Accelerating Learning Bayesian Network Structures by Reducing Redundant CI Tests

Wentao Hu, Shuai Yang, Xianjie Guo, and Kui Yu
School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, China
{wentaohu,yangs,xianjieguo}@mail.hfut.edu.cn; yukui@hfut.edu.cn

***Abstract*—The type of constraint-based methods is one of the most important approaches to learn Bayesian network (BN) structures from observational data with conditional independence (CI) tests. In this paper, we find that existing constraint-based methods often perform many redundant CI tests, which significantly reduces the learning efficiency of those algorithms. To tackle this issue, we propose a novel framework to accelerate BN structure learning by reducing redundant CI tests without sacrificing accuracy. Specifically, we first design a CI test cache table to store CI tests. If a CI test has been computed before, the result of the CI test is obtained from the table instead of computing the CI test again. If not, the CI test is computed and stored in the table. Then based on the table, we propose two CI test cache table based PC (CTPC) learning frameworks for reducing redundant CI tests for BN structure learning. Finally, we instantiate the proposed frameworks with existing well-established local and global BN structure learning algorithms. Using twelve benchmark BNs, the extensive experiments have demonstrated that the proposed frameworks can significantly accelerate existing BN structure learning algorithms without sacrificing accuracy.**

***Index Terms*—Bayesian network, Structure learning, Redundant CI test, CI test cache table**

I. Introduction

A Bayesian Network (BN) is a type of a probabilistic graphical model introduced by Pearl [1]. The structure of a BN is a directed acyclic graph (DAG) in which nodes of DAG denote the variables in a dataset and edges represent the dependence relationships between variables. Fig.1 (a) gives an example of a BN structure, a directed edge $B \rightarrow A$ denotes that B is the parent of A and A is the child of B. BNs have been widely used in many real-world applications, such as Earth system, bioinformatics, and neuroscience [2], [3], and thus learning BNs from observational data has been extensively studied in the past two decades [4]–[7].

Existing algorithms for learning BN structures can be mainly divided in two categories: score-based methods and constraint-based methods [4], [8], [9]. Score-based algorithms assign a score to each candidate BN structure for evaluating how well the candidate BN structure fits a dataset [10]. Constraint-based algorithms learn a BN structure with conditional independence (CI) tests by leveraging the probabilistic relationships entailed by the Markov property of BNs [11]. Since constraint-based methods for BN structure learning are computationally feasible for sparse graphs with up to thousands of variables and easily generalize to the problem of causal insufficiency, constraint-based approaches have attracted much attention and have been widely applied to many diverse real-world problems [12], [13].

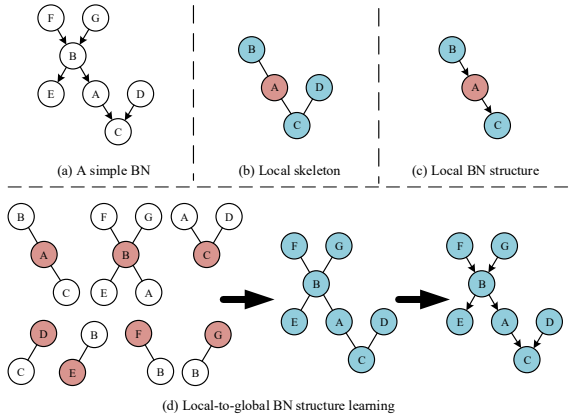

Fig. 1: (a) An example of a BN structure. (b), (c) and (d) Examples to illustrate three types of existing constraint-based algorithms for BN structure learning.

Existing constraint-based algorithms for BN structure learning can be roughly grouped into the following three categories, as shown in Fig. 1 (b) to (d).

- Learning a local skeleton of a variable of interest [8], [14], as shown in Fig. 1 (b). The local skeleton of a variable is the Markov blanket (MB) (or parents and children, PC) of the variable. A MB of a variable in a BN consists of the variable's PC (parents and children) and spouses (the other parents of the children of the variable). As shown in Fig. 1, the MB of A contains B (parent), C (child), and D (spouse). Existing local skeleton learning algorithms only learn an undirected local graph around the variable and do not distinguish parents from children in the learnt MB.
- Learning a local BN structure of a variable of interest [9], [15], as shown in Fig. 1 (c). Existing local structure learning algorithms first learn the local skeleton (i.e. PC) of a variable, then go one step further to distinguish parents from children in this learnt skeleton.
- Learning an entire BN structure, as shown in Fig. 1 (d). Existing approaches mainly employ a local-to-global strategy which first learns the local skeleton (i.e. PC) of each variable in a dataset, then constructs the global skele-

2021 IEEE International Conference on Big Knowledge (ICBK) | 978-1-6654-3858-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICKG52313.2021.00016

978-1-6654-3858-2/21/$31.00 ©2021 IEEE
DOI 10.1109/ICKG52313.2021.00016

ton involved all variables using the learnt local skeletons, and finally orients edges in this global skeleton [4], [16].

As discussed above, we can see that the most important yet common step of the three types of constraint-based algorithms is to learn local skeletons using a PC learning algorithm (e.g. MMPC [17] and HITON-PC [16]). When using a PC learning algorithm for local skeleton learning, we find that it generally produces the following two types of redundant CI tests.

One is a redundant CI test within PC learning. Existing PC learning methods mainly adopt a forward-backward learning strategy to learn the PC of a target variable. They first learn a candidate PC set of the target variable, then remove false positives from the candidate PC set. At each iteration, those methods perform an exhaustive subset search within the currently candidate PC set for finding the PC of the target variable, leading to that many CI tests are repeatedly computed. We call those repeatedly computed CI tests as redundant CI tests within PC learning. For example, in Fig. 2 (a), in order to learn the MB of A, we first need to learn the PC set of A and find that the CI tests such as $I(A,B \mid \emptyset)$ and $I(A,C \mid \emptyset)$ are computed repeatedly many times.

The other is a redundant CI test between PC learning. Existing local/global BN structure learning methods need to learn the PC sets of some/all variables. For example, in Fig. 1 (a), if we learn the local BN structure of A, existing local BN structure learning algorithms first learn the PC set of A ($B-A-C$). Then to orient edges between A and B, they need to learn the PC set of B. Meanwhile, to identify spouses of a variable, the MB (local skeleton) learning methods need to learn the PC of each variable within the PC set of this variable. When learning the PC sets for some/all variables, we find the following type of CI tests repeatedly computed and call them redundant CI tests between PC learning. One is the same CI test with a different order of a target variable and another variable, while the other is the same CI tests with a different order of variables in a conditioning set.

Fig. 2 gives an example using the well-established MMMB algorithm [16] to illustrate redundant CI tests between PC learning. To learn the MB of A, MMMB first learns the PC set of A, then learn the PC sets of B and C for identifying spouse D. The CI tests $I(A,B \mid \emptyset)$, $I(A,B \mid C)$ and $I(A,C \mid \{E,B\})$ have been computed in learning PC of A. However, in learning the PC sets of B and C, we find that these CI tests such as $I(B,A \mid \emptyset)$, $I(B,A \mid C)$ and $I(C,A \mid \{B,E\})$ need to be computed again, although the pairs of CI tests ([$I(A,B \mid \emptyset)$ and $I(B,A \mid \emptyset)$], [$I(B,A \mid C)$ and $I(A,B \mid C)$], [$I(C,A \mid \{B,E\})$ and $I(A,C \mid \{E,B\})$]) are the same CI tests.

