Published in final edited form as:
J Mach Learn Res. 2008 December 1; 9: 2847–2880.

Structural Learning of Chain Graphs via Decomposition

Zongming Ma[*], **Xianchao Xie**[†], and **Zhi Geng**
School of Mathematical Sciences, LMAM, Peking University, Beijing 100871, China

Zongming Ma: ZONGMING@STANFORD.EDU; Xianchao Xie: XXIE@FAS.HARVARD.EDU; Zhi Geng: ZGENG@MATH.PKU.EDU.CN

Abstract

Chain graphs present a broad class of graphical models for description of conditional independence structures, including both Markov networks and Bayesian networks as special cases. In this paper, we propose a computationally feasible method for the structural learning of chain graphs based on the idea of decomposing the learning problem into a set of smaller scale problems on its decomposed subgraphs. The decomposition requires conditional independencies but does not require the separators to be complete subgraphs. Algorithms for both skeleton recovery and complex arrow orientation are presented. Simulations under a variety of settings demonstrate the competitive performance of our method, especially when the underlying graph is sparse.

Keywords

chain graph; conditional independence; decomposition; graphical model; structural learning

1. Introduction

Graphical models are widely used to represent and analyze conditional independencies and causal relationships among random variables. Monographs on this topic include Cowell et al. (1999), Cox and Wermuth (1996), Edwards (2000), Lauritzen (1996), Pearl (1988) and Spirtes et al. (2000). Two most well-known classes of graphical models are *Markov networks* (undirected graph) and *Bayesian networks* (directed acyclic graph). Wermuth and Lauritzen (1990) introduced the broader class of *block-recursive graphical models* (chain graph models), which includes, but is not limited to, the above two classes.

Among a multitude of research problems about graphical models, structural learning (also called model selection in statistics community) has been extensively discussed and continues to be a field of great interest. There are primarily two categories of methods: score-based methods (using AIC, BIC, posterior score, etc.) and constraint-based methods (using significance testing). Lauritzen (1996, Section 7.2) provides a good summary of the most important works done in the last century. Recent works in this area include Ravikumar et al. (2008), Friedman et al. (2007), Kalisch and Bühlmann (2007), Meinshausen and Bühlmann (2006), Tsamardinos et al. (2006), Ellis and Wong (2006), Friedman and Koller (2003), Chickering (2002), Friedman et al. (1999), etc. However, most of these studies are exclusively concerned with either Markov networks or Bayesian networks. To our limited knowledge, Studený (1997) is the only work that addresses the issue of learning chain graph structures in the literature. Recently, Drton and Perlman (2008) studied the special case of Gaussian chain

Editor: David Maxwell Chickering
[*]Also in the Department of Statistics, Stanford University, Stanford, CA 94305.
[†]Also in the Department of Statistics, Harvard University, Cambridge, MA 02138.

graph models using a multiple testing procedure, which requires prior knowledge of the dependence chain structure.

Chain graph models are most appropriate when there are both response-explanatory and symmetric association relations among variables, while Bayesian networks specifically deal with the former and Markov networks focus on the later. Given the complexity of many modern systems of interest, it is usually desirable to include both types of relations in a single model. See also Lauritzen and Richardson (2002).

As a consequence of the versatility, chain graph models have received a growing attention as a modeling tool in statistical applications recently. For instance, Stanghellini et al. (1999) constructed a chain graph model for credit scoring in a case study in finance. Carroll and Pavlovic (2006) employed chain graphs to classify proteins in bioinformatics, and Liu et al. (2005) used them to predict protein structures. However, in most applications, chain graphs are by far not as popular as Markov networks and Bayesian networks. One important reason, we believe, is the lack of readily available algorithms for chain graph structure recovery.

To learn the structure of Bayesian networks, Xie et al. (2006) proposed a 'divide-and-conquer' approach. They showed that the structural learning of the whole DAG can be decomposed into smaller scale problems on (overlapping) subgraphs. By localizing the search of *d*-separators, their algorithm can reduce the computational complexity greatly.

In this paper, we focus on developing a computationally feasible method for structural learning of chain graphs along with this decomposition approach. As in structural learning of a Bayesian network, our method starts with finding a decomposition of the entire variable set into subsets, on each of which the local skeleton is then recovered. However, unlike the case of Bayesian networks, the structural learning of chain graph models is more complicated due to the extended Markov property of chain graphs and the presence of both directed and undirected edges. In particular, the rule in Xie et al. (2006) for combining local structures into a global skeleton is no longer applicable and a more careful work must be done to ensure a valid combination. Moreover, the method for extending a global skeleton to a Markov equivalence class is significantly different from that for Bayesian networks.

In particular, the major contribution of the paper is twofold: (a) for learning chain graph skeletons, an algorithm is proposed which localizes the search for *c*-separators and has a much reduced runtime compared with the algorithm proposed in Studený (1997); (b) a polynomial runtime algorithm is given for extending the chain graph skeletons to the Markov equivalence classes. We also demonstrate the efficiency of our methods through extensive simulation studies.

The rest of the paper is organized as follows. In Section 2, we introduce necessary background for chain graph models and the concept of decomposition via separation trees. In Section 3, we present the theoretical results, followed by a detailed description of our learning algorithms. Moreover, we discuss the issue of how to construct a separation tree to represent the decomposition. The computational analysis of the algorithms are presented in Section 4. Numerical experiments are reported in Section 5 to demonstrate the performance of our method. Finally, we conclude with some discussion in Section 6. Proofs of theoretical results and correctness of algorithms are shown in Appendices.

2. Definitions and Preliminaries

In this section, we first introduce the necessary graphical model terminology in Section 2.1 and then give the formal definition of separation trees in Section 2.2.

2.1 Graphical Model Terminology

For self-containedness, we briefly introduce necessary definitions and notations in graph theory here, most of which follow those in Studený (1997) and Studený and Bouckaert (2001). For a general account, we refer the readers to Cowell et al. (1999) and Lauritzen (1996).

A graph $\mathcal{G} = (V, E)$ consists of a vertex set V and an edge set E. For vertices $u, v \in V$, we say that there is an undirected edge $u - v$ if $(u, v) \in E$ and $(v, u) \in E$. If $(u, v) \in E$ and $(v, u) \notin E$, we say that there is a directed edge from u to v and write $u \to v$. We call undirected edges *lines* and directed edges *arrows*. The *skeleton* $\mathcal{G}'$ of a graph $\mathcal{G}$ is the undirected graph obtained by replacing the arrows of $\mathcal{G}$ by lines. If every edge in graph $\mathcal{G}$ is undirected, then for any vertex u, we define the *neighborhood* $\mathrm{ne}_{\mathcal{G}}(u)$ of u to be the set of vertices v such that $u - v$ in $\mathcal{G}$.

A *route* in $\mathcal{G}$ is a sequence of vertices $(v_0, \cdots, v_k)$, $k \geq 0$, such that $(v_{i-1}, v_i) \in E$ or $(v_i, v_{i-1}) \in E$ for $i = 1, \cdots, k$, and the vertices v_0 and v_k are called *terminals* of the route. It is called *descending* if $(v_{i-1}, v_i) \in E$ for $i = 1, \cdots, k$. We write $u \mapsto v$ if there exists a descending route from u to v. If $v_0, \cdots, v_k$ are distinct vertices, the route is called a *path*. A route is called a *pseudocycle* if $v_0 = v_k$, and a *cycle* if further $k \geq 3$ and $v_0, \cdots, v_{k-1}$ are distinct. A (pseudo) cycle is *directed* if it is descending and there exists at least one $i \in \{1, \cdots, k\}$, such that $(v_i, v_{i-1}) \in \notin E$.

A graph with only undirected edges is called an *undirected graph* (UG). A graph with only directed edges and without directed cycles is called a *directed acyclic graph* (DAG). A graph that has no directed (pseudo) cycles is called a *chain graph*.

By a *section* of a route $\rho = (v_0, \cdots, v_k)$ in $\mathcal{G}$, we mean a maximal undirected subroute $\sigma: v_i - \cdots - v_j$, $0 \leq i \leq j \leq k$ of ρ. The vertices v_i and v_j are called the *terminals* of the section σ. Further the vertex v_i (or v_j) is called a *head-terminal* if $i > 0$ and $v_{i-1} \to v_i$ in $\mathcal{G}$ (or $j < k$ and $v_j \leftarrow v_{j+1}$ in $\mathcal{G}$), otherwise it is called a *tail-terminal*. A section σ of a route ρ is called a *head-to-head* section with respect to ρ if it has two head-terminals, otherwise it is called *non head-to-head*. For a set of vertices $S \subset V$, we say that a section $\sigma: v_i - \cdots - v_j$ is *outside* S if $\{v_i, \cdots, v_j\} \cap S = \varnothing$. Otherwise we say that σ is *hit* by S.

A *complex* in $\mathcal{G}$ is a path $(v_0, \cdots, v_k)$, $k \geq 2$, such that $v_0 \to v_1$, $v_i - v_{i+1}$ (for $i = 1, \cdots, k-2$) and $v_{k-1} \leftarrow v_k$ in $\mathcal{G}$, and no additional edge exists among vertices $\{v_0, \cdots, v_k\}$ in $\mathcal{G}$. We call the vertices v_0 and v_k the *parents* of the complex and the set $\{v_1, \cdots, v_{k-1}\}$ the *region* of the complex. The set of parents of a complex κ is denoted by par(κ). Note that the concept of complex was proposed in Studený (1997) and is equivalent to the notion of 'minimal complex' in Frydenberg (1990).

An arrow in a graph $\mathcal{G}$ is called a *complex arrow* if it belongs to a complex in $\mathcal{G}$. The *pattern* of $\mathcal{G}$, denoted by $\mathcal{G}^*$, is the graph obtained by turning the arrows that are not in any complex of $\mathcal{G}$ into lines. The *moral graph* $\mathcal{G}^m$ of $\mathcal{G}$ is the graph obtained by first joining the parents of each complex by a line and then turning arrows of the obtained graph into lines.

Studený and Bouckaert (2001) introduced the notion of *c-separation* for chain graph models. Hereunder we introduce the concept in the form that facilitates the proofs of our results. We say that a route ρ on $\mathcal{G}$ 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.

Definition 1—Let A, B, S be three disjoint subsets of the vertex set V of a chain graph $\mathcal{G}$, such that A, B are nonempty. We say that A and B are c-separated by S on $\mathcal{G}$, written as $\langle A, B|S\rangle_{\mathcal{G}}^{sep}$, if every route with one of its terminals in A and the other in B is intervented by S. We also call S a c-separator for A and B.

For a chain graph $\mathcal{G} = (V, E)$, let each $v \in V$ be associated with a random variable X_v with domain $\mathcal{X}_v$ and μ the underlying probability measure on $\Pi_{v\in V}\mathcal{X}_v$. A probability distribution P on $\Pi_{v\in V}\mathcal{X}_v$ is *strictly positive* if $dP(x)/d\mu > 0$ for any $x \in \Pi_{v\in V}\mathcal{X}_v$. From now on, all probability distributions considered are assumed to be strictly positive. A probability distribution P on $\Pi_{v\in V}\mathcal{X}_v$ is *faithful* with respect to $\mathcal{G}$ if for any triple (A, B, S) of disjoint subsets of V such that A and B are non-empty, we have

$$\langle A, B|S\rangle_{\mathcal{G}}^{sep} \iff X_A \perp\!\!\!\perp X_B|X_S, \tag{1}$$

where $X_A = \{X_v: v \in A\}$ and $X_A \perp\!\!\!\perp X_B|X_S$ means the conditional independency of X_A and X_B given X_S; P is *Markovian* with respect to $\mathcal{G}$ if (1) is weakened to

$$\langle A, B|S\rangle_{\mathcal{G}}^{sep} \Rightarrow X_A \perp\!\!\!\perp X_B|X_S.$$

In the rest of the paper, $A \perp\!\!\!\perp B|S$ is used as short notation for $X_A \perp\!\!\!\perp X_B|X_S$ when confusion is unlikely.

It is known that chain graphs can be classified into *Markov equivalence classes*, and those in the same equivalence class share the same set of Markovian distributions. The following result from Frydenberg (1990) characterizes equivalence classes graphically: Two chain graphs are Markov equivalent if and only if they have the same skeleton and complexes, that is, they have the same pattern. The following example illustrates some of the concepts that are introduced above.

Example 1—Consider the chain graph $\mathcal{G}$ in Fig. 1(a). $D \to F - E \leftarrow C$ and $F \to K \leftarrow G$ are the two complexes. The route $\rho = (D, F, E, I, E, C)$ is intervented by an empty set since the head-to-head section $(E \to)I(\leftarrow E)$ is outside the empty set. It is also intervented by E since the non head-to-head section $(D \to)F - E(\to I)$ is hit by E. However, D and C are not c-separated by E since the route (D, F, E, C) is not intervented by E. The moral graph $\mathcal{G}^m$ of $\mathcal{G}$ is shown in Fig. 1(b), where edges $C - D$ and $F - G$ are added due to moralization.

2.2 Separation Trees

In this subsection, we introduce the notion of *separation trees* which is used to facilitate the representation of the decomposition. The concept is similar to the junction tree of cliques and the independence tree introduced for DAG as '*d*-separation trees'(Xie et al., 2006).

Let $\mathcal{C} = \{C_1, \cdots, C_H\}$ be a collection of distinct variable sets such that for $h = 1, \cdots, H$, $C_h \subseteq V$. Let $\mathcal{T}$ be a tree where each node corresponds to a distinct variable set in $\mathcal{C}$, to be displayed as a triangle (see, for example, Fig. 2). The term 'node' is used for a separation tree to distinguish from the term 'vertex' for a graph in general. An undirected edge $e = (C_i, C_j)$ connecting nodes C_i and C_j in $\mathcal{T}$ is attached with a *separator* $S = C_i \cap C_j$, which is displayed as a rectangle. A separator S is *connected to a node* C if there is some other node C', such that S attaches to the edge (C, C'). Removing an edge e or equivalently, removing a separator S from $\mathcal{T}$ splits $\mathcal{T}$ into two subtrees $\mathcal{T}_1$ and $\mathcal{T}_2$ with node sets $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively. We use $V_i =$

$\cup_{C\in\mathcal{C}_i} C$ to denote the unions of the vertices contained in the nodes of the subtree $\mathcal{T}_i$ for $i = 1, 2$.

Definition 2—A tree $\mathcal{T}$ with node set $\mathcal{C}$ is said to be a separation tree for a chain graph $\mathcal{G} = (V, E)$ if

1. $\cup_{C\in\mathcal{C}} C = V$, and
2. for any separator S in $\mathcal{T}$ with V_1 and V_2 defined as above by removing S, we have $\langle V_1\backslash S, V_2\backslash S | S\rangle_{\mathcal{G}}^{sep}$.

Notice that a separator is defined in terms of a tree whose nodes consist of variable sets, while the *c*-separator is defined based on a chain graph. In general, these two concepts are not related, though for a separation tree its separator must be some corresponding *c*-separator in the underlying chain graph.

