KTOBS: An Approach of Bayesian Network Learning Based on K-tree Optimizing Ordering-Based Search

Qingwang Zhang, Sihang Liu, Ruihong Xu, Zemeng Yang, and Jianxiao Liu(✉)

College of Informatics, Hubei Key Laboratory of Agricultural Bioinformatics, Huazhong Agricultural University, Wuhan 430070, China

Abstract. How to construct Bayesian Networks (*BN*) efficiently and accurately is a research hotspot in the era of artificial intelligence. By limiting the tree-width of the network, the Bayesian network learning based on *k*-tree can be used to process large-scale of variables. However, this method has the problems of low accuracy, further to optimize the order of adding nodes, *etc*. In order to solve these problems, this work proposes a Bayesian learning method based on k-tree optimizing ordering-based search (*KTOBS*). Firstly, the local learning search strategy is adopted to obtain the candidate parent sets of each variable efficiently and accurately. Then it selects $k + 1$ nodes based on the obtained candidate parent node sets, and constructs the corresponding initial sub-network. Then the heuristic evaluation strategy is used to add subsequent nodes successively, and thus to get the initial network. Finally, it optimizes the network iteratively through switching nodes until the score of network no longer increases. The experimental results show that *KTOBS* can learn a network structure with higher accuracy than other *k*-tree algorithms in a given limited time.

Availability and implementation: codes and experiment dataset are available at: http://122.205.95.139/KTOBS/.

Keywords: Bayesian network · Candidate parent node · *k*-tree · Ordering-based search

1 Introduction

Bayesian network (*BN*), also known as causal networks or probabilistic networks, describes the relationship between variables through probability graph model. Bayesian network has the function of visualizing multiple knowledge diagrams, and has unique advantages in solving uncertainty and reasoning. Bayesian network is currently one of the most effective theoretical models in the field of uncertain knowledge expression and reasoning [1]. At present, Bayesian network has been applied in many fields, including medical diagnosis, language understanding, speech recognition, pattern recognition,

Q. Zhang and S. Liu—These authors contributed equally to this article.

H. Gao and X. Wang (Eds.): CollaborateCom 2021, LNICST 406, pp. 67–82, 2021.
https://doi.org/10.1007/978-3-030-92635-9_5

data mining, and artificial intelligence, etc. How to construct Bayesian network efficiently and accurately has attracted many scholars and become a research hotspot in the field of artificial intelligence.

Bayesian network structure learning is the process of finding a network to fit the sample dataset as much as possible. As the number of nodes increase, the complexity of Bayesian network learning increases exponentially. Bayesian network learning methods can be divided into three categories: constraint-based methods, scoring search-based methods and the hybrid methods. Constraint-based methods use statistics or information theory to quantitatively analyze the dependence relationship between variables firstly, and then find a network that is independent of these conditions and consistent with the dependence relationships. The representative constraint-based methods mainly include *grow*-shrink (*GS*), parents & children (*PC*) [2, 3], three-phase dependency analysis (*TPDA*) [4], incremental association Markov blanket (*IAMB*) [5], interleaved incremental association (*Inter-IAMB*), *etc*.

The scoring search-based method is a commonly used Bayesian network structure learning algorithm. The first step in this method is to determine the candidate parent sets for each variable. The second step is to select a parent set from the candidate parent sets of each variable, and then obtain the *DAG* graph of the network. The classical search scoring-based algorithms include *K2* [1], *K3* [6], hill climbing (*HC*) [7], tabu search (*Tabu*), etc. Gao and Ji combined the Markov Blanket and *BN* scoring function to improve the efficiency of existing score-based learning algorithms [8, 9]. In the view of modeling dependencies between groups of variables rather than between individual variables, Parviainen *et al*. used the groupwise faithfulness assumption and conditional independencies to learn the network between variables in different groups [10]. Later, Niinimäki *et al*. used metropolis-coupled Markov chain Monte Carlo and annealed importance method into sampling partial order to further enhance the *BN* learning efficiency [11]. From the aspect of structure constraint, Campos *et al*. used the decomposable properties of score functions to reduce the time and memory costs without losing global optimality guarantees [12, 13]. Regarding searching the Bayesian Network with the largest score as a constrained optimization problem, Bartlett used the integer linear programming (*ILP*) to get the global optimization Bayesian Network [14, 15].

Nie *et al*. proposed a sampling method to generate representative *k*-trees, and thus to learn a Bayesian Network with bounded tree-width [16, 17]. In 2016–2018, Mauro Scanagatta *et al*. proposed a *k*-tree optimizing Bayesian learning method of bounded tree width from the perspective of tree decomposition of graph structure [18–20]. However, this algorithm has the problems of low accuracy, needing to further optimizing the order of adding nodes, *etc*. In order to solve the above problems, this paper proposes a Bayesian network structure learning method based on *k*-tree optimizing ordering-based search (*KTOBS*).

