mathematics

MDPI

Article

Bayesian Network Structure Learning Using Improved A* with Constraints from Potential Optimal Parent Sets

Chuchao He [1], Ruohai Di [1,*] and Xiangyuan Tan [2]

[1] School of Electronics and Information Engineering, Xi'an Technological University, Xi'an 710021, China; hechuchao@xatu.edu.cn
[2] School of Electronic Information, Northwestern Polytechnical University, Xi'an 710192, China; tanxy2017@mail.nwpu.edu.cn
* Correspondence: diruohai@xatu.edu.cn or xfwtdrh@163.com; Tel.: +86-13-72-0450874

Abstract: Learning the structure of a Bayesian network and considering the efficiency and accuracy of learning has always been a hot topic for researchers. This paper proposes two constraints to solve the problem that the A* algorithm, an exact learning algorithm, is not efficient enough to search larger networks. On the one hand, the parent–child set constraints reduce the number of potential optimal parent sets. On the other hand, the path constraints are obtained from the potential optimal parent sets to constrain the search process of the A* algorithm. Both constraints are proposed based on the potential optimal parent sets. Experiments show that the time efficiency of the A* algorithm can be significantly improved, and the ability of the A* algorithm to search larger Bayesian networks can be improved by the two constraints. In addition, compared with the globally optimal Bayesian network learning using integer linear programming (GOBNILP) algorithm and the max–min hill-climbing (MMHC) algorithm, which are state of the art, the A* algorithm enhanced by constraints still performs well in most cases.

Keywords: Bayesian network; structural learning; A* search; constraint; potential optimal parent sets

MSC: 68T30

Citation: He, C.; Di, R.; Tan, X. Bayesian Network Structure Learning Using Improved A* with Constraints from Potential Optimal Parent Sets. *Mathematics* **2023**, *11*, 3344. https://doi.org/10.3390/math11153344

Academic Editors: Zexun Chen and Bo Wang

Received: 4 July 2023
Revised: 27 July 2023
Accepted: 29 July 2023
Published: 30 July 2023

1. Introduction

Artificial intelligence has been widely used in reality after decades of development. However, capturing and understanding causal relationships from data, known as causal discovery, remains a challenging task in artificial intelligence. Robust causal analysis is widely recognized as a key driver of techniques such as learning, prediction, diagnosis, and counterfactual reasoning, which can have significant implications for almost all fields of science.

A Bayesian network (BN) is a probabilistic graphical model of the combination of probability theory and graph theory, and a BN can support the representation and analysis of causal structure in the field of artificial intelligence. The structure of the BN is a directed acyclic graph (DAG), which represents the dependence relationship between nodes. These relationships are further quantified by a set of conditional probability distributions. In general, a Bayesian network represents the joint probability distribution of a set of random variables.

The number of possible structures increases exponentially with the number of nodes n. Determining the BN structure from data is an NP-hard problem [1,2], and it has been a hot topic in the research field of BN in recent decades. Existing BN structure learning algorithms can be divided into three categories: constraint-based, score-based, and hybrid search approaches [3].

Constraint-based approaches use statistical tests or information theory techniques to test conditional independence (CI), determine the relationship between variables, and ob-

tain the corresponding DAG. Widely adopted constraint-based approaches mainly include the Peter–Clark (PC) algorithm [4], inductive causality (IC) algorithm [5], and grow–shrink (GS) algorithm [6]. These algorithms can return equivalence classes if CI tests are correct. However, such assumptions are difficult to satisfy in practice. The CI test is often affected by statistical tests, and there will be a certain probability of error in the case of insufficient samples or noise. Perhaps even worse, because a series of steps in the algorithm relies on CI tests, these erroneous CI test results can further amplify errors in the subsequent learning process.

Score-based approaches are the most common BN structure learning approaches. Score-based approaches use a scoring function to measure the fitness between the BN structure and data, and return the BN structure with the optimal score. Therefore, score-based approaches treat BN structure learning as a combinatorial optimization problem. From the perspective of combinatorial optimization, greedy search (GS) [7–9], simulated annealing (SA) [10], ordering-based search (OBS) [11], and other algorithms have developed rapidly in the early stage. In addition, the genetic algorithm (GA) [12], particle swarm optimization (PSO) [13], ant colony optimization (ACO) [14], and other swarm intelligence algorithms have also been widely used. However, these algorithms often obtain locally optimal network structures; in particular, swarm intelligence algorithms' convergence to the optimal solution in infinite time has no practical significance. Therefore, exact learning algorithms have begun to enter the field of vision of researchers. Research on exact learning algorithms began with a series of dynamic programming (DP) algorithms [15–17], which require exponential time and space complexity and can only be used for small-scale network structure learning. In recent years, other exact learning algorithms that are more competitive than DP algorithms have also been proposed, such as the A* [18,19], anytime window A* (AWA*) [20], Bidirectional heuristic search (BiHS) [21], CPBayes [22], and Globally Optimal Bayesian Network learning using Integer Linear Programming (GOBNILP) algorithms [23,24]. The A*, AWA*, and BiHS algorithms regard the structure learning problem as the shortest path search problem and use different strategies to search, and A* is the more stable algorithm among them. CPBayes and GOBNILP use constraint programming and integer programming, respectively, to solve the structure learning problem. Later, CPBayes was enhanced by including linear programming techniques to provide more efficient acyclicity checking [25]. Compared with DP algorithms, these algorithms improve the scalability and efficiency of learning BNs, but their efficiency is still relatively low, and there is still room for improvement. Furthermore, algorithms based on continuous optimization have been developed in recent years. For example, the noncombinatorial optimization via trace exponential and augmented Lagrangian for structure learning (NOTEARS) algorithm [26] used the augmented Lagrangian method to address continuous optimization problems.

Hybrid search approaches try to combine the advantages of score-based and constraint-based algorithms. The most famous hybrid search approach is the max–min hill-climbing (MMHC) algorithm [27]. This algorithm is divided into two stages: in the first stage, the max–min parent and children (MMPC) algorithm [28] is used to learn the parent–child set of each node; in the second stage, the hill-climbing algorithm is conducted within the limited range of the parent–child set. The constrained optimal search (COS) algorithm [29] performs an optimal search through the DP algorithm under superstructure constraints. The edge-constrained optimal search (ECOS) algorithm [30] clusters the superstructure based on the COS algorithm, learns in each cluster, and finally merges to obtain the complete BN structure. The constrained hill-climbing (CHC) algorithm [31] improves algorithm efficiency and accuracy by dynamically limiting the hill-climbing algorithm process. The separation and reunion (SAR) algorithm [32] decomposes a large BN into learning some relatively small BNs through the CI test and builds the actual network structure by remerging these smaller network structures. Kuipers et al. [33] proposed a hybrid algorithm that creates a restricted search space using the PC algorithm followed by Markov Chain Monte Carlo (MCMC) sampling.

In this paper, we continue to study the structural learning problem from the perspective of the shortest path search problem. This paper proposes an improved A* algorithm based on constraints from potential optimal parent sets (POPS), which is specifically improved in two aspects. On the one hand, the number of POPS is reduced through parent–child sets constraints; on the other hand, obtaining path constraints from POPS constrains the A* algorithm. The search efficiency of A* can be improved by the two aspects of constraints about POPS.

The remainder of this paper is organized as follows. In Section 2, the relevant theoretical basis of the problem is introduced and formulated. The details of the proposed algorithms are designed in Section 3. The proposed algorithms are demonstrated with experiments in Section 4, followed by conclusions in Section 5.

2. Preliminaries for Bayesian Networks

This section will introduce BN and the structure learning problem based on the shortest path search perspective, providing the theoretical basis for the new algorithm.

2.1. Bayesian Networks

The Bayesian network $BN = (G, P)$ consists of two parts, DAG G and probability distribution P. G is called the structure of a BN, in which each node corresponds to the variables in variable set $\boldsymbol{V} = \{X_1, \ldots, X_n\}$ one by one, and the directed edges between nodes reflect the dependencies between nodes. The probability distribution P is called the parameter of BN, specifically $P(X_i|PA_i)$, where PA_i represents the parent set of X_i. The joint probability of all variables can be decomposed into the product of conditional probability distributions. Figure 1 shows an example of DAG, and PA_2 is formed by $\{X_1\}$ and $\{\}$.

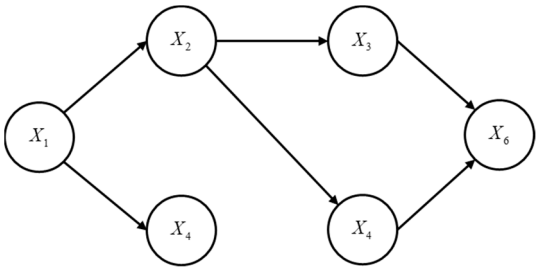

Figure 1. The order graph of a 4-node BN.

Given the dataset $\boldsymbol{D} = \{D_1, \ldots, D_N\}$ (N is the sample size), the scoring function can give the fitness of network structure $\boldsymbol{G}$ and dataset $\boldsymbol{D}$. The goal of BN structure learning is to find a network structure G^* so that the scoring function can obtain the optimal value. Many scoring functions can be used in the BN structure learning problem. Since this paper is based on the perspective of the shortest path search problem, the score of the minimum description length (MDL) [34] is selected. The smaller the MDL score is, the better the corresponding BN structure. Many scoring functions, including the MDL score, are decomposable, namely, the following:

$$MDL(G) = \sum_{i=1}^{n} MDL(X_i|PA_i) \tag{1}$$

where $MDL(X_i|PA_i)$ is called the local score.

Each local score $MDL(X_i|PA_i)$ is calculated as follows:

$$MDL(X_i|PA_i) = H(X_i|PA_i) + \frac{\log N}{2} K(X_i|PA_i) \tag{2}$$

$$H(X_i|PA_i) = -\sum_{x_i,pa_i} N_{x_i,pa_i} \log \frac{N_{x_i,pa_i}}{N_{pa_i}} \tag{3}$$

$$K(X_i|PA_i) = -(r_i - 1) \prod_{X_l \in PA_i} r_l \tag{4}$$

where N_{x_i,pa_i} is the number of data points that satisfy $X_i = x_i$ and $PA_i = pa_i$ in the dataset $\boldsymbol{D}$, N_{pa_i} is the number of data points that satisfy $PA_i = pa_i$ in the dataset $\boldsymbol{D}$, and r_i is the number of states of X_i.

