

# Learning Bayesian Networks based on Order Graph with Ancestral Constraints

Zidong Wang\*, Xiaoguang Gao, Xiangyuan Tan

*School of Electronics and Information, Northwestern Polytechnical University, Xi'an, China*

Yu Yang

*China Electronics Technology Group Corp 10th Research Institute , Chengdu, China*

Daqing Chen

*School of Engineering, London South Bank University, London, UK*

---

## Abstract

We consider incorporating ancestral constraints into structure learning for Bayesian Networks (BNs) when executing an exact search based on order graph; this is thought to be impossible because ancestral constraints are non-decomposable. In order to adapt to the constraints, the node in an Order Graph (OG) is generalized as a series of directed acyclic graphs (DAGs). Then, we design a novel revenue function to breed out infeasible and suboptimal nodes to expedite the graph search. A breadth-first search algorithm is implemented in the new search space, verifying the validity and efficiency of the proposed framework. It has been demonstrated that, when the ancestral constraints are consistent with the ground-truth network or deviate from it, the new framework can navigate a path that leads to a global optimization in almost all cases with less time and space required for orders of magnitude than the state-of-the-art framework, such as EC-Tree.

*Keywords:* Bayesian Network, Structure Learning, Ancestral Constraints

---

## 1. Introduction

As a type of graphic model, Bayesian Networks (BNs) are powerful tools for solving uncertainty in various applications, such as classification, causal discovery, and intelligent decision-making[1, 2, 3, 4]. A BN is composed of  
5 a structure and parameters, where the structure is the basis of the model. It

---

\*Corresponding author

*Email addresses:* `nwpu_wzd@mail.nwpu.edu.cn` (Zidong Wang), `cxg2012@nwpu.edu.cn` (Xiaoguang Gao), `tanxy2017@mail.nwpu.edu.cn` (Xiangyuan Tan), `youngiv@126.com` (Yu Yang), `chend@lsbu.ac.uk` (Daqing Chen)

is necessary to identify the structure of a BN to use it for modeling a system. However, it is tough to build a BN model purely based on the experience and domain knowledge of human-beings/experts; thus, the structure is typically modeled from training data[5].

10 In this paper, we consider the task of incorporating expert knowledge when learning the BN structure from the training data. The prior knowledge of an expert can be extracted and generalized as beliefs of a causal relationship among variables; hence the knowledge contains topological ordering constraints and structural constraints. In general, structure constraints fall into ancestral  
15 constraints and edge constraints according to whether they are decomposable [6]. Li proposed a constraint-based hill-climbing approach to incorporate all these constraints[6]. Cussens considered integer linear programming(ILP) as constrained optimization and treated all constraints as cutting planes [7]. Parvainen analyzed the existence of ancestor relations in the order space [8]. Chen  
20 claimed that the Markov equivalence is not satisfied because the corresponding sets of consistent directed acyclic graphs (DAGs) are overlapping [9]. Therefore, he proposed a new search space: the Bayesian network graph (BNG), a space of DAGs, for learning structures with non-decomposable scores [10]. To process ancestral constraints, Chen not only projected them using specific edge constraints  
25 but also implemented them through a tree of equivalent class(EC-Tree) [11, 12]. They demonstrated that ILP requires orders-of-magnitude computational time than their methods.

In this paper, we intend to fill in the research gap that utilizing ancestral constraints is infeasible in the decomposable implicit state-space search graphs,  
30 such as order graph(OG)[13]. As the K2 algorithm [14] and the approaches based on the OG (dynamic programming and heuristic search) cannot enforce ancestral constraints through pruning specific nodes, the global optimum solution theoretically only exists in a structural space, such as BNG and EC-Tree. Unfortunately, the extreme complexity of the frameworks mentioned above re-  
35 stricts the scalability of the algorithm (no more than 20 variables, as mentioned in the paper). Hence, it is necessary to study how to impose ancestral constraints into OG, a more inclusive and effective space when learning BN structure. It is easy to impose ordering constraints into OG [15]. However, incorporating structural constraints is tough since the order cannot convert to the parent or  
40 ancestor relations. Although some researchers have suggested utilizing edge constraints[16], ancestral constraints are still difficult to process using decomposable scores due to their non-decomposable nature. Furthermore, the exact search based on OG quickly gets stuck in a local optimum if ancestral constraints are incorporated as the technique in handling edge constraints. So we prepare  
45 to extend the scope of OG to tackle this challenge.

The contributions of the study are: 1) We propose and develop a new search space, named ancestral constrained order graph (ACOG). Such a framework combines the advantages of OG and BNG. When conducting a candidate node, all the suboptimal structures are reserved. When expanding a sink, only the  
50 structure with the best parents is encoded. 2) ACOG does not require to decompose ancestral constraints into edge constraints or any other constraints.

Furthermore, the only rule to follow is that a candidate node should conform to all the relevant constraints when adding a new sink. 3) We introduce the methods for eliminating violated and suboptimal nodes in ACOG when ancestral constraints are incorporated. The efficiency of the exact search can be highly improved, and the learned result can escape from the local optimum solution as much as possible based on pruned ACOG.

We empirically evaluate the effectiveness of the proposed framework through a breadth-first search strategy. Furthermore, when the ancestral constraints are consistent with the ground-truth network or deviate from it, the new framework can navigate a path that leads to a global optimization in almost all cases. Moreover, the proposed framework can effectively reduce the space and time complexity of learning BN structure with ancestral constraints. To verify the robustness of the proposed framework, we conduct a comparative test when there are minor errors and fatal errors in prior knowledge.

This paper is organized as follows: In Section 2, we review the relevant works on Bayesian Network structure learning and order graph. In Section 3, we propose the basic structure of the ACOG. In Section 4, we first discuss the relevant concepts of violated nodes and introduce the regulations for pruning them based on ancestral constraints. Then, we theoretically analyze how to discard the suboptimal DAGs by a novel revenue function based on ancestral constraints. We also introduce a practical example to illustrate it. In Section 5, we present the experiments to evaluate the proposed algorithms. Section 6 concludes the paper.

## 2. Preliminaries

Structure learning for Bayesian Networks has been proved to be NP-hard [17]; specifically, it has been formalized as a highly non-convex optimization problem in search space. There are two general approaches for learning BN structure: approximate methods [18, 19, 20, 21] and exact methods. In recent years, the exact approaches have attracted considerable research attention. An exact approach attempts to separate the learning process into two phases: parent set identification and structure optimization [22]. The first phase's purpose is to determine all the feasible candidate parent sets and their scores for each variable. Most of the structure optimization methods assign a parent set to each variable, maximizing the score of the observed structure while avoiding cycles. There are numerous efficient algorithms for the second phase, such as dynamic programming (DP) [23, 24], linear and integer programming (ILP) [25, 7], and shortest-path heuristic [26, 27].

### 2.1. Parent Set Identification

The structure  $G$  of BN is a directly acyclic graph (DAG), which consists of random variables  $\mathbf{V} = \{X_1, \dots, X_n\}$  and arcs to their parent set. When learning the optimal BN structure from a dataset  $\mathbf{D}$ , the candidate parent sets for every variable and the corresponding score of them assess how well the model

fits the given data. As such, finding the best structure can be considered as a combinatorial optimization problem. The common criteria, such as BIC [28], CH[29] and BDeu [30] *et. al.*, are all decomposable, meaning that the DAG score is the sum of its local scores for every variable.

$$\text{Score}(G|\mathbf{D}) = \sum_{i=1}^n \text{Score}(\langle X_i, Pa(X_i) \rangle | \mathbf{D})$$

90 where  $Pa(X_i)$  represents a parent set of  $X_i$ . Usually, the score of the parent set is computed in sequential order, and the calculation complexity depends on the maximum in-degree, which is limited by the maximum size of parents.

In order to explain the underlying theory of the proposed approach more clearly, we adopt a specific mark of parent sets here. In a structure  $G(\mathbf{V})$ , 95 where  $\mathbf{V}$  is a set of variables in the graph, for any  $X$  in  $\mathbf{V}$ ,  $Pa(X|G)$  are the parents of  $X$ , and  $D(X|G)$  are the descendants of  $X$ . The constraint sets are defined as follows.

**Definition 1. (Edge constraints):**  $\mathcal{E} : \mathbf{V} \rightarrow P(\mathbf{V})$ , where  $P(\mathbf{V})$  is the power set of  $\mathbf{V}$ . For every  $X \in \mathbf{V}$ ,  $\mathcal{E}(X) \in P(\mathbf{V})$  is the required parent set of  $X$ , 100 denoted as  $\mathcal{E}(X) \rightarrow X$ .

**Definition 2. (Ancestral constraints):**  $\mathcal{A} : \mathbf{V} \rightarrow P(\mathbf{V})$ , where  $P(\mathbf{V})$  is the power set of  $\mathbf{V}$ . For every  $X \in \mathbf{V}$ ,  $\mathcal{A}(X) \in P(\mathbf{V})$  is the required ancestor set of  $X$ , denoted as  $\mathcal{A}(X) \rightsquigarrow X$ .  $\mathcal{D} : \mathbf{V} \rightarrow P(\mathbf{V})$ , where  $P(\mathbf{V})$  is the power set of  $\mathbf{V}$ . For every  $X \in \mathbf{V}$ ,  $\mathcal{D}(X) \in P(\mathbf{V})$  is the required descendent set of  $X$ .

## 105 2.2. Structure Optimization

For the sake of completeness, we briefly introduce OG and the exact search strategies below. OG is a Hasse diagram that contains all the subset of variables. The node in OG represents an ordering with the highest score, and the arc implies the lowest cost when adding a sink. Figure 1 shows an OG for the three 110 variables.

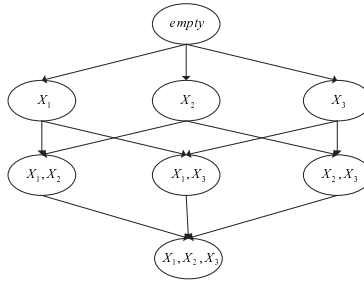


Figure 1: OG for three variables.

