Journal of Machine Learning Research 21 (2020) 1-31 Submitted 4/19; Published 8/20

Learning Big Gaussian Bayesian Networks: Partition, Estimation and Fusion

Jiaying Gu GUJY.LOLA@GMAIL.COM
Qing Zhou ZHOU@STAT.UCLA.EDU
Department of Statistics
University of California, Los Angeles
Los Angeles, CA 90095, USA

Editor: XuanLong Nguyen

Abstract

Structure learning of Bayesian networks has always been a challenging problem. Nowadays, massive-size networks with thousands or more of nodes but fewer samples frequently appear in many areas. We develop a divide-and-conquer framework, called partition-estimation-fusion (PEF), for structure learning of such big networks. The proposed method first partitions nodes into clusters, then learns a subgraph on each cluster of nodes, and finally fuses all learned subgraphs into one Bayesian network. The PEF method is designed in a flexible way so that any structure learning method may be used in the second step to learn a subgraph structure as either a DAG or a CPDAG. In the clustering step, we adapt hierarchical clustering to automatically choose a proper number of clusters. In the fusion step, we propose a novel hybrid method that sequentially adds edges between subgraphs. Extensive numerical experiments demonstrate the competitive performance of our PEF method, in terms of both speed and accuracy compared to existing methods. Our method can improve the accuracy of structure learning by 20% or more, while reducing running time up to two orders-of-magnitude.

Keywords: Bayesian network, conditional independence, directed acyclic graph, divide-and-conquer, structure learning

1. Introduction

The structure of a Bayesian network for p random variables $X_1, \ldots, X_p$ is represented by a directed acyclic graph (DAG) $\mathcal{G} = (V, E)$. The node set $V = \{1, \ldots, p\}$ represents the set of random variables, and $E = \{(j, i) \in V \times V : j \to i\}$ is the edge set, where $j \to i$ is a directed edge in $\mathcal{G}$. Let $\Pi_i^{\mathcal{G}} = \{j \in V : (j, i) \in E\}$ denote the parent set of node i. The joint probability density function f of $(X_1, \ldots, X_p)$ can be factorized according to the structure of $\mathcal{G}$:

$$f(x_1, \ldots, x_p) = \prod_{i=1}^{p} f(x_i | \pi_i), \tag{1}$$

where $f(x_i|\pi_i)$ is the conditional probability density (CPD) of X_i given $\Pi_i^{\mathcal{G}} = \pi_i$. Hereafter, we may use X_i and the node i interchangeably.

The problem of structure learning of Bayesian networks from data has been an active research area due to its wide applications in machine learning, statistical modeling, and

©2020 Jiaying Gu and Qing Zhou.

License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/. Attribution requirements are provided at http://jmlr.org/papers/v21/19-318.html.

causal inference (Spirtes et al., 1993; Pearl, 2000). There are a few different approaches to this problem. The first one is the constraint-based approach, which determines the existence of edges by a sequence of conditional independence tests. The PC algorithm (Spirtes and Glymour, 1991) and its further developments (Tsamardinos et al., 2003; Kalisch and Bühlmann, 2007; Colombo and Maathuis, 2014) are typical examples of constrained-based methods. The second category is so-called score-based learning, which searches for a graphical structure that optimizes a certain scoring function, such as early works in Heckerman et al. (1995); Geiger and Heckerman (1994); Chickering (2002b); Chickering and Meek (2002). Recently, fast algorithms have been developed to handle large and high-dimensional datasets (Fu and Zhou, 2013; Xiang and Kim, 2013; Aragam and Zhou, 2015; Ramsey et al., 2017; Zheng et al., 2018; Yuan et al., 2019). In addition, there are also hybrid methods that combine the above two approaches. These methods first restrict the search space using a constraint-based method, and then learn the DAG structure by optimizing a score over the restricted search space (Tsamardinos et al., 2006; Gámez et al., 2011; Gasse et al., 2012).

Despite these great efforts, structure learning of Bayesian networks remains challenging, especially for datasets with a large number of variables. The DAG space grows super-exponentially in the number of nodes p (Robinson, 1977), and learning Bayesian networks has been shown to be an NP-hard problem in general (Chickering et al., 2004). Nowadays, it is common to generate and collect data from thousands of variables or more. As p increases, however, many of the aforementioned methods slow down dramatically and become much less accurate, making them incompetent for large datasets. This motivates our development of a divide-and-conquer method that can learn massive-size Bayesian networks efficiently and accurately. Our method consists of three steps, Partition, Estimation and Fusion (PEF for short):

1. *P-step*: Partition the p nodes into clusters based on a modified hierarchical clustering algorithm.

2. *E-step*: Apply an existing structure learning algorithm to estimate a subgraph on each cluster of nodes.

3. *F-step*: Develop a new hybrid method to merge the estimated subgraphs into a full DAG on all nodes.

Note that the number of nodes in a cluster is usually much smaller than p. This greatly speeds up structure learning in the estimation step, as most algorithms scale at least as $O(p^k)$ for some $k \geq 2$, e.g. Kalisch and Bühlmann (2007). Moreover, this step can be parallelized in an obvious way, leading to further improvement in computational efficiency. The hybrid method in the fusion step first uses statistical tests to generate a candidate set of node pairs between estimated subgraphs, and then minimizes a modified BIC score by adding between-subgraph edges and updating within-subgraph edges. Since our conditional independence tests are performed based on the structure of subgraphs, the number of tests needed for our method is substantially smaller than a constraint-based method on a p-node problem. Our method is designed with maximum flexibility. The user can apply any structure learning algorithm in the second step as long as it outputs a partially directed acyclic graphs (PDAG), including DAGs and completed PDAGs (CPDAGs) as special cases.

Our PEF method works very well on Bayesian networks with a block structure to some degree, having relatively weak connections between subgraphs. It is quite common for a large network to show such a block structure, due to the underlying heterogeneity among the nodes (Chin et al., 2015; Decelle et al., 2011; Abbe et al., 2016). From extensive numerical comparisons with existing methods, we find that the PEF method can significantly improve the accuracy of structure learning of Bayesian networks, while reducing computing time substantially, up to two orders-of-magnitude for big graphs. On the other hand, our numerical results show that, even for big networks with no block structure, our PEF method can still significantly reduce computing time without losing much accuracy.

The remaining part of this paper is organized as follows. Section 2 contains a necessary background review for our method. Section 3 describes the partition and the estimation steps of the PEF method, while Section 4 develops the fusion step in detail. Section 5 provides numerical results of our method on real networks in comparison to other DAG learning algorithms. Section 6 summarizes this work with a discussion of future directions. Some technical details are deferred to an Appendix.

2. Review of Bayesian networks

In this section, we briefly review some concepts about Bayesian networks that are most relevant to our method. The joint distribution P that factorizes according to the DAG structure of a Bayesian network as in (1) satisfies so-called Markov properties (Lauritzen, 1996). Let $X, Y \in V$ and $\boldsymbol{Z} \subseteq V \setminus \{X, Y\}$. If $\boldsymbol{Z}$ d-separates X from Y in DAG $\mathcal{G}$, then the random variables X and Y are conditionally independent given $\boldsymbol{Z}$. Using $\mathcal{D}_{\mathcal{G}}(X; Y|\boldsymbol{Z})$ to denote d-separation in $\mathcal{G}$ and $\mathcal{I}_P(X; Y|\boldsymbol{Z})$ for conditional independence in P, the above (global) Markov property says that $\mathcal{D}_{\mathcal{G}}(X; Y|\boldsymbol{Z}) \Rightarrow \mathcal{I}_P(X; Y|\boldsymbol{Z})$.

2.1. Faithfulness

Note that the implication in a Markov property goes only in one direction. To estimate the structure of a DAG, we need to infer edges from conditional independence statements learned from data, which often requires the faithfulness assumption (Spirtes et al., 1993) to build up the equivalence between the two.

Definition 1 (Faithfulness) *Suppose $\mathcal{G}$ is a DAG and P is a joint probability distribution over a set V of random variables. Then $\mathcal{G}$ and P are faithful to each other if and only if*

$$\mathcal{I}_P(X; Y|\boldsymbol{Z}) \Leftrightarrow \mathcal{D}_{\mathcal{G}}(X; Y|\boldsymbol{Z})$$

for any $X, Y \in V$ and $\boldsymbol{Z} \subseteq V \setminus \{X, Y\}$.

If $(\mathcal{G}, P)$ satisfies the faithfulness assumption, we can use conditional independence (CI) test to infer d-separation in $\mathcal{G}$. Theorem 2 provides a useful criterion to determine the existence of an edge using CI tests.

Theorem 2 (Spirtes et al. (1993)) *Suppose $(\mathcal{G}, P)$ satisfies the faithfulness assumption. Then there is no edge between a pair of nodes $X, Y \in V$ if and only if there exists a subset $\boldsymbol{Z} \subseteq V \setminus \{X, Y\}$ such that $\mathcal{I}_P(X; Y|\boldsymbol{Z})$.*

Consequently, faithfulness is commonly assumed in the development of many structure learning algorithms, especially constraint-based and hybrid methods, such as the PC algorithm and the MMHC algorithm (Spirtes et al., 1993; Tsamardinos et al., 2006).

2.2. Markov equivalence

Multiple DAGs may imply the same set of d-separations, and thus encode the same set of CI statements, if they are *Markov equivalent*:

Definition 3 (Markov equivalence) *Two DAGs $\mathcal{G}$ and $\mathcal{G}'$ defined on the same set of nodes V are Markov equivalent if $\mathcal{D}_{\mathcal{G}}(X;Y|\boldsymbol{Z}) \Leftrightarrow \mathcal{D}_{\mathcal{G}'}(X;Y|\boldsymbol{Z})$ for any $X, Y \in V$ and $\boldsymbol{Z} \subseteq V \setminus \{X, Y\}$.*

As shown by Verma and Pearl (1990), two DAGs are Markov equivalent if and only if they have the same skeletons and the same v-structures. A v-structure is a triplet $\{i, j, k\} \subseteq V$ of the form $i \to k \leftarrow j$, where i and j are not adjacent, and the node k is called an *uncovered collider*. DAGs that are Markov equivalent form an equivalence class in the space of DAGs. A Markov equivalence class can be uniquely represented by a CPDAG (Chickering, 2002a). An edge that must have the same orientation in all DAGs in an equivalent class is called a compelled edge, and otherwise it is reversible. The CPDAG for an equivalence class consists of directed edges for all compelled edges and undirected edges for all reversible ones.

Since DAGs in the same equivalence class cannot be distinguished from observational data, some structure learning algorithms (Chickering and Meek, 2002; Spirtes et al., 1993) output a CPDAG, instead of a particular DAG in the equivalence class. Thus, depending on which structure learning algorithm is used, the estimation step of our PEF method may output a DAG, a CPDAG, or in general a PDAG, from each cluster of nodes. The fusion step will merge these PDAGs into a full DAG as the final estimate.

2.3. Gaussian Bayesian networks

In this paper, we focus on Gaussian Bayesian networks for continuous data, in which the conditional distributions are specified by a linear structural equation model,

$$X_j = \sum_{i \in \Pi_j^{\mathcal{G}}} \beta_{ij} X_i + \varepsilon_j, \qquad j = 1, \ldots, p, \tag{2}$$

where $\varepsilon_j \sim \mathcal{N}(0, \sigma_j^2)$ and $\beta_{ij} \neq 0$ if and only if $i \in \Pi_j^{\mathcal{G}}$. Let $B = (\beta_{ij})_{p \times p}$ be an edge coefficient matrix, which can be regarded as a weighted adjacency matrix for the DAG $\mathcal{G}$, with β_{ij} being the weight for the edge $i \to j$. Put $\Omega = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_p^2)$ as a $p \times p$ diagonal matrix of error variances. Then the joint distribution of $(X_1, \ldots, X_p)$ defined by (2) is a multivariate Gaussian distribution $\mathcal{N}_p(0, \Sigma)$ with covariance matrix $\Sigma = (I - B)^{-\mathsf{T}} \Omega (I - B)^{-1}$, where I denotes the identity matrix.

