entropy

MDPI

Article

Dynamic Programming BN Structure Learning Algorithm Integrating Double Constraints under Small Sample Condition

Zhigang Lv [1,2], Yiwei Chen [2], Ruohai Di [2,*], Hongxi Wang [1], Xiaojing Sun [3], Chuchao He [2] and Xiaoyan Li [2]

[1] School of Mechatronic Engineering, Xi'an Technological University, Xi'an 710021, China
[2] School of Electronic Information Engineering, Xi'an Technological University, Xi'an 710021, China
[3] General Office, Northwest Institute of Mechanical and Electrical Engineering, Xianyang 712099, China
* Correspondence: xfwtdrh@163.com

Abstract: The Bayesian Network (BN) structure learning algorithm based on dynamic programming can obtain global optimal solutions. However, when the sample cannot fully contain the information of the real structure, especially when the sample size is small, the obtained structure is inaccurate. Therefore, this paper studies the planning mode and connotation of dynamic programming, restricts its process with edge and path constraints, and proposes a dynamic programming BN structure learning algorithm with double constraints under small sample conditions. The algorithm uses double constraints to limit the planning process of dynamic programming and reduces the planning space. Then, it uses double constraints to limit the selection of the optimal parent node to ensure that the optimal structure conforms to prior knowledge. Finally, the integrating prior-knowledge method and the non-integrating prior-knowledge method are simulated and compared. The simulation results verify the effectiveness of the method proposed and prove that the integrating prior knowledge can significantly improve the efficiency and accuracy of BN structure learning.

Keywords: Bayesian network; prior knowledge; dynamic programming; edge constraint; path constraint

check for updates

Citation: Lv, Z.; Chen, Y.; Di, R.; Wang, H.; Sun, X.; He, C.; Li, X. Dynamic Programming BN Structure Learning Algorithm Integrating Double Constraints under Small Sample Condition. *Entropy* **2022**, *24*, 1354. https://doi.org/10.3390/e24101354

Academic Editors: Jaroslaw Krzywanski, Yunfei Gao, Marcin Sosnowski, Karolina Grabowska, Dorian Skrobek, Ghulam Moeen Uddin, Anna Kulakowska, Anna Zylka and Bachil El Fil

Received: 10 August 2022
Accepted: 21 September 2022
Published: 24 September 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Many problems in the real world face uncertainty factors, and artificial intelligence today deals with problems of uncertainty, such as image recognition, speech recognition, intelligent decision-making, and so on. A Bayesian Network (BN) [1], as a type of Graphical Model, has become a powerful tool to treat uncertainty problems because of its strict mathematical foundation, visual and understandable graphic topological structure, as well as a natural expression of reality problems. In recent years, Bayesian Networks have been successfully applied in various fields such as medical diagnosis [2,3], fault diagnosis [4], decision analysis [5], gene analysis [6,7], target identification [8], threat assessment [9,10], and system reliability analysis [11,12].

However, before a BN is used to solve problems in engineering practice, the BN structure needs to be constructed first. Compared with the approximate BN structure search algorithm based on constraint and a heuristic algorithm, the accurate solution of the BN structure learning has recently become a popular topic in academic research. The accurate solution includes the branch and bound method [13], integer programming [14,15], and Dynamic Programming (DP) [16,17]. Although the traditional BN structure learning algorithm based on DP can obtain the global optimal solution, the acquired structure is inaccurate when the sample does not completely contain the information of the real structure, especially when the sample size is small. The complexity problem is also the bottleneck faced by the current DP method. However, in reality, there is a lot of deterministic prior knowledge available in BN modeling. The prior-knowledge distribution of the BN structure is learned to put forward a method of Bayesian network model averaging [18]. The BN structure learning is transformed into a constrained objective function extremum problem with the

node order in [19]. Campos [20] considers various deterministic constraints, analyzes the interaction between constraints, and realizes it by the hill climbing method and PC algorithm. The Nicholson [21] Incorporating expert elicited structural information in the CaMML causal discovery program. The results show that with prior knowledge, CaMML has excellent properties. Castelo [22] conducted BN structure learning by specifying the prior knowledge. Borboudakis [23] took the probability of edge and path existence as prior knowledge through rigorous mathematical derivation and conducted structured learning through the BD score and hill-climbing method. The node order prior knowledge is integrated into the process of dynamic programming in [24]. The path constraints are used to learn the BN structure with integer programming [25]. Li proposed a constraint-based hill-climbing approach to incorporate all these constraints [26]. Cussens [27] considered integer linear programming (ILP) as constrained optimization and treated all constraints as cutting planes.

As can be seen, the use of prior knowledge can not only improve the learning accuracy but also the learning efficiency. However, no one has ever studied the use of edge and path prior knowledge in the process of DP structure learning. Therefore, this paper proposes a BN structure learning algorithm based on DP, which combines expert prior knowledge and sample information effectively. The proposed algorithm incorporates edge constraints and path constraints to limit the search process of DP and delete parts of the planning space, so that all search processes meet the prior-knowledge requirements, thus reducing the complexity of the algorithm.

The rest of the paper is organized as follows. Section 2 introduces the theoretical basis of a Bayesian Network. Section 3 introduces in detail the dynamic programming BN structure learning algorithm integrating prior knowledge. In Section 4, the algorithm proposed in this paper is simulated and analyzed in terms of effectiveness and complexity. Section 5 is the conclusion.

2. Theoretical Basis of Bayesian Network

Prior to the general definition of Bayesian networks, several basic concepts in graph theory need to be introduced.

X and Y are two nodes in the directed graph G. $X \to Y$ means there is an edge from X to Y where X is called the parent node of Y while Y is called the child node of X. For any node X, $Ch(X)$ represents the set of all child nodes while $Pa(X)$ represents the set of all parent nodes of X. If X has no parent node, then X is a root node. The set of all root nodes of G is $Root(G)$. If X has no child node, then X is a leaf node. The set of all leaf nodes of G is $Leaf(G)$. If there are k nodes, i.e., $X_1, \ldots, X_k$ in G, and for each $i = 1, \ldots, k-1$, there is $X_i \to X_{i+1}$, then there is a directed path from X_i to X_k, marked as $X_1 \Rightarrow X_k$. For any $X \Rightarrow Y$, X is called the ancestor node of Y while Y is the descendant node of X. Likewise, $An(X)$ represents the set of all ancestor nodes of X and $De(X)$ represents the set of all descendant nodes. If there is a node in G, and the node is its own ancestor node, then the graph has a directed cycle. If the directed graph does not have any directed cycle, then it is a Directed Acyclic Graph (DAG).

A Bayesian Network consists of a DAG and a Conditional Probability Table (CPT), and its complete definition is as follows:

Definition 1 [28]. *A Bayesian Network is a binary group* $1\langle G, \Theta \rangle$*, in which* $G = (\boldsymbol{V}, \boldsymbol{E})$ *represents the structure of the Bayesian network, a DAG where* $\boldsymbol{V} = \{X_1, X_2, \ldots, X_n\}$ *represents a set of random variables, and* $\boldsymbol{E}$ *is a directed edge set indicating the nature of causal association between variables.* $\Theta = \{P(X_i|Pa(X_i)) : X_i \in \boldsymbol{V}\}$ *is a Conditional Probability Table (CPT).*

Definition 2 [28]. *(node order) A node order o refers to the linear arrangement of some variables in which* $X_i \prec X_j$ *means* X_i *is in front of* X_j*. The node order o is the node order of G. If and only if for arbitrary* $\{X_i, X_j\} \subset vari(o)$ *there is* $X_i \prec X_j$ *in o, then* X_j *cannot be an ancestor node of* X_i*.*

Theorem 1 [28]. *With BIC score as the criterion, in an optimal Bayesian network, any node has at most* $\lfloor \log(2N/\log N) \rfloor$ *parent nodes, where N is the number of samples. This article refers to* $\lfloor \log(2N/\log N) \rfloor$ *as* n_{mp} *(max number parents).*