Those redundant CI tests significantly reduce the learning efficiency of existing constraint-based BN structure learning algorithms, especially with high-dimensional data. However, the problem of the redundant CI tests discussed above has not received any attention in the literature. To tackle this problem, the contributions of this paper are summarized as follows.

- We first design a CI test cache table to store CI tests and the CCT_select algorithm for retrieving CI tests. Then based on the table and the CCT_select algorithm, we propose two CI test cache table based PC (CTPC) learning frameworks to reduce the two types of redundant CI tests in BN structure learning. The two frameworks are able to be instantiated by any existing constraint-based BN structure learning algorithms without violating the original ideas of these algorithms.
- Using twelve benchmark BN datasets, we present ten BN structure learning algorithms by instantiating the CTPC frameworks using ten existing BN structure learning algorithms and conduct a series of experiments to validate the efficiency of the proposed CTPC frameworks. The experiments have demonstrated that the CTPC frameworks can significant accelerate existing BN structure learning algorithms without sacrificing accuracy.

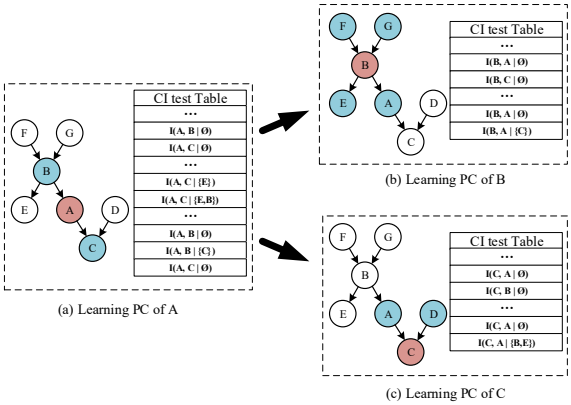

Fig. 2: (a), (b) and (c) take an example to illustrate two types of redundant CI tests.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 proposes the CI test cache table and the CTPC frameworks. Section 4 presents and discusses the experiments. Section 5 concludes the paper.

II. Related Work

A. MB learning

Recently, a variety of methods has been designed for MB discovery. Existing methods can be divided into simultaneous MB learning and divide-and-conquer MB learning [8]. The simultaneous MB learning algorithms learn the MB of a target variable simultaneously without distinguishing spouses from PC. One typical simultaneous MB learning method is GSMB [18], which consists of growing and shrinking phases. In the growing phase, the variable that is dependent on the target variable conditioning on the currently candidate MB set will be added to the candidate MB set. In the shrinking phase, all false positives will be removed from the candidate MB set. Based on GSMB, IAMB [19] and its variants such as IAMBnPC [20], Inter-IAMB [20], and Fast-IAMB [21] have been proposed to improve the performance of GSMB. Existing simultaneous MB learning algorithms are computationally efficient, but the quality of MBs learnt by those algorithms

degrades greatly in practical settings when the size of the MB is large and the data samples are insufficient.

To reduce the data requirements of the simultaneous MB learning algorithms, divide-and-conquer MB learning algorithms have been proposed, which learn PC and spouses of a target variable separately. The MMMB algorithm [17] learn the PC set of a target variable by performing a subset search within the candidate PC set currently selected instead of conditioning on the entire candidate PC set. Based on MMMB, HITON-MB [22] and semi-HITON-MB [22] removes false positives from the candidate PC set as early as possible during PC learning by interleaving the growing and shrinking phases. Although MMMB and HITON-MB have achieved promising performance, they are proved to be incorrect under the faithfulness assumption [23]. PCMB [23] was proposed to solve this problem, which uses symmetry constraints to ensure the correctness of the PC learning.

B. Local BN structure learning

To distinguish parents from children in a local BN structure around a target variable, several local structure learning methods have been developed, such as PCD-by-PCD [24], MB-by-MB [25] and CMB [15]. Given a target variable, PCD-by-PCD [24] recursively identifies the PC set of the target variable using a PC learning algorithm, then constructs a skeleton with the learnt PC and iteratively identifies v-structures by learning PC sets of other variables and uses the found v-structures and Meek rules [26] for edge orientations. MB-by-MB is developed based on PCD-by-PCD, and it learns the MB of the target variable instead of learning its PC set. Compared to MB-by-MB, CMB also employs standard MB learning algorithms (such as HITON-MB) to find the MB to construct the local skeleton, but CMB needs to track conditional independence changes after MB learning for edge orientations. Since these local structure learning methods need to perform an exhaustive subset within the currently selected variables for PC learning, which are computationally expensive or even prohibitive on high-dimensional data, Ling et al. [9] recently employed feature selection for PC learning to speed up local BN structure learning.

C. Global BN structure learning

Constraint-based global BN learning methods aim to learn an entire BN structure involved all variables in a dataset using CI tests. The pioneer algorithms include the SGS [11] and PC [27] algorithms. They first construct a BN structure skeleton by testing whether there exist the edges between each pair of variables, then identify all v-structures in the skeleton, and finally orient edges as many as possible using Meek rules. However, SGS and PC are often computationally infeasible on high-dimensional datasets. To tackle this issue, many local-to-global BN structure learning methods have been proposed. GSBN [18] first employs GSMB to build a global BN skeleton, then orients edges using CI tests. Inspired by GSBN, several local-to-global structure learning algorithms have been proposed, such as MMHC [4], SLL+C [28] and SLL+G [28]. MMHC [4] first constructs a global skeleton using the MMPC algorithm [17], and then employs a score-based method and hill-climbing search strategy for orienting edges. SLL+C and SLL+G employ SLL [28] that is a score-based MB learning algorithm to build a global BN skeleton, and then use constraint-based or score-based methods to determine the orientation of edges. Instead of identifying the MB or PC of all variables first, GGSL [29] starts from randomly selected variable and employs a score-based MB discovery algorithms (i.e., S^2TMB [30]) to build the local structure around the target variable, then gradually expands the local structure until the entire BN structure has been learned.

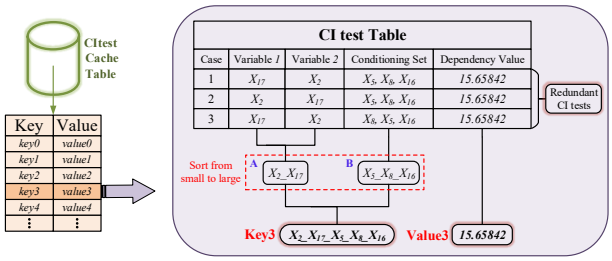

Fig. 3: An example of the CI test cache table. (a) The left part shows the storage structure of the CI test cache table. (b) The right part takes a table as an example to illustrate how to cache CI tests by the data structure. In the CI test Table, V_i denotes the i-th variable in the dataset (Columns are variables and rows are samples).