Suppose we have observed an iid sample of size n, $\mathbf{x} = [\mathbf{x}_1 | \ldots | \mathbf{x}_p] \in \mathbb{R}^{n \times p}$, from a Gaussian Bayesian network parameterized by (B, Ω). Let B_j be the jth column of B. Then

the log-likelihood under this model is

$$\ell(B,\Sigma)=\sum_{j=1}^{p}\left[-\frac{n}{2}\log(\sigma_j^2)-\frac{1}{2\sigma_j^2}\|\mathbf{x}_j-\mathbf{x}B_j\|^2\right], \tag{3}$$

which forms the basis for score-based learning, subject to certain regularization or constraint on model complexity, e.g. the total number of edges in the DAG. For Gaussian random variables, conditional independence is equivalent to zero partial correlation, which is completely determined by the covariance matrix Σ. Consequently, in constraint-based methods, CI tests are performed based on sample partial correlations; see Appendix A.1 for a brief description.

3. Partition and estimation

In this section, we describe the first two steps of our PEF method. Besides some modifications to meet our specific needs in learning large networks, these two steps follow quite standard methods in clustering and structure learning. We will devote the entire Section 4 to the fusion step.

3.1. Partition

As we have mentioned in the introduction, the first step (P-step) of our method is to partition nodes into clusters. Each node is associated with a data column $\mathbf{x}_j \in \mathbb{R}^n$ for $j=1,\ldots,p$. Let C_i, $i=1,\ldots,k$, be the k clusters generated by the P-step, and $S_i=|C_i|$ the size of the ith cluster. Accordingly, the underlying DAG $\mathcal{G}$ is cut into k subgraphs. Let s_w be the number of edges of $\mathcal{G}$ within a subgraph, and s_b the number of edges between subgraphs. In other words, s_b is the number of edges in the partition-cut with respect to the k clusters, which may be recovered later by the fusion step of our algorithm. In general, we want to control s_b to a small value so that our recovery of the DAG structure will be more accurate. On the other hand, we wish that k is quite large and the cluster size is as uniform as possible across the k clusters, which will lead to maximum savings in computing time for parallel learning of subgraphs in the E-step.

To meet these specific needs for our problem, we propose a modified hierarchical clustering with average linkage that automatically chooses the number of clusters k. Define the distance between two nodes i and j by

$$d(i,j)=1-|r_{ij}|\in[0,1], \tag{4}$$

where $r_{ij}=\mathrm{cor}(\mathbf{x}_i,\mathbf{x}_j)$ is the correlation between $\mathbf{x}_i$ and $\mathbf{x}_j$ for $i,j=1,\ldots,p$. Hartigan (1981) suggests that one should only consider clusters with at least 5% of the data points, which will be referred to as "big clusters" hereafter.

Following this suggestion, we require the minimum cluster size be $0.05p$. As a result, there will be at most 20 clusters. Let $k_{\max}\le 20$ be the maximum number of clusters specified by the user. For $h=0,1,\ldots,p-1$, let $\mathcal{C}_h$ be the set of clusters formed at the hth step of the hierarchical clustering that proceeds in a bottom-up manner (Figure 1). In particular, $\mathcal{C}_0=\{\{1\},\{2\},...,\{p\}\}$ consists of p singleton clusters and $\mathcal{C}_{p-1}=\{\{1,...,p\}\}$ is

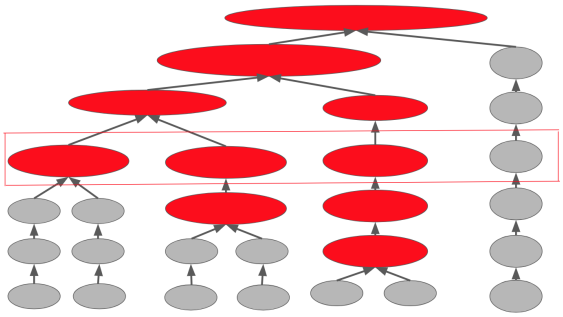


Figure 1: Example for determining k and ℓ. Shown is the upper portion of a dendrogram. Red clusters are big clusters with more than $0.05p$ nodes, and the grey ones are small clusters. The level ℓ is marked by the red box, and in this case $k = 3$.

just one cluster of all p nodes. Let k_i be the number of big clusters in $\mathcal{C}_i$. We choose

$$k = \min\left\{k_{\max}, \max_{0\le i\le p-1} k_i\right\}, \tag{5}$$

which is the maximum number of big clusters subject to the user-specified $k_{\max}$. Let ℓ be the highest level on the dendrogram with k big clusters, i.e.

$$\ell = \operatorname*{argmax}_{0\le i\le p-1}\{i : k_i = k\}. \tag{6}$$

Note that two big clusters will be merged at the next level $(\ell + 1)$ by the hierarchical clustering. Figure 1 shows an example of k and ℓ on a dendrogram.

Relabel the clusters in $\mathcal{C}_\ell$ so that $S_1 \ge S_2 ... \ge S_{p-\ell}$, where $S_i = |C_i|$. Then the first k clusters are big clusters of interest. We assign the remaining small clusters to the k big clusters by recursively merging two closest clusters if at least one of them is a small cluster. An outline of our clustering algorithm is shown in Algorithm 1.

Algorithm 1 Modified hierarchical clustering

1: Hierarchical clustering given the dissimilarity matrix $D = (d(i,j))_{p\times p}$.
2: Generate the dendrogram T_D of the hierarchical clustering.
3: Choose k by (5) and ℓ by (6).
4: Relabel clusters in $\mathcal{C} \leftarrow \mathcal{C}_\ell$ so that $S_1 \ge ... \ge S_{p-\ell}$.
5: **while** $|\mathcal{C}| > k$ **do**
6: $\quad (i^*, j^*) \leftarrow \operatorname{argmin}_{(i,j)}\{d(C_i, C_j) : i < j \text{ and } j > k\}$.
7: $\quad C_{i^*} \leftarrow C_{i^*} \cup C_{j^*}$, $\mathcal{C} \leftarrow \mathcal{C} \setminus \{C_{j^*}\}$.
8: **end while**
9: Return $\mathcal{C} = \{C_1, C_2, ..., C_k\}$.

Remark 4 *In Line 1 of Algorithm 1, by default average linkage is used when merging clusters. Note that in Line 6, minimum distance between clusters is calculated as in single*

linkage clustering. This is because the main purpose of Line 6 is to merge a singleton or small cluster into its closest big cluster, for which finding the closest node is more appropriate than averaging over all nodes in the big cluster.

3.2. Estimation

In the estimation step (E-step) we learn the structure of each subgraph individually. Under our PEF framework, this estimation step acts like a blackbox, and the user may use any structure learning algorithm to estimate the subgraphs without knowing its technical details. The output of this step is in general k PDAGs. Note that both DAGs and CPDAGs are special cases of PDAGs.

In this work, we choose the CCDr algorithm (Aragam and Zhou, 2015) in the `R` package `sparsebn` (Aragam et al., 2019), the PC algorithm in the `R` package `pcalg` (Kalisch et al., 2012) and the MMHC algorithm (Tsamardinos et al., 2006) in the `bnlearn` package (Scutari, 2017) as examples for the E-step. CCDr is a score-based method that outputs a DAG, the PC algorithm is constraint-based and outputs a CPDAG or PDAG, and MMHC is a hybrid approach. As such, we can illustrate the performance of the PEF method with a representative from each of the three structure learning approaches. We choose the CCDr algorithm as a representative of score-based methods for two reasons: 1) It has competitive performance in terms of accuracy for structure learning of DAGs on high-dimensional data, which is our focus. 2) The way it is formulated and coded enables CCDr to learn quite large graphs, allowing for manageable comparisons with PEF in terms of running time.

When the time complexity of a structure learning method grows faster than $O(p^2)$, the running time of learning small subgraphs in the E-step will be much shorter than estimating the full DAG as a whole. Furthermore, we can easily distribute the estimation step. Suppose in the partition step we have divided nodes into k clusters $C_1, \ldots, C_k$, and the running time for learning a PDAG on C_i is t_i. Learning k subgraphs on k cores in parallel will reduce the time for the E-step to $\max\{t_i, i = 1, \ldots, k\}$, which is usually determined by the size of the largest cluster. As supported by our numerical experiments, we can save majority of the computing time with the E-step.

4. Fusion

The fusion step (F-step) is a novel hybrid method developed to add edges between estimated subgraphs from the E-step and to learn the full DAG structure. It proceeds in two stages. First, we generate a candidate edge set A to restrict our search space. By using a sequence of statistical tests, we identify a set A^* of candidate edges between subgraphs. Then the candidate edge set A consists of A^* and all edges learned in each subgraph from the E-step. Second, we minimize a modified BIC score to learn the DAG structure by sequentially updating the edges in the set A. The final output of our PEF method is a DAG.

4.1. Candidate edge set

Recall that Theorem 2 provides a justification for using conditional independence tests to infer edges of a DAG. In light of this result, we develop a method to produce a set A^* of candidate edges between the subgraphs estimated from the E-step. Let $\mathcal{G}_m = (V_m, E_m)$,

$m = 1, \ldots, k$, denote these subgraphs and $z(i) \in \{1, \ldots, k\}$ the cluster label of node i. In general, the subgraphs $\mathcal{G}_m$ are PDAGs. We define the neighbors of a node i in the subgraph $\mathcal{G}_{z(i)}$ as

$$N_i(z(i)) = \{j \in V_{z(i)} : j \to i \in E_{z(i)} \text{ or } (i, j) \in E_{z(i)}\},$$

where $j \to i$ denotes a directed edge and (i, j) an undirected one. By Theorem 2, it is sufficient to find any subset of nodes $\boldsymbol{Z}$ such that X_i and X_j are conditionally independent given $\boldsymbol{Z}$ to conclude that there is no edge between i and j. Unfortunately, for our problem size it is impractical to search all possible subsets. To save calculation, we use the correlation $\tilde{\rho}_{ij} = \text{cor}(\tilde{R}_i, \tilde{R}_j)$, where $\tilde{R}_i$ is the residual after projecting X_i onto its neighbors $N_i(z(i))$ in $\mathcal{G}_{z(i)}$, to filter out unlikely between-subgraph edges. More specifically, we produce an initial candidate set

$$\tilde{A}^* = \{(i, j) : z(i) \neq z(j) \text{ and } \tilde{\rho}_{ij} = 0 \text{ is rejected at significance level } \alpha\}, \tag{7}$$

which will be refined further to define A^*. This test is done using a z-test with Fisher transformation on the correlation coefficient $\tilde{\rho}_{ij}$. Proposition 5 shows that, under certain conditions, $\tilde{A}^*$ will include all between-subgraph edges if the test against $\tilde{\rho}_{ij} = 0$ is perfect. Its proof can be found in Appendix A.2.

Proposition 5 *Suppose the joint Gaussian distribution of $(X_1, \ldots, X_p)$ defined by (2) is faithful to the DAG $\mathcal{G}$. Let $R_{i \cdot A} = X_i - \mathbb{E}(X_i \mid X_A)$ be the residual after regressing X_i onto $X_A := (X_k)_{k \in A}$. If there is an edge $X_i \to X_j$ in $\mathcal{G}$, then $R_{i \cdot A}$ and $R_{j \cdot B}$ are correlated for any disjoint $A, B \subseteq V \setminus \{i, j\}$ as long as $R_{i \cdot A}$ is independent of X_A given X_B.*

Since $R_{i \cdot A}$ is the residual after projecting X_i onto X_A, by definition it is always independent of X_A. So the conclusion of the above proposition holds if $R_{i \cdot A}$ and X_A do not become dependent after conditioning on X_B. Our rule (7) could produce false positive statements: X_i and X_j may become independent conditioning on other subsets. Therefore, we develop a sequential way to screen $\tilde{A}^*$ and define the final candidate edge set A^* between subgraphs, described in Line 9 to Line 14 of Algorithm 2: We go through each node pair $(i, j) \in \tilde{A}^*$ and run conditional independence test given the union of their updated neighbors, $N_i(z(i)) \cup N_j(z(j)) \cup P_{ij}$, where

$$P_{ij} = \{v : (v, i) \in A^* \text{ or } (v, j) \in A^*\} \tag{8}$$

is the set of neighbors of i or j in the current candidate set A^* between subgraphs.

