Learning the structure of Bayesian networks with ancestral and/or heuristic partition

Xiangyuan Tan *a*, Xiaoguang Gao *a* [[1]](#footnote-1)\*, Zidong Wang *a*, Hao Han *a*, Xiaohan Liu *a*, Daqing Chen *b*

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

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

Abstract

Developing efficient strategies for searching larger Bayesian networks in exact structure learning is an open challenge. In this study, ancestral and heuristic partition constraints are proposed to develop a series of exact learning algorithms, in which an ancestral partition is used to prune the order graph of a Bayesian network, and a heuristic partition is utilized to improve the tightness of the heuristic function. Algorithms for calculating these two types of constraints are established through thorough theoretical proof. Comparative experiments have been undertaken with state-of-the-art algorithms. It has been demonstrated that an algorithm improved with the proposed ancestral partition or combined ancestral and heuristic partition outperforms the algorithm in its original form, and it can have lower running time, fewer expanded states, and higher accuracy, as well as the ability to search larger networks within 100 nodes.

**Keywords:** Bayesian network; structure learning; order graph; heuristic function;

# 1. Introduction

Artificial intelligence (AI) has penetrated all aspects of our life with great impact. As one of the most popular and widely used techniques in AI, deep learning has a broad range of applications in image processing[1][2], natural language processing[3], autonomous driving[4], and industrial process[5], etc. However, deep learning is not a panacea, and it remains difficult and challenging for researchers to understand the causal relationships of data[6]. The Bayesian network (BN), as a mathematical modelling tool to support such a causal structure representation and analysis in AI, has attracted considerable research attention for applications in reliability assessment[7], fault detection[8], multi-label learning[9], and other fields.

A BN is a probabilistic graph model based on probability and graph theories. A BN can be expressed as a directed acyclic graph (DAG), in which each node represents a random variable, and the directed edges between nodes represent the dependencies between variables, and each node is further quantified by a set of conditional probability distributions. In general, a BN represents the joint probability distribution of a set of random variables.

Learning a BN includes two parts—structure learning and parameter learning. Since parameter learning only needs to be performed when a DAG structure is known, BN structure learning is the foundation of BN parameter learning. Compared with parameter learning, structure learning requires higher accuracy because redundant directed edges can result in incorrect domain structure and dependence and an increase in the number of probability parameters to be learned in parameter learning. In addition, higher accuracy is required in parameter learning as a compensation for the error caused by the lack of "correct" directed edges. Therefore, BN structure learning is the core of BN learning.

BN structure learning determines the topological relationships between the nodes in a BN using a sample dataset available, and this remains a challenge and has therefore been extensively studied over the past few decades. However, learning the optimal BN structure from data has proved to be NP-hard[10] [11]. The space of the possible network structures grows super-exponentially with the number of variables. Usually, BN structure learning algorithms can be grouped into two categories—approximate and exact learning algorithms—based on learning results. Approximate learning algorithms are fast and suitable for larger networks, but they tend to fall into local optima easily and rarely obtain the optimal BN structure. By contrast, exact learning algorithms can theoretically ensure that the learned network is optimal. However, these algorithms are less efficient than approximate learning algorithms and cannot be applied to the structure learning problem for larger networks.

Approximate learning algorithms can be divided into score-based, constraint-based, and hybrid search algorithms[12]. Score-based algorithms are the most commonly used approximate learning algorithms. Such an algorithm uses a scoring function to estimate the quality of BN structures and consequently searches for a network structure with the best score. A scoring function is used to measure the goodness of fit of a DAG structure to a given dataset. These algorithms include K2[13], Hill-Climbing (HC)[14], Simulated Annealing (SA)[15], and Ordering-Based Search (OBS)[16] algorithms. The core idea of these algorithms is to start from an initial state and update the state through several operations so that the score of the new state can gradually approach the optimal score gradually. Such algorithms can easily fall into local optima owing to the greedy strategies applied. Constraint-based algorithms determine the conditional independent relations among variables by performing conditional independence (CI) tests from data and then build a BN structure that satisfies the independence relations. Common constraint-based algorithms include Sparse Causal Graphs (SCG)[17], Peter and Clark (PC)[18], and Grow-Shrink (GS)[19] algorithms. In practice, however, it is difficult to determine whether any two variables are conditionally independent. CI tests require a large number of samples and are sensitive to noise. If there are insufficient or noisy data, they may perform poorly, and an error in these CI tests has a cascading effect in the subsequent learning process. Hybrid search algorithms combine the advantages of score-based and constraint-based algorithms. The most well-known hybrid algorithm is the Max-Min Hill-Climbing (MMHC)[20] algorithm. MMHC first uses the Max-Min Parents and Children (MMPC)[21] algorithm to learn the parent and children sets of each node in the BN and then uses the HC algorithm to determine the direction of edges in the network. The MMHC algorithm cannot guarantee a global optimal network because it also uses CI tests and can easily fall into local optima. Recently, novel approximate algorithms have been proposed. For example, Non-combinatorial Optimization via Trace Exponential and Augmented lagRangian for Structure learning (NOTEARS)[22] and DAG-GNN[23]. The NOTEARS algorithm formulates the structure learning problem as a purely continuous optimization problem. The DAG-GNN extends NOTEARS by incorporating graph neural network functions and variational inference such that the score function is the evidence lower bound.

Exact learning algorithms inherit the idea of score-based algorithms. The difference is that these algorithms can theoretically ensure that the obtained network is optimal. Koivisto[24] proposed an order graph search space for the exact learning of BN structures and used a dynamic programming (DP) algorithm to search the order graph. Singh[25] and Silander[26] improved the DP algorithms. Malone[27] performed DP by leveraging the layered structure present within the order graph and proposed Memory-Efficient Dynamic Programming (MEDP), which dramatically improves efficiency. These DP algorithms require a complete search in the order graph, and the search space increases exponentially with the variables; therefore, these algorithms always belong to the exponential algorithm category. Later, Yuan and Malone[28] formulated the structure learning problem of BN as the shortest path problem in the order graph. They proposed the A\*[29], Anytime Window A\* (AWA\*)[30], and Breadth-First Branch and Bound search (BFBnB)[31] algorithms, and introduced heuristic functions for these heuristic searches[32]. These heuristic functions are admissible[[2]](#footnote-2) and consistent[[3]](#footnote-3), ensuring that these heuristic search algorithms eventually converge to the optimal score. Tan[33] continued the idea of shortest path and proposed the bidirectional heuristic search (BiHS). Compared with the previous DP algorithms, these algorithms based on the shortest path search need only to perform a partial search in the order graph and have a higher efficiency than DP algorithms. In addition, some algorithms based on mathematical programming have achieved good results. For example, CPBayes[34] defines the structure learning problem as a constraint programming problem. Other examples include Globally Optimal Bayesian Network learning using Integer Linear Programming (GOBNILP) [35][36]. This algorithm expresses the structure learning problem as an integer linear programming (ILP) problem, which can be solved efficiently using existing ILP frameworks such as SCIP. Malone[37] conducted comparative experiments on three algorithms, A\*, CPBayes, and GOBNILP (the old version), and the results showed that no single algorithm dominates the others in terms of efficiency. A new version of GOBNILP is considered the current state-of-the-art approach for exact structure learning[22][23][38][39].

In recent years, structure learning algorithms based on partitions have also been developed. The Separation And Union (SAR)[40] algorithm decomposes the task of learning a large BN into learning some relatively small BNs using a few CI tests and builds the actual BN by re-unifying these small BNs. Zhao[41] used mutual information to divide the learning process into stages. At each stage, a subnet is learned over a selected subset of the variables. The selected subset grows in stages and eventually includes all the variables. The Layered Optimal Learning (LOL)[42]algorithm uses CI tests to layer nodes of BN and then use MEDP to learn the structure for each layer. Li[43] used an improved *k*-means algorithm with mutual information to partition the network, and then MMPC and DP were performed to learn the partitioned network. Decomposition-based BN structure learning algorithm (Local-DSL)[44] uses local topology information to decompose the task of reconstructing a whole BN into a series of subtasks of learning small DAGs, and the entire structure can be obtained by merging such locally learned subgraph.

Most of the algorithms mentioned above use the CI tests for partitioning. However, given insufficient samples, CI tests have some probability of making errors. The accuracy of the subsequent structure learning is not guaranteed when incorrect results are used to guide the partitioning. In addition, most of the above algorithms use approximate learning algorithms to learn the partitioned parts, which cannot guarantee a global optimal network. Even if exact learning algorithms such as DP and MEDP are used, these algorithms are still limited by the drawbacks of CI tests. Moreover, many exact learning algorithms are more efficient than DP/MEDP algorithms. In summary, there is plenty of room for further research on partition-based BN structure learning algorithms.

Inspired by the idea of partitioning, this study aims to improve the efficiency of exact learning by examining and employing two partition constraints—ancestral partition and heuristic partition. The rationale for this research is as follows: First, many of the aforementioned algorithms use the concept of ordering[13][16][24][25][26][27][28][29][30][31][33]. Ordering represents a strict constraint, and if an optimal ordering is obtained, the optimal BN can be established in a few simple steps. However, finding an optimal ordering is still a challenge because the search space of candidate orderings or the order graph (a graphical ordering space) is very large. As such, this research is concentrated on a generalized constraint, namely, the ancestral partition, which is easier to obtain than ordering. Second, some experimental results[29][32] show that different partitions of the variable set of the BN impact the heuristic function and affect the efficiency of order graph-based heuristic algorithms. To address this issue, this study explores a better partition to improve the tightness of the heuristic function, namely, the heuristic partition.

In summary, the essential contributions of this paper are as follows:

* An ancestral partition, a generalized and easier-to-obtain constraint, is established. The ancestral partition can prune an order graph and divide the entire learning process into stages, thereby improving the efficiency of order graph-based algorithms. A thorough theoretical analysis is provided to prove that order graph-based algorithms with consistent ancestral partition can still find the optimal BN. In addition, an algorithm is developed to calculate the ancestral partition. It is shown that algorithms with our calculated ancestral partition can have higher efficiency and can search for much larger BNs than the original algorithms.
* A heuristic partition is proposed to improve the tightness of a heuristic function along with an algorithm for calculating the heuristic partition. Algorithms with our calculated heuristic partition have higher efficiency and search for much larger BNs than before. Since the heuristic function is admissible and consistent, algorithms with a heuristic partition can always obtain the optimal solution.

The remainder of this paper is organized as follows. The preliminaries of the BNs and the exact learning algorithms in the order graph are presented in Section 2. In Section 3, the ancestral partition and its calculation algorithm are introduced, and related theorems and proofs are provided. Section 4 describes the heuristic partition and its calculation algorithm. In Section 5, we conduct various experiments to evaluate the proposed partition constraints and analyze them. Finally, the concluding remarks are presented in Section 6.

# 2. Preliminaries

This section presents the preliminaries of the BNs and the exact learning algorithms within the order graph.

## 2.1 Bayesian networks

A BN  is a type of probability graph model that can be expressed by DAG,, and the joint probability distribution , where  represents all the nodes in the graph, corresponding to each random variable, and  is the number of nodes, and  is a directed edge set, indicating the dependence relations between each node. If there is a directed edge  in a DAG, then  is identified as a parent node of , and  is a child node of . Let  represent the parent set of . Starting from node , if there is a path to reach node  along directed edges, then  is said to be an ancestor node of , and  is a descendant node of . The joint probability distribution  of an entire BN  can be decomposed into the product of each conditional probability as:

|  |  |  |
| --- | --- | --- |
|  |  | （1） |

BN structure learning aims to find the optimal BN structure  that best reflects the dependence among variables in the complete dataset , where  is the number of samples. This problem is known to be NP-hard[10][11]. Related works show that, except for constraint-based algorithms, most of the algorithms use a scoring function to measure the quality of the BN structure based on the dataset . Common scoring functions include the Bayesian information criterion (BIC)[45], Bayesian Dirichlet score with score equivalence (BDe)[46], minimum description length (MDL)[47], and Akaike information criterion (AIC) [48]. Usually, the smaller the MDL score, the better the BN structure and for the other scoring functions, the higher the other scores, the better the network structure. The fundamental goal of this paper is to study the BN structure exact learning algorithm from the viewpoint of the shortest path on the order graph. Therefore, this study uses the MDL score to measure the quality of a learned BN structure (note that related theories also use the MDL score). Hence, learning the optimal BN structure can be expressed as

|  |  |  |
| --- | --- | --- |
|  |  | （2） |

The aforementioned scoring functions have a fundamental property: these scoring functions are decomposable. Therefore, a BN score can be decomposed into the sum of the scores of each node. The MDL score of the BN structure can be decomposed into

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | （3） |

where

|  |  |  |
| --- | --- | --- |
|  |  | （4） |
|  |  | （5） |
|  |  | （6） |

In the equations above, the state space of  is denoted as , and the state space of  is denoted as .  denotes the log-likelihood of  and its parent set.  is the maximum likelihood estimate of the conditional probability .  and  represent the number of data records consistent with  and  in the dataset , respectively.  is the complexity penalization of the BN structure complexity. For simplicity, the symbol "" in the scoring function is omitted in the following equations.

Notably, the above scoring functions are score-equivalent. Multiple BN structures that represent the same probability distribution with all possible parameterizations are called equivalence classes. It has been proven that BN structures of equivalence classes have the same score value under the same score-equivalent scoring function. Therefore, the optimal BN structure  is not unique, and finding one of them is sufficient.

## 2.2 Exact structure learning in order graph

Before introducing the exact learning algorithms for BN structure learning based on an order graph, let us first consider the term of ordering. The concept of ordering was first discussed in the K2[13] algorithm and has been applied to many algorithms such as OBS[16], DP[26][27], and A\*[28][29], etc. This article defines the ordering as follows:

**Definition 1.** (Ordering) In a BN of random variables , a relation  () is specified. For any ,  (),  and . Then, this relation is called an ordering, indicating that the structure learning of the BN must be performed under the following condition: For any , the possible parent sets of  are selected from the subsets of .

**Definition 2.** (Optimal ordering) In a BN of random variables , an ordering  is specified. If for any element  in this ordering, the optimal parent set of the variable  in the BN always satisfies , then  is called consistent with the BN and considered the optimal ordering.

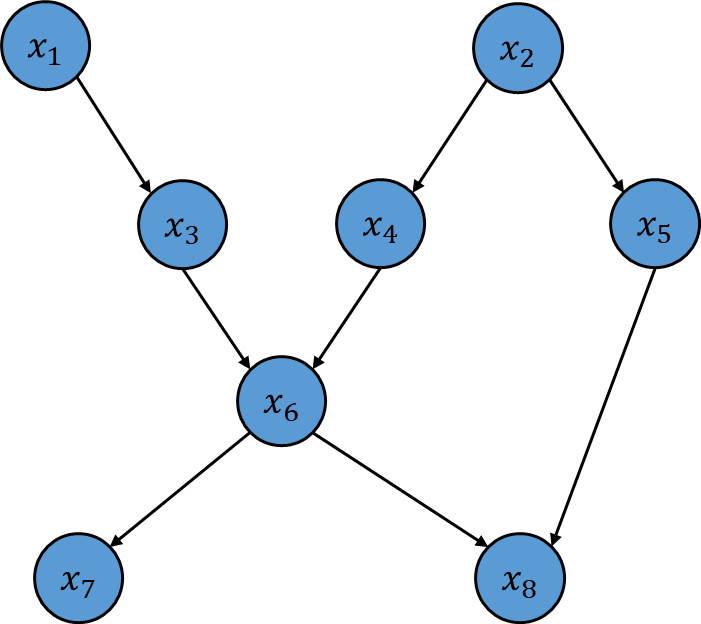


Figure 1: Asia network structure.

An ordering poses constraints on the range of possible parent sets of each node in a BN. In the structure learning process under an ordering, only the possible parent sets of a node that belong to the set of nodes preceding it can be selected. If the relation in the ordering is consistent with the actual network structure, the ordering is considered optimal. Consider the classic Asia BN as shown in Fig. 1. Both  and  are orderings, but the former is an optimal ordering, while the latter is not because it does not satisfy Definition 2. The optimal ordering of the Asia network can be , ,  or other orderings. A BN can have multiple optimal orderings. Notably, in this study, we are interested in finding the optimal ordering that is close to the lexicographic order. For example, for the Asia network, we prefer to use  to represent the corresponding optimal ordering.

The DP algorithm is derived from the decomposability of the scoring function. The DP algorithm aims to decompose the task of searching for the optimal BN structure into finding the optimal sub-network until reaching the empty set and recursively find the entire optimal network. More formally, these recursive formulas exist as

|  |  |  |
| --- | --- | --- |
|  |  | （7） |
|  |  | （8） |

The above recursive relations show that the optimal network structure score for the variable set  can be decomposed into two parts: (1) the optimal score for the sub-network on the variable set , and (2) the optimal score for choosing  an optimal parent set from . Then, the variable set  can be decomposed according to Eqs. (7) and (8) until the empty set is obtained. Given the above recurrence relations, the procedure of DP is as follows: Starting from the empty set, it searches for the optimal structures for every individual variable. After adding variables into these structures, it builds optimal sub-networks with increasingly larger variable sets until the optimal one for  is found. This entire process can be represented by an order graph, and as an example, the order graph of a four-variable BN is shown in Fig.2. DP algorithms can find an optimal BN in  time and space[24][25][26] and must search for a total of  states in the order graph (the initial state  need not be searched). The MEDP[27] algorithm reduces memory consumption by leveraging the layered structure of the order graph. However, a complete search of the order graph is still required.

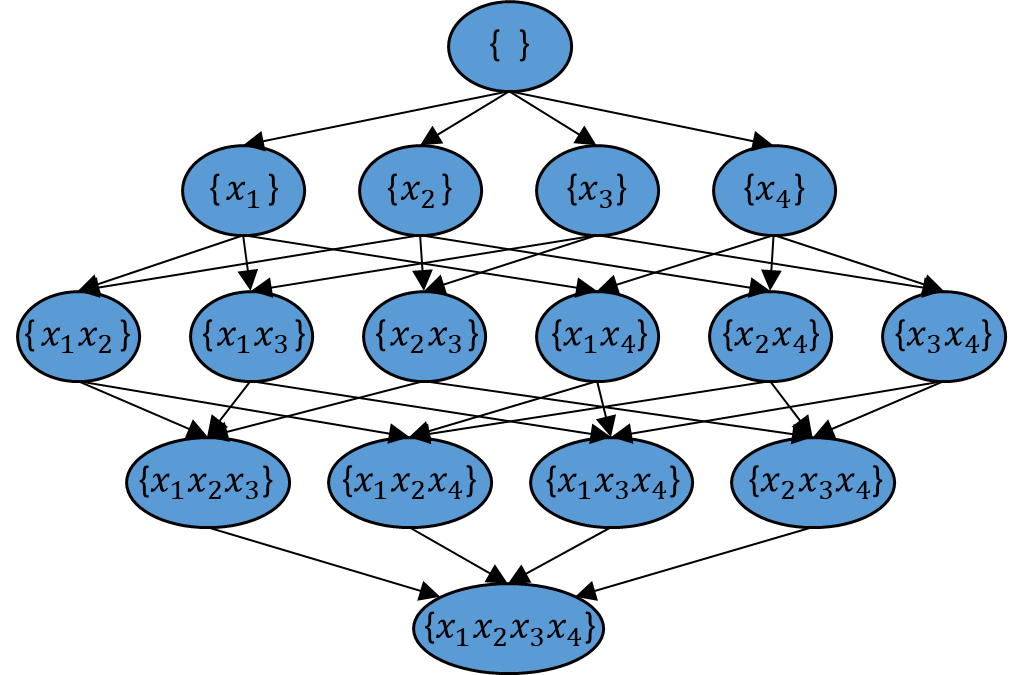


Figure 2: Order graph of a four-variable Bayesian network.

In the order graph, the directed path from the initial state  at the top to the goal state  at the bottom is the sequence in which the variables appear, corresponding to an ordering. Thus, the order graph obtains its name. Based on the above phenomena, Malone and Yuan proposed a series of algorithms[29][30][31] from the shortest path perspective. Now that a path from  to  corresponds to an ordering, the shortest path from  to  corresponds to the optimal ordering. For example, the optimal path  corresponds to the optimal ordering  of the Asia network. Therefore, the shortest path can be searched in the order graph and the optimal parent set for each node in the process can be recorded. Finally, an optimal BN structure can be built using the ordering and optimal parent sets. From the shortest path perspective, the path cost in the order graph from the current state  to the successor state  is

|  |  |  |
| --- | --- | --- |
|  |  | （9） |

In the A\* algorithm, for the current state , the generated path cost  from the initial state  to  is calculated, and a heuristic function  is used to estimate the cost  from the current state  to the goal state . In the search process,  is used to estimate the optimal cost of the path through . A priority queue  is used to store a list of states to be expanded, sorted by the *f* -value, and  is used to store the already-expanded states. At each search step, the current state with the lowest *f* -value in  is selected to be expanded, the current state is put into , and the states that are successors of the current state are moved to . Until  is expanded, the shortest path from  to  is found. Thus, optimal ordering and the optimal BN structure are found.

In the AWA\* algorithm, a sliding window search strategy is introduced in A\* to search the order graph over several iterations. AWA\* uses a sliding window to encourage a deeper search of the order graph. This algorithm uses the parameter  to control the window size, which increases with the number of iterations and keeps track of the depth of all states. The states are expanded as usual by A\*. After the states of the layer (the cardinality of the set corresponding to the state is its number of layers) are expanded, all states in a layer less than  are placed in . After one iteration ends, a path from  to  is found, from which an upper bound is obtained. The lowest *f* -value in  is considered to be the lower bound. Then, the window size  is increased by one, and the states in  are placed into . The entire process continues until the upper and lower bounds converge to obtain an optimal solution.

In the BFBnB algorithm, a fast approximate algorithm, such as the HC algorithm, is used in advance to rapidly obtain the upper bound. The states in the order graph are expanded in layers using the breadth-first search strategy. If the current state has a higher *f* -value than the upper bound during the search, then the search for the state  and its successors can be safely abandoned without losing accuracy.

The three algorithms A\*, AWA\*, and BFBnB have heuristic functions, so they do not need to search the entire order graph. The efficiency of these three algorithms is higher than that of the DP algorithms. The heuristic functions of these algorithms are admissible, ensuring that these algorithms can converge to the optimal score. In addition, these heuristic functions are consistent and can converge faster than non-consistent heuristic functions.

# 3. Ancestral partition constraints

## 3.1 Ancestral partition

As mentioned earlier, multiple optimal orderings are consistent with the Asia network. However, ordering is a strict constraint, and finding one of the multiple optimal orderings is still a challenge. As an alternative, in this study, we attempt to formulate a general constraint instead of ordering. With the Asia network in Fig. 1 as an example, the new constraints are defined as follows:  and  should be ancestors of all other nodes, , , and  should be ancestors of , , and . Under such constraints, we can obtain multiple optimal orderings. These nodes indicate the relationships between ancestors and descendants. Based on this simple example, we define the following:

**Definition 3.** (Ancestral partition) In a BN of random variables , a relation  (, ) is specified. For any ,  (), , and . Then, this relation is called the ancestral partition, indicating that the structure learning of the BN must be performed under the following condition: For any , the possible parent sets of  () are selected from the subsets of .

**Definition 4.** (Consistent ancestral partition) In a BN of random variables , an ancestral partition  is specified. If for any set  in this ancestral partition, the optimal parent set of any variable  () in the BN always satisfies , then  is called the ancestral partition consistent with the BN or consistent ancestral partition.

Let  and  denote the partition block and its partition block size, respectively. Definitions 3 and 4 can be regarded as extensions of Definitions 1 and 2, respectively. If for any partition block  in Definitions 3 and 4, , then Definitions 3 and 4 become Definitions 1 and 2. Therefore, an ancestral partition can be considered a generalized ordering. Similar to ordering, the ancestral partition poses constraints on the range of possible parent sets. If the relation in an ancestral partition is consistent with the actual BN structure, that is, if the optimal parent set of each node exists in the possible parent sets derived from the ancestral partition, the ancestral partition is consistent.

We are still using the Asia network as an example to explain the ancestral partition. The possible consistent ancestral partitions are  or . Similar to the optimal ordering, the consistent ancestral partition of a BN is not unique. An example of an Asia network based on the consistent ancestral partition  is shown in Fig.3. The consistent ancestral partition represents a partition relation for the BN structure, and this partition relation also reflects the priority order for BN structure learning. If the sub-network structure of  cannot be determined, it would not be possible to build the sub-network structure of . Only by determining the sub-network structure of  can it be possible to further learn the sub-network structure of .

