Available online at www.sciencedirect.com

SCIENCE DIRECT®

Artificial Intelligence 170 (2006) 422–439

Artificial Intelligence

www.elsevier.com/locate/artint

Decomposition of structural learning about directed acyclic graphs

Xianchao Xie [a], Zhi Geng [a,*], Qiang Zhao [a,b]

[a] *School of Mathematical Sciences, LMAM, Peking University, Beijing 100871, China*
[b] *Institute of Population Research, Peking University, Beijing 100871, China*

Received 8 February 2005; received in revised form 21 November 2005; accepted 16 December 2005

Available online 3 February 2006

Abstract

In this paper, we propose that structural learning of a directed acyclic graph can be decomposed into problems related to its decomposed subgraphs. The decomposition of structural learning requires conditional independencies, but it does not require that separators are complete undirected subgraphs. Domain or prior knowledge of conditional independencies can be utilized to facilitate the decomposition of structural learning. By decomposition, search for d-separators in a large network is localized to small subnetworks. Thus both the efficiency of structural learning and the power of conditional independence tests can be improved.

Keywords: Bayesian network; Conditional independence; Decomposition; Directed acyclic graph; Junction tree; Structural learning; Undirected graph

1. Introduction

Directed acyclic graphs (DAGs) are widely used to represent independencies, conditional independencies and causal relationships among variables [5,6,8,15,18,19,24]. Structure recovery of DAGs has been discussed by many authors [5,12,19,24,27]. Search for d-separators of vertex pairs is a key issue for orientation of directed edges and for recovering DAG structures and causal relations among variables. To recover structure of DAGs, Verma and Pearl [27] presented the inductive causation (IC) algorithm which searches for a d-separator S from all possible variable subsets such that two variables u and v are independent conditional on S. A systematic way of searching for d-separators in increasing order of cardinality was proposed in [23,24]. The PC algorithm limits possible d-separators to vertices that are adjacent to u and v [19,24]. A decomposition approach of searching for d-separators was presented in [11]. To decompose a graph into two subgraphs, the approach in [11] needs a moral graph and it requires two conditions: (i) variable sets in two subgraphs are independent conditional on their separator and (ii) the separator must be a complete subgraph in the moral graph. The two conditions are often used to define decomposition of an undirected graph, see Definitions 2.1 and 2.2 in [15].

In this paper, we present a decomposition approach for recovering structures of DAGs. The ultimate use of the constructed DAGs is to interpret association and causal relationships among variables. Decomposition in our approach

* Corresponding author.
E-mail addresses: xie1981@water.pku.edu.cn (X.C. Xie), zgeng@math.pku.edu.cn (Z. Geng), zhq@math.pku.edu.cn (Q. Zhao).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2005.12.004

only needs an undirected independence graph which may not be a moral graph and may have extra edges added to the moral graph, and further it only requires condition (i) of conditional independencies but it does not require condition (ii) of complete separators. Thus the decomposition is weaker than the weak decomposition defined in [15] and also than that proposed in [11]. Deleting condition (ii) from decomposition conditions is important since it is difficult with domain or prior knowledge to judge whether a separator is complete or not. In many practical applications, condition (i) of conditional independencies can be judged with domain or prior knowledge or with incompletely observed data patterns, such as Markov chain, chain graphical models, dynamic or temporal models, file-matching for large databases and split questionnaire survey sampling [6,16,20].

Section 2 gives notation and definitions. In Section 3, we show a condition for decomposing structural learning of DAGs. Construction of d-separation trees to be used for decomposition is discussed in Section 4. We propose the main algorithm and then give an example in Section 5 to illustrate our approach for recovering the global structure of a DAG. Section 6 discusses the complexity and advantages of the proposed algorithms. Conclusions are given in Section 7. The proofs of our main results and algorithms are given in Appendix A.

2. Notation and definitions

2.1. Directed acyclic graphs and undirected graphs

Let $\vec{G}_V = (V, \vec{E}_V)$ denote a DAG where $V = \{X_1, \ldots, X_n\}$ is the vertex set and $\vec{E}_V$ the set of directed edges. A directed edge from a vertex u to a vertex v is denoted by $\langle u, v\rangle$. We assume that there is no directed loop in $\vec{G}_V$. We say that u is a parent of v and v is a child of u if there is a directed edge $\langle u, v\rangle$, and denote the set of all parents of a vertex v by $pa(v)$. We say that two vertices u and v are adjacent in $\vec{G}_V$ if there is an edge connecting them. A path l between two distinct vertices u and v is a sequence of distinct vertices in which the first vertex is u, the last one is v and two consecutive vertices are connected by an edge, that is, $l = (c_0 = u, c_1, \ldots, c_{m-1}, c_m = v)$ where $\langle c_{i-1}, c_i\rangle$ or $\langle c_i, c_{i-1}\rangle$ is contained in $\vec{E}_V$ for $i = 1, \ldots, m$ ($m \geqslant 1$), and $c_i \neq c_j$ for all $i \neq j$. We say that u is an ancestor of v and v is a descendant of u if there is a path between u and v in $\vec{G}_V$ and all edges on this path point at the direction toward v. The set of ancestors of v is denoted as $an(v)$, and define $An(v) = an(v) \cup \{v\}$. A path l is said to be d-separated by a set of vertices Z if and only if

(1) l contains a 'chain': $u \to v \to w$ or a 'fork' $u \leftarrow v \to w$ such that the middle vertex v is in Z, or
(2) l contains a 'collider' $u \to v \leftarrow w$ such that the middle vertex v is not in Z and no descendant of v is in Z.

Two distinct sets X and Y of vertices are d-separated by a set Z if Z d-separates every path from any vertex in X to any vertex in Y; We call Z a d-separator of X and Y. In a DAG $\vec{G}_V$, a collider $u \to v \leftarrow w$ is called a v-structure if u and w are non-adjacent in $\vec{G}_V$.

Let $\bar{G}_V = (V, \bar{E}_V)$ denote an undirected graph where $\bar{E}_V$ is a set of undirected edges. An undirected edge between two vertices u and v is denoted by (u, v). For a subset A of V, let $\bar{G}_A = (A, \bar{E}_A)$ be the subgraph induced by A and $\bar{E}_A = \{e \in \bar{E}_V \mid e \in A \times A\} = \bar{E}_V \cap (A \times A)$. An undirected graph is called complete if any pair of vertices is connected by an edge. For an undirected graph, we say that vertices u and v are separated by a set of vertices Z if each path between u and v passes through Z. We say that two distinct vertex sets X and Y are separated by Z if and only if Z separates every pair of vertices u and v for any $u \in X$ and $v \in Y$. We say that an undirected graph $\bar{G}_V$ is an undirected independence graph for a DAG $\vec{G}_V$ if the fact that a set Z separates X and Y in $\bar{G}_V$ implies that Z d-separates X and Y in $\vec{G}_V$. We say that $\bar{G}_V$ can be decomposed into subgraphs $\bar{G}_A$ and $\bar{G}_B$ if

(1) $A \cup B = V$, and
(2) $C = A \cap B$ separates $A \setminus B$ and $B \setminus A$ in $\bar{G}_V$.

The above decomposition does not require that the separator C is complete, which is required for weak decomposition defined in [15] and for decomposition of search for v-structures proposed in [11]. In the next section, we show that a problem of structural learning of a DAG can also be decomposed into problems for its decomposed subgraphs even if the separator is not complete.

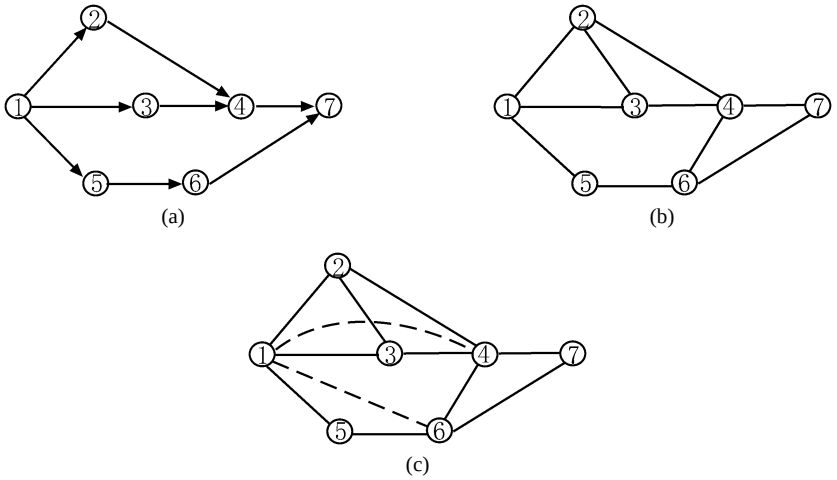


Fig. 1. A directed graph, a moral graph and a triangulated graph. (a) The DAG $\vec{G}_V$. (b) The moral graph $\vec{G}_V^m$. (c) A triangulated graph $\vec{G}_V^t$.

A triangulated graph is an undirected graph whose every cycle of length $\geqslant 4$ possesses a chord [15]. For an undirected graph $\bar{G}_V$ which is not triangulated, we can add extra edges to it such that it becomes to be a triangulated graph, denoted by $\bar{G}_V^t$.

Define a moral graph $\vec{G}_V^m$ for a DAG $\vec{G}_V$ to be an undirected graph $\bar{G}_V = (V, \bar{E}_V)$ whose vertex set is V and whose edge set is constructed by marrying parents and dropping directions, that is, $\bar{E}_V = \{(u, v)\colon \langle u, v\rangle \text{ or } \langle v, u\rangle \in \vec{E}_V\} \cup \{(u, v)\colon (u, w, v) \text{ forms a } v\text{-structure}\}$ [15]. An undirected edge added for marrying parents is called a moral edge. The moral graph $\vec{G}_V^m$ is an undirected independence graph for $\vec{G}_V$ [15].