III. Proposed Algorithms

In this section, we first design a CI test cache table for reducing redundant CI tests. To retrieve CI tests in the table, we propose the CCT_select algorithm. Then based on the CCT_select algorithm, we propose two types of CI test cache table based PC (CTPC) learning frameworks for BN structure learning. Let $\boldsymbol{V}$ denote a set of random variables and let $X \perp\!\!\!\perp Y|S$ denotes that X and Y are conditionally independent conditioned on $S \subseteq V \backslash \{X \cup Y\}$, and $X \not\perp\!\!\!\perp T|S$ represents X and Y are conditionally dependent conditioned on S.

A. CI test cache table

In this section, we design a new data structure, called CI test cache table (CCT), to reduce redundant CI test, as shown in Fig. 3. A CCT is based on a key-value data structure to store the CI tests. As show in Fig. 3, the Key part of a CCT uses key generated from the tested variables and a condition set of the CI test as an index, and the Value part of a CCT stores the result of the CI test. Specifically, we use the key-value data structure "Dictionary" as a CCT, since the "Dictionary" is based on an associative array or a hash table and uses key-value pairs to store and retrieve information, its query and storage are fast, and the uniqueness of the key-value pairs guarantees the CI tests will not be stored repeatedly.

To retrieve and store a CI test in the cache table, we propose the CCT_select algorithm which consists of two phases, as

shown in Algorithm 1. Phase 1 (lines 2 to 8) generates the key for a CI test. Phase 2 (lines 10 to 15) retrieves the result of the CI test based according to the key of the CI test. If the CI test is not in the CI test cache table, the CI test is computed and the corresponding result is stored in the table. Otherwise, the result of the CI test is obtained from the table.

Phase 1 (lines 2 to 8 of Algorithm 1): Based on the given conditioning set S and variables V_1, V_2, the CCT_select generates the key that is used as a keyword for a CI test query in Phase 2. First, V_1, V_2 are sorted in ascending by the values of their subscripts. Then, we splice them into a string κ_1 using char '_' (line 2). If the conditioning set S is empty, we obtain the key κ (line 4), i.e., $\kappa = \kappa_1$. If S is not empty, we sort and splice the variables in S into string κ_2, then we splice strings κ_1 and κ_2 into a key κ (lines 6 to 7). For example, as shown in Fig. 3, assuming that X_i denotes the i-th variable in a dataset. Given conditioning set $S = \{X_8, X_5, X_{16}\}$ and two variables X_{17}, X_2, we first sort and splice the two variables X_{17} and X_2 into $\kappa_1 = X_2\_X_{17}$, then we splice the conditioning set $S = \{X_8, X_5, X_{16}\}$ into $\kappa_2 = X_5\_X_8\_X_{16}$. Finally, we splice the two string $\kappa_1 = X_2\_X_{17}$ and $\kappa_2 = X_5\_X_8\_X_{16}$ into the key $\kappa = X_2\_X_{17}\_X_5\_X_8\_X_{16}$.

Phase 2 (lines 10 to 15 of Algorithm 1): The phase is to obtain the result of a CI test. Specifically, CCT_select first determines whether the key κ of a query CI test exists in the CI test cache table. If κ is in the table, i.e., $\kappa \in CCT.key$, CCT_select will obtain the CI test result (i.e., *dep*) from the cache table. The value of *dep* represents the dependency between variables (line 11). If the key is not in the cache table, CCT_select will compute the CI test using the CI-Test($\cdot$) function to get the value of *dep*, and store the CI test and corresponding value of *dep* in the cache table with the key, i.e., $CCT[\kappa].value \leftarrow dep$. The CI-Test($\cdot$) function can be instantiated by the G^2 test, chi-squared test, mutual information and so on.

Algorithm 1: The CCT_select algorithm

```
Require: variables V1, V2; conditioning set S; CI test cache table CCT
Ensure: dependency value dep
 1: // Phase 1: Generate the key
 2: κ1 ← sort and splice variables V1, V2
 3: if S = ∅ then
 4:       key κ ← κ1
 5: else
 6:       κ2 ← sort and splice variables inside S
 7:       key κ ← splice κ1, κ2
 8: end if
 9: // Phase 2: Query cache table by the key
10: if κ ∈ CCT.key then
11:       dep ← CCT[κ].value
12: else
13:       dep ← CI-Test(V1, V2, S)
14:       CCT[κ].value ← dep
15: end if
16: return dep
```

B. CI test cache table based PC learning frameworks

There are two general strategies for learning the PC set of a variable [8], [31]. One is the standard strategy which sequentially performs a forward step and a backward step. At the forward step, it starts with a (usually empty) candidate PC set of a variable and adds candidate PC features to it until a given stopping criterion is met. And the backward step starts with the candidate PC set obtained from the forward step and removes false PC features from that set until a stopping criterion is met, such as the MMPC algorithm [17]. The other is the interleaving strategy which runs the forward step and backward step alternatively, such as the HITON-PC algorithm [22]. Specifically, if a feature is added to the candidate PC set at the forward step, the interleaving strategy immediately triggers the backward step and implements both steps alternatively.

Based on the two existing PC learning strategies, in this section, we propose two CI test cache table based PC (CTPC) learning frameworks. One framework is called CTPC-S which sequentially performs a forward step and a backward step, as shown in Algorithm 2. The other framework is referred to as CTPC-I which interleaves the forward step and backward step, as shown in Algorithm 3. In the following, we will describe the details of the two frameworks[1] .

The CTPC-S framework: as shown in Algorithm 2, given a target variable T, CTPC-S first performs the forward step to obtain the candidate PC of T, then carries out the backward step to remove false positives using the CCT_select algorithm. CTPC-S starts with an empty candidate PC set (i.e., $CPC = \emptyset$) and a candidate set R that contains all variables which are dependent on T given an empty set.

At each iteration in the forward step, for each variable $X \in R$, line 4 implements CCT_select(T, X, S, CCT) to obtain the dependency value of X and T conditioning on all possible subsets $S \subseteq CPC$, then adds the best variable X that its dependency value with T satisfies the greedy strategy of an algorithm to CPC and removes it from R. If the greedy strategy cannot select the best variable from R, or the chosen variable is conditionally independent of T, the forward step will be terminated.

Then CTPC-S performs the backward step to remove false positives from CPC. At each iteration, CTPC-S examines whether each variable Y in CPC is independent of T conditioning on all possible subsets $S \subseteq CPC \backslash Y$ by performing CCT_select(Y, T, S, CCT). If Y and T are conditionally independent, CTPC-S removes it from CPC. The backward step will be terminated until all the variables in CPC are not independent of T.

