306	**0.0078 ± 0.0690**	**0.9396 ± 0.0509**	**0.9907 ± 0.0070**
p=20, N=3	LCD	**0.9280 ± 0.0549**	0.0055 ± 0.0057	0.9698 ± 0.0308	0.9840 ± 0.0108
	MbLWF	0.9260 ± 0.0453	0.0120 ± 0.0055	0.9362 ± 0.0271	0.9782 ± 0.0080
	KDLCG	0.9133 ± 0.0467	**0.0027 ± 0.0382**	**0.9845 ± 0.0214**	**0.9840 ± 0.0084**
p=40, N=2	LCD	0.8900 ± 0.0479	0.0190 ± 0.0049	0.7199 ± 0.0570	0.9763 ± 0.0057
	MbLWF	0.9000 ± 0.0420	0.0175 ± 0.0049	0.7384 ± 0.0575	0.9782 ± 0.0055
	KDLCG	**0.9000 ± 0.0415**	**0.0170 ± 0.0049**	**0.7441 ± 0.0590**	**0.9787 ± 0.0055**
p=40, N=3	LCD	0.8773 ± 0.0319	0.0120 ± 0.0036	0.8605 ± 0.0367	0.9795 ± 0.0039
	MbLWF	**0.8833 ± 0.0338**	0.0112 ± 0.0041	0.8696 ± 0.0426	0.9807 ± 0.0050
	KDLCG	0.8820 ± 0.0348	**0.0099 ± 0.0035**	**0.8828 ± 0.0375**	**0.9818 ± 0.0043**

cases, see Figure 4. It is because the KDLCG algorithm not only performs structure learning to obtain the Markov equivalence classes of the chain graphs like the LCD algorithm and the MbLWF algorithm, but the KDLCG algorithm also orients some undirected edges in the Markov equivalence classes, guided by block domain knowledge. Thus, the KDLCG algorithm can obtain relatively small SHD values.

(6) From the perspective of running time, The running times returned by our proposed KDLCG algorithm, the LCD algorithm, and the MbLWF algorithm increase as n increases, and the running times give better results at $\alpha = 0.005$ than at $\alpha = 0.05$, see Figure 5. Compared with the MbLWF algorithm, the KDLCG algorithm has a short local running time in learning Markov equivalence classes. In all settings, the KDLCG algorithm has a longer running time. This is because the KDLCG algorithm further orients some undirected edges in the Markov equivalence class under the guidance of block domain knowledge.

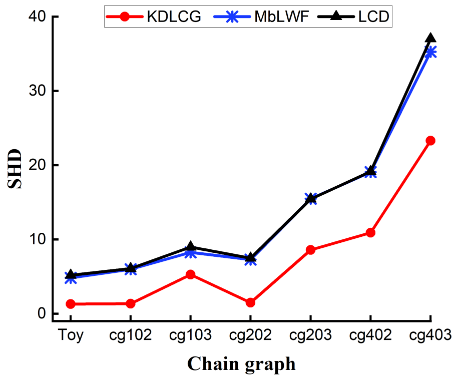
Summarizing, the KDLCG algorithm outperforms the LCD algorithm and the MbLWF algorithm in FPR, TDR, ACC and SHD. The experimental results of low-dimensional setting show that the KDLCG exhibits better performance.

Table 6: Performance of our proposed KDLCG algorithm, the LCD algorithm and the MbLWF algorithm for $alpha = 0.05$ and $n = 5000$.

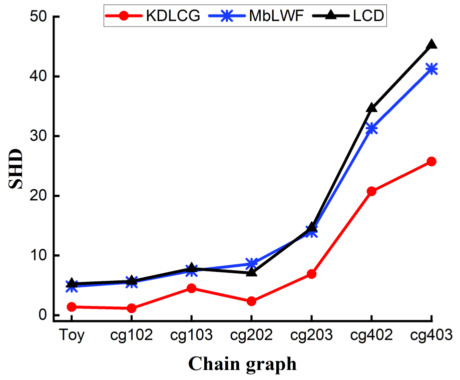
Chain graphs	Algorithm	TPR	FPR	TDR	ACC
Toy graph	LCD	**0.9950 ± 0.0200**	0.0102 ± 0.0134	0.9665 ± 0.0434	0.9909 ± 0.0111
	MbLWF	0.9933 ± 0.0228	0.0060 ± 0.0103	0.9797 ± 0.0345	0.9938 ± 0.0101
	KDLCG	0.9917 ± 0.0252	**0.0042 ± 0.0090**	**0.9860 ± 0.0301**	**0.9950 ± 0.0090**
p=10, N=2	LCD	**1.0000 ± 0.0000**	0.0086 ± 0.0166	0.9737 ± 0.0500	0.9933 ± 0.0123
	MbLWF	**1.0000 ± 0.0000**	0.0074 ± 0.0127	0.9764 ± 0.0403	0.9942 ± 0.0098
	KDLCG	**1.0000 ± 0.0000**	**0.0057 ± 0.0115**	**0.9818 ± 0.0367**	**0.9956 ± 0.0089**
p=10, N=3	LCD	**0.9427 ± 0.0603**	0.0167 ± 0.0254	0.9671 ± 0.0481	**0.9698 ± 0.0307**
	MbLWF	0.8387 ± 0.0647	0.0233 ± 0.0303	0.9510 ± 0.0590	0.9307 ± 0.0297
	KDLCG	0.9333 ± 0.0659	**0.0160 ± 0.0204**	**0.9679 ± 0.0412**	0.9671 ± 0.0281
p=20, N=2	LCD	**1.0000 ± 0.0000**	0.0116 ± 0.0089	0.9139 ± 0.0598	0.9896 ± 0.0079
	MbLWF	**1.0000 ± 0.0000**	0.0101 ± 0.0087	0.9246 ± 0.0577	0.9909 ± 0.0077
	KDLCG	**1.0000 ± 0.0000**	**0.0096 ± 0.0082**	**0.9278 ± 0.0563**	**0.9914 ± 0.0073**
p=20, N=3	LCD	**0.9967 ± 0.0101**	0.0054 ± 0.0054	0.9728 ± 0.0268	0.9949 ± 0.0049
	MbLWF	0.9920 ± 0.0144	0.0221 ± 0.0088	0.8952 ± 0.0381	0.9801 ± 0.0078
	KDLCG	0.9830 ± 0.0215	**0.0014 ± 0.0029**	**0.9930 ± 0.0150**	**0.9961 ± 0.0041**
p=40, N=2	LCD	**1.0000 ± 0.0000**	0.0196 ± 0.0050	0.7374 ± 0.0522	0.9814 ± 0.0048
	MbLWF	**1.0000 ± 0.0000**	0.0180 ± 0.0047	0.7535 ± 0.0514	0.9829 ± 0.0045
	KDLCG	**1.0000 ± 0.0000**	**0.0174 ± 0.0050**	**0.7606 ± 0.0561**	**0.9835 ± 0.0048**
p=40, N=3	LCD	**0.9983 ± 0.0051**	0.0117 ± 0.0035	0.8777 ± 0.0314	0.9891 ± 0.0031
	MbLWF	0.9980 ± 0.0064	0.0122 ± 0.0034	0.8734 ± 0.0299	0.9886 ± 0.0030
	KDLCG	0.9980 ± 0.0064	**0.0098 ± 0.0033**	**0.8957 ± 0.0315**	**0.9908 ± 0.0030**