3. BN Structure Learning Algorithm for Dynamic Programming Integrating Prior-Knowledge

3.1. Dynamic Programming Algorithm

A Bayesian network structure learning algorithm based on dynamic programming is a process of accurately solving mathematical programming problems, and with exponential computational complexity, it is limited by the number of nodes. The state transition equations of dynamic programming are:

$$\max Score(\boldsymbol{V}) = \max_{X \in \boldsymbol{V}} \{\max Score(\boldsymbol{V} \backslash X) + \max Score(X, \boldsymbol{V} \backslash X)\}, \tag{1}$$

$$best score(X, \boldsymbol{V} \backslash X) = \max Score(X, \boldsymbol{V} \backslash X) = \max_{Pa(X) \in \boldsymbol{V} \backslash \{X\}} Score(X, Pa(X)), \tag{2}$$

where $\boldsymbol{V}$ is a set of variables, X is a leaf node in the optimal structure, and $Score(\bullet)$ is a decomposable scoring function [29]. Equations (1) and (2) connect the relationship between the whole structure and its substructures, and the optimal network on the remaining nodes $\boldsymbol{V} \backslash X$ is recursively constructed through the above process until the remaining nodes are a variable. All the child node sets form a Hasse Diagram, showing the whole process of dynamic programming. When the DP algorithm calculates from top to bottom, the root node is determined first, and then the leaf nodes that are gradually added to the remaining nodes are universal set variables. When the DP algorithm calculates from bottom to top, leaf nodes are determined first, and then the root nodes that are gradually added to the remaining nodes are empty set variables. Because a Hasse Diagram contains the node order information of the network, it is also called the Order Graph. There is another similar graph called the parents graph [17]. Figure 1 shows a node order graph with the number of nodes as $n = 4$ and the parent node graph of node X_1.

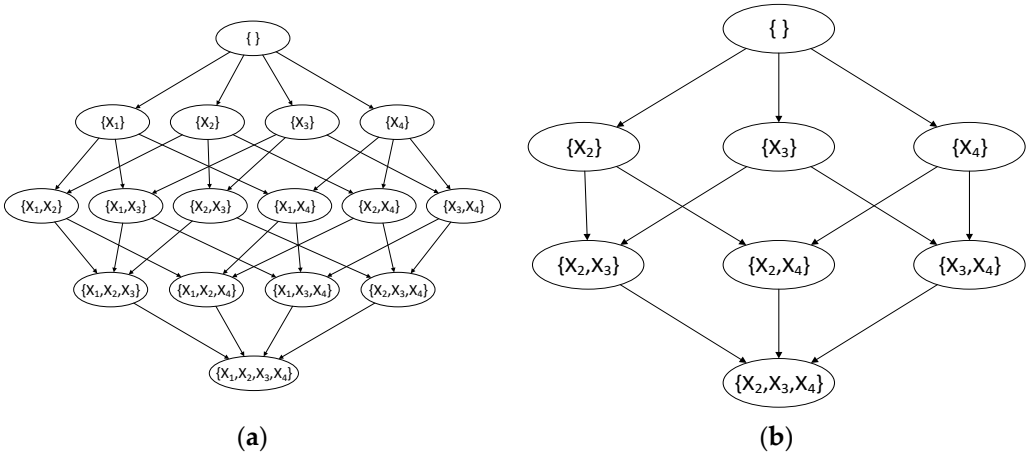


Figure 1. Node order graph and parent node graph of X_1: (**a**) Node order graph of four nodes; (**b**) Parent node graph of X_1.

3.2. Expression of Constraints

In this paper, deterministic prior knowledge is directly transformed into some constraints. Prior knowledge and prior constraints are equivalent concepts. For convenience of expression, constraints are used to refer to prior knowledge later. $\boldsymbol{C}$ is used in this article to refer to a set of constraints representing edges or paths, which are expressed as follows:

- $X \to Y$ means X is the parent node of Y. $X \to Y$ means X cannot be the parent node of Y. $edge(X, Y)$ is used to express any edge constraint between X and Y;

- $X \Rightarrow Y$ means X is the ancestor node of Y. $X \Rightarrow Y$ means X cannot be an ancestor node of Y. In these two cases, Y is called a tail node and is a head node. $path(X, Y)$ is used to express any X path constraint between X and Y;
- Suppose there is an arbitrary node order o and constraint set $\boldsymbol{C}$ in which o and $\boldsymbol{C}$ are consistent. If and only if for arbitrary $\{X_1, X_2\} \subset (vari(o) \cap vari(\boldsymbol{C}))$, there is $X_1 \prec X_2$ in o, then X_2 cannot be an ancestor node of X_1 in $\boldsymbol{C}$.

This paper uses constraints in the following two steps: (1) to limit the construction process of the node order graph. Specifically, some illegal nodes are deleted from the node order graph, which can reduce the complexity, especially the space complexity. (2) A sparse parent node graph and query algorithm are constructed, so that the results of the optimal parent node query can satisfy the constraints. Theorem 2 is given as the basis for realizing constraints.

Theorem 2. *In a given set of constraints on edges or paths* $edge(X_1, Y_1) \in \boldsymbol{C}$, $path(X_2, Y_2) \in \boldsymbol{C}$, *for any optimal substructure* G *in the dynamic programming process, if* $\{X_1, Y_1\} \subset vari(G)$, *then there must be* $edge(X_1, Y_1) \in G$. *Similarly, if* $\{X_2, Y_2\} \subset vari(G)$, *then there must be* $path(X_2, Y_2) \in G$.

Proof of Theorem 2. If $edge(X_1, Y_1) \in \boldsymbol{C}$ and $\{X_1, Y_1\} \subset vari(G)$, and if there is no $edge(X_1, Y_1)$ in G, then due to the non-aftereffect property of the dynamic programming method, all the extended structures G will not satisfy constraints $\boldsymbol{C}$, so there must be $edge(X_1, Y_1)$ in G. $path(X_2, Y_2) \in \boldsymbol{C}$ can also be proven in the same way. The proof is completed. □

3.3. Integrating Constraints of Edge

3.3.1. Pruning Node Order Graph

With the given constraints of edges $X_1 \rightarrow X_2$, node $\{X_2\}$ in the node order graph needs to be deleted because it violates the constraint: the structures produced by the optimal substructure of this node all satisfy the node order $X_2 \prec X_1$ which is obviously inconsistent with constraints $X_1 \prec X_2$, so it is unnecessary to calculate node X_2 when constructing the node order graph. As can be seen from the above example, if a node in the node order graph violates a constraint, it needs to be deleted. The theorem is given as follows:

Theorem 3. *In a given constraint set* $\boldsymbol{C}$, *there is a node* $\boldsymbol{U}$ *and its set of node order* $\boldsymbol{o}_U$ *in the node order graph, then* $\boldsymbol{U}$ *needs to be deleted from the node order graph if and only if there is such a* $\{X_1, Y_1\} \subset vari(\boldsymbol{C})$, *satisfying the condition that* $X_1 \Rightarrow X_2$ *can be inferred from* $\boldsymbol{C}$ *and there are* $X_2 \in \boldsymbol{U}$, $X_1 \notin \boldsymbol{U}$.