Example 1. (Continued)—Suppose that

$$\mathcal{C}=\{\{A,B,C\},\{B,C,D\},\{C,D,E\},\{D,E,F\},\{E,I,J\},\{D,F,G,H,K\}\}$$

is a collection of vertex sets. A separation tree $\mathcal{T}$ of $\mathcal{G}$ in Fig. 1(a) with node set $\mathcal{C}$ is shown in Fig. 2. If we delete the separator $\{D, E\}$, we obtain two subtrees $\mathcal{T}_1$ and $\mathcal{T}_2$ with node sets $\mathcal{C}_1 = \{\{A, B,C\}, \{B,C, D\}, \{C, D, E\}\}$ and $\mathcal{C}_2 = \{\{D, E, F\}, \{E, I, J\}, \{D, F, G, H, K\}\}$. In $\mathcal{G}$, the separator $S = \{D, E\}$ *c*-separates $V_1\backslash S = \{A, B,C\}$ and $V_2\backslash S = \{F, G, H, I, J, K\}$.

Not surprisingly, the separation tree could be regarded as a scheme for decomposing the knowledge represented by the chain graph into local subsets. Reciprocally, given a separation tree, we can combine the information obtained locally to recover the global information, an idea that will be formalized and discussed in subsequent sections.

The definition of separation trees for chain graphs is similar to that of junction trees of cliques, see Cowell et al. (1999) and Lauritzen (1996). Actually, it is not difficult to see that a junction tree of a chain graph $\mathcal{G}$ is also its separation tree. However, we point out two differences here: (a) a separation tree is defined with *c*-separation and it does not require that every node is a clique or that every separator is complete on the moral graph; (b) junction trees are mostly used as inference engines, while our interest in separation trees is mainly derived from its power in facilitating the decomposition of structural learning.

3. Structural Learning of Chain Graphs

In this section, we discuss how separation trees can be used to facilitate the decomposition of the structural learning of chain graphs. Theoretical results are presented first, followed by descriptions of several algorithms that are the summary of the key results in our paper. It should be emphasized that even with perfect knowledge on the underlying distribution, any two chain graph structures in the same equivalence class are indistinguishable. Thus, we can only expect to recover a pattern from the observed data, that is, an equivalence class to which the underlying chain graph belongs. To this end, we provide two algorithms: one addresses the issue of learning the skeleton and the other focuses on extending the learned skeleton to an equivalence class. We also discuss at the end of the section the problem of constructing separation trees. Throughout the rest of the paper, we assume that any distribution under consideration is faithful with respect to some underlying chain graph.

3.1 Theoretical Results

It is known that for P faithful to a chain graph $\mathcal{G} = (V, E)$, an edge (u, v) is in the skeleton $\mathcal{G}'$ if and only if $X_u \not\perp\!\!\!\perp X_v | X_S$ for any $S \subseteq V \backslash \{u, v\}$ (see Studený 1997, Lemma 3.2 for a proof). Therefore, learning the skeleton of $\mathcal{G}$ reduces to searching for c-separators for all vertex pairs. The following theorem shows that with a separation tree, one can localize the search into one node or a small number of tree nodes.

Theorem 3—Let $\mathcal{T}$ be a separation tree for a chain graph $\mathcal{G}$. Then vertices u and v are c-separated by some set $S_{uv} \cup V$ in $\mathcal{G}$ if and only if one of the following conditions hold:

1. u and v are not contained together in any node C of $\mathcal{T}$,
2. u and v are contained together in some node C, but for any separator S connected to C, $\{u, v\} \not\subseteq S$, and there exists $S'_{uv} \subseteq C$ such that $\langle u, v | S'_{uv} \rangle^{sep}_{\mathcal{G}}$,
3. u and v are contained together in a node C and both of them belong to some separator connected to C, but there is a subset S'_{uv} of either $\cup_{u \in C'} C'$ or $\cup_{v \in C'} C'$ such that $\langle u, v | S'_{uv} \rangle^{sep}_{\mathcal{G}}$.

For DAGs, condition 3 in Theorem 3 is unnecessary, see Xie et al. (2006, Theorem 1). However, the example below indicates that this is no longer the case for chain graphs.

Example 1. (Continued)—Consider the chain graph in Fig. 1(a) and its separation tree in Fig. 2. Let $u = D$ and $v = E$. The tree nodes containing u and v together are $\{C, D, E\}$ and $\{D, E, F\}$. Since the separator $\{D, E\}$ is connected to both of them, neither condition 1 nor 2 in Theorem 3 is satisfied. Moreover, $u \not\perp\!\!\!\perp v | S$ for $S = \emptyset$, $\{C\}$ or $\{F\}$. However, we have $u \perp\!\!\!\perp v | \{C, F\}$. Since $\{C, F\} = S'$ is a sub-set of both $\cup_{u \in C'} C' = \{B, C, D\} \cup \{C, D, E\} \cup \{D, E, F\} \cup \{D, F, G, H, K\} = \{B, C, D, E, F, G, H, K\}$ and $\cup_{v \in C'} C' = \{C, D, E\} \cup \{D, E, F\} \cup \{E, I, J\} = \{C, D, E, F, I, J\}$, condition 3 of Theorem 3 applies.

Given the skeleton of a chain graph, we extend it into the equivalence class by identifying and directing all the complex arrows, to which the following result is helpful.

Proposition 4—Let $\mathcal{G}$ be a chain graph and $\mathcal{T}$ a separation tree of $\mathcal{G}$. For any complex κ in $\mathcal{G}$, there exists some tree node C of $\mathcal{T}$ such that $\text{par}(\kappa) \subseteq C$.

By Proposition 4, to identify the parents for each complex, we need only pay attention to those vertex pairs contained in each tree node.

3.2 Skeleton Recovery with Separation Tree

In this subsection, we propose an algorithm based on Theorem 3 for the identification of chain graph skeleton with separation tree information.

Let $\mathcal{G}$ be an unknown chain graph of interest and P be a probability distribution that is faithful to it. The algorithm, summarized as Algorithm 1, is written in its population version, where we assume perfect knowledge of all the conditional independencies induced by P.

Algorithm 1 consists of three main parts. In part 1 (lines 2–10), local skeletons are recovered in each individual node of the input separation tree. By condition 1 of Theorem 3, edges deleted in any local skeleton are also absent in the global skeleton. In part 2 (lines 11–16), we combine all the information obtained locally into a partially recovered global skeleton which may have extra edges not belonging to the true skeleton. Finally, we eliminate such extra edges in part 3 (lines 17–22).

The correctness of Algorithm 1 is proved in Appendix B. We conclude this subsection with some remarks on the algorithm.

Remarks on Algorithm 1

1. Although we assume perfect conditional independence knowledge in Algorithm 1, it remains valid when these conditional independence relations are obtained via performing hypotheses tests on data generated over P as in most real-world problems. When this is the case, we encounter the problem of multiple testing. See Sections 3.4 and 5 for more details.
2. The c-separator set $\mathcal{S}$ is not necessary for skeleton recovery. However, it will be useful later in Section 3.3 when we try to recover the pattern based on the learned skeleton.
3. Part 3 of Algorithm 1 is indispensable as is illustrated by the following example.

Example 1. (Continued): We apply Algorithm 1 to the chain graph $\mathcal{G}$ in Fig. 1(a) and the corresponding separation tree given in Fig. 2. The local skeletons recovered in part 1 of the algorithm are shown in Fig. 3(a). The c-separators found in this part are $S_{BC} = \{A\}$, $S_{CD} = \{B\}$, $S_{EJ} = \{I\}$, $S_{DK} = \{F, G\}$, $S_{DH} = \{F, G\}$, $S_{FG} = \{D\}$, $S_{FH} = \{G\}$ and $S_{HK} = \{G\}$. In part 2, the local skeletons are combined by deleting the edges that are absent in at least one of the local skeletons, leading to the result in Fig. 3(b). The edge $B – C$ is deleted since it is absent in the local skeleton for the tree node $\{A, B,C\}$. In part 3, we need to check $D – E$ and $D – F$. $D – E$ is deleted since $D \perp\!\!\!\perp E | \{C, F\}$, and we record $S_{DE} = \{C, F\}$. The recovered skeleton is finally shown in Fig. 3(c).

3.3 Complex Recovery

In this subsection, we propose Algorithm 2 for finding and orienting the complex arrows of $\mathcal{G}$ after obtaining the skeleton $\mathcal{G}'$ in Algorithm 1. We call this stage of structural learning *complex recovery*. As in the previous subsection, we assume separation tree information and perfect conditional independence knowledge. For simplicity, let S^c denote $V \backslash S$ for any vertex set $S \subseteq V$.

Example 1. (Continued)—After performing Algorithm 1 to recover the skeleton of $\mathcal{G}$, we apply Algorithm 2 to find and orient all the complex arrows. For example, when we pick $[C, D]$ in line 2 of Algorithm 2 and consider $C – E$ in line 3 for the inner loop, we find $S_{CD} = \{B\}$ and $C \perp\!\!\!\perp D | S_{CD} \cup \{E\}$. Therefore, we orient $C \rightarrow E$ in line 5. Similarly, we orient $D \rightarrow F$, $F \rightarrow K$ and $G \rightarrow K$ when considering $[D,C]$ with $D – F$ (in the inner loop), $[F, G]$ with $F – K$ and $[G, F]$ with $G – K$ and observing that $C \perp\!\!\!\perp D | S_{CD} \cup \{F\}$ and $F \perp\!\!\!\perp G | S_{FG} \cup \{K\}$, where the c-separators $S_{CD} = \{B\}$ and $S_{FG} = \{K\}$ were obtained during the execution of Algorithm 1. We do not orient any other edge in Algorithm 2. The resulting graph is shown in Fig. 4, which is exactly the pattern for our original chain graph $\mathcal{G}$ in Fig. 1(a).

The correctness of Algorithm 2 is guaranteed by Proposition 4. For the proof, see Appendix B.

Remarks on Algorithm 2

1. As Algorithm 1, Algorithm 2 is valid when independence test is correctly performed using data and again, multiple testing becomes an important issue here.
2. In Algorithm 2, the set $\mathcal{S}$ obtained in Algorithm 1 helps avoid the computationally expensive looping procedure taken in the pattern recovery algorithm in Studený (1997) for complex arrow orientation.
3. Line 9 of Algorithm 2 is necessary for chain graphs in general, see Example 2 below.

4. To get the pattern of $\mathcal{G}^*$ in line 9, at each step, we consider a pair of candidate complex arrows $u_1 \to w_1$ and $u_2 \to w_2$ with $u_1 \neq u_2$, then we check whether there is an undirected path from w_1 to w_2 such that none of its intermediate vertices is adjacent to either u_1 or u_2. If there exists such a path, then $u_1 \to w_1$ and $u_2 \to w_2$ are labeled (as complex arrows). We repeat this procedure until all possible candidate pairs are examined. The pattern is then obtained by removing directions of all unlabeled arrows in $\mathcal{G}^*$.

Example 2—Consider the chain graph $\mathcal{G}^\sim$ in Fig. 5(a) and the corresponding separation tree $\mathcal{T}^\sim$ in Fig. 5(e). After applying Algorithm 1, we obtain the skeleton $\mathcal{G}^{\sim\prime}$ of $\mathcal{G}^\sim$. In the execution of Algorithm 2, when we pick $[B, F]$ in line 2 and A in line 3, we have $B \perp\!\!\!\perp F | \varnothing$, that is, $S_{BF} = \varnothing$, and find that $B \not\perp\!\!\!\perp F | A$. Hence we orient $B - A$ as $B \to A$ in line 5, which is not a complex arrow in $\mathcal{G}^\sim$. Note that we do not orient $A - B$ as $A \to B$: the only chance we might do so is when $u = A$, $v = F$ and $w = B$ in the inner loop of Algorithm 2, but we have $B \in S_{AF}$ and the condition in line 4 is not satisfied. Hence, the graph we obtain before the last step in Algorithm 2 must be the one given in Fig. 5(c), which differs from the recovered pattern in Fig. 5(d). This illustrates the necessity of the last step in Algorithm 2. To see how the edge $B \to A$ is removed in the last step of Algorithm 2, we observe that, if we follow the procedure described in Remark 4 on Algorithm 2, the only chance that $B \to A$ becomes one of the candidate complex arrow pair is when it is considered together with $F \to D$. However, the only undirected path between A and D is simply $A - D$ with D adjacent to B. Hence $B \to A$ stays unlabeled and will finally get removed in the last step of Algorithm 2.

3.4 Sample Versions of Algorithms 1 and 2

In this section, we present a brief description on how to obtain a sample version of the previous algorithms and the related issues.

To apply the previous methods to a data set, we proceed in the exactly same way as before. The only difference in the sample version of the algorithms lies in that statistical hypothesis tests are needed to evaluate the predicate on line 5, 13 and 19 in Algorithm 1 and on line 4 in Algorithm 2. Specifically, conditional independence test of two variables u and v given a set C of variables is required. Let the null hypothesis H_0 be $u \perp\!\!\!\perp v | C$ and alternative H_1 be that H_0 may not hold. Generally we can use the likelihood ratio test statistic

$$G^2 = -2\log\frac{\sup\{L(\theta|D) \text{ under } H_0\}}{\sup\{L(\theta|D) \text{ under } H_1\}},$$

where $L(\theta|D)$ is the likelihood function of parameter θ with observed data D. Under H_0, the statistic G^2 asymptotically follows the χ^2 distribution with df degrees of freedom being equal to the difference of the dimensions of parameters for the alternative and null hypothesis (Wilks, 1938).

Let $\mathbf{X}_k$ be a vector of variables and N be the sample size. For the case of a Gaussian distribution, the test statistic for testing $X_i \perp\!\!\!\perp X_j | \mathbf{X}_k$ can be simplified to

$$\begin{aligned} G^2 &= -N \times \log(1 - \mathrm{corr}^2(X_i, X_j|\mathbf{X}_k)) \\ &= N \times \log\frac{\det(\widehat{\Sigma}_{\{i,k\}\{i,k\}})\det(\widehat{\Sigma}_{\{j,k\}\{j,k\}})}{\det(\widehat{\Sigma}_{\{i,j,k\}\{i,j,k\}})\det(\widehat{\Sigma}_{k,k})}, \end{aligned}$$

which has an asymptotic χ^2 distribution with $df = 1$. Actually, the exact null distribution or a better approximate distribution of G^2 can be obtained based on Bartlett decomposition, see Whittaker (1990) for details.

For the discrete case, let N_s^m be the observed frequency in a cell of $X_s = m$ where s is an index set of variables and m is category of variables X_s. For example, N_{ijk}^{abc} denotes the frequency of $X_i = a$, $X_j = b$ and $\mathbf{X}_k = c$. The G^2 statistic for testing $X_I \perp\!\!\!\perp X_j | \mathbf{X}_k$ is then given by

$$G^2 = 2 \sum_{a,b,c} N_{ijk}^{abc} \log \frac{N_{ijk}^{abc} N_k^c}{N_{ik}^{ac} N_{jk}^{bc}},$$

which is asymptotically distributed as a χ^2 distribution under H_0 with degree of freedom

$$\mathrm{df} = (\#(X_i) - 1)(\#(X_j) - 1) \prod_{X_l \in \mathbf{X}_k} \#(X_l),$$

where $\#(X)$ is the number of categories of variable X.

3.4.1 The Multiple Testing Problem—As mentioned in the remarks of Algorithms 1 and 2, when perfect knowledge of the population conditional independence structure is not available, we turn to hypotheses testing for obtaining these conditional independence relations from data and run into the problem of multiple testing. This is because we need to test the existence of separators for multiple edges, and we typically consider more than one candidate separators for each edge.

Although a theoretical analysis of the impact of the multiple testing problem on the overall error rate (see, for example, Drton and Perlman 2008) for the proposed method is beyond the scope of this paper, simulation studies are conducted on randomly generated chain graphs to show the impact of choices of different significance levels on our method in Section 5.1. Based on our empirical study there, we feel that choosing a small significance level (e.g., $\alpha = 0.005$ or 0.01) for the individual tests usually yields good and stable results when the sample size is reasonably large and the underlying graph is sparse.