- It uses dynamic programming and local learning methods to obtain the candidate parent node sets of each node efficiently and accurately. Then it selects $k + 1$ nodes and constructs initial corresponding sub-network on the basis of the obtained candidate parent sets.

- The heuristic evaluation strategy is used to add subsequent nodes successively, and thus to obtain the initial network. Then it exchanges the position of the adjacent node pairs in the initial ordering iteratively until the *BN* score of the corresponding network no longer increases.
- The classical datasets of *Alarm1*, *Alarm3*, *Alarm5*, *Insurance1*, *Insurance3*, *Insurance5*, are used for experimental verification. The experimental results show that the network constructed using *KTOBS* has the highest *BN* score. It can obtain a network structure with higher accuracy than other *k*-tree algorithms in given limited time.

2 Bayesian Network and k-tree

2.1 Bayesian Network

The scoring search-based method mainly includes two parts: scoring function and search strategy. The most commonly used scoring functions include *K2* (also known as *CH* score) [21], *BD* score [22], *BIC* (Bayesian Information Criterion) [23], *etc*. The *BIC* scoring function is an approximate calculation method of the marginal likelihood function within a large number of samples. This work mainly uses the *BIC* scoring function. The *BIC* scoring is decomposable, that is, the score of the entire network can be obtained by the sum of all the nodes. The *BIC* scoring function is shown in Eq. (1).

$$\begin{aligned} \mathrm{BIC}(G) &= \sum_{i=1}^{n} \mathrm{BIC}(X_i, \Pi_i) = \\ &\sum_{i=1}^{n} (\mathrm{LL}(X_i|\Pi_i) + Pen(X_i, \Pi_i)) = \\ &\sum_{i=1}^{n} \left(\sum\nolimits_{\pi \in |\Pi_i|, x \in |X_i|} N_{x,\pi} \widehat{\theta}_{x|\pi} - \frac{\log N}{2} (|Xi| - 1)(|\Pi_i|) \right) \end{aligned} \tag{1}$$

In Eq. (1), $\hat{\theta}_{x|\pi}$ represents the conditional probability $P(X_i = x \mid \prod_i = \pi)$ of the maximum likelihood estimation. $N_{x,\ \pi}$ represent the number of occurrences satisfying $(X = x \wedge \prod_i = \pi)$ in the dataset. $|X_i|$ represents the number of values that X_i can take and $|\prod_i|$ represents the product of the value number of the parent node of X_i.

2.2 Tree Width and k-tree

$H = (V, E)$ represents the undirected graph, V represents the node set and E represents the edge set. The tree decomposition of the undirected graph H is expressed as (C, T), in which $C = \{C_1, C_2, \ldots, C_m\}$ represents the subsets of V.

Tree Width: The tree width is defined as $\max(|C_i|)$-1, where $|C_i|$ represents the number of vertices in C_i. The tree width of H is the smallest width among all possible tree decompositions of G.

k-tree: If the undirected graph $T_k = (V, E)$ represents the largest graph whose tree width is k, then it is a k-tree, and adding any edge to T_k will increase its tree width. The definition of k-tree is shown as follows: for a $(k + 1)$-clique, it is a complete graph with $k + 1$ nodes, and $(k + 1)$-clique is a k-tree. A $(k + 1)$-clique can be decomposed into multiple k-cliques. Z represents a node that has not been included in V. The graph obtained by connecting Z to each node in the k-clique of T_k is also a k-tree.

3 BN Learning Based on K-tree Optimizing Ordering-Based Search

3.1 Obtaining Candidate Parent Set

We adopt the algorithms of local learning and dynamic programming to determine the candidate parent sets for all the nodes. It can help to enhance the calculation efficiency and improve the accuracy of the candidate parent sets. Supposing the node set $X = \{X_1, X_2, ..., X_n\}$ and the number of nodes is n, the process of obtaining candidate parent nodes for node X_i is shown in Algorithm 1.

```
Algorithm 1. Algorithm of getting CandidateParentSets
Input: Xi: variable, n: the number of variables
Output: Π: CandidateParentSet
1: P(Xi)←∅
2: for j←1, …, n
3:     if Xj!=Xi
4:         insert Xj into T(Xi)
5:     endif
6: endfor
7: insert Xi into P(Xi)
8: while (T(Xi)!=∅)
9:     Remove v from T(Xi)
10:    Insert v into P(Xi)
11:    S←FindCandidateParents(Xi, P(Xi))
12:    P(Xi)←S∩P(Xi)
13: endwhile
14: Π←GetSubsets(S)
15: return Π
```

In Algorithm 1, n represents the number of all nodes. $T(X_i)$ represents the candidate parent variable set to be searched of X_i. S represents the local candidate parent set has been found, and it can be obtained using *FindCandidateParents*(X_i, $P(X_i)$) [24]. Π represents the candidate parent sets of X_i. Initially, we set $P(X_i)$ to Ø and add all the nodes to $T(X_i)$ except for X_i, as shown in Step 1–6. Then it adds X_i to $P(X_i)$, and takes

out a node from $T(X_i)$ and adds it to $P(X_i)$ in each iteration, Then it uses *FindCandidateParents*(X_i, $P(X_i)$) to get S, as shown in Step 7–11. Then it assigns $P(X_i)$ to $S \cap P(X_i)$. The above operations are repeated until $T(X_i)$ is Ø, and it adds all the subsets of S to Π, as shown in Step 12–15.