In the step of the structure optimization, search strategies in OG, such as breadth-first search(DP) [23] and heuristic search(A\*) [27], could find a global

optimum Bayesian Network easily. DP solves the structure learning problem by dividing it into sub-problems based on score decomposability. Let  $\mathbf{U}$  be the domain variables. Assume that the variable  $X$  is a sink in the optimal structure:

$$\text{Score}(\mathbf{U}) = \max_{X \in \mathbf{U}} \{\text{Score}(\mathbf{V} \setminus X) + \text{BestScore}(X, \mathbf{U} \setminus X)\}$$

where

$$\text{BestScore}(X, \mathbf{U} \setminus X) = \max_{Pa(X) \subset \mathbf{U} \setminus X} \text{Score}(X, Pa(X))$$

the remaining variables  $\mathbf{U} \setminus X$  must form an optimal subnetwork, and  $X$  can find its best parents set from  $\mathbf{U} \setminus X$ . Therefore, by comparing the cases in which every variable in  $\mathbf{U}$  is a sink, the optimal structure can be obtained. With the guidance of the relation, the entire learning process can be divided into phases and starts with an empty network. In each phase, the algorithm adds a sink to every subnetwork obtained in the previous phase and generates more complex subnetworks. This process continues recursively until the complete network is observed.

A\* is a heuristic search strategy to find a shortest-path in OG. For every node  $\mathbf{U}$  in a graph, evaluation function  $f(\mathbf{U})$  is applied to measure its quality, and node with lowest  $f(\mathbf{U})$  is expanded during the exploration of OG. So the entire search is guided towards the minimum  $f(\mathbf{U})$ .  $f(\mathbf{U})$  is defined as follows:

$$f(\mathbf{U}) = g(\mathbf{U}) + h(\mathbf{U})$$

where  $g(\mathbf{U})$  is the past cost from the initial node to the current node, and  $h(\mathbf{U})$  is the estimate cost from the current node to the final node. In BN structure learning, the score of expanding a sink represent the path cost in each step. Thus,  $g(\mathbf{U})$  and  $h(\mathbf{U})$  can be fomulated as follows:

$$g(\mathbf{U}) = g(\mathbf{U} \setminus X) + \text{BestScore}(X, \mathbf{U} \setminus X)$$

$$h(\mathbf{U}) = \sum_{X \in \mathbf{V} \setminus \mathbf{U}} \text{BestScore}(X, \mathbf{V} \setminus X)$$

Because of the heuristic function's consistency, the search would converge to the global optimum solution with the minimum path cost. Furthermore, A\* is proven to be a more efficient algorithm [27].

The rest of the paper focuses on resolving the problem of incorporating ancestral constraints into OG through exact search strategies.

### 3. Ancestral Constrained Order Graph

Structure constraints are usually projected as edge constraints and ancestral constraints. The principle of incorporating edge constraints is as follows [16]. We assume that there is an edge constraint set  $\mathcal{E}$ . Then, two rules should be obeyed to check for the nodes in OG.

Table 1: Local scores of variables

$(X_i, Pa(X_i))$	SCORE	$(X_i, Pa(X_i))$	SCORE
$(X, empty)$	-11	$(Y, \{X\})$	-10
$(X, \{Y\})$	-9	$(Z, \{X\})$	-8
$(Y, empty)$	-11	$(Z, \{Y\})$	-1

Table 2: Order graph of variables

LAYER	NODE	SCORE	STRUCTURE
LAYER1	<i>empty</i>	-INF	NULL
LAYER2	$X$	-11	$X$
LAYER2	$Y$	-11	$Y$
LAYER2	$Z$	ILLEGAL	NULL
LAYER3	$\{X, Y\}$	-20	$Y \rightarrow X$
LAYER3	$\{X, Z\}$	-19	$X \rightarrow Z$
LAYER3	$\{Y, Z\}$	ILLEGAL	NULL
LAYER4	$\{X, Y, Z\}$	-28	$Y \rightarrow X \rightarrow Z$

1. Let  $\mathbf{U}$  be the set of variables at a node of the OG. If for some  $X \in \mathbf{U}$ , we have  $\mathcal{E}(X) \not\subseteq \mathbf{U}$ , then the node should be pruned.
2. When a variable  $X$  is added to a node, the best parent set of  $X$  should contain  $\mathcal{E}(X)$ .

These rules ensure that the constraints are consistent with nodes in OG; thus, the structure in the final layer maximizes the score. With the guidance of Rule 1, several violated nodes are discarded, and therefore the time and space cost is effectively reduced. For example, if the number of edge constraints is  $m$ , the space complexity can be reduced from  $O(C_n^{\frac{n}{2}})$  to  $O(mC_{n-m}^{\frac{n}{2}-1})$ . Rule 2 restricts the candidate parent set, whose time complexity is  $O(1)$ . Therefore, OG can be efficiently simplified under edge constraints. However, the approach is not suitable for optimization with ancestral constraints. For example, Table 1 shows the variables with their local scores; assume that there is an ancestral constraint  $X \rightsquigarrow Z$ .

All the other scores that are not listed are -Inf. According to the above rules, Table 2 shows the process of searching the optimal structure.  $Y \rightarrow X \rightarrow Z$  is the optimal structure under the constraints. However, there is another structure  $X \rightarrow Y \rightarrow Z$ , with a higher score of  $-22$  that satisfies the constraints. The cause of the mistake is that the structure  $X \rightarrow Y$  is discarded at node  $\{X, Y\}$ , which is permitted in the exact search. When incorporating ancestral constraints, it is essential to solving the conflict between decomposable scores and non-decomposable constraints [12]. Therefore, it is not appropriate to expand the subnetworks to a global optimum BN structure by adding sinks. The BNG and

EC-Tree [10, 12] can feasibly address this problem. However, the number of nodes in the frameworks mentioned above is vast, which could pose a restriction on the scalability when increasing the size of variables in the learning problem. Therefore, we introduce a new search graph called Ancestral constrained Order Graph (ACOG), that stores reasonable DAGs at its nodes. Figure 2 shows a simple ACOG with three variables.

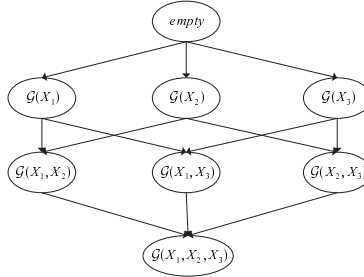


Figure 2: ACOG for three variables

A node  $\mathcal{G}(\mathbf{U})$  in the graph is formulated as

$$\begin{aligned}
 \mathcal{G}(\mathbf{U}) &= \bigcup_{\forall X \in \mathbf{U}, \mathbf{T} = \mathbf{U} \setminus X} \mathcal{G}(X|\mathbf{T}) \\
 \mathcal{G}(X|\mathbf{T}) &= \bigcup_{\forall G(\mathbf{T}) \in \mathcal{G}(\mathbf{T})} \text{Best}(X, G(\mathbf{T})) \\
 \text{Best}(X, G(\mathbf{T})) &= G(\mathbf{T}) \bigcup_{Pa(X) \subseteq \mathbf{T}} G(X, \arg \max (\text{Score}(X, Pa(X))))
 \end{aligned} \tag{1}$$

$\text{Best}(X, G)$  denotes the best structure based on  $G$  with a sink  $X$ .  $G(\mathbf{T})$  is a DAG of  $\mathbf{T}$ , and  $G(X, Pa(X))$  is a DAG for  $X$  with the parent set  $Pa(X)$ . Note that if a sink is a constrained descendant variable, the best parent set must conclude at least one of the specific ancestor and its descendants. Otherwise, the optimal parent set will be taken in the same way as OG. It is easy to conclude some features of ACOG:

1. The scope of ACOG is higher than OG. OG's node encodes the best DAG (or its equivalent structure) of the current variables, whereas ACOG's node encodes a series of DAGs that consist of all the subsets with a corresponding sink. ACOG reserves many partial suboptimal structures that potentially compose the final optimal DAG due to ancestral constraints. Thus, the proposed framework is more likely to lead to global optimization.
2. The scope of ACOG is lower than EC-Tree. When adding a sink to a DAG, only the best parent set is considered instead of all parent sets. Theoretically, the exact search is difficult to reach a global optimum solution based on ACOG because the framework is simplified. However, it is usually inconsiderable to reserve too many suboptimal parent sets, which cost massive

memory, especially when the ancestral constraints are consistent with the ground-truth network.

The nodes in the ACOG only store the best structures of the related nodes in the previous layer. However, it is still computationally inefficient and time-consuming to search in the ACOG due in no small number of DAGs in the previous layer. In fact there are numerous violated and suboptimal structures in the ACOG with ancestral constraints. Moreover, we would discuss the condition in detail in the next section.

#### 4. Simplifying the ACOG under ancestral constraints

ACOG can be efficiently simplified under the ancestral constraints by discarding the violated and suboptimal structures. We detect the pruning rules next.

##### 4.1. Pruning violated structures in ACOG

The purpose of introducing the ACOG is to incorporate ancestral constraints; thus, the necessary precondition for doing so is that DAGs at the nodes of the ACOG should be consistent with the constraints. Some nodes that violate the constraints need to be discarded. The definition of violation is as follows. Let  $\mathcal{G}(\mathbf{U})$  be a node in an ACOG, and  $\mathbf{U}$  be the set of domain variables in  $\mathcal{G}(\mathbf{U})$ .

**Definition 3. (Violated node):** If for every  $X \in \mathbf{U}$ , we have  $\mathcal{A}(X) \subseteq \mathbf{U}$ , then the node  $\mathcal{G}(\mathbf{U})$  conforms to the constraints; otherwise  $\mathcal{G}(\mathbf{U})$  is a violated node.