4.2.2 The High-Dimensional Experimental Analysis

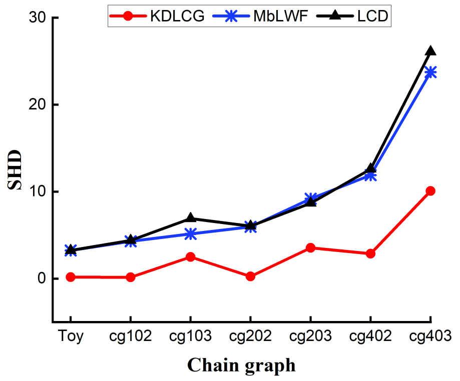
We check the performance in terms of the TPR, FPR, TDR, ACC, and SHD. Figures $6-11$ and Tables $7-8$ show the corresponding experimental results on different CGs. Each reported statistic is the average and standard deviation values obtained by independently running 100 times the proposed algorithm in Tables $8-9$. Each reported statistic is the average value obtained by independently running 100 times the algorithms in Figure 6. Each reported statistic is the result of 100 independent runs of the algorithm in Figures $7-11$, where the black line in a box indicates the median of the group and the green point in a box indicates the average value for that group. The measurement values of the LCD algorithm in the table are "$-$", and the measurement values of the LCD algorithm are not shown in the figures. Since the CGs that generate the datasets are very sparse (p is large and N is small), it causes the LCD algorithm to make an error when building the separation tree. The following conclusions can be obtained from experimental results.



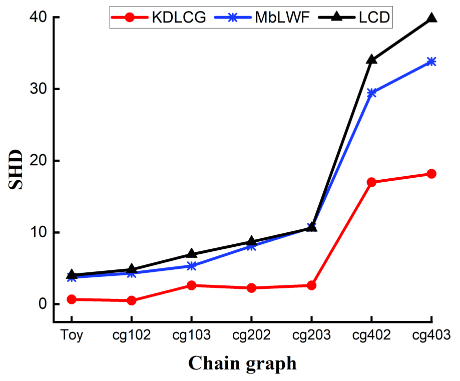
(a) $alpha = 0.005, n = 500$



(b) $alpha = 0.05, n = 500$



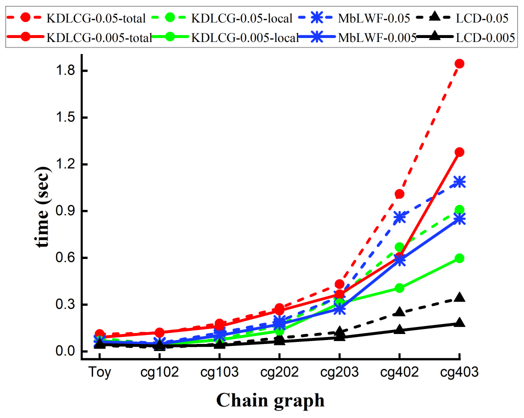
(c) $alpha = 0.005, n = 5000$



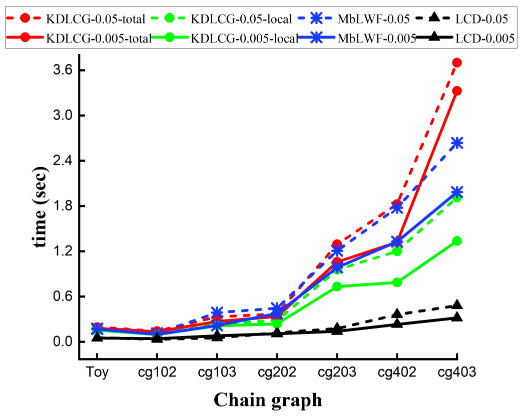
(d) $alpha = 0.05, n = 5000$

Figure 4: The SHD performance of our proposed KDLCG algorithm, the LCD algorithm and the MbLWF algorithm. To facilitate the presentation of the experimental results graph, we abbreviate $p = 10, N = 2$ as "cg102". The other abbreviations are "cg103, cg202, cg203, cg402, cg403". The red line indicates the KDLCG algorithm, the blue line indicates the MbLWF algorithm, and the black line indicates the LCD algorithm.