We use an example to illustrate the key steps in Algorithm 2, in which the true DAG shown in Figure 2 has six edges (solid arrows in the figure). Suppose the P-step has divided the nodes into two clusters, $\mathcal{C}_1 = \{X_1, X_2, X_3, X_4\}$ and $\mathcal{C}_2 = \{X_5, X_6, X_7\}$. As a result, the two edges $X_2 \to X_5$ and $X_2 \to X_7$ between the two clusters are cut by this step. Assume the E-step with enough data has estimated the following edges: $X_1 \to X_3$, $X_4 \to X_3$, $X_3 \to X_2$, $X_5 \to X_7$, and $X_7 \to X_6$. Note that $X_5 \to X_7$ is a false positive edge in $\mathcal{C}_2$, caused by the missing common parent X_2 due to the P-step. Take the node pair (X_3, X_7) as an example to illustrate how we include between-subgraph pairs in the candidate set A^*. First regressing X_3 onto $\{X_1, X_4\}$ and X_7 onto $\{X_5\}$, we calculate the residuals $R_{3 \cdot \{1,4\}}$ and $R_{7 \cdot \{5\}}$, respectively. Then we test if $\text{cor}(R_{3 \cdot \{1,4\}}, R_{7 \cdot \{5\}}) = 0$ (Line 4 of Algorithm 2).

Algorithm 2 Find candidate edge set A

1: Input data matrix $\mathbf{x}$ and estimated subgraphs $\mathcal{G}_1, ..., \mathcal{G}_k$.
2: Set $\tilde{A}^* = \emptyset$.
3: **for** all pairs (i, j) such that $z(i) \neq z(j)$ **do**
4: **if** $\tilde{\rho}_{ij} = 0$ is rejected at level α **then**
5: $\tilde{A}^* \leftarrow \tilde{A}^* \cup (i, j)$.
6: **end if**
7: **end for**
8: Set $A^* = \emptyset$.
9: **for** all $(i, j) \in \tilde{A}^*$ **do**
10: Let $\boldsymbol{Z} = N_i(z(i)) \cup N_j(z(j)) \cup P_{ij}$, where P_{ij} is defined in (8).
11: **if** $\mathcal{I}_P(X_i; X_j | \boldsymbol{Z})$ is rejected at level α **then**
12: $A^* \leftarrow A^* \cup (i, j)$.
13: **end if**
14: **end for**
15: Return $A = A^* \cup SK(\mathcal{G})$.

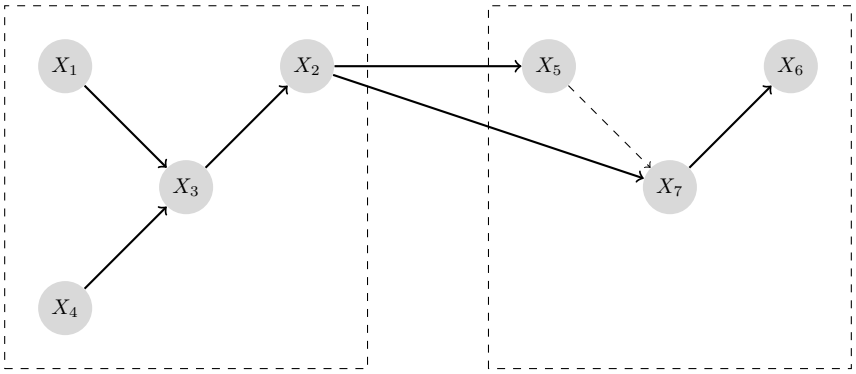


Figure 2: An example DAG for fusion step illustration. Solid edges are the true edges, while the dashed arrow $X_5 \to X_7$ is a false positive generated in the estimation step. The two dashed boxes indicate the clustering result.

Assume the null hypothesis is rejected so that we include $(X_3, X_7) \in \tilde{A}^*$. Next consider the loop through Line 9 to Line 14. Suppose we have added the pair (X_2, X_7) to A^* before the test for $\mathcal{I}_P(X_3; X_7 | \boldsymbol{Z})$, where the conditioning set $\boldsymbol{Z} = \{X_1, X_2, X_4, X_5\}$ as defined in Line 10. Since $X_3 \perp X_7 \mid \boldsymbol{Z}$ implied by d-separation of the true DAG, the test will be accepted (in the large-sample limit) and consequently the pair (X_3, X_7) will not be included in the set A^*.

Remark 6 *In Algorithm 2, node pairs are added to A^* sequentially (Line 12) and thus, the result depends on the order we go through $\tilde{A}^*$. In our implementation, we sort the node pairs in $\tilde{A}^*$ in the ascending order of their p-values in testing against $\tilde{\rho}_{ij} = 0$ (Line 4). In this way, node pairs that are more significant will have a higher priority to be included in*

the set A^*. Similarly, we also sort the node pairs in A^* according to their p-values calculated in Line 11.

Remark 7 *Algorithm 2 will introduce some false positives (FPs) to the candidate set A^*. Here, a false positive refers to a node pair $(i, j) \in A^*$ that is non-adjacent in the true DAG. Let $\mathcal{T}$ be the set of tests performed in Line 4 of Algorithm 2, and $|\mathcal{T}|$ the number of tests. Assuming there are no edges between the subgraphs, the expected number of false positives in the set $\tilde{A}^*$ is $\alpha|\mathcal{T}|$. Due to the additional tests in Line 11, $A^* \subset \tilde{A}^*$ and thus $\alpha|\mathcal{T}|$ serves as an upper bound for the expected number of FPs in A^*. Because the tests in Line 11 and Line 4 are not independent, it is in general hard to calculate a tighter FP upper bound for A^*. When there are edges between clusters, more false positives may be present in $\tilde{A}^*$ due to the cut of these between-subgraph edges. This is the reason why we include the additional tests in Line 11, which are designed to remove false positives by adding potential parents in other subgraphs to the conditioning set $\boldsymbol{Z}$. Note that the purpose of A^* is to restrict the search space for the fusion step, which uses a modified BIC to further decide whether an edge between $(i, j) \in A^*$ exists in the DAG. Most of the false positive pairs in A^* will not be connected by an edge, so that the final accuracy of our method is quite satisfactory. We provide simulation results in supplemental material to calculate the actual number of false positives in the candidate set A^*. The empirical results show that the number of false positives in A^* is less than $\alpha|\mathcal{T}|$ for more than 75% of the datasets. See supplemental material for more detailed results and discussion.*

Breaking a full DAG into subgraphs not only might introduce false negatives, i.e. the cut edges between two subgraphs, but also it could result in false positive edges within a subgraph. Suppose two non-adjacent nodes i and j share a common parent v in the full DAG, but the P-step has put v into a different cluster than i and j. Then in the subgraph containing i and j, there will be an edge (i, j) in the estimated skeleton since they are *not* independent without conditioning on v. By cutting some of the edges in the P-step, we have changed the structure of a subgraph by adding such false positive edges. Figure 2 shows an example of this situation. Although X_5 and X_7 are conditionally independent given X_2 in the full DAG, they will be connected in the subgraph on $\mathcal{C}_2$ (the dashed arrow in the figure) because their common parent X_2 is not in this cluster. To fix this problem, we will revisit all edges learned in the E-step and remove some of them based on the new edges added between subgraphs, using a score-based approach (Section 4.2). Let $\mathcal{G} = (V, E)$ be the PDAG consisting of disconnected subgraphs learned from the E-step and $SK(\mathcal{G})$ be the (undirected) edge set in the skeleton of $\mathcal{G}$, i.e.,

$$SK(\mathcal{G}) = \{(i, j) : (i, j) \in E \text{ or } i \to j \in E\}.$$

Our candidate edge set A is formed by attaching $SK(\mathcal{G})$ to the end of A^* (Line 15 in Algorithm 2). The edges of our final output DAG will be restricted to a subset of A. The complete algorithm for finding the candidate edge set A is summarized in Algorithm 2.

4.2. Learning full DAG structure

The last stage in the fusion step is to determine, for each node pair $(i, j) \in A$, whether there is an edge and its orientation if an edge does exist. This is done by minimizing a

modified BIC score, called the risk inflation criterion (RIC) (Foster and George, 1994), over the candidate edge set in a sequential manner. The RIC score has two components, a log-likelihood part to measure how good a graph $\mathcal{G}$ fits the data and a regularization term to enforce sparsity:

$$\mathrm{RIC}(\mathcal{G}) = -2\ell(\widehat{B}, \widehat{\Omega} \mid \mathcal{G}) + \lambda d(\mathcal{G}), \tag{9}$$

where $\ell(\cdot \mid \mathcal{G})$ is the log-likelihood (3) evaluated at the MLE $(\widehat{B}, \widehat{\Omega})$ given the DAG $\mathcal{G}$, $d(\mathcal{G})$ is the number of edges, and $\lambda = 2\log p$. We use this score when the number of nodes is large with $p > \sqrt{n}$. When $p \leq \sqrt{n}$, we switch back to the regular BIC score, i.e. $\lambda = \log n$.

For each $(i, j) \in A$, we need to compare three models:

$$\begin{aligned} &M_0 : \text{no edge between } i \text{ and } j, \\ &M_1 : i \text{ is a parent of } j, \\ &M_2 : j \text{ is a parent of } i, \end{aligned} \tag{10}$$

while holding other edges in $\mathcal{G}$ fixed. Since the RIC score (9) is decomposable, this comparison reduces to comparing the score difference for the involved child nodes (see Appendix A.3). If there is a true edge between the two nodes i and j, both M_1 and M_2 will have a lower RIC score than M_0 in the large-sample limit, due to the nonzero partial correlation between X_i and X_j given any other set of variables. Thus, we will add an edge between (i, j) if and only if

$$\max\{\mathrm{RIC}(M_1), \mathrm{RIC}(M_2)\} < \mathrm{RIC}(M_0), \tag{11}$$

where $\mathrm{RIC}(M)$ is the RIC score for model M. If criterion (11) is met, we will further decide the edge orientation. To enforce acyclicity, if the edge $i \to j$ (or $j \to i$) induces a directed cycle, we add $j \to i$ (or $i \to j$). If neither direction induces a directed cycle, we choose the model with a smaller RIC following a default tie-breaking rule. See Appendix A.3 for more technical details.

The full fusion step is shown in Algorithm 3, which cycles through A iteratively until the structure of $\mathcal{G}$ does not change. Denote by $N_i(\mathcal{G})$ the neighbors of node i in the current $\mathcal{G}$. At any iteration, if $\mathcal{I}_P(X_i; X_j | N_i(\mathcal{G}) \cup N_j(\mathcal{G}))$ according to the conditional independence test (Line 8), we will remove the pair (i, j) from A permanently. This rule is again justified by Theorem 2 under faithfulness. In order to reduce the number of false positive edges for large p, the significance level for all tests, including the α in Algorithm 2, is set to 0.001 in our implementation.

5. Numerical experiments

In this section, we test our PEF method on Gaussian data generated from real networks. We choose to use three different structure learning algorithms in the E-step: 1) a score-based method, the CCDr algorithm, which estimates a DAG; 2) a constraint-based method, the PC algorithm, which outputs a PDAG; 3) a hybrid method, the MMHC algorithm, which outputs a DAG. We will call the three implementations PEF-CCDr, PEF-PC, and PEF-MMHC hereafter. Accordingly, we compare the results from the PEF methods with those from the CCDr algorithm, the PC algorithm, and the MMHC algorithm applied on the whole data, which will demonstrate the advantages of our divide-and-conquer strategy in learning large networks.