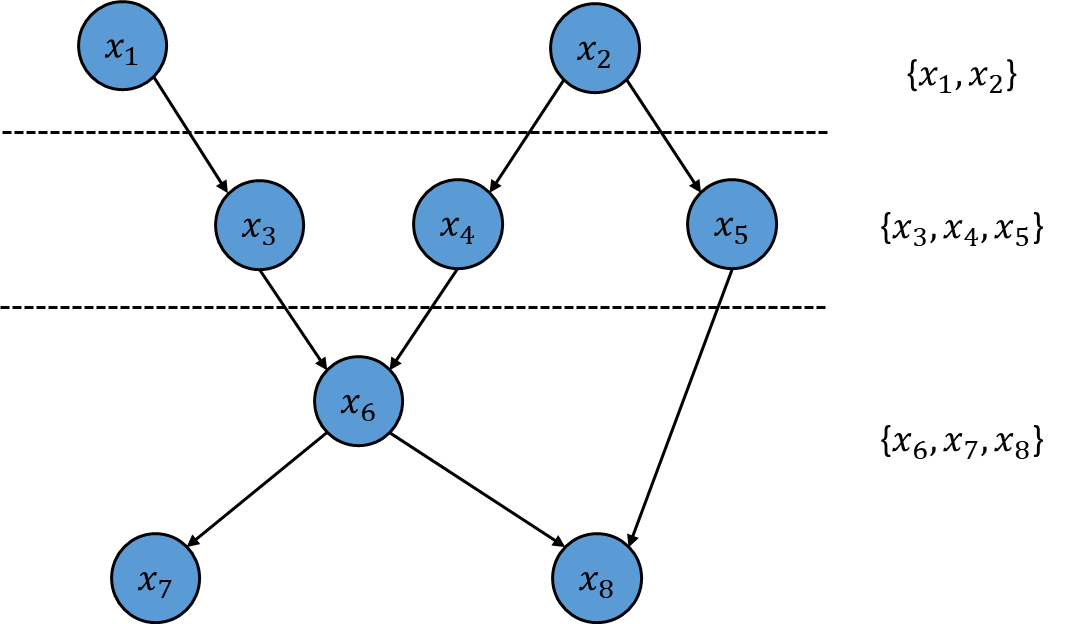


Figure 3: Example for Asia network according to ancestral partition .

Since the (consistent) ancestral partition is more generalized than the (optimal) ordering, obtaining the ancestral partition would be easier than obtaining the ordering. Therefore, introducing an easier-to-obtain ancestral partition constraint into the BN structure learning problem would be beneficial for improving the efficiency of structure learning algorithms.

## 3.2 Ancestral partition for order graph

We know that an optimal ordering corresponds to the (optimal) path in the order graph. Therefore, does an ancestral partition correspond to a more "abstract" path in an order graph than a generalized ordering?

Continuing with the Asia network as an example, without any constraints, the order graph corresponding to the Asia network is similar to that in Fig. 2, as shown in Fig. 4. In such an order graph, there are  states. Such an order graph grows exponentially with the number of nodes and many search algorithms based on the order graph must search a vast state space. This has led to the poor efficiency of these algorithms.

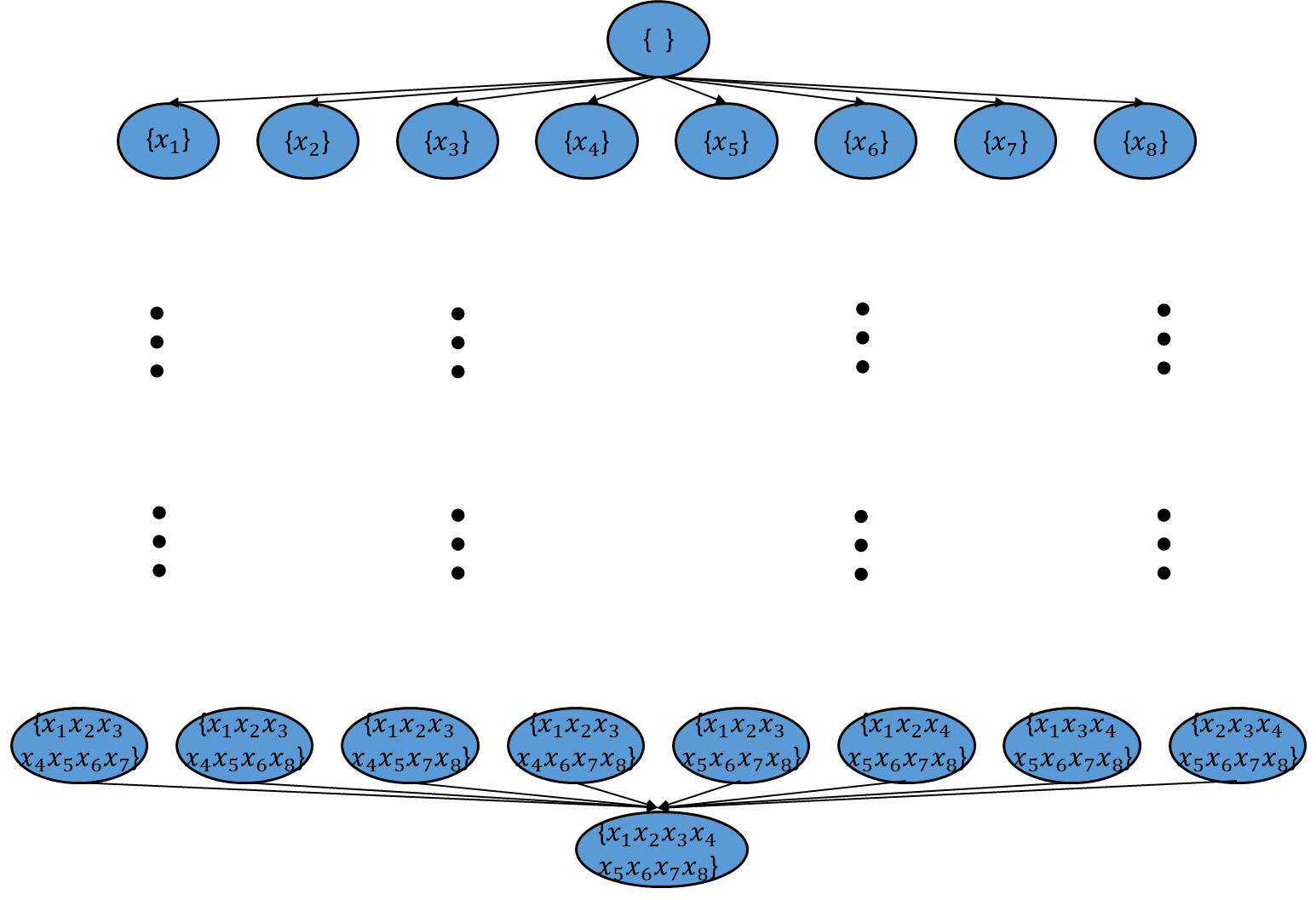


Figure 4: Order graph for Asia network without any constraints.

We assume that a consistent ancestral partition  is known. To create an order graph, the sub-network of  should be built first, followed by the sub-network of , and finally the sub-network of . Therefore, the complete order graph in Fig.4 can be pruned with ancestral partition constraints, as shown in Fig. 5.

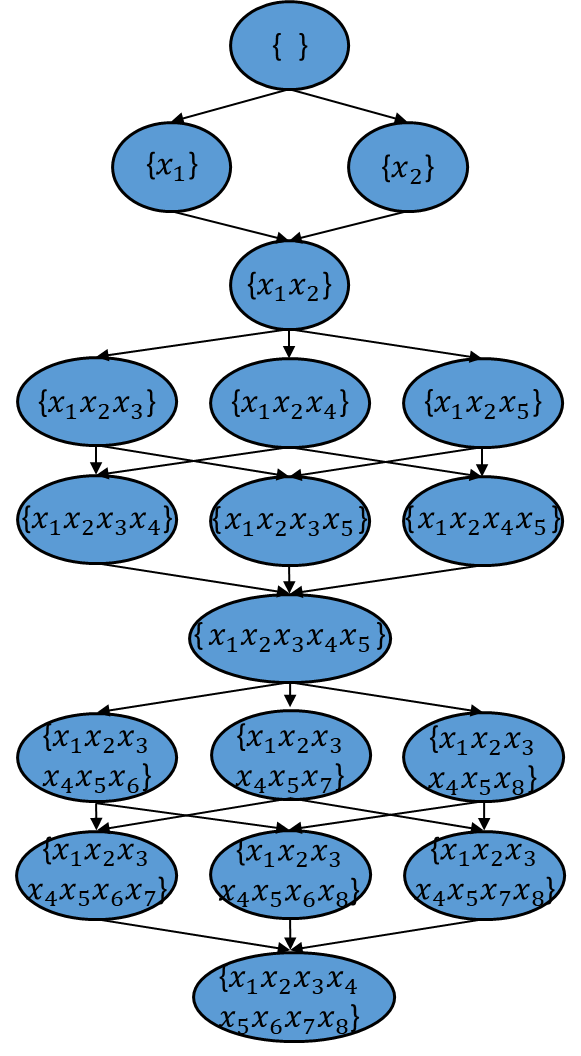


Figure 5: Pruned order graph for Asia network with ancestral partition constraints .

As shown in the pruned order graph in Fig. 5, there are only 18 states. The state space is significantly reduced under the ancestral partition constraints, and as a result, the efficiency of search algorithms based on the order graph is improved. The relay states , , and  can be considered the points that must be visited when searching for the optimal path. This undoubtedly strengthens the restriction on the search for the optimal path so that the optimal path can be obtained faster. In addition, states ,  and  can be regarded as stages in the entire learning process. Given , we can use any algorithm based on the order graph to search the subgraph of , then the subgraph of ,..., and finally the subgraph of . The final BN can be obtained by combining the results of each stage.

Several problems remain to be solved for ancestral partition. For the order graph of the Asia network in Fig. 4, does the corresponding optimal path still exist in the order graph in Fig. 5 ? In other words, has any state in the optimal path of the Asia network been pruned during the change in the order graph under a consistent ancestral partition? Theorem 1 provided below answers this question.

**Lemma 1.** In a BN on random variables , an ordering  and an ancestral partition  are consistent with this BN. For any  in the ancestral partition, and  () must exist such that .

***Proof.***According to Definition 2, Definition 4, and the analysis in Section 3.1, the optimal ordering is a particular case of a consistent ancestral partition. Therefore, for any partition block  in the consistent ancestral partition, one node or several adjacent nodes must be found in the consistent ordering to form .

∎

**Lemma 2.** If the ancestral partition  (,) is consistent with the BN of random variables , the order graph of  can be changed to the pruned order graph of  (,, ). Then, for any state  of the optimal path  (, , ) in the order graph of ,  and  must exist satisfying .

***Proof.*** The ancestral partition  is consistent with the BN. The path  is optimal; thus, its corresponding ordering  () is also optimal and consistent with the BN. According to Lemma 1,  must exist such that  and . Therefore, , , and , the relation  holds.

∎

**Theorem 1.** In a BN on random variables , if the ancestral partition  (,) is consistent, the order graph of  can be changed to the pruned order graph of  (, , ). Then, for the optimal path  (, , ) in the order graph of , it will not be pruned in the pruned order graph of .

***Proof.*** According to Lemma 2, if the ancestral partition  is consistent with the BN, any state  in the optimal path will be located between adjacent stages  and . In other words, any state in the optimal path still exists in the pruned order graph. Therefore, the optimal path will not be pruned.

∎

Theorem 1 shows that the optimal path will not be pruned in a pruned order graph if an ancestral partition is consistent. The BN structure found in the order graph, pruned by the consistent ancestral partition, has the same optimal score as that of the BN structure found in the original order graph. It is guaranteed that the found network is still optimal, and the change in the order graph will not affect the search for the optimal BN. If the ancestral partition is not consistent, then all the above results will not be guaranteed, and the optimal network will not be found.

**Theorem 2.** In a BN of random variables , if through the ancestral partition  (, ), the order graph of  is pruned to become the pruned order graph of , where  (, ). Then, the number of states to be searched in the order graph is reduced from  to .

***Proof.*** The complete order graph shown in Figs. 2 and 4 has a total of  states. For the initial state , no search is required, and a total of  states must be searched. For the pruned order graph with the ancestral partition  in any subgraph , there are  states. There are  subgraphs of the order graph and  total states. However, this method repeatedly calculates the states  and does not require an initial state . Hence, a total of  states must be pruned from the calculation. Finally, the number of states to be searched is .

∎

Theorem 2 shows that ancestral partition constraints can reduce the number of states in the order graph. Searching in the pruned order graph changed by the ancestral partition is more efficient than searching in the original order graph. In particular, for any partition block  in the ancestral partition , , that is, when the ancestral partition becomes an ordering, , , and the number of blocks is equal to the number of states of a path from  to . It also shows that the path is unique under the given ordering, and the BN structure corresponding to the path can be obtained. This means that these heuristic search algorithms based on the order graph, such as A\*, AWA\*, and BFBnB, only need to perform a partial search on the order graph. With ancestral partition constraints, the number of states that these algorithms must expand will be less than .

## 3.3 Calculate ancestral partition

The ancestral partition can prune unnecessary states of an original order graph, reduce the search space, and improve the efficiency of various algorithms based on the order graph. In this section, an algorithm is presented to obtain the constraints of the ancestral partition.

In theory, it would be more appropriate to use CI tests to discover the relationships among variables to build the ancestral partition. However, CI tests require sufficient samples. Given limited data, any statistical test will have a probability of error. Consequently, many previous studies using CI tests did not achieve the expected results. Even if exact learning algorithms were used in the subsequent search process, the errors generated by the CI tests prevented them from returning to the optimal score. Therefore, considering the drawbacks of CI tests, this study does not consider using CI tests to calculate the ancestral partition.

Before using these exact learning algorithms to search for the optimal BN structure, the possible parent sets (PPSs) must be calculated for each variable from the given data. In theory, for each node, the scores of  PPSs must be calculated, and the entire BN has a total of  scores for all PPSs. The number of PPSs can be reduced by some score pruning rules without losing the global optima. In this study, the focus is not on the calculation and pruning of scores (for detailed theory, readers can refer to, for example [29]). Instead, it is based on the search theory of the algorithm. After score calculation and pruning, the remaining PPSs and their corresponding scores can be used to search for the optimal network. Meanwhile, PPSs contain a lot of information on parent–child relationships and can be used to calculate the ancestral partition in this study.

We now describe how to calculate the ancestral partition, which is related to a new concept, the strongly connected components (SCCs). A directed graph is strongly connected if every vertex is reachable from every other vertex. A directed graph can be decomposed into SCCs. An SCC of a directed graph is a maximal set of vertices in which there is a path from any vertex in the set to any other vertex in the set. Equivalently, an SCC of a directed graph is a subgraph that is strongly connected and is maximal with this property, and no additional edges or vertices from this directed graph can be included in the subgraph without breaking its property of being strongly connected. The collection of SCCs forms a partition of the set of vertices in this directed graph.

The Asia network is used as an example to illustrate the role of SCCs. Assume that after Asia's score calculation and pruning, the PPSs are listed in Table 1. As shown in Fig. 6(a), a directed graph can be obtained by connecting all the nodes and their PPSs. This directed graph can be easily partitioned into three blocks, akin to Fig.3, as shown in Fig. 6(b). The blocks marked in Fig. 6(b) are the SCCs of the directed graph in Fig.6(a). The directed graph composed of the PPSs of a BN contains a large number of parent-child relationships of the BN, and the parent-child relationships between nodes further constitute the relationships between ancestors and descendants; that is, the directed graph implies the ancestral partition constraints. Therefore, the extracted SCCs also represent the ancestral partition constraints. An SCC can be treated simply as a partition block in an ancestral partition. If each SCC is contracted to a single vertex, the resulting graph is an acyclic component graph, as shown in Fig. 6(c). The inherent topological relationship of the acyclic component graph indicates the ancestral partition . The classic Kosaraju algorithm was used to calculate the SCCs of the directed graph. The complete algorithm for the ancestral partition calculation is presented in Algorithm 1.

Table 1: Possible parent sets (PPSs) for each node in the Asia network.

|  |  |
| --- | --- |
| variable | possible parent sets |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

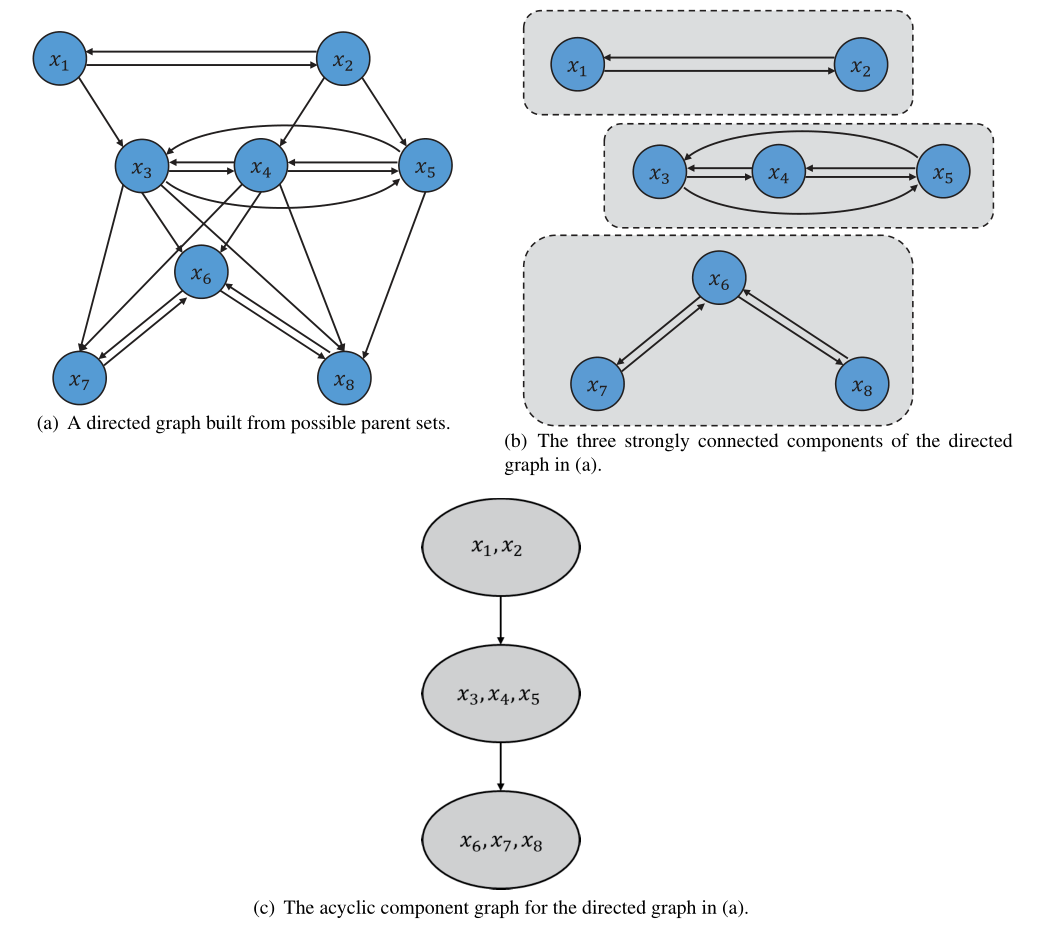


Figure 6: Example of strongly connected components (SCCs) for Asia network.



Algorithm 1 considers two cases in an ancestral partitioning process as discussed below.

**Case 1**: (The ideal case) Suppose that the directed graph composed of all PPSs can be divided into SCCs of appropriate size. Such an ancestral partition is consistent, can improve the search efficiency on the order graph, and ensures the optimal BN. However, a practical problem must be considered. Suppose that all PPSs of each node are used to build a directed graph. In this case, the directed graph may have only one SCC, or the maximum size of the SCCs may be larger than . If the partition block size is too large, algorithms based on the order graph are still difficult to search quickly. This problem leads to Case 2 as a solution.

**Case 2**: When this happens, we do not use all PPSs of each node to build the directed graph, but instead use the best  PPSs of each node to build the directed graph and then extract the SCCs. As  gradually decreases from a constant  value to 1, the edges of the directed graph also become sparse, and it is easier to extract the SCCs. However, considering the above greedy strategy, the ancestral partition may not be consistent with the BN, thus reducing the accuracy of BN structure learning. This will be further analyzed in the experiments. Cases 1 and 2 avoid invalid ancestral partition constraints as much as possible. The worst occurs when using the best PPSs () of each node to build the directed graph, and appropriate SCCs cannot be obtained. Then, each node can be treated as an SCC. However, such cases rarely occur.

In addition, is designed to limit the size of the maximum partition block. For smaller BNs, the default value  is the number of nodes in the BN minus one, and we can obtain an ancestral partition partitioned into at least two blocks. For larger BNs, although they can be extracted into multiple SCCs, the size of these SCCs may still be considerable. By controlling the maximum partition block size, appropriate SCCs can be achieved to address the problem where algorithms with a large partition block size search slowly.

# 4. Heuristic partition constraints

## 4.1 Heuristic partition

To start the discussion on heuristic partition, it is worth noting the work by Yuan on heuristic functions. To meet the search requirements of the A\*, AWA\*, and BFBnB algorithms within order graphs, Yuan proposed three heuristic functions—a simple heuristic, dynamic k-cycle conflict heuristic, and static k-cycle conflict heuristic. All heuristic functions are admissible and consistent[29][32]. Since the static k-cycle conflict heuristic is better than the simple heuristic and more efficient than the dynamic k-cycle conflict heuristic, they preferred to use the static k-cycle conflict heuristic in their subsequent research. This study also considers the static k-cycle conflict heuristic as the main research objective and abbreviates it as .

Yuan developed a better heuristic function  based on the pattern database technique, which is an approach for calculating a heuristic for a problem by solving a relaxed problem. This approach partitions the variables into non-overlapping blocks and focuses on avoiding directed cycles within each block. A subset of a partition block is considered a pattern. Yuan then calculated the heuristic using a backward breadth-first search to create static pattern databases. Once the pattern databases have been built, researchers can calculate the heuristic function directly by performing a query.

**Theorem 3.** [29]The cost of the pattern ,, is equal to the shortest distance from  to the goal state in an order graph.

The detailed method for creating the heuristic function  is as follows: For a BN with a variable set , the method is to partition it into mutually exclusive partition blocks, and  (as with , we also name  the partition block and  the partition block size of ). For each partition block , a pattern database is created. The pattern database of the partition block  corresponds to the reverse order graph of , and the backward breadth-first search can be used to create static pattern databases. The cost of the path  in this reverse order graph is equal to . Then, all subsets of each block  can be enumerated as patterns and stored as hash tables. Once these static pattern databases are created, the  cost for the state  can be calculated as

|  |  |  |
| --- | --- | --- |
|  |  | （10） |

As shown in Fig. 7, using the Asia network (eight variables) as an example, the network can be partitioned into two partition blocks  and . A reverse order graph for each block can be further created like Fig. 2 but all edges are reversed. According to Theorem 3, the shortest path from  to the goal state is the cost of the pattern  (note that, here,  is only considered the goal state rather than the entire variable set, that is,  can be either  or ). For example, the shortest path from state  to state  is the cost of the pattern . The shortest path cost for each pattern is calculated using a backward breadth-first search. All patterns and their shortest path costs are stored to create a pattern database for a reverse order graph. Assuming that the bold paths shown in Fig.7 are the shortest paths for patterns  and  respectively, we have





The costs of the other patterns can be calculated as  and  and are stored in hash tables. Finally, static pattern databases are created. The heuristic cost of the state  in the order graph must be calculated according to Eq. (10),



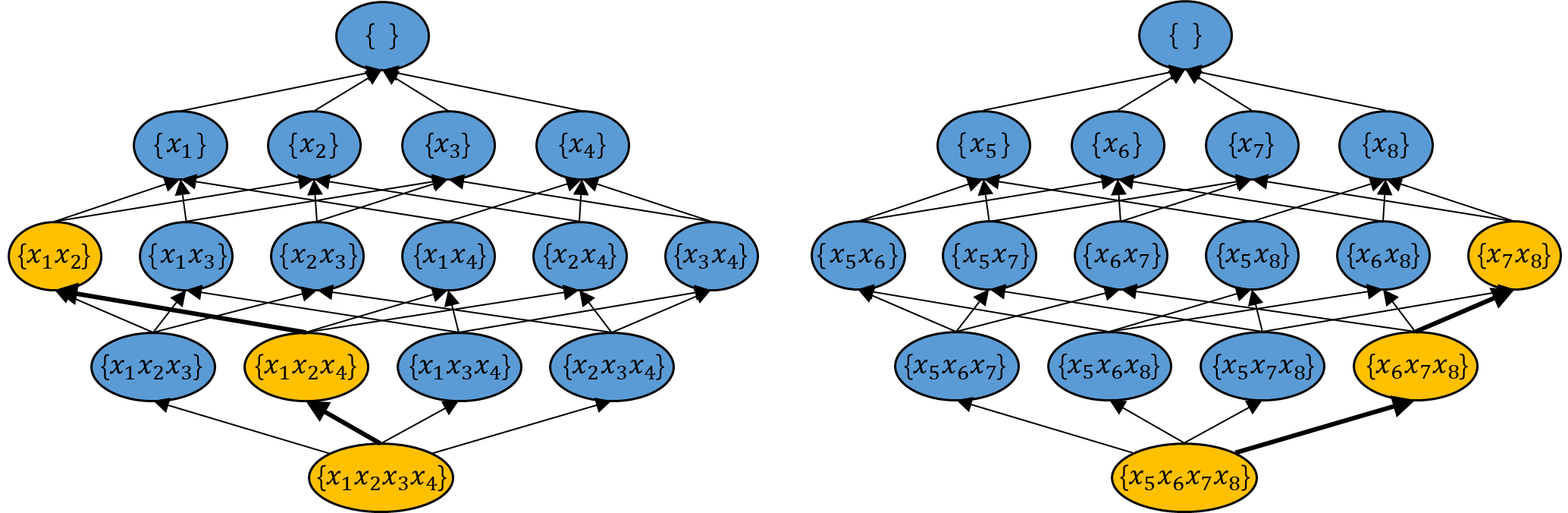


Figure 7: Two pattern databases for eight-variables networks.