3.5 Construction of Separation Trees

Algorithms 1 and 2 depend on the information encoded in the separation tree. In the subsection, we address the issue of how to construct a separation tree.

As proposed by Xie et al. (2006), one can construct a *d*-separation tree from observed data, from domain or prior knowledge of conditional independence relations or from a collection of databases. Their arguments are also valid in the current setting. In the rest of this subsection, we first extend Theorem 2 of Xie et al. (2006), which guarantees that their method for constructing a separation tree from data is valid for chain graph models. Then we propose an algorithm for constructing a separation tree from background knowledge encoded in a labeled block ordering (Roverato and La Rocca, 2006) of the underlying chain graph. We remark that Algorithm 2 of Xie et al. (2006), which constructs *d*-separation trees from hyper-graphs, also works in the current context.

3.5.1 From Undirected Independence Graph to Separation Tree—For a chain graph $\mathcal{G} = (V, E)$ and a faithful distribution P, an undirected graph $\mathcal{U}$ with a vertex set V is an undirected independence graph (UIG) of $\mathcal{G}$ if for any u, v ∈ V,

$$(u, v) \text{ is not an edge of } \mathcal{U} \Rightarrow u \perp\!\!\!\perp v | V \backslash \{u, v\} \text{ in } P. \tag{2}$$

We generalize Theorem 2 of Xie et al. (2006) as follows.

Theorem 5: Let $\mathcal{G}$ be a chain graph. Then a junction tree constructed from any undirected independence graph of $\mathcal{G}$ is a separation tree for $\mathcal{G}$.

Since there are standard algorithms for constructing junction trees from UIGs (see Cowell et al. 1999, Chapter 4, Section 4), the construction of separation trees reduces to the construction of UIGs. In this sense, Theorem 5 enables us to exploit various techniques for learning UIGs to serve our purpose.

As suggested by relation (2), one way of learning UIGs from data is testing the required conditional independencies. Agresti (2002) and Anderson (2003) provides general techniques for discrete and Gaussian data respectively. Recently, there are also works on estimating UIGs via ℓ_1-regularization, see, for instance, Friedman et al. (2007) for Gaussian data and Ravikumar et al. (2008) for the discrete case. Edwards (2000, Chapter 6) presents some other established methods for UIG learning, including those that are valid when X_V includes both continuous and discrete variables. All these methods can be used to construct separation trees from data.

3.5.2 From Labeled Block Ordering to Separation Tree—When learning graphical models, it is a common practice to incorporate substantive background knowledge in structural learning, mostly to reduce the dimensionality of the search spaces for both computational and subject matter reasons.

Recently, Roverato and La Rocca (2006) studied in detail a general type of background knowledge for chain graphs, which they called *labeled block ordering*. We introduce this concept here and investigate how it is related to the construction of separation trees.

By summarizing Definitions 5 and 6 in Roverato and La Rocca (2006), we define the labeled block ordering as the following.

Definition 6: Let $V_1, \cdots, V_k$ be a partition of a set of vertices V. A labeled block ordering $\mathcal{B}$ of V is a sequence ($V_i^{l_i}$, i = 1, ⋯, k) where $l_i \in \{u, d, g\}$ and with the convention that $V_i = V_i^g$. We say a chain graph $\mathcal{G} = (V, E)$ is $\mathcal{B}$-consistent if

1. every edge connecting vertices $A \in V_i$ and $B \in V_j$ for $i \neq j$ is oriented from $A \rightarrow B$ if $i < j$;
2. for every $l_i = u$, the subgraph $\mathcal{G}_{V_i}$ is a UG;
3. for every $l_i = d$, the subgraph $\mathcal{G}_{V_i}$ is a DAG;
4. for every $l_i = g$, the subgraph $\mathcal{G}_{V_i}$ may have both directed and undirected edges.

Example 1. (Continued): Let $V_1 = \{A, B, C, D, E, F\}$, $V_2 = \{G, H, K\}$ and $V_3 = \{I, J\}$. Then Fig. 6 shows that for a labeled block ordering $\mathcal{B} = (V_1^g, V_2^g, V_3^d)$, $\mathcal{G}$ in Fig. 1(a) is $\mathcal{B}$-consistent.

To show how labeled block ordering helps learning separation trees, we start with the following simple example.

Example 3: Suppose that the underlying chain graph is the one in Fig. 7(a) which is $\mathcal{B}$-consistent with $\mathcal{B}=\{V_1^g, V_2^g, V_3^g\}$. Then for $V_1 = \{A, B\}$, $V_2 = \{C, D\}$ and $V_3 = \{E, F\}$, we have $V_1 \perp\!\!\!\perp V_3 | V_2$. Together with the total ordering on them, we can construct the DAG in Fig. 7(b) with vertices as blocks V_i, $i = 1, 2, 3$, which depicts the conditional independence structures on the three blocks. By taking the junction tree of this DAG and replacing the blocks by the vertices they contain, we obtain the tree structure in Fig. 7(c). This is a separation tree of the chain graph in Fig. 7(a).

Below we propose Algorithm 3 for constructing a separation tree from labeled block ordering knowledge. The idea is motivated by the preceding toy example. By omitting independence structures within each block, we can treat each block as a vertex in a DAG. The total ordering on the blocks and the conditional independence structures among them enable us to build a DAG on these blocks. By taking the junction tree of this particular DAG, we obtain a separation tree of the underlying chain graph model.

We note that a labeled block ordering gives a total ordering of the blocks V_i, $i = 1, \cdots, k$. Given a total ordering of the vertices of a DAG, one can use the Wermuth-Lauritzen algorithm (Spirtes et al., 2000; Wermuth and Lauritzen, 1983) for constructing (the equivalence class of) the DAG from conditional independence relations. Spirtes et al. (2000) also discussed how to incorporate the total ordering information in the PC algorithm for constructing DAGs from conditional independence relations. Since these approaches rely only on conditional independence relations, they can be used in Algorithm 3 for constructing the DAG $\mathcal{D}$. The only difference is that each vertex represents a random vector rather than a random variable now. The correctness of Algorithm 3 is proved in Appendix B.

3.5.3 Separation Tree Refinement—In practice, the nodes of a separation tree constructed from a labeled block ordering or other methods may still be large. Since the complexities of our skeleton and complex recovery algorithms are largely dominated by the cardinality of the largest node on the separation tree, it is desirable to further refine our separation tree by reducing the sizes of the nodes. To this end, we propose the following algorithm.

If the current separation tree contains a node whose cardinality is still relatively large, the above algorithm can be repeatedly used until no further refinement is available. However, we remark that the cardinality of the largest node on the separation tree is eventually determined by the sparseness of the underlying chain graph together with other factors including the specific algorithms for constructing undirected independence subgraphs and junction tree. The correctness of the algorithm is proved in Appendix B.

4. Computational Complexity Analysis

In this section, we investigate the computational complexities of Algorithms 1 and 2 and compare them with the pattern recovery algorithm proposed by Studený (1997). We divide our analysis into two stages: (a) skeleton recovery stage (Algorithm 1); (b) complex recovery stage (Algorithm 2). In each stage, we start with a discussion on Studený's algorithm and then give a comprehensive analysis of ours.

4.1 Skeleton Recovery Stage

Let $\mathcal{G} = (V, E)$ be the unknown chain graph, p the number of vertices and e the number of edges, including both lines and arrows. In Studený's algorithm, to delete an edge between a vertex pair u and v, we need to check the independence of u and v conditional on all possible subsets

S of $V\backslash\{u, v\}$. Therefore, the complexity for investigating each possible edge in the skeleton is $O(2^p)$ and hence the complexity for constructing the global skeleton is $O(p^2 2^p)$.

For Algorithm 1, suppose that the input separation tree has H nodes $\{C_1, \cdots, C_H\}$ where $H \leq p$ and $m = \max\{\text{card}(C_h), 1 \leq h \leq H\}$. The complexity for investigating all edges within each tree node is thus $O(m^2 2^m)$. Thus, the complexity for the first two parts of Algorithm 1 is $O(Hm^2 2^m)$. For the analysis of the third step, suppose that $k = \max\{\text{card}(S), S \in \mathcal{T}\}$ is the cardinality of the largest separator on the separation tree. Since the tree with H nodes has $H - 1$ edges, we also have $H - 1$ separators. Thus, the edge to be investigated in step 3 is $O(Hk^2)$. Then, let d be the maximum of the degrees of vertices in the partially recovered skeleton obtained after step 2. By our algorithm, the complexity for checking each edge is $O(2^d)$. Hence, the total complexity for step 3 is $O(Hk^2 2^d)$. Combining all the three parts, the total complexity for our Algorithm 1 is $O(H(m^2 2^m + k^2 2^d))$. It is usually the case that m, k and d are much smaller than p, and therefore, our Algorithm 1 is computationally less expensive than Studený's algorithm, especially when $\mathcal{G}$ is sparse.

4.2 Complex Recovery Stage

For complex arrow orientation, Studený's algorithm needs to first find a candidate complex structure of some pre-specified length l and then check a collection of conditional independence relations. Finding a candidate structure can be accomplished in a polynomial (of p) time. The complexity of the algorithm is determined by the subsequent investigation on conditional independence relations. Actually, the number of conditional independence relations to be checked for each candidate structure is $2^{p-2} - 1$. Hence, the complexity of Studený's algorithm is exponential in p.

In Algorithm 2, the computational complexity of the large double loop (lines 2–8) is determined by the number of conditional independence relations we check in line 4. After identifying u, w and v, we need only to check one single conditional independence with the conditioning set $S_{uv} \cup \{w\}$, which can be done in $O(1)$ time. The pair $\{u, w\}$ must be connected on $\mathcal{G}'$, thus the number of such pairs is $O(e)$. Since the number of v for each $\{u, w\}$ pair is at most p, we execute line 4 at most $O(pe)$ times. The complexity for the double loop part is thus controlled by $O(pe)$. Next we show that the complexity for taking the pattern of the graph is $O(e^2(p + e))$. As in the remarks on Algorithm 2, in each step, we propose a pair of candidate complex arrows $u_1 \rightarrow w_1$ and $u_2 \rightarrow w_2$ and then check whether there is an undirected path from w_1 to w_2 whose intermediate vertices are adjacent to neither u_1 nor u_2. This can be done by performing a graph traversal algorithm on an undirected graph which is obtained by deleting all existing arrows on the current graph with the adjacency relations checked on the original graph. By a breadth-first search algorithm, the complexity is $O(p + e)$ (Cormen et al., 2001). Since the number of candidate complex arrow pairs is controlled by $O(e^2)$, the total complexity for finding the pattern is $O(e^2(p + e))$. Since $pe = O(e^2(p + e))$, the total complexity for Algorithm 2 is $O(e^2(p + e))$.

By incorporating the information obtained in the skeleton recovery stage, we greatly reduce the number of conditional independence tests to be checked and hence obtain an algorithm of only polynomial time complexity for the complex recovery stage. The improvement is achieved with additional cost: we need to store the c-separator information S_{uv}'s obtained from Algorithm 1 in the set $\mathcal{S}$. The possible number of $\{u, v\}$ combinations is $O(p^2)$, while the length of S_{uv} is $O(p)$. Hence, the total space we need to store $\mathcal{S}$ is $O(p^3)$, which is still polynomially complex.

4.3 Comparison with a DAG Specific Algorithm When the Underlying Graph is a DAG

In this subsection, we compare our algorithm with Algorithm 1 in Xie et al. (2006) that is designed specifically for DAG structural learning when the underlying graph structure is a DAG. We make this choice of the DAG specific algorithm so that both algorithms can have the same separation tree as input and hence are directly comparable.

Combining the analyses in the above two subsections, we know that the total complexity of our general algorithm is $O(H(m^2 2^m + k^2 2^d) + e^2(p + e))$, while the complexity of the DAG specific algorithm is $O(Hm^2 2^m)$ as claimed in Xie et al. (2006, Section 6). So the extra complexity in the worst case is $O(Hk^2 2^d + e^2(p + e))$. The term that might make a difference is $O(Hk^2 2^d)$, which occurs as the complexity of step 3 in our Algorithm 1 and involves an exponential term in d. Note that d is defined as the maximum degree of the vertices in the partially recovered skeleton $\mathcal{G}'$ obtained after step 2 of Algorithm 1, where $\mathcal{G}'$ is exactly the skeleton for the underlying DAG in the current situation (i.e., step 3 of Algorithm 1 does not make any further modification to $\mathcal{G}'$ when $\mathcal{G}$ is a DAG). Hence, if the underlying graph is sparse, d is small and the extra complexity $O(Hk^2 2^d + e^2(p + e))$ is well under control.

Therefore, if we believe that the true graph is sparse, the case where our decomposition approach is most applicable, we can apply our general chain graph structural learning algorithm without worrying much about significant extra cost even when the underlying graph is indeed a DAG.

5. Simulation

In this section, we investigate the performance of our algorithms under a variety of circumstances using simulated data sets. We first demonstrate various aspects of our algorithms by running them on randomly generated chain graph models. We then compare our methods with DAG-specific learning algorithms on data generated from the ALARM network, a Bayesian network that has been widely used in evaluating the performance of structural learning algorithms. The simulation results show the competitiveness of our algorithms, especially when the underlying graph is sparse. From now on, we refer to our method as the LCD (Learn Chain graph via Decomposition) method. Algorithms 1 and 2 have been implemented in the R language. All the results reported here are based on the R implementation.

5.1 Performance on Random Chain Graphs

To assess the quality of a learning method, we adopt the way Kalisch and Bühlmann (2007) used in investigating the performance of PC algorithm on Bayesian networks. We perform our algorithms on randomly generated chain graphs and report summary error measures.

5.1.1 Data Generation Procedure—First we discuss the way in which the random chain graphs and random samples are generated. Given a vertex set V, let $p = |V|$ and N denote the average degree of edges (including undirected and pointing out and pointing in) for each vertex. 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 $\{1, \cdots, p\}$ as the number of chain components;
5. Split the interval $[1, p]$ into k equal-length subintervals $I_1, \cdots, I_k$ 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 ($A_{i\,j} = A_{ji} = 1$) 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 $\mathbb{E}$[vertex degree] = N where an adjacent vertex can be linked by either an undirected or a directed edge.

Given a randomly generated chain graph $\mathcal{G}$ with ordered chain components $C_1, \cdots, C_k$, we generate a Gaussian distribution on it via the incomplete block-recursive regression as described in Wermuth (1992). Let X_m be the $|C_m| \times 1$ random vector, and $X=(X_k^T, \cdots, X_1^T)^T$. Then we have the block-recursive regression system as

$$B^*X=W^*,$$

where

$$B^*=\begin{pmatrix} \sum^{k,k} & \sum^{k,k-1} & \cdots & \sum^{k,1} \\ 0 & \sum^{k-1,k-1\cdot k} & \cdots & \sum^{k-1,1\cdot k} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \sum^{1,1.23\cdots k} \end{pmatrix}.$$

Let

$$T=\begin{pmatrix} \sum^{k,k} & & 0 \\ & \ddots & \\ 0 & & \sum^{1,1.23\cdots k} \end{pmatrix}$$

be the block-diagonal part of B^*. Each block diagonal element $\Sigma^{i,i.i+1\cdots k}$ of B^* is the inverse covariance matrix of X_i conditioning on $(X_{i+1}, \cdots, X_k)$, an element of which is set to be zero if the corresponding edge within the chain component is missing. The upper triangular part of B^* has all the conditional covariances between X_i and $(X_1, \cdots, X_{i-1})$. The zero constraints on the elements correspond to missing directed edges among different components. Finally, $W^* \sim N(0, T)$.