Algorithm 3 Fuse subgraphs

Input data matrix $\mathbf{x}$ and estimated subgraphs $\mathcal{G}_1, ..., \mathcal{G}_k$.
Run Algorithm 2 to generate candidate edge set A.
Initialize $\mathcal{G}$ to be the PDAG consisting of $\mathcal{G}_1, \ldots, \mathcal{G}_k$.
for all $(i, j) \in A$ **do**
    if i, j are adjacent in $\mathcal{G}$ **then**
        remove the edge from $\mathcal{G}$.
    end if
    if $\mathcal{I}_P(X_i; X_j | N_i(\mathcal{G}) \cup N_j(\mathcal{G}))$ **then**
        $A \leftarrow A \setminus \{(i, j)\}$.
    else
        $\text{RIC}_{\max} = \max(\text{RIC}(M_1), \text{RIC}(M_2))$.
        if $\text{RIC}_{\max} < \text{RIC}(M_0)$ **then**
            if adding edge $i \to j$ induces a cycle **then**
                add $j \to i$ to $\mathcal{G}$
            else if adding edge $j \to i$ induces a cycle **then**
                add $i \to j$ to $\mathcal{G}$
            else
                choose the direction that leads to a smaller RIC.
            end if
        end if
    end if
end for
Repeat 4 to 22 until the structure of $\mathcal{G}$ does not change and return $\mathcal{G}$.

Zeng and Poh (2004) also developed a divide-and-conquer method for structure learning of Bayesian networks, but their approach is quite different from ours. A big difference is that they partition the full network into overlapping sub-networks. Their algorithm learns the structure of the overlap parts first, and then use the v-structures learned in the overlaps as constraints for each sub-network. Their algorithm is purely constraint-based, while ours is a hybrid method that can use any existing structure learning algorithm in the E-step. They only tested their method on a small network with ≤ 40 nodes, while our PEF method is designed mainly for big networks with hundreds or thousands of nodes. Since the two methods have very different scopes, we did not include their method in the comparison.

5.1. Data generation

All network structures were downloaded from the repository of the `R` package `bnlearn` (Scutari, 2010, 2017). The networks used in this work are: PATHFINDER, ANDES, DIABETES, PIGS, LINK, and MUNIN, with the number of nodes $p = 135$, 223, 413, 441, 724, 1041 and the number of edges $s_{\text{sub}} = 195$, 338, 602, 592, 1125, 1397. In order to generate large DAGs, we replicate each network k times and randomly add some edges between copies of the network. For easy reference, define $\text{Net}(k, c)$ to be the DAG composed of k replicates of Net with $c \cdot ks_{\text{sub}}$ edges added between subgraphs, where $c \geq 0$ is a constant and Net

is one of the above six networks. Let $s_w = ks_{\text{sub}}$ be the number of within-subgraph edges and s_b be the number of between-subgraph edges. Then $c = s_b/s_w$ is the ratio between the numbers of the two types of edges. For example, ANDES$(5, 0.1)$ refers to a network constructed by 5 copies of the ANDES network with $s_b = 0.1s_w$ edges added between the 5 sub-networks. Denote the number of true edges in a DAG by $s_0 = s_w + s_b$. We have three network generation schemes:

i. Net$(5, c)$ for $c \in \{0, 0.1\}$ and Net $\in$ {PATHFINDER, ANDES, DIABETES, PIGS, LINK}. In total, ten networks were generated by this scheme.

ii. Mixed networks: We combined networks PATHFINDER, ANDES, DIABETES, PIGS, LINK to build a DAG with $k = 5$ different subgraphs. Similar to scheme (i), we randomly added $s_b = cs_w$ edges between subgraphs for $c \in \{0, 0.1\}$. We refer to these two networks as Mix$(5, c)$.

iii. MUNIN$(k, 0)$ for $k = 1, \ldots, 10$: MUNIN is the largest network available on the `bnlearn` repository. We did not add any edges between the subgraphs. So the number of edges for each DAG generated here was $s_0 = s_w = ks_{\text{sub}}$.

Data sets from the above DAGs were generated according to the linear structural equation model in (2). We first drew β_{ij} uniformly from $[-1, -0.5] \cup [0.5, 1]$ if $(i, j) \in E$ and set $\beta_{ij} = 0$ otherwise. The error variances $\sigma_j^2, j = 1, ..., p$ were sampled uniformly from $[0, 1]$. After sampling each data column (data for one node), we rescaled it so that all data column had the same mean and standard deviation. As a result, the β_{ij} after rescaling was no longer in the same interval. The number of observations in the simulated data sets ranged between $n = 1,000$ to $10,000$. For each network generated by schemes (i) and (ii), we simulated 10 data sets. Networks in scheme (iii) were mainly used to test the limit of structure learning algorithms, so only five data sets were generated from each DAG.

5.2. Accuracy metrics

We propose a few metrics to evaluate the accuracy of PDAGs learned by structure learning algorithms. As DAGs can be regarded as a special class of PDAGs, metrics defined here can be used to assess the quality of estimated DAGs as well. Since we are using observational data, structure learning algorithms may not determine all edge orientations due to Markov equivalence (Definition 3). In our assessment, we take v-structures and compelled edges into account in the following definitions of structural accuracy metrics:

- T, the number of edges in the true graph.

- P, the number of predicted edges by a structure learning algorithm.

- E, the number of expected edges. We define an estimated directed edge to be expected if it meets either of the following two criteria: (1) This edge is in the true DAG with the correct orientation; (2) The edge coincides after converting the estimated DAG and the true DAG to CPDAGs. An estimated undirected edge is considered expected if it satisfies condition 2.

- R, the number of reversed edges. This is the number of predicted edges in the true skeleton, excluding expected edges.
- FP = P − E − R, the number of false positive edges.
- SHD = R + M + FP, the structural Hamming distance between the estimated and the true graphs, where M = T − E − R is the number of missing edges.
- JI = E/(T + P − E), the Jaccard index, i.e. the ratio of the number of common edges over the size of the union of the edge sets of two graphs.

In particular, SHD and JI are overall accuracy metrics. Small SHD and/or high JI indicate high accuracy in structure learning. Furthermore, we also use the BIC and test data likelihood of an estimated DAG as metrics to quantify its accuracy in estimating the underlying joint distribution.

5.3. Comparison with the CCDr algorithm

We will first show the improvement in speed of our PEF-CCDr method compared to the CCDr algorithm. Then we will show that the PEF-CCDr method actually improves the accuracy of the CCDr algorithm. For all the experiments, we ran CCDr provided in the `R` package `sparsebn`. The CCDr algorithm outputs a solution path with an increasing number of edges. In order to enforce sparsity, we simply chose the DAG along the solution path with around $1.5p$ edges, and stopped running CCDr when the number of estimated edges on the path became greater than $2p$ by setting `edge.threshold` $= 2p$. Note that we used exactly the same settings for CCDr applied to learn the full graph and in the E-step of the PEF method.

5.3.1. Timing comparison

Figure 3 reports the $\log_{10}$ running times of the two algorithms on data with $n = 1,000$. Figure 3(a) illustrates how the two methods scaled when the size of the subgraphs increased, tested on the networks Net$(5, 0)$ for Net $\in$ {PATHFINDER, ANDES, DIABETES, LINK, MUNIN}. Figure 3(b) illustrates how the two methods scaled when the number of subgraphs increased, using the networks MUNIN$(k, 0)$ for $k = 1, ..., 10$. Table 1 reports the total running time (T) of CCDr and PEF-CCDr, as well as the running time of each step (P, E, F) of PEF-CCDr for all 22 networks (Section 5.1). For the E-step in our PEF-CCDr method, we report the time for parallel estimation of multiple subgraphs.

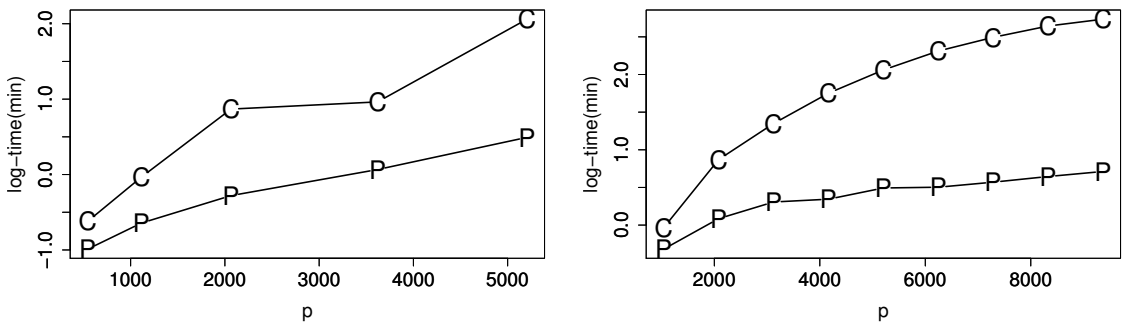
From Figure 3(a) we see that when the number of subgraphs stayed the same and the size of the sub-graphs became larger, the running time of PEF-CCDr increased monotonically. The scalability of the E-step depends on the CCDr algorithm. Therefore, the running time of the E-step of our PEF-CCDr method increased with the size of the subgraphs, in a similar pattern as the CCDr algorithm did. As reported in Table 1, for the largest network MUNIN$(5, 0)$ included in Figure 3(a), PEF-CCDr was 37 times faster than CCDr.

From the lower panel of Table 1 as well as Figure 3(b), we see that as k increased, improvement of our PEF method in speed became more substantial. The number of clusters our PEF-CCDr method identified ($\hat{k}$) is shown in Table 1. The PEF-CCDr method identified the correct number of subgraphs for $k \geq 3$, and therefore the running time of the E-step

Table 1: Timing comparison (in minutes) between CCDr and PEF-CCDr

		CCDr	PEF-CCDr					
Network	p	T	T	P	E	F	$\hat{k}$	r_T
PATHFINDER(5, 0)	545	0.24	0.10	0.01	0.01	0.08	5.0	2.40
PATHFINDER(5, 0.1)	545	0.23	0.10	0.01	0.01	0.08	5.0	2.30
ANDES(5, 0)	1115	0.93	0.24	0.02	0.02	0.20	9.5	3.88
ANDES(5, 0.1)	1115	0.59	0.38	0.02	0.02	0.34	8.4	1.55
MIX(5, 0)	1910	4.65	0.46	0.06	0.08	0.32	8.6	10.11
MIX(5, 0.1)	1910	2.51	0.67	0.06	0.16	0.45	7.2	3.75
DIABETES(5, 0)	2065	7.38	0.53	0.06	0.09	0.38	8.1	13.92
DIABETES(5, 0.1)	2065	4.73	0.60	0.05	0.08	0.47	8.2	7.88
PIGS(5, 0)	2205	10.84	0.64	0.07	0.16	0.41	5.8	16.94
PIGS(5, 0.1)	2205	6.60	0.74	0.07	0.14	0.53	6.2	8.92
LINK(5, 0)	3620	9.19	1.17	0.17	0.16	0.84	8.1	7.85
LINK(5, 0.1)	3620	9.90	1.59	0.16	0.17	1.26	9.2	6.23
MUNIN(1, 0)	1041	0.93	0.48	0.02	0.20	0.27	7.0	1.94
MUNIN(2, 0)	2082	7.42	1.22	0.07	0.79	0.36	4.2	6.08
MUNIN(3, 0)	3123	22.32	2.03	0.12	1.04	0.87	3.0	11.00
MUNIN(4, 0)	4164	56.42	2.21	0.22	1.13	0.86	4.0	25.53
MUNIN(5, 0)	5205	114.85	3.11	0.34	1.21	1.57	5.0	36.93
MUNIN(6, 0)	6246	204.93	3.18	0.46	1.28	1.44	6.0	64.44
MUNIN(7, 0)	7287	311.59	3.71	0.64	1.28	1.79	7.0	83.99
MUNIN(8, 0)	8328	440.02	4.42	0.82	1.26	2.33	8.0	99.55
MUNIN(9, 0)	9369	542.56	5.15	1.04	1.33	2.78	9.0	105.35
MUNIN(10, 0)	10410	NA	5.94	1.32	1.39	3.23	10.0	NA

Note: p is the number of nodes, T is the total running time, P, E, and F are the running times for the P-step, the E-step, and the F-step, respectively, $\hat{k}$ is the average number of estimated clusters in the P-step, and r_T is the ratio of total running time of CCDr over that of PEF-CCDr.