Example 1. Consider a DAG $\vec{G}_V$ in Fig. 1(a). $2 \to 4 \leftarrow 3$ and $4 \to 7 \leftarrow 6$ are two v-structures. A path $l = (2, 1, 5)$ is d-separated by vertex 1, and another path $l' = (2, 4, 7, 6, 5)$ is d-separated by an empty set since $4 \to 7 \leftarrow 6$ is a collider. Vertices 2 and 5 are d-separated by vertex 1. $an(4) = \{1, 2, 3\}$ and $An(4) = \{1, 2, 3, 4\}$. The moral graph $\vec{G}_V^m$ is shown in Fig. 1(b), whose edges $(2, 3)$ and $(4, 6)$ are moral edges. Set $\{2, 3, 5\}$ separates $\{1\}$ and $\{4, 6, 7\}$, and thus $\vec{G}_V^m$ can be decomposed into two undirected subgraphs over $\{1, 2, 3, 5\}$ and $\{2, \ldots, 7\}$. An undirected independence graph for $\vec{G}_V$ may have extra undirected edges added to the moral graph, say edges $(1, 4)$ and $(1, 6)$ added to $\vec{G}_V^m$, see dashed edges in Fig. 1(c). The graph in Fig. 1(c) is a triangulated graph of $\vec{G}_V^m$.

Given a DAG $\vec{G}_V$, a joint distribution or density of variables $X_1, \ldots, X_n$ is

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid pa_i),$$

where $P(x_i \mid pa_i)$ is the conditional probability or density of X_i given $pa(X_i) = pa_i$. The DAG $\vec{G}_V$ and the distribution P are said to be compatible [19] and P obeys the global directed Markov property of $\vec{G}_V$ [15]. Let $X \perp\!\!\!\perp Y$ denote the independence of X and Y, and $X \perp\!\!\!\perp Y \mid Z$ the conditional independence of X and Y given Z. If sets X and Y are d-separated by Z, then X is independent of Y conditional on Z in every distribution that is compatible with $\vec{G}_V$ [19]. In this paper, we assume that all the distributions are compatible with $\vec{G}$. We also assume that all independencies of a probability distribution of variables in V can be checked by d-separations of $\vec{G}_V$, called the faithfulness assumption [24]. The faithfulness assumption means that all independencies and conditional independencies among variables can be represented by $\vec{G}_V$.

The global skeleton is an undirected graph obtained by dropping direction of a DAG. Thus the absence of an edge (u, v) implies that there is a variable subset S of V such that u and v are independent conditional on S, that is, $u \perp\!\!\!\perp v|S$ for some $S \subseteq (V \setminus \{u, v\})$. Two DAGs over the same variable set are called Markov equivalent if they induce the same conditional independence restrictions. Two DAGs are Markov equivalent if and only if they have the same global skeleton and the same set of v-structures [27]. An equivalence class of DAGs consists of all DAGs which are Markov equivalent, and it is represented as a partially directed graph where the directed edges represent arrows

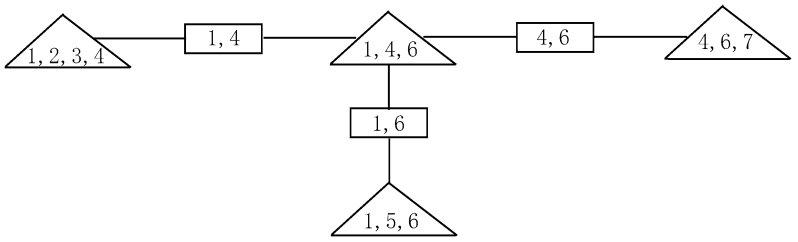


Fig. 2. A d-separation tree.

that are common to every DAG in it, while the undirected edges represent that any proper orientation of them leads to a Markov equivalent DAG. Therefore the goal of structural learning is to construct a partially directed graph to represent the equivalence class. A local skeleton for a subset A of variables is an undirected subgraph for A in which the absence of an edge (u, v) implies that there is a subset S of A such that $u \perp\!\!\!\perp v|S$.

2.2. d-separation trees

To depict separations and conditional independencies among multiple variable sets, we introduce a notion of d-separation trees in this subsection. Let $\mathcal{C}$ be a collection of variable sets, that is, $\mathcal{C} = \{C_1, \ldots, C_H\}$, such that $\bigcup_{h=1}^{H} C_h = V$ and $C_i \not\subseteq C_j$ for $i \neq j$. Let T be a tree whose every node is a variable set in $\mathcal{C}$ and is displayed as a triangle. The term 'node' is used for a tree to distinguish the term 'vertex' for a graph. An edge $e_h = (C_i, C_j)$ connects nodes C_i and C_j in T, and it is attached with a separator S displayed as a rectangle, which is the intersection of adjacent nodes, that is, $S = C_i \cap C_j$; We say that the separator S connects the nodes C_i and C_j. Removing a separator S from the tree T splits T into two subtrees T_1 and T_2 with node sets $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively. Let $V_i = \bigcup_{C \in \mathcal{C}_i} C$ be the union of the nodes in the subtree T_i for $i = 1$ and 2.

Definition 1. A tree T is said to be a d-separation tree for a DAG $\vec{G}$ if any separator S in T d-separates, in $\vec{G}$, the vertex sets $V_1 \setminus S$ and $V_2 \setminus S$ of two subtrees T_1 and T_2 obtained by removing S.

The notion of a d-separation tree is very similar to that of a junction tree. A d-separation tree is defined with d-separation and it does not need that every node is a clique, while a junction tree is a d-separation tree (see Theorem 2).

Example 1. (*Continued*) Let $\mathcal{C} = \{\{1, 2, 3, 4\}, \{1, 4, 6\}, \{1, 5, 6\}, \{4, 6, 7\}\}$ be a collection of variable sets. A d-separation tree with $\mathcal{C}$ as the node set is depicted in Fig. 2. Removing the separator $S = \{1, 4\}$, we obtain two subtrees T_1 and T_2 with the node sets $\mathcal{C}_1 = \{\{1, 2, 3, 4\}\}$ and $\mathcal{C}_2 = \{\{1, 4, 6\}, \{1, 5, 6\}, \{4, 6, 7\}\}$ respectively, and the separator S d-separates $V_1 \setminus S = \{2, 3\}$ and $V_2 \setminus S = \{5, 6, 7\}$ in $\vec{G}_V$.

2.3. Hypergraph

A collection of variable sets $\mathcal{C} = \{C_1, \ldots, C_H\}$ is said to be a hypergraph on V where each hyperedge C_h is a nonempty subset of variables, and $\bigcup_{h=1}^{H} C_h = V$, see Chapter 17 in [4]. A hypergraph is a reduced hypergraph if $C_i \not\subseteq C_j$ for $i \neq j$ [3]. In this paper, only reduced hypergraphs are used, and thus simply called hypergraphs. In Section 4.2, a hyperedge can be used to represent the domain knowledge of association among variables or to represent multiple databases with overlapping.

Example 1. (*Continued*) Let $\mathcal{C} = \{\{1, 2, 3, 4\}, \{1, 3, 5, 6\}, \{4, 6, 7\}\}$ be a hypergraph, as shown in Fig. 3.

3. Decomposition of structural learning

In this section, we show that if vertices u and v are d-separated by a d-separator S in a DAG $\vec{G}_V$, then u and v are not contained by any node of the d-separation tree for $\vec{G}_V$ or there exists a node C that contains u, v and S' such that $u \perp\!\!\!\perp v|S'$, and vice versa. Applying this result to structural learning, we can split a problem of searching for

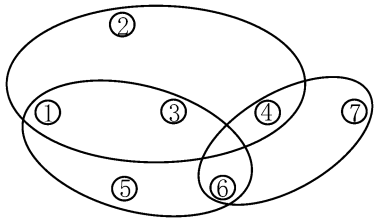


Fig. 3. A hypergraph.

d-separators and building the skeleton of a DAG into small problems for every node of T. This result may also be useful for designing observational studies for recovering local causal relationships.

Theorem 1. *Let T be a d-separation tree for a DAG $\vec{G}$. Vertices u and v are d-separated by S ($\subseteq V$) in $\vec{G}_V$ if and only if* (i) *u and v are not contained together in any node C of T or* (ii) *there exists a node C that contains both u and v such that a subset S' of C d-separates u and v.*

According to Theorem 1, a problem of searching for a d-separator S of u and v in all possible subsets of V is localized to all possible subsets of nodes in a d-separation tree that contain u and v. For a given d-separation tree T with the node set $\mathcal{C} = \{C_1, \ldots, C_H\}$, we can recover the skeleton and all v-structures for a DAG as follows. First we construct a local skeleton for every node C_h of T, which is constructed by starting with a complete undirected subgraph and removing an undirected edge (u, v) if there is a subset S of C_h such that u and v are independent conditional on S. Next to construct the global skeleton, we combine all these local skeletons together and remove edges that are present in some local skeletons but absent in other local skeletons. Then we determine every v-structure if two non-adjacent vertices u and v have a common neighbor in the global skeleton but the neighbor is not contained in the d-separator of u and v. Finally we can orient more undirected edges if each opposite of them creates either a directed cycle or a new v-structure [17]. This process is formally described in the following algorithm.

Algorithm 1 *(Construct the equivalence class of DAGs from a d-separation tree).*

1. Input: a d-separation tree T with a node set $\mathcal{C} = \{C_1, \ldots, C_H\}$.
2. Construct a local skeleton $\bar{G}_h$ for each h separately:
   - Initialize $\bar{G}_h$ as a complete undirected graph;
   - If there exists a subset S_{uv} of $C_h \setminus \{u, v\}$ such that $u \perp\!\!\!\perp v | S_{uv}$, then delete edge (u, v) from $\bar{G}_h$ and save S_{uv} to the d-separator list $\mathcal{S}$;
3. Construct the global skeleton $\bar{G}_V$:
   - Initialize the edge set $\bar{E}_V$ of $\bar{G}_V$ as the union of all edge sets of $\bar{G}_h$, $h = 1, \ldots, H$;
   - For a pair of vertices u and v contained in several local skeletons, delete edge (u, v) from $\bar{E}_V$ if it is absent in some skeleton.
4. For each d-separator S_{uv} in the list $\mathcal{S}$, determine a v-structure $u \to w \leftarrow v$ if u–w–v appears in the global skeleton and w is not in S_{uv}.
5. Orient other edges if each opposite of them creates either a directed cycle or a new v-structure.
6. Output: the equivalence class of DAGs.