For any variable X_i, the other $n-1$ variables can be its parent nodes, and thus, each node can have 2^{n-1} parent sets. There are $n2^{n-1}$ possible parent sets in total, and the corresponding $n2^{n-1}$ local scores must be calculated theoretically. Obviously, it is impossible to calculate all local scores, and this number can be further reduced by some pruning rules. The following theorems, which have been proven in [19,35], provide a basis for ignoring some parent sets when searching for an optimal parent set for a variable with the MDL function.

Theorem 1 ([19,35]). *In an optimal Bayesian network based on the MDL scoring function, each variable has at most* $\lfloor \log(2N/\log N) \rfloor$ *parents.*

Theorem 2 ([19]). *Let* $\boldsymbol{U}$ *and* $\boldsymbol{S}$ *be two candidate parent sets for* X_i, $\boldsymbol{U} \subset \boldsymbol{S}$, and $K(X_i|\boldsymbol{S}) - MDL(X_i, \boldsymbol{U}) > 0$. *Then,* $\boldsymbol{S}$ *and all supersets of* $\boldsymbol{S}$ *cannot possibly be optimal parent sets for* X_i.

Theorem 3 ([19]). *Let* $\boldsymbol{U}$ *and* $\boldsymbol{S}$ *be two candidate parent sets for* X_i *such that* $\boldsymbol{U} \subset \boldsymbol{S}$, *and* $MDL(X_i, \boldsymbol{U}) \leq MDL(X_i, \boldsymbol{S})$. *Then,* $\boldsymbol{S}$ *is not the optimal parent set of* X_i *for any candidate set.*

The pruning rules are lossless and can ensure that the optimal parent set of each node can be obtained in the remaining parent sets. The remaining parent sets are called the potential optimal parent sets, and the potential optimal parent sets of X_i are denoted as $\boldsymbol{POPS}_i$.

For exact learning algorithms, the local scores that correspond to the POPS are usually calculated in advance. Then, the local scores are used as the input to obtain the output optimal BN score. Therefore, for the exact learning algorithms, BN structure learning is regarded as a combinatorial optimization problem, as follows:

$$\begin{aligned} &\textbf{Input}: \text{A set} \boldsymbol{V} = \{X_1, \ldots, X_n\} \text{ and a set of } \boldsymbol{POPS}_i \text{ for each } X_i \\ &\textbf{Output}: \text{Find } a \text{ DAG } G^* \text{ such that} \\ &G^* \in \underset{G}{\operatorname{argmin}} \sum_{i=1}^{n} MDL(X_i|PA_i) \\ &\text{where } PA_i \in \boldsymbol{POPS}_i \end{aligned} \tag{5}$$

2.2. Structure Learning in Order Graph

A series of DP algorithms are proposed to solve the abovementioned combinatorial optimization problem. DP algorithms are mainly based on the following recursive formula:

$$MDL(\boldsymbol{V}) = \min_{X_i} \{MDL(\boldsymbol{V} \backslash X_i) + BestMDL(X_i, \boldsymbol{V} \backslash X_i)\} \tag{6}$$

$$BestMDL(X_i, \boldsymbol{V} \backslash X_i) = \min_{PA_i \subseteq \boldsymbol{V} \backslash \{X_i\}, PA_i \in \boldsymbol{POPS}_i} MDL(X_i, PA_i) \tag{7}$$

According to the recursive relationship, the basic principle of DP algorithms is as follows. First, the optimal network structure is found for a single variable starting from the empty set. Then, nodes are gradually added to build the optimal subnetwork for an increasingly large set of variables until the optimal network corresponding to

$\boldsymbol{V} = \{X_1, \ldots, X_n\}$ is found. DP algorithms can find the optimal BN in the time and space complexity of $O(n2^n)$, and the whole process can be graphically represented by the order graph. Figure 2 shows the order graph of a four-node BN.

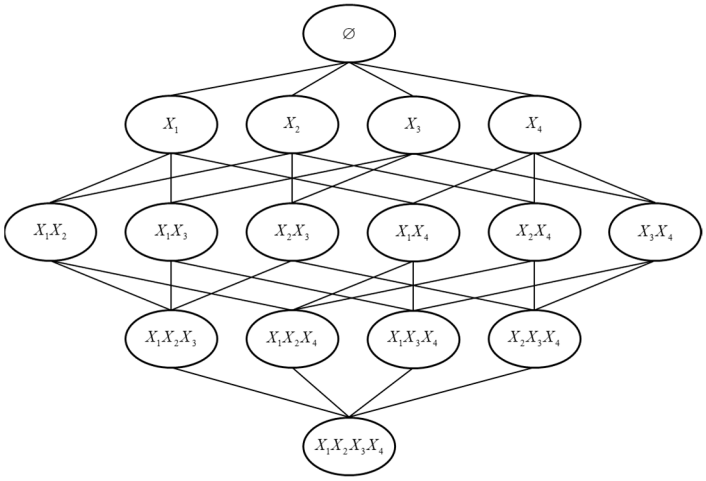

Figure 2. The order graph of a four-node BN.

Yuan and Malone [18,19] further transformed the combinatorial optimization problem into the shortest path search problem. They regarded the top empty set $\boldsymbol{O} = \varnothing$ as the start state and the bottom $\boldsymbol{V}$ as the goal state. Therefore, a path from the start state to the goal state corresponds to the ordering of nodes, which is how the order graph obtains its name. In the order graph, state $\boldsymbol{U}$ to the next state $\boldsymbol{S} = \boldsymbol{U} \cup \{X_i\}(X_i \in \boldsymbol{V} \backslash \boldsymbol{U})$ is equivalent to adding node X_i based on subnetwork $\boldsymbol{U}$, and the path cost from state $\boldsymbol{U}$ to the next state $\boldsymbol{S}$ is

$$\begin{aligned} &cost(\boldsymbol{U}, \boldsymbol{S}) = BestMDL(X_i, \boldsymbol{U}) \\ &= \min_{PA_i \subseteq \boldsymbol{U}, PA_i \in \boldsymbol{POPS}_i} MDL(X_i, PA_i) \end{aligned} \tag{8}$$

where $BestMDL(X_i, \boldsymbol{U})$ is obtained by replacing $\boldsymbol{V} \backslash X_i$ in (7) by $\boldsymbol{U}$. The cost of the path is equal to the score of selecting an optimal parent set for X_i out of $\boldsymbol{U}$, i.e., $BestMDL(X_i, \boldsymbol{U})$. For example, the path $\{X_2, X_3\} \rightarrow \{X_1, X_2, X_3\}$ has a cost equal to $BestMDL(X_1, \{X_2, X_3\})$.

Then, the corresponding optimal ordering can be obtained in the order graph by finding the shortest path from the start state $\boldsymbol{O}$ to the goal state $\boldsymbol{V}$. In the process of finding the shortest path, the optimal parent sets corresponding to the path are recorded. The optimal BN can be built by combining the optimal node ordering with the optimal parent set of each node.

Yuan and Malone searched for the optimal BN structure using the classical A* algorithm based on the shortest path search. In the A* algorithm, for each current state $\boldsymbol{U}$ in the order graph, the path cost $g(\boldsymbol{U})$ generated from start state $\boldsymbol{O}$ to $\boldsymbol{U}$ is calculated, and the heuristic function h is used to estimate the cost $h(\boldsymbol{U})$ from the current state $\boldsymbol{U}$ to the goal state $\boldsymbol{V}$. During the search, $f(\boldsymbol{U}) = g(\boldsymbol{U}) + h(\boldsymbol{U})$ is used to estimate the optimal cost of the path through state $\boldsymbol{U}$, the *Open* list is used to store the states that will be expanded, and the *Closed* list is used to store the states that have been expanded. In the *Open* list, the current state with the lowest f value is expanded each time, and the current state is put into the *Closed* list, while the state expanded by the current state is put into the *Open* list. Until the goal state $\boldsymbol{V}$ is expanded, the shortest path from $\boldsymbol{O}$ to $\boldsymbol{V}$ is found, and the optimal node ordering is also found; thus, the corresponding optimal BN can also be built.

AWA* and BiHS also search for the shortest path in the order graph, only their search strategies are different from A*. Although AWA* and BiHS have the ability to return more upper and lower bounds for the optimal score than A*, in most datasets, A* expands fewer states and has better stability than AWA* and BiHS in the order graph.

3. Two Constraints for Improved A* Algorithm

According to the introduction of BN structure learning theories and the A* algorithm based on the order graph in Section 2, we can obtain two key factors that restrict the efficiency of the A* algorithm:

(1) Potential optimal parent sets. According to the BN structure learning problem description and Formulas (3) and (4), it is necessary to find an optimal parent set from the $\boldsymbol{POPS}_i$ of each node X_i. Obviously, the number of POPS limits the efficiency of the search. If the number of remaining POPS is very small, it is easy to find the parent sets that meet the requirements.

(2) Order graph. The total number of states in the order graph is 2^n, and its scale increases exponentially with the increase in the nodes in BN. The size of the search space also limits the efficiency of the search.

Based on the above two points, this paper will propose solutions to improve the efficiency of the A* algorithm.

3.1. Pruned Potential Optimal Parent Sets with MMPC

Although pruning rules can further reduce the number of POPS, the remaining number is still considerable. In the problem of BN structure learning, we ultimately need optimal parent sets. Therefore, other sets are relatively unnecessary. Given a target variable T, the MMPC algorithm can quickly return the parent–child set $CPC(T)$ of the target variable T under the CI test. For the two variables X, T and the set $\boldsymbol{Z}$, the CI $P_{IT}(X, T|\boldsymbol{Z})$ can be calculated by G^2 statistics under the null hypothesis of conditional independence. Let N^{abc} represent the occurrence times of $X = a$, $T = b$, and $\boldsymbol{Z} = \boldsymbol{c}$ in the dataset $\boldsymbol{D}$ (respectively, a, b, and $\boldsymbol{c}$ denote the values specifically taken by X, T, and $\boldsymbol{Z}$. a and b generally are integers, and $\boldsymbol{c}$ is a combination of the integers.); then, the statistical variable G^2 is defined as

$$G^2 = 2\sum_{a,b,c} N^{abc} \ln\left(\frac{N^{abc}N^{c}}{N^{ac}N^{bc}}\right) \tag{9}$$

Under the null hypothesis, the G^2 statistic asymptotically obeys the distribution of χ^2 statistics. Therefore, given the significance level α, if the value p calculated by the test, namely, $P_{IT}(X, T|\boldsymbol{Z})$, is less than α, the hypothesis is rejected, and the variables X and T are considered to be conditionally dependent under a given $\boldsymbol{Z}$. Otherwise, X and T are considered to be conditionally independent under a given $\boldsymbol{Z}$. The pseudocode of the MMPC algorithm is shown in Algorithm 1. In Algorithm 1, $CPC(T)$ is the parent–child set of the target variable T.