(a) (b)

Figure 3: Log_{10} running time for different size of DAGs. The line with -C- is for CCDr and the line with -P- for PEF-CCDr.

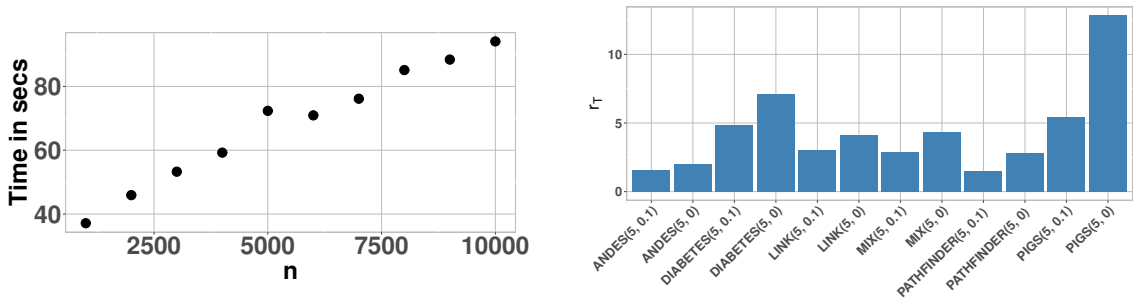
stayed comparable to the running time of CCDr on a single MUNIN network (around 1 minute). When the number of subgraphs $k = 3$, PEF-CCDr was 11 times faster than CCDr, and when $k = 9$, it was 105 times faster. When k was increased to 10, our device for running the tests, MacBook Pro with 3.1 GHz Intel Core i7 processor, ran out of memory for the CCDr algorithm. Our PEF-CCDr method, on the other hand, took only 5.94 minutes to run MUNIN(10, 0). This example shows the huge advantage of our PEF-CCDr method in terms of computational efficiency for learning big Bayesian networks.

5.3.2. Scalability with sample size n

So far, we have seen that the PEF method scales well as p grows, while fixing the sample size to $n = 1,000$. In this section, we examine the scalability of the PEF method with respect to the sample size n. Figure 4(a) plots the running times on data sets simulated from DIABETES(5, 0.1), with n increased from $1,000$ to $10,000$. From the plot, we see that the running time of PEF-CCDr increased linearly in n, and it finished within two minutes on data sets consisting $n = 10,000$ data points. In addition, we applied PEF-CCDr on datasets of size $n = 5,000$ from 12 networks with $p \in (500, 4000)$ to get a general sense of how our method performs with bigger data sets. As reported in Figure 4(b), runtime ratios r_T of CCDr over PEF-CCDr ranged between 1.5 and 13, with more detailed results in supplemental material. We see that PEF-CCDr was still much faster than CCDr and that even for the biggest networks LINK(5) PEF-CCDr finished within three minutes.

5.3.3. Accuracy comparison

Next, we compare the accuracy between the PEF-CCDr method and the CCDr algorithm. Table 2 reports the summary of accuracy for the two methods on ten networks generated by the first two schemes (Section 5.1). In this and subsequent tables, Net(5) refers to either Net$(5, 0)$ or Net$(5, 0.1)$ with the value of c implicitly given by (s_0, s_b). We see from Table 2 that for all cases the SHD of PEF-CCDr was much smaller than CCDr, and the JI was higher than CCDr. BIC scores were quite comparable between the two methods.



(a) (b)

Figure 4: Running time comparison with different sample size n. (a) Running time of PEF-CCDr as n increases; (b) Runtime ratio r_T of CCDr over PEF-CCDr for $n = 5000$.

For networks with $c = 0$ ($s_b = 0$), CCDr tends to have a smaller BIC, while for Net$(5, 0.1)$ PEF-CCDr tends to have a smaller BIC. For PATHFINDER(5) with $p = 545 < n = 1000$, the advantage of our PEF-CCDr method was not as obvious as the rest of the big networks. For all other networks where $p > n$, the number of expected edges of our PEF-CCDr method increased more than 15% compared to CCDr in most of the cases, while the reversed edges and the false positives decreased more than 20%. The overall metrics SHD decreased more than 20% and the JI increased over 35% for all cases.

The number of edges between subgraphs, s_b in Table 2, did not show a substantial impact on the accuracy of PEF-CCDr. This is because when we fuse DAGs from clusters we also correct their structures learned in the E-step. Therefore, even if we cut some edges in the P-step, which may alter the subDAG structures, we can still correct them in the F-step. Therefore, our PEF-CCDr method has some tolerance for errors in the first two steps. Even if the full DAG does not have a clear block structure, in which case many edges will be cut in the P-step, PEF-CCDr can still recover a reasonable amount of these edges. This is demonstrated by the comparable performance of our method on DAGs with a different s_b in the table. See Section 5.7 for more results with regard to this aspect.

On the other hand, performance of PEF-CCDr clearly depends on the structure learning algorithm plugged in the E-step. In the final F-step, we only remove or flip within-subgraph edges, so the missing edges within any subgraph introduced in the E-step will never be added back in the fusion step. In addition, the learned subgraph structures may also affect our choice for the candidate set A (Section 4.1) and thus the final accuracy.

5.3.4. Recovery rate of the fusion step

To examine the role of the fusion step, we compare DAGs learned by our full PEF method with DAGs learned from the first two steps only, i.e. the partition step and the estimation step. We call the latter PE-CCDr, whose results are also reported in Table 2. It is clear from the results that the fusion step always improved the structure of an estimated DAG with increased E, JI and decreased R, FP, SHD. The fusion step also decreased the BIC score for majority of the cases.

Table 2: Accuracy comparison with CCDr

(s_0, s_b)	Method	P	E	R	FP	SHD	JI	BIC
				PATHFINDER(5), $p = 545$				
$(975, 0)$	CCDr	823.0	252.7	149.6	420.7	1143.0	0.164	972.4
	PE-CCDr	818.4	250.6	146.4	421.4	1145.8	0.163	976.9
	PEF-CCDr	660.4	254.7	122.4	283.3	1003.6	0.186	1039.1
$(1073, 98)$	CCDr	838.0	329.5	123.4	385.1	1128.6	0.209	938.4
	PE-CCDr	830.0	275.0	134.3	420.7	1218.7	0.169	989.2
	PEF-CCDr	768.5	361.2	119.1	288.2	1000.0	0.245	1013.8
				ANDES(5), $p = 1115$				
$(1690, 0)$	CCDr	1586.0	931.4	447.0	207.6	966.2	0.397	2322.2
	PE-CCDr	1652.0	873.3	456.4	322.3	1139.0	0.354	2358.7
	PEF-CCDr	1563.0	1187.7	224.3	151.0	653.3	0.576	2273.4
$(1859, 169)$	CCDr	1721.8	1051.6	452.1	218.1	1025.5	0.416	2247.3
	PE-CCDr	1691.8	860.5	452.3	379.0	1377.5	0.320	2377.4
	PEF-CCDr	1766.1	1406.1	186.6	173.4	626.3	0.634	2169.5
				DIABETES(5), $p = 2065$				
$(3010, 0)$	CCDr	3166.3	1327.3	1067.9	771.1	2453.8	0.274	3834.3
	PE-CCDr	3119.5	1281.3	1050.6	787.6	2516.3	0.264	3888.1
	PEF-CCDr	2779.8	1580.2	779.1	420.5	1850.3	0.376	3868.9
$(3311, 301)$	CCDr	3069.6	1499.4	978.9	591.3	2402.9	0.307	3827.6
	PE-CCDr	3058.5	1256.6	994.6	807.3	2861.7	0.246	4048.3
	PEF-CCDr	3202.7	2010.1	702.4	490.2	1791.1	0.447	3721.8
				PIGS(5), $p = 2205$				
$(2960, 0)$	CCDr	3285.6	1677.4	832.0	776.2	2058.8	0.367	4310.6
	PE-CCDr	3290.1	1632.7	834.7	822.7	2150.0	0.354	4355.0
	PEF-CCDr	2809.9	1933.5	541.6	334.8	1361.3	0.504	4371.0
$(3256, 296)$	CCDr	3262.5	1874.0	800.8	587.7	1969.7	0.404	4243.2
	PE-CCDr	3312.9	1574.7	825.4	912.8	2594.1	0.315	4472.9
	PEF-CCDr	3182.1	2308.3	489.9	383.9	1331.6	0.559	4192.0
				LINK(5), $p = 3620$				
$(5625, 0)$	CCDr	5329.4	2640.6	1421.7	1267.1	4251.5	0.318	7113.3
	PE-CCDr	5432.8	2514.9	1432.3	1485.6	4595.7	0.294	7215.8
	PEF-CCDr	5021.9	3211.4	972.6	837.9	3251.5	0.432	7162.0
$(6188, 563)$	CCDr	5799.6	3096.9	1436.5	1266.2	4357.3	0.348	6928.1
	PE-CCDr	5471.0	2422.9	1459.3	1588.8	5353.9	0.262	7447.1
	PEF-CCDr	5849.9	4018.3	878.1	953.5	3123.2	0.501	6849.7
				Mix(5), $p = 1910$				
$(2852, 0)$	CCDr	2893.1	1423.4	766.4	703.3	2131.9	0.329	3695.5
	PE-CCDr	2848.1	1330.9	765.6	751.6	2272.7	0.305	3775.5
	PEF-CCDr	2620.2	1685.9	526.2	408.1	1574.2	0.446	3724.7
$(3138, 286)$	CCDr	2923.5	1564.6	766.0	592.9	2166.3	0.348	3657.5
	PE-CCDr	2880.4	1313.1	779.1	788.2	2613.1	0.279	3852.4
	PEF-CCDr	2965.3	2005.0	497.5	462.8	1595.8	0.489	3613.9

The results in Table 2 show that as s_b increased, the fusion step recovered an increasing number of expected edges. The number of expected edges recovered by the fusion step can reach 60% of that recovered in the first two steps, such as for ANDES$(5, 0.1)$, DIABETES$(5, 0.1)$ and LINK$(5, 0.1)$. In addition, we see that our fusion step not only recovered expected edges, but also was able to remove reversed and false positive edges. Across different cases, the F-step reduced 30% to 60% FPs and 10% to 60% Rs, which substantially improved the structure learning accuracy.

All these observed improvements in accuracy demonstrate the critical role of the fusion step. Not only does it add edges cut by the P-step back to the full DAG, but also gets rid of false positive edges produced by the E-step. This suggests that the fusion step can largely correct the mistakes made by the first two steps, and thus our PEF method may handle networks with a moderate number of between-subgraph edges, relaxing to some degree the assumption of a block structure on the true DAG.

5.4. Comparison with the PC algorithm

In this section, we test our PEF framework with the PC algorithm used for the E-step, which we call PEF-PC. The PC algorithm is a well-known constraint-based method that outputs a PDAG in general. This will complement our comparison with CCDr in the previous subsection, which estimates a DAG via a score-based approach.

In our experiments, we used the PC algorithm in the `pcalg` package (Kalisch et al., 2012) in the E-step. An important tuning parameter of PC is the significance level α for conditional independence tests, which controls the sparsity of an estimated graph: The smaller the α, the sparser the estimated graph and the faster the algorithm. With the default setting $\alpha = 0.05$, PC took too long, more than 24 hours, to learn some DAGs like the PATHFINDER(5) networks. Furthermore, for high-dimensional data, a big α usually results in too many false positive edges in the graph learned by the PC algorithm. In order to make an informative comparison, we set $\alpha = 10^{-4}$ so that the PC algorithm can produce quite accurate PDAGs within a reasonable amount of time. Another tuning parameter is the maximal size (`m.max`) of the conditioning sets that are considered in a conditional independence test. The default value of this parameter is infinity, but with this default value, it took up to 6 hours to run PC on a single data set. Thus, in our experiment, we limited this value to 3. We also tried increasing `m.max` to 5, and got similar results with slightly lower accuracy but much longer running time. The same data for the comparisons in Tables 1 and 2 were used in this experiment. The parameter choices for PC in our E-step were the same as those for running PC on full data.

