Accelerating Learning Bayesian Network Structures by Reducing Redundant CI Tests

Wentao Hu[1,2]; Kui Yu[1,2]; Shuai Yang[1,2]; Xianjie Guo[1,2];Hao Wang[1,2]
[1] *Key Laboratory of Knowledge Engineering with Big Data (Hefei University of Technology), Ministry of Education, Hefei, China*
[2] *School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, China*
{yukui,jsjxwangh}@hfut.edu.cn; {wentaohu,yangs,xianjieguo}@mail.hfut.edu.cn

I. Examples instantiated from frameworks CTPC-s and CTPC-I

In this section, we give the examples of how the two CTPC frameworks are instantiated by the two well-established existing PC learning algorithms, MMPC and HITON-PC without sacrificing accuracy. MMPC* is instantiated from the CTPC-S framework and HITON-PC* is instantiated from the CTPC-I framework. The MMMB* and HITION-MB* algorithms employ MMPC* and HITON-PC* for learning the PC set of a varible respectively instead of MMPC and HITION-PC. The detailed descriptions are as follows.

Algorithm 1: The MMPC* algorithm

Require: variable set V, target variable T, CItest cache table CCT, threshold value δ
Ensure: CPC // the candidate PC of T

1: Initialization: $CPC = \emptyset$, $R \leftarrow \{\forall X \in V \wedge X \not\perp\!\!\!\perp T \mid \emptyset\}$
2: // Forward step: Adding variables to CPC
3: **repeat**
4: **for** each $X \in R$ **do**
5: $Dep[X] = \arg\min_{S \subseteq CPC}$ CCT_select(X,T,S,CCT);
6: **end for**
7: $X_{max} = \arg\max_{X \in R} Dep(X)$;
8: **if** $Dep[X_{max}] \geq \delta$ **then**
9: $CPC = CPC \cup X_{max}$ and $R = R \backslash X_{max}$;
10: **end if**
11: **until** CPC does not change;
12: // Backward step: Removing variables from CPC
13: **for** each $Y \in CPC$ **do**
14: **if** $\exists S \subseteq CPC$ such that CCT_select$(Y,T,S,CCT) < \delta$ **then**
15: $CPC = CPC \backslash Y$;
16: **end if**
17: **end for**
18: **return** CPC;

A. MMPC* instantiated from CTPC-s framework

The detailed description of MMPC* is summarized in Algorithm 1. MMPC* first performs the forward step to obtain the candidate PC, then carries out the backward step to remove false positives. Given a target variable T, MMPC* starts with an empty candidate PC set (i.e., $CPC = \emptyset$) and a candidate set R that contains all variables which are dependent on the target T conditioning on an empty set.

At each iteration in the forward step, for each variable $X \in R$, line 5 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 chooses the minimum dependency value as the dependency value of X and T, denote as $Dep[X]$. At line 7, MMPC* selects the variable X that has the maximum relevancy value among the variables ($X \in R$) and denotes it as X_{max}, if $Dep[X_{max}]$ is greater than a given threshold value δ, MMPC* adds X to CPC and removes it from R. In addition, the threshold value δ denotes the selection criteria of an algorithm, if the dependency value of a variable and T is lower than δ, the variable is conditionally independent of T and discarded. Otherwise it is retained. The forward step will be terminated until CPC does not change, that is, no new variables in R are added to CPC.

Then, MMPC* performs the backward step to remove all false positives from CPC. At each iteration, MMPC* checks each variable Y in CPC on all possible subsets $S \subseteq CPC \backslash Y$ by performing CCT_select(Y,T,S,CCT), if there exists a set S such that the dependency value of Y and T conditioning on S is lower than δ, MMPC* considers Y as a false positive and removes it from CPC, otherwise Y is retained. The backward step will be terminated until all the variables in CPC are not removed.

B. HITON-PC* instantiated from CTPC-I framework

In Algorithm 2, HITON-PC* performs the forward step and backward step alternatively. Given a target variable T, HITON-PC* starts with an empty candidate PC set (i.e., $CPC = \emptyset$) and a candidate set R that contains all variables which are dependent on the target T given an empty set.

At each iteration, for each variable $X \in R$, HITON-PC* performs CCT_select$(T,X,\emptyset,CCT)$ to obtain the dependency value of T and X conditioning on an empty set. At lines 7-10, HITON-PC* selects the variable X in R which has the maximum relevancy value with T, then adds the variable X into CPC and removes it from R. When a new variable is added into CPC, HITON-PC* immediately triggers the backward step to remove all false positives from current CPC. Specifically, for each variable $Y \in CPC$, HITON-PC*

Algorithm 2: The HITON-PC* algorithm

Require: variable set V, target variable T,
CItest cache table CCT, threshold value δ
Ensure: CPC // the candidate PC of T
1: Initialization: $CPC = \emptyset$, $R \leftarrow \{\forall X \in V \wedge X \not\perp\!\!\!\perp T \mid \emptyset\}$;
2: **repeat**
3: // Forward step: Adding variables to CPC
4: **for** each $X \in R$ **do**
5: $Dep[X] =$ CCT_select$(X, T, \emptyset, CCT)$;
6: **end for**
7: $X_{max} = \arg\max_{X \in R} Dep[X]$;
8: $CPC = CPC \cup X_{max}$ and $R = R \backslash X_{max}$;
9: // Backward step: Removing variables from CPC
10: **for** each $Y \in CPC$ **do**
11: **if** $\exists S \subseteq CPC$ such that
CCT_select$(Y, T, S, CCT) < \delta$ **then**
12: $CPC = CPC \backslash Y$;
13: **end if**
14: **end for**
15: **until** CPC does not change;
16: **return** CPC;

implements CCT_select(Y, T, S, CCT) to get the dependency value of Y and T on all possible subsets $S \subseteq CPC \backslash Y$, if there exists a set S such that the dependency value of Y and T conditioning on S is lower than a given threshold value δ, Y will be removed from CPC and never added again. After executing the backward step, HITON-PC* triggers the forward step again to add the next variable. The interleaving execution of HITON-PC* terminates until no new variables are added to CPC.

C. Analysis of instantiation

As described above, the two accelerated PC learning methods do not violate the idea of the original MMPC and HITON-PC. Specifically, the main differences of MMPC* from its original MMPC are at lines 5 and 14 of Algorithm 1, instead of computing the CI test repeatedly to get the dependency value, MMPC* employs the CCT_select algorithm to retrieve and obtain a CI test from the cache table, which improves the efficiency and does not change the execution process. If the CI test in the cache table, the result of the CI test (i.e. the dependency value) is obtained from the table instead of computing this CI test again. If not, the CI test is computed using the CI-Test function and is stored in the table. Thus, MMPC* still keeps the max-min greedy search strategy of the origin MMPC to identify the best variable in the forward step, and holds the elimination strategy in the backward step, that is, all and only variables that become independent of the target variable T given any subset of CPC are discarded and never considered again.

Similarly, the main differences of HITON-PC* from its original HITON-PC are at lines 5 and 11 of Algorithm 2, where HITON-PC* employs the CCI_select algorithm to get the dependency value instead of computing the CI test repeatedly, so that more efficient search can be exploited to replace the complex computations. HITON-PC* still obeys the greedy search strategy of its original HITON-PC which adds the highest association with T conditioning on an empty set to CPC, and holds the elimination strategy that it immediately removes false positives from the current CPC after a variable is added to CPC.

In summary, the instantiated PC learning methods are guaranteed to keep its original idea, and reduce the computational cost of redundant CI tests. The CTPC frameworks can significantly accelerate existing BN structure learning algorithms without sacrificing accuracy.