Algorithm 1: MMPC

Input: Target variable T, variable set $\boldsymbol{V}$, and significance level α
Output: Parent-child set of the target variable T: $CPC(T)$

Let parent-child set of the target variable T: $CPC(T) = \varnothing$, $\boldsymbol{R} = \boldsymbol{V}\backslash\{T\}$;
while $\boldsymbol{R} \neq \varnothing$
for $\forall X \in \boldsymbol{R}$ **do**
if $max_{\boldsymbol{Z}\subseteq CPC(T)} P_{IT}(X, T|\boldsymbol{Z}) > \alpha$ **then** $\boldsymbol{R} = \boldsymbol{R}\backslash\{T\}$ **end if**
end for
$Y = argmin_{X\in R} max_{\boldsymbol{Z}\subseteq CPC(T)} P_{IT}(X, T|\boldsymbol{Z})$ and $CPC(T) = CPC(T) \cup \{Y\}$
for $\forall X \in CPC(T)\backslash\{Y\}$ **do**
if $\max_{Z\subseteq CPC(T)\backslash\{X\}} P_{IT}(X, T|\boldsymbol{Z}) > \alpha$ **then** $CPC(T) = CPC(T)\backslash\{X\}$ **end if**
end for
end while

Given a condition set $\boldsymbol{Z}$, the MMPC algorithm not only considers $P_{IT}(X, T|\boldsymbol{Z})$ to determine whether X and T are independent but also considers $\max_{\boldsymbol{Z}' \subseteq \boldsymbol{Z}} P_{IT}(X, T|\boldsymbol{Z}')$, which has stronger robustness, to determine whether X and T are independent. Certainly, this approach requires more χ^2 test calculations. Finally, we can use the MMPC algorithm to compute the parent–child set $CPC(T)$ for each variable X_i.

Through the constraint of the parent–child set $CPC(T)$, we can further prune the unnecessary sets and their corresponding MDL score calculations for the $\boldsymbol{POPS}_i$. Taking a four-node BN as an example, for node X_4, $\{X_1, X_2, X_3\}$ and all of its subsets could be the parent set of X_4. If the traditional pruning rules (Theorems 1–3) are not considered to be in effect, its POPS are still $\{X_1, X_2, X_3\}$ and all its subsets, which are represented as the parent graph of X_4, as shown in Figure 3. If the parent–child set of X_4 obtained by the MMPC algorithm is $CPC(X_4) = \{X_2, X_3\}$, then the parent graph of X_4 shown in Figure 3 can be pruned to the parent graph shown in Figure 4. For larger BNs, this pruning will be more significant in its score calculations. The constraints of the parent–child set calculated by the MMPC algorithm can greatly reduce unnecessary score calculations and storage. Limiting the number of corresponding POPS improves the search efficiency of the A* algorithm.

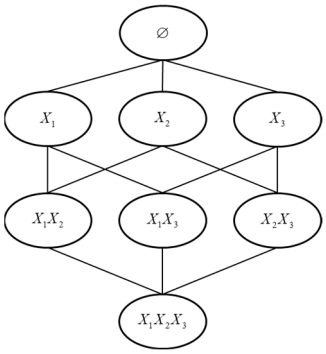

Figure 3. Parent graph of X_4 in a 4-node BN.

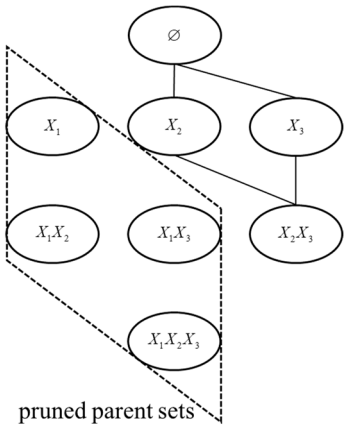

Figure 4. Parent graph of X_4 after pruning in a 4-node BN.

3.2. Pruned Order Graph with Path Constraints

According to Section 2.2, the optimal BN structure learning actually searches for the shortest path from the order graph. Therefore, if the path constraints can be found in the order graph, it will greatly improve the efficiency of the A* algorithm in searching for the shortest path in the order graph.

Before illustrating such path constraints, a simple example can be taken. Table 1 shows the POPS of each variable in a six-node BN. We assume that we have obtained the POPS of each variable by the score pruning rules or MMPC algorithm in Section 3.1. It can be seen from Table 1 that not all nodes can choose all other nodes as their parent nodes due to the

parent–child set constraints obtained from Section 3.1. For example, X_1 can only choose X_2 as its parent node or an empty set with no parent.

Table 1. The POPS of each variable in a 6-node BN.

Variable	POPS
X_1	$\{X_2\}, \{\}$
X_2	$\{X_1\}, \{\}$
X_3	$\{X_1, X_2\}, \{X_1, X_4\}, \{X_2, X_4\}, \{X_1\}, \{\}$
X_4	$\{X_1, X_5\}, \{X_1\}, \{X_5\}, \{\}$
X_5	$\{X_1, X_2\}, \{X_6\}, \{X_1\}, \{X_2\}, \{\}$
X_6	$\{X_2, X_3\}, \{X_3\}, \{\}$

A directed graph can be obtained by connecting each node X_i and its potential optimal parent sets $\boldsymbol{POPS}_i$. We connect from each node X_i and its potential optimal parent sets $\boldsymbol{POPS}_i$ from Table 1 to obtain the directed graph, as shown in Figure 5. In such a directed graph, if X_j is a potential parent node of X_i, then the graph contains directed edges from X_j to X_i.

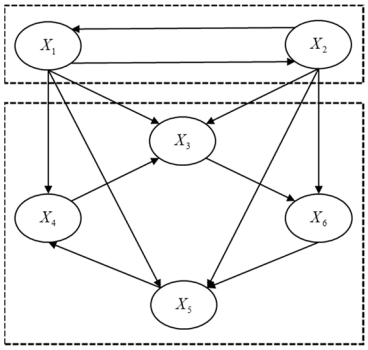

Figure 5. A directed graph based on each variable in Table 1 and its POPS.

An interesting phenomenon can be observed in Figure 5: there are only directed edges from the node in $\{X_1, X_2\}$ to the node in $\{X_3, X_4, X_5, X_6\}$ but no directed edges from the node in $\{X_3, X_4, X_5, X_6\}$ to the node in $\{X_1, X_2\}$; in other words, the node in $\{X_3, X_4, X_5, X_6\}$ cannot be the parent node of the node in $\{X_1, X_2\}$, and thus, it can be split into two parts: $\{X_1, X_2\}$ and $\{X_3, X_4, X_5, X_6\}$. Thus, based on Figure 5, by contracting $\{X_1, X_2\}$ to one node and $\{X_3, X_4, X_5, X_6\}$ to another node, we can finally obtain the acyclic component graph, as shown in Figure 6. Based on the above splitting method, we can split the order graph of learning the six-node BN into two subgraphs, as shown in Figure 6.

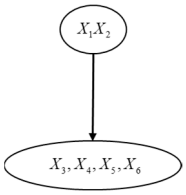

Figure 6. Acyclic component graph based on Figure 5.

Obviously, it can be seen from the above that the complete order graph of the six-variable BN should contain 2^6 states. However, the order graph can be split into two subgraphs based on the constraints from Figure 5, in other words, $\{X_1, X_2\}$ and $\{X_3, X_4, X_5, X_6\}$. We refer to this splitting method as path constraints.

As the structure of the order graph changed, the entire process of searching the order graph also changed. First, we find the shortest path from $\boldsymbol{O}$ to $\{X_1, X_2\}$ in the first subgraph of Figure 7, and then find the shortest path from $\{X_1, X_2\}$ to $\boldsymbol{V} = \{X_1, X_2, X_3, X_4, X_5, X_6\}$ in

the second subgraph of Figure 7. The shortest path from $\boldsymbol{O}$ to $\boldsymbol{V}$ is obtained by concatenating the shortest paths in the two subgraphs. $\{X_1, X_2\}$ becomes the necessary state in the shortest searching process of the order graph. Therefore, the number of the states in the order graph search space of Figure 7 can be reduced to $2^2 + 2^4 - 1$.

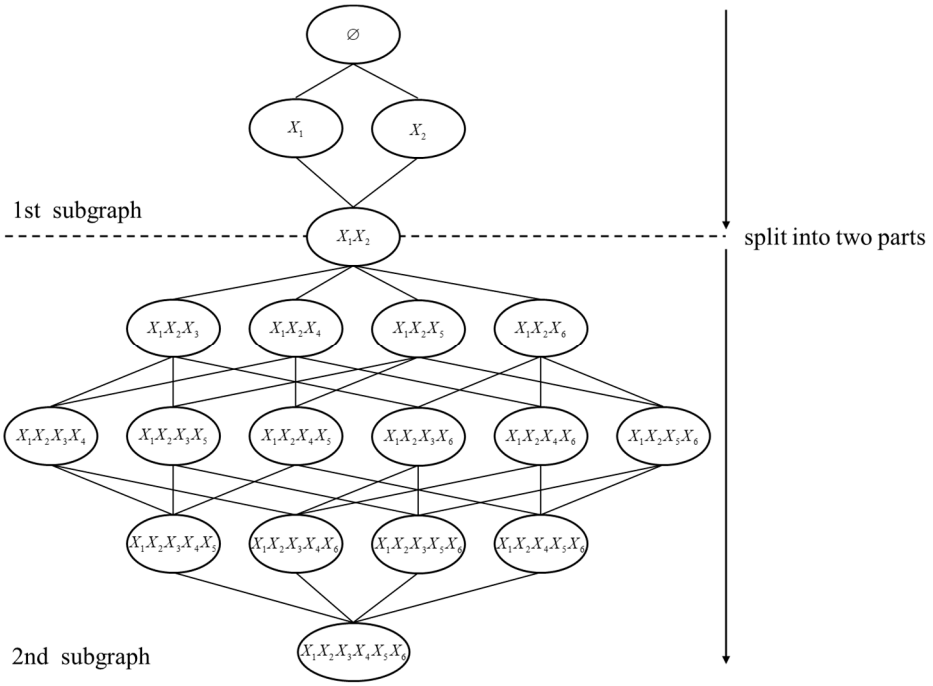

Figure 7. Pruned order graph based on the constraints from Figure 5.