Table 3 compares the running time between PC and PEF-PC with paralleling the E-step. As described above, we did fine tuning on the parameters of PC to improve its speed, and consequently, the algorithm ran very fast on these data sets. Even though, PEF-PC was usually 2 to 8 times faster. Similar to Table 2, timing data reported for the PEF-PC algorithm in this table were calculated by assuming there exist enough cores to estimate all sub-networks simultaneously in the E-step. The running time improvement here was not as substantial as that for the CCDr algorithm, probably because of the different ways these two algorithms scale with the graph size.

Table 3: Timing comparison (in minutes) with PC and MMHC

Network	p	PC	PEF-PC	r_T	MMHC	PEF-MMHC	r_T
PATHFINDER(5, 0)	545	3.50	1.89	1.85	0.28	0.04	7.00
PATHFINDER(5, 0.1)	545	3.54	1.64	2.16	0.33	0.09	3.67
ANDES(5, 0)	1115	0.52	0.32	1.63	1.94	0.20	9.70
ANDES(5, 0.1)	1115	0.59	0.39	1.51	1.99	0.24	8.29
DIABETES(5, 0)	2065	2.67	0.57	4.68	10.74	0.44	24.41
DIABETES(5, 0.1)	2065	2.82	0.67	4.21	11.34	0.60	18.90
PIGS(5, 0)	2205	4.33	1.01	4.29	12.15	0.55	22.09
PIGS(5, 0.1)	2205	4.87	0.96	5.07	12.71	0.60	21.18
LINK(5, 0)	3620	8.37	1.12	7.47	54.40	1.49	36.51
LINK(5, 0.1)	3620	9.00	1.36	6.62	57.06	1.75	32.61
MIX(5, 0)	1910	3.05	1.82	1.68	8.01	0.44	18.20
MIX(5, 0.1)	1910	3.70	1.75	2.11	8.43	0.82	10.28

(See Table 1 for the definitions of T, P, E, F, and r_T.)

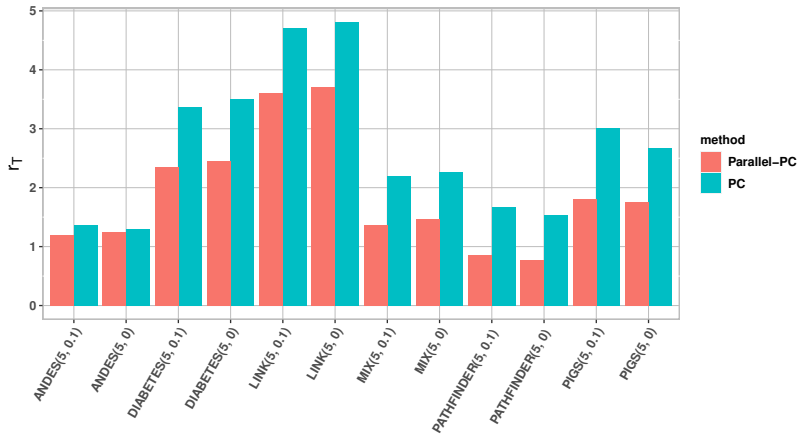


Figure 5: Runtime ratios of Parallel-PC and PC over PEF-PC

There is a parallelized version of the PC algorithm in the R package, `ParallelPC`, developed recently by Le et al. (2015). With the same tuning parameters, the parallelized PC algorithm will produce exactly the same output as the regular PC algorithm. To compare the speed between parallel PC and PEF-PC, we ran both methods on a machine with two physical computing cores. The runtime ratios r_T of parallel PC over PEF-PC are shown in Figure 5 for 12 networks. We see that our PEF-PC was faster than parallel PC for all but the smallest networks, PATHFINDER(5) with $p = 545$, and the speed improvement can be two to four folds for larger networks ($p > 2,000$).

Next, we compare the estimation accuracy between PC and PEF-PC. To confirm the effect of the fusion step in our method, we also compare with PE-PC, which only includes the first two steps of our PEF framework. Table 4 reports the overall accuracy metrics,

while omitting the detailed metrics of E, R and FP to save space. Similar to the results in the comparison with CCDr, we observe significant improvement in accuracy of PEF-PC over PC. For all the networks tested, the Jaccard index of PEF-PC was much higher and the SHD and BIC of PEF-PC was much lower than PC. Consistent with Table 2, the fusion step of PEF-PC substantially improved the results from the P-step and the E-step, by recovering more expected edges and correcting many reversed edges. Take the ANDES(5) network with $s_b = 0$ as an example. Our PEF method found 30% more expected edges, while reducing reversed edges by more than 80%, compared to the other two competitors.

Remark 8 *Comparing the results in this section with those in Section 5.3, it appears that the PEF-PC method outperformed the PEF-CCDr method in terms of accuracy for most of the networks except PATHFINDER(5). This is because the PC algorithm had higher accuracy than the CCDr algorithm on these data. Such differences match our expectation that performance of the PEF framework will depend on the algorithm used in the E-step. On the other hand, our PEF framework showed substantial advantages over both algorithms, demonstrating the robustness of our divide-and-conquer strategy regardless of the performance of the DAG learning algorithm used in the E-step.*

5.5. Comparison with the MMHC algorithm

In this section, we test our PEF framework with the MMHC algorithm used for the E-step, which we call PEF-MMHC. The MMHC algorithm is a hybrid method which first uses the constraint-based MMPC algorithm (Tsamardinos et al., 2003) to restrict the search space and then applies a score-based hill climbing algorithm to learn a DAG structure. This comparison will demonstrate how the PEF method works with a hybrid method in the E-step.

We used the MMHC algorithm in the `bnlearn` package (Scutari, 2010; Scutari and Denis, 2014; Nagarajan and Scutari, 2013; Scutari, 2017). Similar to the PC algorithm, the MMPC algorithm also has two important tuning parameters: the significance level (`alpha`) and the maximum size of a conditioning set (`max.sx`). With its default values of `alpha` (0.05) and `max.sx` (unlimited), the algorithm estimated too many false positives and took a very long time to finish. Therefore, we set `alpha = 0.001` and `max.sx = 3`. Data sets used in this section were the same as those in Tables 1 and 2.

The results on timing and accuracy are reported in Table 3 and Table 4. Table 3 shows the detailed timing data for PEF-MMHC and MMHC. Clearly, PEF-MMHC made a significant improvement in speed: For the biggest network, LINK(5), PEF-MMHC was over 30 times faster than MMHC. Table 4 compares the accuracy among the two algorithms and PE-MMHC, where the F-step is omitted. Overall, PEF-MMHC algorithm achieved a comparable accuracy, in terms of SHD and JI, with MMHC, while it always had a lower BIC score (better in terms of model selection). This comparison shows that our PEF framework can significantly improve the speed of the MMHC algorithm with comparable accuracy. Similar to previous results, the F-step played an important role in our framework, seen from the fact that PEF-MMHC outperformed PE-MMHC with respect to all three overall accuracy metrics.

Table 4: Accuracy comparison with PC and MMHC

		M = PC				M = MMHC			
(s_0, s_b)	Method	P	SHD	JI	BIC	P	SHD	JI	BIC
PATHFINDER(5), $p = 545$									
$(975, 0)$	M	438.0	913.7	0.123	1265.8	281.6	872.9	0.144	1289.8
	PE-M	436.9	913.3	0.123	1255.5	280.0	872.4	0.143	1289.9
	PEF-M	434.9	813.3	0.222	1141.0	290.6	882.8	0.142	1289.8
$(1073, 98)$	M	521.9	966.3	0.148	1235.5	342.8	926.2	0.165	1271.7
	PE-M	492.0	1020.3	0.127	1245.5	341.3	1008.3	0.131	1275.1
	PEF-M	660.8	948.4	0.247	1072.3	579.5	1099.1	0.156	1206.1
ANDES(5), $p = 1115$									
$(1690, 0)$	M	1483.0	567.1	0.564	2540.8	1269.3	679.8	0.534	2415.3
	PE-M	1398.4	680.1	0.516	2569.1	1223.2	774.9	0.487	2453.9
	PEF-M	1520.7	315.3	0.796	2255.0	1382.0	688.7	0.546	2383.4
$(1859, 169)$	M	1635.4	649.9	0.544	2510.8	1337.7	762.6	0.535	2368.1
	PE-M	1387.8	925.9	0.441	2593.3	1222.6	973.0	0.440	2469.5
	PEF-M	1714.3	347.1	0.801	2155.2	1618.1	669.2	0.610	2262.8
DIABETES(5), $p = 2065$									
$(3010, 0)$	M	2563.0	1151.6	0.507	4422.5	2328.3	1542.7	0.393	4132.0
	PE-M	2506.2	1247.7	0.487	4438.7	2278.9	1605.4	0.381	4165.8
	PEF-M	2601.1	874.0	0.641	3873.1	2401.4	1570.3	0.397	4110.0
$(3311, 301)$	M	2850.5	1267.9	0.507	4389.4	2516.2	1572.7	0.439	4051.4
	PE-M	2466.4	1744.3	0.409	4527.1	2247.0	2050.8	0.333	4284.8
	PEF-M	3009.5	929.9	0.669	3721.2	2927.0	1673.2	0.448	3925.0
PIGS(5), $p = 2205$									
$(2960, 0)$	M	2556.6	1114.9	0.519	4968.5	2203.9	1552.5	0.404	4774.1
	PE-M	2497.2	1211.5	0.492	4992.1	2145.8	1624.0	0.385	4815.6
	PEF-M	2586.6	771.4	0.686	4429.3	2263.9	1600.1	0.399	4762.2
$(3256, 296)$	M	2859.6	1175.1	0.530	4861.5	2380.9	1591.7	0.442	4684.6
	PE-M	2525.8	1693.5	0.415	5054.7	2197.3	1995.8	0.354	4866.8
	PEF-M	2989.4	790.5	0.720	4235.9	2771.4	1615.8	0.467	4540.1
LINK(5), $p = 3620$									
$(5625, 0)$	M	4752.8	2301.4	0.505	7951.1	3897.5	2972.8	0.419	7726.2
	PE-M	4510.0	2671.5	0.458	8066.9	3715.4	3243.2	0.384	7827.2
	PEF-M	4734.4	1394.9	0.730	7184.0	4164.2	3081.4	0.416	7652.0
$(6188, 563)$	M	5244.1	2490.0	0.502	7848.0	4197.2	3105.7	0.448	7593.6
	PE-M	4361.3	3627.6	0.372	8162.3	3716.4	3972.7	0.338	7940.0
	PEF-M	5434.0	1545.6	0.735	6900.2	5153.6	3072.1	0.486	7261.1
Mix(5), $p = 1910$									
$(2852, 0)$	M	2376.6	1203.1	0.486	4229.8	2004.2	1509.5	0.409	4064.3
	PE-M	2251.5	1386.3	0.442	4279.6	1927.5	1644.3	0.377	4117.6
	PEF-M	2409.1	826.4	0.673	3759.5	2164.5	1544.6	0.416	4017.6
$(3138, 286)$	M	2621.5	1329.3	0.483	4204.4	2136.6	1655.1	0.415	4027.6
	PE-M	2280.3	1772.5	0.388	4324.4	1968.0	1992.5	0.339	4160.8
	PEF-M	2766.3	941.1	0.676	3641.9	2626.0	1676.6	0.452	3861.6

5.6. Test data likelihood

Test data likelihood is another objective accuracy metric for an estimated DAG, which quantifies its accuracy in estimating the joint distribution encoded by the DAG. To this end, we generated 50 test datasets for each network, and calculated test data likelihood under an estimated DAG. Given the estimated DAG, we applied least-squares regression of a child node X_j onto its parents to estimate the edge coefficients $\widehat{\beta}_{ij}$ and noise variances $\widehat{\sigma}_j^2$ for $j = 1, \ldots, p$ from training data. Then test data log-likelihood was calculated with these estimated parameters. For the PC algorithm, of which an estimated graph is a CPDAG, we first converted it into a DAG in the equivalence class before calculating test data log-likelihood.