According to Theorem 1, we can prove that the global skeleton and all v-structures obtained by applying the above decomposition algorithm are correct, that is, they are the same as those obtained from the joint distribution of V, see Appendix A for the detail proof. Note that separators in a d-separation tree may not be complete in the moral graph. Thus the decomposition is weaker than the decomposition usually defined for parameter estimation [5,10,15].

Example 1. (*Continued*) Consider the d-separation tree in Fig. 2 as the input of Algorithm 1. At step 2, we separately build the local skeleton for each node of T, as shown in Fig. 4(a). At step 3, the global skeleton is obtained by combining the local skeletons in Fig. 4(a). Edge $(1, 6)$ in the subgraph for node $\{1, 4, 6\}$ is a spurious edge and it is removed from the global graph since the edge $(1, 6)$ does not appear in the subgraph for node $\{1, 5, 6\}$, that is,

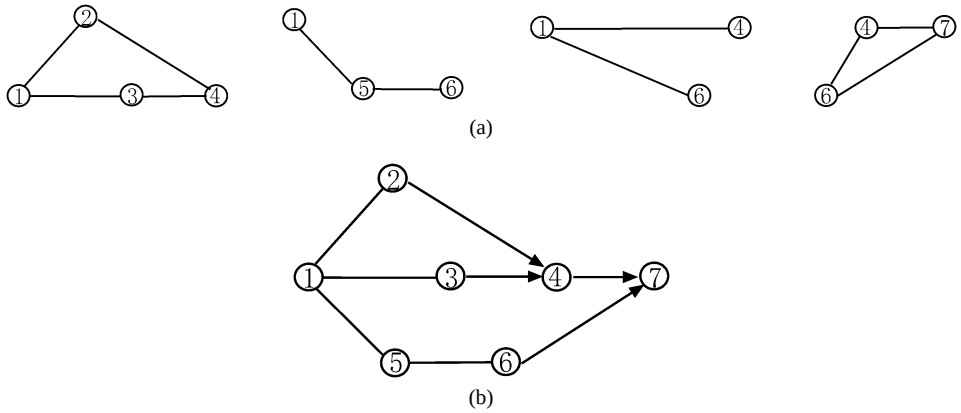


Fig. 4. Skeleton and v-structures for $\vec{G}_V$. (a) Local skeletons for every node of T. (b) The global skeleton and all v-structures.

there is a d-separator $S_{16} = \{5\}$ which d-separates 1 and 6 in the subgraph. Similarly edge $(4, 6)$ is removed since it is absent in the third local skeleton. At step 4, all v-structures can be determined, see Fig. 4(b). For example, the v-structure $2 \to 4 \leftarrow 3$ can be determined since for $S_{23} = \{1\} \in \mathcal{S}$, 2–4–3 appears in the global skeleton and vertex 4 is not contained in S_{23}. Similarly, the v-structure $4 \to 7 \leftarrow 6$ is found since for $S_{46} = \{1\} \in \mathcal{S}$, 4–7–6 appears in the global skeleton and vertex 7 is not contained in S_{46}. But for $S_{16} = \{5\} \in \mathcal{S}$, 1–5–6 appears, but vertex 5 is contained in S_{16}, and thus it is not a v-structure. For this DAG, no other edges can be oriented at step 5. The partially directed graph in Fig. 4(b) is the equivalence class of the DAG in Fig. 1(a).

By Theorem 1, it is ensured that the global skeleton and all v-structures in $\vec{G}_V$ can be recovered by combining subgraphs over all nodes of T. In the next section, we discuss how to construct a d-separation tree.

4. Constructing a d-separation tree

In this section, we discuss how to construct a d-separation tree from observed data or from domain or prior knowledge of conditional independencies or from a collection of databases. We first propose an approach in which an undirected independence graph and then a junction tree are built from observed data, and we show that a junction tree is a d-separation tree. Next we propose an approach for constructing a d-separation tree from domain knowledge or from a collection of databases with different observed variable sets.

4.1. Constructing a d-separation tree from observed data

In several algorithms of structural learning, the first step is to construct an undirected independence graph in which the absence of an edge (u, v) implies $u \perp\!\!\!\perp v | V \setminus \{u, v\}$. To construct such an undirected graph, we can start with a complete undirected graph, and then for each pair of variables u and v, an undirected edge (u, v) is removed if u and v are independent conditional on the set of all other variables. For linear Gaussian models, the undirected graph can be efficiently constructed by removing an edge (u, v) if and only if the corresponding entry in the inverse covariance matrix is zero. For discrete data, a test of conditional independence given a large number of discrete variables may be extremely low power. To cope with such difficulty, for a vertex u, we first use information criterion to find a variable subset which contains the Markov blanket of u [18], and then we test independence of u and another variable conditionally on the variable subset [14,26]. This test is efficient if the Markov blanket of u is not large. For discrete data, we can also use the algorithm proposed in [9] which reduces the requirement for testing high order conditional independencies.

A d-separation tree can be built by constructing a junction tree from an undirected independence graph [5] or by using the algorithm presented in Section 4.2.

Theorem 2. *A junction tree constructed from an undirected independence graph for $\vec{G}_V$ is a d-separation tree for $\vec{G}_V$.*

A d-separation tree T only requires that all d-separation properties of T also hold for $\vec{G}_V$, but the reverse is not required. Thus we only need to construct an undirected independence graph that may have fewer conditional independencies than the moral graph, and this means that the undirected independence graph may have extra edges added to the moral graph. Thus the null hypothesis of the absence of an undirected edge may be tested statistically at a larger significance level provided that no nodes of a d-separation tree contain too many variables.

Example 1. (*Continued*) To construct a d-separation tree for $\vec{G}_V$ in Fig. 1(a), at first an undirected independence graph is constructed by starting with a complete graph and removing an edge (u, v) if $u \perp\!\!\!\perp v | V \setminus \{u, v\}$. An undirected graph obtained in this way is the moral graph in Fig. 1(b). In fact, we only need to construct an undirected independence graph which may have extra edges added to the moral graph. Next triangulate the undirected graph and finally obtain the d-separation tree, as shown in Fig. 1(c) and Fig. 2 respectively.

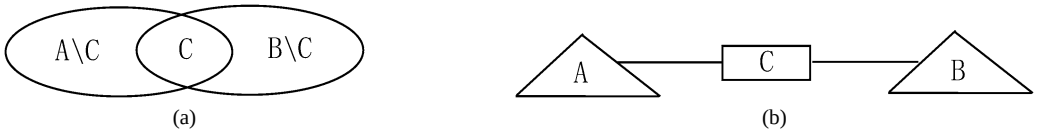
4.2. Constructing a d-separation tree from domain knowledge or from observed data patterns

In this subsection, we propose an approach for constructing a d-separation tree from domain knowledge or from observed data patterns without conditional independence tests. The domain knowledge may be experts' prior knowledge of dependencies among variables, such as Markov chains, chain graphical models and dynamic or temporal models. In many practical applications, such as file-matching and split questionnaire survey sampling [16,20], there are many large databases that have different data patterns, that is, databases have different attributes but may overlay each other. Based on the domain knowledge of dependencies, data patterns of databases can be designed to recover the entire structure for the full variable set V correctly. The problem of reconstructing a DAG from multiple overlapping databases has been considered, and two rules for determining the absence of edges were proposed in [7]. As the author noticed, however, the two rules do not exhaust all possible rules and the existence of others remains an open problem. We theoretically prove that our approach is perfect, that is, the entire DAG reconstructed by using multiple databases is the same as that reconstructed by using joint data over the full variable set V, if multiple databases are designed based on the domain knowledge of dependencies.

We first consider a simple case with two variable sets A and B. Let $C = A \cap B$. Suppose that we have a domain knowledge that any variable in $A \setminus C$ associates with any variable in $B \setminus C$ only through variables in C, which implies $(A \setminus C) \perp\!\!\!\perp (B \setminus C) | C$. We can depict this with a hypergraph and thus obtain a d-separation tree, as shown in Fig. 5(a) and (b) respectively. The domain knowledge can also be seen as the Markov property that the future state $B \setminus C$ is independent of the previous state $A \setminus C$ conditional on the current state C.

For a general case, a domain knowledge of variable dependencies can be represented as a collection of variable sets $\mathcal{C} = \{C_1, \ldots, C_H\}$, in which variables contained in the same set may associate each other directly but variables contained in different sets associate each other through other variables. This means that two variables that are not contained in the same set are independent conditionally on all other variables. We depict such a domain knowledge with a hypergraph. Then equivalently the domain knowledge is legitimate if every edge of the moral graph $\bar{G}_V^m$ of the underlying DAG is contained in some hyperedge in $\mathcal{C}$. A slightly stronger condition for judging the legitimacy of a domain knowledge is that for each variable u, there is a hyperedge C_h in $\mathcal{C}$ which contains both u and its parent set.