For the chain component C_i, suppose the corresponding vertices are $V_{i_1}, \cdots, V_{i_r}$, and in general, let $B^*[V_l,V_m]$ be the element of B^* that corresponds to the vertex pair (V_l,V_m). In our simulation, we generate the B^* matrix in the following way:

1. For the diagonal block $\Sigma^{i,i.i+1\cdots k}$ of B^*, for $1 \le j < j' \le r$, we fix $B^*[V_{i_j}, V_{i_j}] = 1$ and set $B^*[V_{i_j}, V_{i_{j'}}] = 0$ if the vertices V_{i_j} and $V_{i_{j'}}$ are non-adjacent in C_i and otherwise sampled randomly from $(-1.5/r, -0.5/r) \cup (0.5/r, 1.5/r)$, and finally we symmetrize the matrix according to its upper triangular part.
2. For $\Sigma^{i,\,j,i+1\cdots k}$, $1 \le j \le i-1$, an element $B^*[V_l,V_m]$ is set to be zero if $V_l \in C_i$ is not pointed to by an arrow starting from $V_m \in C_1 \cup \cdots \cup C_{i-1}$, and sampled randomly from $-1.5/r, -0.5/r) \cup (0.5/r, 1.5/r)$ otherwise.
3. If any of the block diagonal elements in B^* is not positive semi-definite, we repeat Step (1) and (2).

After setting up the B^* matrix, we take its block diagonal to obtain the T matrix. For fixed B^* and T, we first draw i.i.d. samples of W^* from $N(0, T)$, and then pre-multiply them by $(B^*)^{-1}$ to get random samples of X. We remark that faithfulness is not necessarily guaranteed by the current sampling procedure and quantifying the deviation from the faithfulness assumption is beyond the scope of this paper.

5.1.2 Performance under Different Settings—We examine the performance of our algorithm in terms of three error measures: (a) the true positive rate (TPR) and (b) the false positive rate (FPR) for the skeleton and (c) the structural Hamming distance (SHD) for the pattern. In short, TPR is the ratio of # (correctly identified edge) over total number of edges, FPR is the ratio of # (incorrectly identified edge) over total number of gaps and SHD is the number of legitimate operations needed to change the current pattern to the true one, where legitimate operations are: (a) add or delete an edge and (b) insert, delete or reverse an edge orientation. In principle, a large TPR, a small FPR and a small SHD indicate good performance.

In our simulation, we change three parameters p (the number of vertices), n (sample size) and N (expected number of adjacent vertices) as follows:

- $p \in \{10, 40, 80\}$,
- $n \in \{100, 300, 1000, 3000, 10000, 30000\}$,
- $N \in \{2, 5\}$

For each (p, N) combination, we first generate 25 random chain graphs. We then generate a random Gaussian distribution based on each graph and draw an identically independently distributed (i.i.d.) sample of size n from this distribution for each possible n. For each sample, three different significance levels ($\alpha = 0.005$, 0.01 or 0.05) are used to perform the hypothesis tests. We then compare the results to access the influence of the significance testing level on the performance of our algorithms. A separation tree is obtained through the following 'one step elimination' procedure:

1. We start from a complete UIG over all p vertices;
2. We test zero partial correlation for each element of the sample concentration matrix at the chosen significance level α and delete an edge if the corresponding test doesn't reject the null hypothesis;
3. An UIG is obtained after Step 2 and its junction tree is computed and used as the separation tree in the algorithms.

The plots of the error measures are given in Fig. 8, 9 and 10. From the plots, we see that: (a) our algorithms yield better results on sparse graphs ($N = 2$) than on dense graphs ($N = 5$); (b) the TPR increases with sample size while the SHD decreases; (c) the behavior of FPR is largely regulated by the significance level α used in the individual tests and has no clear dependence on the sample size (Note that FPRs and their variations in the middle columns of Fig. 8, 9 and 10 are very small since the vertical axes have very small scales); (d) large significance level $\alpha(=0.05)$ typically yields large TPR, FPR and SHD while the advantage in terms of a larger TPR (compared to $\alpha = 0.005$ or 0.01) fades out as the sample size increases and the disadvantage in terms of a larger SHD becomes much worse; (e) accuracy in terms of TPR and SHD based on $\alpha = 0.005$ or $\alpha = 0.01$ is very close while choosing $\alpha = 0.005$ does yield a consistently (albeit slightly) lower FPR across all the settings in the current simulation. Such empirical evidence suggests that in order to account for the multiple testing problem, we can choose a small value (say $\alpha = 0.005$ or 0.01 for the current example) for the significance level of individual tests. However, the optimal value for a desired overall error rate may depend on the sample size and the sparsity of the underlying graph.

Finally, we look at how our method scales with the sample size, which is not analyzed explicitly in Section 4. The average running times vs. the sample sizes are plotted in Fig. 11. It can be seen that: (a) the average run time scales approximately linearly with log(sample size); and (b) the scaling constant depends on the sparsity of the graph. The simulations were run on an Intel Core Duo 1.83GHz CPU.

5.2 Learning the ALARM Network

As we have pointed out in Section 1, Bayesian networks are special cases of chain graphs. It is of interest to see whether our general algorithms still work well when the data are actually generated from a Bayesian network. For this purpose, in this subsection, we perform simulation studies on both Gaussian and discrete case for the ALARM network in Fig. 12 and compare our algorithms for general chain graphs with those specifically designed for Bayesian networks. The network was first proposed in Beinlich et al. (1989) as a medical diagnostic network.

5.2.1 The Gaussian Case—In the Gaussian case, for each run of the simulation, we repeat the following steps:

1. A Gaussian distribution on the network is generated using a recursive linear regression model, whose coefficients are random samples from the uniform distribution on $(-1.5, -0.5) \cup (0.5, 1.5)$ and residuals are random samples from $N(0, 1)$.
2. A sample of size n is generated from the distribution obtained at step 1.
3. We run the LCD algorithms, the DAG learning algorithms proposed in Xie et al. (2006) and the PC algorithm implemented in the R package `pcalg` (Kalisch and Bühlmann, 2007) all with several different choices of the significance level α. The one step elimination procedure described in Section 5.1.2 was used to construct the separation trees for the LCD and DAG methods.
4. We record the number of extra edges (FP), the number of missing edges (FN) and the structural Hamming distance (SHD) compared with the true pattern for all the three learned patterns.

We performed 100 runs for each sample size $n \in \{1000, 2000, 5000, 10000\}$. For each sample, we allow three different significance levels $\alpha \in \{0.05, 0.01, 0.005\}$ for all the three methods. Table 1 documents the averages and standard errors (in parentheses) from the 100 runs for each method-parameter-sample size combination.

As shown in Table 1, compared with the DAG method (Xie et al., 2006), the LCD method consistently yields a smaller number of false positives and the differences in false negatives are consistently smaller than two on recovering the skeleton of the network. The SHDs obtained from our algorithms are usually comparable to those from the DAG method when $\alpha = 0.05$ and the difference is usually less than five when $\alpha = 0.01$ or 0.005. Moreover, we remark that as sample size grows, the power of the significance test increases, which leads to better performance of the LCD method as in the case of other hypothesis testing based methods. However, from Table 1, we find that the LCD performance increases more rapidly in terms of SHD. One plausible reason is that in Algorithm 2, we identify complex arrows by rejecting conditional independence hypotheses rather than direct manipulation as in the algorithms specific for DAG and hence have some extra benefit in terms of accuracy as the power of the test becomes greater.

Finally, the LCD method consistently outperforms the PC algorithm in all three error measures. Such simulation results confirm that our method is reliable when we do not know the information that the underlying graph is a DAG, which is usually untestable from data.

5.2.2 The Discrete Case—In this section, a similar simulation study with discrete data sampled from the ALARM network is performed. The variables in the network are allowed to have two to four levels. For each run of the simulation, we repeat the following steps:

1. For each variable X_i and fixed configuration pa_i of its parents, we define the conditional probability $P(X_i=j|pa_i)=r_j/\sum_{k=1}^{L} r_k$, where L is the number of levels of Xi and $\{r_1, \cdots, r_L\}$ are random numbers from the Uniform(0,1) distribution.

2. A sample of size n is generated from the above distribution.

3. We run three algorithms designed specifically for DAG that have been shown to have a good performance: MMHC (Max-Min Hill Climbing: Tsamardinos et al., 2006), REC (Recursive: Xie and Geng, 2008) and SC (Sparse Candidate: Friedman et al., 1999) with several different choice of parameters. We also perform the LCD learning algorithm with different choices of the significance level. For the LCD method, a grow-shrink Markov blanket selection is performed on the data to learn the UIG and the junction tree of the UIG is supplied as the separation tree for the algorithm.

4. For each algorithm, we recorded the FP, FN and SHD of the recovered pattern under each choice of the learning parameter.

We performed 100 runs for each of the four different sample sizes $n \in \{1000, 2000, 5000, 10000\}$, and in each run, we allow the following choices of the learning parameters:

- For MMHC, REC and LCD, we allow three different significance levels $\alpha \in \{0.05, 0.01, 0.005\}$; and
- For SC, we allow the learning parameters to be either 5 or 10.

The means and standard errors from the 100 runs of all the four learning methods are summarized in Table 2. It can be seen that REC could yield the best result when the learning parameter is appropriately chosen. However, all the other methods are more robust against the choice of learning parameters. For the LCD method, all the three error measures: FP, FN and SHD are consistently comparable to those of methods specifically designed for DAG. Moreover, as in the Gaussian case, the power of the tests used in the LCD method grows as the sample size become larger, which makes the LCD method even more competitive, especially in terms of SHD. For example, when the sample size reaches 10000, the LCD method with $\alpha = 0.01$ or 0.005 outperforms the sparse candidate method with parameters 5 or 10.

6. Discussion

In this paper, we presented a computationally feasible method for structural learning of chain graph models. The method can be used to facilitate the investigation of both response-explanatory and symmetric association relations among a set of variables simultaneously within the framework of chain graph models, a merit not shared by either Bayesian networks or Markov networks.

Simulation studies illustrate that our method yields good results in a variety of situations, especially when the underlying graph is sparse. On the other hand, the results also reveal that the power of the significance test has an important influence on the performance of our method. With fixed number of samples, one can expect a better accuracy if we replace the asymptotic test used in our implementation with an exact test. However, there is a trade-off between accuracy and computational time.

The results in this paper also raised a number of interesting questions for future research. We briefly comment on some of those questions here. First, the separation tree plays a key role in Algorithms 1 and ****2. Although the construction of separation trees has been discussed in Xie et al. (2006) and Section 3.5 here, we believe that there is room for further improvements. Second, we have applied hypothesis testing for the detection of local separators in Algorithm 1 and also in complex arrow determination in Algorithm 2. It shall be interesting to see whether there exists some alternative approach, preferably not based on hypothesis testing, to serve the same purpose here. A theoretical analysis of the effect of multiple testing on the overall error rate of the procedure is also important. In addition, it is a common practice to incorporate prior information about the order of the variables in graphical modelling. Therefore, incorporation of such information into our algorithms is worth investigation. Finally, our approach might be extendible to the structural learning of chain graph of alternative Markov properties, for example, AMP chain graphs (Andersson et al., 2001) and multiple regression chain graphs (Cox and Wermuth, 1996).

An R language package `lcd` that implements our algorithms is available on the first author's website: www.stanford.edu/~zongming/software.html.

Acknowledgments

The authors would like to thank two referees for their valuable suggestions and comments which improve the presentation of the previous version of the paper. We would also like to thank Professor John Chambers and Xiangrui Meng for their help on the simulation study. This research was supported by NSFC (10771007, 10431010, 10721403), 863 Project of China (2007AA01Z437), MSRA and MOE-Microsoft Key Laboratory of Statistics and Information Technology of Peking University. The first author was also supported in part by grants NSF DMS 0505303 and NIH EB R01 EB001988.

References

Agresti, A. Categorical Data Analysis. 2. John Wiley & Sons; Hoboken, NJ: 2002.

Anderson, TW. An Introduction to Multivariate Statistical Analysis. 3. John Wiley & Sons; Hoboken, NJ: 2003.

Andersson SA, Madigan D, Perlman MD. Alternative Markov properties fror chain graphs. Scand J Statist 2001;28:33–85.

Beinlich, I.; Suermondt, H.; Chevaz, R.; Cooper, G. The alarm monitoring system: A case study with two probabilistic inference techniques for belief networks. Proceedings of the 2nd European Conference on Artificail Intelligence in Medicine; Berlin: Springer-Verlag; 1989. p. 247-256.

Carroll S, Pavlovic V. Protein calassification using probabilistic chain graphs and the gene ontology structure. Bioinformatics 2006;22(15):1871–1878. [PubMed: 16705013]

Chickering DM. Learning equivalence classes of bayesian-network structures. J Mach Learn Res 2002;2:445–498.

Cowell, RG.; Dawid, AP.; Lauritzen, SL.; Spiegelhalter, DJ. Probabilistic Networks and Expert Systems. Springer-Verlag; New York: 1999.

Cox, DR.; Wermuth, N. Multivariate Dependencies: Models, Analysis and Interpretation. Chapman and Hall; London: 1996.

Drton M, Perlman M. A sinful approach to gaussian graphical model selection. J Stat Plan Infer 2008;138:1179–1200.

Edwards, D. Introduction to Graphical Modelling. 2. Springer-Verlag; New York: 2000.

Ellis, B.; Wong, WH. Learning bayesian network structures from experimental data. 2006. URL http://www.stanford.edu/group/wonglab/doc/EllisWong-061025.pdf

Friedman J, Hastie T, Tibshirani R. Sparse inverse covariance estimation with the graphical lasso. Biostatistics. 200710.1093/biostatistics/kxm045

Friedman N, Koller D. Being bayesian about network structure: a bayesian approach to structure discovery in bayesian networks. Mach Learn 2003;50:95–126.

Friedman, N.; Nachmana, I.; Pe'er, D. Learning bayesian network structure from massive datasets: The "sparse candidate" algorithm. Proceedings of the Fifteenth Conference on Uncertainty in Artificail Intelligence; Stockholm, Sweden: 1999. p. 206-215.

Frydenberg M. The chain graph markov property. Scand J Statist 1990;17:333–353.

Kalisch M, Bühlmann P. Estimating high-dimensional directed acyclic graphs with the pc-algorithm. J Mach Learn Res 2007;8:616–636.

Lauritzen, SL. Graphical Models. Claredon Press; Oxford: 1996.

Lauritzen SL, Richardson TS. Chain graph models and their causal interpretations (with discussion). J R Statist Soc B 2002;64:321–361.

Liu, Y.; Xing, EP.; Carbonell, J. Predicting protein folds with structural repeats using a chain graph model. Proceedings of the 22nd International Conference on Machine Learning; 2005.

Meinshausen N, Bühlmann P. High-dimensional graphs and variable selection with the lasso. Ann Statist 2006;34:1436–1462.

Pearl, J. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann; San Francisco, CA: 1988.

Ravikumar, P.; Wainwright, MJ.; Lafferty, J. Technical Report 750. Department of Statistics, University of California; Berkeley: 2008. High-dimensional graphical model selection using ℓ_1-regularized logistic regression. URL http://www.stat.berkeley.edu/tech-reports/750.pdf

Roverato A, La Rocca L. On block ordering of variables in graphical modelling. Scand J Statist 2006;33:65–81.