(1) From the perspective of TPR, the TPR values returned by our proposed KDLCG algorithm and the MbLWF algorithm increase as n increases, and the TPR values give better results at $\alpha = 0.05$ than at $\alpha = 0.005$. The TPR values of the KDLCG algorithm are larger for the sample size $n = 50$, and the TPR values of the KDLCG algorithm are smaller for the sample size $n = 100$, see the third column of Tables 7, 8 and Figure 7. This is because the KDLCG algorithm prunes them multiple times when learning the adjacencies and spouses to learn accurate CG structure, see Algorithm 1. The pruning operation may make the KDLCG algorithm delete true false positive when $n = 50$ (the sample size is small, which indicates the amount of information contained in the dataset is small) and may make



(a) $alpha = 0.005, n = 500$



(b) $alpha = 0.05, n = 500$

Figure 5: The running time of our proposed KDLCG algorithm, the LCD algorithm, and the MbLWF algorithm. To facilitate the presentation of the experimental results graph, we abbreviate $p = 10, N = 2$ as ”cg102”. The other abbreviations are ”cg103, cg202, cg203, cg402, cg403”. The red line indicates the total time for the KDLCG algorithm to learn the chain graphs driven by knowledge, The green line indicates the local time for the KDLCG algorithm to learn Markov equivalence classes, the blue line indicates the MbLWF algorithm and the black line indicates the LCD algorithm.

Table 7: Performance of our proposed KDLCG algorithm, the LCD algorithm and the MbLWF algorithm for $p = 500$, $N = 2$ and $n = 50$.

alpha	Algorithm	TPR	FPR	TDR	ACC
0.005	LCD	–	–	–	–
	MbLWF	0.4568 ± 0.0118	**0.0002 ± 0.0000**	0.8918 ± 0.0184	0.9975 ± 0.0000
	KDLCG	**0.4694 ± 0.0133**	**0.0002 ± 0.0000**	**0.8941 ± 0.0158**	**0.9976 ± 0.0000**
0.05	LCD	–	–	–	–
	MbLWF	0.4329 ± 0.0243	0.0017 ± 0.0002	0.5068 ± 0.0200	0.9960 ± 0.0001
	KDLCG	**0.5662 ± 0.0154**	**0.0011 ± 0.0000**	**0.6748 ± 0.0193**	**0.9971 ± 0.0001**

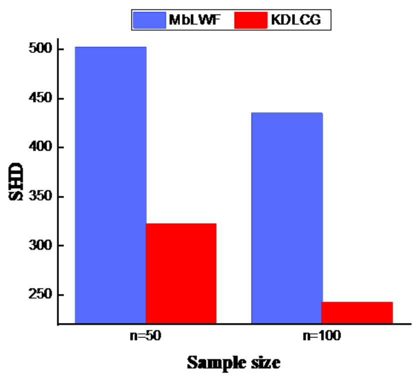
the KDLCG algorithm exclude some ambiguous but true positives when $n = 100$. Thus, the TPR values of the KDLCG algorithm are lower than the values of the LCD algorithm in some cases.

(2) From the perspective of FPR, the FPR values returned by our proposed KDLCG algorithm and the MbLWF algorithm roughly decrease as n increases, and FPR values give better results at $\alpha = 0.005$ than at $\alpha = 0.05$. The TPR values of the KDLCG algorithm are smaller for most cases, see the fourth column of Tables 7, 8 and Figure 10. This is because the KDLCG algorithm prunes them multiple times when learning the adjacencies

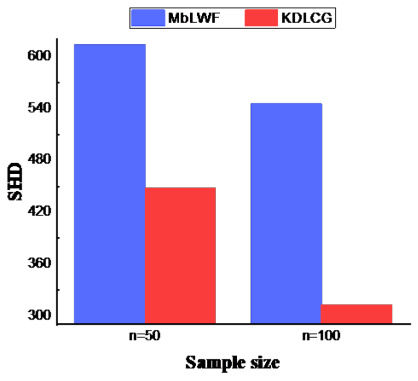
Table 8: Performance of our proposed KDLCG algorithm, the LCD algorithm and the MbLWF algorithm for $p = 500$, $N = 2$ and $n = 100$.

alpha	Algorithm	TPR	FPR	TDR	ACC
0.005	LCD	–	–	–	–
	MbLWF	**0.6457 ± 0.0147**	0.0003 ± 0.0000	0.9116 ± 0.0121	0.9983 ± 0.0000
	KDLCG	0.6446 ± 0.0143	**0.0002 ± 0.0000**	**0.9311 ± 0.0118**	**0.9984 ± 0.0000**
0.05	LCD	–	–	–	–
	MbLWF	**0.7529 ± 0.0160**	0.0020 ± 0.0001	0.6105 ± 0.0176	0.9970 ± 0.0002
	KDLCG	0.7417 ± 0.0133	**0.0011 ± 0.0000**	**0.7296 ± 0.0159**	**0.9978 ± 0.0001**

and spouses to learn accurate CG structure, see Algorithm 1. Because of the three pruning of the adjacencies and spouses, the FPR values obtained by the KDLCG algorithm are small.