Figure 6 shows the boxplots for test data log-likelihood values for all networks with weak connection ($c = 0.1$). Note for each test dataset, we computed its likelihood under the estimated DAG from each of the 10 training datasets generated from the same network. So a boxplot in the figure reports the distribution of $50 \times 10 = 500$ log-likelihood values. When comparing CCDr with PEF-CCDr, PC with PEF-PC, MMHC with PEF-MMHC, we see our PEF method had a significantly higher test data log-likelihood for most of the networks, indicating that its estimated DAGs were more accurate in predictive modeling for a joint distribution. Test data likelihood comparison for networks with $c = 0$, provided in the supplemental material, shows a quite consistent pattern as that observed in Figure 6.

5.7. Networks with no block structure

The above numerical results have demonstrated that our PEF framework may achieve significant speed improvement over standard structure learning methods, with higher or comparable estimation accuracy, for networks with a clear block structure. Of course, there are many real-world networks without any block structures (Olesen and Madsen, 2002). Although not designed specifically for such networks, it is worth testing our method on these networks to provide a complete spectrum of its performance.

To do this, we chose four networks: two real networks, LINK and MUNIN, and two simulated small-world networks with $p = 2000$ and $p = 5000$. Note that LINK and MUNIN were not duplicated in this experiment so there were no block structures. Small-world networks were generated by the R package `igraph` (Csárdi and Nepusz, 2006) using the Watts-Strogatz model (Watts and Strogatz, 1998). For each network, 10 datasets of size $n = 1000$ were generated. We applied all six methods (CCDr, PEF-CCDr, PC, PEF-PC, MMHC, PEF-MMHC) on these datasets with the same parameter settings as in previous comparisons. Reported in Tables 5 and 6 are, respectively, the timing and the accuracy results.

No matter whether the true DAG has a block structure or not, our partition step will cluster the nodes to several small clusters. The number of clusters $\hat{k}$ ranged from 6 to 11 for the networks in this comparison. In general, our PEF framework scaled much better than the standard structure learning algorithms used in the E-step, showing significant speed improvement without losing much accuracy. It is seen from Table 5 that PEF was always faster ($r_T > 1$) than the competitors, except for the smallest network LINK when compared with CCDr. The improvement in speed was especially significant for large small-world networks with $r_T > 20$. In terms of estimation accuracy (Table 6), PEF-CCDr always

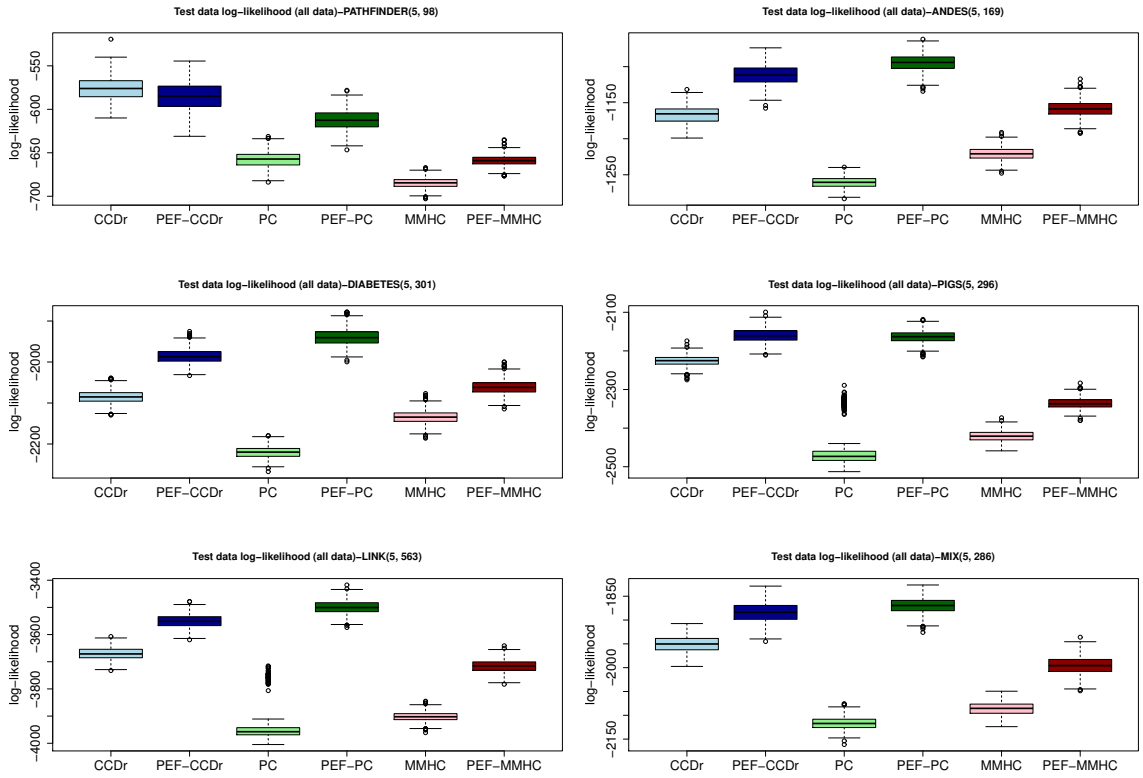


Figure 6: Boxplots of test data log-likelihood for networks with weak between-subgraph connections ($c = 0.1$).

outperformed CCDr substantially, achieving lower SHD and higher JI. Consistent with what we have seen for networks with a block structure in Section 5.5, PEF-MMHC showed a quite comparable performance with the MMHC algorithm. The PEF-PC algorithm was more accurate than the PC algorithm for MUNIN, but less accurate for small-world networks. Cutting a connected DAG into subgraphs will introduce substantial changes in the CI structures, as we discussed in Section 4.1. This could have more substantial impact on the PEF method when the E-step is done by a constraint-based method that performs a large number of CI tests. This is the reason for the observed degraded performance of the PEF-PC algorithm on small-world networks.

This comparison suggests that our proposed PEF framework works quite well for general types of networks. This greatly enlarges the scope of the applicability of our method, as it can efficiently learn a reliable estimated graph without the assumption that the true DAG has a block structure. We leave it as future work how to further improve its accuracy, especially for constraint-based learning on DAGs with no block structures.

Table 5: Timing comparison on networks with no block structure

		M = CCDr			M = PC			M = MMHC		
Network (p)	$\hat{k}$	M	PEF-M	r_T	M	PEF-M	r_T	M	PEF-M	r_T
LINK (724)	8.3	0.12	0.21	0.57	0.35	0.20	1.75	0.58	0.13	4.46
MUNIN (1041)	6.9	0.82	0.35	2.34	0.78	0.55	1.42	1.58	0.45	3.51
Small-world (2000)	10.3	4.00	0.50	8.00	2.17	0.54	4.02	10.92	0.50	21.84
Small-world (5000)	11.0	57.06	2.33	24.49	22.80	1.91	11.94	194.10	3.07	63.22

Table 6: Accuracy comparison on networks with no block structure

(p, s_0)	method	P	E	R	FP	SHD	JI	BIC
LINK	CCDr	1071.0	543.9	276.4	250.7	831.8	0.330	1428.8
$(724, 1125)$	PEF-CCDr	1109.6	711.8	194.9	202.9	616.1	0.468	1410.3
	PC	1102.8	841.3	230.0	31.5	315.2	0.607	1556.3
	PEF-PC	995.0	825.7	103.7	65.6	364.9	0.638	1582.6
	MMHC	778.2	561.6	189.8	26.8	590.2	0.419	1553.6
	PEF-MMHC	962.7	625.6	199.8	137.3	636.7	0.429	1500.1
MUNIN	CCDr	1534.4	696.5	380.9	457.0	1157.5	0.312	2003.2
$(1041, 1397)$	PEF-CCDr	1380.0	839.2	243.8	297.0	854.8	0.433	2005.9
	PC	1230.8	918.0	212.5	100.3	579.3	0.537	2308.8
	PEF-PC	1201.2	963.8	124.2	113.2	546.4	0.590	2313.3
	MMHC	1056.0	743.0	250.9	62.1	716.1	0.435	2161.4
	PEF-MMHC	1121.4	754.6	251.1	115.7	758.1	0.428	2148.0
Small-world	CCDr	3006.0	1205.2	1108.5	692.3	3487.1	0.208	3388.7
$(2000, 4000)$	PEF-CCDr	3323.9	1427.7	1156.9	739.3	3311.6	0.242	3307.9
	PC	3002.7	1968.7	815.0	219.0	2250.3	0.392	3871.8
	PEF-PC	3025.0	1494.9	1205.6	324.5	2829.6	0.270	3839.4
	MMHC	2764.8	1461.1	1109.9	193.8	2732.7	0.276	3487.1
	PEF-MMHC	3195.2	1580.1	1175.0	440.1	2860.0	0.281	3380.8
Small-world	CCDr	7529.0	3061.2	2784.6	1683.2	8622.0	0.212	8534.6
$(5000, 10000)$	PEF-CCDr	8224.8	3618.5	2835.8	1770.5	8152.0	0.248	8355.6
	PC	7526.0	5039.5	1921.2	565.3	5525.8	0.404	9729.1
	PEF-PC	7558.3	3773.5	2969.8	815.0	7041.5	0.274	9655.1
	MMHC	6920.9	3660.4	2754.7	505.8	6845.4	0.276	8793.3
	PEF-MMHC	7932.0	4014.7	2862.6	1054.7	7040.0	0.288	8530.9

6. Discussion

We have developed a divide-and-conquer framework for structure learning of big Bayesian networks from continuous data. The key novel step in our method is the fusion step, which merges the subgraphs learned from subsets of nodes partitioned by a modified clustering

algorithm. Our numerical results suggest that this fusion step can correct and fix the DAG structure damaged by the partition step, so that the overall accuracy of the PEF method is usually comparable to or even higher than the structure learning algorithm used in the estimation step. We also observed quite significant boost in speed, ranging from a few folds to orders-of-magnitude.

There are certain limitations of our current design and implementation of the PEF method. First of all, in the partition step, we need to calculate and store the dissimilarity matrix for all pairs of nodes. When the number of nodes p is really large, this becomes memory-intensive. A promising potential solution to this issue is to borrow ideas from the subsample clustering method for big data (Marchetti and Zhou, 2016). We may subsample a small fraction of the nodes for clustering, and then assign the remaining large number of nodes based on the clustering of the subsample, which can be implemented in a sequential way. For the fusion step, our current implementation takes as input the correlation matrix of the data columns. Again, when the number of nodes is too big, we may implement the algorithm to calculate correlations whenever needed, instead of pre-computing all correlations. Our current fusion step was implemented with the `Rcpp` package `Armadillo`. If we code it in pure C++, the speed of the fusion step may be further improved.

At a conceptual level, it seems straightforward to generalize the PEF method to discrete Bayesian networks. For discrete data, one can still use our clustering method, with a suitable similarity measure, for the partition step, and plug in an appropriate structure learning algorithm in the estimation step. As for the fusion step, the conditional independence test is no longer for zero partial correlations, instead we may use the G^2 test for discrete data as in the PC algorithm. Finally we may substitute linear regression with the multinomial logistic regression as used in Gu et al. (2019) for BIC-based edge selection (Section 4.2). This is left as future work.

Acknowledgments

This work was supported by NSF grants IIS-1546098 and DMS-1952929.

Appendix A.