Spirtes, P.; Glymour, C.; Scheines, R. Causation, Prediction and Search. 2. MIT Press; Cambridge, MA: 2000.

Stanghellini E, McConway KJ, Hand DJ. A discrete variable chain graph for applicants for credit. J R Statist Soc C 1999;48:239–251.

Studený M. A recovery algorithm for chain graphs. Int J Approx Reasoning 1997;17:265–293.

Studený M, Bouckaert RR. On chain graph models for description of conditional independence structures. Ann Statist 2001;26:1434–1495.

Tsamardinos I, Brown LE, Aliferis CF. The max-min hill-climbing bayesian network structure learning algorithm. Mach Learn 2006;65:31–78.

Wermuth N. On block-recursive linear regression equations. Revista Brasileira de Probabilidade e Estatistica 1992;6:1–56.

Wermuth N, Lauritzen SL. Graphical and recursive models for contingency tables. Biometrika 1983;72:537–552.

Wermuth N, Lauritzen SL. On substantive research hypotheses, conditional independence graphs and graphical chain models. J R Statist Soc B 1990;52:21–50.

Whittaker, J. Graphical Models in Applied Multivariate Statistics. John Wiley & Sons; 1990.

Wilks S. The large-sample distribution of the likelihood ratio for testing composite hypotheses. Ann Math Stat 1938;20:595–601.

Xie X, Geng Z. A recursive method for structural learning of directed acyclic graphs. J Mach Learn Res 2008;9:459–483.

Xie X, Geng Z, Zhao Q. Decomposition of structural learning about directed acyclic graphs. Artif Intell 2006;170:442–439.

Appendix A. Proofs of Theoretical Results

In this part, we give proofs to theorems and propositions. We first give a definition and several lemmas that are to be used in later proofs.

Definition 7

Let $\mathcal{T}$ be a separation tree for a CG $\mathcal{G}$ with the node set $\mathcal{C} = \{C_1, \cdots, C_H\}$. For any two vertices u and v in $\mathcal{G}$, the distance between u and v in the tree $\mathcal{T}$ is defined by

$$d(u, v) = \min_{C_i \ni u, C_j \ni v} d(C_i, C_j),$$

where $d(C_i, C_j)$ is the distance between nodes C_i and C_j in $\mathcal{T}$. We call C_i and C_j minimizers for u and v if they minimize the distance $d(C_i, C_j)$.

Lemma 8

Let ρ be a route from u to v in a chain graph $\mathcal{G}$, and W the set of all vertices on ρ (W may or may not contain the two end vertices). Suppose that ρ is intervented by $S \subset V$. If $W \subset S$, ρ is also intervented by W and any vertex set containing W.

Proof

Since ρ is intervented by S and $W \subset S$, there must be a non head-to-head section σ of ρ that is hit by S and actually every non head-to-head section of ρ is hit by S. Thus, σ is also hit by W and any vertex set containing W. Hence, ρ is intervented.

Lemma 9

Let $\mathcal{T}$ be a separation tree for a chain graph over vertex set V and K a separator of $\mathcal{T}$ which separates $\mathcal{T}$ into two subtrees $\mathcal{T}_1$ and $\mathcal{T}_2$ with variable sets V_1 and V_2. Suppose that $u \in V_1 \backslash K$, $v \in V_2 \backslash K$ and ρ is a route from u to v in $\mathcal{G}$. Let W denote the set of all vertices on ρ (W may or may not contain the two end vertices). Then ρ is intervented by $W \cap K$ and by any vertex set containing $W \cap K$.

Proof

Since $u \in V_1 \backslash K$ and $v \in V_2 \backslash K$, there must be a sequence from s (may be u) to y (may be v) in $\rho = (u, \cdots, s, t, \cdots, x, y, \cdots, v)$ such that $s \in V_1 \backslash K$, $y \in V_2 \backslash K$ and all vertices from t to x in this sequence are contained in K. Otherwise, every vertex of ρ is either in $V_1 \backslash K$ or in $V_2 \backslash K$. This implies that there exists $w \in V_1 \backslash K$ and $z \in V_2 \backslash K$ on ρ that are adjacent, which is contradictory to the fact that $\langle V_1 \backslash K, V_2 \backslash K | K \rangle_{\mathcal{G}}^{sep}$. Without loss of generality, we can suppose that y is the first vertex (from the left) of ρ that is not in V_1.

Let ρ′ be the sub-route of the sequence $(s, t, \cdots, x, y)$, and W' be the vertex set of ρ′ excluding s and y. Since $W' \subset K$, we know from Lemma 8 that there is at least one non head-to-head section (w.r.t. ρ′) on ρ′ and every non head-to-head section of ρ′ is hit by W'. We are to show that there is at least one non head-to-head section of ρ that is hit by K and hence $W \cap K$ as well as any set containing $W \cap K$.

The only problem arises when ρ′ is part of a head-to-head section of ρ. Otherwise, there is some non head-to-head section of ρ′ that is (part of) a non head-to-head section of ρ.

Thus, we suppose that the head-to-head section of ρ is

$$s' \rightarrow s'' - \cdots - s - t - \cdots - x - y - \cdots - y'' \leftarrow y'.$$

By our assumption on y, we know that $s' \in V_1$. If $s' \in K$, then the non head-to-head section containing s' is hit by K. If $s' \in V_1 \backslash K$ and $y' \in K$, then the non head-to-head section containing

y' gives the result. If $s' \in V_1 \backslash K$ and $y' \in V_1 \backslash K$, then we can consider the sub-route starting from y'. This is legitimate since every non head-to-head section of that sub-route is also non head-to-head w.r.t. ρ. Hence, we need only consider the case that $s' \in V_1 \backslash K$ and $y' \in V_2 \backslash K$. In this case, let t' be the last (from left) vertex in this section that is adjacent to s' and x' the first vertex after t' in this section that is adjacent to y'. Since chain graphs cannot have directed pseudocycles, we know that $s' \to t'$ and $y' \to x'$. Then we have $s' \not\perp\!\!\!\perp y' | K$, which is contradictory to the property of separation trees that $\langle V_1 \backslash K, V_2 \backslash K | K \rangle_{\mathcal{G}}^{sep}$. This completes our proof.

Lemma 10

Let u and v be two non adjacent vertices in a chain graph $\mathcal{G}$and ρ a route from u to v in $\mathcal{T}$. If ρ is not contained in An(u) ∪ An(v), then ρ is intervented by any subset S of An(u) ∪ An(v).

Proof

Since ρ is not contained in An(u) ∪ An(v), there exist four vertices s,t, x and y, such that ρ = (u, ⋯, s,t, ⋯, x, y, ⋯, v), with {s, y}⊂ An(u) ∪An(v) and {t, ⋯, x}∩[An(u) ∪An(v)] = ∅. Then we have $s \to t$ and $x \leftarrow y$, since otherwise t and/or x must be in An(u) ∪An(v). Thus, there exists at least one head-to-head section between s and y on ρ such that it is not hit by any subset of An(u) ∪ An(v). Hence, ρ is intervented by any subset S of An(u) ∪ An(v).

Lemma 11

Let $\mathcal{T}$ be a separation tree for a chain graph $\mathcal{G}$over V and C a node of $\mathcal{T}$. Let u and v be two vertices in C which are non adjacent in $\mathcal{G}$. If u and v are not contained simultaneously in any separator connected to C, then there exists a subset S of C which c-separates u and v in $\mathcal{G}$.

Proof

Define

$$S = [\,\mathrm{An}(u) \cup \mathrm{An}(v)] \cap [\,C \backslash \{u, v\}].$$

We show below that $\langle u, v | S \rangle_{\mathcal{G}}^{sep}$.

To this end, let ρ be any fixed route from u to v in $\mathcal{G}$. If ρ is not contained in An(u) ∪ An(v), by Lemma 10, ρ is intervented by S. Otherwise, we divide the problem into the following six possible situations:

1. u –⋯– $u' \leftarrow x$, $x \neq v$, $x \in C$, where u –⋯– u' means the first (from left) section of ρ that contains u;
2. u –⋯– $u' \to x$ –⋯– $x' \to y$, $x \neq v$, $x \in C$;
3. u –⋯– $u' \to x$ –⋯– x' y, $x \neq v$, $x \in C$, $y \in C$;
4. u –⋯– $u' \to x$ –⋯– $x' \leftarrow y$, $x \neq v$, $x \in C$, $y \notin C$;
5. $u - u'$ ⋯– $v' - v$, $u - u'$ –⋯– $v' \to v$ or $u - u'$ –⋯– $v' \leftarrow v$;
6. u –⋯– $u' \to x$ or u –⋯– $u' \leftarrow x$, $x \notin C$.

We prove the desired result situation by situation.

For situation 1, we have that $x \in \mathrm{An}(u)$, which, together with $x \in C$ implies that $x \in S$. The non head-to-head section containing x is hit by S, and ρ is thus intervented.

For situation 2, since $x \in \mathrm{An}(u) \cup \mathrm{An}(v)$ and $x \notin \mathrm{An}(u)$, we have $x \in \mathrm{An}(v)$. Together with $x \in C$, this gives $x \in S$ and the non head-to-head section containing x is hit by S.

For situation 3, since chain graphs do not admit directed pseudocycles, we know that $x \notin \mathrm{An}(u)$ and $y \neq v$. Similar to situation 2, we have $x \in \mathrm{An}(v)$ and hence $y \in \mathrm{An}(v)$. The non head-to-head section containing y is hit by S.

For situation 4, suppose that C' is one of the nodes on $\mathcal{T}$ that contains y. Consider first the case when v belongs to the separator K connected to C and the next node on the path from C to C' on $\mathcal{T}$. By our assumption, $u \notin K$. We divide the problem into the following three cases:

- **i.** $\{u, \cdots, u'\} \cap S \neq \emptyset$: $u - \cdots - u'$ is hit by S and ρ is hence intervented;
- **ii.** $\{u, \cdots, u'\} \cap S = \emptyset$, $\{x, \cdots, x'\} \cap S = \emptyset$: the head-to-head section $x - \cdots - x'$ is not hit by S and ρ is intervented;
- **iii.** $\{u, \cdots, u'\} \cap S = \emptyset$, $\{x, \cdots, x'\} \cap S \neq \emptyset$: there must exist some $x^* \in \{x, \cdots, x'\}$ such that $x^* \in C \cap \mathrm{An}(v)$ and $x^* \neq = v$. Since $\{u, \cdots, u'\} \cap S = \emptyset$ and $u \not\perp\!\!\!\perp y | K$, there should be no complex in the induced subgraph of $(u, \cdots, u', x, \cdots, x', y)$. Otherwise, there exists some $u^* \in \{u, \cdots, u'\}$, such that (u^*, y) is an edge on $(\mathcal{G}_{\mathrm{An}(u,y,K)})^m$, which implies $u \not\perp\!\!\!\perp y | K$ since $\{u, \cdots, u'\} \cap K = \emptyset$. However, this requires that there is some $u^{**} \in \{u, \cdots, u'\}$ such that (u^{**}, y) is an edge on $\mathcal{G}$, which again implies that $u \not\perp\!\!\!\perp y | K$. Hence, this case can never happen.

Next, we consider the case that $v \notin K$. The assumption that $\{x, \cdots, x'\} \subset \mathrm{An}(v)$ implies that there exists at least one $' \rightarrow '$ on the sub-route l' of ρ from y to v. Consider the rightmost one of such arrows, there is no further $' \leftarrow '$ closer to the right end v than it is. Otherwise, any vertex w between them satisfies $w \in \mathrm{An}(u)$ and $w \in \mathrm{de}(v)$, which is contradictory to the fact that $v \in \mathrm{de}(u)$ here. Thus, this case reduces to one of the situations 1, 5 and 6 with u replaced by v.

For situation 5, since u and v are non adjacent, we know that $v' \neq u$ and $u' \neq v$. If $\{u', \cdots, v'\} \cap S = \emptyset$, then $\{u', \cdots, v'\} \cap C \subset \{u, v\}$. We can eliminate vertices from the left such that $\{u', \cdots, v'\} \cap C \subset \{v\}$. This will not influence our result since any non head-to-head section of the sub-route is (part of) a non head-to-head section of ρ. Since $u' \neq v$ and $\{u', \cdots, v'\} \cap C \subset \{v\}$, we have $u' \notin C$. Suppose that $u' \in C'$ and K is the separator related to C and the next node on the path from C to C' on $\mathcal{T}$. Then $u \in K$, $v \notin K$ and $u' \not\perp\!\!\!\perp v | K$. However, since $\{u', \cdots, v'\} \cap C \subset \{v\}$, we have $\{u, \cdots, u'\} \cap K = \emptyset$, which is impossible. Hence $\{u', \cdots, v'\} \cap S \neq \emptyset$ and ρ is intervented.

For situation 6, consider first the case that $u - \cdots - u' \leftarrow x$. If $\{u, \cdots, u'\} \cap S \neq \emptyset$, then ρ is intervented by S. Otherwise, suppose that $x \in C'$ and K is the separator connected to C and the next node on the path from C to C' in $\mathcal{T}$. If $v \in K$, then $u \notin K$ and $\{u, \cdots, u'\} \cap K \subset \{v\}$. If $\{u, \cdots, u'\} \cap K = \{v\}$, then it reduces to situation 5. If $\{u, \cdots, u'\} \cap K = \emptyset$, then $u \not\perp\!\!\!\perp x | K$, which is contradictory to the definition of separation tree. If $v \notin K$, then by the above argument, we must have $u \in K$. Consider the sub-route of ρ starting with x. It is legitimate to do so since any non head-to-head section of the sub-route is also non head-to-head in ρ. By Lemma 9, at least one non head-to-head section is hit by $W \cap K$ where W is the vertex set of the sub-route excluding the two end vertices. We know that $W \cap K \subset S \cup \{u, v\}$. If the non head-to-head section is hit at u or v, we can consider the further sub-route starting at that point and it is again legitimate by the same reason. Finally, we can reduce to the case where $W \cap K \subset S$. Thus, ρ is intervented by S. This also completes the proof of situation 4. For the other case in this situation, all the argument is the same up to the point where $v \notin K$ and $u \in K$. We can consider

reversing the vertex sequence, then with u replaced by v, it must be in one of the situations 1 to 4, the second case in situation 6 or the first case in situation 6 with $u \notin K$. This complete the proof of the lemma.

Proof of Theorem 3

The sufficiency of condition 1 is given by Lemma 9. The sufficiencies of conditions 2 and 3 are trivial by the definition of c-separation.

Now we show the necessity part of the theorem. If $d(u, v) > 0$, by Lemma 9, any separator K on the path from minimizers C_i to C_j c-separates u and v. If $d(u, v) = 0$, we consider the following two possible cases: (1) u and v are not contained simultaneously in any separator connected to C for some node C on $\mathcal{T}$ containing both u and v; (2) otherwise. For the first case, Lemma 11 shows that there exists some $S' \subset C$ that c-separates u and v. Otherwise, since $\langle u, v|\mathrm{bd}_{\mathcal{G}}(u) \cup \mathrm{bd}_{\mathcal{G}}(v)\rangle_{\mathcal{G}}^{sep}$, $\mathrm{bd}_{\mathcal{G}}(u) \subset \cup_{u \in C} C$ and $\mathrm{bd}_{\mathcal{G}}(v) \subset \cup_{v \in C} C$, we know that at least one of the conditions 2 and 3 holds.