For such a heuristic function , Yuan and Malone initially partitioned  into two blocks  and  (referred to as the default simple partition) for calculation. According to their further research[49], the tightness of the heuristic function depends highly on the partition being used. A good partition method should reduce the number of directed cycles between the variables and enforce the acyclicity as much as possible. Since no cycles are allowed within each partition block, the correlation between the variables in each partition block should be maximized. In addition, since cycles are allowed between partition blocks, the correlation between partition blocks should be minimized. Therefore, we continue to improve the partitioning method, which is called the heuristic partition.

## 4.2 Calculate heuristic partition

This section discusses what partition result is a good heuristic partition. According to previous research[29][32], the following characteristics of the heuristic partition can be observed:

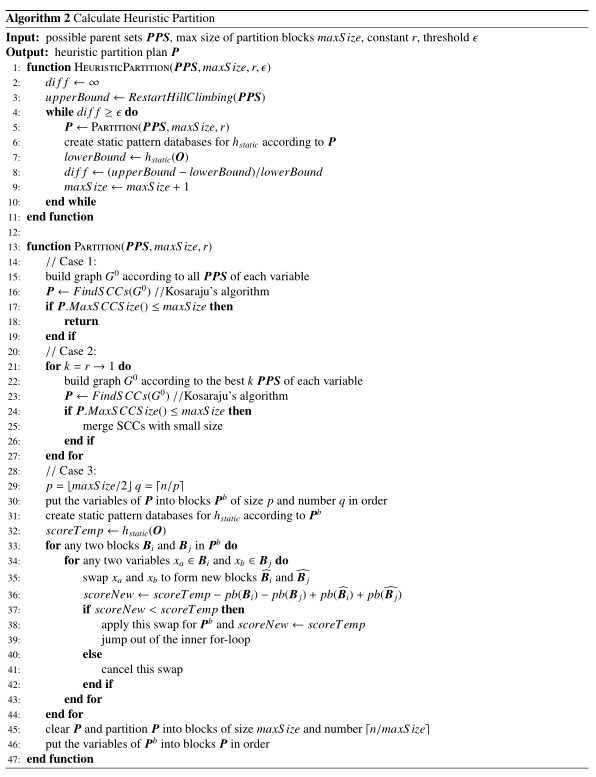
1. Balancing the number of partitions (or the partition block size) and the tightness of the heuristic function.

2. Reducing the number of directed cycles between the variables and enforcing acyclicity as much as possible.

For the first feature, if the number of partitions is 1 (the partition block size is ), the heuristic function can be created using the entire variable set ; then, for any state  in the order graph,  is the actual cost from  to . This case is related to an ideal heuristic function, equivalent to performing a backward breadth-first search in the reverse order graph for the entire variable set  to create static pattern databases. However, the time and space complexity of this case is , and the cost is too high to achieve. If the number of partitions is  (the partition block size is 1),  becomes a simple heuristic. In other words, the simple heuristic is a particular case of the static k-cycle conflict heuristic, as it simply contains costs for the individual variables, which is the worst case of . A simple heuristic search searches more states in the order graph than  and is more inefficient than . In summary, the greater the number of partitions, the smaller the partition block size, the worse the heuristic function  obtained, the higher the efficiency of creating static pattern databases, and the lower the efficiency of searching the order graph. However, the smaller the number of partitions, the larger the partition block size, the better (tighter) the heuristic function  obtained, the lower the efficiency of creating static pattern databases, and the higher the efficiency of searching the order graph. Therefore, the number of partitions (partition block size) and the tightness of the heuristic function should be relatively balanced.

The second feature can be formalized as a graph-partitioning problem that divides the graph into two or more components while minimizing the weight of the edges between different components. The definition of the SCCs mentioned in Section 3.3 satisfies these requirements. As shown in Fig. 6(b), the directed cycles are concentrated within each SCC. Suppose that static pattern databases are created for  using SCCs, and all the directed cycles are broken within each SCC, which satisfies the need to reduce the number of directed cycles between the variables and enforces acyclicity as much as possible. Therefore, this section also uses an idea similar to that in Section 3.3 for designing a heuristic partition algorithm based on SCCs.

Considering the needs of the two features discussed above, Algorithm 2 for calculating the heuristic partition, is proposed.



Algorithm 2 converges within the upper and lower bounds of the BN to obtain a heuristic partition. The algorithm uses a technology similar to the BFBnB algorithm to quickly obtain the upper bound through an approximate algorithm. This algorithm uses the hill-climbing algorithm with restarts to calculate . Since the hill-climbing algorithm itself is fast, it is worth bearing the cost to use proper restarts to obtain a better upper bound. For the lower bound, since a heuristic partition affects the tightness of the heuristic function , the value of  can be used to measure the quality of the heuristic partition and use it as . Moreover, the smaller the difference between the upper and lower bounds, the better the heuristic partition. Algorithm 2 can limit the upper and lower bounds by the threshold  to obtain a heuristic partition that meets the requirements.

When executing Algorithm 2, lines 14–27 are similar to those in Algorithm 1. **Case 1**: Case 1 is ideal.  is returned immediately for the heuristic partition that can be obtained by extracting the SCCs and meets the size requirement of the maximum partition block . **Case 2**: If Case 1 does not occur, the best  PSSs in each node are used to build a directed graph, and the SCCs are then extracted. As  gradually decreases from a constant value  to 1, the corresponding edges of the built directed graph also become sparse, and it is easier to extract the SCCs. It is necessary to merge small SCCs because small SCCs imply a greater number of partitions that reduce the tightness of the heuristic function. Conversely, with merging partitions of relatively small SCCs, the new partition blocks formed will not be too large, and it will take less time to create static pattern databases. For each SCC, the current size of the SCC can be checked to determine if it is less than  minus the previous SCCs size. If so, we merged these SCCs. **Case 3**: Considering that the heuristic partition obtained by the above strategies may not meet the threshold requirements , a new partition calculation method is added here. In this method, the score  corresponding to the current heuristic partition is first retained. Then, the current partition  is further partitioned into smaller partition blocks of size  and becomes . Subsequently, the elements of the partition blocks in  are swapped, and the updated score  is calculated corresponding to the new heuristic partition. If the updated score is better than , the swap is retained; otherwise, the swap is withdrawn. A better heuristic partition can be obtained by applying this greedy swap strategy and integrating it into the original size.

The main body of Algorithm 2 uses  to control the entire loop.  starts from an initial user-specified value and increases by 1. Different  values return different heuristic partitions in the  function, and the threshold  judges these partitions. Therefore, the final heuristic partition formed considers both the partition block size and the tightness of the heuristic function.

Yuan proved that  is always admissible and consistent[29]. Our heuristic partition constraints offer only the different partition schemes for . As such, the heuristic partition constraints do not affect the accuracy of these heuristic algorithms based on the order graph but only their efficiency.

# 5. Experiments

To evaluate and demonstrate the algorithms proposed in this study for exact structure learning, a considerable number of experiments were conducted and are presented in this section.

To test and evaluate the performance of the algorithms proposed for ancestral partition and heuristic partition the following algorithms were used for comparison:

1. Order graph-based algorithms, including A\*, AWA\*, and BFBnB[[4]](#footnote-4), with\without partition constraints being added.
2. GOBNILP[[5]](#footnote-5) algorithm (the latest stable version 1.6.3), considered the current state-of-the-art algorithm of exact learning of the BN structure.
3. MMHC algorithm, the most well-known hybrid algorithm.
4. NOTEARS[[6]](#footnote-6) algorithm, considered the first to recast the structure learning problem as a continuous optimization problem.
5. DAG-GNN[[7]](#footnote-7) algorithm, which uses a deep generative model and applies a structural constraint variant to learn a DAG.

All the algorithms in 1 and 3 were coded in C++. The performances of the algorithms in 1 and 2 were analyzed with various parameter settings.

A group of benchmark BNs and datasets from the UCI machine learning repository[[8]](#footnote-8) were used in the experiments, and they are publicly available. Except for the boerlage92 BN[50], the other benchmark BNs are publicly available in the BN repository[[9]](#footnote-9). Samples were randomly selected from these benchmark BNs of different scales with different sample sizes . In addition, synthetic data were generated for larger networks with over 100 nodes.

All the experiments were executed on Windows 10 with a 4-core Intel (TM) i7-7700 3.6 GHz processor and 16 GB RAM.

For brevity, in the following experiments, we abbreviate ancestral partition as "AP" and abbreviate heuristic partition as "HP." "-AP" indicates an algorithm with only ancestral partition constraints from Algorithm 1. "-HP" indicates an algorithm with only heuristic partition constraints from Algorithm 2. "AP-HP" indicates an algorithm with both ancestral partition and heuristic partition constraints.

Notably, Section 3.2 theoretically proved that a consistent ancestral partition does not affect the optimal score of exact learning algorithms based on the order graph. However, Algorithm 1 for calculating an ancestral partition in Section 3.3 uses a greedy strategy in Case 2. Therefore, the proposed algorithm may return an inaccurate ancestral partition that affects the accuracy of these exact learning algorithms. To reflect the influence of accuracy, a score error ratio is applied to the optimal score as follows:

|  |  |  |
| --- | --- | --- |
|  |  | （11） |

where  is the optimal MDL score of the DAG from the original exact learning algorithm, and  is the MDL score of the DAG from these algorithms with partition constraints. The better the score obtained by an algorithm with partition constraints, the closer the score error ratio to 0.

In the following experiments, some symbols are used to indicate different situations: "O" for an original algorithm and "OT" for an experiment running up to 8 hours but not converging (out of time). For a larger number of expanded states (over 10 million), "M" represents millions. All results were rounded to three decimal places.

## 5.1 Experiments of ancestral partition and heuristic partition for algorithms based on order graph

To quantitatively analyze the efficacy and accuracy of the proposed partition constraints for order graph-based exact learning algorithms, including A\*, AWA\*, and BFBnB, the performances of these algorithms were observed and compared with those of ancestral and/or heuristic partitions on the benchmark BNs and UCI datasets.

The comparison was based on the following criteria:

**Time**: The running time of an algorithm (in seconds or hours).

**The number of expanded states (Exp)**: The number of expanded states in an order graph. The fewer the expanded states, the lower the memory cost. This criterion is only valid for order graph-based exact learning algorithms (A\*, AWA\*, and BFBnB).

**Score error ratio (Error)**: This criterion is calculated using Eq. (11). In general, the GOBNILP algorithm was employed to obtain the optimal MDL score . If GOBNILP could not obtain an optimal score, a score returned by A\*[[10]](#footnote-10) was used. Note that a lower Error value indicates higher accuracy.

For the ancestral partition calculation algorithm, the maximum size of the partition blocks was set to  for BNs with ,  for BNs with , respectively, and the constant to . For the heuristic partition calculation algorithm, the maximum size of the partition blocks was set to , constant to , and threshold to .

The results of running time, number of expanded states, and score error ratios for A\*, AWA\*, and BFBnB with ancestral partition and/or heuristic partition on benchmark BNs and UCI datasets are presented in Tables 2–7.

Table 2: Running time, number of expanded states, and score error ratio for A\* with ancestral partition and/or heuristic partition on benchmark BNs.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | O | | AP | | | HP | | | AP-HP | | |
| Time(s) | Exp | Time(s) | Exp | Error | Time(s) | Exp | Error | Time(s) | Exp | Error |
| sachs | 11 | 500 | 0.001 | 164 | 0.001 | 53 | 0% | 0.001 | 11 | 0% | 0.001 | 11 | 0% |
| sachs | 11 | 1000 | 0.002 | 207 | 0.001 | 56 | 0% | 0.001 | 11 | 0% | 0.001 | 11 | 0% |
| sachs | 11 | 5000 | 0.003 | 503 | 0.002 | 91 | 0% | 0.002 | 11 | 0% | 0.002 | 15 | 0% |
| sachs | 11 | 10000 | 0.003 | 634 | 0.003 | 110 | 0% | 0.003 | 14 | 0% | 0.003 | 11 | 0% |
| sachs | 11 | 15000 | 0.004 | 765 | 0.003 | 131 | 0% | 0.003 | 36 | 0% | 0.003 | 14 | 0% |
| child | 20 | 500 | 0.125 | 34679 | 0.040 | 13764 | 0% | 0.023 | 292 | 0% | 0.022 | 100 | 0% |
| child | 20 | 1000 | 0.209 | 54271 | 0.073 | 22160 | 0% | 0.025 | 75 | 0% | 0.024 | 81 | 0% |
| child | 20 | 5000 | 1.132 | 257793 | 0.076 | 100564 | 0% | 0.066 | 5123 | 0% | 0.062 | 3031 | 0% |
| child | 20 | 10000 | 1.746 | 422718 | 0.707 | 174422 | 0% | 0.134 | 12190 | 0% | 0.104 | 7623 | 0% |
| child | 20 | 15000 | 2.130 | 515066 | 0.914 | 221324 | 0% | 0.227 | 21229 | 0% | 0.173 | 12624 | 0% |
| boerlage92 | 23 | 500 | 9.231 | 2002228 | 4.418 | 1004319 | 0% | 0.029 | 1604 | 0% | 0.025 | 1332 | 0% |
| boerlage92 | 23 | 1000 | 9.821 | 2027860 | 0.443 | 143271 | 0% | 0.173 | 29536 | 0% | 0.010 | 137 | 0% |
| boerlage92 | 23 | 5000 | 15.049 | 3170608 | 8.820 | 2229531 | 0% | 22.769 | 4355273 | 0% | 9.928 | 2130129 | 0% |
| boerlage92 | 23 | 10000 | 17.632 | 3600952 | 9.142 | 2230655 | 0% | 26.324 | 5042508 | 0% | 11.882 | 2574403 | 0% |
| boerlage92 | 23 | 15000 | 18.026 | 3667071 | 9.367 | 2236236 | 0% | 29.501 | 5582066 | 0% | 13.014 | 2831004 | 0% |
| insurance | 27 | 500 | 187.219 | 23.824M | 61.519 | 8875295 | 0% | 0.640 | 73228 | 0% | 0.136 | 18317 | 0% |
| insurance | 27 | 1000 | 241.479 | 26.759M | 146.363 | 17.438M | 0% | 2.928 | 292463 | 0% | 1.404 | 146644 | 0% |
| insurance | 27 | 5000 | 252.730 | 24.618M | 143.290 | 15.790M | 0% | 3.091 | 285418 | 0% | 1.432 | 143307 | 0% |
| insurance | 27 | 10000 | 261.516 | 24.781M | 65.748 | 7986905 | 0% | 10.033 | 30146 | 0% | 1.886 | 8497 | 0% |
| insurance | 27 | 15000 | 262.485 | 24.799M | 147.272 | 15.778M | 0% | 1.974 | 27673 | 0% | 1.845 | 13838 | 0% |
| water | 32 | 500 | OT |  | 1.709 | 457528 | 0.048% | 11.974 | 994743 | 0% | 0.019 | 542 | 0.048% |
| water | 32 | 1000 | OT |  | 2.319 | 632824 | 0.008% | 31.442 | 2076364 | 0% | 0.026 | 930 | 0.008% |
| water | 32 | 5000 | OT |  | 2.812 | 765076 | 0.003% | 6.968 | 492616 | 0% | 0.025 | 343 | 0.003% |
| water | 32 | 10000 | OT |  | 0.184 | 90357 | 0.001% | 7.428 | 479481 | 0% | 0.033 | 135 | 0.001% |
| water | 32 | 15000 | OT |  | 2.802 | 742632 | 0.004% | 28.802 | 1775929 | 0% | 0.033 | 1175 | 0.004% |
| alarm | 37 | 500 | OT |  | 0.107 | 197 | 0.401% | OT |  |  | 0.164 | 37 | 0.401% |
| alarm | 37 | 1000 | OT |  | 0.167 | 61582 | 0.062% | OT |  |  | 0.091 | 4908 | 0.062% |
| alarm | 37 | 5000 | OT |  | 0.226 | 73802 | 0.021% | OT |  |  | 0.296 | 130 | 0.021% |
| alarm | 37 | 10000 | OT |  | 0.256 | 76196 | 0.010% | OT |  |  | 0.155 | 8349 | 0.010% |
| alarm | 37 | 15000 | OT |  | 0.307 | 77581 | 0.005% | OT |  |  | 0.161 | 8341 | 0.005% |
| hailfinder | 56 | 500 | OT |  | 0.066 | 1670 | 0.013% | OT |  |  | 0.098 | 76 | 0.013% |
| hailfinder | 56 | 1000 | OT |  | 0.064 | 1931 | 0.023% | OT |  |  | 0.084 | 66 | 0.023% |
| hailfinder | 56 | 5000 | OT |  | 1.564 | 356025 | 0.025% | OT |  |  | 0.231 | 68 | 0.025% |
| hailfinder | 56 | 10000 | OT |  | 0.291 | 491 | 0.067% | OT |  |  | 0.356 | 62 | 0.067% |
| hailfinder | 56 | 15000 | OT |  | 1.662 | 362282 | 0.043% | OT |  |  | 0.469 | 10378 | 0.043% |

Table 3: Running time, number of expanded states, and score error ratio for AWA\* with ancestral partition and/or heuristic partition on benchmark BNs.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | O | | AP | | | HP | | | AP-HP | | |
| Time(s) | Exp | Time(s) | Exp | Error | Time(s) | Exp | Error | Time(s) | Exp | Error |
| sachs | 11 | 500 | 0.002 | 211 | 0.001 | 63 | 0% | 0.001 | 11 | 0% | 0.001 | 11 | 0% |
| sachs | 11 | 1000 | 0.002 | 271 | 0.001 | 69 | 0% | 0.001 | 11 | 0% | 0.001 | 11 | 0% |
| sachs | 11 | 5000 | 0.003 | 629 | 0.002 | 102 | 0% | 0.002 | 11 | 0% | 0.002 | 16 | 0% |
| sachs | 11 | 10000 | 0.003 | 783 | 0.003 | 144 | 0% | 0.003 | 15 | 0% | 0.003 | 11 | 0% |
| sachs | 11 | 15000 | 0.004 | 914 | 0.003 | 160 | 0% | 0.003 | 37 | 0% | 0.003 | 14 | 0% |
| child | 20 | 500 | 0.107 | 45312 | 0.045 | 16371 | 0% | 0.026 | 371 | 0% | 0.027 | 123 | 0% |
| child | 20 | 1000 | 0.198 | 69590 | 0.074 | 26905 | 0% | 0.029 | 118 | 0% | 0.029 | 108 | 0% |
| child | 20 | 5000 | 1.377 | 323846 | 0.405 | 119191 | 0% | 0.070 | 7438 | 0% | 0.068 | 5133 | 0% |
| child | 20 | 10000 | 2.404 | 534444 | 0.792 | 208862 | 0% | 0.149 | 19720 | 0% | 0.117 | 12522 | 0% |
| child | 20 | 15000 | 2.90 | 595660 | 1.050 | 254369 | 0% | 0.249 | 32735 | 0% | 0.212 | 19590 | 0% |
| boerlage92 | 23 | 500 | 12.487 | 3641617 | 5.687 | 1695447 | 0% | 0.027 | 2466 | 0% | 0.026 | 1881 | 0% |
| boerlage92 | 23 | 1000 | 14.904 | 3958712 | 0.450 | 214525 | 0% | 0.207 | 58129 | 0% | 0.010 | 169 | 0% |
| boerlage92 | 23 | 5000 | 33.738 | 7824034 | 24.236 | 6323678 | 0% | 45.366 | 8138119 | 0% | 19.229 | 3950159 | 0% |
| boerlage92 | 23 | 10000 | 43.527 | 9344601 | 23.560 | 5597403 | 0% | 55.968 | 9425791 | 0% | 22.662 | 4554849 | 0% |
| boerlage92 | 23 | 15000 | 52.107 | 11.364M | 26.969 | 6952358 | 0% | 50.801 | 11.392M | 0% | 26.449 | 5287172 | 0% |
| insurance | 27 | 500 | 331.862 | 35.222M | 109.414 | 11.732M | 0% | 0.633 | 129828 | 0% | 0.729 | 26900 | 0% |
| insurance | 27 | 1000 | 445.477 | 36.488M | 296.259 | 23.662M | 0% | 4.885 | 517640 | 0% | 2.316 | 249705 | 0% |
| insurance | 27 | 5000 | 426.415 | 32.291M | 269.710 | 21.367M | 0% | 3.653 | 378471 | 0% | 1.843 | 184711 | 0% |
| insurance | 27 | 10000 | 426.873 | 32.538M | 101.347 | 9989889 | 0% | 10.309 | 47955 | 0% | 1.828 | 11116 | 0% |
| insurance | 27 | 15000 | 431.401 | 31.664M | 264.387 | 20.265M | 0% | 1.988 | 40444 | 0% | 1.866 | 19587 | 0% |
| water | 32 | 500 | OT |  | 2.156 | 619534 | 0.048% | 16.152 | 2386387 | 0% | 0.020 | 779 | 0.048% |
| water | 32 | 1000 | OT |  | 3.498 | 881337 | 0.008% | 41.585 | 3669217 | 0% | 0.028 | 1140 | 0.008% |
| water | 32 | 5000 | OT |  | 3.943 | 927943 | 0.003% | 8.628 | 905978 | 0% | 0.025 | 424 | 0.003% |
| water | 32 | 10000 | OT |  | 0.218 | 110199 | 0.001% | 9.856 | 851281 | 0% | 0.034 | 150 | 0.001% |
| water | 32 | 15000 | OT |  | 4.357 | 968292 | 0.004% | 34.204 | 2860343 | 0% | 0.035 | 1574 | 0.004% |
| alarm | 37 | 500 | OT |  | 0.109 | 216 | 0.401% | OT |  |  | 0.108 | 37 | 0.401% |
| alarm | 37 | 1000 | OT |  | 0.200 | 104005 | 0.062% | OT |  |  | 0.097 | 7694 | 0.062% |
| alarm | 37 | 5000 | OT |  | 0.270 | 117787 | 0.021% | OT |  |  | 0.216 | 124 | 0.021% |
| alarm | 37 | 10000 | OT |  | 0.325 | 157903 | 0.010% | OT |  |  | 0.163 | 10939 | 0.010% |
| alarm | 37 | 15000 | OT |  | 0.298 | 133828 | 0.005% | OT |  |  | 0.169 | 11348 | 0.005% |
| hailfinder | 56 | 500 | OT |  | 0.067 | 2168 | 0.013% | OT |  |  | 0.070 | 73 | 0.013% |
| hailfinder | 56 | 1000 | OT |  | 0.064 | 2445 | 0.023% | OT |  |  | 0.077 | 75 | 0.023% |
| hailfinder | 56 | 5000 | OT |  | 2.901 | 667896 | 0.025% | OT |  |  | 1.816 | 73 | 0.025% |
| hailfinder | 56 | 10000 | OT |  | 0.285 | 648 | 0.067% | OT |  |  | 0.290 | 64 | 0.067% |
| hailfinder | 56 | 15000 | OT |  | 2.890 | 653044 | 0.043% | OT |  |  | 1.582 | 12991 | 0.043% |