(a) *alpha* = 0.005

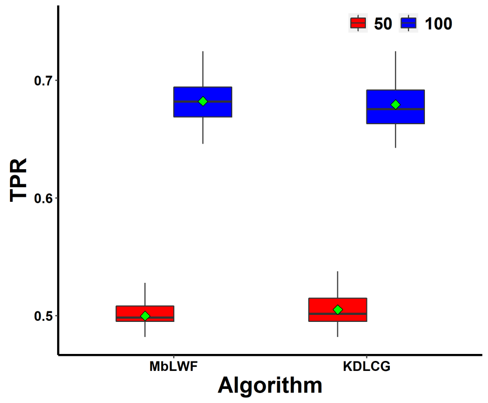


(b) *alpha* = 0.05

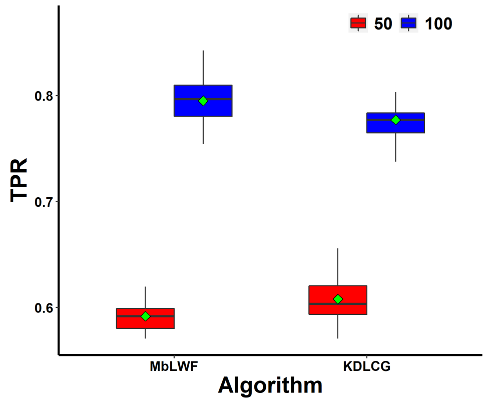
Figure 6: The SHD performance comparison between our proposed KDLCG algorithm and MbLWF algorithm for $p = 500$, $N = 2$. The red box indicates $n = 50$ and the blue box indicates $n = 100$. The black line in a box indicates the median of the group. The green point in a box indicates the mean value for that group.

(3) From the perspective of TDR, the TDR values returned by our proposed KDLCG algorithm and the MbLWF algorithm decrease as p increases when N is the same, and the TDR values give better results at $\alpha = 0.005$ than at $\alpha = 0.05$. The TDR values of the KDLCG algorithm are larger for most cases, see the fifth column of Tables 7, 8 and Figure 8. This is because the KDLCG algorithm prunes them multiple times when learning the adjacencies and spouses to learn accurate CG structure, see Algorithm 1. The Pruning operation also means removing as many false positives as possible from the adjacencies and spouses and ensuring that the adjacencies and spouses contain as many true positives as possible. Thus, the TDR values obtained by the KDLCG algorithm are larger.

(4) From the perspective of ACC, the ACC values returned by our proposed KDLCG algorithm and the MbLWF algorithm increase as n increases. The ACC values of the KDLCG algorithm are larger for most cases, see the sixth column of Tables 7, 8 and Figure 9. This is because the KDLCG algorithm prunes them multiple times when learning the adjacencies and spouses to learn accurate CG structure, see Algorithm 1. Thus, the ACC values obtained by the KDLCG algorithm are larger.

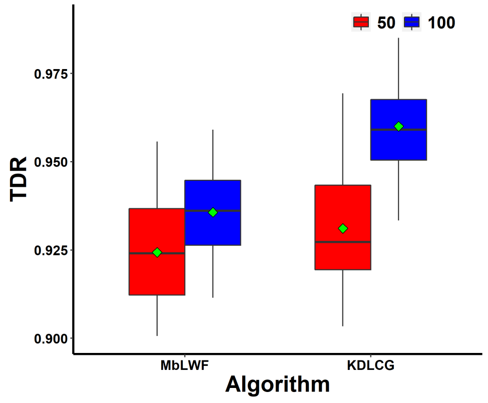


(a) $alpha = 0.005$

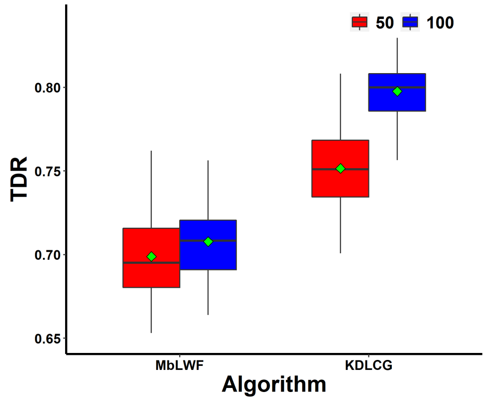


(b) $alpha = 0.05$

Figure 7: The TPR performance comparison between our proposed KDLCG algorithm and MbLWF algorithm for $p = 300$, $N = 2$. The red box indicates $n = 50$ and the blue box indicates $n = 100$. The black line in a box indicates the median of the group. The green point in a box indicates the mean value for that group.

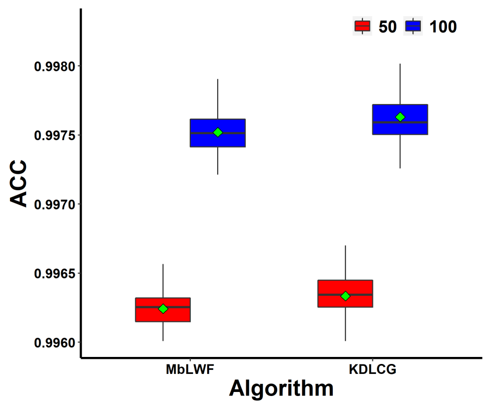


(a) $alpha = 0.005$

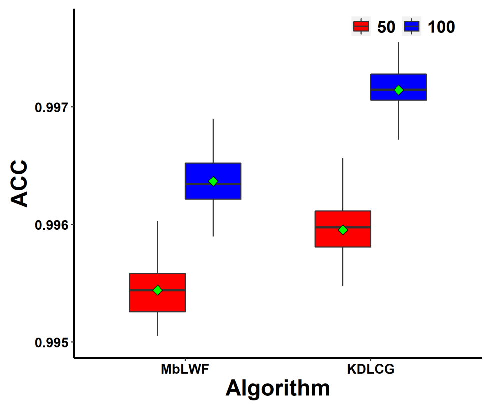


(b) $alpha = 0.05$

Figure 8: The TDR performance comparison between our proposed KDLCG algorithm and MbLWF algorithm for $p = 300$, $N = 2$. The red box indicates $n = 50$ and the blue box indicates $n = 100$. The black line in a box indicates the median of the group. The green point in a box indicates the mean value for that group.