**Definition 4. (Violated DAG):** If in a DAG  $G$ , for every  $X \in \mathbf{U}$ ,  $Pa(X|G)$  satisfies the conditions :

i)  $Pa(X|G) \subseteq \mathbf{U}$ ;

ii) for every  $Y \in \mathcal{A}(X)$ ,  $Pa(X|G) \cap D(Y|G) \neq \emptyset$ ;

then  $G$  conform to the constraints; otherwise  $G$  is a violated DAG.

Definitions 3 and 4 guide the pruning violated structures. If a node dissatisfies the ancestral constraints or contains violated partial DAGs, these structures should be pruned from ACOG. Although the candidate DAGs in ACOG are fewer than those in EC-Tree apparently, the size of nodes is still excessively large compared to that in OG. For example, if  $\mathbf{U} = \{X_1, X_2, \dots, X_{12}\}$  and  $\mathcal{A}(X_2) = X_1$ , there are several DAGs at initial layer nodes. However, the number of DAGs at node  $\mathcal{G}(\mathbf{U} \setminus X_1)$  is enormous since it accumulates through all the structures from the entire ACOG except  $\mathcal{G}(\mathbf{U})$ . In order to resolve this issue, a novel method is proposed for pruning suboptimal structures in ACOG so that space complexity could be further reduced.



## 4.2. Pruning suboptimal structures in ACOG

After the violated nodes and DAGs have been discarded, most of the DAGs  
 215 at the remaining nodes are still suboptimal. When lacking the expert knowledge,  
 the DAG with the highest score at each node should be reserved according to  
 the principle of the exact strategies. All the DAGs at every node should be  
 encoded in theory under ancestral constraints, but most suboptimal structures  
 can still be pruned based on the following theorems. For consistency, let  $\mathbf{U}$   
 220 denote the set of variables of  $\mathcal{G}(\mathbf{U})$ . In the following discussion, we assume that  
 all the structures in ACOG conform to the constraints.

### 4.2.1. Principle for elimination

Consider a special condition:

**Theorem 1.** *If for all  $X \in \mathbf{U}$ ,  $\mathcal{D}(X) \subseteq \mathbf{U}$  is true, then all the DAGs in  $\mathcal{G}(\mathbf{U})$   
 225 can be pruned except those DAGs with the highest score.*

PROOF. Suppose that  $\mathcal{G}(\mathbf{U}_1)$  satisfies the condition being a node in the ACOG,  
 and  $\mathcal{G}(\mathbf{U}_2)$  is a descendant of  $\mathcal{G}(\mathbf{U}_1)$ . Consider a sink variable  $X$  is added to  
 $\mathcal{G}(\mathbf{U}_2)$ : If  $\mathcal{A}(X) = \emptyset$ , for all  $G \in \mathcal{G}(\mathbf{U}_1)$ , the best parents of  $X$  in  $G$  are  
 constant because the candidate parent set of  $X$  is always  $\mathbf{U}_1 \cap \mathbf{U}_2$ . If  $\mathcal{A}(X) \neq \emptyset$ ,  
 230 then  $\mathcal{A}(X) \subset \mathbf{U}_2/\mathbf{U}_1$  holds. If the above conclusion is false, then  $\mathcal{A}(X) \subset \mathbf{U}_1$   
 and  $\mathcal{D}(\mathcal{A}(X)) \subseteq \mathbf{U}_1$ ; thus, we deduce  $X \in \mathbf{U}_1$ , which violates the assumption.  
 Therefore, for all  $G \in \mathcal{G}(\mathbf{U}_1)$ , the best parents of  $X$  in  $G$  are constant because  
 the required parents are from  $\mathbf{U}_2 \setminus \mathbf{U}_1$ . The differences between DAGs in  $\mathcal{G}(\mathbf{U}_1)$   
 are useless in choosing the optimal structures, and thus only DAGs with the  
 235 highest score should be reserved.  $\square$

Virtually, the property that ancestors with requires descendants are present  
 in the same set is called ‘all-satisfied’ node. Such a node is tractable. How-  
 ever, most nodes of ACOG which waste too much vague memory are ‘partially-  
 satisfied’. For a ‘partially-satisfied’ node  $\mathcal{G}(\mathbf{U})$ , it is necessary to construct  
 240 the hypothetic ‘all-satisfied’ structure for pruning according to theorem1. We  
 now introduce  $IF(\mathbf{U})$ , which is an unstable factor of node  $\mathcal{G}(\mathbf{U})$  such that  
 $IF(\mathbf{U}) = \mathcal{D}(\mathbf{U}) \setminus (\mathcal{D}(\mathbf{U}) \cap \mathbf{U})$ .  $IF(\mathbf{U})$  consider some required but absent descen-  
 dants in advance. It is no longer suitable to reserve the DAG with the highest  
 score in new structures obviously; thus, another problem is how to prune the  
 245 useless DAGs at nodes. We now introduce a method of discarding suboptimal  
 structures with invalid constraints.

**Definition 5. (Invalid constraints):** For  $X \in IF(\mathbf{U})$  and all  $Y \in \mathcal{A}(X) \cap \mathbf{U}$ ,  
 the revenue function  $\mathcal{F}_G(X, Y)$  for a certain  $G$  of  $\mathcal{G}(\mathbf{U})$  is defined as

$$\begin{aligned} \mathcal{F}_G(X, Y) &= \max_{Pa_1, s, t, \forall \alpha \in Pa_1, \alpha \in \phi_1(Y, G)} \text{Score}(X, Pa_1) - \\ &\quad \max_{Pa_2, s, t, \exists \alpha \in Pa_2, \alpha \in \phi_2(Y, G)} \text{Score}(X, Pa_2) \\ \phi_1(Y, G) &= \mathbf{U} \setminus \{Y \cup D(Y|G)\} \\ \phi_2(Y, G) &= Y \cup D(Y|G) \end{aligned} \tag{2}$$

If  $\mathcal{F}_G(X, Y) < 0$ , the constraint  $Y \rightsquigarrow X$  is invalid in  $G$ . If  $\mathcal{F}_G(X, Y) \geq 0$ , the constraint is valid.

250 Definition 5 provides a useful guidance to the pruning process of suboptimal structures. Using the revenue function  $\mathcal{F}_G(X, Y)$ , a set of certain rules is discussed below regarding the elimination of suboptimal structures in the ACOG.

**Theorem 2.** Assume that  $G_1$  and  $G_2$  are two DAGs in  $\mathcal{G}(\mathbf{U})$  satisfying  $\text{Score}(G_1) > \text{Score}(G_2)$ , and  $IF(\mathbf{U}) = Y, \mathcal{A}(Y) = \{X\}$  is the single ancestral constraint in  $\mathbf{U}$ .

- 255 i). If the constraint  $X \rightsquigarrow Y$  in  $G_1$  and  $G_2$  is invalid, the DAGs with the highest score should be retained.  
 ii). If the constraint  $X \rightsquigarrow Y$  in  $G_1$  and  $G_2$  is valid and  $D(X|G_2) \subseteq D(X|G_1)$ , then  $G_2$  can be pruned.

PROOF. We construct a ‘‘all-satisfied’’ structure for  $\mathcal{G}(\mathbf{U})$  and examine the value of  $\mathcal{F}_G(X, Y)$ .

260 ii). is obviously true. If  $Y \in \mathbf{U}$ , we only retain the best structure in  $\mathcal{G}(\mathbf{U})$  according to Theorem 1. If  $Y \notin \mathbf{U}$  and  $D(X|G_2) \subseteq D(X|G_1)$ , the candidate constraint parent variables for  $X$  in  $G_1$  are always more than those in  $G_2$ . If  $\text{Score}(G_1) > \text{Score}(G_2)$ , conclusion holds in all subsequent nodes with respect to  $\mathbf{U}$  in the ACOG. The pruning rule is suitable for both  $\mathcal{F}(X, Y, G) > 0$  and  $\mathcal{F}(X, Y, G) < 0$ .

265 i). is proved as follows. Generally, if the best parent set is  $P \cup Q (P \notin \mathbf{U}, Q \in X \cup D(X|G))$ . regardless of whether  $P \in \mathcal{A}(Y)$ , the discussion should be on node  $\mathcal{G}(\mathbf{U} \cup P)$ . Therefore, we are only concerned with the case in which  $Q$  is the best parent. We construct the ‘‘all-satisfied’’  $\mathcal{G}(\mathbf{U} \cup Y)$ , where  $Y$  is a sink. The best parent set of  $Y$  must contain a variable  $\alpha \in X \cup D(X|G_i) (i = 1, 2)$ . When considering the revenue function, the formula (3) holds:

$$\begin{aligned} \max_{Pa \subset \mathbf{U}} \text{Score}(Y, Pa) &= \max_{Pa_1, s, t, \forall \alpha \in Pa_1, \alpha \in \phi_1(X, G)} \text{Score}(Y, Pa_1), \\ &\quad \max_{Pa_2, s, t, \exists \alpha \in Pa_2, \alpha \in \phi_2(X, G)} \text{Score}(Y, Pa_2) \end{aligned} \quad (3)$$

If  $\mathcal{F}_G(X, Y) < 0$  in both  $G_1$  and  $G_2$ , then:

$$\begin{aligned} \max_{Pa \subset \mathbf{U}} \text{Score}(Y, Pa) &= \max_{Pa_2, s, t, \exists \alpha \in Pa_2, \alpha \in \phi_2(X, G_1)} \text{Score}(Y, Pa_2) \\ &= \max_{Pa_2, s, t, \exists \alpha \in Pa_2, \alpha \in \phi_2(X, G_2)} \text{Score}(Y, Pa_2) \end{aligned} \quad (4)$$

275 For all the subsequent nodes  $\mathcal{G}(\mathbf{U} \cup \mathbf{S})$  satisfying  $Y \in \mathbf{S}$ , the difference between arbitrary two DAGs  $G_1(\mathbf{U} \cup \mathbf{S})$  and  $G_2(\mathbf{U} \cup \mathbf{S})$  in  $\mathcal{G}(\mathbf{U} \cup \mathbf{S})$  is the best parent set restricted by the ancestral constraints of  $Y$  in  $G_1(\mathbf{U} \cup \mathbf{S}), G_2(\mathbf{U} \cup \mathbf{S})$ . However, if  $Q \in \mathbf{S}$ ,  $Q$  is a constant and  $\text{Score}(G_1(\mathbf{U} \cup \mathbf{S})) > \text{Score}(G_2(\mathbf{U} \cup \mathbf{S}))$ .  
 280 If  $Q \in \mathbf{U}$ , Eq. (4) suggests that  $Q$  is still the same in the two new DAGs, and  $\text{Score}(G_1(\mathbf{U} \cup \mathbf{S})) > \text{Score}(G_2(\mathbf{U} \cup \mathbf{S}))$  always holds true. Therefore, only  $G_1$ , whose score is the highest, should be retained. The correctness of conclusion on  $\mathcal{G}(\mathbf{U} \cup P)$  can be proved similarly if the best parent set is  $P \cup Q$ .