The CTPC-I framework: In Algorithm 3, CTPC-I performs the forward step and backward step alternatively. Given a target variable T, CTPC-I starts with CPC and R, which is the same as CTPC-S. At each iteration, for each variable $X \in R$, CTPC-I implements CCT_select(T, X, S, CCT) to obtain the dependency value of T and X conditioning on all possible subsets $S \subseteq CPC$, then chooses the best variable X with the

[1]The examples of how the two CTPC frameworks are instantiated by the two well-established existing PC learning algorithms, MMPC and HITON-PC, are available at https://github.com/wt-hu/AccpyCausalFS/blob/master/Supplementary.pdf

Algorithm 2: The CTPC-S framework

Require: variable set V, target variable T, CI test cache table CCT
Ensure: CPC // the candidate PC of T
1: Initialization: $CPC = \emptyset$, $R = \{\forall X \in V \ \& \ X \not\perp\!\!\!\perp T | \emptyset\}$;
2: // Forward step: Adding relevant variables to CPC
3: **repeat**
4: Obtain the dependency value of each variable $X \in R$ and T conditioning on $S \subseteq CPC$ by CCT_select(X, T, S, CCT), and select the best variable X with a greedy strategy;
5: $CPC = CPC \cup X$, $R = R \backslash X$;
6: **until** no variables in R are added to CPC;
7: // Backward step: Removing false positives from CPC
8: **repeat**
9: Consider each variable $Y \in CPC$, and find a subset $S \subseteq CPC \backslash Y$, obtain the dependency value of Y and T conditioning on S by CCT_select(Y, T, S, CCT);
10: **if** $Y \perp\!\!\!\perp T | S$ **then**
11: $CPC = CPC \backslash Y$;
12: **end if**
13: **until** no variables in CPC are removed;
14: **return** CPC

dependency value with T satisfying the greedy strategy of an algorithm. If so, CTPC-I adds X to CPC and then removes it from R. Once a new variable is added to CPC, at lines 8-12, CTPC-I immediately triggers the backward step to check each variable in CPC for removing false positives to keep the size of CPC as small as possible. Specifically, for each variable $Y \in CPC$, CTPC-I checks whether Y is independent of T conditioning on $S \subseteq CPC$ using CCT_select(T, Y, S, CCT). If so, Y will be removed from CPC and never added again. The interleaving execution of CTPC-I terminates until no new variables can be added to CPC.

Algorithm 3: The CTPC-I framework

Require: variable set V, target variable T, CI test cache table CCT
Ensure: CPC // the candidate PC of T
1: Initialization:$CPC = \emptyset$, $R = \{\forall X \in V \ \& \ X \not\perp\!\!\!\perp T | \emptyset\}$;
2: **repeat**
3: // Forward step: Adding relevant variables to CPC
4: Obtain the dependency value between each variable $X \in R$ and T conditioning on $S \subseteq CPC$ by CCT_select(X, T, S, CCT), and select the best variable X with a greedy strategy;
5: $CPC = CPC \cup X$, $R = R \backslash X$;
6: // Backward step: Removing false positives from CPC
7: **repeat**
8: Consider each variable $Y \in CPC$, and find a subset $S \subseteq CPC \backslash Y$, obtain the dependency value between Y and T conditioning on S by CCT_select(Y, T, S, CCT);
9: **if** $Y \perp\!\!\!\perp T | S$ **then**
10: $CPC = CPC \backslash Y$;
11: **end if**
12: **until** no variables in CPC are removed;
13: **until** no variables in R are added to CPC;
14: **return** CPC

IV. Experiments

In this section, we will systematically evaluate the performance of the proposed CTPC frameworks, and this section is organized as follows. Section 4.1 describes the experiment setting, including datasets, accelerated methods, implementation details and evaluation metrics. Section 4.2 reports the results of the MB learning algorithms accelerated by the two CTPC frameworks. Sections 4.3 and 4.4 summarize the results of three local structure learning methods and three local-to-global structure learning methods accelerated by the CTPC frameworks, respectively.

A. Experiment setting

1) Datasets: To evaluate the two CTPC frameworks, we use twelve standard benchmark BNs, as shown in Table I. Among them, Alarm, Child, Insurance and Hailfinder are small-size BNs. Alarm3, Child3, Insurance3 and Hailfinder3 are middle-size BNs, which are generated by tiling three copies of Alarm, Child, Insurance and Hailfinder, respectively. Hailfinder5 is generated by tiling 5 copies of Hailfinder, and Alarm10, Child10, Insurance10 are generated by tiling 10 copies of Alarm, Child, Insurance, respectively, which represent large-size BNs. For each benchmark BN, we use a group of data including 5 datasets with 5000 data instances.

2) Accelerated Methods: We compare ten BN structure learning algorithms accelerated by the two CTPC frameworks against the same original algorithms (not be accelerated), including four representative MB learning algorithms, MMMB [17], HITON-MB [22], semi-HITON-MB [31] and PCMB [23], three state-of-the-art local structure learning algorithms, PCD-by-PCD [24], MB-by-MB [25], and CMB [15], and three local-to-global structure learning algorithms, MMHC [4], MMMB-CSL [8] and HITON-MB-CSL [8]. We use * as the superscript of a BN structure learning algorithm to represent that the algorithm is accelerated by the CTPC frameworks. Accordingly, we get the following ten accelerating algorithms, MMMB*, HITON-MB*, semi-HITON-MB*, PCMB*, PCD-by-PCD*, MB-by-MB*, CMB*, MMHC*, MMMB-CSL*, and HITON-MB-CSL*.

3) Implementation Details: All BN structure learning algorithms are implemented using the open-source pyCausalFS package in Python[2]. In the experiments, G^2-test with the significance level of 0.01 is used to compute the conditional dependence between variables. All experiments are conducted on a computer with Intel(R) Core(TM) i7-8700 @3.2GHz CPU, and 16GB memory.

4) Evaluation Metrics: We evaluate the efficiency of the CTPC frameworks using the following metrics.

- *Utilization (Uti). Utilization* = $\frac{CIT - CIT^*}{CIT}$, where CIT and CIT^* represent the number of CI tests of a BN structure learning algorithm and its accelerated BN structure learning algorithm, respectively. If an algorithm performs much more redundant CI tests, the value of its Utilization will be higher.
- *Acceleration (Acc). Acceleration* = $\frac{Time}{Time^*}$, where $Time$ and $Time^*$ denote the runtime of a BN structure learning algorithm and its accelerated BN structure learning algorithm, respectively. The value of Utilization is higher, the value of Acceleration is larger.

[2]The source codes are available at https://github.com/wt-hu/Acc-pyCausalFS

TABLE I: Summary of benchmark BNs.

Network	Num. Vars	Num. Edges	Max In/Out-Degree	Min/Max |PCset|	Domain Range
Alarm	37	46	4/5	1/6	2-4
Alarm3	111	149	4/5	1/6	2-4
Alarm10	370	570	4/7	1/9	2-4
Child	20	25	2/7	1/8	2-6
Child3	60	79	3/7	1/8	2-6
Child10	200	257	2/7	1/8	2-6
Insurance	27	52	3/7	1/9	2-5
Insurance3	81	163	4/7	1/9	2-5
Insurance10	270	556	5/8	1/11	2-5
Hailfinder	56	66	4/16	1/17	2-11
Hailfinder3	168	283	5/18	1/19	2-11
Hailfinder5	280	458	5/18	1/19	2-11

TABLE II: Results of MB learning algorithms on datasets.