Through the above steps, we can get the candidate parent sets of nodes A, B, C, D, E, F. The candidate parent node set of each node is arranged in descending order according to the Bayesian network score, as follows, A: {D, F}, {D}, {F}, {}; B: {C, D}, {C}, {D}, {}; C: {B}, {E}, {}; D: {A, B}, {A, E}, {B, E}, {A}, {B}, {}; F: {A}, {E}, {}.

Firstly, it determines $T(X_i)$ that means the candidate parent set to be searched of node X_i, takes a node from $T(X_i)$ and adds it to the local operation set $P(X_i)$ each iteration. Then it determines X_i's local candidate parent set S by calling *FindCandidateParents*(X_i, $P(X_i)$) [24]. Then it updates $P(X_i)$ to be the intersection of $P(X_i)$ and S. When $T(X_i)$ is null, it takes all the subsets of S as the candidate parent sets of X_i.

3.2 Initial Network Construction

Firstly, it selects $k + 1$ nodes according to the initial sub-network nodes selection rule. The accurate Bayesian network learning algorithm [25] is used to get the initial sub-network of the $k + 1$ nodes. Then the heuristic evaluation rule is used to add subsequent nodes iteratively, and it uses the k-tree constraint condition to reduce the complexity of processing large number of variables.

Select the Initial k + 1 Nodes: Supposing node set $R = \emptyset$, which expresses the node set that have been selected to construct the initial sub-network. The node set $Z = \emptyset$, it expresses the node set that can be selected to construct the initial sub-network. Firstly, it randomly selects a node to join R, and adds all the candidate parent nodes of R to Z. Then it selects a node in Z, and adds all the candidate parent nodes of the node to R. Repeated the above operations until R includes $k + 1$ nodes. In the process of selecting nodes from Z, if $Z = \emptyset$, it selects a node from all the nodes that have not been added in Z and updates Z. When $k = 2$, $2 + 1 = 3$ nodes are selected to construct the initial sub-network. The specific process of selecting $k + 1$ nodes from the 6 nodes and constructing the initial sub-network is shown in Fig. 1.

In Fig. 1, $R = \emptyset$, $Z = \emptyset$, it first randomly selects node A to join R, and adds all candidate parent nodes {D, F} of node A to Z, as shown in Fig. 1(2). Then it selects node D to join R in Z, and adds all its candidate parent nodes {B, E} to Z, as shown in Fig. 1(3). Then it selects node B to join R in Z, and adds all its candidate parent nodes {C, D} to Z, as shown in Fig. 1(4). Then the number of nodes in R is $k + 1$, and ends the algorithm.

Generate Sub-Network of the Initial k + 1 Nodes: The accurate Bayesian learning algorithm [25] is used to construct the initial sub-network G_{k+1} and k-tree K_{k+1} of $k + 1$ nodes.

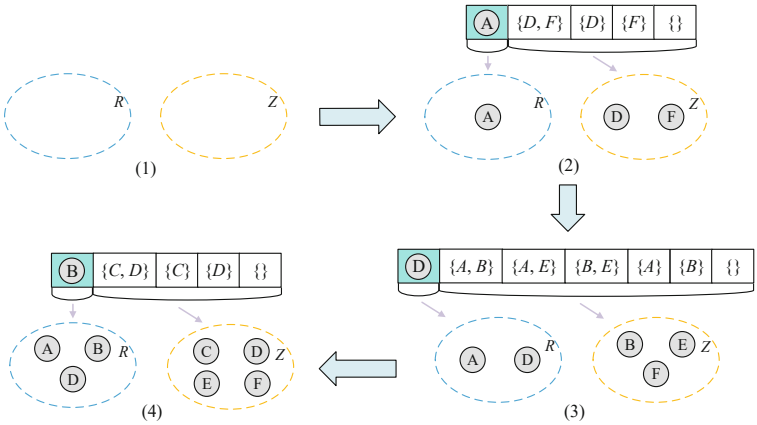

Fig. 1. Select $k + 1$ nodes to construct the initial sub-network

Heuristic Evaluation Method: It uses Eq. (2) to calculate the heuristic score $e(X_i)$ for the remaining other nodes X_i, and it selects nodes with larger BN scores to join the network structure. There are two parts in Eq. (2): the first part is the relative size measurement between the BN score of the feasible parent node set that each node can currently reach and the BN scores of all its candidate parent node sets. The latter part is the absolute size measurement between the BN score of the feasible parent node set that each node can currently reach and the BN scores of all its candidate parent node sets.