We can demonstrate the pruning rules under invalid constraints by the example shown in Figure 3. Assume  $\mathbf{U} = \{P, A, B, C\}$ ,  $\mathbf{S} = \{D, E, F\}$  and ancestral

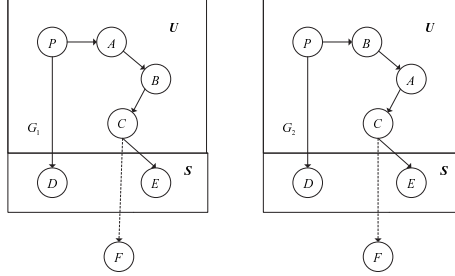


Figure 3: Example of  $\mathcal{F}_G(X, Y)$

285

constraint  $\mathcal{D}(B) = F$ .  $G_1$  and  $G_2$  are DAGs in node  $\mathcal{G}(\mathbf{U})$  with  $\text{Score}(G_1) > \text{Score}(G_2)$ . If  $\mathcal{F}_{G_i}(B, F)(i = 1, 2)$  is false, the best parent set of  $F$  contains at least one of  $\{B, C\}$  (here, we assume that  $C$  is selected). As a descendant of  $B$  in  $G_2$ ,  $A$  would not appear in the best parent set without  $C$ . Further we consider nodes  $\mathcal{G}(\mathbf{U} \cup \mathbf{S})$ , where  $F$  is a sink. Without loss of generality, consider variables  $D(B|G_i)(i = 1, 2)$  (such as  $E$ ), whereas others are  $\mathbf{S} \setminus D(B|G_i)(i = 1, 2)$  (such as  $D$ ). If we add  $F$  to the two new DAGs, the required parent could be  $C$  or  $E$  but can not be  $A$ . Therefore, the best parent sets for the two new structures are the same; this indicates that  $G_2$ , whose score is always lower than that of  $G_1$ , can be pruned, regardless of the added sink. Actually, if there exists a best path  $A \rightarrow X \rightarrow F$ , the optimal DAG that contains such a path is reserved in node  $\mathbf{U} = \{P, A, B, X\}$ . However, if  $\mathcal{F}_{G_i}(B, F)(i = 1, 2)$  is true,  $G_2$  cannot be pruned because  $\text{Score}(G_1) + \text{Score}(F, C) < \text{Score}(G_2) + \text{Score}(F, A)$  may hold.

290

295

#### 4.2.2. Algorithm for elimination

Similarly, considering multiple ancestral constraints, we have the following conclusion: assume that  $G_1$  and  $G_2$  are two DAGs in  $\mathcal{G}(\mathbf{U})$  with  $\text{Score}(G_1) > \text{Score}(G_2)$  and  $|IF(\mathbf{U})| > 1$ . i) If for all  $Y \in IF(\mathbf{U})$ , the constraint  $\mathcal{A}(Y) \rightsquigarrow Y$  is invalid in  $G_1$  and  $G_2$ , then the DAGs with the highest score should be retained. ii) If for some  $\mathbf{Q} \subseteq IF(\mathbf{U})$ , for all  $Y \in \mathbf{Q}$ , the constraint  $\mathcal{A}(Y) \rightsquigarrow Y$  is valid and  $D(\mathcal{A}(Y)|G_2) \subseteq D(\mathcal{A}(Y)|G_1)$ , then  $G_2$  can be pruned. A summary is provided below on the general approach for pruning suboptimal DAGs of  $\mathcal{G}(\mathbf{U})$  in ACOG: Firstly, we set a revenue list for all instable factors in  $\mathcal{G}(\mathbf{U})$  as follows:

$$M(\mathcal{G}(\mathbf{U})) = \begin{array}{ccccc} & IF(\mathbf{U})_1 & IF(\mathbf{U})_2 & \dots & IF(\mathbf{U})_m \\ G_1 & 1 & 1 & \dots & 1 \\ G_2 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ G_n & 0 & 1 & \dots & 1 \end{array}$$

---

**Algorithm 1** ACOG with Ancestral Constraints

---

```
Initialize  $PreLayer \leftarrow \emptyset$ .
for  $Layer = 1$  to  $n$  do
  for each node  $\mathcal{G}(U)$  in the  $PreLayer$  do
    for  $X \in \mathbf{V} \setminus U$  and  $U \cup X$  isn't violated do
       $NowLayer[U \cup X].visited = NowLayer[U \cup X].visited \cup X$ 
      for  $G(U) \in \mathcal{G}(U)$  do
         $G(U \cup X) = \text{BestStructure}(G(U), X, \mathcal{A}(X))$ 
         $NowLayer[U \cup X].str = NowLayer[U \cup X].str \cup G(U \cup X)$ 
      end for
    end for
  end for
  if  $NowLayer[U \cup X].visited = U \cup X$  then
    if  $U \cup X$  is all-satisfied then
       $NowLayer[U \cup X].str = \max(NowLayer[U \cup X].str)$ 
    else
       $M(\mathcal{G}(U \cup X)) = \text{SetRevMat}(NowLayer[U \cup X].str, IF(U \cup X))$ 
      Group  $M(\mathcal{G}(U \cup X))$  with same values of row into  $groups(U \cup X)$ 
       $NowLayer[U \cup X].str = \emptyset$ 
      for  $Gro \in groups(U \cup X)$  do
         $Gro = \text{CutInvalid}(Gro)$ 
         $NowLayer[U \cup X].str = NowLayer[U \cup X].str \cup Gro$ 
      end for
    end if
  end if
   $PreLayer \leftarrow NowLayer, NowLayer \leftarrow \emptyset$ 
end for
```

---

300 where 1 indicates  $\mathcal{F}_G(\mathcal{A}(IF(U)), IF(U)) \geq 0$ , whereas 0 implies  $\mathcal{F}_G(\mathcal{A}(IF(U))$   
with  $IF(U) < 0$ . Secondly, we divide the DAGs into different groups and  
ensure that the values of  $IF(U)_i (i = 1 : m)$  are the same in each rows. Finally,  
we remove all suboptimal DAGs in every group and store the retained DAGs  
for the next stage.

305 The pseudocode of the entire pruning procedure for violated and suboptimal  
structures in ACOG is described as Algorithm 1.

So far, in the previous sections, we have discussed in detail our proposed  
approaches for pruning violated nodes and DAGs and suboptimal DAGs with  
the relevant principles and the proof. These approaches can prevent the compu-  
tational time cost from increasing exponentially when the number of variables  
310 increases since many violated nodes can be removed. In the next section, some  
practical experiments are given to demonstrate the effectiveness of the benefits  
of the proposed approaches.

## 5. Experiments

315 In this section, we first discuss the necessity of incorporating weak expert  
knowledge in a qualitative form; then evaluate the performance of the proposed  
framework under ancestral constraints in a quantitative form. As mentioned  
in Section 1, learning BN structure can be divided into two phases: Parent  
Identification and Structure Optimization. Either exact search strategies based  
320 on ACOG, OG, and EC-Tree or other approximate methods are all related to  
phase two. In phase one, the BIC scores have been pre-computed based on  
independence selection ordering without restriction on the in-degree of parent  
variables(BIC\*)[31, 32]. For a small-scale network, parent sets are identified  
with a 1s time limit, and 10s on middle-scale networks<sup>1</sup>. For the sake of fair-  
325 ness, the computational cost of phase one is not considered in the comparison.  
ACOG is implemented in R language and run on an Intel Pentium G4560 CPU  
with a 12GB memory limit. We apply memory-efficient dynamic programming  
(MEDP)[34] to search the exact solution based on proposed framework<sup>2</sup>.

330 The independent variables in the simulations included the number of vari-  
ables ( $n$ ), the number of observations from the ground-truth network ( $m$ ), the  
constraint rate ( $p$ ) and the error constraint rate ( $wp$ ):

- $n$ : We chose a random sub-network of a given size from four standard BN benchmarks: Insurance, Alarm, Hailfinder, and Hepar2<sup>3</sup>.
- $m$ :The training dataset was generated from the above sub-networks.
- 335 •  $p$ : The  $p\%$  of constraints that represent directed paths in the ground-  
truth network have been generalized and utilized in the BN structure  
learning. For each test instance, we used part of the constraints with a  
certain probability ranging from 0 to 1. Notably, the number of paths  
presented in the network has limited the maximum size of constraints,  
340 and the negative constraints have not been taken into account.
- $wp$ : A percentage of fatal constraints:  $wp\%$  of constraints have declared  
that the path does not exist in the ground-truth network.

The comparison was based on the following criteria, and we chose the different  
criteria according to candidate algorithms and the propose of experiments.

- 345 •  $t$ : Time recorded the running time of the algorithm.

---

<sup>1</sup>As literature[33], we judge the scale of networks according to the number of variables: a small network with  $n < 20$ , a middle network with  $20 \leq n < 50$ , and a large network with  $n \geq 50$ .

<sup>2</sup>Although A\* search is known to be faster than DP, the reason DP is applied is that some nodes which are suboptimal without ancestral constraints but are part of the optimal graph with ancestral constraints in ACOG may not be traversed to based on A\* search, while DP always takes all nodes into account.