Datasets	Algorithms	CIT / CIT*	Time / Time*	Uti	Acc
Alarm	MMMB / MMMB*	394.02 / 245.08	4.08 / 2.84	0.38	1.44
	HITON-MB / HITON-MB*	372.45 / 258.11	3.97 / 2.81	0.31	1.41
	semi-HITON-MB / semi-HITON-MB*	338.12 / 263.06	3.33 / 2.82	0.22	1.18
	PCMB /PCMB*	2743.08 / 589.76	25.34 / 7.02	0.79	3.61
Alarm3	MMMB / MMMB*	655.46 / 501.43	6.42 / 4.93	0.24	1.30
	HITON-MB / HITON-MB*	633.72 / 512.68	6.19 / 4.92	0.19	1.26
	semi-HITON-MB / semi-HITON-MB*	591.81 / 520.79	5.52 / 4.98	0.12	1.11
	PCMB /PCMB*	6239.18 / 1116.81	52.23 / 11.77	0.82	4.44
Alarm10	MMMB / MMMB*	1663.94 / 1467.59	21.62 / 18.71	0.12	1.16
	HITON-MB / HITON-MB*	1639.67 / 1482.26	21.18 / 18.59	0.10	1.14
	semi-HITON-MB / semi-HITON-MB*	1587.57 / 1485.96	20.13 / 18.87	0.06	1.07
	PCMB /PCMB*	19760.63 / 3021.22	238.79 / 42.19	0.84	5.66
Child	MMMB / MMMB*	580.17 / 323.74	14.38 / 9.02	0.44	1.59
	HITON-MB / HITON-MB*	628.76 / 352.11	16.67 / 9.54	0.44	1.75
	semi-HITON-MB / semi-HITON-MB*	438.52 / 342.92	10.51 / 8.87	0.22	1.19
	PCMB /PCMB*	4370.74 / 756.14	102.17 / 18.63	0.83	5.49
Child3	MMMB / MMMB*	686.89 / 440.30	11.87 / 7.67	0.36	1.55
	HITON-MB / HITON-MB*	673.87 / 462.27	11.71 / 7.93	0.31	1.48
	semi-HITON-MB / semi-HITON-MB*	562.74 / 465.95	8.71 / 7.79	0.17	1.12
	PCMB /PCMB*	6549.96 / 1067.64	89.18 / 18.62	0.84	4.79
Child10	MMMB / MMMB*	1186.00 / 940.50	15.32 / 11.33	0.21	1.35
	HITON-MB / HITON-MB*	1163.29 / 951.57	15.47 / 11.40	0.18	1.36
	semi-HITON-MB / semi-HITON-MB*	1059.56 / 958.90	12.85 / 11.35	0.10	1.13
	PCMB /PCMB*	14785.19 / 2173.42	151.39 / 26.72	0.85	5.67
Insurance	MMMB / MMMB*	538.16 / 305.67	9.08 / 5.80	0.43	1.57
	HITON-MB / HITON-MB*	510.47 / 312.92	9.07 / 5.73	0.39	1.58
	semi-HITON-MB / semi-HITON-MB*	399.92 / 301.94	6.54 / 5.35	0.25	1.22
	PCMB /PCMB*	3611.31 / 668.09	151.39 / 26.72	0.82	4.59
Insurance3	MMMB / MMMB*	1213.22 / 770.40	22.12 / 15.41	0.37	1.44
	HITON-MB / HITON-MB*	1221.81 / 825.95	24.22 / 15.54	0.32	1.56
	semi-HITON-MB / semi-HITON-MB*	1018.76 / 863.91	18.46 / 16.45	0.15	1.12
	PCMB /PCMB*	14093.84 / 2114.08	200.95 / 40.59	0.85	4.95
Insurance10	MMMB / MMMB*	1966.57 / 1537.86	31.92 / 23.28	0.22	1.37
	HITON-MB / HITON-MB*	1959.07 / 1573.13	32.58 / 22.99	0.20	1.42
	semi-HITON-MB / semi-HITON-MB*	1773.75 / 1619.20	26.73 / 24.12	0.09	1.11
	PCMB /PCMB*	29977.75 / 3837.15	350.96 / 58.37	0.87	6.01
Hailfinder	MMMB / MMMB*	276.01 / 211.47	1.98 / 1.89	0.27	1.15
	HITON-MB / HITON-MB*	287.72 / 209.77	2.17 / 1.59	0.24	1.25
	semi-HITON-MB / semi-HITON-MB*	257.92 / 208.66	1.79 / 1.53	0.19	1.17
	PCMB /PCMB*	1523.00 / 338.11	10.44 / 2.83	0.78	3.69
Hailfinder3	MMMB / MMMB*	763.04 / 639.43	7.14 / 5.64	0.16	1.27
	HITON-MB / HITON-MB*	742.23 / 633.12	6.81 / 5.40	0.15	1.26
	semi-HITON-MB / semi-HITON-MB*	712.51 / 634.14	6.50 / 5.45	0.11	1.19
	PCMB /PCMB*	6006.40 / 1147.22	51.66 / 10.44	0.81	4.95
Hailfinder5	MMMB / MMMB*	1066.85 / 954.83	10.89 / 10.38	0.11	1.05
	HITON-MB / HITON-MB*	1047.66 / 948.13	11.15 / 10.17	0.10	1.10
	semi-HITON-MB / semi-HITON-MB*	1024.67 / 949.87	10.66 / 10.37	0.07	1.03
	PCMB /PCMB*	8386.50 / 1635.37	83.96 / 17.96	0.81	4.68

B. Results of MB learning algorithms

In this section, we first evaluate the efficiency of the two CTPC frameworks for accelerating four representative MB learning algorithms. We run each MB learning algorithm to learn the MBs for all variables in each BN, the average results of each variable over five datasets are summarized in Table II. From the experimental results, we have the following observations.

(1) We can see that PCMB* achieves the most significant acceleration performance among the MB learning algorithms. Specifically, PCMB* has the highest rate of Utilization (0.83 on average). Moreover, PCMB* is 4.87 times faster than PCMB on average. The main reason is that PCMB uses the two subroutines, called GetPCD and GetPC, to identify PC. The GetPCD subroutine is to find candidate PC, while the GetPC subroutine employs symmetry constraints to remove false positives from the candidate PC set to ensure the correctness of the learnt PC. Using the symmetry constraints, GetPC needs to learn the PC sets of all variables in the candidate PC set, thus, this symmetry constraints checking leads to many repeatedly computed CI tests.

(2) The performance of MMMB* is almost the same as that of HITON-MB* in terms of Acceleration metric, since MMMB and HITON-MB have the same computational complexity in theory. Specifically, the rate of Utilization of MMMB* and HITON-MB* are 0.28 and 0.24 on average, respectively. And their time efficiency boosts about 1.35 and 1.38 times on average, respectively. The difference of MMMB and HITON-MB lies in that MMMB employs MMMPC that utilizes standard forward-backward strategy to search for PC, while HITON-MB employs HITON-PC that interleaves forward step and backward step for PC learning.