On the other hand, in an application study, observed data may have a collection of different observed patterns, $\mathcal{C} = \{C_1, \ldots, C_H\}$, where C_h is the set of observed variables for the hth group of individuals. For example, observed data patterns $\mathcal{C} = \{\{a, b, c\}, \{b, c, d\}\}$ mean that there are two groups of individuals: (1) variables a, b and c are observed but variable d is missing for the first group, and (2) variables b, c and d are observed but variable a is missing for the second group. Data having such observed patterns are not uncommon, such as in the file-matching and split questionnaire survey [16,20]. Given observed data patterns $\mathcal{C} = \{C_1, \ldots, C_H\}$, there is no information on the association among variables that are never observed together, and thus parameters that relate to the association are



(a) (b)

Fig. 5. A domain knowledge about associations among variables. (a) A hypergraph. (b) A d-separation tree.

inestimable without other assumptions. The condition to make our algorithms correct for structural learning from a collection $\mathcal{C}$ is that $\mathcal{C}$ must contain sufficient data such that parameters of the underlying DAG are estimable. For the DAG, its parameters are estimable if, for each variable u, there is an observed data pattern C_h in $\mathcal{C}$ which contains both u and its parent set. Thus a collection $\mathcal{C}$ of observed patterns has sufficient data for correct structural learning if there is a pattern C_h in $\mathcal{C}$ for each u such that C_h contains both u and its parent set in the underlying DAG. Every pattern C_h can be seen as a hyperedge or a maximum complete undirected graph to depict possible association among variables in C_h. When all variables are categorical, a log-linear model with a generating class $\mathcal{C} = \{C_1, \ldots, C_H\}$ is the highest order interaction model without latent variables. We assume that all independencies inferred from observed data patterns are true for the underlying DAG.

Example 1. (*Continued*) Suppose that we have a hypergraph $\mathcal{C} = \{\{1, 2, 3, 4\}, \{1, 3, 5, 6\}, \{4, 6, 7\}\}$, as depicted in Fig. 3. It can be considered as a domain or prior knowledge of associations among variables. The hypergraph in Fig. 3 can also be considered as observed data patterns in three databases, that have different attributes and overlap each other. In the case that all variables are categorical and there is no latent variable, the log-linear model with the generating class $\{[1234], [1356], [467]\}$ includes all interactions among variables that can be estimated from the observed data. Since for each variable u, there is a hyperedge in $\mathcal{C}$ which contains both u and its parent set, the hypergraph $\mathcal{C}$ is a legitimate domain knowledge and a legitimate collection of databases for the underlying DAG in Fig. 1(a). However, if the hypergraph $\mathcal{C}$ is changed to $\mathcal{C}' = \{\{1, 2, 4\}, \{1, 3, 5, 6\}, \{4, 6, 7\}\}$, that is, the hyperedge $\{1, 2, 3, 4\}$ in $\mathcal{C}$ is replaced by $\{1, 2, 4\}$, then there is not any hyperedge in $\mathcal{C}'$ which contains both variable 4 and its parent set $\{2, 3\}$, and thus the hypergraph $\mathcal{C}'$ is neither a legitimate collection of databases nor a legitimate domain knowledge for the DAG.

Now we discuss how to construct a d-separation tree from a hypergraph that represents domain knowledge of dependencies or observed data patterns. First we explain in the following example why the global skeleton of a DAG $\vec{G}_V$ cannot be constructed by combining all subgraphs obtained separately from every data pattern. Then we propose an algorithm for constructing a correct global skeleton.

Example 2. Consider a DAG $\vec{G}_V$ and observed data patterns as depicted in Fig. 6(a). From three data patterns, we get separately three undirected subgraphs 1–2, 1–3 and 2–3. Combining them together, we obtain a combined undirected graph in Fig. 6(b), which is not a correct skeleton for $\vec{G}_V$ since edge $(2, 3)$ is a spurious edge.

Below we propose an algorithm for constructing a d-separation tree T from domain knowledge or from observed data patterns such that a correct skeleton can be constructed by combining subgraphs for nodes of T. Since a hyperedge C_h represents all possible associations among variables in C_h, we first use a complete subgraph over C_h as the undirected independence graph over variables in C_h, and then piece all subgraphs together into an entire undirected graph. To reduce the sizes of tree nodes, we construct a junction tree in terms of triangulating the undirected graph. In this way, the following algorithm constructs a d-separation tree T from a hypergraph $\mathcal{C} = \{C_1, \ldots, C_H\}$ of domain knowledge or observed data patterns.

Algorithm 2 *(Construct a d-separation tree from a hypergraph).*

1. Input: a hypergraph $\mathcal{C} = \{C_1, \ldots, C_H\}$ whose each hyperedge C_h is a variable set.
2. For each hyperedge C_h, construct a complete undirected graph $\bar{G}_h$ with the edge set $\bar{E}_h = \{(u, v), \forall u, v \in C_h\} = C_h \times C_h$.

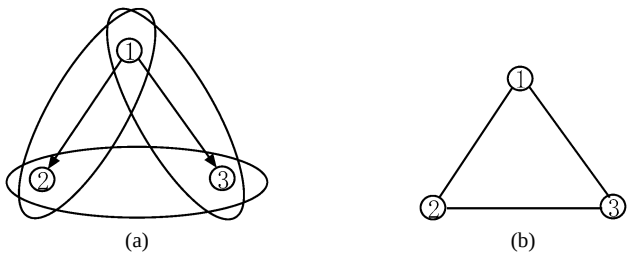


Fig. 6. An incorrect skeleton combined from subgraphs. (a) A DAG and observed data patterns. (b) The combined undirected graph.

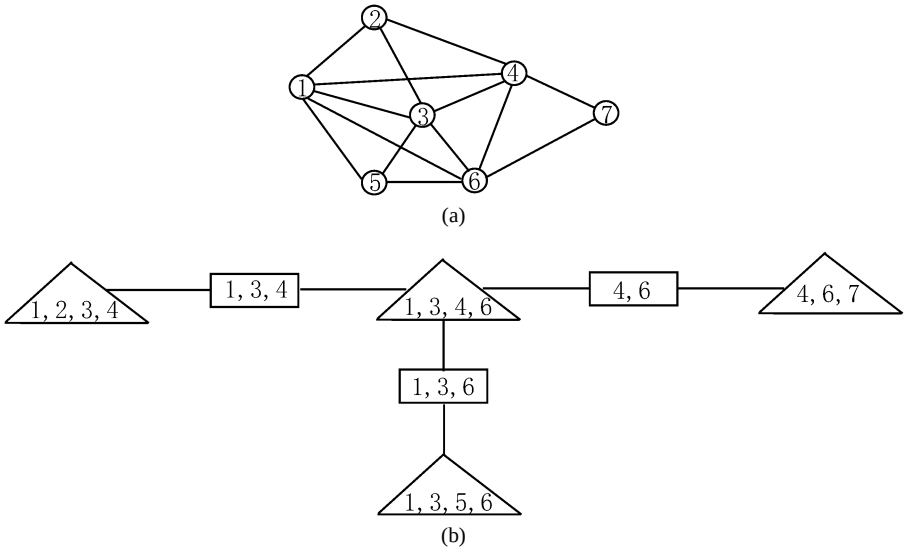


Fig. 7. Construct a d-separation tree from a hypergraph. (a) The undirected graph obtained by combining complete subgraphs. (b) The d-separation tree.

3. Construct the entire undirected graph $\bar{G}_V = (V, \bar{E})$ whose edge set $\bar{E} = \bar{E}_1 \cup \cdots \cup \bar{E}_H$.
4. Construct a junction tree T by triangulating $\bar{G}_V$.
5. Output: T, which is a d-separation tree for the hypergraph $\mathcal{C}$.

The correctness of Algorithm 2 is proven in Appendix A. Note that we do not need any conditional independence test in Algorithm 2 to construct a d-separation tree. In this algorithm, we can use a heuristic triangulation algorithm with less computational complexity to construct a junction tree [2,13,21]. Below we give two examples to illustrate Algorithm 2.

Example 1. (*Continued*) Consider the domain knowledge of associations among all variables given in Fig. 3. At step 2, we construct a complete subgraph for each hyperedge, and then at step 3 we combine them together, as depicted in Fig. 7(a). At step 4, we construct the d-separation tree in Fig. 7(b). Note that no edges are added for triangulation since the undirected graph in Fig. 7(a) is triangulated.

Example 3. Suppose that $V = \{1, \ldots, 6\}$ and that the domain knowledge is $\mathcal{C} = \{C_1, C_2, C_3, C_4\} = \{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 5\}, \{4, 5, 6\}\}$ as depicted in Fig. 8(a). At step 2, we construct four complete subgraphs, and then at step 3 we combine them together, as shown by the undirected graph of solid lines in Fig. 8(b). At step 4, we triangulate it with an edge $(3, 4)$ of a dash line, and then we obtain the d-separation tree in Fig. 8(c).

The nodes of a d-separation tree T constructed from domain knowledge or from observed data patterns may be still quite large. In this case, we use conditional independence tests to reduce the node sizes. We first can construct an undirected independence subgraph for each node, then combine these subgraphs into a global undirected independence graph and finally construct a refined d-separation tree, see Section 5. For every node C_h of T for $h = 1, \ldots, H$, an undirected independence subgraph $\bar{G}_h = (C_h, E_h)$ can be constructed by starting with a complete subgraph and removing an undirected edge (u, v) if u and v are independent conditional on $C_h \setminus \{u, v\}$.

Theorem 3. *Suppose that $\bar{G}_h = (C_h, \bar{E}_h)$ is an independence subgraph for the node C_h of the d-separation tree for $h = 1, \ldots, H$. The undirected graph $\bar{G}_V$ with the edge set $E_V = \bigcup_{h=1}^{H} E_h$ is an undirected independence graph for $\vec{G}_V$.*

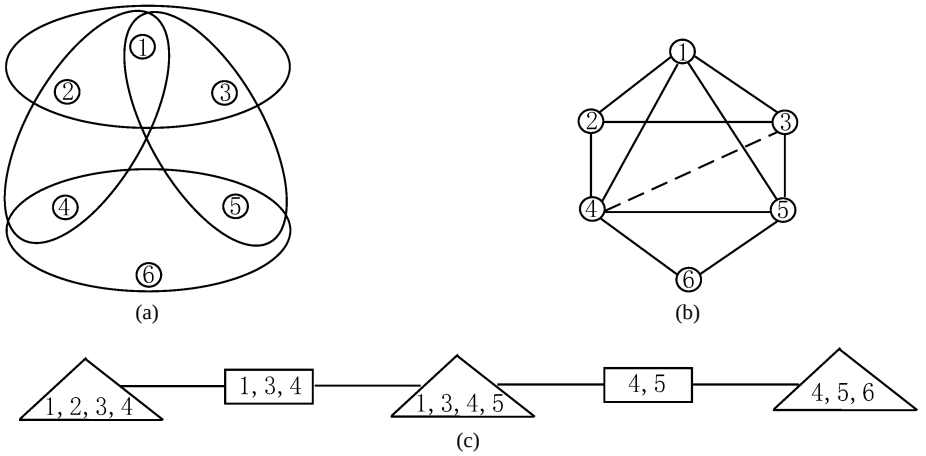


Fig. 8. Construct the d-separation tree. (a) Domain knowledge of associations. (b) The undirected graph and triangulation. (c) The d-separation tree.