Compared with 2^6 states in the complete order graph of the six-variable BN, path constraints can reduce the number of states in the order graph. As the number of nodes n increases, the path constraint reduces the number of states in the order graph more significantly. We give Theorem 4 to generalize and quantify this reduction.

Theorem 4. *In a Bayesian network with node set* $\boldsymbol{V} = \{X_1, \ldots, X_n\}$, *given path constraints,* $\boldsymbol{V}$ *can be split into subsets* $\boldsymbol{P}_1, \boldsymbol{P}_2, \ldots, \boldsymbol{P}_m (\sum_{i=1}^{m} \boldsymbol{P}_i = \boldsymbol{V})$. *Then, the number of states in the order graph is reduced from* 2^n *to* $\sum_{i=1}^{m} 2^{|P_i|} - m + 1$.

Proof of Theorem 4. Obviously, the total 2^n states of the complete order graph correspond to the Bayesian network of n nodes. For the ordered graph under path constraints, where the number of states of any subgraph split by $\boldsymbol{P}_i$ is $2^{|\boldsymbol{P}_i|}$, there are m such subgraphs, and the total number of states is $\sum_{i=1}^{m} 2^{|\boldsymbol{P}_i|}$. However, this will double count $m-1$ states. Therefore, $m-1$ states are removed from the total number of computations. Finally, the total number of states is $\sum_{i=1}^{m} 2^{|P_i|} - m + 1$. □

This simple example shows that the directed graph built from each node X_i and its potential optimal parent sets $\boldsymbol{POPS}_i$ implies path constraints, which can be used to prune the order graph.

The internal principle is briefly described as follows, requiring the help of Theorem 5 (it is proved in the literature [19]).

Theorem 5. *Let* $\boldsymbol{U}$ *and* $\boldsymbol{S}$ *be two candidate parent sets for* X_i *such that* $\boldsymbol{U} \subset \boldsymbol{S}$. *We must have* $BestMDL(X_i, \boldsymbol{S}) \leq BestMDL(X_i, \boldsymbol{U})$.

In general, if there is only a directed path from X_j to X_i in the directed graph but no directed path from X_i to X_j, then the order graph does not need to generate states containing X_i but excluding X_j. One way to think about this phenomenon is the following. For a current state $\boldsymbol{U}$ in the order graph that does not include X_i and X_j, if we expand X_j first and then X_i, then the path cost from state $\boldsymbol{U}$ to state $\boldsymbol{U} \cup \{X_i, X_j\}$ is $BestMDL(X_j, \boldsymbol{U}) + BestMDL(X_i, \boldsymbol{U} \cup \{X_j\})$. On the other hand, if we expand X_i first and then X_j, the path cost from state $\boldsymbol{U}$ to state $\boldsymbol{U} \cup \{X_i, X_j\}$ is $BestMDL(X_i, \boldsymbol{U}) + BestMDL(X_j, \boldsymbol{U} \cup \{X_i\})$. However, since only a directed path from X_j to X_i can exist, $BestMDL(X_j, \boldsymbol{U} \cup \{X_i\}) = BestMDL(X_j, \boldsymbol{U})$. For these two path expansion plans, we should continue to compare the values of $BestMDL(X_i, \boldsymbol{U} \cup \{X_j\})$ and $BestMDL(X_i, \boldsymbol{U})$. According to Theorem 5 and $\boldsymbol{U} \subseteq \boldsymbol{U} \cup \{X_j\}$, it is more likely to obtain a better value that makes this path smaller in a larger set, and thus, $BestMDL(X_i, \boldsymbol{U} \cup \{X_j\}) \leq BestMDL(X_i, \boldsymbol{U})$. Therefore, the plan that expands X_j first and then X_i is more likely to achieve the shortest path from $\boldsymbol{U}$ to $\boldsymbol{U} \cup \{X_i, X_j\}$. Thus, there is no need to generate states that contain X_i but exclude X_j.

Based on the previous simple example, we discuss the general method of obtaining path constraints in order to prune the order graph.

A new concept, the strongly connected component (SCC), is actually involved in the process of splitting the directed graph built from each node X_i and its potential optimal parent sets $\boldsymbol{POPS}_i$. In a directed graph, if there is a directed path from V_i to V_j between two nodes and a directed path from V_j to V_i, the two nodes are said to be strongly connected. A directed graph is a strongly connected graph if any two nodes are strongly connected. The extremely strongly connected subgraph of a directed graph is called a strongly connected component. The strongly connected components of a directed graph form an acyclic component graph, which is also a DAG. Each node C_i in the acyclic component graph corresponds to a strongly connected component $\boldsymbol{SCC}_i$ and to a subset of the node set $\boldsymbol{V} = \{X_1, \ldots, X_n\}$ in a BN. The acyclic component graph gives more intuitive path constraints. In the acyclic component graph, if there are directed paths from C_i to C_j, the variable in $\boldsymbol{SCC}_j$ cannot be the parent node of the variable in $\boldsymbol{SCC}_i$.

Based on the above concept, we try to obtain path constraints by extracting SCCs to prune the order graph. At present, there are mature algorithms for SCC extraction, among which the Kosaraju algorithm is the most commonly used. The pseudocode of the algorithm that obtains path constraints by extracting SCCs is shown in Algorithm 2.

In this algorithm, the potential optimal parent sets $\boldsymbol{POPS}_i$ of each node X_i are used to build the directed graph G^0, and the SCC $\{\boldsymbol{SCC}_1, \ldots \boldsymbol{SCC}_i, \ldots \boldsymbol{SCC}_m\}$ of the directed graph G^0 is extracted by the Kosaraju algorithm. It is worth noting that if the size of the SCC is too large, it is still not conducive to improving the efficiency of the algorithm and to searching for a larger network. For example, the original A* algorithm itself cannot search the network of over 50 nodes. If, in the operation of building a directed graph G^0 and extracting SCCs through the POPS, two SCCs with $|\boldsymbol{SCC}_1| = 1$ and $|\boldsymbol{SCC}_2| = 49$ are obtained, and the path constraints are determined, such a method is still meaningless. Because the A* algorithm still cannot search a network of 49 nodes. Thus, we limit the size of the SCC with the parameter t. If the size of the maximum SCC exceeds the parameter t, a part of the set of potential optimal parent sets $\boldsymbol{POPS}_i$ is selected to rebuild the directed graph G^0. We prefer to select the sets that correspond to the local scores of the top k in $POPS_i$. The parameter t will gradually decrease from the maximum number of POPS until the SCC that meets the conditions can be extracted from the built directed graph. Then, Algorithm 2 breaks out of the loop and returns the extracted SCCs under the constraints of the parameter t. However, this method is greedy because only part of POPS is used to build the directed graph, and the extracted SCCs lose some information. The pruned order graph formed according to the path constraints of SCCs has certain problems, which will affect the shortest path search and affect the accuracy of the final BN. This effect will be analyzed in detail in the experimental section.

Algorithm 2: Obtain path constraints algorithm

Input: Potential optimal parent sets of each node X_i $\boldsymbol{POPS}_i$, maximum size t
Output: Path constraints $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_i, \ldots, \boldsymbol{P}_m$

$p \leftarrow \max(|\boldsymbol{POPS}_1|, \ldots |\boldsymbol{POPS}_i|, \ldots |\boldsymbol{POPS}_m|)$
build graph G^0 according to all $POPS_i$ of X_i
$\{\boldsymbol{SCC}_1, \ldots, \boldsymbol{SCC}_i, \ldots, \boldsymbol{SCC}_m\} \leftarrow Kosaraju(G^0)$;
$q \leftarrow \max(|\boldsymbol{SCC}_1|, \ldots |\boldsymbol{SCC}_i|, \ldots |\boldsymbol{SCC}_m|)$;
if $q > t$ **then**
for $k = p \rightarrow 1$ **do**
build graph G^0 according to the best k $POPS_i$ of X_i
$\{\boldsymbol{SCC}_1, \ldots, \boldsymbol{SCC}_i, \ldots, \boldsymbol{SCC}_m\} \leftarrow Kosaraju(G^0)$;
$q \leftarrow \max(|\boldsymbol{SCC}_1|, \ldots |\boldsymbol{SCC}_i|, \ldots |\boldsymbol{SCC}_m|)$;
if $q \leq t$ **then break; endif**
end for
end if
$\boldsymbol{P}_1, \ldots, \boldsymbol{P}_i, \ldots, \boldsymbol{P}_m \leftarrow \boldsymbol{SCC}_1, \ldots, \boldsymbol{SCC}_i, \ldots, \boldsymbol{SCC}_m$