(3) It can be seen that semi-HITON-MB* achieves a little improvement on efficiency on all of datasets. The explanation is that semi-HITON-MB only considers the elimination of the newly added variables before the candidate variable set becomes empty, and a full variable elimination in candidate PC set will be performed after the candidate variable set is empty. Therefore, semi-HITON-MB performs less redundant CI tests.

(4) MMMB*, HITON-MB* and semi-HITON-MB* obtain excellent acceleration on small-size BNs. Their time efficiency boosts 1.44, 1.50 and 1.17 times on average, respectively. On large-size BNs, MMMB*, HITON-MB* and semi-HIOTN-MB* achieve poor acceleration performance, and the time efficiency only boosts 1.23, 1.26 and 1.09 times, respectively. Since large-size BNs were generated by tiling multiple copies of the small-sized BNs, the number of variables on the large-size BNs is several times that of the small-size BNs. With the increase of the search scope, the MB learning algorithms thus perform more CI tests to identify MB on larger-size BNs. However, the ratio of MB of a variable to the total variables on large-size BNs is less than on small-size BNs, and thus the proportion of the redundant CI tests between PC learning on large-size BNs is less than on small-size BNs. We also note that the MB learning algorithms achieve poor acceleration performance on the Hailfinder, Hailfinder3 and Hailfinder5. It means that the MB learning algorithms learn few relevant variables on these datasets and thus perform less redundant CI tests.

C. Results of local structure learning algorithms

In this section, we compare three accelerated local structure learning algorithms against their original algorithms to evaluate the efficiency of our proposed frameworks. For each BN dataset, we select ten variables (nodes 1 to 10) to learn their local BN structure, and report the average experimental results in Table III. Based on the experimental results, we have the following observations.

(1) PCD-by-PCD* has 0.44 rate of Utilization on average, and the time efficiency of PCD-by-PCD* boosts over 1.50 times on most datasets. When learning a local BN structure,

TABLE III: Results of local structure learning algorithms

Datasets	Algorithms	CIT / CIT*	Time / Time*	Uti	Acc
Alarm	PCD-by-PCD / PCD-by-PCD*	3745.20 / 1793.95	34.71 / 21.76	0.52	1.60
	MB-by-MB / MB-by-MB*	1930.10 / 679.41	27.88 / 14.06	0.64	1.98
	CMB / CMB*	1212.34 / 418.26	13.42 / 4.96	0.66	2.70
Alarm3	PCD-by-PCD / PCD-by-PCD*	15197.52 / 8343.44	139.38 / 89.77	0.45	1.55
	MB-by-MB / MB-by-MB*	2670.94 / 1220.62	33.67 / 18.71	0.54	1.80
	CMB / CMB*	1700.96 / 659.97	17.47 / 6.74	0.61	2.59
Alarm10	PCD-by-PCD / PCD-by-PCD*	141598.30 / 77454.26	1613.10 / 998.11	0.45	1.62
	MB-by-MB / MB-by-MB*	8443.36 / 4356.77	123.31 / 65.78	0.48	1.87
	CMB / CMB*	6972.08 / 2977.08	83.51 / 34.57	0.57	2.42
Child	PCD-by-PCD / PCD-by-PCD*	2050.40 / 998.54	34.71 / 22.59	0.51	1.54
	MB-by-MB / MB-by-MB*	7483.74 / 1002.82	174.44 / 26.09	0.87	6.69
	CMB / CMB*	9485.30 / 1100.29	272.00 / 45.49	0.88	5.98
Child3	PCD-by-PCD / PCD-by-PCD*	6348.80 / 3371.21	78.23 / 51.58	0.47	1.52
	MB-by-MB / MB-by-MB*	3956.66 / 1376.92	73.75 / 32.53	0.65	2.27
	CMB / CMB*	7667.06 / 1702.09	176.91 / 52.01	0.78	3.40
Child10	PCD-by-PCD / PCD-by-PCD*	41613.20 / 22804.03	436.67 / 293.76	0.45	1.49
	MB-by-MB / MB-by-MB*	5844.76 / 2565.85	87.75 / 41.08	0.56	2.14
	CMB / CMB*	10357.00 / 3490.31	206.68 / 74.79	0.66	2.76
Insurance	PCD-by-PCD / PCD-by-PCD*	3126.40 / 1485.04	41.17 / 25.84	0.53	1.59
	MB-by-MB / MB-by-MB*	4769.72 / 1087.50	84.22 / 24.66	0.77	3.42
	CMB / CMB*	5864.88 / 1237.49	112.78 / 30.82	0.79	3.66
Insurance3	PCD-by-PCD / PCD-by-PCD*	18559.80 / 9131.42	302.91 / 188.65	0.51	1.61
	MB-by-MB / MB-by-MB*	11336.30 / 3049.46	252.89 / 75.93	0.70	3.33
	CMB / CMB*	13622.22 / 3160.35	1075.13 / 203.72	0.81	5.28
Insurance10	PCD-by-PCD / PCD-by-PCD*	48431.80 / 30996.35	665.85 / 451.65	0.36	1.47
	MB-by-MB / MB-by-MB*	21601.50 / 6329.24	373.67 / 106.85	0.71	3.50
	CMB / CMB*	23196.22 / 6425.35	854.10 / 170.85	0.72	5.00
Hailfinder	PCD-by-PCD / PCD-by-PCD*	700.40 / 498.68	5.70 / 3.90	0.29	1.30
	MB-by-MB / MB-by-MB*	660.30 / 277.99	4.99 / 2.41	0.58	2.07
	CMB / CMB*	285.32 / 185.74	2.10 / 1.47	0.35	1.43
Hailfinder3	PCD-by-PCD / PCD-by-PCD*	18188.28 / 11003.91	176.02 / 116.79	0.40	1.51
	MB-by-MB / MB-by-MB*	2982.98 / 1387.08	29.39 / 13.83	0.54	2.13
	CMB / CMB*	2171.78 / 1166.24	24.17 / 12.36	0.46	1.96
Hailfinder5	PCD-by-PCD / PCD-by-PCD*	45132.42 / 28072.37	508.96 / 334.91	0.38	1.52
	MB-by-MB / MB-by-MB*	5550.44 / 2347.84	60.17 / 25.92	0.58	2.32
	CMB / CMB*	1874.76 / 1209.22	23.35 / 13.78	0.36	1.69