Proof of Theorem 3. In subsequent nodes of any, $\boldsymbol{U}$, X_1 is added as a leaf node, so obviously for any $o \in \boldsymbol{o}_U$, there is $X_2 \prec X_1$, which is inconsistent with $X_1 \Rightarrow X_2$. Suppose for any $X_1 \Rightarrow X_2$ relationship in $\boldsymbol{C}$, there is no $X_2 \in \boldsymbol{U}$, $X_1 \notin \boldsymbol{U}$, then o_1 in $vari(o_1) = \boldsymbol{U}$ is consistent with $\boldsymbol{C}$, and o_2 in $vari(o_2) = \boldsymbol{V} \backslash \boldsymbol{U}$ is consistent with $\boldsymbol{C}$. Moreover, for any $X_1 \Rightarrow X_2$, there is no $X_2 \in vari(o_1)$ or $X_1 \in vari(o_2)$. So o made up of o_1 and o_2 is consistent with $\boldsymbol{C}$. The proof is completed. □

Theorem 4. *In a given edge constraint set* $\boldsymbol{C}$ *and the variable set* $\boldsymbol{V}$ *of the problem domain, when traversing to any node* $\boldsymbol{U}$ *during the construction of node order graph, make* $G_s = sub(G_C, vari(\boldsymbol{C}) \backslash \boldsymbol{U})$. *If the resulting new node* $\boldsymbol{U} \cup X$ *of* $\boldsymbol{U}$ *must satisfy* $X \in (\boldsymbol{V} \backslash vari(\boldsymbol{C}) \cup root(G_s))$, *then all the constructed nodes in the last node order graph satisfy the constraint* $\boldsymbol{C}$ *and all deleted nodes violate the constraint* $\boldsymbol{C}$.

Proof of Theorem 4. The node order graph is constructed from an empty set, which satisfies the constraint $\boldsymbol{C}$. At this time, no node in the node order graph is deleted. When traversing to any node $\boldsymbol{U}$, suppose all the existing nodes in the node order graph satisfy $\boldsymbol{C}$ and all the deleted nodes violate $\boldsymbol{C}$. Then, it is necessary to prove that the new nodes constructed in the node order graph satisfy the constraints and the deleted nodes violate the constraints.

The newly constructed nodes satisfy the constraints: any constructed new node is $\boldsymbol{U} \cup X$, in which $X \in (\boldsymbol{V} \backslash vari(\boldsymbol{C}) \cup root(G_s))$ and $G_s = sub(G_C, vari(\boldsymbol{C}) \backslash \boldsymbol{U})$. If $X \in (\boldsymbol{V} \backslash vari(\boldsymbol{C}))X_1, \ldots, X_n$, because $\boldsymbol{U}$ satisfies the constraint, and the new variable X has nothing to do with constraint $\boldsymbol{C}$, then obviously $\boldsymbol{U} \cup X$ also satisfies the constraint. If $X \in (root(sub(G_C, vari(\boldsymbol{C}) \backslash \boldsymbol{U})))$, the newly added variable X is the remaining variable in $vari(\boldsymbol{C})$ and it is the root node in the subgraph of the remaining variables of the edge constraint graph. So there is no such ancestor node $Y \in vari(\boldsymbol{C}) \backslash (\boldsymbol{U} \cup X)$ of X in the remaining nodes, causing there to be $Y \Rightarrow X$ in $\boldsymbol{C}$. Therefore, according to Theorem 3, any newly added node satisfies the constraints.

No deleted node satisfies the constraints. Suppose $\boldsymbol{H}$ is deleted: $\boldsymbol{H}$ can be remade by combining $\boldsymbol{H} \backslash X$ with X, with X as any variable in $\boldsymbol{H}$. If there is $\boldsymbol{H} \backslash X$ and $\boldsymbol{H} \backslash X$ is the node that satisfies the constraint, then X is a non-root node in the corresponding subgraph, but in this subgraph, there must be a corresponding root node Y. There is $Y \Rightarrow X$ and $Y \notin \boldsymbol{H}$, so according to Theorem 3, $\boldsymbol{H}$ does not satisfy the constraint. When there is no arbitrary X to make $\boldsymbol{H} \backslash X$ satisfy the constraint and if $\boldsymbol{H}$ satisfies the constraint, according to Theorem 3, there is a Y in $\boldsymbol{H}$ and $X \Rightarrow Y$. Then, a node order of constraint $\boldsymbol{C}$, i.e., $X \prec Y \prec \ldots \prec Z$, can be constructed by using $\boldsymbol{H} \cap vari(\boldsymbol{C})$. Take the last variable Z: if Z does not exist, then $Z = Y$ and $\boldsymbol{H} \backslash Z$ satisfy the constraint, which contradicts the condition. Therefore, the hypothesis is invalid, which proves that the arbitrarily deleted node does not satisfy the constraints. The proof is completed. □

Theorem 3 provides the basis for pruning the node order graph. The most direct way for the pruning is to make judgments on each node $\boldsymbol{U}$ so that they satisfy Theorem 3. However, even the simplified judgment algorithms cannot perform with the best efficiency. Therefore, Theorem 4 proposes a method to construct a node order graph, so that all nodes in the graph satisfy the constraint. According to the method in Theorem 4, we can make full use of the constraint to prune the node order graph, reduce the space complexity, and obtain the optimal structure after the node order graph is constructed.

As the score of sets in the node order graph needs to be queried repeatedly, in order to increase the efficiency of sets querying, this paper designs the hash function of sets in which different sets correspond to different hash values. The hash function is designed as follows: Suppose the set of all variable in the problem domain is $\{X_1, \ldots, X_n\}$. Set binary number b with the number of digits as n. For a set $\boldsymbol{U}$ in the node order graph, if $X_i \in \boldsymbol{U}$, then set the ith place of b to 1, otherwise set it to 0. Finally, convert b to decimal which will be the corresponding hash value.

The specific algorithm flow of the node order graph construction is shown in Algorithm 1.

3.3.2. Construction and Query of Sparse Parent Node Graph

The construction algorithm of the sparse parent node graph is as follows: As the sparse parent node graph stores the information of the first n_{mp} layers of nodes in the complete parent node graph, we first construct a complete parent node graph based on the constraint according to Theorem 4, and then store it in the sparse parent node graph every time a node is constructed. When the first n_{mp} layers are completely constructed, the sparse parent node graph will be obtained. Here is an example to illustrate how to construct a complete parent node graph based on this constraint.

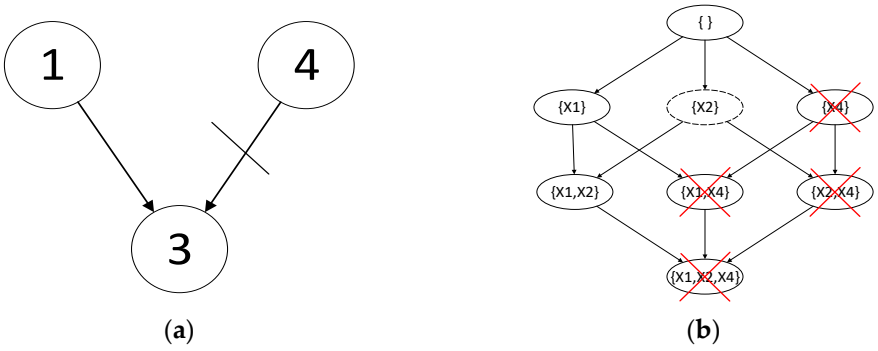
Algorithm 1. Construction algorithm of node order graph.