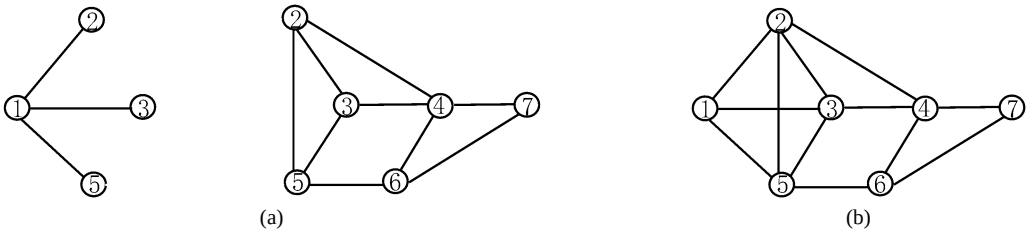


Fig. 9. An undirected independence graph obtained by combining subgraphs. (a) Undirected independence subgraphs for each node. (b) Combining subgraphs in (a).

Note that this combination for obtaining a global undirected independence graph is different from that for obtaining a global skeleton proposed in Section 3. The former pools all edges in subgraphs into the global graph, while the latter deletes an edge (u, v) from the global graph if it is present in a subgraph but absent in another subgraph.

Example 1. (*Continued*) We can suppose that a d-separation tree for $\vec{G}_V$ has the node set $\mathcal{C} = \{\{1, 2, 3, 5\}, \{2, 3, 4, 5, 6, 7\}\}$ since their intersection $S = \{2, 3, 5\}$ d-separates $\{1\}$ and $\{4, 6, 7\}$. Construct undirected independence subgraphs for every node separately, as shown in Fig. 9(a); and then construct the global undirected independence graph with all edges of subgraphs, as shown in Fig. 9(b). The undirected independence graph is not the moral graph since it has two extra edges $(3, 5)$ and $(2, 5)$ added to the moral graph.

5. Illustration of structural learning via decomposition

In this section, we first formally describe the complete algorithm for structural learning of DAGs via decomposition. Then we illustrate the algorithm using the ALARM network in Fig. 10 that is often used to evaluate structural learning algorithms [1,12,24].

Main algorithm *(The decomposition approach for structural learning of DAGs with domain knowledge).*

1. Input: a hypergraph $\mathcal{C} = \{C_1, \ldots, C_H\}$ as the domain knowledge, or databases with observed data patterns $\mathcal{C}$.
2. Call Algorithm 2 to construct a d-separation tree T from the hypergraph $\mathcal{C}$.
3. If the sizes of nodes in T are too large, then refine T with observed data:
   - Construct an undirected independence subgraph over each node of T;
   - Let the edge set $\bar{E}_V$ of the entire undirected independence graph $\bar{G}_V$ be the union of all edge sets of subgraphs;
   - Construct a junction tree T' by triangulating $\bar{G}_V$, which is a d-separation tree.

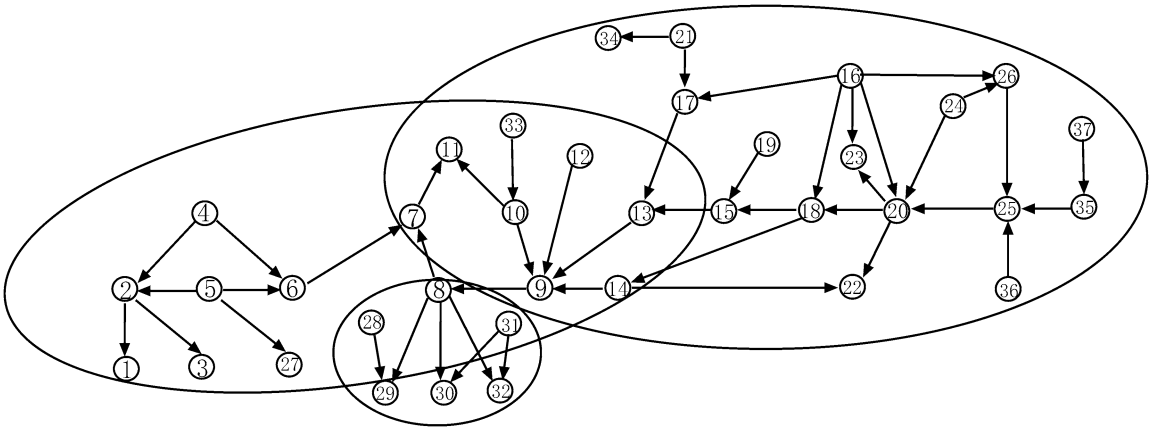


Fig. 10. The ALARM network and domain knowledge.

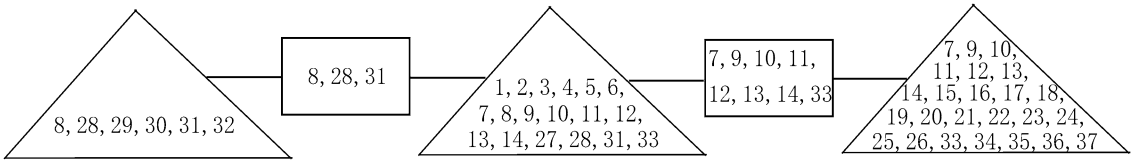


Fig. 11. The d-separation tree T from domain knowledge.

4. Call Algorithm 1 to construct the equivalence class of DAGs from T' (or T if the sizes of nodes are not very large).
5. Output: the equivalence class of DAGs.

The correctness of the main algorithm is proven in Appendix A. To reduces the requirement for testing high order conditional independencies, we first construct a d-separation tree at step 2 based on the prior knowledge, and then if necessary, we further reduce the size of nodes in the d-separation tree at step 3. Note that the undirected independence graph obtained at step 3 may have extra edges, and thus the tests of conditional independencies can be performed with a large significance level. The conditional independence tests performed in Algorithm 1 are marginalized to nodes of the d-separation tree.

The ALARM network in Fig. 10 describes causal relations among 37 variables in a medical diagnostic system for patient monitoring. Using the network, some researchers generate continuous data from normal distributions and others generate discrete data from multinomial distributions [12,24]. Our approach is applicable for both continuous and discrete data. Since the correctness of the algorithms are proved in Appendix A and the complexity analysis and the accuracy of independence tests are discussed in the next section, the main algorithm is illustrated by using conditional independencies from the underlying DAG rather than conditional independence tests from simulated data.

Suppose that we have the domain knowledge of associations among all variables as depicted by the hypergraph in Fig. 10. The hypergraph can also be seen as three databases with overlap. Since for each variable u in the ALARM, there is a hyperedge which contains both u and its parent set, the domain knowledge or the collection of three database is legitimate for structural learning of the ALARM network. However, if the hyperedge $\{8, 28, 29, 30, 31, 32\}$ is changed to another hyperedge $\{8, 29, 30, 31, 32\}$, then there is no hyperedge which contains both variable 29 and its parent set $\{8, 28\}$, and thus parameters of the ALARM network are inestimable and the changed hypergraph as a domain knowledge or as a collection of databases is not legitimate.

At step 2, the d-separation tree T is constructed from the hypergraph, as shown in Fig. 11. Note that at step 2 we need not do any conditional independence test, and we construct the d-separation tree only based on the domain knowledge.

Since the three nodes of T obtained at step 2 are still quite large, we first construct undirected independence subgraphs for three nodes separately at step 3, as shown in Fig. 12; Next combining three subgraphs together and

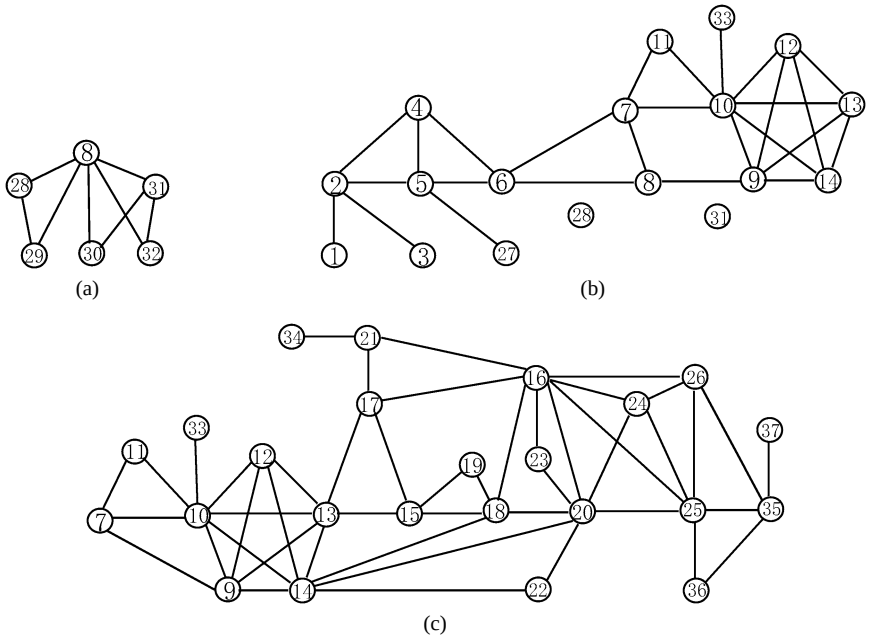


Fig. 12. Undirected independence graphs for nodes of T. (a) A subgraph for the left node of T. (b) A subgraph for the middle node of T. (c) A subgraph for the right node of T.