TABLE IV: Results of local-to-global structure learning algorithms

Datasets	Algorithms	CIT / CIT*	Time / Time*	Uti	Acc
Alarm	MMHC / MMHC*	3840.20 / 1812.57	62.97 / 51.58	0.53	1.22
	MMMB-CSL / MMMB-CSL*	17983.00 / 3003.20	283.95 / 81.38	0.83	3.49
	HITON-MB-CSL / HITON-MB-CSL*	17061.40 / 3054.00	260.29 / 73.68	0.82	3.53
Alarm3	MMHC / MMHC*	19394.20 / 9386.80	415.71 / 348.56	0.52	1.19
	MMMB-CSL / MMMB-CSL*	83809.80 / 13661.00	1177.38 / 311.14	0.84	3.78
	HITON-MB-CSL / HITON-MB-CSL*	80702.00 / 13638.60	1168.68 / 300.96	0.83	3.88
Alarm10	MMHC / MMHC*	167553.00 / 82603.60	5005.56 / 4109.53	0.51	1.22
	MMMB-CSL / MMMB-CSL*	662517.60 / 101365.20	8980.29 / 1921.54	0.85	4.67
	HITON-MB-CSL / HITON-MB-CSL*	650769.80 / 100869.30	10614.10 / 2167.69	0.85	4.90
Child	MMHC / MMHC*	2080.80 / 1000.90	45.59 / 33.58	0.52	1.36
	MMMB-CSL / MMMB-CSL*	14243.20 / 1880.10	399.97 / 70.63	0.87	5.66
	HITON-MB-CSL / HITON-MB-CSL*	15154.20 / 1925.60	289.58 / 66.69	0.87	4.34
Child3	MMHC / MMHC*	8669.20 / 4152.50	235.44 / 194.33	0.52	1.21
	MMMB-CSL / MMMB-CSL*	54002.40 / 9288.40	1340.12 / 354.77	0.83	3.78
	HITON-MB-CSL / HITON-MB-CSL*	51515.60 / 8551.60	1247.38 / 289.58	0.83	4.31
Child10	MMHC / MMHC*	57114.00 / 25086.20	1951.76 / 1674.08	0.56	1.17
	MMMB-CSL / MMMB-CSL*	281409.00 / 46151.10	5495.38 / 1383.43	0.84	3.97
	HITON-MB-CSL / HITON-MB-CSL*	271600.60 / 43456.10	5068.35 / 1173.73	0.84	4.32
Insurance	MMHC / MMHC*	3224.60 / 1499.40	52.85 / 38.50	0.54	1.37
	MMMB-CSL / MMMB-CSL*	17679.40 / 2634.20	358.19 / 83.68	0.85	4.28
	HITON-MB-CSL / HITON-MB-CSL*	16954.20 / 2678.80	357.57 / 86.85	0.84	4.12
Insurance3	MMHC / MMHC*	18808.20 / 9122.00	575.77 / 464.41	0.52	1.24
	MMMB-CSL / MMMB-CSL*	125280.80 / 19794.40	3284.36 / 785.87	0.84	4.18
	HITON-MB-CSL / HITON-MB-CSL*	124571.60 / 19557.80	3351.13 / 747.09	0.84	4.49
Insurance10	MMHC / MMHC*	112800.60 / 55497.90	4587.80 / 3812.39	0.51	1.20
	MMMB-CSL / MMMB-CSL*	613665.40 / 87754.20	14191.32 / 2959.05	0.86	4.80
	HITON-MB-CSL / HITON-MB-CSL*	605890.40 / 85430.60	13993.64 / 2707.38	0.86	5.17
Hailfinder	MMHC / MMHC*	5538.40 / 2531.00	55.49 / 38.44	0.54	1.44
	MMMB-CSL / MMMB-CSL*	18601.60 / 3478.50	226.93 / 59.41	0.81	3.82
	HITON-MB-CSL / HITON-MB-CSL*	17933.80 / 3425.40	181.52 / 49.76	0.81	3.65
Hailfinder3	MMHC / MMHC*	39070.80 / 18675.80	543.18 / 404.65	0.52	1.34
	MMMB-CSL / MMMB-CSL*	145836.40 / 26250.60	2079.24 / 602.94	0.82	3.45
	HITON-MB-CSL / HITON-MB-CSL*	141890.80 / 25682.20	1875.19 / 519.24	0.82	3.61
Hailfinder5	MMHC / MMHC*	355355.60 / 174835.00	6943.72 / 5023.81	0.51	1.38
	MMMB-CSL / MMMB-CSL*	1233608.00 / 204779.00	16907.36 / 3637.68	0.83	4.65
	HITON-MB-CSL / HITON-MB-CSL*	1220205.60 / 202554.00	16696.66 / 3638.24	0.83	4.72

PCD-by-PCD recursively identifies the PC for learning a local BN skeleton and orienting edges, the process of calling a PC learning subroutine results in performing many redundant CI tests. Thus, the larger the local structure of PCD-by-PCD* expands, the higher its Utilization is achieved.

(2) We can observe that MB-by-MB* is significantly superior to PCD-by-PCD* on all of datasets in terms of both Utilization and Acceleration metric. Specifically, MB-by-MB* has 0.64 rate of Utilization on average, and its average time efficiency boosts about 2.74 times. The explanation is that MB-by-MB is developed from PCD-by-PCD, and it employs MB learning algorithms to learn a local BN structure instead of using PC learning algorithms.

(3) Among these algorithms, the acceleration performance of CMB* is better than PCD-by-PCD* and MB-by-MB* on most datasets. Since CMB employs MB learning algorithms to construct a local skeleton, and orients edges by tracking independence relationship changes in MB of a target variable, then sequentially constructs local structure by the learnt MBs of the variables connected to the target variable. Therefore, both the process of MB learning and tracking independence relationship changes lead to more redundant CI tests.

D. Results of local-to-global structure learning algorithms

In this section, to further evaluate the efficiency of the CTPC frameworks on BN global learning algorithms, we select three local-to-global BN learning algorithms, MMHC, MMMB-CSL and HITON-MB-CSL. Table IV summarizes the experimental results. In addition, MMMB-CSL and HION-MB-CSL are variants of GSBN algorithms, the main difference is that they employ MMMB and HITON-MB to learn MB for constructing global skeleton instead of using GSMB, respectively.

According to the experimental results, we have the following observations.

(1) Table IV shows that the CTPC frameworks achieve a significant acceleration on local-to-global structure learning algorithms. Since learning a PC set of a variable is the key to local-to-global BN learning algorithms, and the efficiency of the algorithms depends on the PC learning subroutine. More specifically, when learning a global BN structure, all of the algorithms start with learning the PC of all variables to construct global skeleton, and thus they perform many redundant CI tests.

(2) The average Utilization of MMHC* is 0.53, since MMHC employs MMPC to learn PC of all variables to construct a global skeleton and thus have a high Utilization. The CTPC frameworks only speed up MMHC* 1.28 times on average in terms of the Acceleration metric. The reason is that for edge orientations, MMHC uses score-based methods instead of using CI tests and this process takes up the main execution time.

(3) As for MMMB-CSL* and HITON-MB-CSL*, we can see that they have a higher cache utilization and achieve better acceleration performance than MMHC* in Table IV. Since they employ MMMB and HITON-MB to learn MB of all variables, respectively, and a MB learning subroutine performs more redundant CI tests than a PC learning subroutine. Moreover, both MMMB-CSL and HITON-MB-CSL use CI tests to orient edges after constructing a global skeleton, and MMMB-CSL* and HITON-MB-CSL* can also accelerate the process of edge orientations.