Proof of Proposition 4

We verify Proposition 4 by contradiction. Let us suppose that u and v are parents of a complex $\kappa = (u, w_1, \cdots, w_k, v)$, $k \leq 1$ in $\mathcal{T}$ and that for any node C on $\mathcal{T}$, $\{u, v\} \cap C \neq \{u, v\}$. Now suppose that $u \in C_1$, $v \in C_2$ and K is the separator related to C_1 and the next node on the path from C_1 to C_2 on $\mathcal{T}$. If $u \notin K$ and $v \notin K$, we must have that $\{w_1, \cdots, w_k\} \cap K \neq \varnothing$. This implies that $u \not\perp\!\!\!\perp v|K$, which is contrary to the definition of separation trees. Hence, without loss of generality, we may suppose that $u \in K$, and this enables us to go one node closer to C_2 on the path. Then after finite steps, we will consider two adjacent nodes on $\mathcal{T}$. Repeating the above argument ensures that $\{u, v\}$ belongs to one of these two nodes.

Appendix B. Proofs for Correctness of the Algorithms

Before proving the correctness of the algorithms, we need several more lemmas.

Lemma 12

Suppose that u and v are two adjacent vertices in $\mathcal{G}$, then for any separation tree $\mathcal{T}$ for $\mathcal{G}$, there exists a node C in $\mathcal{T}$ which contains both u and v.

Proof

If not, then there exists a separator K on $\mathcal{T}$, such that $u \in V_1 \backslash K$ and $v \in V_2 \backslash K$ where V_i denotes the variable set of the subtree $\mathcal{T}_i$ induced by removing the edge attached by the separator S, for $i = 1$ and 2. This implies $u \not\perp\!\!\!\perp v|K$, which is impossible.

Lemma 13

Any arrow oriented in line 5 of Algorithm 2 is correct in the sense that it is an arrow with the same orientation in $\mathcal{G}$.

Proof

We prove the lemma by induction. If we don't orient any arrow in line 5, then the lemma holds trivially. Otherwise, suppose $u \rightarrow w$ is the first arrow we orient by considering the ordered triple $\langle u, v, w\rangle$, then we show that it cannot be $u - w$ or $u \leftarrow w$ in $\mathcal{G}$. If it is $u - w$ in $\mathcal{G}$, then by Lemma 11, if u and v are not in any separator simultaneously, there exists some $S_{uv} \subset C_h$ such

that $u \perp\!\!\!\perp v | S_{uv}$ and $w \in S_{uv}$. Otherwise, $u \perp\!\!\!\perp v | \mathrm{bd}_{\mathcal{G}}(u) \cup \mathrm{bd}_{\mathcal{G}}(v)$, and we know that $w \in \mathrm{bd}_{\mathcal{G}}(u) \cup \mathrm{bd}_{\mathcal{G}}(v) \subset \mathrm{bd}_{\mathcal{G}'}(u) \cup \mathrm{bd}_{\mathcal{G}'}(v)$. Thus we won't orient it as $u \to w$. A similar argument holds for the case when $u \leftarrow w$ in $\mathcal{G}$.

Now suppose that the k-th arrow we orient is correct, let's consider the $k + 1$-th. Suppose it's $u' \to w'$ by considering the order triple $\langle u', v', w' \rangle$. Then the above argument holds exactly with u, v and w substituted by u', v' and w'. However, for here, the claim that $\mathrm{bd}_{\mathcal{G}}(u') \cup \mathrm{bd}_{\mathcal{G}}(v') \subset \mathrm{bd}_{\mathcal{G}'}(u') \cup \mathrm{bd}_{\mathcal{G}'}(v')$ holds by the induction assumption.

Lemma 14

Suppose that $\mathcal{H}$ is a graph, if we disorient any non-complex arrow in $\mathcal{H}$, the pattern of $\mathcal{H}$ does not change.

Proof

First, we note that we will not add or delete edge in $\mathcal{H}$, so the skeleton of $\mathcal{H}$ does not change.

Second, we only disorient non-complex arrows, and hence those complexes in $\mathcal{H}$ before disorientation remain complexes after disorientation since the subgraph induced by any complex does not change.

Finally, we show that there will not be new complex. If there appears a new complex, say $u \to w_1 - \cdots, -w_l \leftarrow v$, we must have $l \geq 2$. Since it was not a complex before disorientation, one of the lines in $w_1 - \cdots - w_l$ must be the arrow disoriented. Suppose we have $w_i \to w_{i+1}$ before disorientation, then $w_i \to w_{i+1} - \cdots - w_l \leftarrow v$ was a complex before disorientation. Hence, the disoriented arrow $w_i \to w_{i+1}$ was a complex arrow, which contradicts our assumption.

Correctness of Algorithm 1

On the one hand, by Studený (1997, Lemma 3.2), we know that for any chain graph $\mathcal{G}$, there is an edge between two vertices u and v if and only if $u \not\perp\!\!\!\perp v | S$ for any subset S of V. Thus, line 6 of Algorithm only deletes those edges that cannot appear in the true skeleton. So do lines 14 and 19.

On the other hand, if u and v are not adjacent in $\mathcal{G}$, by Theorem 3, it must be under one of the three possible conditions. If it is in condition 1, we will never connect them since we do not connect any vertex pair that is never contained in any node simultaneously. If it is in condition 2, then we will disconnect them in line 6 and line 14. Finally, if it is in condition 3, then either $u \perp\!\!\!\perp v | \mathrm{bd}_{\mathcal{G}}(u)$ or $u \perp\!\!\!\perp v | \mathrm{bd}_{\mathcal{G}}(v)$. By our previous discussion, we know that before starting line 17, we have $\mathrm{bd}_{\mathcal{G}}(u) \subset \mathrm{ne}_{\mathcal{G}'}(u)$ and $\mathrm{bd}_{\mathcal{G}}(v) \subset \mathrm{ne}_{\mathcal{G}'}(v)$ for the $\mathcal{G}'$ at that moment. Thus, we will disconnect u and v in line 19.

Correctness of Algorithm 2

By Proposition 4, Lemma 12 and the correctness of Algorithm 1, we know that every ordered vertex triple $\langle u, v, w \rangle$ in $\mathcal{G}$with $u \to w$ a complex arrow and v the parent of (one of) the corresponding complex(es) is considered in line 4 of Algorithm 2. If the triple $\langle u, v, w \rangle$ is really in this situation, then we know that $u \not\perp\!\!\!\perp v | S_{uv} \cup \{w\}$, and hence we orient $u - w$ as $u \to w$. Moreover, Lemma 13 prevents us from orienting $u - w$ as $w \to u$ during the execution of Algorithm 2. This proves that we will orient every complex arrow right before starting line 9 in Algorithm 2.

Then by Lemma 13, the $\mathcal{G}^*$before starting line 9 is a hybrid graph with the same pattern as $\mathcal{G}$. Thus Lemma 14 guarantees that we obtain the pattern of $\mathcal{G}$after line 9.

Correctness of Algorithm 3

First of all, we show that any conditional independence relation represented by $\mathcal{D}$ is also represented by $\mathcal{G}$. This is straightforward by noting the following two facts:

1. assuming positivity, both DAG models and chain graph models are closed subset of graphoids under 5 axioms, see Pearl (1988) and Studený and Bouckaert (2001);
2. any conditional independence relation used in constructing $\mathcal{D}$ is represented by $\mathcal{G}$.

Then by Xie et al. (2006, Theorem 2), the $\mathcal{T}$ in line 2 is a separation tree of $\mathcal{D}$. With the block vertices substituted, by the definition of separation trees, the output $\mathcal{T}$ of Algorithm 3 is a separation tree of $\mathcal{G}$.

Correctness of Algorithm 4

Xie et al. (2006, Theorem 3) guarantees the correctness of Algorithm 4.

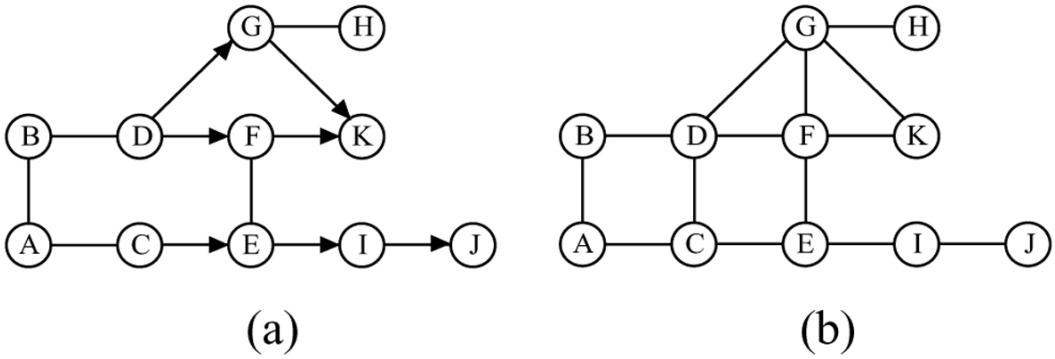

Figure 1.
(a) a chain graph $\mathcal{G}$; (b) its moral graph $\mathcal{G}^m$.

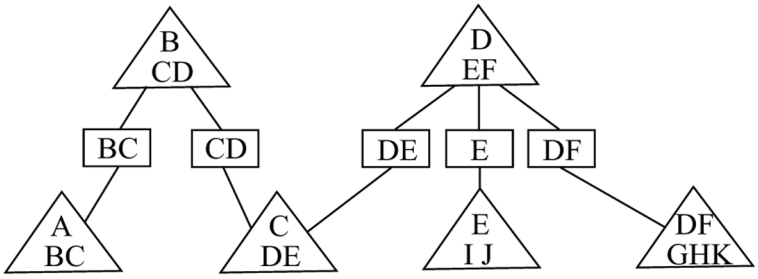

Figure 2.
A separation $\mathcal{T}$ of the graph $\mathcal{G}$ in Fig. 1(a).

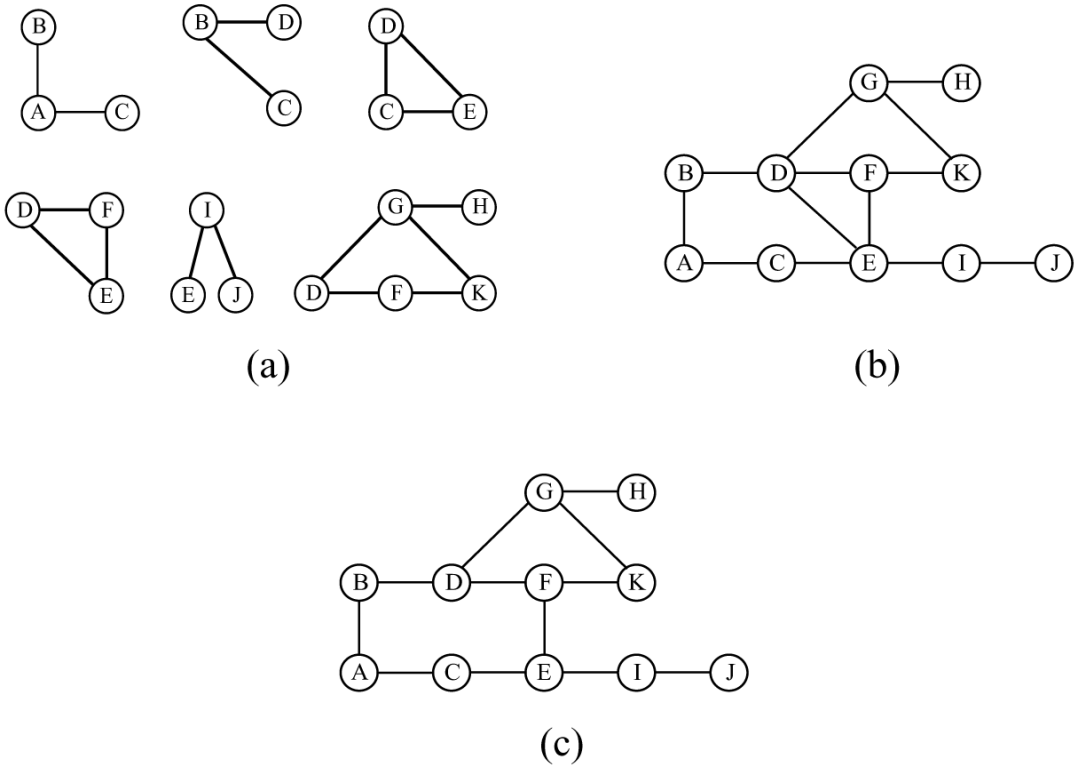

Figure 3.
(a) local skeletons recovered in part 1 of Algorithm 1 for all nodes of $\mathcal{T}$ in Fig. 2; (b) partially recovered global skeleton of $\mathcal{G}$ in part 2 of Algorithm 1; (c) completely recovered global skeleton of $\mathcal{G}$ in part 3 of Algorithm 1.

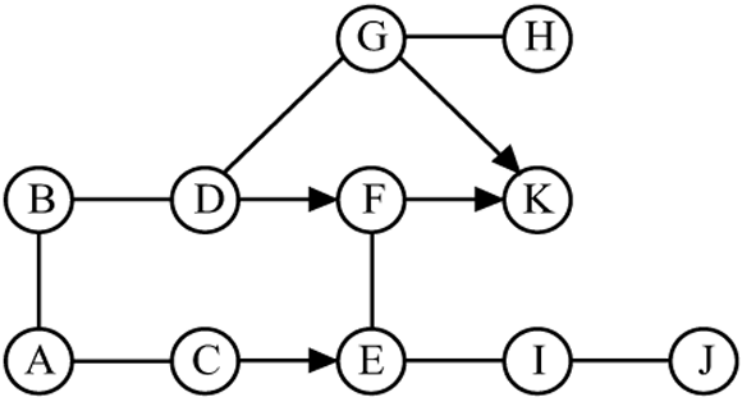

Figure 4.
The pattern of $\mathcal{G}$ recovered by applying Algorithm 2.

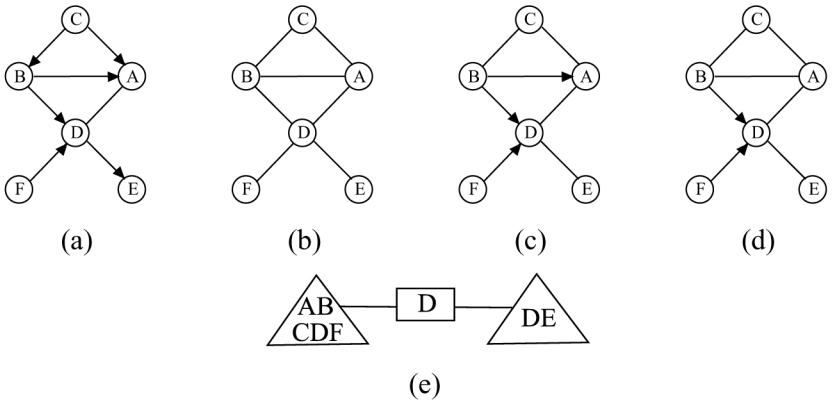

Figure 5.
(a) the chain graph $\mathcal{G}^{\sim}$ in Example 2; (b) the skeleton $\mathcal{G}^{\sim\prime}$ of $\mathcal{G}^{\sim}$; (c) the graphical structure $\mathcal{G}^{\sim *}$ before executing the last line in Algorithm 2; (d) the graphical structure $\mathcal{G}^{\sim *}$ obtained after executing Algorithm 2; (e) a separation tree $\mathcal{T}^{\sim}$ for $\mathcal{G}^{\sim}$ in (a).