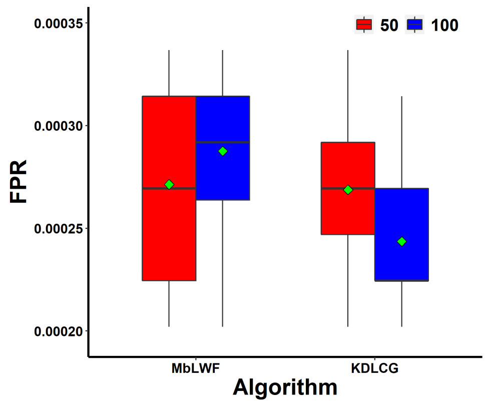


(a) $alpha = 0.005$

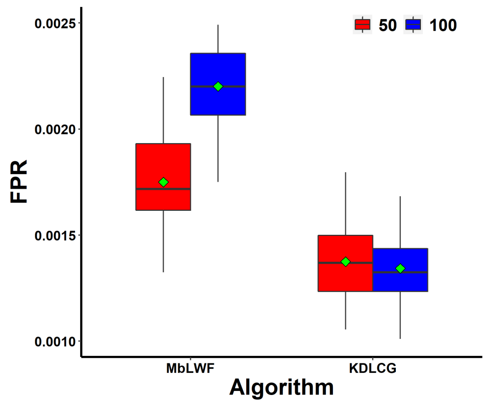


(b) $alpha = 0.05$

Figure 9: The ACC performance comparison between our proposed KDLCG algorithm and MbLWF algorithm for $p = 300$, $N = 2$. The red box indicates $n = 50$ and the blue box indicates $n = 100$. The black line in a box indicates the median of the group. The green point in a box indicates the mean value for that group.



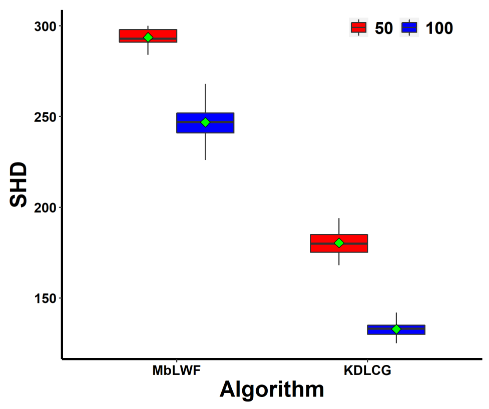
(a) $alpha = 0.005$



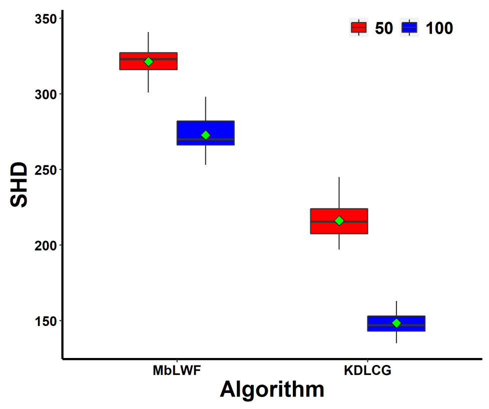
(b) $alpha = 0.05$

Figure 10: The FPR performance comparison between our proposed KDLCG algorithm and MbLWF algorithm for $p = 300$, $N = 2$. The red box indicates $n = 50$ and the blue box indicates $n = 100$. The black line in a box indicates the median of the group. The green point in a box indicates the mean value for that group.

(5) From the perspective of SHD, The SHD values returned by our proposed KDLCG algorithm and the MbLWF algorithm decrease as n increases. The SHD values increase as p increases when N is the same, and the SHD values increase as N increases when p is the same, and the SHD values give better results at $\alpha = 0.005$ than at $\alpha = 0.05$, see Figures 6 and 11. The SHD values of the KDLCG algorithm are relatively small in all cases. It is because the KDLCG algorithm not only performs structure learning to obtain the Markov



(a) $alpha = 0.005$



(b) $alpha = 0.05$

Figure 11: The SHD performance comparison between our proposed KDLCG algorithm and MbLWF algorithm for $p = 300$, $N = 2$. The red box indicates $n = 50$ and the blue box indicates $n = 100$. The black line in a box indicates the median of the group. The green point in a box indicates the mean value for that group.

equivalence classes of the chain graphs like the LCD algorithm and the MbLWF algorithm, but the KDLCG algorithm also orients some undirected edges in the Markov equivalence classes, guided by block domain knowledge. Therefore, the KDLCG algorithm can obtain relatively small SHD values.

In summary, the KDLCG algorithm outperforms the LCD algorithm and the MbLWF algorithm in TPR, FPR, TDR, ACC, and SHD. The experimental results of the high-dimensional setting show that the KDLCG exhibits better performance.

4.2.3 Summary of Experimental Analysis

We verify the performance of the algorithms in low-dimensional and high-dimensional settings, respectively, and conduct experimental analyses. In the overall context, the sample size n, the density of the chain graphs (the number of variables p and the number of adjacent variables N), and the significance level α have an impact on the performance of the algorithms.