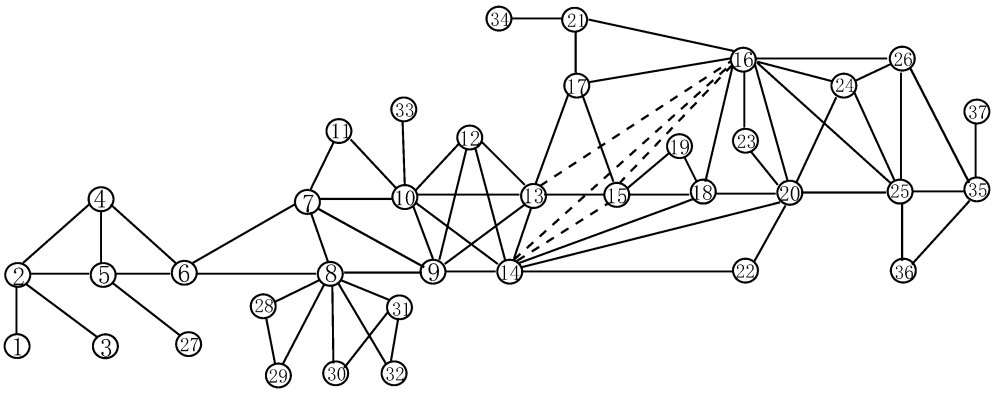


Fig. 13. The global triangulated graph.

triangulating it, we obtain the triangulated independence graph in Fig. 13. From the triangulated graph, we construct a d-separation tree T' in Fig. 14 (separators are omitted), in which the largest node has only 5 variables.

At step 4, we construct a sub-skeleton for every node in T', as shown in Fig. 15. Combining all sub-skeletons together, determining v-structures and orienting edges as much as possible, we obtain the equivalence class in Fig. 16, in which all directed edges are oriented correctly, except that four undirected edges $(5, 27)$, $(10, 33)$, $(21, 34)$ and $(35, 37)$ cannot be oriented because any of their orientation leads to a Markov equivalent DAG. Note that some edges in Fig. 16 are oriented at step 5 of Algorithm 1. For example, the direction of the undirected edge $(1, 2)$ is determined as $\langle 2, 1\rangle$ by $\langle 4, 2\rangle$ so as not to create a new v-structure, and the direction of the undirected edge $(16, 23)$ is determined as $\langle 16, 23\rangle$ by $\langle 16, 20\rangle$ and $\langle 20, 23\rangle$ so as not to create a cycle.

The global skeleton and all v-structures depicted in Fig. 16 are the same as those obtained from the joint distribution of all variables in V. Applying decomposition, we split the graph with 37 vertices into 28 subgraphs, of which the largest one contains only 5 vertices. In such a way, a problem of high-dimensional structural learning is reduced into several small problems.

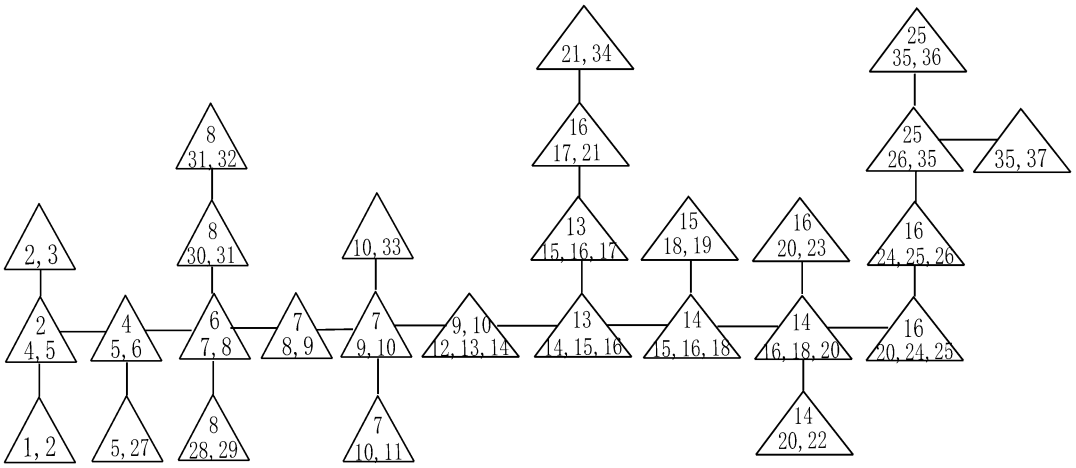


Fig. 14. The d-separation tree T'.

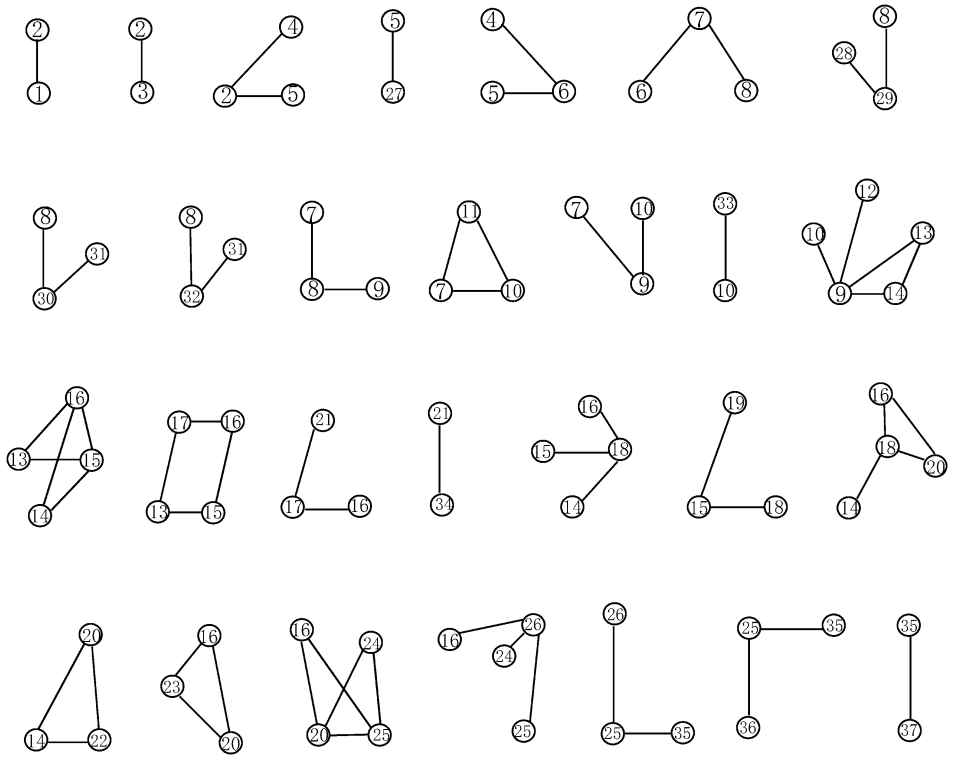


Fig. 15. Skeletons for all nodes of T'.

6. Complexity analysis and advantages

There are several obvious advantages of the decomposition approach for structural learning proposed in this paper. First a d-separation tree can be constructed based on the prior or domain knowledge rather than conditional independence tests. By using the d-separation tree, independence tests are performed only conditionally on smaller sets contained in a node of the d-separation tree rather than on the full set of all other variables. Thus our algorithm has higher power for statistical tests.

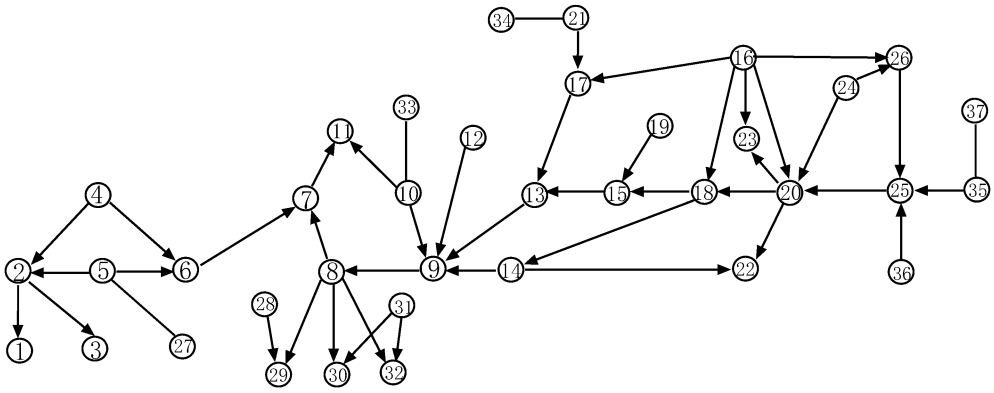


Fig. 16. The partially directed acyclic graph.

Second, the theoretical results proposed in this paper can be applied to scheme design of multiple databases. Without loss of information on structural learning of DAGs, a joint data set can be replaced by a group of incomplete data sets based on the domain or prior knowledge of conditional independencies among variables.

Finally, the computational complexity can be reduced. This complexity analysis focuses only on the number of conditional independence tests for constructing the equivalence class. Decomposition of graphs is a computationally simple task compared to the conditional independence tests. The triangulation of an undirected graph is used in our algorithms to construct a d-separation from an undirected independence graph. Although the problem for optimally triangulating an undirected graph is NP-hard, sub-optimal triangulation methods [2,13,21] may be used provided that the obtained tree does not contain too large nodes to test conditional independencies. Two most well-known algorithms are the lexicographic search [22] and the maximum cardinality search [25], and their complexities are $O(ne)$ and $O(n+e)$ respectively. Thus in our algorithms, the conditional independence tests dominate the algorithmic complexity.

Let V denote the set of all variables, and n the number of variables in V. In the IC algorithm, to delete an edge between a pair of variables u and v, we may need to test independencies of u and v conditionally on all possible subsets S of $V \setminus \{u, v\}$. Thus the complexity for deleting an edge in the global skeleton is $O(2^n)$, and then the complexity for constructing the global skeleton is $O(n^2 2^n)$.