<sup>3</sup><https://www.bnlearn.com/bnrepository/>

- **Score:** Score estimated the accuracy of the learned network. When discussing the influence of the constraint rate, we used the score of learned structure based on EC-Tree as the benchmark and recorded the deviation rate in other algorithms. In addition to other situations, the actual score value was reserved. 350
- **Nodes:** The maximum number of nodes considered for a comparison between the exact graph search strategies. The fewer nodes to be expanded, the less the memory cost.
- **Constraint Satisfaction Rate (CSR):** It should note that the learned network based on exact methods always satisfied all the ancestral constraints, whereas approximate heuristic search ignored some constraints. So the constraint satisfaction rate, an index of required paths presented in observed DAG, should be considered besides. 355
- **Structural Hamming Distance (SHD):** SHD measured the distance between the learned structure and target structure. A higher SHD value indicated that the result deviated from the ground-truth network further. 360

### 5.1. Qualitative analysis of ACOG under ancestral constraints

To qualitatively analyze the superiority of incorporating ancestral constraints, we have compared ACOG against other mainstream methods for BN structure learning without constraints, including constraint-based algorithm PC, score-based algorithm HC, hybrid algorithm MMHC, and exact search algorithm based on OG. All of PC, HC, MMHC were approximate methods. The approximate algorithms have been implemented in R language (<https://cran.r-project.org/web/packages/bnlearn/>). SHD and Score were adopted as the criteria of accuracy. We simulated the subnetworks from three benchmark networks with  $n \in [10, 20]$  and  $m \in \{1000, 5000\}$ . Figures 4 and 5 display the performance of the five algorithms considered. 365

In Figure 4, HC always resulted in a high SHD as a score-based algorithm is driven by some heuristic strategies focusing on the fitness of the edges and the training data. As a result, the judgment on the orientation of edges heavily depended on the dataset, which has caused a deviation from the target network. On the contrary, PC carried out conditional independent tests on the variables, then identified the v-structure and equivalent classes to learn an optimal BN structure. PC has focused on exploring the distribution of training data with respect to the causal relationships between the variables. Thus, the result learned by constrained-based algorithms got closer to the target network. Moreover, the performance of the hybrid algorithm MMHC was between HC and PC. The exact methods, OG and ACOG, can find a global optimum solution with a low SHD. Furthermore, ACOG projected the prior knowledge and attempted to make a trade-off between the training data and the real network structure. As a result, the learned network of ACOG was always the closest to the ground-truth network in almost all conditions. The comparison of SHD conformed to the conclusion that  $HC > MMHC > PC \approx OG > ACOG$ , and 370

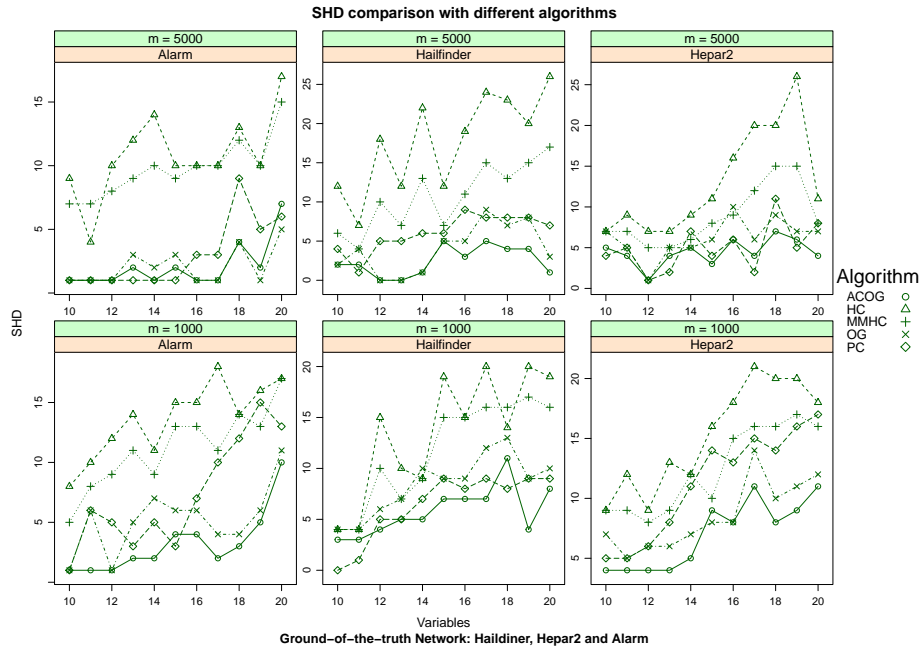


Figure 4: SHD comparison with different algorithms

has demonstrated the efficiency and the necessity of enforcing the qualitative  
 390 prior knowledge.

Figure 5 showed the scoring trends with the variation of the benchmark  
 networks and the datasets, and we can find that the gap between different  
 algorithms was insignificant. Especially in Hepar2, a sparse network, all the  
 algorithms behaved almost the same regardless of the size of the datasets. OG  
 395 always resulted in the highest score compared to the approximate approaches  
 HC, MMHC, and PC. Incorporating ancestral constraints did not promote a  
 structure with a higher score; On the contrary, the learned network obtained  
 a worse evaluation in many conditions. It was hard to guarantee that the  
 ground-truth network entirely could fall into the global optimum solution with  
 400 the given training data, and a real path did not always acquire a high score,  
 although the training data was generated from the standard network. Thus,  
 the score generally decreased after incorporating ancestral constraints compared  
 to the original exact algorithm. All the subfigures in Figure 5 also indicated  
 that the score deviation was usually small when the ancestral constraints were  
 405 sampled from the ground-truth network without wrong prior knowledge. In  
 such a condition, ACOG performed better than the approximate methods such  
 as MMHC and PC.

Additionally, figure 6 showed a comparison of the number of nodes for each

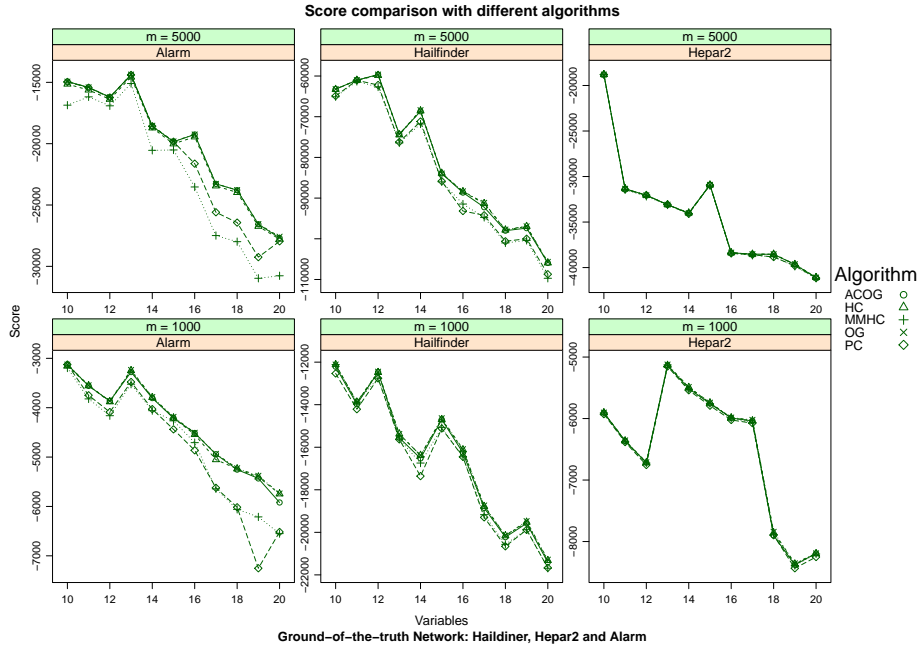


Figure 5: Score comparison with different algorithms

layer in ACOG and OG. The line in red in the figure ( $p = 0$ ) indicated the scale of OG at different stages without ancestral constraints, and other lines gave the scale of ACOG with different numbers of constraints. It can be seen that the trend of the nodes in each layer was the same as the variation of the total expanded nodes, and ACOG with ancestral constraints always cost less memory than OG. This validated the necessity and efficiency of incorporating ancestral constraints.

### 5.2. Quantitative analysis of ACOG under ancestral constraints

To evaluate the effectiveness of the proposed framework under ancestral constraints in a quantitative form, we compared it with EC-Tree[12], which was a state-of-the-art framework. Notably, the exact search strategies always reached the global optimum in EC-Tree despite imposing ancestral constraints. We also compared ACOG against an efficient constraint-based heuristic strategy (MINOBSx)[6]<sup>4</sup>. MINOBSx, an approximate method, intended to find an optimal network under the conditions that satisfied the ancestral constraints as much as possible. The result on the same dataset was highly associated with

<sup>4</sup><https://github.com/acliuw/MINOBS-anc>



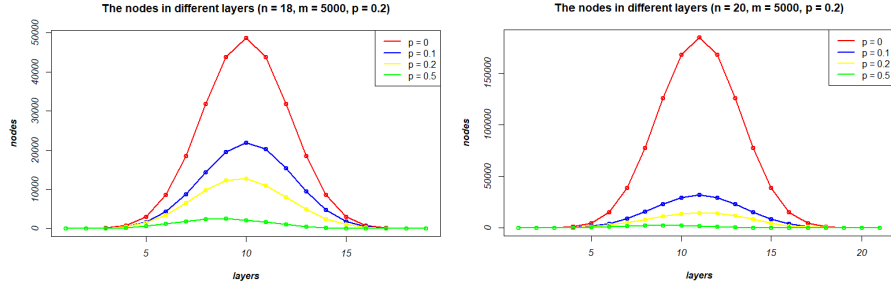


Figure 6: Nodes in different layers of ACOG when incorporating ancestral constraints

425 the initial network in approximate search. Without loss of generality, the mean values of criteria in 10 experiments were adopted when executing MINOBSx.