Finally, we discuss the search complexity in the pruned order graph. $\{\boldsymbol{SCC}_1, \ldots, \boldsymbol{SCC}_i, \ldots, \boldsymbol{SCC}_m\}$ obtained by Algorithm 2 can split the original order graph into m subgraphs, where the connection state between each subgraph is $\boldsymbol{F}_i = \cup_{k=1}^{i} \boldsymbol{SCC}_k$, and $\boldsymbol{F}_m = \cup_{k=1}^{m} \boldsymbol{SCC}_k = \boldsymbol{V}$, $\boldsymbol{F}_0 = \{\} = \boldsymbol{O}$. Thus, for each subgraph, the start state is $\boldsymbol{F}_{i-1}$ and the goal state is $\boldsymbol{F}_i$. For the entire order graph, it is equivalent to searching the shortest paths from $\boldsymbol{F}_0$ to $\boldsymbol{F}_1$, then from $\boldsymbol{F}_1$ to $\boldsymbol{F}_2$, all the way to $\boldsymbol{F}_{m-1}$ to $\boldsymbol{F}_m$. Still taking Figure 7 as an example, because there are $\boldsymbol{SCC}_1 = \{X_1, X_2\}$ and $\boldsymbol{SCC}_2 = \{X_3, X_4, X_5, X_6\}$, there are $\boldsymbol{F}_1 = \{X_1, X_2\}$ and $\boldsymbol{F}_2 = \{X_1, X_2, X_3, X_4, X_5, X_6\}$. Therefore, we search the shortest path from $\boldsymbol{F}_0 = \varnothing$ to $\boldsymbol{F}_1 = \{X_1, X_2\}$ and then search the shortest path from $\boldsymbol{F}_1 = \{X_1, X_2\}$ to $\boldsymbol{F}_2 = \{X_1, X_2, X_3, X_4, X_5, X_6\}$. Finally, it only remains to connect each shortest path to obtain the entire shortest path on the order graph. For each subgraph, the maximum complexity of the A* search is $O\left(2^{|SCC_i|}\right)$. This case is the worst case, which is almost impossible because A* uses heuristic functions. Therefore, in the pruned order graph, which is split into m subgraphs using $\{\boldsymbol{SCC}_1, \ldots, \boldsymbol{SCC}_i, \ldots, \boldsymbol{SCC}_m\}$, the maximum complexity of the A* search is

$$O\left(2^{|\boldsymbol{SCC}_1|} + \ldots + 2^{|\boldsymbol{SCC}_i|} \ldots + 2^{|\boldsymbol{SCC}_m|}\right) = O\left(m \max_i 2^{|\boldsymbol{SCC}_i|}\right) \tag{10}$$

This conclusion also corroborates Theorem 4. This finding shows that the maximum complexity depends on the size of the maximum SCC. Therefore, it is necessary for Algorithm 2 to use the parameter t to limit the size of the maximum SCC, which effectively limits the maximum complexity of the A* search of the pruned order graph.

4. Experiments

To evaluate the effect of A* under MMPC constraints (Section 3.1) and path constraints (Section 3.2), experiments will be performed on some common benchmark BNs and UCI datasets. The A* algorithm using only MMPC constraints is named A*-MMPC, the A* algorithm using only path constraints is named A*-PC, and the algorithm using both constraints is named A*-MM2PC. Experiments are mainly divided into two parts. First, the improvement effect of the two constraints on A* is tested; in other words, the various indices between the A*, A*-MMPC, A*-PC, and A*-MM2PC algorithms are tested. Then, the A*-MM2PC with two constraints is compared with the typical GOBNILP and MMHC algorithms.

First, the comparison experiment will compare A*, A*-MMPC, A*-PC, and A*-MM2PC from the following three aspects:

1. **Time**: Time recorded the running time of the algorithm (OT means out of time);

2. **States**: The number of expanded states in the order graph;
3. **Error**: The percentage error $(MDL(G_{obtained})/MDL(G_{exact}) - 1) \times 100\%$ where $MDL(G_{obtained})$ is the MDL score of the BN obtained by the performing algorithm after learning the data, and $MDL(G_{exact})$ is the exact MDL score of the BN obtained by the original A* or GOBNILP algorithms.
4. Both the original A* and GOBNILP algorithms are exact learning algorithms. If scoring values can be obtained, they are both optimal and equal. Therefore, these two algorithms do not need to calculate the percentage error, and they are both 0%. On the one hand, the MMPC algorithm uses a CI test, and there will be a certain probability of error in the case of insufficient or noisy samples. On the other hand, path constraints adopt a greedy strategy when extracted SCCs do not meet parameter t. Therefore, A*-MMPC, A*-PC, and A*-MM2PC should all consider the influence of the accuracy, namely, the percentage error. The smaller the percentage error is, the higher the accuracy of the corresponding algorithm. Two decimal places are retained for each index, and scientific notation is used for numbers that are too large or too small.

The benchmark BNs are selected for the comparison experiment, and the sampled data are obtained from the sample sizes of 1000, 3000, 5000, 7000, and 10,000. Then, BN is learned from the sampled data. Table 2 records the performances of the A*, A*-MMPC, A*-PC, and A*-MM2PC algorithms on benchmark BNs. By adding path constraints, the scale of BNs that the A* algorithm can search is expanded, and Water and Alarm networks that cannot be searched before can be searched.

Table 2. Performances of the A*, A*-MMPC, A*-PC, and A*-MM2PC algorithms on benchmark BNs. The best-performing results (including minimum time, minimum number of expanded states and minimum percentage error) are highlighted in bold in the Table 2 and the following tables.

Name	n	N	A*		A*-MMPC			A*-PC			A*-MM2PC		
			Time(s)	States	Time(s)	States	Error	Time(s)	States	Error	Time(s)	States	Error
Sachs	11	1000	2.61×10^{-3}	215	$\mathbf{2.02 \times 10^{-3}}$	199	**0%**	3.55×10^{-3}	58	**0%**	3.05×10^{-3}	**49**	**0%**
Sachs	11	3000	2.64×10^{-3}	416	$\mathbf{2.27 \times 10^{-3}}$	368	**0%**	4.52×10^{-3}	84	**0%**	3.37×10^{-3}	**76**	**0%**
Sachs	11	5000	3.04×10^{-3}	492	$\mathbf{2.33 \times 10^{-3}}$	396	**0%**	5.01×10^{-3}	89	**0%**	3.71×10^{-3}	**79**	**0%**
Sachs	11	7000	3.92×10^{-3}	599	$\mathbf{2.82 \times 10^{-3}}$	487	**0%**	5.64×10^{-3}	100	**0%**	4.04×10^{-3}	**86**	**0%**
Sachs	11	10,000	3.65×10^{-3}	629	$\mathbf{2.86 \times 10^{-3}}$	501	**0%**	5.83×10^{-3}	113	**0%**	4.16×10^{-3}	**98**	**0%**
Child	20	1000	1.94×10^{-1}	50,887	1.28×10^{-1}	36,678	0.21%	7.89×10^{-2}	18,714	**0%**	$\mathbf{6.30 \times 10^{-2}}$	**16,226**	0.21%
Child	20	3000	5.69×10^{-1}	158,853	2.48×10^{-1}	63,946	0.27%	2.08×10^{-1}	58,677	**0%**	$\mathbf{7.44 \times 10^{-2}}$	**29,427**	0.27%
Child	20	5000	4.07×10^{-1}	115,639	3.13×10^{-1}	65,100	0.19%	1.73×10^{-1}	52,226	**0%**	$\mathbf{7.53 \times 10^{-2}}$	**30,242**	0.19%
Child	20	7000	4.19×10^{-1}	119,577	3.05×10^{-1}	59,259	**0%**	1.81×10^{-1}	54,496	**0%**	$\mathbf{7.39 \times 10^{-2}}$	**28,040**	**0%**
Child	20	10,000	4.64×10^{-1}	137,545	3.36×10^{-1}	67,065	**0%**	2.02×10^{-1}	63,799	**0%**	$\mathbf{8.87 \times 10^{-2}}$	**31,196**	**0%**
Insurance	27	1000	142.45	$2.05 \times 10^{+7}$	36.47	$4.71 \times 10^{+6}$	0.72%	44.98	$7.00 \times 10^{+6}$	**0%**	**12.36**	$\mathbf{1.95 \times 10^{+6}}$	0.72%
Insurance	27	3000	175.68	$2.44 \times 10^{+7}$	27.87	$3.42 \times 10^{+6}$	0.60%	48.42	$8.14 \times 10^{+6}$	**0%**	**10.44**	$\mathbf{1.67 \times 10^{+6}}$	0.60%
Insurance	27	5000	192.73	$2.56 \times 10^{+7}$	21.22	$2.63 \times 10^{+6}$	0.49%	50.76	$8.08 \times 10^{+6}$	**0%**	**8.84**	$\mathbf{1.35 \times 10^{+6}}$	0.49%
Insurance	27	7000	218.43	$2.66 \times 10^{+7}$	23.67	$2.91 \times 10^{+6}$	0.45%	52.54	$8.48 \times 10^{+6}$	**0%**	**9.13**	$\mathbf{1.38 \times 10^{+6}}$	0.45%
Insurance	27	10,000	215.52	$2.55 \times 10^{+7}$	26.95	$3.16 \times 10^{+6}$	0.42%	50.95	$8.15 \times 10^{+6}$	**0%**	**9.53**	$\mathbf{1.46 \times 10^{+6}}$	0.42%
Water	32	1000	OT		OT			23.65	$5.67 \times 10^{+6}$	**0%**	**18.98**	$\mathbf{4.98 \times 10^{+6}}$	0.19%
Water	32	3000	OT		OT			26.31	$6.20 \times 10^{+6}$	**0%**	**18.55**	$\mathbf{4.77 \times 10^{+6}}$	0.18%
Water	32	5000	OT		OT			27.07	$6.39 \times 10^{+6}$	**0%**	**17.11**	$\mathbf{4.20 \times 10^{+6}}$	0.11%
Water	32	7000	OT		OT			27.66	$6.41 \times 10^{+6}$	**0%**	**19.60**	$\mathbf{4.46 \times 10^{+6}}$	0.06%
Water	32	10,000	OT		OT			27.86	$6.45 \times 10^{+6}$	**0%**	**21.08**	$\mathbf{4.85 \times 10^{+6}}$	0.07%
Alarm	37	1000	OT		OT			3.17×10^{-1}	$1.01 \times 10^{+6}$	**0%**	$\mathbf{1.40 \times 10^{-1}}$	**29,265**	0.03%
Alarm	37	3000	OT		OT			1.37×10^{-1}	4992	**0.22%**	$\mathbf{4.96 \times 10^{-2}}$	**2731**	1.99%
Alarm	37	5000	OT		OT			2.59×10^{-1}	78,817	**0.02%**	$\mathbf{7.14 \times 10^{-2}}$	**29,283**	0.44%
Alarm	37	7000	OT		OT			3.33×10^{-1}	10,3821	**0%**	$\mathbf{8.42 \times 10^{-2}}$	**30,415**	0.02%
Alarm	37	10,000	OT		OT			2.48×10^{-1}	86,143	**0.02%**	$\mathbf{8.10 \times 10^{-2}}$	**29,695**	0.21%
Hailfinder	56	1000	OT		OT			6.67×10^{-2}	1670	**0.013%**	$\mathbf{2.52 \times 10^{-2}}$	**573**	0.11%
Hailfinder	56	3000	OT		OT			6.41×10^{-2}	1931	**0.023%**	$\mathbf{3.42 \times 10^{-2}}$	**922**	0.42%
Hailfinder	56	5000	OT		OT			1.56	$3.62 \times 10^{+5}$	**0.025%**	$\mathbf{8.26 \times 10^{-1}}$	$\mathbf{1.51 \times 10^{+5}}$	1.35%
Hailfinder	56	7000	OT		OT			2.91×10^{-2}	491	**0.067%**	$\mathbf{1.21 \times 10^{-2}}$	**202**	2.21%
Hailfinder	56	10,000	OT		OT			1.66	$3.66 \times 10^{+5}$	**0.043%**	$\mathbf{8.61 \times 10^{-1}}$	$\mathbf{1.53 \times 10^{+5}}$	1.21%
Win95pts	76	1000	OT		OT			1.47	$3.23 \times 10^{+5}$	**0.49%**	$\mathbf{6.48 \times 10^{-1}}$	$1.32 \times 10^{+5}$	1.61%
Win95pts	76	3000	OT		OT			2.07	$4.66 \times 10^{+5}$	**0.67%**	**1.08**	$2.37 \times 10^{+5}$	2.13%
Win95pts	76	5000	OT		OT			3.36	$7.63 \times 10^{+5}$	**0.50%**	**2.29**	$4.40 \times 10^{+5}$	2.99%
Win95pts	76	7000	OT		OT			3.98	$8.91 \times 10^{+5}$	**0.59%**	**2.72**	$5.44 \times 10^{+5}$	2.15%
Win95pts	76	10,000	OT		OT			3.70	$8.36 \times 10^{+5}$	**0.56%**	**3.28**	$6.56 \times 10^{+5}$	1.94%