$$e(X_i) = \frac{sc^C(X_i) - sc^W(X_i)}{sc^B(X_i) - sc^W(X_i)} + \frac{sc^C(X_i) - sc^Q(X_i)}{sc^P(X_i) - sc^Q(X_i)} \tag{2}$$

$$sc^C(X_i) = \max_{\Pi \in L_i^*} score(\Pi) \tag{2-1}$$

$$sc^B(X_i) = \max_{\Pi \in L_i} score(\Pi) \tag{2-2}$$

$$sc^W(X_i) = \min_{\Pi \in L_i} score(\Pi) \tag{2-3}$$

$$sc^P(X_i) = \max_{i \subset X} \left(\max_{\Pi \in L_i^*} score(\Pi) \right) \tag{2-4}$$

$$sc^Q(X_i) = \min_{i \subset X} \left(\max_{\Pi \in L_i^*} score(\Pi) \right) \tag{2-5}$$

In the above equations, L_i represents all the feasible parent sets of node X_i. L^*_i represents a subset of all feasible parent sets of node X_i. Π represents a set of candidate parent nodes of X_i, and score(Π) represents the BN score of parent node set Π.

Add Subsequent Nodes to Build the Network: $G_{i\text{-}1}$ and $K_{i\text{-}1}$ represent the DAG and k-tree respectively before node X_i updated. Adding node X_i to $G_{i\text{-}1}$, it can get the updated DAG G_i. It connects node X_i to the nodes corresponding to k-tree and constrains its parent Π_i to be the k-clique of $K_{i\text{-}1}$, and thus to obtain the updated k-tree K_i. Compared with $K_{i\text{-}1}$ before updated, K_i has an additional $(k + 1)$-clique. Figure 2 shows the change process of the network and k-tree structure after adding nodes iteratively.

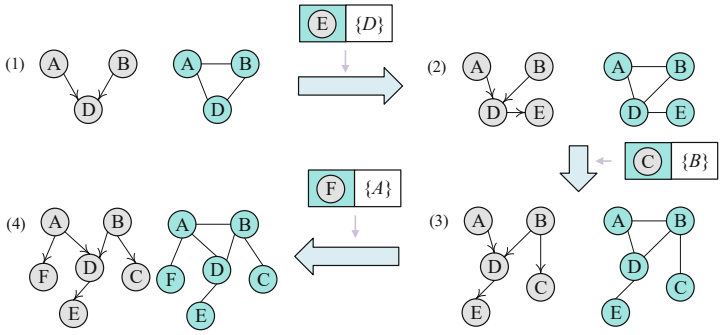

Fig. 2. Add subsequent nodes to build the network

In Fig. 2, when to add the first subsequent node, $e(E) > e(C)$ and $e(E) > e(F)$, node E is added to the current network. The candidate parent set with the highest score of E is $\{D\}$, and the updated DAG and k-tree are shown in Fig. 2(2). When to add the second subsequent node, $e(C) > e(F)$, it adds node C to the network. The candidate parent set with the highest score for node C is $\{B\}$, and the updated DAG and k-tree are shown in Fig. 2(3). When to add the last node, only node F has not yet been joined the network. Then node F is directly added to the network, and the parent set with the largest score is $\{A\}$. The updated DAG and k-tree are shown in Fig. 2(4). The specific process of constructing the initial network is shown in Algorithm 2.

```
Algorithm 2. Algorithm of initial network construction
Input: X: variable set, Π: CandidateParentSet, k:
treewidth
Output: Graph, order: variable addition order, T: Sub-
network node set
1: Graph←∅, order←∅, T←∅
2: Randomly pick k+1 nodes in X and put them in T
3: Graph=Exact algorithm(T)
4: X←X-T
5: while X!=∅
6:     e_max←∞, index=0
7:      for each X_i ∈X do
8:          if e(X_i)>e_max then
9:          e_max=e(X_i)
10:         index=i
11:     end for
12:     Graph←Update(Graph, Π_index)
13:     order←order.append(X_index)
14:     X←X-X_index
15: end while
16: return order, T, Graph
```

In Algorithm 2, X represents the nodes set, Π represents the candidate parent set of a node, k represents the tree width, *order* represents the order in which subsequent nodes are added into the network, T represents the $k + 1$ nodes to be constructed the initial sub-network, and *Graph* represents the obtained *BN* network. Firstly, the initial sub-network node selection rule is used to select the node set T, and it constructs the corresponding network and k-tree structure through the exact *BN* learning algorithm, as shown in Step 2–3. Then it removes T from X to avoid adding the repeated nodes, as shown in Step 4. Then it selects the node X_{index} with the largest evaluation index $e(*)$ from X. Then it adds node X_{index} to *Graph* and gets the node adding order. Then it removes X_{index} from X, as shown in Step 6–14. It repeats the above operations iteratively until X is empty.