### 5.2.1. Efficiency and Accuracy of ACOG

We varied the number of variables  $n \in \{10, 12, 14\}$  and the size of dataset  $m \in \{1000, 5000\}$  in the benchmark networks. Table 3 showed the comparison of the three small-scale networks with the constraints completely generalized from the ground-truth network.

The performance of ACOG and EC-Tree was compared. In terms of accuracy, the score gap between ACOG and EC-Tree was 0% in all the 24 cases, which indicated that ACOG always conducted an optimal global search. As for efficiency, the time consumption in 24/24<sup>5</sup> results based on ACOG was lower than those based on EC-Tree. It has also been observed that: 1). The relationship between the constraint rate and time consumption. The cost of ACOG and EC-Tree both decreased with the increase of the constraint rate. 2). The relationship between the number of variables and time consumption. With the scale of the learning problem becoming large, the growth was more rapid in EC-Tree, whereas it was slow in ACOG. For example, if  $m = 5000$  and  $p = 0.1$ , the cost of EC-Tree was 1, 75, 227 times of those of ACOG, respectively, when the number of variables was 10, 12, and 14. In terms of memory consumption, the expanded nodes of EC-Tree were less than ACOG when the number of variables was small because A\* search was implemented in EC-Tree, whereas MEDP in ACOG. However, the complexity of EC-Tree dramatically grew, even though the number of variables slightly increased. If  $m = 1000$  and  $p = 0.2$ , the expanded nodes ratio was 0.27, 0.9 and 4.87 between EC-Tree and ACOG when  $n = 10, 12$  and 14, respectively. The experiments asserted a higher efficiency of ACOG compared to EC-Tree without loss of accuracy when incorporating

<sup>5</sup>x/x in the whole section represents that the experiment sets satisfied the following conclusion / the total experiment sets.

Table 3: Results comparisons in three small-scale networks for different algorithms when  $w_p = 0$

(n, m, p)	EC-Tree		ACOG			MINOBSx		
	t(s)	Nodes	t(s)	Score	Nodes	t(s)	Score	CSR
(10, 1000, 0.1)	0.035	205	0.033	0%	665	0.1604	6.01%	1
(10, 1000, 0.2)	0.013	112	0.013	0%	413	0.29	6.3%	1
(10, 1000, 0.5)	0.003	41	0.003	0%	109	0.1724	6.67%	1
(10, 1000, 1.0)	0.002	21	0.002	0%	74	0.2306	0%	1
(10, 5000, 0.1)	0.26	1070	0.024	0%	699	0.2076	0.3%	1
(10, 5000, 0.2)	0.078	596	0.004	0%	239	0.1872	0%	0.88
(10, 5000, 0.5)	0.033	183	0.003	0%	117	0.2342	0%	1
(10, 5000, 1.0)	0.005	40	0.001	0%	66	0.178	0%	1
(12, 1000, 0.1)	5.561	7588	0.074	0%	1652	0.3436	2.23%	1
(12, 1000, 0.2)	0.112	491	0.045	0%	543	0.5774	3.27%	0.97
(12, 1000, 0.5)	0.005	72	0.004	0%	108	0.6218	0.55%	0.96
(12, 1000, 1.0)	0.002	35	0.002	0%	45	0.4636	0%	1
(12, 5000, 0.1)	14.2538	22907	0.091	0%	1919	0.7236	0.21%	1
(12, 5000, 0.2)	4.757	8974	0.061	0%	1381	0.5156	1.25%	0.93
(12, 5000, 0.5)	0.016	141	0.006	0%	203	0.5824	0.02%	0.99
(12, 5000, 1.0)	0.004	67	0.003	0%	84	0.758	0%	1
(14, 1000, 0.1)	37.588	40440	0.687	0%	12287	0.689	0%	1
(14, 1000, 0.2)	49.562	52958	1.037	0%	10859	0.7344	0.06%	1
(14, 1000, 0.5)	10.655	13075	0.462	0%	4102	1.1356	0%	1
(14, 1000, 1.0)	0.23	1055	0.134	0%	1639	1.2504	0%	1
(14, 5000, 0.1)	202.641	146615	0.889	0%	12454	0.6554	0.34%	1
(14, 5000, 0.2)	101.109	74354	0.6	0%	7071	0.7638	0.009%	1
(14, 5000, 0.5)	28.1861	21636	0.169	0%	2716	1.4402	0.009%	0.98
(14, 5000, 1.0)	0.127	531	0.027	0%	659	1.1904	0%	0.99

ancestral constraints.

Now consider the performance of ACOG and MINOBSx. It was easy to find that ACOG always consumed less time than MINOBSx in 22/24 simulation conditions. When imposing ancestral constraints, 14/24 results based on MINOBSx were deviated from the best solution and got stuck in local optimum. Furthermore, the maximum deviation rate was 6.67%. More detailed comparisons have revealed: 1). The relationship between time consumption and the number of variables. When the learning problem was simple, traversing the whole search space was much easier than optimizing through a heuristic strategy. So the time consumption of ACOG was lower than MINOBSx when  $n = 10$  and 12. Nevertheless, the growth of MINOBSx tended to be smoother, with the scale of the network becoming large. When the number of variables varied from 10 to 12 and 12 to 14 (assume that  $m = 5000$ ,  $p = 0.1$ ), the time consumption increased by two times and two times of MINOBSx, whereas two times and ten times of ACOG. Moreover, this trend was more apparent in medium-scale networks. 2). The relationship between the time consumption and constraint rate. Different from the trends of EC-Tree and ACOG, it was a more general case that the cost of MINOBSx did not increase even more constraints provided. Additionally,

MINOBSx can utilize most of the ancestral constraints: 17/24 results satisfied  
 470 all constraints, and CSR in 23/24 results was no more than 3%.

### 5.2.2. Robustness of ACOG

We also empirically evaluated the robustness of ACOG when there were mi-  
 nor errors and fatal errors in ancestral constraints. Table 4 and Table 5 showed  
 the performance of the three approaches when ancestral constraints violated the  
 475 ground-truth network. Because of the inappropriate domain knowledge, there  
 was no feasible solution in 7/24 results when  $w_p = 0.2$  and 10/24 results when  
 $w_p = 0.5$  based on EC-Tree and ACOG.

Table 4: Results comparisons in three small-scale networks for different algorithms when  $w_p = 0.2$  (\ \ indicate that there is no feasible solution)

(n, m, p)	EC-Tree		ACOG			MINOBSx		
	t(s)	Nodes	t(s)	Score	Nodes	t(s)	Score	CSR
(10, 1000, 0.1)	0.016	142	0.028	0%	598	0.0978	6.17%	1
(10, 1000, 0.2)	0.005	79	0.011	0%	375	0.1534	1.33%	0.96
(10, 1000, 0.5)	0.003	41	0.001	0.52%	75	0.1916	-0.57%	0.92
(10, 1000, 1.0)	\	\	\	\	\	0.2994	\	0.9
(10, 5000, 0.1)	0.835	2764	0.196	0%	673	0.1384	0.23%	0.9
(10, 5000, 0.2)	0.342	1032	0.008	0.16%	326	0.1388	0.17%	0.92
(10, 5000, 0.5)	\	\	\	\	\	0.287	\	0.91
(10, 5000, 1.0)	\	\	\	\	\	0.1696	\	0.88
(12, 1000, 0.1)	2.159	4943	0.196	0%	1607	0.5136	0.97%	1
(12, 1000, 0.2)	0.038	262	0.038	0%	403	0.5178	2.53%	0.93
(12, 1000, 0.5)	\	\	\	\	\	0.7896	\	0.94
(12, 1000, 1.0)	\	\	\	\	\	0.7298	\	0.92
(12, 5000, 0.1)	13.78	15952	0.123	0%	1636	0.5664	0.76%	1
(12, 5000, 0.2)	2.234	5504	0.144	0.16%	1275	0.701	0.21%	0.97
(12, 5000, 0.5)	\	\	\	\	\	1.1398	\	0.89
(12, 5000, 1.0)	\	\	\	\	\	1.1844	\	0.93
(14, 1000, 0.1)	103.046	89271	1.305	0%	14376	0.8658	0.13%	1
(14, 1000, 0.2)	60.414	53850	0.868	0%	10088	0.5728	0%	1
(14, 1000, 0.5)	26.6942	28858	0.215	0.47%	3119	1.5476	0.03%	1
(14, 1000, 1.0)	0.164	887	0.097	0%	1407	1.058	0%	1
(14, 5000, 0.1)	155.249	119160	0.803	0%	12287	0.5808	0.04%	1
(14, 5000, 0.2)	196.669	126462	1.01	0%	8504	0.6638	0.13%	1
(14, 5000, 0.5)	3.2	5197	0.226	0%	1631	0.8554	0.03%	0.98
(14, 5000, 1.0)	0.726	1841	0.017	0%	419	1.0598	0.41%	0.97

When the error rate was 20%, ACOG still navigated a path leading to global  
 optimum in 13/17 results. Furthermore, the maximum score deviation rate in  
 480 the four negative examples was no more than 0.6%, which was a tiny error.  
 It can be interpreted as when constraints were consistent with the ground-  
 truth network or slightly violated; there was a tiny difference between the best  
 structure learned with and without constraints. If we expanded the search  
 space, ACOG still can observe the optimal BN structure. The trends of time  
 485 consumption in ACOG and EC-Tree were same as the condition when  $w_p =$

Table 5: Results comparisons in three small-scale networks for different algorithms when  $w_p = 0.5$  (\ \ indicate that there is no feasible solution)