In our decomposition algorithm, suppose that the set V of all variables is decomposed into H nodes $\{C_1, \ldots, C_H\}$ in the d-separation tree T, where $H \leqslant n$. Let m denote the number of variables in the largest node, that is, $m = \max_h |C_h|$ where $|C_h|$ denotes the number of variables in C_h. The complexity for constructing a local skeleton is $O(m^2 2^m)$, and thus that of all local skeletons is $O(Hm^2 2^m)$. Since m usually is much less than n, our algorithm is less computationally complex than the IC algorithm.

In the PC algorithm, to delete an edge between a pair of variables u and v, we may need to check all possible subsets of variables adjacent to u and v. The decomposition of a large graph into small subgraphs can also be used to improve the PC algorithm. The sets of variables adjacent to u and v can be split into small subsets by the decomposition. Thus to delete an edge between u and v, we only need to check each subset of variables adjacent to u or v that is contained in a subgraph with u or v. Let $Adj(u)$ denote the set of variables adjacent to u in the global skeleton, and $Adj_h(u)$ the set of variables adjacent to u in the hth local skeleton. Since $Adj_h(u) \subseteq Adj(u)$, the complexity for checking all subsets of $Adj_h(u)$ is less than that for checking all subsets of $Adj(u)$. Thus the decomposition can reduce the complexity of the PC algorithm.

7. Conclusions

We proposed a decomposition approach for structural learning of DAGs. In this approach, a problem of learning a large DAG is split into problems of learning small subgraphs. Domain or prior knowledge of conditional independencies can be utilized to facilitate the decomposition of structural learning. We theoretically proved the correctness of the proposed algorithms. Both the complexity of the algorithms and the power of conditional independence tests can be improved by decomposing a large graph into small subgraphs. The theoretical results can also be used for scheme

design of multiple databases. Although we assumed in this paper that there is no latent variables, some of our results may be applied to the case with latent variables. For example, suppose that we have some domain knowledge. Then we can apply our approach for local structural learning, which is discussed in an uncompleted paper.

Acknowledgements

We would like to thank the referees for their valuable comments and suggestions that improved the presentation of this paper. This research was supported by NSFC, MSAR and NBRP 2003CB715900.

Appendix A

A.1. Proofs of theorems

First we give a definition and several lemmas to be used in proofs of theorems.

Definition A.1. Let T be a d-separation tree for a *DAG* $\vec{G}_V$ with the node set $\mathcal{C} = \{C_1, \ldots, C_H\}$. For any two vertices u and v in $\vec{G}_V$, the distance between u and v in the tree 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 T. We call C_i and C_j minimizers for u and v which minimize the distance $d(C_i, C_j)$.

Lemma A.1. *Let l be a path from u to v, and W be the set of all vertices on l (W may or may not contain u and v). Suppose that a path l is d-separated by S. If W is contained in S, then the path l is d-separated by W and by any set containing W.*

Proof. Since the d-separation of the path l depends on which vertices between u and v are contained in the d-separator and W contains all vertices on l, l is also d-separated by $S \cap W = W$ if l is d-separated by S. Since all colliders on l have already been made active conditionally on W, adding other vertices into the conditional set does not make any new collider active on l. This implies that l remains to be d-separated by any set containing W. □

Lemma A.2. *Let T be a d-separation tree for a DAG $\vec{G}_V$, and K be a separator of T which separates T into two subtrees T_1 and T_2 with variable sets V_1 and V_2 respectively. Suppose that l is a path from u to v in $\vec{G}_V$ where $u \in V_1 \setminus K$ and $v \in V_2 \setminus K$. Let W denote the set of all vertices on l (W may or may not contain u and v). Then the path l is d-separated by $W \cap K$ and by any set containing $W \cap K$.*

Proof. Since $u \in V_1 \setminus K$ and $v \in V_2 \setminus K$, there is a series from s (may be u) to y (may be v) in $l = (u, \ldots, s, t, \ldots, x, y, \ldots, v)$ such that $s \in V_1 \setminus K$ and $y \in V_2 \setminus K$ and all vertices from t to x are contained in K. Let l' be the sub-path of l from s to y and W' the vertex set from t to x and thus $W' \subset K$. Since $s \in V_1 \setminus K$ and $y \in V_2 \setminus K$, we have from definition of d-separation tree that K d-separates s and y in $\vec{G}_V$, that is, K separates l'. By Lemma 1, we get that l' is d-separated by W' ($\subset K$) and by any set containing W'. Since $W' \subset (W \cap K)$, l' is d-separated by $W \cap K$ and by any set containing $W \cap K$. Thus l ($\supseteq l'$) is also d-separated by them. □

Lemma A.3. *Let u and v be two non-adjacent vertices in $\vec{G}_V$, and let l be a path from u to v. If l is not contained in $An(u) \cup An(v)$, then l is d-separated by any subset S of $an(u) \cup an(v)$.*

Proof. Since $l \not\subseteq An(u) \cup An(v)$, there is a series from s (may be u) to y (may be v) in $l = (u, \ldots, s, t, \ldots, x, y, \ldots, v)$ such that s and y are contained in $An(u) \cup An(v)$ and all vertices from t to x are out of $An(u) \cup An(v)$. Then the edges s–t and x–y must be oriented as $s \to t$ and $x \leftarrow y$, otherwise t or x belongs to $an(u) \cup an(v)$. Thus there exists at least one collider between s and y on l. The middle vertex w of the collider closest to s between s and y is not contained in $an(u) \cup an(v)$, and any descendant of w is not in $an(u) \cup an(v)$, otherwise there is a directed cycle. So l is d-separated by the collider, and it cannot be made active conditionally on any vertex in S where $S \subseteq an(u) \cup an(v)$. □

Lemma A.4. *Let T be a d-separation tree for a DAG $\vec{G}_V$ and C a node of T. Let u and v be two vertices in C which are non-adjacent in $\vec{G}_V$. If u and v are not contained in the same separator connecting C, then there exists a subset of C which d-separates u and v in $\vec{G}_V$.*

Proof. Define $S = (an(u) \cup an(v)) \cap C$. We show below that u and v are d-separated by S, that is, every path between u and v in $\vec{G}_V$ is d-separated by S.

If l is not contained in $An(u) \cup An(v)$, then we obtain from Lemma A.3 that l is d-separated by S.

When l is contained in $An(u) \cup An(v)$, let x and y be two vertices that are closest to u on l, that is, $l = (u, x, y, \ldots, v)$. The path l can have the following possible cases:

(1) $u \leftarrow x$ and $x \in C$,
(2) $u \rightarrow x \rightarrow y$ and $x \in C$,
(3) $u \rightarrow x \leftarrow y$ and both x and $y \in C$,
(4) $u \rightarrow x \leftarrow y$, $x \in C$ but $y \notin C$, and
(5) $x \notin C$.

For cases (1) and (2), since x is contained in C and u–x–y is not a collider, the path l is d-separated by x which is contained in S. Thus l is d-separated by S.

For case (3), we have from $x \in An(u) \cup An(v)$ that y must not be v, otherwise there exists a directed cycle in $\vec{G}_V$. Since y is not the middle vertex of a collider on l and $y \in C$, the path l is d-separated by y which is contained in S. Thus l is d-separated by S.

For case (4), let C' ($\neq C$) be the node of T which contains y. Since $y \in C'$ but $v \in C$, there is a sub-path l' from y to v that passes through a separator K connecting C toward C', which d-separates $C \setminus K$ from $C' \setminus K$. From $K \subseteq C$ and $l' \subseteq An(u) \cup An(v)$, we have that $[K \cap l'] \subseteq [C \cap (an(u) \cup an(v))] = S$. Since $y \notin C$ and vertices x and y are adjacent in $\vec{G}_V$, we have that $x \in K$. Notice that $u \rightarrow x \leftarrow y$ is a collider, thus we know $u \in K$ from the definition of the d-separation tree. By the assumption that u and v are not in the same separator connecting C, we have $v \in C \setminus K$. Since $y \in C' \setminus C \subset C' \setminus K$ and $v \in C \setminus K$, by Lemma A.2, we obtain that the sub-path l' is d-separated by $K \cap (l' \setminus \{y, v\})$ and by S. Thus l is d-separated by S.

For case (5), let C' ($\neq C$) be the node of T which contains x. Similar to case (4), we can show the result. □

Lemma A.5. *Let T be a d-separation tree for a DAG $\vec{G}_V$. For any vertex u there exists at least one node of T which contains u and $pa(u)$.*

Proof. If $pa(u)$ is empty, it is trivial. Otherwise let C denote the node of T which contains u and the most parents of u.

Since no set can separate u from a parent, there must be a node of T that contains u and the parent. If u has only one parent, then we obtained the lemma. If u has two or more parents, we suppose, by reduction to absurdity, that u has two parents v and w which are not contained in a single node but are contained in two different nodes of T, say $\{u, v\} \subseteq C$ and $\{u, w\} \subseteq C'$ respectively, since all vertices in V appear in T. On the path from C to C' in T, all separators must contain u, otherwise they cannot separate C from C'. However any separator containing u cannot separate v and w. Thus we got a contradiction. □

Lemma A.6. *Let T be a d-separation tree for a DAG $\vec{G}_V$ and C a node of T. Let u and v be two vertices in C which are non-adjacent in $\vec{G}_V$. If u and v are contained in the same separator S connecting C in T, then there exists a node C' of T containing u, v and a set S' such that S' d-separates u and v in $\vec{G}_V$.*

Proof. Without loss of generality, we can suppose that v is not a descendant of u in $\vec{G}_V$. By Lemma A.5, there is a node C_1 of T that contains u and $pa(u)$. Let C_2 contains both u and v, and the distance $d(C_1, C_2)$ is minimum.

If C_1 and C_2 are the same node of T, then S' defined as the parents of u separates u from v.