3.3 Optimizing Network Using Ordering-Based Search

Based on the ordering obtained by adding nodes, we optimize the network structure through switching the nodes order iteratively until the score of the network not increases. The nodes in the ordering satisfy the following property: only the node sets and its subsets in front of X_i can be regarded as the parent node set of X_i. Through the strategy of exchanging adjacent nodes in ordering, we can get a network with higher *BN* score through multiple iterations. The operator of exchanging adjacent nodes in ordering is shown in Eq. (3).

$$Swap(i) : (X_1, ..., X_i, X_{i+1}, ..., X_m) \mapsto (X_1, ..., X_{i+1}, X_i, ..., X_m) \tag{3}$$

In Eq. (3), *order*[*i*] represents the *i*-th node X_i in ordering. Firstly, it selects the node pair of *order*[*i*] and *order*[*i* + 1] that increases the network score the most, and then exchanges them to update ordering. The above operation is executed iteratively until the exchange operations fail to increase the network score. The relative positions of nodes that are not involved in the exchange operation have not changed, and only the supersets of *order*[*i*] and *order*[*i* + 1] need to be recalculated after the exchange operation. We use the following two rules to improve calculation efficiency.

(1) If the parent node of *order*[*i*] contains *order*[*i* + 1], the parent node set of *order* [*i*] needs to be recalculated.
(2) If the parent node of *order*[*i* + 1] contains *order*[*i*], the parent node set of *order* [*i* + 1] needs to be recalculated.

The specific process of using ordering-based search to optimize the initial network is shown in Algorithm 3.

```
Algorithm 3. Algorithm of optimizing network using order-
ing-based search
Input: order: variable addition order, T: variable set,
Graph: initial network, L: best parent set, k: tree width
Output: G: Bayesian network
1: forbidden[n], index←0, maxScore←-∞
2: for each T_i∈T do
3:     forbidden[T_i]=false
4: end for
5: while maxScore not changing
6:     for each order[i] do
7:         oldScore←getScore(Graph, or-
der[i])+getScore(Graph, order[i+1])
8:         newScore←0
9:         forbidden[order[i+1]]←false
10:        for each Π∈L_order[i] do
11:            flag←true
12:            for each x∈Π do
13:                if forbidden[x]==true then
14:                    flag←false, break
15:            if flag==true then
16:                newScore←score(Π)
17:            for each Π in L_order[i+1] do
18:                flag←true
19:            for each x in Π do
20:                if forbidden[x]==true then
21:                    flag←false, break
22:            if flag==true then
23:               newScore←newScore+score(Π)
24:        forbidden[order[i+1]]←true
25:        if newScore>oldScore and newScore>maxScore
26:            maxScore←newScore
27:            index=i
28:    end for
29:    if maxScore change then
30:         forbidden[order[index+1]]←false
31:         Swap(order[index], order[index+1])
32:         Update(G)
33: end while
34: return G
```

In Algorithm 3, *forbidden* expresses whether the node can be the parent node of the current node or not. *index* indicates the number of the node that currently needs to be

exchanged, and *maxScore* indicates the largest score that the current exchange operation can achieve. In Step 2–4, it sets the *forbidden* of all nodes in T to false, which means that all nodes in T can be regarded as parent nodes of the subsequent nodes. In Step 5–33, it optimizes the network by repeatedly traversing the ordering, and ends the traversal when all the exchange operations fail to increase the network score. In Step 7, it calculates the sum of scores of the current parent node set of *order*[i] and *order*[i + 1], and records it as *oldScore*. In Step 8, *newScore* is used to record the sum of the scores of the parent node set after exchanging *order*[i] and *order*[i + 1]. In Step 11, it sets *forbidden* of *order*[i + 1] to false to ensure that *order*[i] can pick *order*[i + 1] as its parent node. In Step 12–16, it recalculates the best score that *order*[i] can reach after the exchanging, and denotes it as *newScore*. In Step 17–23, it recalculates the best score that *order*[i + 1] can achieve after switching nodes, and denotes it as *newScore*. The Step 24 is used to undo the exchange operation of *order*[i] and *order*[i + 1]. The Step 25–37 is used to judge whether the current exchange operation is the best exchange operation, and record the position information (*index*) of the exchange operation. In Step 29–32, it performs the exchange operation of the best node pair, and updates *order* and *forbidden*. After traversing *ordering* several times, it gets the network G with the largest *BN* score, as shown in Step 34.

Figure 3 shows an example of using ordering to optimize the initial network.

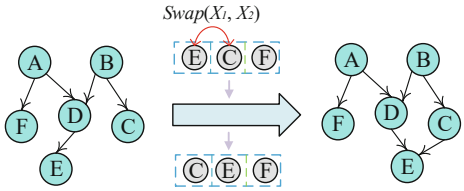

Fig. 3. Switching node order to optimize the network

In Fig. 3, it traverses the node order and gets the best switching node pair of E, C. Then it exchanges the position of E, C and updates the network and *order*.