In terms of n, the TPR, TDR, and ACC values increase with increasing n, while the FPR and SHD values decrease with increasing n. This suggests that as the size of n increases, the datasets contain richer information, which provides greater certainty for learning the structure of CGs.

In terms of the density of the chain graphs, when p is held constant, the TPR, TDR, and ACC values decrease as N increases, while the FPR and SHD values become larger. This is because an increase in N indicates a denser CG structure and a dense network structure implies a more complex relationship between variables, making it more challenging to learn CG. Conversely, when N is held constant, the TPR, TDR, and ACC values generally decrease as p increases, while the FPR and SHD values tend to increase. This is because an increase in p corresponds to a larger CG structure, and the larger the network structure, the more complex the calculation.

In terms of α, the TPR, TDR, and ACC values increase with increasing α, and the FPR and SHD values become larger. This is because an increase in α adds possible true positives to the set when the algorithm conducts conditionally independent tests, resulting in higher TPR, TDR, and ACC values. However, the inclusion of potential true positives also introduces additional false positive variables to the set, leading to higher FPR and SHD values.

In summary, the performance of the algorithms is influenced by n, the density of the chain graphs, and α, so we conduct experiments on our proposed KDLCG algorithm, LCD algorithm, and MbLWF algorithm using different parameter settings. In both low-dimensional and high-dimensional settings, the TPR values of KDLCG are lower than the other algorithms. This is because our algorithm removes some ambiguous true positives through pruning operations. In low-dimensional and high-dimensional settings, the FPR, TDR, ACC, and SHD values of our proposed algorithm outperformed other algorithms in most cases. In conclusion, our proposed KDLCG algorithm outperforms the comparison algorithms.

4.3 Experiments with the Real-World Datasets

In this section, we evaluate all algorithms on the insilico_size10 dataset in the DREAM4 Network Inference Challenge (Marbach, Schaffter, Mattiussi, & Floreano, 2009; Greenfield, Madar, Ostrer, & Bonneau, 2010). The DREAM4 datasets use GeneNetWeaver (GNW) (Schaffter, Marbach, & Floreano, 2011) to extract sub-gene regulatory networks from the real E.coli transcriptional regulatory network (1502 nodes, 3587 edges) (Gama-Castro, Jiménez-Jacinto, Peralta-Gil, Santos-Zavaleta, Peñaloza-Spinola, Contreras-Moreira, Segura-Salazar, Muniz-Rascado, Martinez-Flores, Salgado, et al., 2008) and the Yeast transcriptional regulatory network (4441 nodes, 12873 edges) (Balaji, Babu, Iyer, Luscombe, & Aravind, 2006). GNW then generates approximately real-world gene expression datasets corresponding to the sub-networks by considering biological factors such as noise, time delays, and feedback loops.

In this section, we chose to verify the effectiveness of all algorithms on time series datasets of insilico_size10_1 and insilico_size10_2 with feedback loops ($n = 105$, $p = 10$). In addition, the literatures show that LWF CGs can be used to model a network containing feedback loops (Lauritzen & Richardson, 2002; Sonntag & Peña, 2015b). Therefore, we group the variables in the network that form a feedback loop into a block and repeat this process until the set of variables in the entire network is divided into multiple blocks. We use these blocks as prior domain block knowledge to better guide network Inference.

We check the performance in terms of the TPR, FPR, TDR, ACC, and SHD. Table 9 shows the corresponding experimental results on insilico_size10_1 and insilico_size10_2. The experimental results in Table 9 show that our proposed KDLCG algorithm outperforms the comparison algorithms in terms of TPR, FPR, TDR, ACC, and SHD evaluation metrics. Considering the small number of samples in this real-world dataset and the limited information it can provide, coupled with the presence of noise, time delays, feedback loops, etc., the data complexity increases. As a result, the LCD and MbLWF algorithms do not perform well in inferring network structures from this real-world dataset. However, our proposed KDLCG algorithm gains more information in structure learning guided by domain block knowledge. Consequently, KDLCG effectively reduces potential errors and improves the performance of network structure learning. Therefore, our KDLCG algorithm exhibits superior performance in handling real-world datasets.

Table 9: Performance of our proposed KDLCG algorithm, the LCD algorithm and the MbLWF algorithm for the insilico_size10 datasets in the DREAM4 Network Inference Challenge.

network	α	Algorithm	TPR	FPR	TDR	ACC	SHD
insilico_size10_1	0.005	LCD	**0.3846**	**0.0313**	**0.8333**	0.8000	13.0000
		MbLWF	0.3077	**0.0313**	0.8000	0.7778	13.0000
		KDLCG	**0.3846**	**0.0313**	**0.8333**	0.8000	**10.0000**
	0.05	LCD	**0.4615**	0.1407	0.5834	0.7445	16.0000
		MbLWF	0.3846	0.0938	0.6250	0.7556	15.0000
		KDLCG	**0.4615**	**0.0625**	**0.7500**	**0.8000**	**12.0000**
insilico_size10_2	0.005	LCD	0.2500	0.1515	0.3750	0.6889	17.0000
		MbLWF	0.2500	**0.090**	0.5000	0.7333	15.0000
		KDLCG	**0.5000**	0.1212	**0.6000**	**0.7778**	**13.0000**
	0.05	LCD	**0.5000**	**0.2121**	0.4167	0.6889	19.0000
		MbLWF	0.3333	0.2424	0.3333	0.6464	20.0000
		KDLCG	**0.5000**	**0.2121**	**0.4615**	**0.7111**	**17.0000**

5. Discussion and Conclusion

In this paper, we defined the block domain knowledge and proposed the KDLCG algorithm to identify more causal relationships in the Markov equivalence classes in chain graphs structure learning. In our work, the KDLCG algorithm firstly uses the learned adjacencies and spouses by the learn-AS algorithm to learn the Markov equivalence class of the chain graphs. Then, the KDLCG algorithm oriented the undirected edges in the Markov equivalence class using the Meek rules and the estimation of the causal effects between variables, driven by block domain knowledge. Meanwhile, the correctness of the KDLCG algorithm is proved from theoretical analysis, and the effectiveness of the KDLCG algorithm is verified from experimental tests.