Table 4: Running time, number of expanded states, and score error ratio for BFBnB with ancestral partition and/or heuristic partition on benchmark BNs.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | O | | AP | | | HP | | | AP-HP | | |
| Time(s) | Exp | Time(s) | Exp | Error | Time(s) | Exp | Error | Time(s) | Exp | Error |
| sachs | 11 | 500 | 0.004 | 389 | 0.003 | 80 | 0% | 0.004 | 335 | 0% | 0.004 | 53 | 0% |
| sachs | 11 | 1000 | 0.005 | 567 | 0.004 | 99 | 0% | 0.004 | 519 | 0% | 0.004 | 71 | 0% |
| sachs | 11 | 5000 | 0.005 | 751 | 0.005 | 111 | 0% | 0.005 | 487 | 0% | 0.005 | 67 | 0% |
| sachs | 11 | 10000 | 0.006 | 887 | 0.005 | 134 | 0% | 0.005 | 487 | 0% | 0.005 | 67 | 0% |
| sachs | 11 | 15000 | 0.006 | 848 | 0.005 | 152 | 0% | 0.005 | 139 | 0% | 0.005 | 67 | 0% |
| child | 20 | 500 | 0.113 | 36914 | 0.029 | 15394 | 0% | 0.035 | 7746 | 0% | 0.034 | 3882 | 0% |
| child | 20 | 1000 | 0.106 | 55805 | 0.045 | 23253 | 0% | 0.038 | 7784 | 0% | 0.038 | 3900 | 0% |
| child | 20 | 5000 | 0.394 | 259156 | 0.168 | 101145 | 0% | 0.102 | 12371 | 0% | 0.105 | 6743 | 0% |
| child | 20 | 10000 | 0.801 | 423868 | 0.347 | 174816 | 0% | 0.120 | 19582 | 0% | 0.137 | 11175 | 0% |
| child | 20 | 15000 | 0.917 | 516250 | 0.499 | 221228 | 0% | 0.221 | 28710 | 0% | 0.207 | 12605 | 0% |
| boerlage92 | 23 | 500 | 3.414 | 2012963 | 1.394 | 1022502 | 0% | 0.033 | 5857 | 0% | 0.034 | 5782 | 0% |
| boerlage92 | 23 | 1000 | 3.460 | 2041343 | 0.185 | 144115 | 0% | 0.078 | 34539 | 0% | 0.015 | 287 | 0% |
| boerlage92 | 23 | 5000 | 5.732 | 3174387 | 3.336 | 2230335 | 0% | 8.593 | 4359792 | 0% | 3.142 | 2131497 | 0% |
| boerlage92 | 23 | 10000 | 6.679 | 3603342 | 3.466 | 2231478 | 0% | 9.832 | 5050544 | 0% | 4.485 | 2578268 | 0% |
| boerlage92 | 23 | 15000 | 6.830 | 3668565 | 3.564 | 2237462 | 0% | 10.628 | 5585074 | 0% | 3.434 | 2835181 | 0% |
| insurance | 27 | 500 | 93.145 | 23.922M | 28.597 | 8928857 | 0% | 0.213 | 97491 | 0% | 0.067 | 24817 | 0% |
| insurance | 27 | 1000 | 110.116 | 26.786M | 73.064 | 17.472M | 0% | 0.761 | 312161 | 0% | 0.392 | 157598 | 0% |
| insurance | 27 | 5000 | 104.588 | 24.641M | 67.418 | 15.798M | 0% | 0.792 | 296061 | 0% | 0.399 | 151183 | 0% |
| insurance | 27 | 10000 | 106.364 | 24.790M | 28.314 | 7991544 | 0% | 9.367 | 39833 | 0% | 1.809 | 12685 | 0% |
| insurance | 27 | 15000 | 107.699 | 24.802M | 68.937 | 15.781M | 0% | 1.831 | 30521 | 0% | 1.848 | 15263 | 0% |
| water | 32 | 500 | OT |  | 0.566 | 459629 | 0.048% | 17.034 | 5772642 | 0% | 0.037 | 3042 | 0.048% |
| water | 32 | 1000 | OT |  | 0.828 | 634958 | 0.008% | 8.669 | 2346917 | 0% | 0.037 | 3082 | 0.008% |
| water | 32 | 5000 | OT |  | 1.020 | 765489 | 0.003% | 1.390 | 514388 | 0% | 0.033 | 593 | 0.003% |
| water | 32 | 10000 | OT |  | 0.192 | 90459 | 0.001% | 1.634 | 532970 | 0% | 0.034 | 251 | 0.001% |
| water | 32 | 15000 | OT |  | 1.009 | 742810 | 0.004% | 7.143 | 1795402 | 0% | 0.052 | 1536 | 0.004% |
| alarm | 37 | 500 | OT |  | 0.127 | 205 | 0.401% | OT |  |  | 0.120 | 60 | 0.401% |
| alarm | 37 | 1000 | OT |  | 0.117 | 62268 | 0.062% | OT |  |  | 0.085 | 6048 | 0.062% |
| alarm | 37 | 5000 | OT |  | 0.166 | 74354 | 0.021% | OT |  |  | 0.283 | 1500 | 0.021% |
| alarm | 37 | 10000 | OT |  | 0.184 | 76756 | 0.010% | OT |  |  | 0.153 | 9588 | 0.010% |
| alarm | 37 | 15000 | OT |  | 0.194 | 78120 | 0.005% | OT |  |  | 0.144 | 9995 | 0.005% |
| hailfinder | 56 | 500 | OT |  | 0.109 | 2316 | 0.013% | OT |  |  | 0.091 | 1115 | 0.013% |
| hailfinder | 56 | 1000 | OT |  | 0.070 | 2411 | 0.023% | OT |  |  | 0.088 | 805 | 0.023% |
| hailfinder | 56 | 5000 | OT |  | 0.638 | 356943 | 0.025% | OT |  |  | 0.215 | 2235 | 0.025% |
| hailfinder | 56 | 10000 | OT |  | 0.344 | 583 | 0.067% | OT |  |  | 0.335 | 185 | 0.067% |
| hailfinder | 56 | 15000 | OT |  | 0.783 | 363272 | 0.043% | OT |  |  | 0.404 | 11274 | 0.043% |

Table 5: Running time, number of expanded states, and score error ratio for A\* with ancestral partition and/or heuristic partition on UCI datasets.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | O | | AP | | | HP | | | AP-HP | | |
| Time(s) | Exp | Time(s) | Exp | Error | Time(s) | Exp | Error | Time(s) | Exp | Error |
| zoo | 17 | 101 | 0.053 | 20468 | 0.022 | 5916 | 0% | 0.025 | 68 | 0% | 0.031 | 28 | 0% |
| voting | 17 | 435 | 0.010 | 2521 | 0.010 | 789 | 0% | 0.024 | 59 | 0% | 0.027 | 44 | 0% |
| statlog | 19 | 752 | 0.918 | 220235 | 0.473 | 66275 | 0% | 0.100 | 4162 | 0% | 0.134 | 1536 | 0% |
| hepatitis | 20 | 126 | 0.032 | 8514 | 0.021 | 4807 | 0% | 0.022 | 94 | 0% | 0.025 | 57 | 0% |
| segment | 20 | 2310 | 1.889 | 428082 | 0.441 | 107899 | 0% | 0.054 | 404 | 0% | 0.099 | 108 | 0% |
| imports | 22 | 205 | 11.520 | 2052429 | 5.237 | 927463 | 0% | 0.563 | 71244 | 0% | 0.291 | 41964 | 0% |
| meta | 22 | 528 | 5.876 | 441823 | 3.318 | 225950 | 0% | 7.667 | 24 | 0% | 7.880 | 22 | 0% |
| heart | 23 | 212 | 23.241 | 4456333 | 10.877 | 2238602 | 0% | 0.531 | 95464 | 0% | 0.240 | 52690 | 0% |
| horse | 23 | 300 | 15.775 | 3126900 | 3.727 | 781730 | 0% | 1.877 | 278894 | 0% | 0.391 | 69733 | 0% |
| mushroom | 23 | 8124 | 0.614 | 49592 | 0.410 | 33164 | 0% | 10.912 | 102 | 0% | 10.644 | 100 | 0% |
| autos | 26 | 159 | 46.271 | 4762545 | 23.824 | 2546893 | 0% | 9.790 | 939725 | 0% | 5.251 | 4697864 | 0% |
| steel | 28 | 1941 | OT |  | 147.794 | 17.705M | 0% | 0.999 | 82014 | 0% | 0.365 | 27296 | 0% |
| flag | 29 | 194 | OT |  | 14.822 | 1196267 | 0.004% | 78.112 | 6236857 | 0% | 21.965 | 2143722 | 0.004% |
| soybean | 36 | 266 | OT |  | 0.398 | 15302 | 0.487% | OT |  |  | 0.393 | 4311 | 0.487% |
| bands | 39 | 277 | OT |  | 0.120 | 66 | 0.194% | OT |  |  | 0.133 | 39 | 0.194% |
| spectf | 45 | 267 | OT |  | 0.069 | 47 | 0.098% | OT |  |  | 0.087 | 45 | 0.098% |
| sponge | 45 | 76 | OT |  | 0.310 | 54 | 1.304% | OT |  |  | 0.300 | 46 | 1.304% |
| lung cancer | 57 | 32 | OT |  | 0.472 | 19223 | 2.159% | OT |  |  | 0.441 | 703 | 2.159% |
| splice | 61 | 3190 | OT |  | 0.167 | 65 | 0.123% | OT |  |  | 0.158 | 61 | 0.123% |

Table 6: Running time, number of expanded states, and score error ratio for AWA\* with ancestral partition and/or heuristic partition on UCI datasets.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | O | | AP | | | HP | | | AP-HP | | |
| Time(s) | Exp | Time(s) | Exp | Error | Time(s) | Exp | Error | Time(s) | Exp | Error |
| zoo | 17 | 101 | 0.059 | 25131 | 0.048 | 6899 | 0% | 0.028 | 96 | 0% | 0.034 | 28 | 0% |
| voting | 17 | 435 | 0.011 | 3771 | 0.010 | 1152 | 0% | 0.022 | 57 | 0% | 0.030 | 46 | 0% |
| statlog | 19 | 752 | 1.113 | 338997 | 0.278 | 101894 | 0% | 0.108 | 5954 | 0% | 0.146 | 2137 | 0% |
| hepatitis | 20 | 126 | 0.032 | 10641 | 0.021 | 6134 | 0% | 0.024 | 137 | 0% | 0.027 | 117 | 0% |
| segment | 20 | 2310 | 3.264 | 908547 | 0.663 | 240903 | 0% | 0.057 | 532 | 0% | 0.103 | 129 | 0% |
| imports | 22 | 205 | 17.668 | 3075638 | 6.731 | 1264646 | 0% | 0.797 | 92703 | 0% | 0.365 | 52723 | 0% |
| meta | 22 | 528 | 7.201 | 517954 | 3.668 | 252176 | 0% | 7.665 | 22 | 0% | 7.662 | 22 | 0% |
| heart | 23 | 212 | 38.600 | 7787034 | 15.584 | 3419396 | 0% | 0.635 | 127183 | 0% | 0.306 | 67492 | 0% |
| horse | 23 | 300 | 29.253 | 5386046 | 6.063 | 1327575 | 0% | 2.944 | 457832 | 0% | 0.495 | 107789 | 0% |
| mushroom | 23 | 8124 | 0.665 | 64057 | 0.557 | 46724 | 0% | 11.668 | 101 | 0% | 11.767 | 100 | 0% |
| autos | 26 | 159 | 70.130 | 7528551 | 70.130 | 4059585 | 0% | 13.251 | 1323745 | 0% | 7.024 | 677144 | 0% |
| steel | 28 | 1941 | OT |  | 286.447 | 41.099M | 0% | 4.959 | 510918 | 0% | 1.157 | 152173 | 0% |
| flag | 29 | 194 | OT |  | 18.649 | 1320734 | 0.004% | 96.354 | 8321170 | 0% | 28.707 | 2980926 | 0.004% |
| soybean | 36 | 266 | OT |  | 0.430 | 22255 | 0.487% | OT |  |  | 0.398 | 5802 | 0.487% |
| bands | 39 | 277 | OT |  | 0.149 | 69 | 0.194% | OT |  |  | 0.118 | 39 | 0.194% |
| spectf | 45 | 267 | OT |  | 0.070 | 47 | 0.098% | OT |  |  | 0.069 | 45 | 0.098% |
| sponge | 45 | 76 | OT |  | 0.337 | 66 | 1.304% | OT |  |  | 0.342 | 48 | 1.304% |
| lung cancer | 57 | 32 | OT |  | 0.490 | 22922 | 2.159% | OT |  |  | 0.465 | 854 | 2.159% |
| splice | 61 | 3190 | OT |  | 0.171 | 69 | 0.123% | OT |  |  | 0.160 | 61 | 0.123% |

Table 7: Running time, number of expanded states, and score error ratio for BFBnB with ancestral partition and/or heuristic partition on UCI datasets.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | O | | AP | | | HP | | | AP-HP | | |
| Time(s) | Exp | Time(s) | Exp | Error | Time(s) | Exp | Error | Time(s) | Exp | Error |
| zoo | 17 | 101 | 0.056 | 22065 | 0.027 | 6241 | 0% | 0.029 | 444 | 0% | 0.034 | 133 | 0% |
| voting | 17 | 435 | 0.012 | 4982 | 0.015 | 1723 | 0% | 0.027 | 2459 | 0% | 0.033 | 962 | 0% |
| statlog | 19 | 752 | 0.356 | 249602 | 0.359 | 75415 | 0% | 0.155 | 15262 | 0% | 0.164 | 5524 | 0% |
| hepatitis | 20 | 126 | 0.036 | 22930 | 0.034 | 20490 | 0% | 0.033 | 7681 | 0% | 0.037 | 5705 | 0% |
| segment | 20 | 2310 | 0.877 | 537611 | 0.276 | 136505 | 0% | 0.093 | 26583 | 0% | 0.116 | 6682 | 0% |
| imports | 22 | 205 | 4.166 | 2475145 | 1.829 | 1127967 | 0% | 0.324 | 75604 | 0% | 0.215 | 44605 | 0% |
| meta | 22 | 528 | 4.026 | 443153 | 3.205 | 228331 | 0% | 7.272 | 310 | 0% | 7.644 | 167 | 0% |
| heart | 23 | 212 | 7.846 | 4579686 | 3.414 | 2255507 | 0% | 0.242 | 113342 | 0% | 0.115 | 59797 | 0% |
| horse | 23 | 300 | 6.250 | 3670783 | 1.298 | 907128 | 0% | 0.691 | 322343 | 0% | 0.199 | 75274 | 0% |
| mushroom | 23 | 8124 | 0.395 | 58203 | 0.366 | 37431 | 0% | 11.764 | 8735 | 0% | 11.730 | 4368 | 0% |
| autos | 26 | 159 | 14.652 | 5138539 | 6.969 | 2743292 | 0% | 3.867 | 1070505 | 0% | 2.184 | 535253 | 0% |
| steel | 28 | 1941 | OT |  | 62.711 | 17.751M | 0% | 0.346 | 88357 | 0% | 0.179 | 31437 | 0% |
| flag | 29 | 194 | OT |  | 4.024 | 1497750 | 0.004% | 28.055 | 7238302 | 0% | 7.369 | 2655608 | 0.004% |
| soybean | 36 | 266 | OT |  | 0.416 | 47425 | 0.487% | OT |  |  | 0.421 | 16621 | 0.487% |
| bands | 39 | 277 | OT |  | 0.326 | 96 | 0.194% | OT |  |  | 0.138 | 78 | 0.194% |
| spectf | 45 | 267 | OT |  | 0.097 | 61 | 0.098% | OT |  |  | 0.095 | 55 | 0.098% |
| sponge | 45 | 76 | OT |  | 0.326 | 74 | 1.304% | OT |  |  | 0.321 | 69 | 1.304% |
| lung cancer | 57 | 32 | OT |  | 0.501 | 31912 | 2.159% | OT |  |  | 0.446 | 4439 | 2.159% |
| splice | 61 | 3190 | OT |  | 0.200 | 76 | 0.123% | OT |  |  | 0.190 | 72 | 0.123% |

As shown in Tables 2–7, the original A\*, AWA\*, BFBnB and these algorithms with ancestral partition and/or heuristic partition were used to learn structures on benchmark BNs and UCI datasets, and their trends of running time, number of expanded states and score error ratios were similar. Since the original A\*, AWA\*, and BFBnB are all exact learning algorithms, their score error ratios are always 0%. Therefore, the score error ratios for the original algorithms are not included in these tables. In addition, under the same constraints, the three algorithms achieved the same score error ratio. In general, the running time and the number of expanded states of the algorithms with ancestral partition and/or heuristic partition have significantly reduced compared with the original algorithms. In addition, with these partition constraints, the algorithms can search larger BNs.

In general, these algorithms with HP have demonstrated lower running times and fewer expanded states than those with AP on BNs with , and the algorithms with AP-HP have resulted in the lowest running times and fewest expanded states among almost all datasets. On BNs with , the algorithms with AP had lower running times and fewer expanded states than those with HP, and AP constraints performed better than HP constraints at improving the scalability of the algorithms. There are two reasons for this: (1) The main factor limiting these search algorithms is the scale of the order graph (it grows exponentially) rather than the tightness of the heuristic function. According to the analysis in Sections 3 and 4, AP constraints can significantly prune the order graph and reduce the number of states, which completely changes the structure of the order graph. However, HP constraints cannot change the structure of the order graph; rather, they can only improve the tightness of the heuristic function to avoid searching for more states in the original order graph. In short, compared with HP constraints, AP constraints can directly affect the order graph more significantly, and the AP constraints are the main factors that affect the efficiency of these algorithms. Suppose AP constraints  () are always available for any network. According to Theorem 2, the ratio of the number of remaining states to the number of original states after applying AP constraints is  ().  (), the ratio decreases as the network size increases; that is, the number of remaining states that must be expanded decreases when the network size increases. For example, if in an 8-variables network and AP constraints  () are applied, then the number of expanded states can be reduced from 255 to 30, which is 11.375% of the original value. If, in a 16-variable network with similar AP constraints  (), then the number of expanded states can be reduced from 65535 to 510, which is 0.778% of the original value. Evidently, as the network size increases, more expanded states can be reduced by AP. (2) Another reason is the limitation of the partition block size. For AP, a smaller partition block size means that the order graph is partitioned into smaller parts, which can improve efficiency. However, for HP, the number of partitions (or the partition block size) and the tightness of the heuristic function (Section 4.2) must be balanced, but this is difficult for larger networks. For example, in the alarm (with 10000 samples), the size of the AP partition block was {1, 16, 1, 1, 17, 1}, which was relatively quick at creating static pattern databases for 16 and 17 variables and is feasible. However, the size of the HP partition block was {16, 15, 6}. As mentioned earlier, the original algorithms used a default simple partition (the size of AP partition blocks was {19, 18} for alarm), and they cannot obtain the solution. According to Section 4.2, {16, 15, 6} was less tight than {19, 18}, indicating that the algorithms with the HP partition block {16, 15, 6} expanded more states than the original algorithms and could not obtain the solution. In addition, we did not limit the maximum size of the partition blocks  and obtained another size of the HP partition block as {36, 1}. Algorithms cannot create static pattern databases because the backward breadth-first search for 36 variables runs out of memory. Therefore, for larger networks, HP constraints often face a dilemma. Either the partition of small blocks cannot guarantee fewer expanded states, or static pattern databases cannot be created for the partition of large blocks.

In summary, for larger BNs, the algorithms with AP constraints are more efficient than those with HP constraints and combining these two constraints can improve efficiency.

In terms of accuracy, in larger-scale BNs, algorithms with AP constraints performed more poorly than those with HP constraints. The reason for this is as follows. From Sections 3 and 4, if a directed graph composed of all PPSs may have only one SCC or the maximum size of SCCs may be larger than , then the best  PPSs are used for each node to build the directed graph and extract SCCs. In this case, AP constraints may be inaccurate; that is, the ancestral partition is not consistent with the actual BN structure, leading to a slight loss of accuracy in structure learning. For a directed graph built from all PPSs in a larger-scale BN, it is challenging to extract SCCs that meet the required size . Therefore, the greedy strategy of using the best  PPSs is more likely to be applied in larger-scale networks, and it is more likely to lose accuracy in larger-scale networks. In contrast, regardless of the heuristic partition,  is always admissible and consistent, and it does not affect the structure learning accuracy. Owing to the above reasons, the accuracy loss of algorithms with AP-HP is derived only from AP. With only a slight loss of accuracy, AP-HP and AP can significantly improve the efficiency of many algorithms in the order graph.

Some interesting results deserve further attention. In some cases of boerlage92, especially when the sample size is 5000, 10000, and 15000, algorithms with HP have more running time and expanded states than their original versions. This is because the effect of the heuristic partitions obtained by Algorithm 2 is inferior to that of the default simple partition  and  on boerlage92. Such cases only appeared in boerlage92. In other cases, the algorithms with HP had fewer expanded states. However, fewer states do not always mean less running time. For example, in the meta and mushroom datasets, algorithms with HP and algorithms with AP-HP had a small number of expanded states, but their running times increased. Heuristic partitions of (, ) and (, ) were obtained for the meta and mushroom datasets, respectively. These partitions are unbalanced. Under these HP constraints, algorithms performed a backward breadth-first search in the reverse order graph, which is only one smaller than the current size required to create static pattern databases for , and the time cost increased.

The results of the three algorithms A\*, AWA\*, and BFBnB have been also reviewed under the same constraints in Tables 2–7. In general, with AP and/or HP constraints, AWA\* had a longer running time than the other two algorithms. Despite its lower time efficiency, it can obtain the current approximate solution within a limited time, as with GOBNILP. In general, BFBnB had a much greater number of expanded states than the other two algorithms because it expanded states in a breadth-first manner, while the other two algorithms choose the current lowest *f*-value state. However, BFBnB can save the expanded states to disk in a layered structure, thereby reducing RAM consumption. Furthermore, this algorithm can use a high-speed read and write technique to read and save related information about states. Despite more expanded states, it can obtain a solution more rapidly than the other two algorithms in larger-scale BNs. Considering the running time and the number of expanded states, A\* is a more balanced algorithm.

In addition, in Tables 8–9, the running times for calculating AP (Algorithm 1) and HP (Algorithm 2) were recorded on the benchmark BN and UCI datasets. For Algorithm 1, calculating the ancestral partition was relatively quick. The AP can be easily obtained, and a graph-partitioning algorithm only requires a few iterations. By contrast, calculating the heuristic partition required more time because it involved creating static pattern databases based on the current HP to calculate . The  of Algorithm 2 was set to obtain the maximum size of a heuristic partition block of approximately 15–20. Sizes near this value can still create static pattern databases relatively quickly. However, because the meta and mushroom heuristic partitions are ,  and , , the running time for creating static pattern databases for 21 or 22 variables increased significantly. Hence, the running time of Algorithm 2 also increased significantly. This result is consistent with previous conclusions. According to the data in Tables 8–9, Algorithms 1 and 2 generally consume less time to calculate AP and HP within a reasonable running time.