V. Conclusion

In this paper, we design a CI test cache table for reducing redundant CI-tests in constraint-based BN structure learning algorithms. Based on the CI test cache table, we propose two CTPC frameworks to accelerate PC learning for BN structure learning algorithms. The two frameworks can be instantiated by any existing constraint-based BN structure learning algorithms. Using a series of standard benchmark BNs, we compare ten representative BN structure learning algorithms with their corresponding accelerated algorithms by the CTPC frameworks. The experimental results demonstrate that the CTPC frameworks achieve significant acceleration performance on existing BN structure learning algorithms without sacrificing accuracy.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (under Grant 61876206), and the Open Project Foundation of Intelligent Information Processing Key Laboratory of Shanxi Province (under grant CICIP2020003).

References

[1] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[2] J. Runge, S. Bathiany, E. Bollt, G. Camps-Valls, D. Coumou, E. Deyle, C. Glymour, M. Kretschmer, M. D. Mahecha, J. Muñoz-Marí *et al.*, "Inferring causation from time series in earth system sciences," *Nature communications*, vol. 10, no. 1, pp. 1–13, 2019.

[3] M. Prosperi, Y. Guo, M. Sperrin, J. S. Koopman, J. S. Min, X. He, S. Rich, M. Wang, I. E. Buchan, and J. Bian, "Causal inference and counterfactual prediction in machine learning for actionable healthcare," *Nature Machine Intelligence*, vol. 2, no. 7, pp. 369–375, 2020.

[4] I. Tsamardinos, L. E. Brown, and C. F. Aliferis, "The max-min hill-climbing bayesian network structure learning algorithm," *Machine learning*, vol. 65, no. 1, pp. 31–78, 2006.

[5] K. Yu, L. Liu, J. Li, W. Ding, and T. D. Le, "Multi-source causal feature selection," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 9, pp. 2240–2256, 2019.

[6] Y. Yu, J. Chen, T. Gao, and M. Yu, "Dag-gnn: Dag structure learning with graph neural networks," in *International Conference on Machine Learning*, 2019, pp. 7154–7163.

[7] S. Yang, H. Wang, K. Yu, F. Cao, and X. Wu, "Towards efficient local causal structure learning," *IEEE Transactions on Big Data*, 10.1109/TBDATA.2021.3062937, 2021.

[8] K. Yu, X. Guo, L. Liu, J. Li, H. Wang, Z. Ling, and X. Wu, "Causality-based feature selection: Methods and evaluations," *ACM Computing Surveys*, vol. 53, no. 5, pp. 111:1–111:36, 2020.

[9] Z. Ling, K. Yu, H. Wang, L. Li, and X. Wu, "Using feature selection for local causal structure learning," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. PP, no. 99, pp. 1–11, 2020.

[10] D. M. Chickering, "Optimal structure identification with greedy search," *Journal of machine learning research*, vol. 3, no. Nov, pp. 507–554, 2002.

[11] P. Spirtes, C. N. Glymour, R. Scheines, and D. Heckerman, *Causation, prediction, and search*. MIT press, 2000.

[12] D. Colombo and M. H. Maathuis, "Order-independent constraint-based causal structure learning," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3741–3782, 2014.

[13] J. Zhang, T. D. Le, L. Liu, and J. Li, "Inferring and analyzing module-specific lncrna–mrna causal regulatory networks in human cancer," *Briefings in bioinformatics*, vol. 20, no. 4, pp. 1403–1419, 2019.

[14] Z. Ling, K. Yu, H. Wang, L. Liu, W. Ding, and X. Wu, "Bamb: A balanced markov blanket discovery approach to feature selection," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 5, pp. 1–25, 2019.

[15] T. Gao and Q. Ji, "Local causal discovery of direct causes and effects," in *Advances in Neural Information Processing Systems*, 2015, pp. 2512–2520.

[16] C. F. Aliferis, A. R. Statnikov, I. Tsamardinos, S. Mani, and X. D. Koutsoukos, "Local causal and markov blanket induction for causal discovery and feature selection for classification part I: algorithms and empirical evaluation," *J. Mach. Learn. Res.*, vol. 11, pp. 171–234, 2010.

[17] I. Tsamardinos, C. F. Aliferis, and A. Statnikov, "Time and sample efficient discovery of markov blankets and direct causal relations," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 673–678.

[18] D. Margaritis and S. Thrun, "Bayesian network induction via local neighborhoods," in *Advances in neural information processing systems*, 2000, pp. 505–511.

[19] I. Tsamardinos and C. F. Aliferis, "Towards principled feature selection: Relevancy, filters and wrappers," in *International Workshop on Artificial Intelligence and Statistics*. PMLR, 2003, pp. 300–307.

[20] I. Tsamardinos, C. F. Aliferis, A. R. Statnikov, and E. Statnikov, "Algorithms for large scale markov blanket discovery." in *FLAIRS conference*, vol. 2, 2003, pp. 376–380.

[21] S. Yaramakala and D. Margaritis, "Speculative markov blanket discovery for optimal feature selection," in *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), 27-30 November 2005, Houston, Texas, USA*. IEEE Computer Society, 2005, pp. 809–812.

[22] C. F. Aliferis, I. Tsamardinos, and A. Statnikov, "Hiton: a novel markov blanket algorithm for optimal variable selection," in *AMIA annual symposium proceedings*, vol. 2003. American Medical Informatics Association, 2003, p. 21.

[23] J. M. Pena, R. Nilsson, J. Björkegren, and J. Tegnér, "Towards scalable and data efficient learning of markov boundaries," *International Journal of Approximate Reasoning*, vol. 45, no. 2, pp. 211–232, 2007.

[24] J. Yin, Y. Zhou, C. Wang, P. He, C. Zheng, and Z. Geng, "Partial orientation and local structural learning of causal networks for prediction," in *Causation and Prediction Challenge*, 2008, pp. 93–105.

[25] C. Wang, Y. Zhou, Q. Zhao, and Z. Geng, "Discovering and orienting the edges connected to a target variable in a dag via a sequential local learning approach," *Computational Statistics & Data Analysis*, vol. 77, pp. 252–266, 2014.

[26] C. Meek, "Causal inference and causal explanation with background knowledge," in *UAI '95: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence, Montreal, Quebec, Canada, August 18-20, 1995*, P. Besnard and S. Hanks, Eds. Morgan Kaufmann, 1995, pp. 403–410.

[27] P. Spirtes and C. Glymour, "An algorithm for fast recovery of sparse causal graphs," *Social science computer review*, vol. 9, no. 1, pp. 62–72, 1991.

[28] T. Niinimaki and P. Parviainen, "Local structure discovery in bayesian networks," in *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14-18, 2012*, N. de Freitas and K. P. Murphy, Eds. AUAI Press, 2012, pp. 634–643.

[29] T. Gao, K. Fadnis, and M. Campbell, "Local-to-global bayesian network structure learning," in *International Conference on Machine Learning*, 2017, pp. 1193–1202.

[30] T. Gao and Q. Ji, "Efficient markov blanket discovery and its application," *IEEE transactions on cybernetics*, vol. 47, no. 5, pp. 1169–1179, 2017.

[31] C. F. Aliferis, A. Statnikov, I. Tsamardinos, S. Mani, and X. D. Koutsoukos, "Local causal and markov blanket induction for causal discovery and feature selection for classification part i: Algorithms and empirical evaluation." *Journal of Machine Learning Research*, vol. 11, no. 1, 2010.