In fact, domain knowledge is crucial for discovering causal relationships in chain graphs. When dealing with real-world data, the quality of data affects the accuracy of the learned structures. Incorporating prior domain knowledge can effectively mitigate potential errors. However, this study has some limitations. The KDLCG algorithm for the utilization of domain knowledge relies on the assumption that the domain knowledge is consistent with the chain graph (or the learned graph), and it also assumes no selection bias in the data. Therefore, in future work, we will consider generalizing chain graphs structure learning to inconsistent domain knowledge. Additionally, we will explore representing and utilizing knowledge in systems with selection bias, further improving the performance and expanding the application scope of chain graphs learning.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No.62376145, 61976128), and the Science and Technology Innovation Talent Team of Shanxi Province (Grant No.202204051002016).

References

Andersson, S. A., Madigan, D., & Perlman, M. D. (1996). An alternative markov property for chain graphs. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence*, pp. 40–48. AI.

Andrews, B., Spirtes, P., & Cooper, G. F. (2020). On the completeness of causal discovery in the presence of latent confounding with tiered background knowledge. In *Proceedings of the 23th International Conference on Artificial Intelligence and Statistics*, pp. 4002–4011. PMLR.

Balaji, S., Babu, M. M., Iyer, L. M., Luscombe, N. M., & Aravind, L. (2006). Comprehensive analysis of combinatorial regulation using the transcriptional regulatory network of yeast. *Journal of molecular biology*, *360*(1), 213–227.

Bhattacharya, R., Malinsky, D., & Shpitser, I. (2020). Causal inference under interference and network uncertainty. In *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence*, pp. 1028–1038. PMLR.

Chen, C., Chang, K. C.-C., Li, Q., & Zheng, X. (2018). Semi-supervised learning meets factorization: Learning to recommend with chain graph model. *ACM Transactions on Knowledge Discovery from Data*, *12*(6), 1–24.

Cowell, R. G., Dawid, P., Lauritzen, S. L., & Spiegelhalter, D. J. (2007). *Probabilistic networks and expert systems: Exact computational methods for Bayesian networks*. Springer Science & Business Media.

Cox, D. R., & Wermuth, N. (2014). *Multivariate dependencies: Models, analysis and interpretation*. Chapman and Hall/CRC.

Eigenmann, M. F., Nandy, P., & Maathuis, M. H. (2017). Structure learning of linear gaussian structural equation models with weak edges. In *Proceedings of the 33th Conference on Uncertainty in Artificial Intelligence*.

Gama-Castro, S., Jiménez-Jacinto, V., Peralta-Gil, M., Santos-Zavaleta, A., Peñaloza-Spinola, M. I., Contreras-Moreira, B., Segura-Salazar, J., Muniz-Rascado, L., Martinez-Flores, I., Salgado, H., et al. (2008). Regulondb (version 6.0): gene regulation model of escherichia coli k-12 beyond transcription, active (experimental) annotated promoters and textpresso navigation. *Nucleic acids research*, *36*(suppl_1), D120–D124.

Greenfield, A., Madar, A., Ostrer, H., & Bonneau, R. (2010). Dream4: Combining genetic and dynamic information to identify biological networks and dynamical models. *PloS one*, *5*(10), e13397.

Javidian, M. A. (2019). *Properties, Learning Algorithms, and Applications of Chain Graphs and Bayesian Hypergraphs*. Ph.D. thesis, University of South Carolina.

Javidian, M. A., Valtorta, M., & Jamshidi, P. (2020a). Amp chain graphs: Minimal separators and structure learning algorithms. *Journal of Artificial Intelligence Research*, *69*, 419–470.

Javidian, M. A., Valtorta, M., & Jamshidi, P. (2020b). Learning lwf chain graphs: A markov blanket discovery approach. In *Proceedings of the 36th conference on Uncertainty in Artificial Intelligence*, pp. 1069–1078. PMLR.

Javidian, M. A., Valtorta, M., & Jamshidi, P. (2020c). Learning lwf chain graphs: an order independent algorithm. *arXiv preprint arXiv:2005.14037*.

Kalisch, M., Mächler, M., Colombo, D., Maathuis, M. H., & Bühlmann, P. (2012). Causal inference using graphical models with the r package pcalg. *Journal of statistical software*, *47*(11), 1–26.

Lappenschaar, M., Hommersom, A., & Lucas, P. J. (2014). Qualitative chain graphs and their application. *International Journal of Approximate Reasoning*, *55*(4), 957–976.

Lauritzen, S. L., & Wermuth, N. (1989). Graphical models for associations between variables, some of which are qualitative and some quantitative. *The Annals of Statistics*, *17*(1), 31–57.

Lauritzen, S. L. (1996). *Graphical models*, Vol. 17. Clarendon Press.

Lauritzen, S. L., & Richardson, T. S. (2002). Chain graph models and their causal interpretations. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *64*(3), 321–348.

Ma, Z., Xie, X., & Geng, Z. (2008). Structural learning of chain graphs via decomposition. *Journal of Machine Learning Research*, *9*(Dec), 2847–2880.