Table 8: Running time for calculating the ancestral and heuristic partition on benchmark BNs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | n | N | AP Time(s) | HP Time(s) |
| sachs | 11 | 500 | 0.001 | 0.001 |
| sachs | 11 | 1000 | 0.001 | 0.001 |
| sachs | 11 | 5000 | 0.001 | 0.002 |
| sachs | 11 | 10000 | 0.001 | 0.002 |
| sachs | 11 | 15000 | 0.001 | 0.002 |
| child | 20 | 500 | 0.001 | 0.012 |
| child | 20 | 1000 | 0.001 | 0.014 |
| child | 20 | 5000 | 0.003 | 0.022 |
| child | 20 | 10000 | 0.005 | 0.028 |
| child | 20 | 15000 | 0.008 | 0.042 |
| boerlage92 | 23 | 500 | 0.001 | 0.019 |
| boerlage92 | 23 | 1000 | 0.001 | 0.017 |
| boerlage92 | 23 | 5000 | 0.001 | 0.087 |
| boerlage92 | 23 | 10000 | 0.001 | 0.112 |
| boerlage92 | 23 | 15000 | 0.002 | 0.110 |
| insurance | 27 | 500 | 0.001 | 0.015 |
| insurance | 27 | 1000 | 0.001 | 0.021 |
| insurance | 27 | 5000 | 0.002 | 0.050 |
| insurance | 27 | 10000 | 0.002 | 1.508 |
| insurance | 27 | 15000 | 0.002 | 0.393 |
| water | 32 | 500 | 0.001 | 0.027 |
| water | 32 | 1000 | 0.001 | 0.042 |
| water | 32 | 5000 | 0.001 | 0.046 |
| water | 32 | 10000 | 0.001 | 0.045 |
| water | 32 | 15000 | 0.001 | 0.051 |
| alarm | 37 | 500 | 0.001 | 0.133 |
| alarm | 37 | 1000 | 0.001 | 0.075 |
| alarm | 37 | 5000 | 0.003 | 0.170 |
| alarm | 37 | 10000 | 0.003 | 0.137 |
| alarm | 37 | 15000 | 0.003 | 0.120 |
| hailfinder | 56 | 500 | 0.004 | 0.175 |
| hailfinder | 56 | 1000 | 0.006 | 0.273 |
| hailfinder | 56 | 5000 | 0.007 | 0.697 |
| hailfinder | 56 | 10000 | 0.007 | 0.750 |
| hailfinder | 56 | 15000 | 0.007 | 0.753 |

Table 9: Running time for calculating the ancestral and heuristic partitions on UCI datasets.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | n | N | AP Time(s) | HP Time(s) |
| zoo | 17 | 101 | 0.001 | 0.015 |
| voting | 17 | 435 | 0.001 | 0.009 |
| statlog | 19 | 752 | 0.004 | 0.047 |
| hepatitis | 20 | 126 | 0.001 | 0.013 |
| segment | 20 | 2310 | 0.004 | 0.047 |
| imports | 22 | 205 | 0.001 | 0.208 |
| meta | 22 | 528 | 0.047 | 4.224 |
| heart | 23 | 212 | 0.001 | 0.017 |
| horse | 23 | 300 | 0.001 | 0.081 |
| mushroom | 23 | 8124 | 0.013 | 4.084 |
| autos | 26 | 159 | 0.003 | 1.590 |
| steel | 28 | 1941 | 0.006 | 0.073 |
| flag | 29 | 194 | 0.001 | 0.138 |
| soybean | 36 | 266 | 0.004 | 0.504 |
| bands | 39 | 277 | 0.001 | 0.084 |
| spectf | 45 | 267 | 0.001 | 0.080 |
| sponge | 45 | 76 | 0.002 | 0.242 |
| lung cancer | 57 | 32 | 0.003 | 2.184 |
| splice | 61 | 3190 | 0.001 | 0.153 |

## 5.2 Comparisons with other algorithms

### 5.2.1 Comparisons with GOBNILP

This section compares the A\*, AWA\*, and BFBnB under AP/HP/AP-HP constraints with GOBNILP on benchmark BNs, UCI datasets, and datasets with more than 100 nodes, as shown in Tables 10–12. Only the running times for A\*, AWA\*, and BFBnB with AP/HP/AP-HP that exceeded the running time of GOBNILP are presented, and the bold numbers indicate the shortest running times of the four algorithms. In addition, the score error ratios for A\*, AWA\*, and BFBnB with AP/AP-HP are shown in the last column.

As shown in Tables 10–12, with the AP and/or HP, A\*, AWA\*, and BFBnB had lower running times than GOBNILP in many cases. Comparing Tables 2–7 with Tables 10–12, without these constraints, GOBNILP had a shorter running time and could search larger-scale networks than the original A\*, AWA\*, and BFBnB. With the partition constraints being added, most of the shortest running times occurred in A\* and BFBnB. Moreover, the shortest running time was mainly based on AP-HP constraints, followed by AP constraints. Among the three algorithms, AWA\* was usually the least efficient in terms of running time, and thus, under the same constraints, its efficiency was also low. In terms of accuracy, A\*, AWA\*, and BFBnB with HP always obtained optimal scores because  is always admissible and consistent. As for the AP and AP-HP constraints, when , A\*, AWA\*, and BFBnB had lower running times than GOBNILP in most cases, but could also achieve the optimal scores as GOBNILP. When , they had a slight loss of accuracy with AP or AP-HP constraints because Algorithm 1 tended to use a greedy strategy to extract the SCCs for larger-scale networks, and the accuracy loss of this greedy strategy would be negligible. For datasets with over 100 nodes, their accuracy was reduced. In this case, Algorithm 1 extracted the SCCs when  and returned many partitions of a single node. These partitions formed AP constraints similar to partial ordering, and they were not consistent with the actual BNs, even incorrect, resulting in a large loss of accuracy.

In boerlage92, with a sample size of 5000, 10000, and 15000 in Table 10, algorithms with the AP and/or HP did not outperform GOBNILP in terms of running time. Considering AP in these situations, the size of the AP partition block was {22,1}, indicating that a network only smaller than the current size was searched, and therefore, the running time was only slightly reduced. With regard to HP in these situations, Algorithm 2 returned a worse heuristic partition plan than the default simple partitions  and , resulting in more expanded states and a higher running time. With the combination of the two factors, algorithms with AP and/or HP have achieved worse running time.

In summary, in most of the cases, A\*, AWA\*, and BFBnB under AP/AP-HP demonstrated a lower running time than GOBNILP, and the accuracy was close to that of GOBNILP within 100 nodes.

Table 10: Running time for A\*, AWA\*, BFBnB with AP/HP/AP-HP and GOBNILP, and score error ratio for A\*, AWA\*, BFBnB with AP/AP-HP on benchmark BNs.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | GOBNILP | A\* | | | AWA\* | | | BFBnB | | | Error |
| AP | HP | AP-HP | AP | HP | AP-HP | AP | HP | AP-HP |
| sachs | 11 | 500 | 0.161 | **0.001** | **0.001** | **0.001** | **0.001** | **0.001** | **0.001** | 0.003 | 0.004 | 0.004 | 0% |
| sachs | 11 | 1000 | 0.257 | **0.001** | **0.001** | **0.001** | **0.001** | **0.001** | **0.001** | 0.004 | 0.004 | 0.004 | 0% |
| sachs | 11 | 5000 | 0.297 | **0.002** | **0.002** | **0.002** | **0.002** | **0.002** | **0.002** | 0.005 | 0.005 | 0.005 | 0% |
| sachs | 11 | 10000 | 0.304 | **0.003** | **0.003** | **0.003** | **0.003** | **0.003** | **0.003** | 0.005 | 0.005 | 0.005 | 0% |
| sachs | 11 | 15000 | 0.219 | **0.003** | **0.003** | **0.003** | **0.003** | **0.003** | **0.003** | 0.005 | 0.005 | 0.005 | 0% |
| child | 20 | 500 | 0.247 | 0.040 | 0.023 | **0.022** | 0.045 | 0.026 | 0.027 | 0.029 | 0.035 | 0.034 | 0% |
| child | 20 | 1000 | 0.410 | 0.073 | 0.025 | **0.024** | 0.074 | 0.029 | 0.029 | 0.045 | 0.038 | 0.038 | 0% |
| child | 20 | 5000 | 4.159 | 0.076 | 0.066 | **0.062** | 0.405 | 0.070 | 0.068 | 0.168 | 0.102 | 0.105 | 0% |
| child | 20 | 10000 | 9.767 | 0.707 | 0.134 | **0.104** | 0.792 | 0.149 | 0.117 | 0.347 | 0.120 | 0.137 | 0% |
| child | 20 | 15000 | 22.668 | 0.914 | 0.227 | **0.173** | 1.050 | 0.249 | 0.212 | 0.499 | 0.221 | 0.207 | 0% |
| boerlage92 | 23 | 500 | 0.156 |  | 0.029 | **0.025** |  | 0.027 | 0.026 |  | 0.033 | 0.034 | 0% |
| boerlage92 | 23 | 1000 | 0.214 |  | 0.173 | **0.010** |  | 0.207 | **0.010** | 0.185 | 0.078 | 0.015 | 0% |
| boerlage92 | 23 | 5000 | 0.221 |  |  |  |  |  |  |  |  |  | 0% |
| boerlage92 | 23 | 10000 | 0.475 |  |  |  |  |  |  |  |  |  | 0% |
| boerlage92 | 23 | 15000 | 1.095 |  |  |  |  |  |  |  |  |  | 0% |
| insurance | 27 | 500 | 0.928 |  | 0.640 | 0.136 |  | 0.633 | 0.729 |  | 0.213 | **0.067** | 0% |
| insurance | 27 | 1000 | 0.921 |  |  |  |  |  |  |  | 0.761 | **0.392** | 0% |
| insurance | 27 | 5000 | 5.874 |  | 3.091 | 1.432 |  | 3.653 | 1.843 |  | 0.792 | **0.399** | 0% |
| insurance | 27 | 10000 | 7.136 |  |  | 1.886 |  |  | 1.828 |  |  | **1.809** | 0% |
| insurance | 27 | 15000 | 9.559 |  | 1.974 | 1.845 |  | 1.988 | 1.866 |  | **1.831** | 1.848 | 0% |
| water | 32 | 500 | 1.833 | 1.709 |  | **0.019** |  |  | 0.020 | 0.566 |  | 0.037 | 0.048% |
| water | 32 | 1000 | 4.717 | 2.319 |  | **0.026** | 3.498 |  | 0.028 | 0.828 |  | 0.037 | 0.008% |
| water | 32 | 5000 | 2.573 |  |  | **0.025** |  |  | **0.025** | 1.020 | 1.390 | 0.033 | 0.003% |
| water | 32 | 10000 | 7.391 | 0.184 |  | **0.033** | 0.218 |  | 0.034 | 0.192 | 1.634 | 0.034 | 0.001% |
| water | 32 | 15000 | 18.770 | 2.802 |  | **0.033** | 4.357 |  | 0.035 | 1.009 | 7.143 | 0.052 | 0.004% |
| alarm | 37 | 500 | 2.241 | **0.107** |  | 0.164 | 0.109 |  | 0.108 | 0.127 |  | 0.120 | 0.401% |
| alarm | 37 | 1000 | 3.154 | 0.167 |  | 0.091 | 0.200 |  | 0.097 | 0.117 |  | **0.085** | 0.062% |
| alarm | 37 | 5000 | 12.876 | 0.226 |  | 0.296 | 0.270 |  | 0.216 | **0.166** |  | 0.283 | 0.021% |
| alarm | 37 | 10000 | 25.663 | 0.256 |  | 0.155 | 0.325 |  | 0.163 | 0.184 |  | **0.153** | 0.010% |
| alarm | 37 | 15000 | 23.002 | 0.307 |  | 0.161 | 0.298 |  | 0.169 | 0.194 |  | **0.144** | 0.005% |
| hailfinder | 56 | 500 | 0.811 | **0.066** |  | 0.098 | 0.067 |  | 0.070 | 0.109 |  | 0.091 | 0.013% |
| hailfinder | 56 | 1000 | 1.920 | **0.064** |  | 0.084 | **0.064** |  | 0.077 | 0.070 |  | 0.088 | 0.023% |
| hailfinder | 56 | 5000 | 6.559 | 1.564 |  | 0.231 | 2.901 |  | 1.816 | 0.638 |  | **0.215** | 0.025% |
| hailfinder | 56 | 10000 | 34.393 | 0.291 |  | 0.356 | **0.285** |  | 0.290 | 0.344 |  | 0.335 | 0.067% |
| hailfinder | 56 | 15000 | 68.880 | 1.662 |  | 0.469 | 2.890 |  | 1.582 | 0.783 |  | **0.404** | 0.043% |
| win95pt | 76 | 500 | 527.834 | **1.469** |  | 1.492 | 2.571 |  | 1.505 | 2.190 |  | 1.590 | 0.653% |
| win95pt | 76 | 1000 | 244.868 | 2.071 |  | **2.059** | 3.588 |  | 2.076 | 2.425 |  | 2.282 | 2.830% |
| win95pt | 76 | 5000 | 3169.231 | 3.359 |  | **3.350** | 4.330 |  | 3.360 | 3.759 |  | 3.597 | 2.881% |
| win95pt | 76 | 10000 | 2920.649 | 3.979 |  | **3.972** | 5.361 |  | 4.004 | 4.837 |  | 4.225 | 2.706% |
| win95pt | 76 | 15000 | 4814.765 | 3.703 |  | **3.514** | 4.739 |  | 4.567 | 4.121 |  | 3.831 | 2.880% |

Table 11: Running time for A\*, AWA\*, BFBnB with AP/HP/AP-HP and GOBNILP, and score error ratio for A\*, AWA\*, BFBnB with AP/AP-HP on UCI datasets.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | GOBNILP | A\* | | | AWA\* | | | BFBnB | | | Error |
| AP | HP | AP-HP | AP | HP | AP-HP | AP | HP | AP-HP |
| zoo | 17 | 101 | 4.754 | **0.022** | 0.025 | 0.031 | 0.048 | 0.028 | 0.034 | 0.027 | 0.029 | 0.034 | 0% |
| voting | 17 | 435 | 0.562 | **0.010** | 0.024 | 0.027 | **0.010** | 0.022 | 0.030 | 0.015 | 0.027 | 0.033 | 0% |
| statlog | 19 | 752 | 13.600 | 0.473 | **0.100** | 0.134 | 0.278 | 0.108 | 0.146 | 0.359 | 0.155 | 0.164 | 0% |
| hepatitis | 20 | 126 | 0.397 | **0.021** | 0.022 | 0.025 | **0.021** | 0.024 | 0.027 | 0.034 | 0.033 | 0.037 | 0% |
| segment | 20 | 2310 | 12.963 | 0.441 | **0.054** | 0.099 | 0.663 | 0.057 | 0.103 | 0.276 | 0.093 | 0.116 | 0% |
| imports | 22 | 205 | 9.944 | 5.237 | 0.563 | 0.291 | 6.731 | 0.797 | 0.365 | 1.829 | 0.324 | **0.215** | 0% |
| meta | 22 | 528 | OT | **3.318** | 7.667 | 7.880 | 3.668 | 7.665 | 7.662 | 3.205 | 7.272 | 7.644 | 0% |
| heart | 23 | 212 | 0.310 |  |  | 0.240 |  |  | 0.306 |  | 0.242 | **0.115** | 0% |
| horse | 23 | 300 | 2.068 |  | 1.877 | 0.391 |  |  | 0.495 | 1.298 | 0.691 | **0.199** | 0% |
| mushroom | 23 | 8124 | OT | 0.410 | 10.912 | 10.644 | 0.557 | 11.668 | 11.767 | **0.366** | 11.764 | 11.730 | 0% |
| autos | 26 | 159 | 13.325 |  | 9.790 | 5.251 |  | 13.251 | 7.024 | 6.969 | 3.867 | **2.184** | 0% |
| steel | 28 | 1941 | 74.392 |  | 0.999 | 0.365 |  | 4.959 | 1.157 |  | 0.346 | **0.179** | 0% |
| flag | 29 | 194 | **1.243** |  |  |  |  |  |  |  |  |  | 0.004% |
| soybean | 36 | 266 | 22.241 | 0.398 |  | **0.393** | 0.430 |  | 0.398 | 0.416 |  | 0.421 | 0.487% |
| bands | 39 | 277 | 31.657 | **0.120** |  | 0.133 | 0.149 |  | 0.118 | 0.326 |  | 0.138 | 0.194% |
| spectf | 45 | 267 | 6.402 | **0.069** |  | 0.087 | 0.070 |  | **0.069** | 0.097 |  | 0.095 | 0.098% |
| sponge | 45 | 76 | 0.670 | 0.310 |  | **0.300** | 0.337 |  | 0.342 | 0.326 |  | 0.321 | 1.304% |
| lung cancer | 57 | 32 | 4.62 | 0.472 |  | **0.441** | 0.490 |  | 0.465 | 0.501 |  | 0.446 | 2.159% |
| splice | 61 | 3190 | 304.946 | 0.167 |  | **0.158** | 0.171 |  | 0.160 | 0.200 |  | 0.190 | 0.123% |

Table 12: Running time for A\*, AWA\*, BFBnB with AP/HP/AP-HP and GOBNILP, and score error ratio for A\*, AWA\*, BFBnB with AP/AP-HP on datasets with more than 100 nodes.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | GOBNILP | A\* | | | AWA\* | | | BFBnB | | | Error |
| AP | HP | AP-HP | AP | HP | AP-HP | AP | HP | AP-HP |
| pathfinder | 109 | 5000 | OT | 48.897 |  | **46.700** | 51.921 |  | 47.933 | 48.340 |  | 48.819 | 42.364% |
| synthetic 1 | 120 | 10000 | 193.093 | **10.051** |  | 10.055 | 11.380 |  | 10.249 | 11.860 |  | 11.809 | 21.298% |
| synthetic 2 | 150 | 10000 | 322.424 | 16.269 |  | **16.096** | 21.111 |  | 16.486 | 18.493 |  | 18.738 | 20.188% |
| synthetic 3 | 175 | 10000 | 584.062 | 16.943 |  | **16.608** | 20.979 |  | 16.667 | 18.053 |  | 17.454 | 30.977% |
| andes | 223 | 5000 | OT | 19.735 |  | **19.039** | 22.677 |  | 19.633 | 19.142 |  | 19.226 | 48.135% |

### 5.2.2 Comparisons with MMHC

This section compares the A\*, AWA\*, and BFBnB under AP/HP/AP-HP constraints with MMHC on benchmark BNs, UCI datasets, and datasets with more than 100 nodes, as shown in Tables 13–15. Only the running times for A\*, AWA\*, and BFBnB with AP/HP/AP-HP that exceeded the running time of MMHC are presented, and the bold numbers indicate the shortest running times of the four algorithms. In addition, the score error ratios for MMHC are listed in the 5th column, and the score error ratios for A\*, AWA\*, and BFBnB with AP/AP-HP are listed in the last column.

As shown in Tables 13–15, with the AP and/or HP, these algorithms had lower running times than MMHC in many cases. Comparing Tables 2–7 with Tables 13–15, without these constraints, MMHC had a shorter running time, less than GOBNILP, and can search larger-scale networks than the original A\*, AWA\*, and BFBnB. With added constraints, most of the shortest running times occurred in A\* and BFBnB. Moreover, the shortest running times mainly occurred with AP-HP constraints, followed by AP constraints. Among the three algorithms, AWA\* was usually the least efficient in terms of running time, and thus, under the same constraints, its efficiency was also low.

In terms of accuracy, the MMHC performed less accurately on many datasets. For the same dataset, MMHC achieved better results with a larger sample size than a smaller sample size. This is because the MMPC algorithm (the first phase of the MMHC algorithm) uses CI tests to learn the parent and children sets. CI tests require a large number of samples; therefore, MMHC has low accuracy when the sample size is smaller. A\*, AWA\*, and BFBnB with HP always obtained optimal scores because  is always admissible and consistent. When , A\*, AWA\*, and BFBnB with AP/AP-HP also achieved optimal scores. When , although A\*, AWA\*, and BFBnB under AP/AP-HP had a slight loss of accuracy, they are still better than MMHC. On datasets with over 100 nodes, A\*, AWA\*, and BFBnB with AP/AP-HP produced worse accuracy than MMHC. In this case, Algorithm 1 extracted the SCCs when  and returned many partitions of a single node. These partitions formed AP constraints similar to partial orderings, and they were incorrect, resulting in a large loss of accuracy.

In boerlage92, with a sample size 5000, 10000, and 15000, algorithms with the AP and/or HP did not outperform MMHC in running time. The reason for this has been discussed in Section 5.2.1.

In summary, in most of the cases, A\*, AWA\*, and BFBnB under AP/AP-HP had a lower running time than MMHC, and the accuracy was higher than that of MMHC within 100 nodes.

Table 13: Running time and score error ratio for A\*, AWA\*, BFBnB with AP/HP/AP-HP and MMHC on benchmark BNs.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | MMHC | | A\* | | | AWA\* | | | BFBnB | | | Error |
| time(s) | Error | AP | HP | AP-HP | AP | HP | AP-HP | AP | HP | AP-HP |
| sachs | 11 | 500 | 0.345 | 0.037% | **0.001** | **0.001** | **0.001** | **0.001** | **0.001** | 0.001 | 0.003 | 0.004 | 0.004 | 0% |
| sachs | 11 | 1000 | 0.352 | 0% | **0.001** | **0.001** | **0.001** | **0.001** | **0.001** | 0.001 | 0.004 | 0.004 | 0.004 | 0% |
| sachs | 11 | 5000 | 0.974 | 0% | **0.002** | **0.002** | **0.002** | **0.002** | **0.002** | 0.002 | 0.005 | 0.005 | 0.005 | 0% |
| sachs | 11 | 10000 | 1.491 | 0% | **0.003** | **0.003** | **0.003** | **0.003** | **0.003** | 0.003 | 0.005 | 0.005 | 0.005 | 0% |
| sachs | 11 | 15000 | 2.289 | 0% | **0.003** | **0.003** | **0.003** | **0.003** | **0.003** | 0.003 | 0.005 | 0.005 | 0.005 | 0% |
| child | 20 | 500 | 0.578 | 4.063% | **0.040** | **0.023** | **0.022** | **0.045** | **0.026** | 0.027 | 0.029 | 0.035 | 0.034 | 0% |
| child | 20 | 1000 | 0.730 | 0.143% | 0.073 | 0.025 | **0.024** | 0.074 | 0.029 | 0.029 | 0.045 | 0.038 | 0.038 | 0% |
| child | 20 | 5000 | 2.552 | 0.346% | 0.076 | 0.066 | **0.062** | 0.405 | 0.070 | 0.068 | 0.168 | 0.102 | 0.105 | 0% |
| child | 20 | 10000 | 5.545 | 0.288% | 0.707 | 0.134 | **0.104** | 0.792 | 0.149 | 0.117 | 0.347 | 0.120 | 0.137 | 0% |
| child | 20 | 15000 | 9.074 | 0.345% | 0.914 | 0.227 | **0.173** | 1.050 | 0.249 | 0.212 | 0.499 | 0.221 | 0.207 | 0% |
| boerlage92 | 23 | 500 | 0.367 | 0.979% |  | 0.029 | **0.025** |  | 0.027 | 0.026 |  | 0.033 | 0.034 | 0% |
| boerlage92 | 23 | 1000 | 0.504 | 0.940% | 0.443 | 0.173 | **0.010** | 0.450 | 0.207 | 0.010 | 0.185 | 0.078 | 0.015 | 0% |
| boerlage92 | 23 | 5000 | 0.773 | 0.191% |  |  |  |  |  |  |  |  |  | 0% |
| boerlage92 | 23 | 10000 | 1.242 | 0.067% |  |  |  |  |  |  |  |  |  | 0% |
| boerlage92 | 23 | 15000 | 1.631 | 0.062% |  |  |  |  |  |  |  |  |  | 0% |
| insurance | 27 | 500 | 0.644 | 8.030% |  | 0.640 | 0.136 |  | 0.633 |  |  | 0.213 | **0.067** | 0% |
| insurance | 27 | 1000 | 0.912 | 4.135% |  |  |  |  |  |  |  | 0.761 | **0.392** | 0% |
| insurance | 27 | 5000 | 4.817 | 1.0493% |  | 3.091 | 1.432 |  | 3.653 | 1.843 |  | 0.792 | **0.399** | 0% |
| insurance | 27 | 10000 | 10.995 | 2.400% |  | 10.033 | 1.886 |  | 10.309 | 1.828 |  | 9.367 | **1.809** | 0% |
| insurance | 27 | 15000 | 23.683 | 2.223% |  | 1.974 | 1.845 |  | 1.988 | 1.866 |  | **1.831** | 1.848 | 0% |
| water | 32 | 500 | 0.382 | 47.208% |  |  | **0.019** |  |  | 0.020 |  |  | 0.037 | 0.048% |
| water | 32 | 1000 | 0.508 | 33.820% |  |  | **0.026** |  |  | 0.028 |  |  | 0.037 | 0.008% |
| water | 32 | 5000 | 1.017 | 10.919% |  |  | **0.025** |  |  | **0.025** |  |  | 0.033 | 0.003% |
| water | 32 | 10000 | 1.871 | 5.933% | 0.184 |  | **0.033** | 0.218 |  | 0.034 | 0.192 | 1.634 | 0.034 | 0.001% |
| water | 32 | 15000 | 2.748 | 4.075% |  |  | **0.033** |  |  | 0.035 |  |  | 0.052 | 0.004% |
| alarm | 37 | 500 | 0.714 | 1.225% | **0.107** |  | 0.164 | 0.109 |  | 0.108 | 0.127 |  | 0.120 | 0.401% |
| alarm | 37 | 1000 | 0.884 | 1.816% | 0.167 |  | 0.091 | 0.200 |  | 0.097 | 0.117 |  | **0.085** | 0.062% |
| alarm | 37 | 5000 | 2.298 | 1.104% | 0.226 |  | 0.296 | 0.270 |  | 0.216 | **0.166** |  | 0.283 | 0.021% |
| alarm | 37 | 10000 | 4.158 | 1.220% | 0.256 |  | 0.155 | 0.325 |  | 0.163 | 0.184 |  | **0.153** | 0.010% |
| alarm | 37 | 15000 | 6.736 | 0.904% | 0.307 |  | 0.161 | 0.298 |  | 0.169 | 0.194 |  | **0.144** | 0.005% |
| hailfinder | 56 | 500 | 7.229 | 2.748% | **0.066** |  | 0.098 | 0.067 |  | 0.070 | 0.109 |  | 0.091 | 0.013% |
| hailfinder | 56 | 1000 | 2.837 | 14.49% | **0.064** |  | 0.084 | **0.064** |  | 0.077 | 0.070 |  | 0.088 | 0.023% |
| hailfinder | 56 | 5000 | 40.775 | 5.587% | 1.564 |  | 0.231 | 2.901 |  | 1.816 | 0.638 |  | **0.215** | 0.025% |
| hailfinder | 56 | 10000 | 142.147 | 3.811% | 0.291 |  | 0.356 | 0.285 |  | 0.290 | 0.344 |  | 0.335 | 0.067% |
| hailfinder | 56 | 15000 | 307.990 | 2.252% | 1.662 |  | 0.469 | 2.890 |  | 1.582 | 0.783 |  | **0.404** | 0.043% |
| win95pt | 76 | 500 | 1.862 | 7.765% | **1.469** |  | 1.492 | 2.571 |  | 1.505 | 2.190 |  | 1.590 | 0.653% |
| win95pt | 76 | 1000 | 5.734 | 3.378% | 2.071 |  | **2.059** | 3.588 |  | 2.076 | 2.425 |  | 2.282 | 2.830% |
| win95pt | 76 | 5000 | 55.928 | 4.911% | 3.359 |  | **3.350** | 4.330 |  | 3.360 | 3.759 |  | 3.597 | 2.881% |
| win95pt | 76 | 10000 | 509.719 | 3.089% | 3.979 |  | **3.972** | 5.361 |  | 4.004 | 4.837 |  | 4.225 | 2.706% |
| win95pt | 76 | 15000 | 659.807 | 5.209% | 3.703 |  | **3.514** | 4.739 |  | 4.567 | 4.121 |  | 3.831 | 2.880% |