4 Experiment

4.1 Dataset and Evaluation Method

The commonly used datasets of *Alarm1*, *Alarm3*, *Alarm5*, *Insurance1*, *Insurance3*, *Insurance5* are used for experiment comparison and analysis. We use the *Precision*(P), *Recall rate*(R) and *F1-score* to do the experiment evaluation and analysis, as shown in the Eq. (4)–Eq. (6).

$$\mathrm{P} = \frac{TP}{TP + FP} \tag{4}$$

$$R = \frac{TP}{TP + FN} \tag{5}$$

$$F1 = \frac{2 * P * R}{P + R} \tag{6}$$

In the equations, *TP* refers to the number of correctly identified edges, *TN* refers to the number of correctly identified non-linked edges, *FP* refers to the number of incorrectly identified edges, and *FN* refers to the number of incorrectly identified non-linked edges.

We carry the experiment on a computer with the configuration of Inter(R) Core (TM) i5-8300H CPU @ 2.30 GHz 2.30 GHz and 8G memory. The number of samples in each dataset is 5000. In order to ensure the accuracy of the experiments, we generate each dataset 10 times, and take the average of 10 datasets for comparison and analysis.

4.2 Experiment Results and Analysis

Experiments of Tree Width

In the *k*-tree related algorithms, the tree width *k* has a greater impact on the accuracy and efficiency of these algorithms. We compare and analyze the following *k*-tree related algorithms: *KG* [18], *KGADV* [19], *KMAX* [20] and *KTOBS*. For the *Alarm* dataset, the tree width is set to 2, 4, 6, and the results are shown in Tables 1, 2, 3 and 4 within a given time limit of 10s. For the *Insurance* dataset, the tree width is set to 2, 4, 6, and the results are shown in Tables 5, 6, 7 and 8 within a given time limit of 10 s.

Table 1. The result of *KG* when *k* takes different values (*Alarm*).

KG	Iterations	Accuracy	Recall	Precision	F1-score
k = 2	391588.3	0.957	0.341	0.259	0.294
k = 4	82304.2	0.953	0.352	0.422	0.383
k = 6	55292.1	0.953	0.352	0.435	0.389

Table 2. The result of *KGADV* when *k* takes different values (*Alarm*).

KGADV	Iterations	Accuracy	Recall	Precision	F1
k = 2	525762.4	0.956	0.319	0.243	0.276
k = 4	110756.1	0.9496	0.309	0.383	0.342
k = 6	68878.3	0.9496	0.309	0.376	0.339

Table 3. The result of *KMAX* when *k* takes different values (*Alarm*).

KMAX	Iterations	Accuracy	Recall	Precision	F1
k = 2	65149	0.958	0.361	0.283	0.317
k = 4	11299.2	0.947	0.289	0.372	0.325
k = 6	10347.3	0.952	0.338	0.417	0.373

Table 4. The result of *KTOBS* when *k* takes different values (*Alarm*).

KTOBS	Iterations	Accuracy	Recall	Precision	F1
k = 2	126199.4	0.962	0.434	0.337	0.381
k = 4	33068.6	0.992	0.884	0.885	0.884
k = 6	22526.9	0.995	0.917	0.939	0.928

Table 5. The result of *KG* when *k* takes different values (*Insurance*).

KG	Iterations	Accuracy	Recall	Precision	F1
k = 2	552799.3	0.930	0.558	0.263	0.358
k = 4	152056.2	0.944	0.655	0.515	0.577
k = 6	95244.4	0.940	0.617	0.504	0.5575

Table 6. The result of *KGADV* when *k* takes different values (*Insurance*).

KGADV	Iterations	Accuracy	Recall	Precision	F1
k = 2	727985.1	0.930	0.560	0.269	0.364
k = 4	194096.6	0.941	0.630	0.510	0.563
k = 6	108880.7	0.946	0.671	0.540	0.599

Table 7. The result of *KMAX* when *k* takes different values (*Insurance*).

KMAX	Iterations	Accuracy	Recall	Precision	F1
k = 2	133164.3	0.936	0.637	0.308	0.415
k = 4	30496.5	0.936	0.588	0.458	0.515
k = 6	31004.6	0.943	0.650	0.494	0.562

Table 8. The result of *KTOBS* when *k* takes different values (*Insurance*).

KTOBS	Iterations	Accuracy	Recall	Precision	F1
k = 2	206243.3	0.939	0.683	0.331	0.446
k = 4	50120.4	0.969	0.880	0.673	0.763
k = 6	32039.2	0.973	0.910	0.698	0.790

In Table 1-Table 8, the number of iterations of the four kinds of algorithms decreases gradually as the tree width *k* increases. It means that the efficiency gradually decreases as tree width *k* increases. In addition, it can be seen that the accuracy of the four kinds of algorithms gradually increases as the tree width *k* increases. On the whole, the four indicators (*Accuracy*, *Precision*, *Recall rate*, *F1-score*) of *KTOBS* are higher than those of *KG*, *KGADV* and *KMAX*. Considering the efficiency and accuracy together, we set tree width *k* to 4 in the following experiments.