If C_1 and C_2 are different nodes, then $d(C_1, C_2) \geqslant 1$, and there is at least one parent p of u that is not contained in C_2. Thus there is a separator K connecting C_2 toward C_1 in T such that K d-separates p from all vertices in $C_2 \setminus K$. Since u and p are adjacent in $\vec{G}_V$ and the distance $d(C_1, C_2) \geqslant 1$ is minimum, we have $u \in K$ but $v \notin K$ (if

$v \in K$, then $d(C_1, C_2)$ is not minimum). For every parent p' of u that is contained in C_1 but not in C_2, we can show in the same way as p that K also separates p' from all vertices in $C_2 \setminus K$.

Define $S' = (An(u) \cup An(v)) \cap C_2$. Similar to proof of Lemma A.4, we show below that every path from u to v in $\vec{G}_V$ is d-separated by S', that is, u and v are d-separated by S'.

If a path l between u and v is not contained in $An(u) \cup An(v)$, then we obtain from Lemma A.3 that l is d-separated by S'.

When l is contained in $An(u) \cup An(v)$, let x be adjacent to u on l, that is, $l = (u, x, \ldots, v)$. Because l is contained in $An(u) \cup An(v)$ and v is not a descendant of u, the edge between u and x must be oriented as $u \leftarrow x$, that is, x is a parent of u.

If x is contained in C_2, then l is d-separated by x which is contained in S'.

If the parent x of u is not contained in C_2, as shown above, we have that x and v are d-separated by K. By Lemma A.2, we can obtain that the sub-path l' from x to v can be d-separated by $W \cap K$ where W denotes the set of all vertices between x and v (not containing x and v) on l'. Since $S' \supseteq (W \cap K)$, we get from Lemma A.2 that l' is also d-separated S'. Hence the path l is d-separated by S'. □

Lemma A.7. *Let T be a d-separation tree for a DAG $\vec{G}_V$. If two vertices u and v have distance $d(u, v) > 0$ in T where C_i and C_j are minimizers of u and v in T, then any separator on the path between C_i and C_j in T d-separates u and v in $\vec{G}_V$.*

Proof. For any separator S on the path between C_i and C_j in T, we have by definition of $d(u, v)$ that neither u nor v can be contained in S, otherwise C_i and C_j are not minimizers of u and v. Since C_i and C_j belong to different subtrees obtained by removing the edge with the separator S attached, we know that u and v are d-separated by S. □

Proof of Theorem 1. Suppose that u and v are two non-adjacent vertices in $\vec{G}_V$. If $d(u, v) > 0$ in T, then we get from Lemma A.7 that u and v are non-adjacent in $\vec{G}_V$. If $d(u, v) = 0$, then we have from Lemmas A.4 and A.6 that there exists a node C of T that contains u, v and a subset S such that S d-separates u and v in $\vec{G}_V$. □

Proof of Theorem 2. From [5, p. 53] , we have that any separator S in the junction tree T separates $V_1 \setminus S$ and $V_2 \setminus S$ in the triangulated graph $\bar{G}^t_V$, where V_i denotes the variable set of the subtree T_i induced by removing the edge with a separator S attached, for $i = 1$ and 2. Since the edge set of $\bar{G}^t_V$ contains that of $\bar{G}_V$, $V_1 \setminus S$ and $V_2 \setminus S$ are also separated in $\bar{G}_V$. Further, since $\bar{G}_V$ is an undirected independence graph for $\vec{G}_V$, we obtain immediately that T is a d-separation tree for $\vec{G}_V$. □

Proof of Theorem 3. Since a moral graph for a DAG $\vec{G}_V$ is an undirected independence graph, by definition of an undirected independence graph, we only need to show that $\bar{G}$ defined in Theorem 3 contains all edges of $\vec{G}^m_V$. It is obvious that E contains all edges obtained by dropping directions of directed edges in $\vec{G}_V$ since any set cannot d-separate two vertices that are adjacent in $\vec{G}_V$.

Now we show that E also contains any moral edge that connects vertices u and v having a common child w, that is, $(u, v) \in E$. We know from Lemma A.5 that there is at least one node C_h in a d-separation tree T that contains u, v and w. The undirected subgraph $\bar{G}_h$ for C_h must contain edges u–w and v–w, that is, $\bar{G}_h$ has an undirected path u–w–v. By definition of an undirected independence graph, w must be contained in a separator to separate u from v in this undirected subgraph, but u and v cannot be d-separated conditional on any set containing w since there is a collider $u \rightarrow w \leftarrow v$. Thus $\bar{G}_h$ contains the moral edge (u, v). □

A.2. Proofs of algorithms' correctness

Proof of Algorithm 1's correctness. By the sufficiency of Theorem 1(1), the initializations at steps 2 and 3 for creating edges guarantee that no edge is created between any two variables which are not in the same node of the d-separation tree. By the sufficiency of Theorem 1(2), deleting edges at steps 2 and 3 guarantees that any other edge between two d-separated variables can be deleted in some local skeleton. Thus the global skeleton obtained at step 3 is correct. According to the necessity of Theorem 1, we have that each moral edge (u, v) in the undirected independence

graph must be deleted at some subgraph over a node of the d-separation tree since a separator of a d-separation tree cannot separate u and v of a v-structure $u \to w \leftarrow v$. Thus we can determine all v-structures at step 4, which completes our proof. □

Proof of Algorithm 2's correctness. Each complete undirected subgraph $\bar{G}_h$ describes a saturated model over C_h and the entire graph $\bar{G}_V$ obtained at step 3 represents all dependencies from the prior knowledge. The triangulation at step 4 does not bring any new conditional independence. Thus the junction tree is a d-separation tree. □

Proof of Main Algorithm's correctness. The correctness of Algorithms 1 and 2 has been proved above. Thus we only need to prove that step 3 is correct for obtaining a d-separation tree. According to Theorem 3, we have that the entire undirected independence graph is constructed correctly at the step 3. Then the d-separation tree can be obtained correctly by using an algorithm for constructing a junction tree. □

References

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

[2] A. Becker, D. Geiger, A sufficiently fast algorithm for finding close to optimal clique trees, Artificial Intelligence 125 (2001) 3–17.

[3] C. Beeri, R. Fagin, D. Maier, M. Yannakakis, On the desirability of acyclic database schemes, J. ACM 30 (1983) 479–513.

[4] C. Berge, Graphs and Hypergraphs, second ed., North-Holland, Amsterdam, 1976.

[5] R.G. Cowell, A.P. David, S.L. Lauritzen, D.J. Spiegelhalter, Probabilistic Networks and Expert Systems, Springer Publications, New York, 1999.

[6] D.R. Cox, N. Wermuth, Multivariate Dependencies: Models, Analysis, and Interpretation, Chapman and Hall, London, 1993.

[7] D. Danks, Learning the causal structures of overlapping variable sets, in: Proceedings of the 5th International Conference on Discovery Science, Springer-Verlag, Berlin, 2002, pp. 178–191.

[8] D. Edwards, Introduction to Graphical Modelling, Springer-Verlag, New York, 1995.

[9] R. Fung, S. Crawford, CONSTRUCTOR—a system for the induction of probabilistic models, in: Proceedings of the Eighth National Conference on AI, American Association for Artificial Intelligence, Boston, 1990, pp. 762–779.

[10] Z. Geng, Decomposability and collapsibility for log-linear models, Appl. Stat. 38 (1) (1989) 189–197.

[11] Z. Geng, C. Wang, Q. Zhao, Decomposition of search for v-structures in DAGs, J. Multivariate Anal. (2005), submitted for publication.

[12] D. Heckerman, A tutorial on learning with Bayesian networks, in: M.I. Jordan (Ed.), Learning in Graphical Models, Kluwer Academic, Dordrecht, 1998, pp. 301–354.

[13] F.V. Jensen, F. Jensen, Optimal junction trees, in: Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, San Fransisco, CA, 1994, pp. 360–366.

[14] D. Koller, M. Sahami, Toward optimal feature selection, in: Proceedings of the 13th International Conference on Machine Learning, Bari, Italy (1996) pp. 284–292.

[15] S.L. Lauritzen, Graphical Models, Clarendon Press, Oxford, 1996.

[16] R.J.A. Little, D.B. Rubin, Statistical Analysis with Missing Data, second ed., Wiley, New York, 2002.

[17] C. Meek, Causal inference and causal explanation with background knowledge, in: Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, San Francisco, CA, 1995, pp. 403–410.

[18] J. Pearl, Probabilistic Reasoning in Intelligent Systems, Morgan Kaufmann, San Mateo, CA, 1988.

[19] J. Pearl, Causality, Cambridge University Press, Cambridge, 2000.

[20] S. Rassler, Statistical Matching, Lecture Notes in Statistics, vol. 168, Springer, New York, 2002.

[21] N. Roberston, P.D. Seymour, Graph minors XIII. The disjoint paths problems, J. Combin. Theory B 53 (1995) 65–100.

[22] D. Rose, R. Tarjan, G. Lueker, Algorithmic aspects of vertex elimination on graphs, SIAM J. Comput. 5 (1976) 266–283.

[23] P. Spirtes, C. Glymour, An algorithm for fast recovery of sparse causal graphs, Social Sci. Comput. Rev. 9 (1991) 62–72.

[24] P. Spirtes, C. Glymour, R. Scheines, Causation, Prediction and Search, second ed., MIT Press, Cambridge, MA, 2000.

[25] R.E. Tarjan, M. Yannakakis, Simple linear-time algorithm to test chordality of graphs, test acyclicity of hypergraphs, and selective reduce acyclic hypergraphs, SIAM J. Comput. 13 (1984) 566–579.

[26] I. Tsamardinos, C. Aliferis, A. Statnikov, Time and sample efficient discovery of Markov blankets and direct causal relations, in: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, 2003, pp. 673–678.

[27] T. Verma, J. Pearl, Equivalence and synthesis of causal models, in: Proceedings of the 6th Conference on Uncertainty in Artificial Intelligence, Elsevier, Amsterdam, 1990, pp. 255–268.