Table 14: Running time and score error ratio for A\*, AWA\*, BFBnB with AP/HP/AP-HP and MMHC on UCI datasets.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | MMHC | | A\* | | | AWA\* | | | BFBnB | | | Error |
| time(s) | Error | AP | HP | AP-HP | AP | HP | AP-HP | AP | HP | AP-HP |
| zoo | 17 | 101 | 0.346 | 4.733% | **0.022** | 0.025 | 0.031 | 0.048 | 0.028 | 0.034 | 0.027 | 0.029 | 0.034 | 0% |
| voting | 17 | 435 | 0.592 | 4.432% | **0.010** | 0.024 | 0.027 | 0.010 | 0.022 | 0.030 | 0.015 | 0.027 | 0.033 | 0% |
| statlog | 19 | 752 | 0.767 | 1.226% | 0.473 | **0.100** | 0.134 | 0.278 | 0.108 | 0.146 | 0.359 | 0.155 | 0.164 | 0% |
| hepatitis | 20 | 126 | 0.344 | 1.469% | **0.021** | 0.022 | 0.025 | 0.021 | 0.024 | 0.027 | 0.034 | 0.033 | 0.037 | 0% |
| segment | 20 | 2310 | 2.621 | 2.637% | 0.441 | **0.054** | 0.099 | 0.663 | 0.057 | 0.103 | 0.276 | 0.093 | 0.116 | 0% |
| imports | 22 | 205 | 0.585 | 5.290% |  | 0.563 | 0.291 |  | 0.797 | 0.365 |  | 0.324 | **0.215** | 0% |
| meta | 22 | 528 | 7.793 | 45.301% | **3.318** | 7.667 |  | 3.668 | 7.665 | 7.662 | 3.205 | 7.272 | 7.644 | 0% |
| heart | 23 | 212 | 0.485 | 1.514% |  |  | 0.240 |  |  | 0.306 |  | 0.242 | **0.115** | 0% |
| horse | 23 | 300 | 0.486 | 0.788% |  |  | 0.391 |  |  |  |  |  | **0.199** | 0% |
| mushroom | 23 | 8124 | 228.549 | 82.367% | 0.410 | 10.912 | 10.644 | 0.557 | 11.668 | 11.767 | **0.366** | 11.764 | 11.730 | 0% |
| autos | 26 | 159 | 0.588 | 5.192% |  |  |  |  |  |  |  |  |  | 0% |
| steel | 28 | 1941 | 18.111 | 0.466% |  | 0.999 | 0.365 |  | 4.959 | 1.157 |  | 0.346 | **0.179** | 0% |
| flag | 29 | 194 | 0.641 | 1.165% |  |  |  |  |  |  |  |  |  | 0.004% |
| soybean | 36 | 266 | 1.133 | 6.240% | 0.398 |  | **0.393** | 0.430 |  | 0.398 | 0.416 |  | 0.421 | 0.487% |
| bands | 39 | 277 | 0.618 | 1.915% | **0.120** |  | 0.133 | 0.149 |  | 0.118 | 0.326 |  | 0.138 | 0.194% |
| spectf | 45 | 267 | 0.721 | 0.037% | **0.069** |  | 0.087 | 0.070 |  | **0.069** | 0.097 |  | 0.095 | 0.098% |
| sponge | 45 | 76 | 0.899 | 10.263% | 0.310 |  | **0.300** | 0.337 |  | 0.342 | 0.326 |  | 0.321 | 1.304% |
| lung cancer | 57 | 32 | 0.622 | 4.758% | 0.472 |  | **0.441** | 0.490 |  | 0.465 | 0.501 |  | 0.446 | 2.159% |
| splice | 61 | 3190 | 110.072 | 0.894% | 0.167 |  | **0.158** | 0.171 |  | 0.160 | 0.200 |  | 0.190 | 0.123% |

Table 15: Running time and score error ratio for A\*, AWA\*, BFBnB with AP/HP/AP-HP and MMHC on datasets with more than 100 nodes.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | MMHC | | A\* | | | AWA\* | | | BFBnB | | | Error |
| time(s) | Error | AP | HP | AP-HP | AP | HP | AP-HP | AP | HP | AP-HP |
| pathfinder | 109 | 5000 | 1600.815 | 53.679% | 48.897 |  | **46.700** | 51.921 |  | 47.933 | 48.340 |  | 48.819 | 42.364% |
| synthetic 1 | 120 | 10000 | 15.237 | 6.468% | **10.051** |  | 10.055 | 11.380 |  | 10.249 | 11.860 |  | 11.809 | 21.298% |
| synthetic 2 | 150 | 10000 | 19.408 | 7.539% | 16.269 |  | **16.096** |  |  | 16.486 | 18.493 |  | 18.738 | 20.188% |
| synthetic 3 | 175 | 10000 | 30.195 | 7.873% | 16.943 |  | **16.608** | 20.979 |  | 16.667 | 18.053 |  | 17.454 | 30.977% |
| andes | 223 | 5000 | 71.521 | 8.159% | 19.735 |  | **19.039** | 22.677 |  | 19.633 | 19.142 |  | 19.226 | 48.135% |

### 5.2.3 Comparisons with NOTEARS

This section compares A\*, AWA\*, and BFBnB under AP/HP/AP-HP constraints with NOTEARS on benchmark BNs, UCI datasets, and datasets with more than 100 nodes, as shown in Tables 16–18. Only the running times for A\*, AWA\*, and BFBnB with AP/HP/AP-HP that exceeded the running time of NOTEARS are presented, and the bold numbers indicate the shortest running times of the four algorithms. In addition, the score error ratios for NOTEARS are listed in the 5th column, and the score error ratios for A\*, AWA\*, and BFBnB with AP/AP-HP are listed in the last column.

In terms of running time, with the AP and/or HP, these algorithms had lower running times than NOTEARS in many cases. Comparing Tables 2–7 with Tables 13–15, without these constraints, NOTEARS had a lower running time, less than GOBNILP, and could search larger-scale networks than the original A\*, AWA\*, and BFBnB. The running time of NOTEARS was close to that of MMHC, but the running time of NOTEARS did not change much with a change in sample and network sizes. After adding the constraints, most of the shortest running times appeared in A\* and BFBnB. In addition, the shortest running times mainly occurred with AP-HP constraints, followed by AP constraints.

In terms of accuracy, NOTEARS exhibited varying degrees of accuracy loss across the datasets. NOTEARS obtained better results with a smaller sample size, which was the opposite of MMHC. A\*, AWA\*, and BFBnB with HP always obtained optimal scores because of admissibility and consistency of the heuristic function. When , A\*, AWA\*, and BFBnB with AP/AP-HP obtained the optimal scores or slight accuracy losses, performing better than NOTEARS. On datasets with more than 100 nodes, A\*, AWA\*, and BFBnB with AP/AP-HP had worse accuracies than NOTEARS in general. Algorithm 1 extracted the SCCs under  and returned many partitions of a single node on datasets with more than 100 nodes. These single nodes formed AP constraints similar to partial orderings, which was incorrect, resulting in a high loss of accuracy. In pathfinder, NOTEARS could not calculate the score because it ran out of memory, indicating a worse score.

In boerlage92 with a sample size of 5000, 10000, and 15000, algorithms with the AP and/or HP did not outperform NOTEARS in terms of running time. The reason for this has been discussed in Section 5.2.1.

In summary, in most of the cases, A\*, AWA\*, and BFBnB under AP/AP-HP had a lower running time than NOTEARS, and the accuracy was higher than that of NOTEARS within 100 nodes.

Table 16: Running time and score error ratio for A\*, AWA\*, BFBnB with AP/HP/AP-HP and NOTEARS on benchmark BNs.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | NOTEARS | | A\* | | | AWA\* | | | BFBnB | | | Error |
| time(s) | Error | AP | HP | AP-HP | AP | HP | AP-HP | AP | HP | AP-HP |
| sachs | 11 | 500 | 0.739 | 4.347% | **0.001** | **0.001** | **0.001** | **0.001** | **0.001** | **0.001** | 0.003 | 0.004 | 0.004 | 0% |
| sachs | 11 | 1000 | 0.692 | 6.317% | **0.001** | **0.001** | **0.001** | **0.001** | **0.001** | **0.001** | 0.004 | 0.004 | 0.004 | 0% |
| sachs | 11 | 5000 | 1.247 | 11.524% | **0.002** | **0.002** | **0.002** | **0.002** | **0.002** | **0.002** | 0.005 | 0.005 | 0.005 | 0% |
| sachs | 11 | 10000 | 1.621 | 12.741% | **0.003** | **0.003** | **0.003** | **0.003** | **0.003** | **0.003** | 0.005 | 0.005 | 0.005 | 0% |
| sachs | 11 | 15000 | 2.263 | 13.264% | **0.003** | **0.003** | **0.003** | **0.003** | **0.003** | **0.003** | 0.005 | 0.005 | 0.005 | 0% |
| child | 20 | 500 | 1.396 | 25.437% | 0.040 | 0.023 | **0.022** | 0.045 | 0.026 | 0.027 | 0.029 | 0.035 | 0.034 | 0% |
| child | 20 | 1000 | 1.860 | 29.093% | 0.073 | 0.025 | **0.024** | 0.074 | 0.029 | 0.029 | 0.045 | 0.038 | 0.038 | 0% |
| child | 20 | 5000 | 3.522 | 29.981% | 0.076 | 0.066 | **0.062** | 0.405 | 0.070 | 0.068 | 0.168 | 0.102 | 0.105 | 0% |
| child | 20 | 10000 | 6.480 | 28.893% | 0.707 | 0.134 | **0.104** | 0.792 | 0.149 | 0.117 | 0.347 | 0.120 | 0.137 | 0% |
| child | 20 | 15000 | 8.955 | 28.248% | 0.914 | 0.227 | **0.173** | 1.050 | 0.249 | 0.212 | 0.499 | 0.221 | 0.207 | 0% |
| boerlage92 | 23 | 500 | 0.679 | 10.852% |  | 0.029 | **0.025** |  | 0.027 | 0.026 |  | 0.033 | 0.034 | 0% |
| boerlage92 | 23 | 1000 | 0.692 | 10.015% | 0.443 | 0.173 | **0.010** | 0.450 | 0.207 | **0.010** | 0.185 | 0.078 | 0.015 | 0% |
| boerlage92 | 23 | 5000 | 1.315 | 12.681% |  |  |  |  |  |  |  |  |  | 0% |
| boerlage92 | 23 | 10000 | 1.949 | 12.278% |  |  |  |  |  |  |  |  |  | 0% |
| boerlage92 | 23 | 15000 | 2.662 | 12.884% |  |  |  |  |  |  |  |  |  | 0% |
| insurance | 27 | 500 | 1.690 | 13.309% |  | 0.640 | 0.136 |  | 0.633 | 0.729 |  | 0.213 | **0.067** | 0% |
| insurance | 27 | 1000 | 1.947 | 16.782% |  |  | 1.404 |  |  |  |  | 0.761 | **0.392** | 0% |
| insurance | 27 | 5000 | 5.531 | 23.689% |  | 3.091 | 1.432 |  | 3.653 | 1.843 |  | 0.792 | **0.399** | 0% |
| insurance | 27 | 10000 | 10.563 | 24.995% |  | 10.033 | 1.886 |  | 10.309 | 1.828 |  | 9.367 | **1.809** | 0% |
| insurance | 27 | 15000 | 15.886 | 25.622% |  | 1.974 | 1.845 |  | 1.988 | 1.866 |  | **1.831** | 1.848 | 0% |
| water | 32 | 500 | 0.920 | 8.598% |  |  | **0.019** |  |  | 0.020 | 0.566 |  | 0.037 | 0.048% |
| water | 32 | 1000 | 0.932 | 7.934% |  |  | **0.026** |  |  | 0.028 | 0.828 |  | 0.037 | 0.008% |
| water | 32 | 5000 | 2.425 | 9.487% |  |  | **0.025** |  |  | **0.025** | 1.020 | 1.390 | 0.033 | 0.003% |
| water | 32 | 10000 | 3.829 | 9.238% | 0.184 |  | **0.033** | 0.218 |  | 0.034 | 0.192 | 1.634 | 0.034 | 0.001% |
| water | 32 | 15000 | 5.632 | 9.308% | 2.802 |  | **0.033** | 4.357 |  | 0.035 | 1.009 |  | 0.052 | 0.004% |
| alarm | 37 | 500 | 2.635 | 24.636% | **0.107** |  | 0.164 | 0.109 |  | 0.108 | 0.127 |  | 0.120 | 0.401% |
| alarm | 37 | 1000 | 4.658 | 27.798% | 0.167 |  | 0.091 | 0.200 |  | 0.097 | 0.117 |  | **0.085** | 0.062% |
| alarm | 37 | 5000 | 9.247 | 27.274% | 0.226 |  | 0.296 | 0.270 |  | 0.216 | **0.166** |  | 0.283 | 0.021% |
| alarm | 37 | 10000 | 15.941 | 27.942% | 0.256 |  | 0.155 | 0.325 |  | 0.163 | 0.184 |  | **0.153** | 0.010% |
| alarm | 37 | 15000 | 23.169 | 27.536% | 0.307 |  | 0.161 | 0.298 |  | 0.169 | 0.194 |  | **0.144** | 0.005% |
| hailfinder | 56 | 500 | 15.957 | 5.161% | 0.066 |  | 0.098 | 0.067 |  | 0.070 | 0.109 |  | 0.091 | 0.013% |
| hailfinder | 56 | 1000 | 14.718 | 7.764% | 0.064 |  | 0.084 | 0.064 |  | 0.077 | 0.070 |  | 0.088 | 0.023% |
| hailfinder | 56 | 5000 | 50.278 | 9.945% | 1.564 |  | 0.231 | 2.901 |  | 1.816 | 0.638 |  | **0.215** | 0.025% |
| hailfinder | 56 | 10000 | 57.964 | 11.214% | 0.291 |  | 0.356 | **0.285** |  | 0.290 | 0.344 |  | 0.335 | 0.067% |
| hailfinder | 56 | 15000 | 79.527 | 11.481% | 1.662 |  | 0.469 | 2.890 |  | 1.582 | 0.783 |  | **0.404** | 0.043% |
| win95pt | 76 | 500 | 1.572 | 42.738% | **1.469** |  | 1.492 |  |  | 1.505 |  |  |  | 0.653% |
| win95pt | 76 | 1000 | 2.138 | 50.191% | 2.071 |  | **2.059** |  |  | 2.076 |  |  |  | 2.830% |
| win95pt | 76 | 5000 | 4.841 | 48.658% | 3.359 |  | **3.350** | 4.330 |  | 3.360 | 3.759 |  | 3.597 | 2.881% |
| win95pt | 76 | 10000 | 9.403 | 45.937% | 3.979 |  | **3.972** | 5.361 |  | 4.004 | 4.837 |  | 4.225 | 2.706% |
| win95pt | 76 | 15000 | 12.221 | 46.292% | 3.703 |  | **3.514** | 4.739 |  | 4.567 | 4.121 |  | 3.831 | 2.880% |

Table 17: Running time and score error ratio for A\*, AWA\*, BFBnB with AP/HP/AP-HP and NOTEARS on UCI datasets.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | NOTEARS | | A\* | | | AWA\* | | | BFBnB | | | Error |
| time(s) | Error | AP | HP | AP-HP | AP | HP | AP-HP | AP | HP | AP-HP |
| zoo | 17 | 101 | 0.771 | 29.105% | **0.022** | 0.025 | 0.031 | 0.048 | 0.028 | 0.034 | 0.027 | 0.029 | 0.034 | 0% |
| voting | 17 | 435 | 0.899 | 4.063% | **0.010** | 0.024 | 0.027 | **0.010** | 0.022 | 0.030 | 0.015 | 0.027 | 0.033 | 0% |
| statlog | 19 | 752 | 3.657 | 23.504% | 0.473 | **0.100** | 0.134 | 0.278 | 0.108 | 0.146 | 0.359 | 0.155 | 0.164 | 0% |
| hepatitis | 20 | 126 | 0.484 | 8.630% | **0.021** | 0.022 | 0.025 | 0.021 | 0.024 | 0.027 | 0.034 | 0.033 | 0.037 | 0% |
| segment | 20 | 2310 | 1.390 | 23.792% | 0.441 | **0.054** | 0.099 | 0.663 | 0.057 | 0.103 | 0.276 | 0.093 | 0.116 | 0% |
| imports | 22 | 205 | 1.061 | 22.590% |  | 0.563 | 0.291 |  | 0.797 | 0.365 |  | 0.324 | **0.215** | 0% |
| meta | 22 | 528 | 0.635 | 82.871% |  |  |  |  |  |  |  |  |  | 0% |
| heart | 23 | 212 | 0.884 | 10.086% |  | 0.531 | 0.240 |  | 0.635 | 0.306 |  | 0.242 | **0.115** | 0% |
| horse | 23 | 300 | 0.568 | 11.571% |  |  | 0.391 |  |  | 0.495 |  |  | **0.199** | 0% |
| mushroom | 23 | 8124 | 47.689 | 63.887% | 0.410 | 10.912 | 10.644 | 0.557 | 11.668 | 11.767 | **0.366** | 11.764 | 11.730 | 0% |
| autos | 26 | 159 | 2.701 | 22.721% |  |  |  |  |  |  |  |  | **2.184** | 0% |
| steel | 28 | 1941 | 3.062 | 19.314% |  | 0.999 | 0.365 |  |  | 1.157 |  | 0.346 | **0.179** | 0% |
| flag | 29 | 194 | 0.575 | 12.451% |  |  |  |  |  |  |  |  |  | 0.004% |
| soybean | 36 | 266 | 1.092 | 28.621% | **0.398** |  | 0.393 | 0.430 |  | 0.398 | 0.416 |  | 0.421 | 0.487% |
| bands | 39 | 277 | 0.907 | 6.259% | **0.120** |  | 0.133 | 0.149 |  | 0.118 | 0.326 |  | 0.138 | 0.194% |
| spectf | 45 | 267 | 0.438 | 1.646% | **0.069** |  | 0.087 | 0.070 |  | **0.069** | 0.097 |  | 0.095 | 0.098% |
| sponge | 45 | 76 | 17.088 | 30.943% | 0.310 |  | **0.300** | 0.337 |  | 0.342 | 0.326 |  | 0.321 | 1.304% |
| lung cancer | 57 | 32 | 1.137 | 22.128% | 0.472 |  | **0.441** | 0.490 |  | 0.465 | 0.501 |  | 0.446 | 2.159% |
| splice | 61 | 3190 | 3.742 | 2.703% | 0.167 |  | **0.158** | 0.171 |  | 0.160 | 0.200 |  | 0.190 | 0.123% |

Table 18: Running time and score error ratio for A\*, AWA\*, BFBnB with AP/HP/AP-HP and NOTEARS on datasets with more than 100 nodes.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | NOTEARS | | A\* | | | AWA\* | | | BFBnB | | | Error |
| time(s) | Error | AP | HP | AP-HP | AP | HP | AP-HP | AP | HP | AP-HP |
| pathfinder | 109 | 5000 | 462.839 | - | 48.897 |  | **46.700** | 51.921 |  | 47.933 | 48.340 |  | 48.819 | 42.364% |
| synthetic 1 | 120 | 10000 | 11.544 | 12.392% | **10.051** |  | 10.055 | 11.380 |  | 10.249 |  |  |  | 21.298% |
| synthetic 2 | 150 | 10000 | 14.737 | 13.191% |  |  |  |  |  |  |  |  |  | 20.188% |
| synthetic 3 | 175 | 10000 | 22.670 | 10.862% | 16.943 |  | **16.608** | 20.979 |  | 16.667 | 18.053 |  | 17.454 | 30.977% |
| andes | 223 | 5000 | 32.073 | 22.899% | 19.735 |  | **19.039** | 22.677 |  | 19.633 | 19.142 |  | 19.226 | 48.135% |

### 5.2.4 Comparisons with DAG-GNN

This section compares A\*, AWA\*, and BFBnB under AP/HP/AP-HP constraints with DAG-GNN on benchmark BNs, UCI datasets, and datasets with more than 100 nodes, as shown in Tables 19–21. DAG-GNN uses a deep learning method, which has a long training time; therefore, unlike other sections, hours were used as the time measurement unit. We set 8 hours as the maximum time limit. The bold numbers indicate the shortest running times of the four algorithms. In addition, the score error ratios for DAG-GNN are listed in the 5th column, and the score error ratios for A\*, AWA\*, and BFBnB with AP/AP-HP are listed in the last column.

The running time of DAG-GNN increased with an increase in network size and sample size. Because of the need to train the neural network, DAG-GNN required a longer running time than GOBNILP, MMHC, NOTEARS, and other algorithms. Therefore, A\*, AWA\*, and BFBnB with AP and/or HP always had shorter running times, as shown in Table 19 –21.

In terms of accuracy, in general, DAG-GNN often obtained higher accuracy with a larger sample size, which was similar to MMHC but the opposite of NOTEARS. Generally, the larger the training sample size of the deep learning model, the better the results. However, some cases did not fit this trend. We believe that these cases were not well trained. Limited by workload and time, we cannot set the parameters of all models to optimal values based on our experiments. Overall, the DAG-GNN obtained good results for a large sample size. However, within 100 nodes, A\*, AWA\*, and BFBnB with the AP/AP-HP constraints achieved higher accuracy than DAG-GNN. These algorithms achieved lower accuracy with more than 100 nodes, and the reasons have been discussed previously. Under these circumstances, Algorithm 1 built AP constraints when , and these AP constraints included many single nodes. AP constraints built with a small number of PPSs were inaccurate, in other words, these single nodes built incorrect orderings, resulting in a significant loss of accuracy. Similar to Table 18, in pathfinder, DAG-GNN cannot calculate the score due to running out of memory.