Comparison and Analysis of Accuracy

Given a limited time (10s), this experiment compares and analyzes the accuracy of *KG* [18], *KGADV* [19], *KMAX* [20], *OBS* [26] and *KTOBS* in the datasets of *Alarm*(*Alarm1*, *Alarm3*, *Alarm5*) and *Insurance*(*Insurance1*, *Insurance3*, *Insurance5*). The experimental results are shown in Fig. 4.

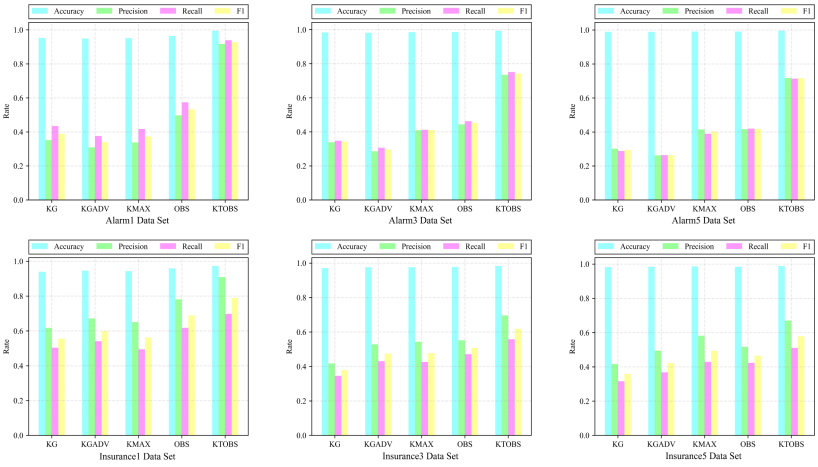

Fig. 4. Accuracy comparison on *Alarm* and *Insurance* dataset

In Fig. 4, the learning accuracy (*Accuracy*, *Precision*, *Recall*, *F1*-score) of *KTOBS* is significantly higher than that of *KG*, *KGADV*, *KMAX* and *OBS*. The accuracy of the five kinds algorithms is relatively high, basically above 0.95. The *Precision*, *Recall*, *F1-Score* of *KG*, *KGADV*, *KMAX* is lower than *OBS* and *KTOBS* apparently. But the *Precision*, *Recall*, *F1-Score* of *OBS* is slightly worse than *KTOBS*.

Efficiency Comparison and Analysis

This experiment compares and analyzes the running time of one iteration of *KG* [18], *KGADV* [19], *KMAX* [20], *OBS* [26] and *KTOBS* in the datasets of *Alarm*(*Alarm1*, *Alarm3*, *Alarm5*) and *Insurance*(*Insurance1*, *Insurance3*, *Insurance5*). The experimental results are shown in Fig. 5.

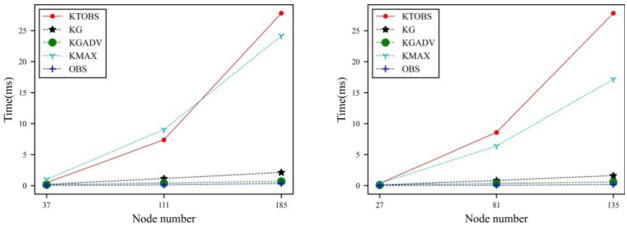

Fig. 5. Efficiency comparison on *Alarm* and *Insurance* dataset

In Fig. 5, the running time of one iteration of all algorithms increases with the increasing of the variable number in the dataset. It is consistent with the law of efficiency decreasing as the complexity of the problem increases. The time spent in one iteration of *KTOBS* is slightly higher than that of other four kinds of algorithms. In the *Alarm* dataset, the learning efficiency of *KTOBS* and *KMAX* is not much different.

In summary, *KTOBS* can obtain a network structure with a higher accuracy within the given limited time. *KTOBS* does not have obvious advantages in the view of the running time of a single iteration. But *KTOBS* has a greater advantage than other methods in the same limited running time in the view of the accuracy.

5 Conclusions

This work proposes a Bayesian network structure learning method (*KTOBS*) based on *k*-tree optimizing ordering-based search. Firstly, the strategy of dynamic programming and local learning is adopted to get the candidate parent sets of each node efficiently and accurately. Then it constructs the initial sub-network of $k + 1$ nodes, and adds the subsequent nodes successively according to the heuristic evaluation strategy, At the same time, it reduces the complexity of processing large number of nodes according to the *k*-tree structure. Based on the node order obtained by adding the subsequent nodes, it exchanges the adjacent nodes in ordering iteratively until the score of the network no longer increases. The classical datasets of *Alarm* and *Insurance* are used in experiment verification and analysis. Experiment results show that *KTOBS* can obtain a network structure with higher accuracy than other *k*-tree related algorithms in a given limited time.