Maathuis, M. H., Kalisch, M., & Bühlmann, P. (2009). Estimating high-dimensional intervention effects from observational data. *The Annals of Statistics*, *37*(6A), 3133–3164.

Marbach, D., Schaffter, T., Mattiussi, C., & Floreano, D. (2009). Generating realistic in silico gene networks for performance assessment of reverse engineering methods. *Journal of computational biology*, *16*(2), 229–239.

Margaritis, D., & Thrun, S. (1999). Bayesian network induction via local neighborhoods. *Advances in neural information processing systems*, *12*.

Meek, C. (1995). Causal inference and causal explanation with background knowledge. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, pp. 403–410.

Neuberg, L. G. (2003). Causality: models, reasoning, and inference, by judea pearl, cambridge university press, 2000. *Econometric Theory*, *19*(4), 675–685.

Ogburn, E. L., Shpitser, I., & Lee, Y. (2020). Causal inference, social networks and chain graphs. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, *183*(4), 1659–1676.

Pearl, J. (1995). Causal diagrams for empirical research. *Biometrika*, *82*(4), 669–688.

Pearl, J. (2003). Statistics and causal inference: A review. *Test*, *12*(2), 281–345.

Peña, J. M., Sonntag, D., & Nielsen, J. D. (2014). An inclusion optimal algorithm for chain graph structure learning. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*, pp. 778–786. PMLR.

Perkovic, E. (2020). Identifying causal effects in maximally oriented partially directed acyclic graphs. In *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence*, pp. 530–539. PMLR.

Rothenhäusler, D., Ernest, J., & Bühlmann, P. (2018). Causal inference in partially linear structural equation models. *The Annals of Statistics*, *46*(6A), 2904–2938.

Schäfer, J., & Strimmer, K. (2005). An empirical bayes approach to inferring large-scale gene association networks. *Bioinformatics*, *21*(6), 754–764.

Schaffter, T., Marbach, D., & Floreano, D. (2011). Genenetweaver: in silico benchmark generation and performance profiling of network inference methods. *Bioinformatics*, *27*(16), 2263–2270.

Scheines, R., Spirtes, P., Glymour, C., Meek, C., & Richardson, T. (1998). The tetrad project: Constraint based aids to causal model specification. *Multivariate Behavioral Research*, *33*(1), 65–117.

Shen, Y., & Cremers, D. (2020). A chain graph interpretation of real-world neural networks. *arXiv preprint arXiv:2006.16856*.

Sherman, E., & Shpitser, I. (2018). Identification and estimation of causal effects from dependent data. *Advances in neural information processing systems*, *31*, 9424–9435.

Shpitser, I., Tchetgen, E. T., & Andrews, R. (2017). Modeling interference via symmetric treatment decomposition. *arXiv preprint arXiv:1709.01050*.

Sonntag, D., Järvisalo, M., Peña, J. M., & Hyttinen, A. (2015). Learning optimal chain graphs with answer set programming. In *Proceedings of the 31th Conference on Uncertainty in Artificial Intelligence*, pp. 822–831.

Sonntag, D., & Peña, J. M. (2015a). Chain graph interpretations and their relations revisited. *International Journal of Approximate Reasoning*, *58*, 39–56.

Sonntag, D., & Peña, J. M. (2015b). Chain graphs and gene networks. *Foundations of Biomedical Knowledge Representation: Methods and Applications*, 159–178.

Sonntag, D., Peña, J. M., & Gómez-Olmedo, M. (2015). Approximate counting of graphical models via mcmc revisited. *International Journal of Intelligent Systems*, *30*(3), 384–420.

Studenỳ, M. (1997). A recovery algorithm for chain graphs. *International Journal of Approximate Reasoning*, *17*(2-3), 265–293.

Studenỳ, M., & Bouckaert, R. R. (1998). On chain graph models for description of conditional independence structures. *The Annals of Statistics*, *26*(4), 1434–1495.

Tchetgen Tchetgen, E. J., Fulcher, I. R., & Shpitser, I. (2021). Auto-g-computation of causal effects on a network. *Journal of the American Statistical Association*, *116*(534), 833–844.

Tsamardinos, I., Aliferis, C. F., Statnikov, A. R., & Statnikov, E. (2003). Algorithms for large scale markov blanket discovery.. In *Proceedings of the 16th Conference on Florida Artificial Intelligence Research Society*, Vol. 2, pp. 376–380. St. Augustine, FL.

Tsamardinos, I., Brown, L. E., & Aliferis, C. F. (2006). The max-min hill-climbing bayesian network structure learning algorithm. *Machine learning*, *65*(1), 31–78.

Vazquez-Bare, G. (2023). Causal spillover effects using instrumental variables. *Journal of the American Statistical Association*, *118*(543), 1–12.

Wang, J., Liu, S., & Zhu, M. (2019). Local structure learning of chain graphs with the false discovery rate control. *Artificial Intelligence Review*, *52*(1), 293–321.

Wang, Y., & Bhattacharyya, A. (2022). Identifiability of linear AMP chain graph models. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence*, pp. 10080–10089. AAAI Press.

Xu, H., Fard, N., & Fang, Y. (2020). Time series chain graph for modeling reliability covariates in degradation process. *Reliability Engineering & System Safety*, *204*, 107207.

Yaramakala, S., & Margaritis, D. (2005). Speculative markov blanket discovery for optimal feature selection. In *Proceedings of the 5th IEEE International Conference on Data Mining*, pp. 809–812. IEEE.

Yu, C. L., Airoldi, E. M., Borgs, C., & Chayes, J. T. (2022). Estimating the total treatment effect in randomized experiments with unknown network structure. *Proceedings of the National Academy of Sciences*, *119*(44), e2208975119.