The reasons for algorithms with the AP and/or HP not outperform DAG-GNN in term of running time in boerlage92 have been discussed in Section 5.2.1.

In summary, in most of the cases, A\*, AWA\*, and BFBnB under AP/AP-HP had a lower running time and higher accuracy than DAG-GNN within 100 nodes.

Table 19: Running time and score error ratio for A\*, AWA\*, BFBnB with AP/HP/AP-HP and DAG-GNN on benchmark BNs.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | DAG-GNN | | A\* | | | AWA\* | | | BFBnB | | | Error |
| time(h) | Error | AP | HP | AP-HP | AP | HP | AP-HP | AP | HP | AP-HP |
| sachs | 11 | 500 | 0.369 | 50.647% | **0.001** | **0.001** | **0.001** | **0.001** | **0.001** | **0.001** | 0.003 | 0.004 | 0.004 | 0% |
| sachs | 11 | 1000 | 0.376 | 10.579% | **0.001** | **0.001** | **0.001** | **0.001** | **0.001** | **0.001** | 0.004 | 0.004 | 0.004 | 0% |
| sachs | 11 | 5000 | 0.588 | 4.231% | **0.002** | **0.002** | **0.002** | **0.002** | **0.002** | **0.002** | 0.005 | 0.005 | 0.005 | 0% |
| sachs | 11 | 10000 | 0.712 | 8.561% | **0.003** | **0.003** | **0.003** | **0.003** | **0.003** | **0.003** | 0.005 | 0.005 | 0.005 | 0% |
| sachs | 11 | 15000 | 0.998 | 10.186% | **0.003** | **0.003** | **0.003** | **0.003** | **0.003** | **0.003** | 0.005 | 0.005 | 0.005 | 0% |
| child | 20 | 500 | 0.378 | 65.361% | 0.040 | 0.023 | **0.022** | 0.045 | 0.026 | 0.027 | 0.029 | 0.035 | 0.034 | 0% |
| child | 20 | 1000 | 0.566 | 22.663% | 0.073 | 0.025 | **0.024** | 0.074 | 0.029 | 0.029 | 0.045 | 0.038 | 0.038 | 0% |
| child | 20 | 5000 | 0.658 | 25.064% | 0.076 | 0.066 | **0.062** | 0.405 | 0.070 | 0.068 | 0.168 | 0.102 | 0.105 | 0% |
| child | 20 | 10000 | 0.958 | 23.716% | 0.707 | 0.134 | **0.104** | 0.792 | 0.149 | 0.117 | 0.347 | 0.120 | 0.137 | 0% |
| child | 20 | 15000 | 1.385 | 23.032% | 0.914 | 0.227 | **0.173** | 1.050 | 0.249 | 0.212 | 0.499 | 0.221 | 0.207 | 0% |
| boerlage92 | 23 | 500 | 0.314 | 12.039% | 4.418 | 0.029 | **0.025** | 5.687 | 0.027 | 0.026 | 1.394 | 0.033 | 0.034 | 0% |
| boerlage92 | 23 | 1000 | 0.574 | 14.363% | 0.443 | 0.173 | **0.010** | 0.450 | 0.207 | **0.010** | 0.185 | 0.078 | 0.015 | 0% |
| boerlage92 | 23 | 5000 | 0.854 | 12.439% | 8.820 | 22.769 | 9.928 | 24.236 | 45.366 | 19.229 | 3.336 | 8.593 | **3.142** | 0% |
| boerlage92 | 23 | 10000 | 0.939 | 9.578% | 9.142 | 26.324 | 11.882 | 23.560 | 55.968 | 22.662 | **3.466** | 9.832 | 4.485 | 0% |
| boerlage92 | 23 | 15000 | 1.221 | 6.367% | 9.367 | 29.501 | 13.014 | 26.969 | 50.801 | 26.449 | 3.564 | 10.628 | **3.434** | 0% |
| insurance | 27 | 500 | 0.373 | 5.853% | 61.519 | 0.640 | 0.136 | 109.414 | 0.633 | 0.729 | 28.597 | 0.213 | **0.067** | 0% |
| insurance | 27 | 1000 | 0.725 | 11.333% | 146.363 | 2.928 | 1.404 | 296.259 | 4.885 | 2.316 | 73.064 | 0.761 | **0.392** | 0% |
| insurance | 27 | 5000 | 0.748 | 26.882% | 143.290 | 3.091 | 1.432 | 269.710 | 3.653 | 1.843 | 67.418 | 0.792 | **0.399** | 0% |
| insurance | 27 | 10000 | 1.336 | 29.565% | 65.748 | 10.033 | 1.886 | 101.347 | 10.309 | 1.828 | 28.314 | 9.367 | **1.809** | 0% |
| insurance | 27 | 15000 | 1.703 | 20.939% | 147.272 | 1.974 | 1.845 | 264.387 | 1.988 | 1.866 | 68.937 | **1.831** | 1.848 | 0% |
| water | 32 | 500 | 0.437 | 43.838% | 1.709 | 11.974 | **0.019** | 2.156 | 16.152 | 0.020 | 0.566 | 17.034 | 0.037 | 0.048% |
| water | 32 | 1000 | 0.871 | 29.075% | 2.319 | 31.442 | **0.026** | 3.498 | 41.585 | 0.028 | 0.828 | 8.669 | 0.037 | 0.008% |
| water | 32 | 5000 | 0.983 | 10.070% | 2.812 | 6.968 | **0.025** | 3.943 | 8.628 | 0.025 | 1.020 | 1.390 | 0.033 | 0.003% |
| water | 32 | 10000 | 1.392 | 5.552% | 0.184 | 7.428 | **0.033** | 0.218 | 9.856 | 0.034 | 0.192 | 1.634 | 0.034 | 0.001% |
| water | 32 | 15000 | 1.749 | 3.194% | 2.802 | 28.802 | **0.033** | 4.357 | 34.204 | 0.035 | 1.009 | 7.143 | 0.052 | 0.004% |
| alarm | 37 | 500 | 0.512 | 14.186% | **0.107** |  | 0.164 | 0.109 |  | 0.108 | 0.127 |  | 0.120 | 0.401% |
| alarm | 37 | 1000 | 0.940 | 16.312% | 0.167 |  | 0.091 | 0.200 |  | 0.097 | 0.117 |  | **0.085** | 0.062% |
| alarm | 37 | 5000 | 0.696 | 15.112% | 0.226 |  | 0.296 | 0.270 |  | 0.216 | **0.166** |  | 0.283 | 0.021% |
| alarm | 37 | 10000 | 2.082 | 23.206% | 0.256 |  | 0.155 | 0.325 |  | 0.163 | 0.184 |  | **0.153** | 0.010% |
| alarm | 37 | 15000 | 2.695 | 12.651% | 0.307 |  | 0.161 | 0.298 |  | 0.169 | 0.194 |  | **0.144** | 0.005% |
| hailfinder | 56 | 500 | 0.767 | 51.132% | **0.066** |  | 0.098 | 0.067 |  | 0.070 | 0.109 |  | 0.091 | 0.013% |
| hailfinder | 56 | 1000 | 0.799 | 14.845% | **0.064** |  | 0.084 | 0.064 |  | 0.077 | 0.070 |  | 0.088 | 0.023% |
| hailfinder | 56 | 5000 | 1.234 | 37.674% | 1.564 |  | 0.231 | 2.901 |  | 1.816 | 0.638 |  | **0.215** | 0.025% |
| hailfinder | 56 | 10000 | 1.838 | 16.196% | 0.291 |  | 0.356 | **0.285** |  | 0.290 | 0.344 |  | 0.335 | 0.067% |
| hailfinder | 56 | 15000 | 4.655 | 15.549% | 1.662 |  | 0.469 | 2.890 |  | 1.582 | 0.783 |  | **0.404** | 0.043% |
| win95pt | 76 | 500 | 0.661 | 19.339% | **1.469** |  | 1.492 | 2.571 |  | 1.505 | 2.190 |  | 1.590 | 0.653% |
| win95pt | 76 | 1000 | 1.224 | 23.342% | 2.071 |  | **2.059** | 3.588 |  | 2.076 | 2.425 |  | 2.282 | 2.830% |
| win95pt | 76 | 5000 | 1.944 | 18.905% | 3.359 |  | **3.350** | 4.330 |  | 3.360 | 3.759 |  | 3.597 | 2.881% |
| win95pt | 76 | 10000 | 2.754 | 21.610% | 3.979 |  | **3.972** | 5.361 |  | 4.004 | 4.837 |  | 4.225 | 2.706% |
| win95pt | 76 | 15000 | OT |  | 3.703 |  | **3.514** | 4.739 |  | 4.567 | 4.121 |  | 3.831 | 2.880% |

Table 20: Running time and score error ratio for A\*, AWA\*, BFBnB with AP/HP/AP-HP and DAG-GNN on UCI datasets.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | DAG-GNN | | A\* | | | AWA\* | | | BFBnB | | | Error |
| time(h) | Error | AP | HP | AP-HP | AP | HP | AP-HP | AP | HP | AP-HP |
| zoo | 17 | 101 | 0.267 | 22.819% | 0.022 | 0.025 | 0.031 | 0.048 | 0.028 | 0.034 | 0.027 | 0.029 | 0.034 | 0% |
| voting | 17 | 435 | 0.312 | 20.580% | 0.010 | 0.024 | 0.027 | 0.010 | 0.022 | 0.030 | 0.015 | 0.027 | 0.033 | 0% |
| statlog | 19 | 752 | 0.492 | 15.547% | 0.473 | 0.100 | 0.134 | 0.278 | 0.108 | 0.146 | 0.359 | 0.155 | 0.164 | 0% |
| hepatitis | 20 | 126 | 0.238 | 10.567% | 0.021 | 0.022 | 0.025 | 0.021 | 0.024 | 0.027 | 0.034 | 0.033 | 0.037 | 0% |
| segment | 20 | 2310 | 1.176 | 12.485% | 0.441 | 0.054 | 0.099 | 0.663 | 0.057 | 0.103 | 0.276 | 0.093 | 0.116 | 0% |
| imports | 22 | 205 | 0.138 | 18.991% | 5.237 | 0.563 | 0.291 | 6.731 | 0.797 | 0.365 | 1.829 | 0.324 | 0.215 | 0% |
| meta | 22 | 528 |  |  | 3.318 | 7.667 | 7.880 | 3.668 | 7.665 | 7.662 | 3.205 | 7.272 | 7.644 | 0% |
| heart | 23 | 212 | 0.453 | 2.293% | 10.877 | 0.531 | 0.240 | 15.584 | 0.635 | 0.306 | 3.414 | 0.242 | 0.115 | 0% |
| horse | 23 | 300 | 0.497 | 8.351% | 3.727 | 1.877 | 0.391 | 6.063 | 2.944 | 0.495 | 1.298 | 0.691 | 0.199 | 0% |
| mushroom | 23 | 8124 | 4.809 | 83.305% | 0.410 | 10.912 | 10.644 | 0.557 | 11.668 | 11.767 | 0.366 | 11.764 | 11.730 | 0% |
| autos | 26 | 159 | 0.195 | 17.688% | 23.824 | 9.790 | 5.251 | 70.130 | 13.251 | 7.024 | 6.969 | 3.867 | 2.184 | 0% |
| steel | 28 | 1941 | 0.521 | 12.715% | 147.794 | 0.999 | 0.365 | 286.447 | 4.959 | 1.157 | 62.711 | 0.346 | 0.179 | 0% |
| flag | 29 | 194 | 0.287 | 4.853% | 14.822 | 78.112 | 21.965 | 18.649 | 96.354 | 28.707 | 4.024 | 28.055 | 7.369 | 0.004% |
| soybean | 36 | 266 | 0.584 | 15.651% | 0.398 |  | 0.393 | 0.430 |  | 0.398 | 0.416 |  | 0.421 | 0.487% |
| bands | 39 | 277 | 0.768 | 5.987% | 0.120 |  | 0.133 | 0.149 |  | 0.118 | 0.326 |  | 0.138 | 0.194% |
| spectf | 45 | 267 | 0.448 | 2.231% | 0.069 |  | 0.087 | 0.070 |  | 0.069 | 0.097 |  | 0.095 | 0.098% |
| sponge | 45 | 76 | 0.264 | 22.216% | 0.310 |  | 0.300 | 0.337 |  | 0.342 | 0.326 |  | 0.321 | 1.304% |
| lung cancer | 57 | 32 | 0.200 | 20.679% | 0.472 |  | 0.441 | 0.490 |  | 0.465 | 0.501 |  | 0.446 | 2.159% |
| splice | 61 | 3190 | 1.301 | 3.278% | 0.167 |  | 0.158 | 0.171 |  | 0.160 | 0.200 |  | 0.190 | 0.123% |

Table 21: Running time and score error ratio for A\*, AWA\*, BFBnB with AP/HP/AP-HP and DAG-GNN on datasets with more than 100 nodes.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | n | N | DAG-GNN | | A\* | | | AWA\* | | | BFBnB | | | Error |
| time(h) | Error | AP | HP | AP-HP | AP | HP | AP-HP | AP | HP | AP-HP |
| pathfinder | 109 | 5000 | 2.271 | - | 48.897 |  | **46.700** | 51.921 |  | 47.933 | 48.340 |  | 48.819 | 42.364% |
| synthetic 1 | 120 | 10000 | 3.225 | 9.366% | **10.051** |  | 10.055 | 11.380 |  | 10.249 | 11.860 |  | 11.809 | 21.298% |
| synthetic 2 | 150 | 10000 | 4.438 | 10.421% | 16.269 |  | **16.096** | 21.111 |  | 16.486 | 18.493 |  | 18.738 | 20.188% |
| synthetic 3 | 175 | 10000 | 7.822 | 9.248% | 16.943 |  | **16.608** | 20.979 |  | 16.667 | 18.053 |  | 17.454 | 30.977% |
| andes | 223 | 5000 | OT |  | 19.735 |  | **19.039** | 22.677 |  | 19.633 | 19.142 |  | 19.226 | 48.135% |

Overall, based on the comparisons in 5.2.1 and 5.2.4, the following summary can be presented as a conclusion. Generally, in terms of running time, the algorithms can be ranked with the best coming first, as algorithms with AP-HP > algorithms with AP > NOTEARS > MMHC > algorithms with HP > GOBNILP > DAG-GNN. In terms of accuracy, the performance ranking of the algorithms with the best results GOBNILP = algorithms with HP > algorithms with AP/AP-HP > MMHC > DAG-GNN > NOTEARS, within datasets of 100 nodes. AP constraints could not perform well on large networks because fewer PPSs were used to build the directed graph to obtain AP constraints. Fewer PPSs contain less information about the actual BN, and the obtained AP constraints deviate from the correct constraints, resulting in a large loss of accuracy. If more PPSs were used to calculate AP constraints, the partition block size of AP would have been too large to be searched by A\*, AWA\*, and BFBnB. Consequently, on larger networks, it is either difficult to search or the accuracy is reduced.

NOTEARS and DAG-GNN can learn a BN structure from the perspective of continuous optimization; therefore, they are better at handling continuous data. However, in this study, A\*, AWA\*, BFBnB, GOBNILP, MMHC, and our proposed algorithms work only with discrete data, and they cannot be compared with NOTEARS and DAG-GNN on continuous data. We believe that NOTEARS and DAG-GNN can achieve good results with continuous data.

## 5.3 Parameters analysis for calculating ancestral partition

According to Algorithm 1,  is controlled by  and , which are adjusted by the user. In this section, the performance of Algorithm 1 under various parameters is analyzed.

According to the analysis in Section 5.1, A\*, AWA\*, and BFBnB have similar trends under the constraints of AP/HP/AP-HP, and the performance of A\* is the most balanced among the three; therefore, A\* will be used in the following experiments in Sections 5.3 and 5.4.

The following criterion is introduced for the results of AP or HP.

**Partition block size set (PBS)**: The set of partition blocks of all sizes. In other words, the set recorded all  for AP or all  for HP.

For example, the PBS for  is {2, 3, 3}. It is unrealistic to adjust the parameters of Algorithms 1 and 2 in all datasets in Section 5.1. Therefore, the insurance\_10000 (insurance with 10000 samples), flag, and water\_10000 (water with 10000 samples) datasets are chosen as examples to test the performance of various parameters. To eliminate the influence of HP, the default simple partition was implemented to create static pattern databases in this section. Once a parameter changed within a specific range, the other parameters used the default values, as described in Section 5.1.

Table 22 shows the results of AP and the performance of A\* with AP under various  on the insurance\_10000, flag, and water\_10000 datasets.  changed between the intervals [n-12, n-1]. Multiple  parameters may share the same AP, and hence, the corresponding A\* had the same number of expanded states, the same score error ratio, and the similar running time. As  increased, the number of partitions for AP decreased (or the partition block size increased), the running time and the number of expanded states for the A\* algorithm increased, and score error ratio for the A\* algorithm decreased.

Table 23 shows the results of AP and the performance for A\* with AP under various  on the insurance\_10000, flag, and water\_10000 datasets.  changed between intervals [1, 10]. The experimental results were similar to those presented in Table 22. Different values of the parameter  may share the same AP; therefore, the corresponding A\* had the same number of expanded states, the same score error ratio, and the similar running time. As  increased, the number of partitions for AP decreased (or the partition block size increased), running time and the number of expanded states for the A\* algorithm increased, and score error ratio for the A\* algorithm decreased.

Table 22: Results of AP and the performance of A\* with AP under various  on the insurance\_10000, flag and water\_10000 datasets.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Name |  | AP Time | PBS | A\* Time | Exp | Error |
| insurance\_10000 | 15 | 0.002 | {8,5,1,1,7,1,3,1} | 0.091 | 143 | 0.177% |
| 16 | 0.002 | {8,5,1,1,7,1,3,1} | 0.082 | 143 | 0.177% |
| 17 | 0.002 | {8,5,1,1,7,1,3,1} | 0.080 | 143 | 0.177% |
| 18 | 0.002 | {8,5,1,1,7,1,3,1} | 0.081 | 143 | 0.177% |
| 19 | 0.002 | {8,5,1,1,7,1,3,1} | 0.081 | 143 | 0.177% |
| 20 | 0.002 | {8,5,1,1,7,1,3,1} | 0.081 | 143 | 0.177% |
| 21 | 0.002 | {8,5,1,1,7,1,3,1} | 0.086 | 143 | 0.177% |
| 22 | 0.002 | {8,5,1,1,7,1,3,1} | 0.082 | 143 | 0.177% |
| 23 | 0.002 | {23,1,1,1,1} | 12.054 | 1760623 | 0% |
| 24 | 0.002 | {24,1,1,1} | 25.916 | 3348330 | 0% |
| 25 | 0.002 | {25,1,1} | 64.855 | 7986905 | 0% |
| 26 | 0.002 | {25,1,1} | 65.285 | 7986905 | 0% |
| flag | 17 | 0.001 | {17,1,1,1,1,1,2,3,1,1} | 0.073 | 7862 | 0.044% |
| 18 | 0.001 | {17,1,1,1,1,1,2,3,1,1} | 0.072 | 7862 | 0.044% |
| 19 | 0.001 | {17,1,1,1,1,1,2,3,1,1} | 0.072 | 7862 | 0.044% |
| 20 | 0.001 | {17,1,1,1,1,1,2,3,1,1} | 0.071 | 7862 | 0.044% |
| 21 | 0.001 | {21,1,1,1,1,1,2,1} | 1.501 | 259626 | 0.044% |
| 22 | 0.001 | {21,1,1,1,1,1,2,1} | 1.481 | 259626 | 0.044% |
| 23 | 0.001 | {21,1,1,1,1,1,2,1} | 1.505 | 259626 | 0.044% |
| 24 | 0.001 | {21,1,1,1,1,1,2,1} | 1.475 | 259626 | 0.044% |
| 25 | 0.001 | {25,1,2,1} | 3.907 | 418550 | 0.010% |
| 26 | 0.001 | {25,1,2,1} | 3.958 | 418550 | 0.010% |
| 27 | 0.001 | {27,1,1} | 14.139 | 1196267 | 0.004% |
| 28 | 0.001 | {27,1,1} | 14.227 | 1196267 | 0.004% |
| water\_10000 | 20 | 0.001 | {6,17,3,3,3} | 0.197 | 90357 | 0.001% |
| 21 | 0.001 | {6,17,3,3,3} | 0.190 | 90357 | 0.001% |
| 22 | 0.001 | {6,17,3,3,3} | 0.202 | 90357 | 0.001% |
| 23 | 0.001 | {6,17,3,3,3} | 0.196 | 90357 | 0.001% |
| 24 | 0.001 | {6,17,3,3,3} | 0.196 | 90357 | 0.001% |
| 25 | 0.001 | {6,17,3,3,3} | 0.193 | 90357 | 0.001% |
| 26 | 0.001 | {6,26} | 421.886 | 55283373 | 0.001% |
| 27 | 0.001 | {6,26} | 421.363 | 55283373 | 0.001% |
| 28 | 0.001 | {6,26} | 422.249 | 55283373 | 0.001% |
| 29 | 0.001 | {6,26} | 421.474 | 55283373 | 0.001% |
| 30 | 0.001 | {6,26} | 421.665 | 55283373 | 0.001% |
| 31 | 0.001 | {6,26} | 423.772 | 55283373 | 0.001% |

Table 23: Results of AP and the performance of A\* with AP under various  on the insurance\_10000, flag and water\_10000 datasets.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Name |  | AP Time | PBS | A\* Time | Exp | Error |
| insurance\_10000 | 1 | 0.002 | {4,2,2,1,2,1,2,1,6,1,1,3,1} | 0.134 | 58 | 0.385% |
| 2 | 0.002 | {8,5,1,1,7,1,3,1} | 0.081 | 143 | 0.177% |
| 3 | 0.002 | {23,1,1,1,1} | 12.177 | 1760623 | 0% |
| 4 | 0.002 | {23,1,1,1,1} | 12.170 | 1760623 | 0% |
| 5 | 0.002 | {23,1,1,1,1} | 12.170 | 1760623 | 0% |
| 6 | 0.002 | {24,1,1,1} | 26.187 | 3348330 | 0% |
| 7 | 0.002 | {25,1,1} | 64.651 | 7986905 | 0% |
| 8 | 0.002 | {25,1,1} | 64.013 | 7986905 | 0% |
| 9 | 0.002 | {25,1,1} | 63.697 | 7986905 | 0% |
| 10 | 0.002 | {25,1,1} | 64.356 | 7986905 | 0% |
| flag | 1 | 0.001 | {17,1,1,1,1,1,2,3,1,1} | 0.109 | 7592 | 0.044% |
| 2 | 0.001 | {21,1,1,1,1,1,2,1} | 1.520 | 259626 | 0.010% |
| 3 | 0.001 | {25,1,2,1} | 3.971 | 418550 | 0.010% |
| 4 | 0.001 | {27,1,1} | 14.393 | 1196267 | 0.004% |
| 5 | 0.001 | {27,1,1} | 14.268 | 1196267 | 0.004% |
| 6 | 0.001 | {27,1,1} | 14.236 | 1196267 | 0.004% |
| 7 | 0.001 | {27,1,1} | 14.418 | 1196267 | 0.004% |
| 8 | 0.001 | {27,1,1} | 14.301 | 1196267 | 0.004% |
| 9 | 0.001 | {27,1,1} | 14.270 | 1196267 | 0.004% |
| 10 | 0.001 | {27,1,1} | 14.468 | 1196267 | 0.004% |
| water\_10000 | 1 | 0.001 | {4,1,1,7,6,3,3,3,4} | 0.038 | 117 | 0.019% |
| 2 | 0.001 | {6,11,3,6,3,3} | 0.080 | 1152 | 0.003% |
| 3 | 0.001 | {6,17,3,3,3} | 0.182 | 90357 | 0.001% |
| 4 | 0.001 | {6,17,3,3,3} | 0.192 | 90357 | 0.001% |
| 5 | 0.001 | {6,17,3,3,3} | 0.183 | 90357 | 0.001% |
| 6 | 0.001 | {6,17,3,3,3} | 0.189 | 90357 | 0.001% |
| 7 | 0.001 | {6,17,3,3,3} | 0.190 | 90357 | 0.001% |
| 8 | 0.001 | {6,17,3,3,3} | 0.182 | 90357 | 0.001% |
| 9 | 0.001 | {6,17,3,3,3} | 0.189 | 90357 | 0.001% |
| 10 | 0.001 | {6,17,3,3,3} | 0.184 | 90357 | 0.001% |

From the analysis of Algorithm 1 in Section 3.3, as  gradually decreases from a constant value  to 1, the edges of the directed graph also become sparse, and it is easier to extract SCCs. Therefore, a smaller  means easier extraction of SCCs, and concurrently, the size of the SCCs will also be reduced. Therefore, Tables 22 and 23 have shown similar trends. When  decreased ( decreased), the size of the SCCs also decreased (the partition block size decreased), indicating that the AP was closer to the ordering, which reflected stronger constraints in the order graph. Stronger constraints mean more pruning of the order graph, resulting in fewer states that need to be expanded and a lower running time. However, a decreasing  indicated that fewer PPSs were used to build the directed graph, extracted SCCs contained less information about the original BN, and the obtained AP was not consistent with the actual network structure. Therefore, the accuracy decreased as  decreased ( decreased). This also explains why the accuracy of these algorithms under AP/AP-HP decreased on datasets with over 100 nodes. The harmful impact of these factors can be further enhanced as the network size increases.