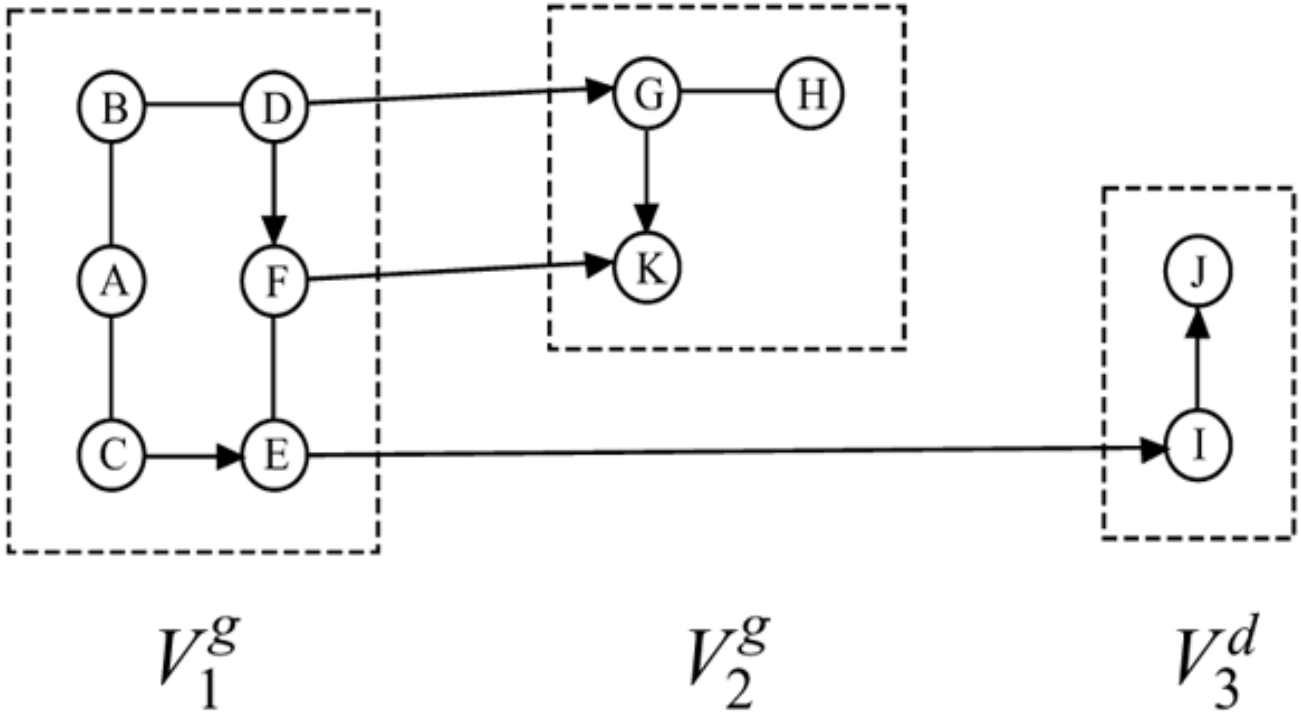

Figure 6.

A labeled block ordering $\mathcal{B}=(V_1^g, V_2^g, V_3^d)$ for which $\mathcal{G}$ in Fig. 1(a) is $\mathcal{B}$-consistent.

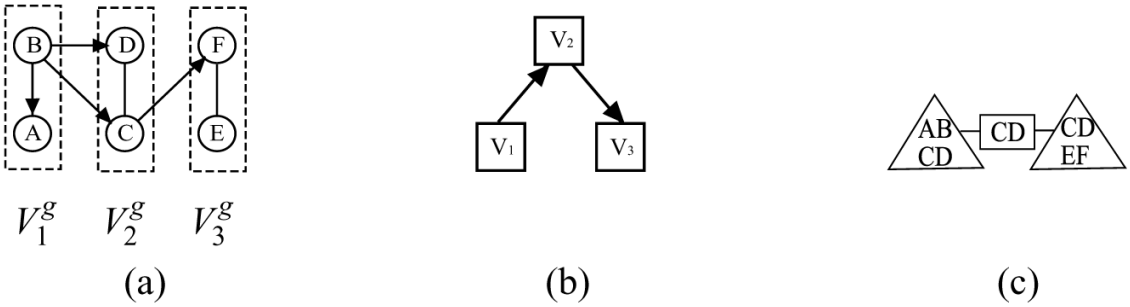

Figure 7.
(a) a chain graph with a labeled block ordering; (b) the DAG constructed for the blocks; (c) the tree structure obtained from the junction tree of the DAG in (b).

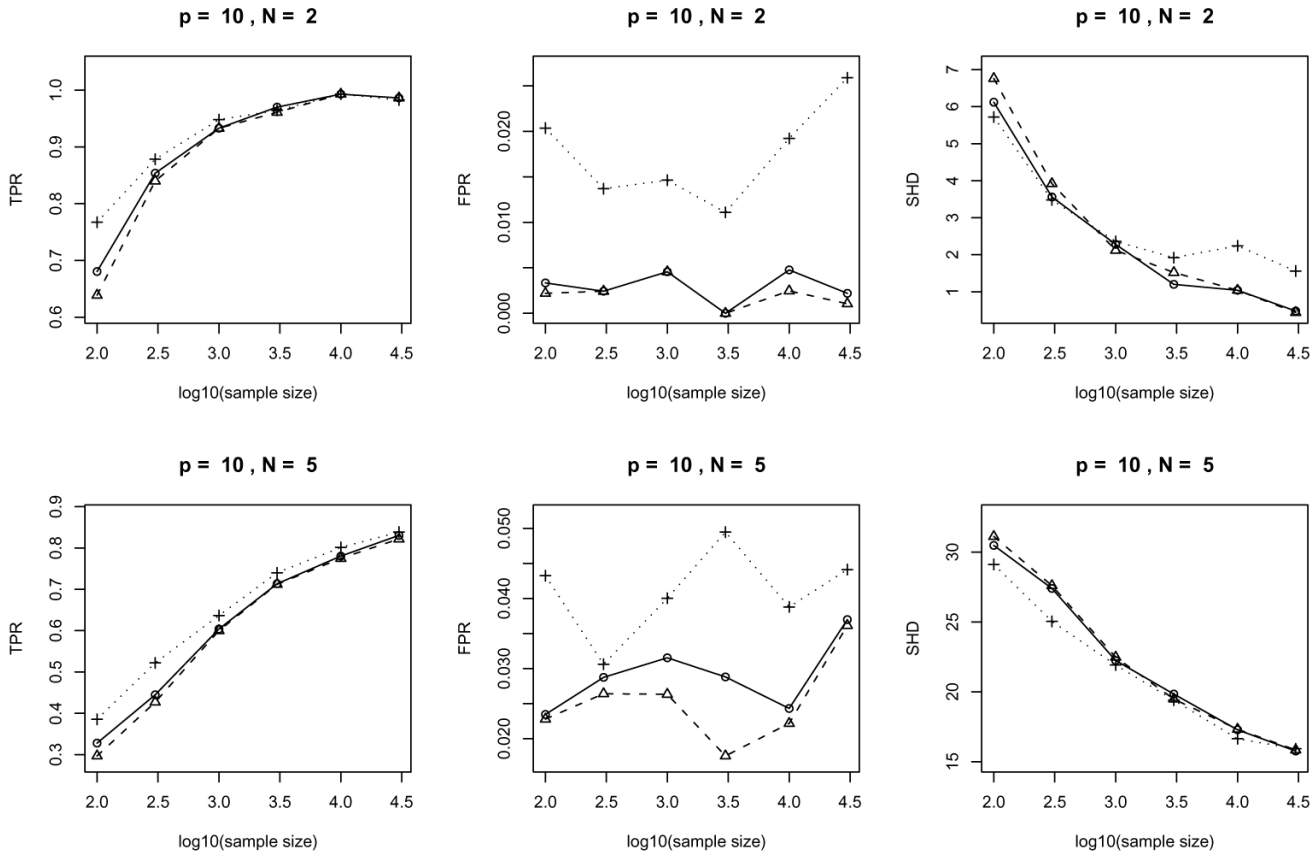

Figure 8.
Error measures of the algorithms for randomly generated Gaussian chain graph models: average over 25 repetitions with 10 variables. The two rows correspond to $N = 2$ and $N = 5$ cases and the three columns give three error measures: TPR, FPR and SHD in each setting respectively. In each plot, the solid/dashed/dotted lines correspond to significance levels α= 0.01/0.005/0.05.

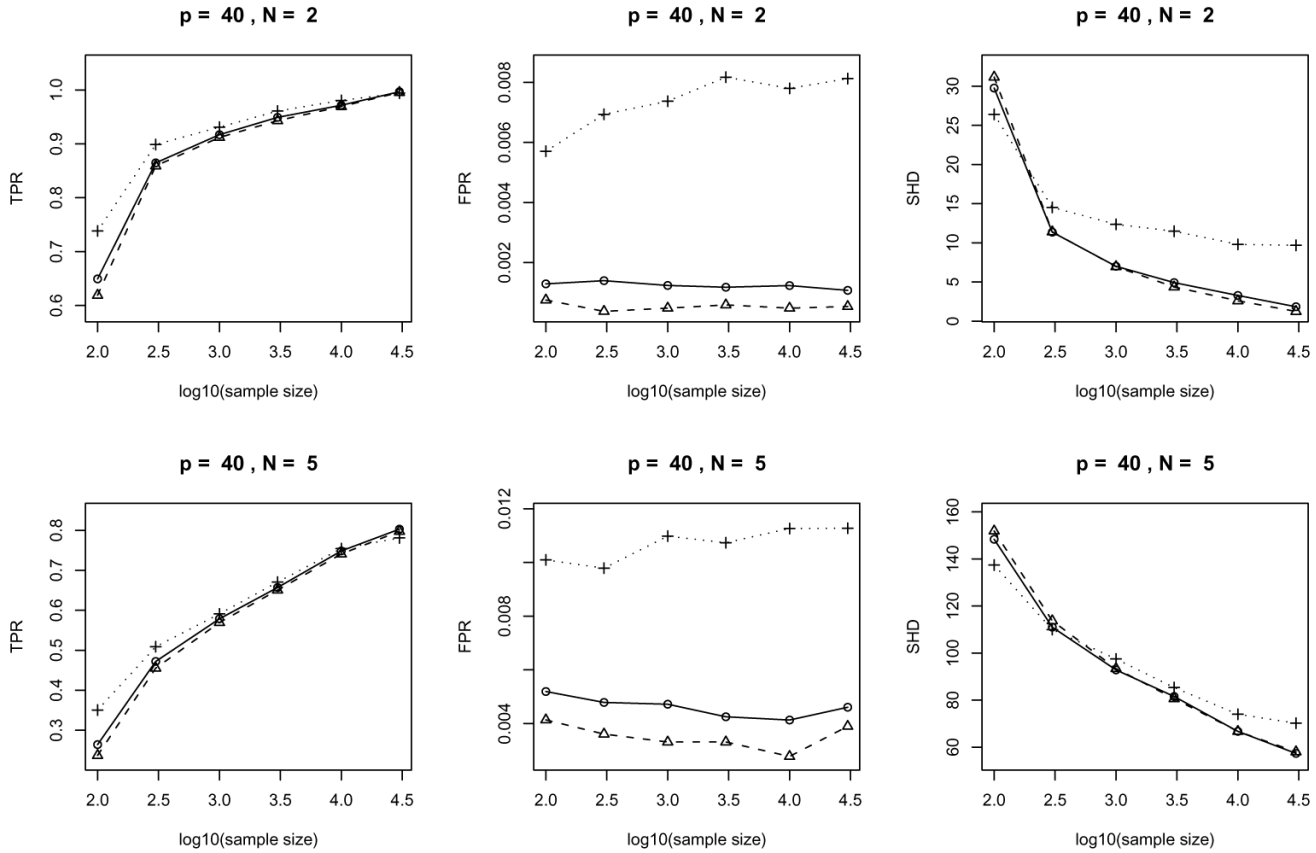

Figure 9.
Error measures of the algorithms for randomly generated Gaussian chain graph models: average over 25 repetitions with 40 variables. Display setup is the same as in Fig. 8.

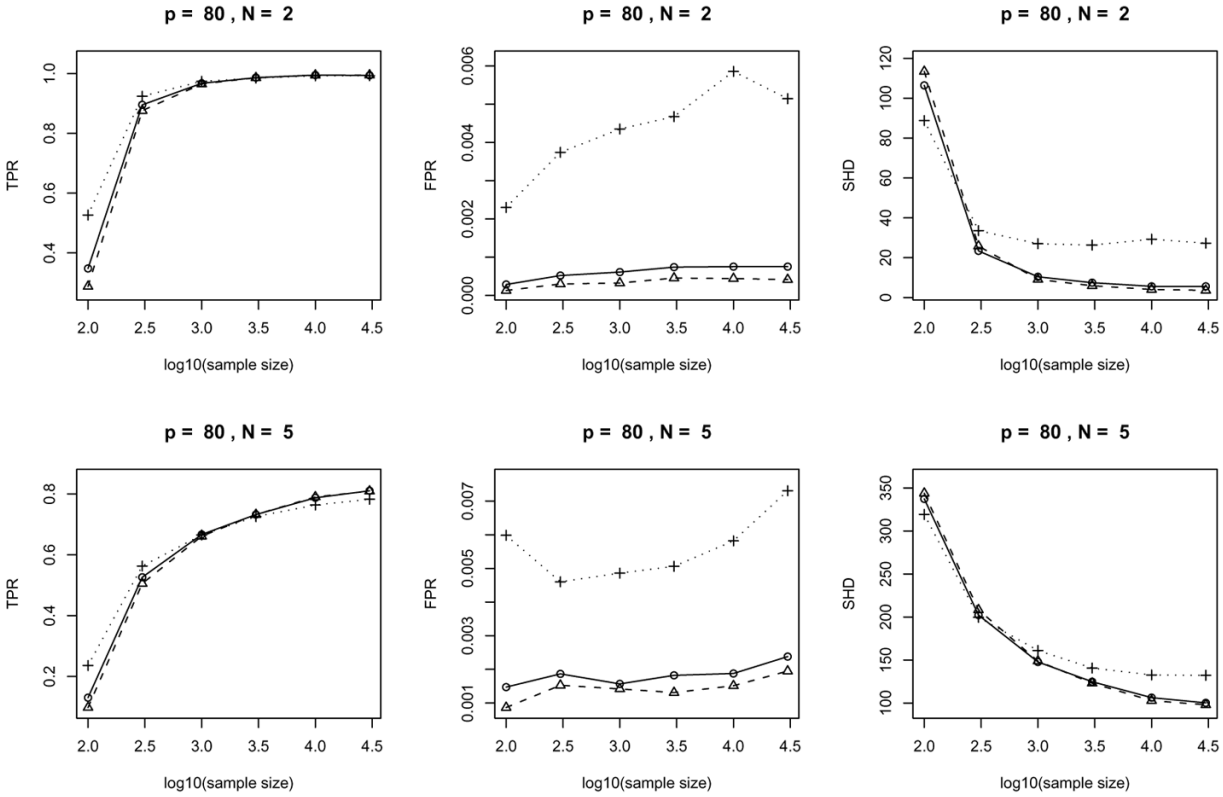

Figure 10.
Error measures of the algorithms for randomly generated Gaussian chain graph models: average over 25 repetitions with 80 variables. Display setup is the same as in Fig. 8.