In terms of the number of expanded states in the order graph, using either MMPC or path constraints can significantly reduce the number of states, and A*-MM2PC using both

constraints has the lowest number of expanded states. Similarly, in general, the trend of the corresponding time consumption follows the trend of the number of expanded nodes, except for the Sachs network. In the Sachs network, although A*-PC and A*-MM2PC have fewer expanded states than A* and A*-MMPC, their corresponding time consumption is higher. The time consumption for structure learning in the Sachs network is already low; however, it takes a certain amount of time to generate heuristic functions every time the A* program is executed. Therefore, in the Sachs network, A*-PC and A*-MM2PC divide the complete order graph into several subgraphs using path constraints, and each subgraph increases the generation time of the heuristic function accordingly. In conclusion, A*-PC and A*-MM2PC use path constraints to reduce the number of expanded states in the order graph, and thus reduce the search time, which is far less than the generation time of the increased heuristic function, resulting in higher time consumption in a smaller network. However, in larger networks, the reduction in more states is the more dominant factor, and thus, their time consumption is significantly reduced. In the Alarm network, the size of the maximum SCC obtained by Algorithm 2 is 18, while the sizes of other SCCs are mostly 2 and 3. Therefore, the experimental results regarding the time consumption and the number of expanded states are similar to the Child network with 20 nodes. In Hailfinder and Win95pts, the size of the maximum SCC obtained by Algorithm 2 is relatively small, so their total time consumption and the number of expanded states are both even smaller than the corresponding results for smaller networks. This phenomenon shows that the maximum complexity of the algorithm depends on the size of the maximum SCC, which is consistent with the conclusion in Section 3.2.

In terms of the accuracy of these algorithms, it is easier to lose accuracy using MMPC constraints. The accuracy loss is more significant in the case of a small sample size, and the accuracy is higher in the case of a large sample size. Since MMPC uses the CI test, it is sensitive to the sample size and can only obtain good results under the condition of sufficient samples; as a result, this drawback is also inherited into the new algorithm. In addition, path constraints lead to accuracy loss in the Alarm network. For calculating path constraints in a large-scale network, Algorithm 2 will attempt to generate a directed graph using the sets that corresponds to the local scores of the top k in $POPS_i$. However, this approach is greedy, which reduces the accuracy of the final BN. Fortunately, path constraints using the greedy approach lead to a lower accuracy loss than MMPC constraints. Furthermore, it can be concluded from the experimental results that the accuracy loss of the A*-MM2PC algorithm using MMPC constraints and path constraints comes more from the accuracy loss caused by MMPC constraints.

Table 3 records the performances of the A*, A*-MMPC, A*-PC, and A*-MM2PC algorithms on 13 common UCI datasets. By adding MMPC constraints and path constraints, the size of BNs that can be searched by the A* algorithm is expanded, and the performance of adding path constraints is more significant than that of adding MMPC constraints. In terms of the number of expanded states in the order graph, MMPC or path constraints can significantly reduce the number of expanded states, and adding both can reduce the number of expanded states even further. The variation trend of time consumption is similar to that of the number of expanded states. In terms of the accuracy of these algorithms, since most UCI datasets have small sample sizes, and the CI test used in MMPC constraints requires a sufficient sample size, it is easier to lose accuracy by using MMPC constraints. Comparatively, the accuracy loss of path constraints appears only in Flag, Soybean, Bands, and Spectf. The accuracy loss of the A*-MM2PC algorithm using both MMPC constraints and path constraints mainly comes from MMPC constraints.

Table 3. Performances of the A*, A*-MMPC, A*-PC, and A*-MM2PC algorithms on UCI datasets.

Name	n	N	A*		A*-MMPC			A*-PC			A*-MM2PC		
			Time(s)	States	Time(s)	States	Error	Time(s)	States	Error	Time(s)	States	Error
Lympho	19	148	5.37×10^{-3}	17,414	2.94×10^{-2}	95	**0%**	5.33×10^{-2}	8757	**0%**	$\mathbf{2.44 \times 10^{-2}}$	**93**	**0%**
Hepatitis	20	126	6.05×10^{-3}	8515	2.80×10^{-2}	2824	0.26%	3.85×10^{-2}	4809	**0%**	$\mathbf{1.86 \times 10^{-3}}$	**1533**	0.26%
Segment	20	2310	1.72	428,083	7.99×10^{-1}	218,456	0.76%	4.01×10^{-1}	107,902	**0%**	$\mathbf{1.71 \times 10^{-1}}$	**54,588**	0.76%
Mushroom	23	8124	5.93×10^{-1}	49,593	3.55×10^{-1}	38,835	2.07%	4.89×10^{-1}	33,167	**0%**	$\mathbf{2.55 \times 10^{-1}}$	**25,669**	2.07%
Autos	26	159	36.55	$4.76 \times 10^{+6}$	12.24	$1.64 \times 10^{+6}$	2.60%	18.38	$2.54 \times 10^{+6}$	**0%**	**6.74**	**906,247**	2.60%
Steel	28	1941	OT		50.48	$7.61 \times 10^{+6}$	2.70%	12.34	$2.55 \times 10^{+6}$	**0%**	**1.46**	**310,744**	2.70%
Flag	29	194	OT		2.99	319,340	0.69%	3.47	418,554	**0.01%**	$\mathbf{3.53 \times 10^{-1}}$	**46,659**	0.69%
Soybean	36	266	OT		OT			1.23	234,185	**0.13%**	**1.16**	**164,698**	3.93%
Bands	39	277	OT		OT			11.33	$1.54 \times 10^{+6}$	**0%**	**0.23**	**15,354**	1.06%
Spectf	45	267	OT		OT			8.45×10^{-2}	76	**0.10%**	$\mathbf{7.76 \times 10^{-2}}$	**74**	0.27%
Sponge	45	76	OT		OT			3.01×10^{-1}	13,106	**0.659%**	$\mathbf{1.73 \times 10^{-1}}$	**6753**	1.304%
LungCancer	57	32	OT		OT			4.42×10^{-1}	19,223	**2.159%**	3.66×10^{-1}	**26,019**	5.54%
Splice	61	3190	OT		OT			$\mathbf{1.62 \times 10^{-1}}$	216	**0.123%**	$\mathbf{1.62 \times 10^{-1}}$	**193**	0.323%

On the whole, MMPC and path constraints can effectively and significantly improve the overall efficiency of the A* algorithm and reduce time consumption and the number of expanded states in the order graph, at the cost of only a slight loss in accuracy.

A*-MM2PC with MMPC constraints and path constraints is compared with other classical algorithms. The GOBNILP algorithm is considered to be the state-of-the-art algorithm among the exact learning algorithms, while the MMHC algorithm is the most well-known algorithm among the hybrid algorithms, which also uses MMPC constraints. In addition, the Insert Neighborhood Ordering-Based Search (INOBS) [36] algorithm is a state-of-the-art improved variant of OBS. The comparison experiment will compare GOBNILP, MMHC, INOBS, and A*-MM2PC from the following two aspects:

1. **Time**: time recorded the running time of the algorithm (OT means out of time);
2. **Error**: the percentage error $(MDL(G_{obtained})/MDL(G_{exact}) - 1) \times 100\%$.

The percentage error is calculated in the same way as before, and since the GOBNILP algorithm is an exact learning algorithm, it is always 0% and is not recorded. In addition, since GOBNILP, MMHC, and INOBS have different search spaces and do not use the order graph as the search space, the number of expanded states in the order graph is also not recorded.

Table 4 records the performances of the GOBNILP, MMHC, INOBS, and A*-MM2PC algorithms on benchmark BNs. Compared with the GOBNILP algorithm, A*-MM2PC consumes less time in the Sachs, Child, Alarm, Hailfinder, and Win95pts networks. Although the accuracy of A*-MM2PC is not as good as GOBNILP, the overall accuracy loss of A*-MM2PC is less than 3%, mainly concentrated within 0.5%. Compared with the MMHC algorithm, A*-MM2PC has less time consumption on the Sachs, Child, Alarm, Hailfinder, and Win95pts networks and always has less accuracy loss than the MMHC algorithm. Although both the MMHC and A*-MM2PC algorithms adopt MMPC constraints, MMHC uses greedy search to further search in the second stage, while A*-MM2PC is based on the A* algorithm, an exact learning algorithm, to further search; thus, A*-MM2PC will achieve higher accuracy than MMHC. The accuracy of the MMHC algorithm also roughly conforms to the trend of low accuracy when the sample size is small and high accuracy when the sample size is large. The reason is the fact that CI tests rely on sufficient samples. Compared with the INOBS algorithm, A*-MM2PC mostly has less time consumption on the Sachs, Child, Alarm, Hailfinder, and Win95pts networks than the INOBS algorithm, and has less accuracy loss than the INOBS algorithm, in most cases.