## 5.4 Parameters analysis for calculating heuristic partition

According to Algorithm 2,  is controlled by ,  and , which are adjusted by the user. In this section, the performance of Algorithm 2 is examined under various parameters.

Similar to Section 5.3, we used A\* with a heuristic partition under various parameters on the insurance\_10000, flag, and water\_10000 datasets. No AP constraints were added to eliminate the influence of the AP. Once a parameter changed within a specific range, the other parameters used the default values, as described in Section 5.1. The HP does not affect the final accuracy of the exact learning algorithms based on previous theories and experiments. Inspired by Algorithm 2, Line 8, and Eq. (11), the following criteria were used to evaluate the tightness of the HP.

**Score difference ratio (Diff)**: This criterion is calculated using Eq. (12). For a heuristic partition plan, static pattern databases are created to calculate .  is calculated using an approximate algorithm for BN structure learning. Note that a lower Diff value indicates a tighter heuristic partition.

|  |  |  |
| --- | --- | --- |
|  |  | （12） |

In Algorithm 2, the hill-climbing algorithm with restarts was used to calculate .

Table 24 shows the results of HP and the performance of A\* with HP under various  on the insurance\_10000, flag, and water\_10000 datasets. The  changed between the intervals [n-12, n-1]. Multiple  parameters may share the same HP. In general, as  increased, the running time for calculating HP increased, size of the maximum partition block increased, score difference ratio for HP decreased, and number of expanded states for A\* also decreased. However, unlike the AP, the change in running time for A\* was not consistent with the change in the number of expanded states (see the data in the flag dataset). A similar phenomenon was discussed in Section 5.1, that is, fewer states do not always mean shorter running time. When  in the flag, a smaller number of expanded states indicates that A\* searched fewer states in the order graph, resulting in reduced running time. When  in the flag, although fewer expanded states reduced the running time, more running time for A\* was spent creating the static pattern database for the largest partition block. There is also an evident rise in running time for calculating HP for the same reason. The results for the insurance\_10000 and water\_10000 datasets were similar to those when  in the flag dataset.

Table 25 shows the results of HP and the performance of A\* with HP under various  on the insurance\_10000, flag, and water\_10000 datasets. As  increased when  in the insurance\_10000 and  in the water\_10000 datasets, the running time for calculating HP increased, size of the maximum partition block increased, score difference ratio for HP decreased, and the number of expanded states for the A\* algorithm also decreased. In other cases, the results remain unchanged with . In fact, from Table 25, the results were not sensitive to . There were apparent changes only when  is smaller under the AP for the insurance\_10000, flag, and water\_10000 datasets. If Algorithm 2 did not add merging partitions of small SCCs (Algorithm 2, lines 23–43), the results in Table 25 would have been similar to those in Table 23.

Table 26 shows the results of HP and the performance of A\* with HP under various  on the insurance\_10000, flag, and water\_10000 datasets. As  decreased, both the score difference ratio for HP and the number of expanded states for the A\* algorithm decreased, indicating that heuristic functions with better tightness can help algorithms search fewer states. The changing trend of the running time of calculating HP and the running time of A\* in Table 26 are similar to those in Table 24, and the reasons are discussed in Section 5.1.

Table 24: Results of HP and the performance of A\* with HP under various  on the insurance\_10000, flag and water\_10000 datasets.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Name |  | HP Time | PBS | Diff | A\* Time | Exp |
| insurance\_10000 | 15 | 1.473 | {22,5} | 1.863% | 10.028 | 30146 |
| 16 | 1.457 | {22,5} | 1.863% | 9.943 | 30146 |
| 17 | 1.445 | {22,5} | 1.863% | 9.954 | 30146 |
| 18 | 1.441 | {22,5} | 1.863% | 10.158 | 30146 |
| 19 | 1.431 | {22,5} | 1.863% | 10.114 | 30146 |
| 20 | 1.422 | {22,5} | 1.863% | 10.037 | 30146 |
| 21 | 1.415 | {22,5} | 1.863% | 10.200 | 30146 |
| 22 | 1.407 | {22,5} | 1.863% | 10.102 | 30146 |
| 23 | 2.936 | {23,4} | 1.109% | 23.183 | 6350 |
| 24 | 6.186 | {24,3} | 0.823% | 50.964 | 3771 |
| 25 | 13.050 | {25,2} | 0.517% | 108.434 | 2786 |
| 26 | 27.156 | {26,1} | 0.268% | 240.706 | 44 |
| flag | 17 | 0.073 | {17,12} | 1.640% | 26.208 | 1953999 |
| 18 | 0.105 | {18,11} | 1.640% | 26.043 | 1953999 |
| 19 | 0.178 | {19,10} | 0.985% | 6.723 | 573760 |
| 20 | 0.332 | {20,9} | 0.793% | 6.058 | 461788 |
| 21 | 0.648 | {21,8} | 0.805% | 11.118 | 556434 |
| 22 | 1.287 | {22,7} | 0.805% | 15.685 | 553070 |
| 23 | 2.688 | {23,6} | 0.739% | 27.991 | 552978 |
| 24 | 5.525 | {24,5} | 0.719% | 54.125 | 552966 |
| 25 | 11.272 | {25,4} | 0.440% | 105.788 | 670 |
| 26 | 23.810 | {26,3} | 0.274% | 241.687 | 626 |
| 27 | 49.220 | {27,2} | 0.305% | OT |  |
| 28 | 104.708 | {28,1} | 0.179% | OT |  |
| water\_10000 | 20 | 0.355 | {6,20,6} | 0.085% | 5.779 | 338160 |
| 21 | 0.346 | {6,20,6} | 0.085% | 5.662 | 338160 |
| 22 | 0.352 | {6,20,6} | 0.085% | 5.691 | 338160 |
| 23 | 2.568 | {23,9} | 0.175% | 145.292 | 8486121 |
| 24 | 2.573 | {23,9} | 0.175% | 148.941 | 8486121 |
| 25 | 2.576 | {23,9} | 0.175% | 146.580 | 8486121 |
| 26 | 25.799 | {6,26} | 0.019% | 208.700 | 126 |
| 27 | 25.728 | {6,26} | 0.019% | 210.659 | 126 |
| 28 | 25.726 | {6,26} | 0.019% | 210.088 | 126 |
| 29 | 25.771 | {6,26} | 0.019% | 210.944 | 126 |
| 30 | 25.784 | {6,26} | 0.019% | 209.706 | 126 |
| 31 | 25.827 | {6,26} | 0.019% | 210.499 | 126 |

Table 25: Results of HP and the performance of A\* with HP under various  on the insurance\_10000, flag and water\_10000 datasets.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Name | m | HP Time | PBS | Diff | A\* Time | Exp |
| insurance\_10000 | 1 | 0.0721 | {21,6} | 3.814% | 6.577 | 241833 |
| 2 | 1.482 | {22,5} | 1.863% | 10.065 | 30146 |
| 3 | 1.463 | {22,5} | 1.863% | 10.093 | 30146 |
| 4 | 1.466 | {22,5} | 1.863% | 9.873 | 30146 |
| 5 | 1.459 | {22,5} | 1.863% | 9.925 | 30146 |
| 6 | 1.460 | {22,5} | 1.863% | 9.975 | 30146 |
| 7 | 1.462 | {22,5} | 1.863% | 10.015 | 30146 |
| 8 | 1.460 | {22,5} | 1.863% | 10.078 | 30146 |
| 9 | 1.462 | {22,5} | 1.863% | 10.065 | 30146 |
| 10 | 1.461 | {22,5} | 1.863% | 9.964 | 30146 |
| flag | 1 | 0.138 | {15,14} | 1.679% | 78.667 | 6236857 |
| 2 | 0.140 | {15,14} | 1.679% | 78.607 | 6236857 |
| 3 | 0.139 | {15,14} | 1.679% | 77.766 | 6236857 |
| 4 | 0.138 | {15,14} | 1.679% | 78.402 | 6236857 |
| 5 | 0.137 | {15,14} | 1.679% | 79.211 | 6236857 |
| 6 | 0.137 | {15,14} | 1.679% | 79.356 | 6236857 |
| 7 | 0.138 | {15,14} | 1.679% | 79.162 | 6236857 |
| 8 | 0.140 | {15,14} | 1.679% | 79.211 | 6236857 |
| 9 | 0.138 | {15,14} | 1.679% | 79.180 | 6236857 |
| 10 | 0.139 | {15,14} | 1.679% | 79.381 | 6236857 |
| water\_10000 | 1 | 0.172 | {19,13} | 0.438% | OT |  |
| 2 | 0.311 | {20,12} | 0.147% | 9.668 | 479439 |
| 3 | 0.347 | {6,20,6} | 0.085% | 5.810 | 338160 |
| 4 | 0.347 | {6,20,6} | 0.085% | 5.815 | 338160 |
| 5 | 0.347 | {6,20,6} | 0.085% | 5.802 | 338160 |
| 6 | 0.346 | {6,20,6} | 0.085% | 5.791 | 338160 |
| 7 | 0.349 | {6,20,6} | 0.085% | 5.675 | 338160 |
| 8 | 0.350 | {6,20,6} | 0.085% | 5.578 | 338160 |
| 9 | 0.345 | {6,20,6} | 0.085% | 5.573 | 338160 |
| 10 | 0.347 | {6,20,6} | 0.085% | 5.734 | 338160 |

Table 26: Results of HP and the performance of A\* with HP under various  on the insurance\_10000, flag and water\_10000 datasets.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Name | epsilon | HP Time | PBS | Diff | A\* Time | Exp |
| insurance\_10000 | 0.10 | 0.038 | {15,12} | 5.129% | 4.457 | 389224 |
| 0.05 | 1.476 | {22,5} | 1.863% | 10.181 | 30146 |
| 0.02 | 1.461 | {22,5} | 1.863% | 10.363 | 30146 |
| 0.01 | 10.532 | {24,3} | 0.823% | 51.008 | 3771 |
| 0.0075 | 23.559 | {25,2} | 0.517% | 108.395 | 2786 |
| 0.005 | 50.776 | {26,1} | 0.268% | 247.301 | 44 |
| flag | 0.10 | 0.141 | {15,14} | 1.679% | 79.749 | 6236857 |
| 0.05 | 0.138 | {15,14} | 1.679% | 78.914 | 6236857 |
| 0.02 | 0.137 | {15,14} | 1.679% | 80.715 | 6236857 |
| 0.01 | 0.372 | {16,13} | 0.984% | 2.205 | 189969 |
| 0.0075 | 5.406 | {23,6} | 0.739% | 28.689 | 552978 |
| 0.005 | 22.075 | {25,4} | 0.440% | 107.488 | 670 |
| water\_10000 | 0.10 | 0.347 | {6,20,6} | 0.085% | 5.983 | 338160 |
| 0.05 | 0.346 | {6,20,6} | 0.085% | 5.864 | 338160 |
| 0.02 | 0.348 | {6,20,6} | 0.085% | 6.035 | 338160 |
| 0.01 | 0.344 | {6,20,6} | 0.085% | 5.791 | 338160 |
| 0.001 | 0.348 | {6,20,6} | 0.085% | 5.816 | 338160 |
| 0.0005 | 31.162 | {6,26} | 0.019% | 210.619 | 126 |

# 6. Conclusion

This study proposed two partition constraints—ancestral and heuristic partition—to improve the efficiency of exact learning algorithms: ancestral partition and heuristic partition. The ancestral partition can prune the order graph by dividing the entire learning process into various stages. A heuristic partition can improve the tightness of a heuristic function. We also theoretically proved that algorithms with ancestral partition can still find the optimal score on the pruned order graph if the ancestral partition is consistent with the actual BN structure. Algorithms with heuristic partition also obtain the optimal solution because the heuristic function is admissible and consistent. Extensive experiments were performed to evaluate the performance of the proposed partition constraints on a variety of benchmark BN, UCI, and synthetic datasets. Experiments have illustrated that both ancestral partition and heuristic partition can significantly improve the efficiency and scalability of a series of exact learning algorithms than before, such as A\*, AWA\*, and BFBnB. The ancestral partition can help algorithms achieve optimal scores within smaller-scale BNs () and may cause a slight loss of accuracy for larger-scale BNs (). Worse results have been observed on a network with more than 100 nodes because fewer PPSs were used to build the ancestral partition constraints. Compared with other algorithms, such as GOBNILP, MMHC, NOTEARS, and DAG-GNN, in most of the cases, algorithms with AP\AP-HP can outperform other algorithms in terms of running time and have achieved a higher accuracy than other algorithms, except GOBNILP within 100 node datasets. Finally, an analysis of the parameters was performed to calculate the ancestral partition and heuristic partition, and the results can guide the user to adjust these parameters.

However, further research questions remain. The greedy strategy employed in Algorithm 1 leads to a slight loss of accuracy within a network of 100 nodes and a larger loss of accuracy for a network with more than 100 nodes. Overcoming this problem or reducing the score error ratio requires further study. We consider that, in future research, using more approximate PPSs rather than exact PPSs may be an appropriate solution. In addition, we would like to perform searches for various stages in parallel with the ancestral partition.

# Acknowledgment

This work was supported by the National Nature Science Foundation of China (No. 61573285).

# Reference

1. Yu J, Rui Y, Tao D. Click prediction for web image reranking using multimodal sparse coding[J]. IEEE Transactions on Image Processing, 2014, 23(5): 2019-2032.
2. Yu J, Tan M, Zhang H, Tao D, Rui Y. Hierarchical deep click feature prediction for fine-grained image recognition[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2019.
3. Young T, Hazarika D, Poria S, Cambria E. Recent trends in deep learning based natural language processing[J]. IEEE Computational Intelligence Magazine, 2018, 13(3): 55-75.
4. Chen C, Seff A, Kornhauser A, Xiao J. Deepdriving: Learning affordance for direct perception in autonomous driving[J]. In Proceedings of the IEEE International Conference on Computer Vision, 2015: 2722-2730.
5. Ai M, Xie Y, Tang Z, Zhang J, Gui W. Deep learning feature-based setpoint generation and optimal control for flotation processes[J]. Information Sciences, 2021, 578: 644-658.
6. Cai R, Zhang Z, Hao Z, Winslett M. Sophisticated merging over random partitions: a scalable and robust causal discovery approach[J]. IEEE Transactions on Neural Networks and Learning Systems, 2017, 29(8): 3623-3635.
7. Tien, Iris, Der, et al. Algorithms for Bayesian network modeling and reliability assessment of infrastructure systems[J]. Reliability Engineering & System Safety, 2016: 134-147.
8. Zhang H, Qian Z, Liu J, et al. Fault Detection and Repairing for Intelligent Connected Vehicles Based on Dynamic Bayesian Network Model[J]. IEEE Internet of Things Journal, 2018, 5(4): 2431-2440.
9. Wang R, Ye S, Li K, Kwong S. Bayesian network based label correlation analysis for multi-label classifier chain[J]. Information Sciences, 2021, 554: 256-275.
10. Chickering D M. Learning Bayesian Networks is NP-Complete[J]. Networks, 1996, 112(2):121-130.
11. Chickering D M, Heckerman D, Meek C. Large-sample learning of Bayesian networks is NP-hard[J]. Journal of Machine Learning Research, 2004, 5: 1287-1330.
12. Gheisari S, Meybodi MR. BNC-PSO: structure learning of Bayesian networks by particle swarm optimization[J]. Information Sciences, 2016, 348: 272-289.
13. Cooper G F, Herskovits E. A Bayesian method for the induction of probabilistic networks from data[J]. Machine Learning, 1992, 9(4):309-347.
14. JA Gámez, Mateo J L, Puerta J M. Learning Bayesian networks by hill climbing: efficient methods based on progressive restriction of the neighborhood[J]. Data Mining & Knowledge Discovery, 2011, 22(1-2):106-148.
15. Lee S, Kim S B. Parallel Simulated Annealing with a Greedy Algorithm for Bayesian Network Structure Learning[J]. IEEE Transactions on Knowledge and Data Engineering, 2019:1157-1166.
16. Teyssier M, Koller D. Ordering-Based Search: A Simple and Effective Algorithm for Learning Bayesian Networks[C]. Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence, 2005: 584-590.
17. Spirtes P, Glymour C. An Algorithm for Fast Recovery of Sparse Causal Graphs[J]. Social Science Computer Review, 1991.
18. Spirtes P, Glymour C, Scheines R. Causation, Prediction, and Search, 2nd Edition. The MIT Press, 2000.
19. D Margaritis, Thrun S. Bayesian Network Induction via Local Neighborhoods[J]. Advances in neural information processing systems, 2000, 12: 505-511.
20. Tsamardinos, Ioannis and Brown, Laura E and Aliferis, Constantin F. The max-min hill-climbing Bayesian network structure learning algorithm[J]. Machine Learning, 2006, 31-78.
21. Tsamardinos I, Aliferis C, Statnikov A. Time and sample efficient discovery of Markov blankets and direct causal relations[C]. Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003),673-678.
22. Zheng X, Aragam B, Ravikumar P, Xing EP. DAGs with NO TEARS: Continuous optimization for structure learning[C]. Proceedings of the 32nd International Conference on Neural Information Processing Systems, 2018: 9492–9503.
23. Yu Y, Chen J, Gao T, Yu M. DAG-GNN: DAG structure learning with Graph Neural Networks. Proceedings of the 36th International Conference on Machine Learning, 2019: 7154-7163.
24. Koivisto M, Sood K. Exact Bayesian structure discovery in Bayesian networks[J]. Journal of Machine Learning Research, 2004: 549-573.
25. Singh A, Moore A W. Finding optimal Bayesian networks by dynamic programming[R]. Carnegie Mellon University, 2005.
26. Silander T, Myllymaki P. A simple approach for finding the globally optimal Bayesian network structure[C]. Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence, 2006: 445-452.
27. Malone B, Yuan C. Memory-efficient dynamic programming for learning optimal Bayesian networks[C]. Proceedings of 25th AAAI Conference on Artificial Intelligence, 2011: 1057-1062.
28. Yuan C, Malone B, Wu X. Learning optimal Bayesian networks using A\* search[C]. Proceedings of the 22nd International Joint Conference on Artificial Intelligence, 2011: 2186–2191.
29. Yuan C, Malone B. Learning optimal Bayesian networks: a shortest path perspective[J]. Journal of Artificial Intelligence Research, 2013, 48(1): 23-65.
30. Malone B, Yuan C. Evaluating anytime algorithms for learning optimal Bayesian networks[C]. Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence, 2013: 381-390.
31. Malone, B, Yuan C, Hansen E, Bridges S. Improving the scalability of optimal Bayesian network learning with external-memory frontier breadth-first branch and bound search[C]. Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence, 2011: 479-488.
32. Yuan C, Malone B. An improved admissible heuristic for learning optimal Bayesian networks[C]. Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence, 2012: 924–933.
33. Tan X, Gao X, Wang Z, He C. Bidirectional heuristic search to find the optimal Bayesian network structure[J]. Neurocomputing, 2021, 426: 35-46.
34. Beek P V, Hoffmann H F. Machine Learning of Bayesian Networks Using Constraint Programming[C]. Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming, 2015: 429-445.
35. Bartlett M, Cussens J. Advances in Bayesian Network Learning using Integer Programming[C]. Proceedings of the Twenty-Nineth Conference on Uncertainty in Artificial Intelligence, 2013:182-191.
36. Bartlett M, Cussens J. Integer linear programming for the Bayesian network structure learning problem[J]. Artificial Intelligence, 2017, 244: 258-271.
37. Empirical hardness of finding optimal Bayesian network structures: algorithm selection and runtime prediction[J]. Machine Learning, 2018, 107: 247–283.
38. Scanagatta M, A Salmerón, Stella F. A survey on Bayesian network structure learning from data[J]. Progress in Artificial Intelligence, 2019, 8(4):425-439.
39. Behjati S, H Beigy. Improved K2 algorithm for Bayesian network structure learning[J]. Engineering Applications of Artificial Intelligence, 2020, 91(5):103617.1-103617.12.
40. Liu H, Zhou S, Lam W, et al. A new hybrid method for learning Bayesian networks: Separation and reunion[J]. Knowledge-Based Systems, 2017, 121(4): 185-197.
41. Zhao Y, Chen Y, Tu K, et al. Learning Bayesian network structures under incremental construction curricula [J]. Neurocomputing, 2017, 258(10): 30-40.
42. Yang Y, Gao X, Guo Z. Finding optimal Bayesian networks by a layered learning method[J]. Journal of Systems Engineering and Electronics, 2019(30): 946-958.
43. Li X, Gao X, Wang C. A Novel BN Learning Algorithm Based on Block Learning Strategy[J]. Sensors, 2020, 20(21):6357.
44. Dai J, Ren J, Du W. Decomposition-based Bayesian network structure learning algorithm using local topology information[J]. Knowledge-Based Systems, 2020, 195: 105602.
45. Schwarz G. Estimating the dimension of a model[J]. The Annals of Statistics, 1978,6 (2): 461-464.
46. D Heckerman, D Geiger, Chickering M. Learning Bayesian networks: The combination of knowledge and statistical data[J]. Machine learning,1995:197-243.
47. Rissanen J. Modeling by shortest data description[J]. Automatica, 1978, 14: 465–471.
48. Akaike H T. A new look at the statistical model identification[J]. IEEE Transactions on Automatic Control, 1974, 19(6):716-723.
49. Fan X, Yuan C, Malone B. Tightening Bounds for Bayesian Network Structure Learning[C]. Proceedings of the 28th AAAI Conference on Artificial Intelligence, 2014.
50. Boerlage B. Link strength in Bayesian networks. University of British Columbia, 1992.

1. \* Corresponding Author

   *E-mail addresses:* tanxy2017@mail.nwpu.edu.cn(Xiangyuan Tan), cxg2012@nwpu.edu.cn(Xiaoguang Gao), nwpu\_wzd@mail.nwpu.edu.cn(Zidong Wang), hanhao@mail.nwpu.edu.cn (Hao Han), Xiaohan Liu (lxhwwey@mail.nwpu.edu.

   cn), Daqing Chen (chend@lsbu.ac.uk)

   This work was supported by the National Nature Science Foundation of China (61573285). [↑](#footnote-ref-1)
2. A heuristic function  is said to be admissible if it never overestimates the cost of reaching the goal. [↑](#footnote-ref-2)
3. A heuristic function  is said to be consistent, or monotone, if its estimate is always less than or equal to the estimated distance from any neighboring state to the goal, plus the cost of reaching that neighbor. [↑](#footnote-ref-3)
4. A\*, AWA\* and BFBnB with source code can be download at https://github.com/bmmalone/urlearning-cpp. [↑](#footnote-ref-4)
5. GOBNILP with source code can be download at https://www.cs.york.ac.uk/aig/sw/gobnilp/. [↑](#footnote-ref-5)
6. NOTEARS with source code can be download at https://github.com/xunzheng/notears. [↑](#footnote-ref-6)
7. DAG-GNN with source code can be download at https://github.com/fishmoon1234/DAG-GNN. [↑](#footnote-ref-7)
8. https://archive.ics.uci.edu/ml/index.php. [↑](#footnote-ref-8)
9. https://www.bnlearn.com/bnrepository/. [↑](#footnote-ref-9)
10. If used datasets cannot be scored for A\* and GOBNILP, such as pathfinder and andes in the next experiment, we get their structures from https://www.bnlearn.com/bnrepository/. [↑](#footnote-ref-10)