Constructing Node Order Graph Based on Edge Constraint

Input: SPG -Sparse parent node graph, G_C -edge constraint graph
Output: G-Global optimal structure

$PreviousLayer.HashTable \leftarrow \varnothing$, $PreviousLayer[\varnothing].score \leftarrow 0$, $\boldsymbol{V_C} = vari(\boldsymbol{C})$
for $Layer \leftarrow 1$ to n **do**
  for each node $\boldsymbol{U}$ In the $PreviousLyaer$ **do**
    $\boldsymbol{V}_r \leftarrow \boldsymbol{V}_c \backslash \boldsymbol{U}$
    $G_r \leftarrow$ Removing variables of $\boldsymbol{V}_r$ And their relative arcs in G_c
    $\boldsymbol{R} \leftarrow$ Root variables of G_r
      for each variable $X \in (\boldsymbol{V} \backslash (\boldsymbol{U} \cup \boldsymbol{V}_r)) \cup \boldsymbol{R}$ **do**
        $[bestparents\ bestscore] \leftarrow GetBestScore(X, \boldsymbol{U}, \boldsymbol{C}, SPG)$
       $curscore \leftarrow \boldsymbol{U}.score + bestscore$
          if $NewLayer[\boldsymbol{U} \cup X]$ Is null
           $NewLayer[\boldsymbol{U} \cup X] \leftarrow [curscore, parents, HashTable]$
          else if $curscore > NewLayer[\boldsymbol{U} \cup X].score$ then
           $NewLayer[\boldsymbol{U} \cup X] \leftarrow [curscore, parents, HashTable]$
          end if
        end for
      end for
      $PreviousLayer \leftarrow NewLayer$
  end for
$G \leftarrow$ *querying* $PreviousLayer.HashTable$

Figure 2 gives some constraints of variables. To calculate a node of X_3 in the graph, such as the node $\{X_1, X_2\}$, we only need to compare the scores of $\{X_1\}$ and $\{X_2\}$, i.e.,$score_{\{X_1\}}$ $score_{\{X_2\}}$ and $score(X_3, \{X_1, X_2\})$ when there is no constraint. Moreover, because X_1 is the parent node in the constraint, only $score_{\{X_1\}}$,$score(X_3, \{X_1, X_2\})$ are compared now. In addition, the crossed nodes in Figure 2b do not need to be solved because these nodes contain variables X_4 and X_4 is by no means the parent node of X_3.



(**a**) (**b**)

Figure 2. Construction of complete parent node graph of X_3 with given edge constraint: (**a**) Edge constraint; (**b**) Construction of complete parent node graph of X_3.

Based on the above ideas, the specific algorithm flow of the sparse parent node graph construction, defined as **PBDP-EDGE,** is shown in Algorithm 2.

The query algorithm idea of the sparse parent node graph is as follows: Suppose δ is a query constraint of X, that is, all the possible $pa(X)$ in $\boldsymbol{U}$ must satisfy the constraint δ. In other words, the front set of $parents_X$ that satisfy δ is the best parent node set in $\boldsymbol{U}$. Furthermore, query constraint δ must satisfy the following two conditions: (1) $\boldsymbol{Y} \subset \boldsymbol{U}$, (2) $\boldsymbol{CPa}(X) \cap \boldsymbol{U} \in \boldsymbol{Y}$, $\boldsymbol{Y} \cap \boldsymbol{CNPa}(X) = \varnothing$, in which $\boldsymbol{CPa}(X)$ means it is the set of parent nodes of X and $\boldsymbol{CNPa}(X)$ means it is not.

Algorithm 2. Construction algorithm of sparse parent node graph.

Constructing Sparse Parent Node Graph Based on Edge Constraint

Input: $\boldsymbol{V}$-set of all variables,$\boldsymbol{C}$-set of constraints,$score(.,.)$-decomposable score function value
Output: SPG-Sparse parent node graph

for $X \in \boldsymbol{V}$ **do**
  $[score_X, parents_X] \leftarrow \varnothing$
  for $layer \leftarrow 0$ to n **do**
    for each node $\boldsymbol{U}$ Such that $\boldsymbol{U} \in \boldsymbol{V}\backslash\{X\}\&|\boldsymbol{U}| == layer$ **do**
      $BestScore(X, \boldsymbol{U}) = \max_{Y \in P, Y \notin Pa(X)} BestScore(X, \boldsymbol{U}\backslash\{Y\})$
      if $\boldsymbol{U} \cap noPa(X) == \varnothing \&\& score(X, \boldsymbol{U}) > BestScore(X, \boldsymbol{U})$
        $BestScore(X, \boldsymbol{U}) \leftarrow score(X, \boldsymbol{U})$
        Append $[score_X, parents_X]$ with $[BestScore(X, \boldsymbol{U}), bitnarize(\boldsymbol{U})|$
      end if
    end for
  end for
  Sort $[score_X, parents_X]$ with $score_X$ In descending
end for
return $SPG \leftarrow$ [$score.$, $patrnts.$]

The specific implementation is as follows: First, set a bit array $valid_X$ of all 1s and with the same length as $parents_X$. Then, according to the first condition in δ, first do $valid_X\& \sim parents_X^{X_i}$ for each X_i that satisfies $X_i \in \boldsymbol{V}\backslash\boldsymbol{U}$. Then, according to the second condition in δ, do $valid_X\& \sim parents_X^{X_i}$ for each X_i that satisfies $X_i \in \boldsymbol{CPa}(X) \cap \boldsymbol{U}$. The purpose of this step is to ensure that all the remaining sets include all the variables in $\boldsymbol{CPa}(X) \cap \boldsymbol{U}$. Finally, conduct $valid_X\& \sim parents_X^{X_i}$ for each X_i that satisfies $X_i \in \boldsymbol{CNPa}(X) \cap \boldsymbol{U}$. This step eliminates the sets which include variables in any $\boldsymbol{CNPa}(X) \cap \boldsymbol{U}$, and the front set in the remaining sets is the best parent node set. Algorithm 3 shows the specific algorithm of the best parent node set query.

Algorithm 3. Query algorithm of best parent node set.

The Optimal Parent Node Set Based on Query Constraints

Input: $\boldsymbol{V}$-set of all variables, $\boldsymbol{C}$-set of constraints, SPG-sparse parent node graph
Output: $bestsparents(.,.)$-The best parent node set, $bestscore(.,.)$-The corresponding score

$valid \leftarrow allScores_X$
for each $Y_1 \in Pa(X) \cap \boldsymbol{U}$ **do**
  $valid \leftarrow valid\&parents_X^{Y_1}$
end for
for each $Y_2 \in (\boldsymbol{V}\backslash\boldsymbol{U} \cup noPa(X))\backslash Pa(X)$ **do**
  $valid \leftarrow valid\& \sim parents_X^{Y_2}$
end for
$index \leftarrow firstSetBit(valid)$
return $scrore_X[index]$, $parents_X[index]$

Here is an example to illustrate the implementation process. As shown in Table 1, from $\{X_1, X_2, X_4, X_5\}$, find the best parent node sets of X_3, $\boldsymbol{CPa}(X_3) = \{X_1, X_2\}$ and $\boldsymbol{CNPa}(X_3) = \{X_4\}$. At this point, the remaining candidate sets in the table all satisfy the first condition of δ, that is, they all are subsets of $\{X_1, X_2, X_4, X_5\}$. If there is no constraint, $\{X_1, X_5\}$ will be the best parent node set. Next, we need to realize the second condition. Because $\{X_1, X_2\} = \boldsymbol{CPa}(X) \cap \boldsymbol{U}$, do $parents_X^{X_1} \& parents_X^{X_2}$. The result is shown in the seventh row of the table, in which a value of 1 means that all sets contain $\{X_1, X_2\}$. Because $\boldsymbol{CNPa}(X_3) = \{X_4\}$, find $\sim parents_X^{X_4}$. The result is shown in the eighth row of the table in which the value of 1 means none of them contains X_4. Finally, sum the seventh and the

eighth row to obtain the final $valid_X$. The first set $\{X_1, X_2, X_5\}$ is the best parent node set satisfying the constraint.

Table 1. Example of query process based on constraints.