Table 4. Performances of GOBNILP, MMHC, and A*-MM2PC algorithms on benchmark BNs.

Name	n	N	GOBNILP	MMHC		INOBS		A*-MM2PC	
			Time(s)	Time(s)	Error	Time(s)	Error	Time(s)	Error
Sachs	11	1000	0.22	0.39	0.67%	1.5×10^{-2}	0%	$\mathbf{3.05 \times 10^{-3}}$	**0%**
Sachs	11	3000	0.21	0.60	**0%**	1.5×10^{-2}	0.098%	$\mathbf{3.37 \times 10^{-3}}$	**0%**
Sachs	11	5000	0.31	0.95	**0%**	1.5×10^{-2}	0.065%	$\mathbf{3.71 \times 10^{-3}}$	**0%**
Sachs	11	7000	0.38	1.11	**0%**	1.5×10^{-2}	0.130%	$\mathbf{4.04 \times 10^{-3}}$	**0%**
Sachs	11	10,000	0.35	1.41	**0%**	1.6×10^{-2}	0.072%	$\mathbf{4.16 \times 10^{-3}}$	**0%**
Child	20	1000	0.48	0.77	4.25%	$\mathbf{1.5 \times 10^{-2}}$	0.33%	$\mathbf{6.30 \times 10^{-2}}$	**0.21%**
Child	20	3000	0.87	1.63	0.34%	$\mathbf{1.6 \times 10^{-2}}$	**0.13%**	$\mathbf{7.44 \times 10^{-2}}$	0.27%
Child	20	5000	1.08	2.54	0.32%	9.24×10^{-2}	0.38%	$\mathbf{7.53 \times 10^{-2}}$	**0.19%**
Child	20	7000	1.47	3.62	0.33%	1.73×10^{-1}	**0%**	$\mathbf{7.39 \times 10^{-2}}$	**0%**
Child	20	10,000	2.03	5.56	0.32%	2.61×10^{-1}	**0%**	$\mathbf{8.87 \times 10^{-2}}$	**0%**
Insurance	27	1000	2.87	0.76	2.08%	**0.32**	1.21%	12.36	**0.72%**
Insurance	27	3000	6.26	2.37	1.20%	**0.33**	0.71%	10.44	**0.60%**
Insurance	27	5000	6.82	4.56	1.09%	**0.57**	0.63%	8.84	**0.49%**
Insurance	27	7000	9.07	6.87	0.83%	**0.89**	**0.37%**	9.13	0.45%
Insurance	27	10,000	10.16	11.05	0.67%	**0.98**	0%	9.53	**0.42%**
Water	32	1000	2.13	0.43	1.94%	**0.31**	0.21%	18.98	**0.19%**
Water	32	3000	2.98	0.59	1.58%	**0.42**	0.28%	18.55	**0.18%**
Water	32	5000	5.25	0.88	1.15%	**0.48**	0.33%	17.11	**0.11%**
Water	32	7000	5.71	1.34	1.16%	**0.63**	0.07%	19.60	**0.06%**
Water	32	10,000	7.51	1.61	0.77%	**0.97**	0.15%	21.08	**0.07%**
Alarm	37	1000	2.58	0.80	6.69%	6.33×10^{-1}	0.69%	$\mathbf{1.40 \times 10^{-1}}$	**0.03%**
Alarm	37	3000	8.61	1.32	6.21%	6.82×10^{-1}	2.17%	$\mathbf{4.96 \times 10^{-2}}$	**1.99%**
Alarm	37	5000	10.77	2.24	3.20%	1.26	0.92%	$\mathbf{7.14 \times 10^{-2}}$	**0.44%**
Alarm	37	7000	14.15	2.78	2.72%	1.41	0.25%	$\mathbf{8.42 \times 10^{-2}}$	**0.02%**
Alarm	37	10,000	22.34	4.27	2.37%	1.25	0.23%	$\mathbf{8.10 \times 10^{-2}}$	**0.21%**
Hailfinder	56	1000	8.11×10^{-1}	7.93	2.75%	5.61×10^{-1}	0.14%	$\mathbf{2.52 \times 10^{-2}}$	**0.11%**
Hailfinder	56	3000	1.92	2.74	14.49%	6.72×10^{-1}	**0.36%**	$\mathbf{3.42 \times 10^{-1}}$	0.42%
Hailfinder	56	5000	6.56	42.78	5.59%	$\mathbf{7.46 \times 10^{-1}}$	1.48%	8.26×10^{-1}	**1.35%**
Hailfinder	56	7000	34.39	138.75	3.81%	1.85	2.31%	$\mathbf{1.21 \times 10^{-1}}$	**2.21%**
Hailfinder	56	10,000	68.88	307.99	2.25%	1.56	1.60%	$\mathbf{8.61 \times 10^{-2}}$	**1.21%**
Win95pts	76	1000	256.87	1.862	8.56%	1.68	**0.83%**	$\mathbf{6.48 \times 10^{-1}}$	1.61%
Win95pts	76	3000	529.38	5.734	3.86%	2.24	2.56%	**1.08**	**2.13%**
Win95pts	76	5000	3201.13	55.928	4.91%	4.84	4.21%	**2.29**	**2.99%**
Win95pts	76	7000	2883.96	509.719	3.01%	6.36	2.89%	**2.72**	2.15%
Win95pts	76	10,000	4798.52	659.807	5.23%	8.22	3.57%	**3.28**	1.94%

Table 5 records the performances of the GOBNILP, MMHC, and A*-MM2PC algorithms on UCI datasets. Compared with the GOBNILP algorithm, the A*-MM2PC algorithm has less time consumption on most datasets. Compared with the MMHC algorithm, it has less time consumption on most datasets and has less accuracy loss. Compared with the INOBS algorithm, A*-MM2PC has less time consumption and has less accuracy loss than the INOBS algorithm on more datasets. Due to the small sample size in the UCI dataset, the accuracy of the A*-MM2PC algorithm and the MMHC algorithm in the UCI dataset is lower than that of benchmark BNs, and the overall accuracy of the A*-MM2PC algorithm is higher than that of the MMHC algorithm. The A*-MM2PC algorithm has more advantages in time and accuracy than the MMHC algorithm. For the Mushroom dataset, due to having a large number of states of each variable and a large number of POPS, the time consumption of GOBNILP and MMHC increases significantly when learning its structure, and the accuracy of the MMHC algorithm decreases significantly. However, the search space of A*-MM2PC is an order graph, which does not increase as the number of variable states and POPS

increases. In contrast, it decreases with appropriate path constraints. Therefore, A*-MM2PC has higher time efficiency than GOBNILP and MMHC on Mushroom.

Table 5. Performances of GOBNILP, MMHC, and A*-MM2PC algorithms on UCI datasets.

Name	n	N	GOBNILP	MMHC		INOBS		A*-MM2PC	
			Time(s)	Time(s)	Error	Time(s)	Error	Time(s)	Error
Lympho	19	148	1.67×10^{-1}	0.49	3.86%	1.11×10^{-1}	**0%**	$\mathbf{2.44 \times 10^{-2}}$	**0%**
Hepatitis	20	126	4.51×10^{-1}	0.32	1.47%	3.13×10^{-1}	**0%**	$\mathbf{1.86 \times 10^{-3}}$	0.26%
Segment	20	2310	13.01	2.57	2.67%	2.36	0.93%	$\mathbf{1.71 \times 10^{-1}}$	**0.76%**
Mushroom	23	8124	OT	258.33	85.33%	6.75×10^{-1}	3.43%	$\mathbf{2.55 \times 10^{-1}}$	**2.07%**
Autos	26	159	13.47	**0.85**	5.19%	1.62	2.67%	6.74	**2.60%**
Steel	28	1941	75.32	20.08	0.63%	$\mathbf{8.21 \times 10^{-1}}$	**1.52%**	1.46	2.70%
Flag	29	194	1.21	0.84	1.83%	4.71×10^{-1}	0.86%	$\mathbf{3.53 \times 10^{-1}}$	**0.69%**
Soybean	36	266	31.50	1.32	6.44%	2.25	**1.13%**	**1.16**	3.93%
Bands	39	277	6.42	0.68	2.14%	6.23×10^{-1}	1.72%	$\mathbf{2.31 \times 10^{-1}}$	**1.06%**
Spectf	45	267	4.85×10^{-1}	0.83	0.29%	3.52×10^{-1}	0.56%	$\mathbf{7.76 \times 10^{-2}}$	**0.27%**
Sponge	45	76	7.61×10^{-1}	9.12×10^{-1}	10.263%	1.78×10^{-1}	**1.296%**	$\mathbf{1.73 \times 10^{-1}}$	1.304%
LungCancer	57	32	4.68	6.32×10^{-1}	4.758%	$\mathbf{2.23 \times 10^{-1}}$	**2.806%**	3.66×10^{-1}	5.54%
Splice	61	3190	303.882	116.56	0.894%	$\mathbf{1.02 \times 10^{-1}}$	0.603%	$\mathbf{1.62 \times 10^{-1}}$	**0.323%**

5. Conclusions

Based on the A* algorithm and POPS, this paper proposes an improved A* algorithm, which is specifically divided into two aspects. On the one hand, it is called MMPC constraints, and POPS can be reduced through parent–child set constraints calculated by the MMPC algorithm. On the other hand, it is called path constraints and uses a directed graph built from POPS, and SCCs are extracted to obtain the path constraints in the order graph. The two constraints based on POPS can improve the search efficiency of A*. A large number of experiments were conducted to test the performance of the constraints. The A*-MMPC algorithm with MMPC constraints and the A*-PC algorithm with path constraints have less time consumption and fewer expanded states in the order graph and search larger networks than the original A* algorithm. Meanwhile, the A*-MM2PC algorithm with two constraints has better performance. The only drawback of both constraints is a slight loss of accuracy, most of which is no more than 0.5%. Compared with the MMPC constraints, the path constraints bring a lower loss of accuracy, and the scale of the Bayesian network that can be searched is larger. Compared with the state-of-the-art GOBNILP algorithm, A*-MM2PC has higher time efficiency in some experiments. Compared with the well-known MMHC algorithm of the hybrid algorithms, A*-MM2PC has more time efficiency and accuracy advantages in most cases. Compared with the state-of-the-art improved variant of OBS, namely, INOBS, A*-MM2PC has less time consumption and has less accuracy loss on more datasets.