(n, m, p)	EC-Tree		ACOG			MINOBSx		
	t(s)	Nodes	t(s)	Score	Nodes	t(s)	Score	CSR
(10, 1000, 0.1)	0.004	65	0.021	0.29%	520	0.1372	0.38%	0.9
(10, 1000, 0.2)	0.004	71	0.018	1.36%	281	0.1948	2.42%	0.96
(10, 1000, 0.5)	0.002	34	0.004	1.2%	119	0.2222	2.93%	0.92
(10, 1000, 1.0)	\	\	\	\	\	0.2962	\	0.8
(10, 5000, 0.1)	0.196	993	0.026	0%	697	0.131	0.18%	1
(10, 5000, 0.2)	\	\	\	\	\	0.1222	\	0.6
(10, 5000, 0.5)	\	\	\	\	\	0.4198	\	0.74
(10, 5000, 1.0)	\	\	\	\	\	0.439	\	0.67
(12, 1000, 0.1)	3.152	5366	0.119	3.54%	1132	0.4072	5.54%	0.9
(12, 1000, 0.2)	0.213	854	0.033	3.22%	386	0.5902	7.39%	1
(12, 1000, 0.5)	\	\	\	\	\	0.7414	\	0.68
(12, 1000, 1.0)	\	\	\	\	\	0.7824	\	0.72
(12, 5000, 0.1)	19.014	27946	0.322	0.79%	2542	0.458	2.06%	1
(12, 5000, 0.2)	0.587	2110	0.024	0%	557	0.4338	2.12%	0.9
(12, 5000, 0.5)	0.324	572	0.004	1.02%	102	1.3138	3.78%	0.91
(12, 5000, 1.0)	\	\	\	\	\	1.4186	\	0.74
(14, 1000, 0.1)	38.18	40440	0.749	0%	12287	0.4562	0.03%	1
(14, 1000, 0.2)	55.3125	55547	1.406	0%	9215	0.67	0%	1
(14, 1000, 0.5)	27.298	25936	0.706	0.37%	2534	0.8342	0.59%	1
(14, 1000, 1.0)	\	\	\	\	\	0.8892	\	0.871
(14, 5000, 0.1)	140.115	104195	0.739	0%	12287	0.763	0.04%	1
(14, 5000, 0.2)	68.3672	54878	1	0%	6687	0.9398	0.04%	0.93
(14, 5000, 0.5)	\	\	\	\	\	1.3042	\	0.778
(14, 5000, 1.0)	\	\	\	\	\	1.5	\	0.86

0, which has demonstrated that the total provided constraint, instead of the correct constraints, would affect efficiency.

Next, we compared ACOG with MINOBSx. In 15/17 conditions, ACOG produced a better solution, and the maximum score deviation rate of MINOBSx was 6.17%, which was much more significant than 0.6% in ACOG. As observed, MINOBSx can always find a solution even there were mirror errors in prior knowledge. 9/17 results based on MINOBSx satisfied all the constraints, and the maximum deviation of CSR was no more than 10% in 6/8 negative examples. It was interesting that when  $n = 10$ ,  $m = 1000$ , and  $p = 0.5$ , the structure learned by MINOBSx made higher quality than that observed by EC-Tree. Because MINOBSx ignored some ancestral constraints that indicated the incredibly wrong domain knowledge with a terrible score. However, EC-Tree and ACOG always satisfied all the constraints, which caused a worse result.

When the error rate was 50%, the accuracy of ACOG and MINOBSx had relatively large fluctuations because of the more inappropriate ancestral constraints. In 6/14 conditions, ACOG searched a global optimum solution, and the maximum score deviation rate was 3.22%. MINOBSx only observed 1/14 best structure with a maximum deviation rate up to 7.39%. In the 13/14 ex-

periment sets, ACOG performed better than MINOBSx in terms of time cost.  
 505 It was worth mentioning that when  $w_p$  increased, the CSR has dropped significantly. Especially the CSR in 13/24 results has reduced to below 90% if  $w_p = 0.5$ .

### 5.2.3. Scalability of ACOG

Table 6 showed the comparison in the medium-scale network where  $n =$   
 510  $\{20, 24\}$  and  $m = \{1000, 5000\}$ . The memory of EC-Tree was overflowing because of the numerous expanded nodes. As such, the result of experiments based on EC-Tree was not available; we recorded the actual score of the learned structure instead of the deviation from the global optimum solution.

Table 6: Results comparisons in three small-scale networks for different algorithms when the data size is 5000 (\ indicate that there is no feasible solution)

(n, $w_p$ , p)	ACOG			MINOBSx		
	t(s)	Score	Nodes	t(s)	Score	CSR
(20, 0.2, 0.1)	73.137	-103406	442098	44.031	-103433	1
(20, 0.2, 0.2)	94.331	-103713	135134	27.17	-103716	1
(20, 0.2, 0.5)	7.361	-103586	31359	46.554	-103595	1
(20, 0.2, 1.0)	1.185	-104574	2423	23.416	-104584	0.95
(20, 0.5, 0.1)	63.762	-103671	516728	47.533	-103688	1
(20, 0.5, 0.2)	80.342	-103486	167760	23.314	-103551	0.94
(20, 0.5, 0.5)	12.906	-105178	16752	21.384	-104923	0.75
(20, 0.5, 1.0)	11.323	-105178	16752	32.243	-105178	0.88
(24, 0.2, 0.1)	1349.18	-125078	1594190	149.03	-125563	1
(24, 0.2, 0.2)	179.196	-125642	171942	55.776	-125866	0.9
(24, 0.2, 0.5)	15.823	-125301	54198	57.764	-126927	0.6
(24, 0.2, 1.0)	\	\	\	86.434	-126464	0.7
(24, 0.5, 0.1)	486.557	-125432	587663	103.865	-125549	0.9
(24, 0.5, 0.2)	120.256	-125542	105466	62.983	-126557	0.95
(24, 0.5, 0.5)	14.572	-128428	1794	133.827	-129806	0.84
(24, 0.5, 1.0)	\	\	\	111.325	-127844	0.72

The score of ACOG in 13/14 experiment sets was higher, and the only  
 515 exception was because MINOBSx satisfied 75% of constraints. Similarly, there were some detailed comparisons: 1). The conclusion on the relationship between the constraint rate and time consumption in the small-scale network was still applicable in a medium-scale network. If  $p < 0.5$ , ACOG spent more time than MINOBSx in 8/8 conditions. And if  $p \geq 0.5$ , the time consumption of ACOG  
 520 was less in all 6/6 experiment sets. 2). Generally, with the increase of the size of variables, the growth of ACOG's time cost was still faster than that of MINOBSx's. So it was obvious MINOBSx can handle more complex problems that ACOG cannot.

We tested the scalability of ACOG in more complex conditions where some  
 525 subnetworks with  $n = \{30, 32\}$  were generated from Hailfinder with the default

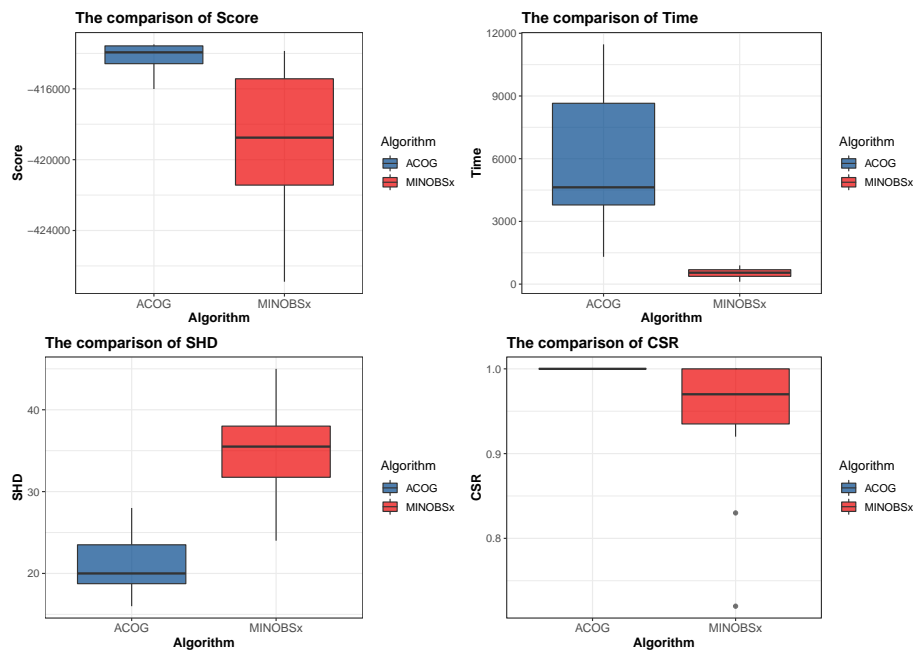


Figure 7: The comparison in Hailfinder\_30

setting of the parameters as  $m = 15000$ ,  $p = 0.25$ , and  $wp = 0$ . Figures 7 and 8 showed a comparison between ACOG and MINOBSx when the number of variables increased. ACOG can incorporate ancestral constraints in a more reasonable way that the quality of the learned network was quite higher. The median score based on ACOG was about 1.11% and 0.71% higher than that based on MINOBSx, and the structure in ACOG was closer to the ground-truth network with an average Hamming distance below 20, whereas it was beyond 30 in MINOBSx. Moreover, ACOG's score gap between the maxima and the minima was 0.56% and 0.14%, whereas it was 3.17% and 1.62% for MINOBSx in Hailfinder\_30 and Hailfinder\_32, and these results have demonstrated the stability of ACOG. Similarly, the hamming distance gap of ACOG had ten arcs and seven arcs less than those of MINOBSx. However, MINOBSx was orders-of-magnitude faster than ACOG, as the time complexity of exact methods was up to  $O(2^n)$ . ACOG spent about 6000s and 24000s in Halifinder\_30 and Hailfinder\_32, and MINOBSx can find the solution within 1000s. Note that the results of ACOG still satisfied all the ancestral constraints, while CRS of MINOBSx significantly fluctuated and depended on the quality of constraints.