1	$parents_{X_3}$	$\{X_1, X_5\}$	$\{X_1, X_2, X_4\}$	$\{X_1, X_2, X_5\}$	$\{\}$	$\{X_1\}$
2	$scores_{X_3}$	5	4	3	2	1
3	$parents_{X_3}^{X_1}$	1	1	1	0	1
4	$parents_{X_3}^{X_2}$	0	1	1	0	0
5	$parents_{X_3}^{X_4}$	0	1	0	0	0
6	$parents_{X_3}^{X_5}$	1	0	1	0	0
7	Operation of $CPa(X_3)$	0	1	1	0	0
8	Operation of $CNPa(X_3)$	1	0	1	1	1
9	*valid*	0	0	1	0	0

3.4. Integrating Path Constraints

3.4.1. Pruning Node Order Graph

The algorithm of the pruning node order graph by path constraint is the same as that by edge constraint. First, construct a constraint graph G_C, then use the algorithm in Table 1 to prune the node order graph, wherein path constraint graph G_C of the constraint set $\boldsymbol{C}$ is a directed acyclic graph containing variable $vari(\boldsymbol{C})$, and for arbitrary $\{X_1, X_2\} \subset vari(\boldsymbol{C})$, there is an edge $X_1 \to X_2$ between X_1, X_2 in G_C, if and only if $(X_1 \Rightarrow X_2) \in \boldsymbol{C}$.

3.4.2. Construction and Query of Sparse Parent Node Graph

As the parent node of Y must contain at least one X or one descendant node of X, when constructing the sparse parent node graph, for Y, it is necessary to store all the parent node sets with the number of variables below n_{mp}. For other variables, the sparse parent node graph is constructed according to the unconstrained condition. The specific construction algorithm of the sparse parent node graph, defined as **PBDP-PATH,** is shown in Algorithm 4.

The query algorithm idea of path constraint is as follows: For a given path constraint $X \Rightarrow Y$, to find the best parent node set $\boldsymbol{S}$ of Y in $\boldsymbol{U}$, if $X \in \boldsymbol{U}$, there is at least one $Z \in \{X \cup des(X)\}$ to make $Z \in \boldsymbol{S}$. $des(X)$ is the descendant node of X in the structure of $\boldsymbol{U}$. If $X \Rightarrow Y$, then there is $Z \notin \boldsymbol{S}$ for all $Z \in \{X \cup des(X)\}$.

The specific query method is as follows: Initialize a bit array $valid_X$ of all 1s and with the same length as $parents_X$. Conduct $valid_X \& \sim parents_X^{X_k}$ for each X_k that satisfies $X_k \in \boldsymbol{V} \backslash \boldsymbol{U}$. For each X_i, set an auxiliary bit array $Cvalid$ of all ones and find the descendant node $des(X_i)$ of X_i. For each $Z \in \{X_i \cup des(X_i)\}$, perform the OR operation $Cvalid \big| parents_X^Z$. Finally, perform the AND operation $valid \leftarrow valid \& Cvalid$. For each X_j, find $des(X_j)$. For each $Z \in \{X_i \cup des(X_i)\}$, perform the AND operation $valid \leftarrow valid \ \& \ parents_X^Z$. Algorithm 5 shows the specific algorithm flow of the best parent node set query.

An example is given below to illustrate the implementation process.

Figure 3 is an example of path constraints, in which $\boldsymbol{C}$ is $X_1 \Rightarrow Y$ and $X_2 \Rightarrow Y$. At this point, we need to find the best parent node set S of Y from $\boldsymbol{U} = \{X_1, X_2, X_3, X_4\}$. Table 2 shows the specific solution process. In the table, parent node sets are selected with part of them as the subset of $\boldsymbol{U}$, so the first condition of δ has been satisfied. At this point, if there is no constraint, $\{X_2, X_4\}$ will be the best parent node set. When a constraint is given, perform the OR operation for the line where the elements of $\{desX_1 \cup X_1\}$ are and obtain $Cvalid_1$, as in line 7. Perform OR operation for the line where the elements of $\{desX_2 \cup X_2\}$ are and obtain $Cvalid_2$, as in line 8. Then, perform the AND operation

$valid \leftarrow valid\&Cvalid_1\&Cvalid_2$. At this time, *valid* equals 1, which shows that there are elements both from $\{desX_1 \cup X_1\}$ and $\{desX_2 \cup X_2\}$; therefore $\{X_3, X_4\}$ is the best solution at this time.

Algorithm 4. Construction algorithm of sparse parent node graph.

Constructing Sparse Parent Node Graph Based on Path Constraints

Input: $\boldsymbol{V}$-set of all variables,$\boldsymbol{C}$-set of constraints,$score(.,.)$-decomposable score function value
Output:SPG-sparse parent node graph

```
1.for X ∈ V do
2.   if X ∈ end(C) do
3.      Construct Full Sparse Parent Graph (V, X, score(.,.))
4.   else do
5.      Construct Sparse Parent Graph without Constraints (V, X, score(.,.))
6.   end if
7. end for
8. return SPG ← [score., parents.]
9. Function Construct Sparse Parent Graph without Constraints (V, X, score(.,.))
10.   [score_X, parents_X ← ∅]
11.   for layer ← 0 to n do
12.      for each node P such that P ∈ V\{X}&|P| == layer do
13.      BestScore(X, P) = max_{Y∈P} BestScore(X, P\{Y})
14.      if score(X, P) > BestScore(X, P)
15.         BestScore(X, P) ← score(X, P)
16.         append [score_X, parents_X] with [Score(X, P), bitnarize(P)]
17.      end if
18.     end for
19.   end for
20.   sort with score_X in descending
21.   return [score_X, parents_X]
22. end function
23. Function Construct Full Sparse Parent Graph (V, X, score(.,.))
24.         [score_X, parents_X] ← ∅
25.         for each P ∈ V\{X} do
26.            Append [score_X, parents_X] with [Score(X, P), bitnarize(P)]
27.         end for
28.         sort with score_X in descending
29.         return [score_X, parents_X]
30. end function
```

Table 2. Example of query process based on path constraint.

1	$parents_Y$	$\{X_2, X_4\}$	$\{X_4\}$	$\{X_3, X_4\}$	$\{\}$	$\{X_3\}$
2	$scores_Y$	5	4	3	2	1
3	$parents_Y^{X_1}$	0	0	0	0	0
4	$parents_Y^{X_2}$	1	0	0	0	0
5	$parents_Y^{X_3}$	0	0	1	0	1
6	$parents_Y^{X_4}$	1	1	1	0	0
7	*Cvalid* of $\{desX_1 \cup X_1\}$	0	0	1	0	1
8	*Cvalid* of $\{desX_2 \cup X_2\}$	1	1	1	0	1
9	*valid*	0	0	1	0	1

Algorithm 5. Query algorithm of best parent node set.

The Best Parent Node Set Based on the Path Constraint Query

Input: ***V***-set of all variables, ***C*** set of path constraints, *SPG*-sparse parent node graph
Output: $bestsparents(.,.)$-the best parent node set, $bestsCore(.,.)$-the corresponding score

1. $valid \leftarrow allScores_X$
2. **for** each **do**
3. $\quad valid \leftarrow valid\& \sim parents_X^{Y_i}$
4. **end for**
5. **for** each Y_j such that $\left(Y_j \Rightarrow X\right) \in \boldsymbol{C}$ **do**
6. $\quad Cvalid \leftarrow allScores_X$
7. $\quad$ **for** each S Holding that $Y_j \Rightarrow S$ in G **do**
8. $\quad\quad Cvalid \leftarrow Cvalid \left| parents_X^S \right.$
9. $\quad\quad Cvalid \leftarrow Cvalid \left| parents_X^{Y_i} \right.$
10. $\quad$ **end for**
11. $\quad valid \leftarrow valid\&Cvalid$
12. **end for**
13. **for** each Y_k such that $(Y_k \Rightarrow X) \in \boldsymbol{C}$ **do**
14. $\quad$ **for** each S Holding that $Y_k \Rightarrow S$ in G **do**
15. $\quad\quad valid \leftarrow valid\& \sim parents_X^S$
16. $\quad$ **end for**
17. **end for**
18. $index \leftarrow firstSetBit(valid)$
19. **return** $scrores_X[index]$