Of course, the constraints we propose can not only be applied to the A* algorithm. DP, AWA*, and BiHS have the same search space as A*, only the specific search method is different. Thus, our proposed constraints can also be applied to these algorithms. The method of application is similar to adding the constraints proposed in this paper into A*. First, the number of potential optimal parent sets for DP, AWA*, and BiHS is reduced by parent–child set constraints from the MMPC algorithm. After that, path constraints are obtained from their pruned potential optimal parent sets to limit the search process of these algorithms.

However, further research questions remain. In Algorithm 2, if the size of the maximum SCC exceeds the parameter t, we prefer to select the sets that correspond to the local scores of the top k in $POPS_i$, which is a greedy strategy, leading to a slight loss in accuracy. Perhaps this can be mitigated by more reasonable rules that filter out better POPS.

In practice, we try to obtain complete and sufficient data. On this basis, local scores and POPS are calculated, POPS are pruned by the MMPC algorithm, and then path constraints files are obtained from Algorithm 2 by the POPS. The A* algorithm is modified to provide the ability to search after reading the path constraints files. Finally, the A* algorithm is invoked through a script to enable it to search for sub-networks under different path constraint files and merge the sub-networks.

In future work, we seek to obtain more useful constraints from POPS to further restrict the learning process of the BN structure to improve the performance of the BN exact structure learning algorithm on larger networks. In addition, determining whether further constraints can be obtained directly from the data is also a direction for new thinking in producing new research.

Author Contributions: Conceptualization, Methodology, C.H.; Software, R.D.; Writing—original draft, X.T. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China (61573285, 62171360), and the Natural Science Foundation of Shaanxi Province of China (2022JQ-590).

Data Availability Statement: The benchmark Bayesian networks are known, and they are publicly available (http://www.bnlearn.com/bnrepository (accessed on 20 November 2022)). The UCI datasets are publicly available (https://archive.ics.uci.edu/ml/index.php (accessed on 20 November 2022)).

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

References

1. Chickering, D.M. Learning Bayesian Networks is NP-Complete. In *Learning from Data: Artificial Intelligence and Statistics V*; Springer: Berlin/Heidelberg, Germany, 1996; Volume 112, pp. 121–130.
2. Chickering, D.M.; Heckerman, D.; Meek, C. Large-sample learning of Bayesian networks is NP-hard. *J. Mach. Learn. Res.* **2004**, *5*, 1287–1330.
3. Kitson, N.K.; Constantinou, A.C.; Guo, Z.; Liu, Y.; Chobtham, K. A survey of Bayesian Network structure learning. *Artif. Intell. Rev.* **2023**, *1*, 1–94. [CrossRef]
4. Spirtes, P.; Glymour, C. An algorithm for fast recovery of sparse causal graphs. *Soc. Sci. Comput. Rev.* **1991**, *9*, 62–72. [CrossRef]
5. Pearl, J.; Verma, T.S. A theory of inferred causation. In *Studies in Logic and the Foundations of Mathematics*; Elsevier: Amsterdam, The Netherlands, 1995; Volume 134, pp. 789–811.
6. Margaritis, D.; Thrun, S. Bayesian network induction via local neighborhoods. In Proceedings of the 12th International Conference on Neural Information Processing Systems, Denver, CO, USA, 29 November–4 December 1999; pp. 505–511.
7. Cooper, G.F.; Herskovits, E. A Bayesian method for the induction of probabilistic networks from data. *Mach. Learn.* **1992**, *9*, 309–347. [CrossRef]
8. Heckerman, D.; Geiger, D.; Chickering, D.M. Learning Bayesian networks: The combination of knowledge and statistical data. *Mach. Learn.* **1995**, *20*, 197–243. [CrossRef]
9. Behjati, S.; Beigy, H. Improved K2 algorithm for Bayesian network structure learning. *Eng. Appl. Artif. Intel.* **2020**, *91*, 103617. [CrossRef]
10. De Campos, L.M.; Huete, J.F. Approximating causal orderings for Bayesian networks using genetic algorithms and simulated annealing. In Proceedings of the Eighth International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, Kansas City, MO, USA, 2–6 November 1999; pp. 333–340.
11. Teyssier, M.; Koller, D. Ordering-Based Search: A Simple and Effective Algorithm for Learning Bayesian Networks. In Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence, Edinburgh, Scotland, 26–29 July 2005; pp. 584–590.
12. Larranaga, P.; Kuijpers, C.M.; Murga, R.H.; Yurramendi, Y. Learning Bayesian network structures by searching for the best ordering with genetic algorithms. *IEEE Trans. Syst. Man Cybern. Part A Syst. Hum.* **1996**, *26*, 487–493. [CrossRef]
13. Gheisari, S.; Meybodi, M.R. BNC-PSO: Structure learning of Bayesian networks by particle swarm optimization. *Inform. Sci.* **2016**, *348*, 272–289. [CrossRef]
14. Daly, R.; Shen, Q. Learning Bayesian network equivalence classes with ant colony optimization. *J. Mach. Learn. Res.* **2009**, *35*, 391–447. [CrossRef]
15. Koivisto, M.; Sood, K. Exact Bayesian structure discovery in Bayesian networks. *J. Mach. Learn. Res.* **2004**, *5*, 549–573.
16. Silander, T.; Myllymaki, P. A simple approach for finding the globally optimal Bayesian network structure. In Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence, Cambridge, MA, USA, 13–16 July 2006; pp. 445–452.
17. Malone, B.; Yuan, C. Memory-efficient dynamic programming for learning optimal Bayesian networks. In Proceedings of the 25th AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 7–11 August 2011; pp. 1057–1062.

18. Yuan, C.; Malone, B.; Wu, X. Learning optimal Bayesian networks using A* search. In Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Catalonia, Spain, 16–22 July 2011; pp. 2186–2191.
19. Yuan, C.; Malone, B. Learning optimal Bayesian networks: A shortest path perspective. *J. Artif. Intell. Res.* **2013**, *48*, 23–65. [CrossRef]
20. Malone, B.; Yuan, C. Evaluating anytime algorithms for learning optimal Bayesian networks. In Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence, Bellevue, WA, USA, 11–15 August 2013; pp. 381–390.
21. Tan, X.; Gao, X.; Wang, Z.; He, C. Bidirectional heuristic search to find the optimal Bayesian network structure. *Neurocomputing* **2021**, *426*, 35–46. [CrossRef]
22. Van Beek, P.; Hoffmann, H.F. Machine learning of Bayesian networks using constraint programming. In Proceedings of the International Conference on Principles and Practice of Constraint Programming, Cork, Ireland, 31 August–4 September 2015; pp. 429–445.
23. Barlett, M.; Cussens, J. Advances in Bayesian network learning using integer programming. In Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence, Bellevue, WA, USA, 11–15 August 2013; pp. 182–191.
24. Bartlett, M.; Cussens, J. Integer linear programming for the Bayesian network structure learning problem. *Artif. Intell.* **2017**, *244*, 258–271. [CrossRef]
25. Trösser, F.; De Givry, S.; Katsirelos, G. Improved Acyclicity Reasoning for Bayesian Network Structure Learning with Constraint Programming. In Proceedings of the 30th International Joint Conference on Artificial Intelligence, Montreal, QC, Canada, 19–27 August 2021; pp. 4250–4257.
26. Zheng, X.; Aragam, B.; Ravikumar, P.; Xing, E.P. DAGs with NO TEARS: Continuous optimization for structure learning. In Proceedings of the 32nd International Conference on Neural Information Processing Systems, Montreal, QC, Canada, 3–8 December 2018; pp. 9492–9503.
27. Tsamardinos, I.; Brown, L.E.; Aliferis, C.F. The max-min hill-climbing Bayesian network structure learning algorithm. *Mach. Learn.* **2006**, *65*, 31–78. [CrossRef]
28. Tsamardinos, I.; Aliferis, C.F.; Statnikov, A. Time and sample efficient discovery of markov blankets and direct causal relations. In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, Washington, DC, USA, 24–27 August 2003; pp. 673–678.
29. Perrier, E.; Imoto, S.; Miyano, S. Finding Optimal Bayesian Network Given a Super-Structure. *J. Mach. Learn. Res.* **2008**, *9*, 2251–2286.
30. Kojima, K.; Perrier, E.; Imoto, S.; Miyano, S. Optimal Search on Clustered Structural Constraint for Learning Bayesian Network Structure. *J. Mach. Learn. Res.* **2010**, *11*, 285–310.
31. Gámez, J.A.; Mateo, J.L.; Puerta, J.M. Learning Bayesian networks by hill climbing: Efficient methods based on progressive restriction of the neighborhood. Data. *Mib. Knowl. Disc.* **2011**, *22*, 106–148. [CrossRef]
32. Liu, H.; Zhou, S.; Lam, W.; Guan, J. A new hybrid method for learning Bayesian networks: Separation and reunion. *Knowl.-Based Syst.* **2017**, *121*, 185–197. [CrossRef]
33. Kuipers, J.; Suter, P.; Moffa, G. Efficient sampling and structure learning of Bayesian networks. *J. Comput. Graph. Stat.* **2022**, *31*, 639–650. [CrossRef]
34. Rissanen, J. Modeling by shortest data description. *Automatica* **1978**, *14*, 465–471. [CrossRef]
35. Tian, J. A branch-and-bound algorithm for MDL learning Bayesian networks. In Proceedings of the Sixteenth conference on Uncertainty in Artificial Intelligence, San Francisco, CA, USA, 30 June–3 July 2000; pp. 580–588.
36. Lee, C.; Beek, P.V. Metaheuristics for score-and-search Bayesian network structure learning. In Proceedings of the Canadian Conference on Artificial Intelligence, Edmonton, AB, Canada, 16–19 May 2017; pp. 129–141.