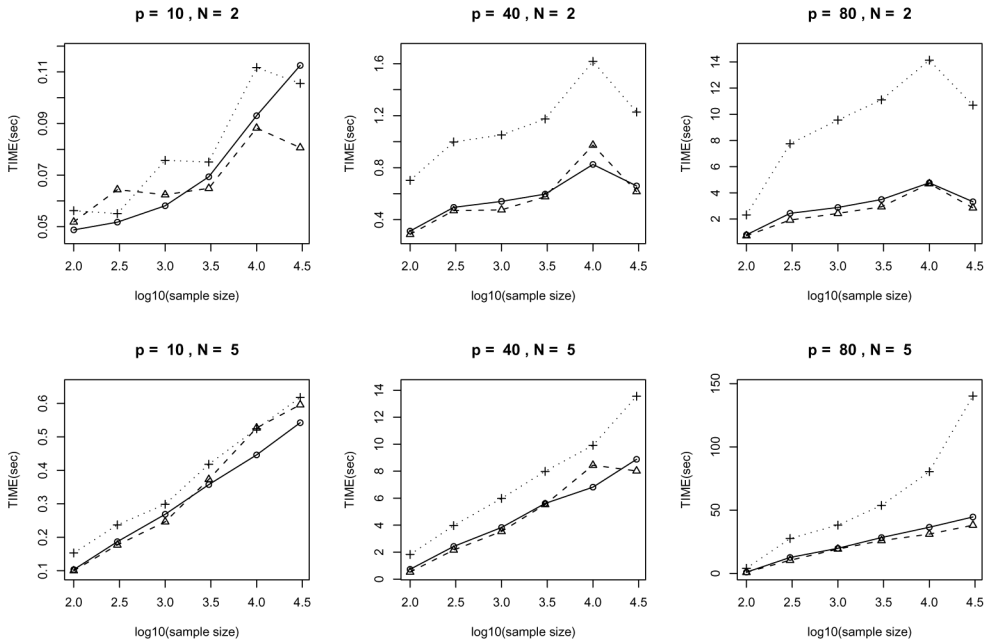

Figure 11.
Running times of the algorithms on randomly generated Gaussian chain graph models: average over 25 repetitions. The two rows correspond to $N = 2$ and 5 cases and the three columns represent $p = 10$, 40 and 80 respectively. In each plot, the solid/dashed/dotted lines correspond to significance levels $\alpha = 0.01/0.005/0.05$.

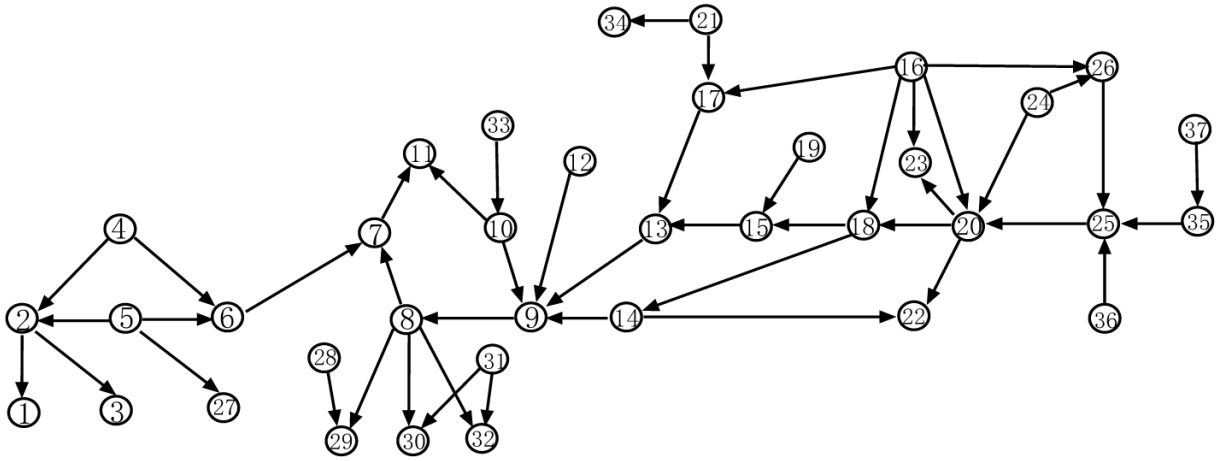

Figure 12.
The ALARM network

Input: A separation tree $\mathcal{T}$ of $\mathcal{G}$; perfect conditional independence knowledge about P.
Output: The skeleton $\mathcal{G}'$ of $\mathcal{G}$; a set $\mathcal{S}$ of c-separators.

Set $\mathcal{S} = \emptyset$;
foreach *Tree node* C_h **do**
  Start from a complete undirected graph $\mathcal{G}_h$ with vertex set C_h;
  foreach *Vertex pair* $\{u, v\} \subset C_h$ **do**
    if $\exists S_{uv} \subset C_h$ *s.t.* $u \perp\!\!\!\perp v | S_{uv}$ **then**
      Delete the edge (u, v) in $\mathcal{G}_h$;
      Add S_{uv} to $\mathcal{S}$;
    end
  end
end
Combine the graphs $\mathcal{G}_h = (C_h, E_h), h = 1, \cdots, H$ into an undirected graph $\mathcal{G}' = (V, \cup_{h=1}^{H} E_h)$;
foreach *Vertex pair* $\{u, v\}$ *contained in more than one tree node and* $(u, v) \in \mathcal{G}'$ **do**
  if $\exists C_h$ *s.t.* $\{u, v\} \subset C_h$ *and* $(u, v) \notin E_h$ **then**
    Delete the edge (u, v) in $\mathcal{G}'$;
  end
end
foreach *Vertex pair* $\{u, v\}$ *contained in more than one tree node and* $(u, v) \in \mathcal{G}'$ **do**
  if $u \perp\!\!\!\perp v | S_{uv}$ *for some* $S_{uv} \subset \mathrm{ne}_{\mathcal{G}'}(u)$ *or* $\mathrm{ne}_{\mathcal{G}'}(v)$ *such that it is not a subset of any* C_h *with* $\{u, v\} \subset C_h$ **then**
    Delete the edge (u, v) in $\mathcal{G}'$;
    Add S_{uv} to $\mathcal{S}$;
  end
end

Algorithm 1.
Skeleton Recovery with a Separation Tree.

Input: Perfect conditional independence knowledge; the skeleton $\mathcal{G}'$ and the set $\mathcal{S}$ of c-separators obtained in Algorithm 1.

Output: The pattern $\mathcal{G}^*$ of $\mathcal{G}$.

Initialize $\mathcal{G}^* = \mathcal{G}'$;
foreach *Ordered pair* $[u, v]$ *such that* $S_{uv} \in \mathcal{S}$ **do**
  foreach $u - w$ *in* $\mathcal{G}^*$ **do**
    if $u \not\perp\!\!\!\perp v | S_{uv} \cup \{w\}$ **then**
      Orient $u - w$ as $u \rightarrow w$ in $\mathcal{G}^*$;
    end
  end
end
Take the pattern of $\mathcal{G}^*$.

Algorithm 2.
Complex Recovery.

Table 1

Simulation results for Gaussian samples from the ALARM network. Averages and standard errors for (FP, FN, SHD) from 100 runs.

Alg (Level α)	*n* = 1000	*n* = 2000	*n* = 5000	*n* = 10000
DAG	(3.09, 4.05, 17.3)	(3.17, 3.14, 15.2)	(3.48, 2.07, 12.4)	(3.16, 1.62, 10.5)
(0.05)	(0.18, 0.19, 0.61)	(0.16, 0.16, 0.54)	(0.18, 0.16, 0.55)	(0.17, 0.11, 0.46)
DAG	(0.87, 3.41, 12.2)	(0.87, 2.60, 9.90)	(0.71, 1.60, 6.02)	(0.58, 1.24, 5.23)
(0.01)	(0.08, 0.19, 0.64)	(0.09, 0.17, 0.54)	(0.08, 0.14, 0.41)	(0.08, 0.10, 0.36)
DAG	(0.61, 3.34, 11.4)	(0.54, 2.50, 8.77)	(0.36, 1.49, 5.43)	(0.33, 1.08, 4.14)
(0.005)	(0.08, 0.19, 0.60)	(0.07, 0.16, 0.52)	(0.07, 0.12, 0.38)	(0.06, 0.11, 0.39)
LCD	(2.06, 5.19, 19.0)	(2.10, 4.25, 16.7)	(2.5, 3.07, 14.0)	(2.15, 2.38, 11.3)
(0.05)	(0.16, 0.19, 0.58)	(0.15, 0.17, 0.53)	(0.15, 0.15, 0.50)	(0.15, 0.14, 0.39)
LCD	(0.41, 4.93, 15.8)	(0.34, 4.01, 12.9)	(0.44, 2.82, 9.79)	(0.30, 2.10, 7.11)
(0.01)	(0.06, 0.21, 0.61)	(0.06, 0.17, 0.50)	(0.07, 0.15, 0.41)	(0.05, 0.13, 0.40)
LCD	(0.22, 4.86, 15.3)	(0.12, 3.85, 12.2)	(0.13, 2.85, 9.14)	(0.14, 1.95, 6.49)
(0.005)	(0.05, 0.21, 0.61)	(0.03, 0.16, 0.52)	(0.04, 0.14, 0.41)	(0.03, 0.13, 0.40)
PC	(4.72, 7.98, 38.4)	(5.03, 6.79, 38.5)	(4.94, 5.19, 35.2)	(4.41, 4.21, 31.9)
(0.05)	(0.22, 0.23, 0.73)	(0.23, 0.23, 0.73)	(0.24, 0.20, 0.79)	(0.26, 0.19, 0.89)
PC	(3.45, 9.23, 37.9)	(3.11, 7.78, 34.8)	(3.27, 5.88, 31.5)	(2.98, 4.87, 30.9)
(0.01)	(0.19, 0.26, 0.78)	(0.19, 0.22, 0.76)	(0.20, 0.21, 0.79)	(0.22, 0.20, 0.88)
PC	(3.14, 9.61, 38.1)	(2.95, 8.05, 35.6)	(3.03, 6.15, 31.4)	(2.88, 5.13, 30.4)
(0.005)	(0.20, 0.27, 0.69)	(0.18, 0.23, 0.79)	(0.20, 0.21, 0.78)	(0.22, 0.20, 0.79)

Table 2

Simulation results for discrete samples from the ALARM network. Averages and standard errors for (FP, FN, SHD) from 100 runs.

Alg (Level α)	$n = 1000$	$n = 2000$	$n = 5000$	$n = 10000$
MMHC	(0.27, 7.77, 34.0)	(0.16, 5.15, 27.9)	(0.08, 2.61, 20.6)	(0.03, 1.53, 16.0)
(0.05)	(0.05, 0.25, 0.57)	(0.04, 0.19, 0.48)	(0.03, 0.15, 0.47)	(0.02, 0.10, 0.45)
MMHC	(0.13, 8.39, 34.6)	(0.08, 5.53, 28.2)	(0.07, 2.96, 21.2)	(0.00, 1.64, 16.1)
(0.01)	(0.04, 0.26, 0.57)	(0.03, 0.19, 0.48)	(0.03, 0.16, 0.46)	(0.00, 0.10, 0.45)
MMHC	(0.13, 8.79, 35.3)	(0.09, 5.77, 28.6)	(0.06, 3.09, 21.4)	(0.01, 1.75, 16.3)
(0.005)	(0.04, 0.26, 0.57)	(0.03, 0.18, 0.51)	(0.03, 0.16, 0.47)	(0.01, 0.11, 0.46)
REC	(6.20, 4.95, 40.1)	(6.49, 3.52, 35.9)	(6.20, 1.77, 28.9)	(6.39, 0.80, 25.1)
(0.05)	(0.24, 0.19, 0.71)	(0.24, 0.13, 0.74)	(0.23, 0.11, 0.68)	(0.23, 0.08, 0.73)
REC	(1.57, 5.52, 33.2)	(1.82, 3.64, 27.2)	(2.06, 1.80, 20.0)	(2.31, 0.86, 16.2)
(0.01)	(0.13, 0.22, 0.58)	(0.13, 0.14, 0.48)	(0.12, 0.11, 0.56)	(0.14, 0.09, 0.52)
REC	(1.02, 5.72, 32.9)	(1.23, 3.76, 26.3)	(1.20, 1.86, 18.6)	(1.64, 0.90, 14.6)
(0.005)	(0.10, 0.22, 0.55)	(0.10, 0.15, 0.49)	(0.08, 0.12, 0.49)	(0.12, 0.09, 0.47)
SC	(0.63, 7.35, 34.2)	(0.50, 4.71, 27.8)	(0.59, 2.50, 21.7)	(0.81, 1.53, 18.1)
(5)	(0.07, 0.26, 0.60)	(0.09, 0.18, 0.49)	(0.09, 0.14, 0.53)	(0.11, 0.10, 0.56)
SC	(0.85, 7.30, 34.5)	(0.76, 4.65, 28.3)	(0.94, 2.36, 22.0)	(1.30, 1.32, 18.1)
(10)	(0.09, 0.25, 0.64)	(0.09, 0.18, 0.52)	(0.09, 0.14, 0.53)	(0.13, 0.10, 0.59)
LCD	(2.92, 8.49, 38.9)	(2.50, 5.94, 32.17)	(2.18, 3.41, 25.17)	(1.99, 2.18, 19.8)
(0.05)	(0.17, 0.21, 0.53)	(0.16, 0.18, 0.51)	(0.14, 0.13, 0.46)	(0.12, 0.10, 0.50)
LCD	(1.07, 8.16, 37.8)	(0.97, 5.63, 31.7)	(0.80, 3.40, 23.1)	(0.68, 2.11, 17.2)
(0.01)	(0.09, 0.20, 0.46)	(0.09, 0.16, 0.42)	(0.09, 0.13, 0.46)	(0.09, 0.09, 0.42)
LCD	(0.69, 8.14, 38.4)	(0.67, 5.84, 31.8)	(0.40, 3.31, 23.2)	(0.41, 2.09, 17.1)
(0.005)	(0.08, 0.19, 0.41)	(0.08, 0.17, 0.43)	(0.07, 0.13, 0.45)	(0.07, 0.09, 0.40)

Algorithm 3

Separation Tree Construction with Labeled Block Ordering.

Input: A labeled block ordering $B = (V_i^{l_i}, i = 1, \square, k)$ of V; perfect conditional independence knowledge.

Output: A separation tree $\mathcal{T}$ of $\mathcal{G}$.

1 Construct a DAG $\mathcal{D}$ with blocks V_i, $i = 1, \cdots, k$;

2 Construct a junction tree $\mathcal{T}$ by triangulating $\mathcal{D}$;

3 In $\mathcal{T}$, replace each block V_i by the original vertices it contains.

Algorithm 4

Separation Tree Refinement.

Input: A crude separation tree $\mathcal{T}_c$ for $\mathcal{G}$; perfect conditional independence knowledge.

Output: A refined separation tree $\mathcal{T}$ for $\mathcal{G}$.

1 Construct an undirected independence subgraph over each node of $\mathcal{T}_c$;

2 Combine the subgraphs into a global undirected independence graph $\mathcal{G}^{-}$ whose edge set is the union of all edge sets of subgraphs;

3 Construct a junction tree $\mathcal{T}$ of $\mathcal{G}^{-}$.