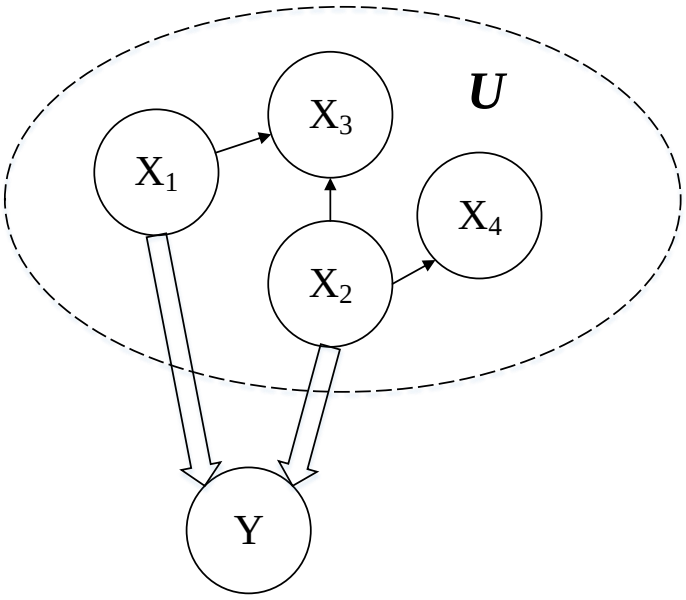


Figure 3. Example of path constraints.

4. Algorithm Simulation and Analysis

4.1. Validity Verification

In this section, in order to verify the effectiveness, first, an 18-node network is generated by using Matlab constructor. Then, the constructed network, Asia network, and Sachs network are simulated and verified with 20 samples. In order to verify that this method can really integrate constraints, some extreme simulation conditions are set.

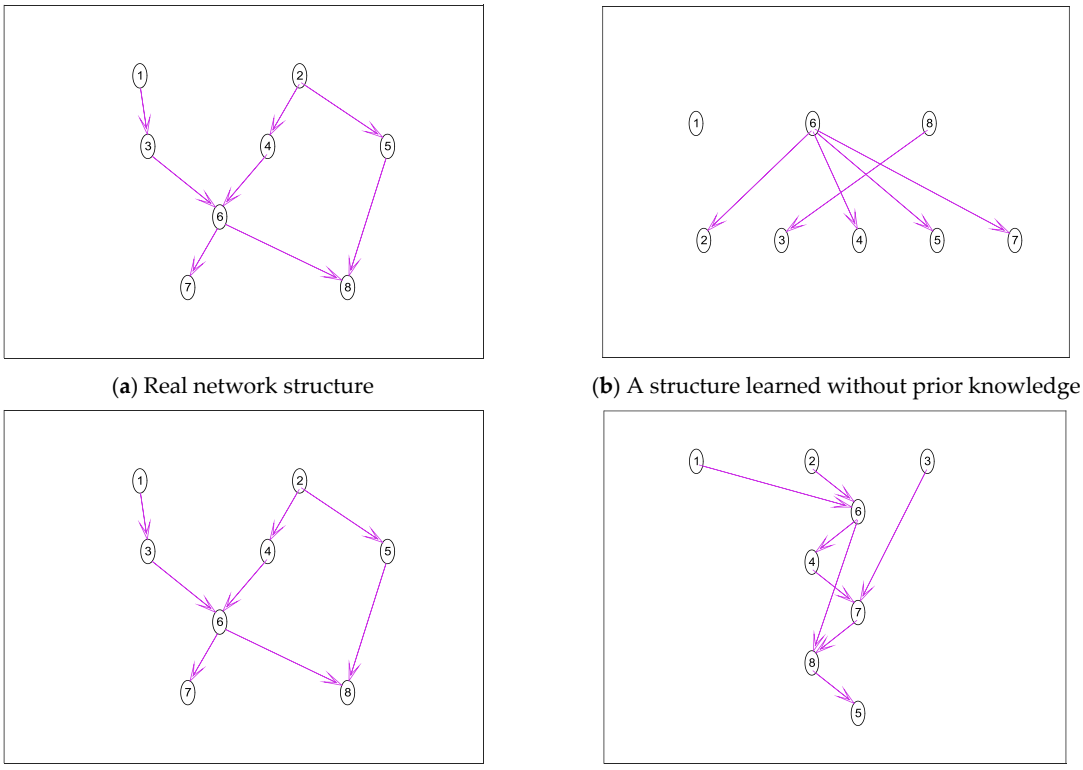
1. The simulation is carried out with the Asia network. All the edge prior knowledge is given, which is verified by the PBDP-EDGE structure. Part of the path prior knowledge is given, specifically $1 \Rightarrow 6$, $2 \Rightarrow 6$, $2 \Rightarrow 8$, $3 \Rightarrow 7$, $3 \Rightarrow 8$, $4 \Rightarrow 7$, and $4 \Rightarrow 8$, which is verified by the PBDP-PATH structure. The results are shown in Figure 4.

The real network structure of the Asia network is shown in Figure 4a. It can be seen from Figure 4b that training samples contain very little information and can only learn a few edges, and a complete structure cannot be constructed. It can be seen from Figure 4c that the correct structure can be learned even if the sample size is small, which demonstrates the correctness and effectiveness of the integrating edge prior-knowledge algorithm proposed in this paper. It can be seen from Figure 4d that it is obvious that all the learned structures contain these paths ($1 \Rightarrow 6$, $2 \Rightarrow 6$, $2 \Rightarrow 8$, $3 \Rightarrow 7$, $3 \Rightarrow 8$, $4 \Rightarrow 7$, and $4 \Rightarrow 8$).

2. The simulation is carried out with the Sachs network. Part of the edge prior knowledge is given, which is verified by the PBDP-EDGE structure. Part of the path prior knowledge is given, specifically $1 \Rightarrow 2$, $1 \Rightarrow 4$, $1 \Rightarrow 5$, $1 \Rightarrow 7$, $1 \Rightarrow 8$, $2 \Rightarrow 5$, $2 \Rightarrow 8$, $3 \Rightarrow 4$, $4 \Rightarrow 6$, $5 \Rightarrow 6$, and $9 \Rightarrow 11$, which is verified by the PBDP-PATH structure. The results are shown in Figure 5.

The real network structure of the Sachs network is shown in Figure 5a. As can be seen from Figure 5b, the training sample contains little information, only a few edges can be learned, and a complete structure cannot be constructed. It can be seen from Figure 5c that partial correct structures can be learned even if the sample size is small, indicating the correctness and effectiveness of integrating the edge prior-knowledge algorithm proposed in this paper. It can be seen from Figure 5d that it is obvious that all the learned structures contain these paths ($1 \Rightarrow 2$, $1 \Rightarrow 4$, $1 \Rightarrow 5$, $1 \Rightarrow 7$, $1 \Rightarrow 8$, $2 \Rightarrow 5$, $2 \Rightarrow 8$, $3 \Rightarrow 4$, $4 \Rightarrow 6$, $5 \Rightarrow 6$, and $9 \Rightarrow 11$).

3. The simulation is carried out with the Constructed network. Part of the edge prior knowledge is given, which is verified by the PBDP-EDGE structure. Part of the path prior knowledge is given, specifically $1 \Rightarrow 4$, $1 \Rightarrow 17$, $2 \Rightarrow 18$, $2 \Rightarrow 13$, $2 \Rightarrow 5$, $3 \Rightarrow 5$, $3 \Rightarrow 9$, $6 \Rightarrow 8$, $5 \Rightarrow 10$, $5 \Rightarrow 12$, $7 \Rightarrow 5$, $10 \Rightarrow 13$, $13 \Rightarrow 9$, and $15 \Rightarrow 10$, which is verified by the PBDP-PATH structure. The results are shown in Figure 6.