In summary, when considering a BN model for tackling a complex problem in the real world, if the most crucial requirement for the model is to integrate all the prior expert knowledge, then only ACOG can be applied. If a modeler is

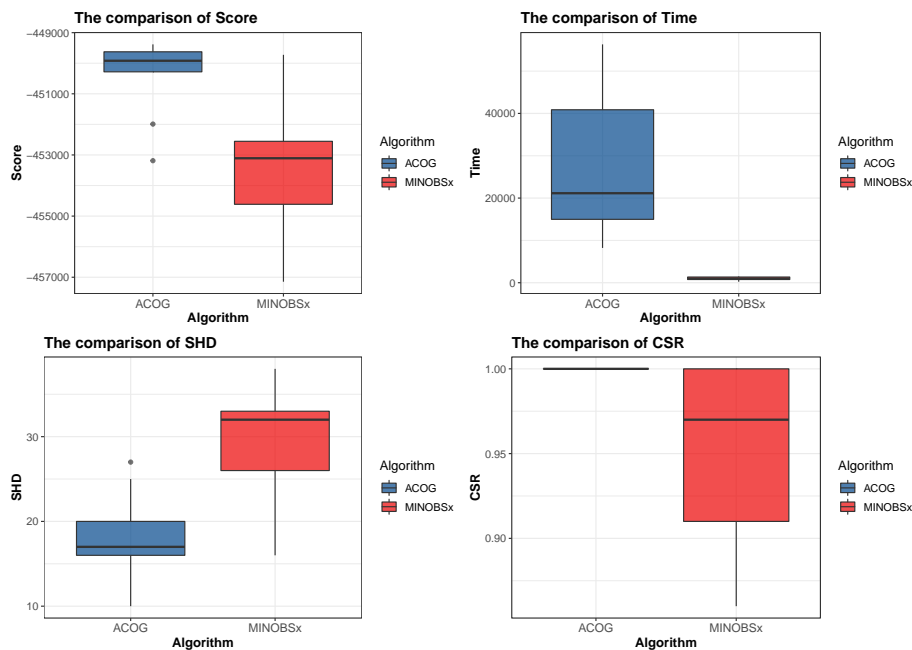


Figure 8: The comparison in Hailfinder\_32

more concerned with the accuracy of the model, ACOG would be preferred to MINOBSx. If the BN structure should be modeled in a limited time, MINOBSx might be the right choice. ACOG cannot project expert knowledge in more complex networks when the number of variables is beyond 35 due to insufficient memory. In theory, the time cost was about 53 hours in such a condition.

Overall, the main advantage of MINOBSx was that it could solve the large-scale problem, and the main advantage of EC-Tree was that it always can find the global optimum solution under the ancestral constraints. The proposed ACOG was a trade-off between accuracy and efficiency. As shown in the simulation results, the ACOG framework has always performed as good as EC-Tree when the ancestral constraints were generalized from the ground-truth network completely. Besides, even if there were some violated constraints, ACOG was still able to find a path leading to a global optimization in most cases. Furthermore, the time and space cost of ACOG's was lower than EC-Tree. However, ACOG still cannot be applied to large-scale networks with ancestral constraints, as almost all the other frameworks.

## 6. Conclusion

We propose a new framework ACOG to learn optimal BN structure. ACOG extends the search scope of OG and can escape from local optimum when imposing ancestral constraints. ACOG prunes violated structures to reduce storage requirements. We have used a “full-satisfied” structure to facilitate the pruning process and employed a simple revenue function to discard suboptimal structures. The pruning rules have been proven in a solid and valid way. Experiments have demonstrated the ACOG can guide an optimal global search even there are some errors in prior knowledge with less time and space consumption compared to EC-Tree. Also, ACOG is more accurate and stable than MINOBSx when utilizing ancestral constraints in terms of the middle-scale networks.

As Yuan and Malone reported on the experimental results with up to 26 variables [27], learning optimal BN structure based on exact search frameworks cannot be applied to large-scale problems. Because many violated nodes are discarded when incorporating ancestral constraints, ACOG slightly enhances the scalability. However, the improvement is not enough to produce a significant qualitative change. When tackling some complex problems, the approximate method, such as MINOBSx, would be a more appropriate choice.

Our further works include studying how to incorporate negative ancestral constraints into order graph. Moreover, the properties of the positive revenue function should also be investigated.

## Acknowledgments

This work was supported by the National Natural Science Foundation of China (61573285).

## References

- [1] J. Pearl, *Causality*, Cambridge university press, 2009.
- [2] A. Darwiche, *Modeling and reasoning with Bayesian networks*, Cambridge university press, 2009.
- [3] L. A. Sierra, V. Yepes, T. García-Segura, E. Pellicer, Bayesian network method for decision-making about the social sustainability of infrastructure projects, *Journal of Cleaner Production* 176 (2018) 521–534.
- [4] M. A. Atoui, A. Cohen, S. Verron, A. Kobi, A single bayesian network classifier for monitoring with unknown classes, *Engineering Applications of Artificial Intelligence* 85 (2019) 681–690.
- [5] A. L. Madsen, F. Jensen, A. Salmerón, H. Langseth, T. D. Nielsen, A parallel algorithm for bayesian network structure learning from large data sets, *Knowledge-Based Systems* 117 (2017) 46–55.



- [6] A. Li, P. Beek, Bayesian network structure learning with side constraints, in: International Conference on Probabilistic Graphical Models, 2018, pp. 225–236.
- [7] M. Bartlett, J. Cussens, Integer linear programming for the bayesian network structure learning problem, *Artificial Intelligence* 244 (2017) 258–271.
- [8] P. Parviainen, M. Koivisto, Ancestor relations in the presence of unobserved variables, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2011, pp. 581–596.
- [9] Y. Chen, L. Meng, J. Tian, Exact bayesian learning of ancestor relations in bayesian networks, in: *Artificial Intelligence and Statistics*, 2015, pp. 174–182.
- [10] E. Chen, A. Choi, A. Darwiche, Learning optimal bayesian networks with dag graphs, in: *Proceedings of the 4th IJCAI Workshop on Graph Structures for Knowledge Representation and Reasoning (GKR’15)*, 2015.
- [11] E. Y.-J. Chen, A. C. Choi, A. Darwiche, Enumerating equivalence classes of bayesian networks using ec graphs, in: *Artificial Intelligence and Statistics*, 2016, pp. 591–599.
- [12] E. Y.-J. Chen, Y. Shen, A. Choi, A. Darwiche, Learning bayesian networks with ancestral constraints, in: *Advances in Neural Information Processing Systems*, 2016, pp. 2325–2333.
- [13] C. Yuan, B. Malone, X. Wu, Learning optimal bayesian networks using a\* search, in: *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.
- [14] G. F. Cooper, E. Herskovits, A bayesian method for the induction of probabilistic networks from data, *Machine learning* 9 (4) (1992) 309–347.
- [15] P. Parviainen, M. Koivisto, Finding optimal bayesian networks using precedence constraints, *The Journal of Machine Learning Research* 14 (1) (2013) 1387–1415.
- [16] C. P. De Campos, Q. Ji, Efficient structure learning of bayesian networks using constraints, *Journal of Machine Learning Research* 12 (20) (2011) 663–689.
- [17] D. M. Chickering, Learning bayesian networks is np-complete, in: *Learning from data*, Springer, 1996, pp. 121–130.
- [18] H. Liu, S. Zhou, W. Lam, J. Guan, A new hybrid method for learning bayesian networks: Separation and reunion, *Knowledge-Based Systems* 121 (2017) 185–197.

- 635 [19] M. Scutari, C. E. Graafland, J. M. Gutiérrez, Who learns better bayesian network structures: Accuracy and speed of structure learning algorithms, *International Journal of Approximate Reasoning* 115 (2019) 235–253.
- [20] S. Behjati, H. Beigy, Improved k2 algorithm for bayesian network structure learning, *Engineering Applications of Artificial Intelligence* 91 (2020) 103617.
- 640 [21] J. Dai, J. Ren, W. Du, Decomposition-based bayesian network structure learning algorithm using local topology information, *Knowledge-Based Systems* (2020) 105602.
- [22] M. Scanagatta, A. Salmerón, F. Stella, A survey on bayesian network structure learning from data, *Progress in Artificial Intelligence* (2019) 1–15.
- 645 [23] M. Koivisto, K. Sood, Exact bayesian structure discovery in bayesian networks, *Journal of Machine Learning Research* 5 (May) (2004) 549–573.
- [24] T. SILANDER, A simple approach for finding the globally optimal bayesian network structure, in: *Proceedings of the twenty-second annual conference on uncertainty in artificial intelligence, 2006, 2006*, pp. 445–452.
- 650 [25] J. Cussens, Bayesian network learning with cutting planes, in: *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, 2011*, pp. 153–160.
- [26] C. Yuan, B. M. Malone, An improved admissible heuristic for finding optimal bayesian networks, in: *Conference on Uncertainty in Artificial Intelligence, 2012*.
- 655 [27] C. Yuan, B. Malone, Learning optimal bayesian networks: A shortest path perspective, *Journal of Artificial Intelligence Research* 48 (2013) 23–65.
- [28] G. Schwarz, et al., Estimating the dimension of a model, *The annals of statistics* 6 (2) (1978) 461–464.
- 660 [29] G. F. Cooper, E. Herskovits, A bayesian method for the induction of probabilistic networks from data, *Machine Learning* 9 (4) (1992) 309–347.
- [30] D. Heckerman, D. Geiger, D. M. Chickering, Learning bayesian networks: The combination of knowledge and statistical data, *Machine learning* 20 (3) (1995) 197–243.
- 665 [31] M. Scanagatta, C. P. de Campos, G. Corani, M. Zaffalon, Learning bayesian networks with thousands of variables, in: *Advances in neural information processing systems, 2015*, pp. 1864–1872.
- [32] M. Scanagatta, G. Corani, C. P. De Campos, M. Zaffalon, Approximate structure learning for large bayesian networks, *Machine Learning* 107 (8–10) (2018) 1209–1227.
- 670

- [33] M. Scutari, J.-B. Denis, Bayesian networks: with examples in R, CRC press, 2014.
- [34] B. Malone, C. Yuan, E. Hansen, Memory-efficient dynamic programming for learning optimal bayesian networks, in: Twenty-Fifth AAAI Conference on Artificial Intelligence, 2011.

675