Acknowledgement. This research is supported by the Fundamental Research Funds for the Central Universities under grant No. 2020BC211, S202010504276.

References

1. Cooper, G.F., Herskovits, E.: A Bayesian method for the induction of probabilistic networks from data. Mach. Learn. **9**(4), 309–347 (1992)
2. Spirtes, P., Glymour, C.: An algorithm for fast recovery of sparse causal graphs. Soc. Sci. Comput. Rev. **9**(1), 62–72 (1990)
3. Spirtes, P., Glymour, C., Scheines, R.: Causality from probability. In: Evolving Knowledge in Natural and Artificial Intelligence, pp. 181–199. Pitman, London (1989)
4. Cheng, J., Bell, D., Liu, W.: Learning Bayesian Networks from Data: An Efficient Approach Based on Information Theory (1997)
5. Tsamardinos, I., Aliferis, C.F., Statnikov, A.R.: Algorithms for Large Scale Markov Blanket Discovery. International Flairs Conference, pp. 376–380 (2003)
6. Bouckaert, R.R.: A stratified simulation scheme for inference in Bayesian belief networks. Uncertainty Proc. **5**(1), 110–117 (1994)
7. Tsamardinos, I., Brown, L.E., Aliferis, C.F.: The max-min hill-climbing Bayesian network structure learning algorithm. Mach. Learn. **65**(1), 31–78 (2006)

8. Gao, T., Ji, Q.: Efficient score-based Markov blanket discovery. Int. J. Approximate Reasoning **80**, 277–293 (2017)
9. Gao, T., Fadnis, K., Campbell, M.: Local-to-global Bayesian network structure learning. In: Proceedings of the 34th International Conference on Machine Learning, pp. 1193–1202 (2017)
10. Parviainen, P., Kaski, S.: Learning structures of Bayesian networks for variable groups. Int. J. Approximate Reasoning **88**, 110–127 (2017)
11. Niinimäki, T., Parviainen, P., Koivisto, M.: Structure discovery in Bayesian networks by sampling partial orders. J. Mach. Learn. Res. **17**(1), 2002–2048 (2016)
12. Campos, C.P., Ji, Q.: Efficient structure learning of Bayesian networks using constraints. J. Mach. Learn. Res. **12**(3), 663–689 (2011)
13. Li, A., Beek, P.: Bayesian network structure learning with side constraints. In: International Conference on Probabilistic Graphical Models, pp. 225–236 (2018)
14. Bartlett, M., Cussens, J.: Integer linear programming for the Bayesian network structure learning problem. Artif. Intell. **244**, 258–271 (2017)
15. Bartlett, M., Cussens, J.: Advances in Bayesian network learning using integer programming. arXiv preprint arXiv:1309.6825 (2013)
16. Nie, S., de Campos, C.P., Ji, Q.: Efficient learning of Bayesian networks with bounded tree-width. Int. J. Approximate Reasoning **80**, 412–427 (2017)
17. Nie, S., De Campos, C.P., Ji, Q.: Learning bounded tree-width Bayesian Networks via sampling. In: European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty. Springer, Cham, pp. 387–396 (2015).https://doi.org/10.1007/978-3-319-20807-7_35
18. Scanagatta, M., et al.: Learning treewidth-bounded Bayesian networks with thousands of variables. In: Advances in Neural Information Processing Systems, pp. 1462–1470 (2016)
19. Scanagatta, M., Corani, G., de Campos, C.P., Zaffalon, M.: Approximate structure learning for large Bayesian networks. Mach. Learn. **107**(8–10), 1209–1227 (2018). https://doi.org/10.1007/s10994-018-5701-9
20. Scanagatta, M., et al.: Efficient learning of bounded-treewidth Bayesian networks from complete and incomplete data sets. Int. J. Approximate Reasoning **95**, 152–166 (2018)
21. Cooper, G., Hersovits, E.: A Bayesian method for the induction of probabilistic networks from data. Mach. Learn. **9**, 309–347 (1992)
22. Heckerman, D., Geiger, D., Chickering, D.M.: Learning Bayesian networks: the combination of knowledge and statistical data. Mach. Learn. **20**(3), 197–243 (1995)
23. Schwarz, G.: Estimating the dimension of a model. Ann. Stat. **6**(2), 461–464 (1978)
24. Silander, T., Myllymaki, P.: A simple approach for finding the globally optimal Bayesian network structure. In: Proceedings of the Twenty-Second Annual Conference on Uncertainty in Artificial Intelligence (2006)
25. Cussens, J.: Bayesian network learning with cutting planes. In: Proceedings of the 27th Conference Annual Conference on Uncertainty in Artificial Intelligence, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, 14–17 July, pp. 153–160 (2011)
26. Teyssier, M., Koller, D.: Ordering-based search: a simple and effective algorithm for learning Bayesian networks. In: Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence. Arlington, USA, pp. 584–590 (2005)