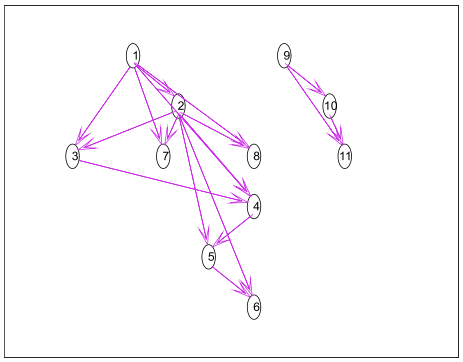
(**a**) Real network structure

(**b**) A structure learned without prior knowledge

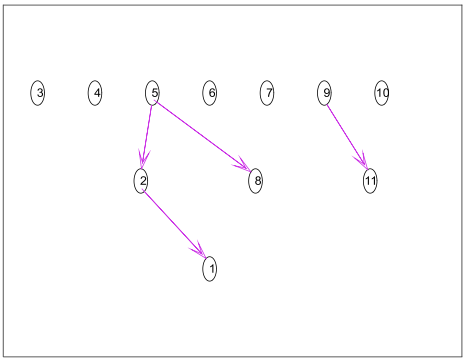
(**c**) The PBDP-EDGE structure

(**d**) The PBDP-PATH structure

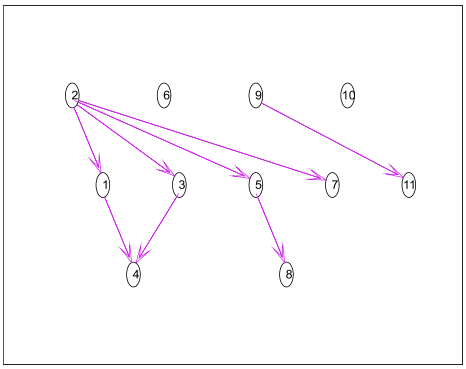
Figure 4. Asia network structure simulation diagram.



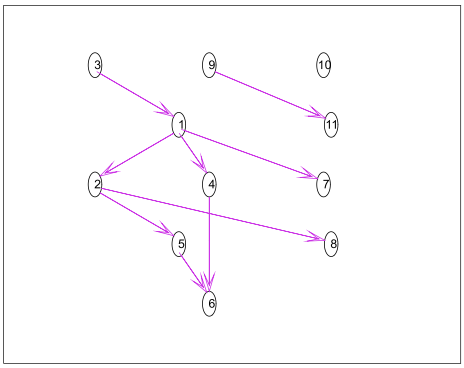
(**a**) Real network structure



(**b**) A structure learned without prior knowledge

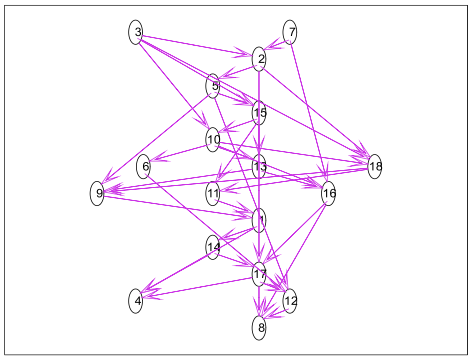


(**c**) The PBDP-EDGE structure

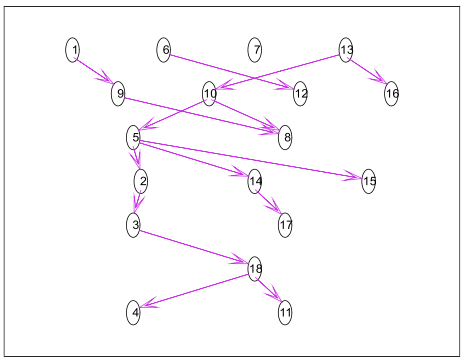


(**d**) The PBDP-PATH structure

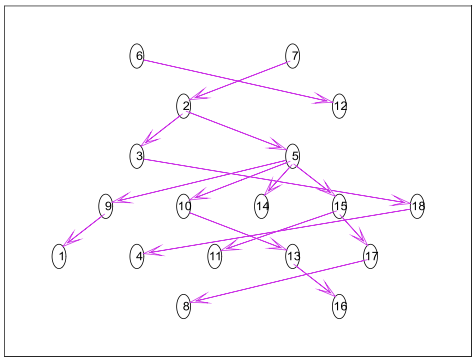
Figure 5. Sachs network structure simulation diagram.



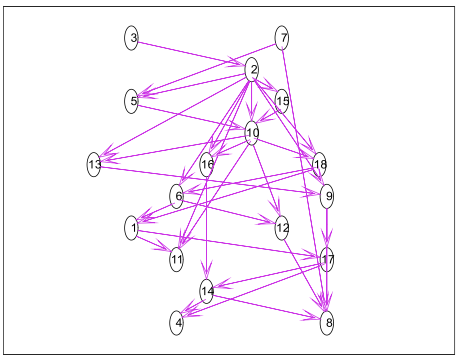
(**a**) Real network structure



(**b**) A structure learned without prior knowledge



(**c**) The PBDP-EDGE structure



(**d**) The PBDP-PATH structure

Figure 6. Constructed network structure simulation diagram.

The real structure of the Constructed network is shown in Figure 6a. As can be seen from Figure 6b, the training samples contain little information, only a few edges can be learned, and a complete structure cannot be constructed. It can be seen from Figure 6c that partial correct structures can be learned even if the sample size is small, indicating the correctness and effectiveness of the integrating constraints of the edge algorithm proposed in this paper. It can be seen from Figure 6d that it is obvious that all the learned structures contain these paths ($1 \Rightarrow 4$, $1 \Rightarrow 17$, $2 \Rightarrow 18$, $2 \Rightarrow 13$, $2 \Rightarrow 5$, $3 \Rightarrow 5$, $3 \Rightarrow 9$, $6 \Rightarrow 8$, $5 \Rightarrow 10$, $5 \Rightarrow 12$, $7 \Rightarrow 5$, $10 \Rightarrow 13$, $13 \Rightarrow 9$, and $15 \Rightarrow 10$).

Therefore, the above simulation results can prove that the method proposed in this paper is correct and reliable and can be realized no matter what kind of prior knowledge is given.

4.2. Complexity Verification

- The integrating edge constraint is simulated by the Halifinder network, a large-scaled network, and half of the real edges are randomly selected as prior knowledge. The training sample size is 200, 500, and 1000, respectively. Table 3 shows the simulation results. PBDP (Priors Based DP) indicates the integrating prior-knowledge method, which is measured in seconds. The space cost refers to the size of the array to be set, and the proportion represents the time and space ratio between the PBDP method and DP method.

  The path constraint is simulated in the same way, with the results shown in Table 4.

Table 3. Simulation comparison of integrating constraints of edge.

Sample Size	Approach	PBDP-EDGE	DP	Proportion
200	PIC Score	−670,352.217	−696,271.251	
	Runtime	3723.053	31,114.242	0.12
	Space	52,223	262143	0.199
500	PIC Score	−629,520.727	−635,543.295	
	Runtime	3141.870	31,925.566	0.098
	Space	52,223	262143	0.199
1000	PIC Score	−629,520.727	−630,672.929	
	Runtime	3218.295	30,909.447	0.104
	Space	52,223	262,143	0.199

Table 4. Simulation comparison of integrating path constraints.

Sample Size	Approach	PBDP-PATH	DP	Proportion
200	PIC Score	−654,151.15	−684,005.61	
	Runtime	5486.113	29,359.588	0.187
	Space	44,159	262,143	0.168
500	PIC Score	−633,710.86	−637,161.52	
	Runtime	5035.867	29,785.541	0.169
	Space	44,159	262,143	0.168
1000	PIC Score	−629,561.51	−630,698.96	
	Runtime	5323.846	27,892.218	0.191
	Space	44,159	262,143	0.168

It can be seen from Tables 3 and 4 that the integrating edge constraint and path constraint can not only improve the scores, but also effectively reduce the complexity of time and space. To sum up, this method can use edge constraints and path constraints to effectively reduce the time and space complexity of the Dynamic Programming algorithm and improve its timeliness significantly.

5. Conclusions

In this paper, the specific process of dynamic planning is analyzed, and its restrictive relationship with edge constraints and path constraints is determined. The prior constraints are used to restrict and guide each link in dynamic planning, and deterministic prior knowledge is integrated into the dynamic planning of BN structure learning. The BN structure learning algorithm of dynamic planning integrating prior knowledge is proposed, and the specific implementation is described in detail. Simulation results show that this algorithm can use edge prior knowledge and path prior knowledge to effectively reduce the time and space complexity of the dynamic programming algorithm. It also reveals the complementary relationship between prior knowledge and learning in BN modeling, that is, only by making full use of prior knowledge and training sample information can an ideal model be obtained. This paper also provides some implications for the breaking through of the node number in the dynamic programming method.

Author Contributions: Conceptualization, Z.L.; methodology, R.D.; software, Y.C.; validation, H.W. and Z.L.; formal analysis, Y.C.; investigation, H.W. and Z.L.; resources, X.L.; data curation, X.S.; writing—original draft preparation, Y.C. and C.H.; writing—review and editing, R.D.; visualization, Y.C.; supervision, Z.L.; project administration, R.D.; funding acquisition, X.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Projects supported by the National Natural Science Foundation of China, grant number 62171360.

Data Availability Statement: Data Availability Statement: The true networks of all data sets are known, and they are publicly available (http://www.bnlearn.com/bnrepository (accessed on 15 June 2021)).

Acknowledgments: The authors are grateful to Wang and Ruohai Di teacher for helpful discussions. The authors also thank the anonymous reviewers for critical and constructive review of the manuscript. This work was supported by National Natural Science Foundation of China (62171360).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Pearl, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*; Morgan Kaufmann Press: San Mateo, CA, USA, 1988; pp. 1–15.
2. Bueno, M.L.; Hommersom, A.; Lucas, P.J.; Lappenschaar, M.; Janzing, J.G.E. Understanding disease processes by partitioned dynamic Bayesian networks. *J. Biomed. Inform.* **2016**, *61*, 283–297. [CrossRef] [PubMed]
3. Zhang, X.; Mahadevan, S. Bayesian network modeling of accident investigation reports for aviation safety assessment. *Relib. Eng. Syst. Safe* **2020**, *209*, 1–19. [CrossRef]
4. Wang, Z.; Wang, Z.; He, S.; Gu, X.; Yan, Z.F. Fault detection and diagnosis of chillers using Bayesian network merged distance rejection and multi-source non-sensor information. *Appl. Energy* **2017**, *188*, 200–214. [CrossRef]
5. Shenton, W.; Hart, B.; Chan, T. Bayesian network models for environmental flow decision-making: 1. *Latrobe River Australia. River Res. Appl.* **2015**, *27*, 283–296. [CrossRef]
6. Yu, K.; Liu, L.; Li, J.; Ding, W.; Le, T.D. Multi-Source Causal Feature Selection. *IEEE Trans. Pattern Anal. Mach. Learn.* **2020**, *42*, 2240–2256. [CrossRef] [PubMed]
7. McLachlan, S.; Dube, K.; Fenton, N. Bayesian Networks in Healthcare: Distribution by Medical Condition. *Artif. Intell. Med.* **2020**, *107*, 101912. [CrossRef] [PubMed]
8. Lin, C.; Liu, Y. Target Recognition and Behavior Prediction based on Bayesian Network. *Inter. J. Perform. Eng.* **2019**, *15*, 1014–1022. [CrossRef]
9. Sun, H.; Xie, X.; Sun, T.; Zhang, L. Threat assessment method for air defense targets of DBN fleet in the state of missing small sample data. *Syst. Eng. Electron.* **2019**, *41*, 1300–1308.
10. Di, R.; Gao, X.g.; Guo, Z. Small data set BN modeling method and its application in threat assessment. *Chin. J. Electron.* **2016**, *44*, 1504–1511.
11. Su, C.; Zhang, H. The introduction of human reliability in aircraft combat effectiveness evaluation. *Acta Aeron Sin.* **2006**, *27*, 262–266.
12. Wu, Y.; Ren, Z. Mission reliability analysis of multiple-phased systems based on Bayesian network. In Proceedings of the IEEE 2014 Prognostics and System Health Management Conference, Zhangjiajie, China, 24–27 August 2014.
13. Campos, C.; Ji, Q. Efficient Structure Learning of Bayesian Networks using Constraints. *J. Mach. Learn. Res.* **2011**, 663–689.

14. Cussens, J. Bayesian network learning with cutting planes. In Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, 14–17 July 2011.
15. Jaakkola, T.; Sontag, D.; Globerson, A.; Meila, M. Learning Bayesian Network Structure using LP Relaxations. *J. Mach. Learn. Res.* **2010**, *9*, 358–365.
16. Ott, S.; Imoto, S.; Miyano, S. Finding optimal models for small gene networks. *Pac. Symp. Biocomput.* **2004**, *9*, 557.
17. Koivisto, M.; Sood, K. Exact Bayesian Structure Discovery in Bayesian Networks. *J. Mach. Learn. Res.* **2004**, *5*, 549–573.
18. Angelopoulos, N.; Cussens, J. Bayesian learning of Bayesian networks with informative priors. *Ann. Math. Artif. Intel.* **2008**, *54*, 53–98. [CrossRef]
19. Zhu, M.; Liu, S.; Wang, C. An optimization method based on prior node order learning Bayesian network structure. *Acta Autom. Sin.* **2011**, *37*, 1514–1519.
20. Campos, L.; Castellanoa, J. Bayesian Network Learning Algorithms using Structural Restrictions. *Int. J. Approx. Reason.* **2007**, *45*, 233–254. [CrossRef]
21. Nicholson, D.; Han, B.; Korb, K.B.; Alam, M.J.; Hope, L.R. Incorporating expert elicited structural information in the CaMML Causal Discovery Program. 2008. Available online: https://bridges.monash.edu/articles/report/Incorporating_Expert_Elicited_Structural_Information_in_the_CaMML_Causal_Discovery_Program/20365395 (accessed on 9 August 2022).
22. Castelo, R.; Siebes, A. Priors on network structures. Biasing the search for Bayesian networks. *Int. J. Approx. Reason.* **2000**, *24*, 39–57. [CrossRef]
23. Borboudakis, G.; Tsamardinos, I. Scoring and searching over Bayesian networks with causal and associative priors. In Proceedings of the 29th International Conference on Uncertainty in Artificial Intelligence, Washington, DC, USA, 12–14 July 2013.
24. Parviainen, P.; Koivisto, M. Finding optimal Bayesian networks using precedence constraints. *J. Mach. Learn. Res.* **2013**, *14*, 1387–1415.
25. Chen, E.; Shen, Y.; Choi, A.; Darwiche, A. Learning Bayesian networks with ancestral constraints. In Proceedings of the 30th Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016.
26. Li, A.; Beek, P. Bayesian network structure learning with side constraints. In Proceedings of the 9th International Conference on Probabilistic Graphical Models, Prague, Czech, 11–14 September 2018.
27. Bartlett, M.; Cussens, J. Integer linear programming for the Bayesian network structure learning problem. *Artif. Intell.* **2017**, *244*, 258–271. [CrossRef]
28. Zhang, L.; Guo, H. *Introduction to Bayesian Nets*; Science Press: Beijing, China, 2006.
29. Malone, B.; Yuan, C.; Hansen, E. Memory-Efficient Dynamic Programming for Learning Optimal Bayesian Networks. In Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 7–11 August 2011.