A.1. Partial correlation

The partial correlation between X and Y given $\boldsymbol{Z}$, $\rho_{XY\cdot\boldsymbol{Z}}$, can be calculated using their covariance matrix. Let m be the size of $\boldsymbol{Z}$, Σ be the covariance matrix of $(X, Y, \boldsymbol{Z})$, and $\Omega = (\omega_{ij})_{(m+2)\times(m+2)} = \Sigma^{-1}$ be the precision matrix. Then the partial correlation

$$\rho_{XY\cdot\boldsymbol{Z}} = -\frac{\omega_{12}}{\sqrt{\omega_{11}\omega_{22}}},$$

and for Gaussian random variables,

$$\mathcal{I}_P(X; Y|\boldsymbol{Z}) \qquad \Longleftrightarrow \qquad \rho_{XY\cdot\boldsymbol{Z}} = 0.$$

In order to test the hypothesis $H_0 : \rho_{XY\cdot\boldsymbol{Z}} = 0$, we apply the Fisher z-transformation,

$$z(X, Y|\boldsymbol{Z}) = \frac{1}{2}\log\left(\frac{1+\hat{\rho}_{XY\cdot\boldsymbol{Z}}}{1-\hat{\rho}_{XY\cdot\boldsymbol{Z}}}\right),$$

where $\hat{\rho}_{XY\cdot \boldsymbol{Z}}$ is the estimated partial correlation calculated from sample covariance matrix of $(X, Y, \boldsymbol{Z})$. Given a significance level α, we reject the null hypothesis H_0 if

$$\sqrt{n-m-3} \cdot |z(X, Y|\boldsymbol{Z})| > \Phi^{-1}(1-\alpha/2),$$

where n is the number of observations and Φ is the cdf of $\mathcal{N}(0,1)$.

A.2. Proof of Proposition 5

By properties of a joint Gaussian distribution, we can write

$$X_i = \sum_{k \in A} \widetilde{\beta}_{ki} X_k + R_{i\cdot A}, \tag{A.1}$$

where $R_{i\cdot A} \perp X_A$ (independence). Similarly, regressing X_j onto $X_{A \cup B \cup \{i\}}$, we arrive at

$$X_j = \sum_{k \in A \cup B} \widetilde{\beta}_{kj} X_k + \widetilde{\beta}_{ij} X_i + \widetilde{\varepsilon}_j, \tag{A.2}$$

with $\widetilde{\varepsilon}_j \perp X_{A \cup B \cup \{i\}}$ and thus $\widetilde{\varepsilon}_j \perp R_{i\cdot A}$. Plugging (A.1) into (A.2) to eliminate X_i, we have

$$X_j = \sum_{k \in A \cup B} \widetilde{\gamma}_{kj} X_k + \widetilde{\beta}_{ij} R_{i\cdot A} + \widetilde{\varepsilon}_j, \tag{A.3}$$

for some $\widetilde{\gamma}_{kj}$'s after rearranging terms in the summation. Denote by $R_{i\cdot A\cdot B}$ the residual of regressing $R_{i\cdot A}$ onto X_B. Since $R_{i\cdot A} \perp X_A \mid X_B$ by assumption, the coefficient

$$\widetilde{\beta}_{ij} = \frac{\mathbb{E}(R_{j\cdot B} R_{i\cdot A\cdot B})}{\mathbb{E}(R_{i\cdot A\cdot B})^2} = \frac{\mathbb{E}(R_{j\cdot B} R_{i\cdot A})}{\mathbb{E}(R_{i\cdot A\cdot B})^2} = \frac{\mathrm{cov}(R_{j\cdot B}, R_{i\cdot A})}{\mathrm{var}(R_{i\cdot A\cdot B})},$$

where the second equality is due to $R_{j\cdot B} \perp \mathbb{E}(R_{i\cdot A} \mid X_B)$. By Theorem 2, $\widetilde{\beta}_{ij} \neq 0$, because otherwise $\mathcal{I}_P(X_j; X_i | X_{A \cup B})$, and thus $\mathrm{cov}(R_{j\cdot B}, R_{i\cdot A}) \neq 0$. The proof is complete.

A.3. RIC for model selection

Recall we want to compare three models, M_0, M_1, M_2, defined in (10). Suppose the current DAG is $\mathcal{G}$, which has no edge between i and j. Now consider the following two linear models

$$X_i = \beta_{ji} X_j + \sum_{k \in \Pi_i^{\mathcal{G}}} \beta_{ki} X_k + \varepsilon_i, \tag{A.4}$$

$$X_j = \beta_{ij} X_i + \sum_{k \in \Pi_j^{\mathcal{G}}} \beta_{kj} X_k + \varepsilon_j. \tag{A.5}$$

Then, M_0 is equivalent to $\beta_{ij} = \beta_{ji} = 0$, M_1 equivalent to $\beta_{ij} \neq 0$ and $\beta_{ji} = 0$, and M_2 equivalent to $\beta_{ij} = 0$ and $\beta_{ji} \neq 0$. Note that when undirected edges exist, we consider all neighbors as the parents.

In order to choose from the three models, we calculate their RIC scores. In our implementation, we find least-squares estimates (LSEs) of the regression coefficients for (A.4)

and (A.5). Let ℓ_{ji} be the log-likelihood evaluated at the LSE under the linear model (A.4), and ℓ_{0i} the log-likelihood under (A.4) when $\beta_{ji} = 0$. Similarly, ℓ_{ij} denotes the log-likelihood at the LSE for the linear model (A.5), and ℓ_{0j} the log-likelihood when $\beta_{ij} = 0$. Since the structure of $\mathcal{G}$ is identical except for the node pair (i, j), these four likelihood scores are sufficient for comparing M_0, M_1 and M_2. Let $\ell(M_i)$ be the log-likelihood of M_i for $i = 0, 1, 2$. Then we have $\ell(M_0) = \ell_{0i} + \ell_{0j}$, $\ell(M_1) = \ell_{0i} + \ell_{ij}$, and $\ell(M_2) = \ell_{ji} + \ell_{0j}$. Thus, the RIC selection criterion (11) is equivalent to $2\min\{\ell_{ij} - \ell_{0j}, \ell_{ji} - \ell_{0i}\} > \lambda$, where λ is the penalty parameter in (9). The motivation for this criterion is to add an edge between i and j only when $i \not\perp j | \Pi_i^{\mathcal{G}}$ and $i \not\perp j | \Pi_j^{\mathcal{G}}$ (Theorem 2).

References

Emmanuel Abbe, Afonso S Bandeira, and Georgina Hall. Exact recovery in the stochastic block model. *IEEE Transactions on Information Theory*, 62(1):471–487, 2016.

Bryon Aragam and Qing Zhou. Concave penalized estimation of sparse gaussian bayesian networks. *Journal of Machine Learning Research*, 16:2273–2328, 2015.

Bryon Aragam, Jiaying Gu, and Qing Zhou. Learning large-scale Bayesian networks with the sparsebn package. *Journal of Statistical Software*, 91(11):1–38, 2019.

David Maxwell Chickering. Learning equivalence classes of bayesian-network structures. *Journal of Machine Learning Research*, 2(Feb):445–498, 2002a.

David Maxwell Chickering. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3(Nov):507–554, 2002b.

David Maxwell Chickering and Christopher Meek. Finding optimal bayesian networks. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pages 94–102. Morgan Kaufmann Publishers Inc., 2002.

David Maxwell Chickering, David Heckerman, and Christopher Meek. Large-sample learning of bayesian networks is np-hard. *Journal of Machine Learning Research*, 5(Oct): 1287–1330, 2004.

Peter Chin, Anup Rao, and Van Vu. Stochastic block model and community detection in sparse graphs: A spectral algorithm with optimal rate of recovery. In *Conference on Learning Theory*, pages 391–423, 2015.

Diego Colombo and Marloes H Maathuis. Order-independent constraint-based causal structure learning. *Journal of Machine Learning Research*, 15(1):3741–3782, 2014.

Gábor Csárdi and Tamás Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL `http://igraph.org`.

Aurelien Decelle, Florent Krzakala, Cristopher Moore, and Lenka Zdeborová. Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. *Physical Review E*, 84(6):066106, 2011.

Dean P Foster and Edward I George. The risk inflation criterion for multiple regression. *The Annals of Statistics*, 22(4):1947–1975, 1994.

Fei Fu and Qing Zhou. Learning sparse causal Gaussian networks with experimental intervention: Regularization and coordinate descent. *Journal of the American Statistical Association*, 108:288–300, 2013.

José A Gámez, Juan L Mateo, and José M Puerta. Learning bayesian networks by hill climbing: efficient methods based on progressive restriction of the neighborhood. *Data Mining and Knowledge Discovery*, 22(1-2):106–148, 2011.

Maxime Gasse, Alex Aussem, and Haytham Elghazel. An experimental comparison of hybrid algorithms for bayesian network structure learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 58–73. Springer, 2012.

Dan Geiger and David Heckerman. Learning gaussian networks. In *Proceedings of the Tenth international conference on Uncertainty in artificial intelligence*, pages 235–243. Morgan Kaufmann Publishers Inc., 1994.

Jiaying Gu, Fei Fu, and Qing Zhou. Penalized estimation of directed acyclic graphs from discrete data. *Statistics and Computing*, 29(1):161–176, 2019.

John A Hartigan. Consistency of single linkage for high-density clusters. *Journal of the American Statistical Association*, 76(374):388–394, 1981.

David Heckerman, Dan Geiger, and David M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

Markus Kalisch and Peter Bühlmann. Estimating high-dimensional directed acyclic graphs with the pc-algorithm. *Journal of Machine Learning Research*, 8:613–636, 2007.

Markus Kalisch, Martin Mächler, Diego Colombo, Marloes H Maathuis, and Peter Bühlmann. Causal inference using graphical models with the R package pcalg. *Journal of Statistical Software*, 47(11):1–26, 2012.

Steffen L Lauritzen. *Graphical models*. Oxford University Press, 1996.

Thuc Duy Le, Tao Hoang, Jiuyong Li, Lin Liu, and Shu Hu. ParallelPC: an R package for efficient constraint based causal exploration. *arXiv preprint*, 1510.03042, 2015. URL `http://arxiv.org/pdf/1510.03042.pdf`.

Yuliya Marchetti and Qing Zhou. Iterative subsampling in solution path clustering of noisy big data. *Statistics and Its Interface*, 9(4):415–431, 2016.

Radhakrishnan Nagarajan and Marco Scutari. *Bayesian Networks in R with Applications in Systems Biology*. Springer, New York, 2013. doi: 10.1007/978-1-4614-6446-4. ISBN 978-1-4614-6445-7, 978-1-4614-6446-4.

Kristian G Olesen and Anders L Madsen. Maximal prime subgraph decomposition of bayesian networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 32(1):21–31, 2002.

Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2000.

Joseph Ramsey, Madelyn Glymour, Ruben Sanchez-Romero, and Clark Glymour. A million variables and more: the fast greedy equivalence search algorithm for learning high-dimensional graphical causal models, with an application to functional magnetic resonance images. *International Journal of Data Science and Analytics*, 3(2):121–129, 2017.

Robert W Robinson. Counting unlabeled acyclic digraphs. In *Combinatorial Mathematics V*, pages 28–43. Springer, 1977.

Marco Scutari. Learning bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3):1–22, 2010. doi: 10.18637/jss.v035.i03.

Marco Scutari. Bayesian network constraint-based structure learning algorithms: Parallel and optimized implementations in the bnlearn R package. *Journal of Statistical Software*, 77(2):1–20, 2017. doi: 10.18637/jss.v077.i02.

Marco Scutari and Jean-Baptiste Denis. *Bayesian Networks with Examples in R*. Chapman and Hall, Boca Raton, 2014. ISBN 978-1-4822-2558-7, 978-1-4822-2560-0.

Peter Spirtes and Clark Glymour. An algorithm for fast recovery of sparse causal graphs. *Social Science Computer Review*, 9(1):62–72, 1991.

Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search*. Springer-Verlag, 1993.

Ioannis Tsamardinos, Constantin F Aliferis, and Alexander 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*, pages 673–678. ACM, 2003.

Ioannis Tsamardinos, Laura E Brown, and Constantin F Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78, 2006.

Tomas Verma and Judea Pearl. Equivalence and synthesis of causal models. In *Sixth Annual Conference on Uncertainty in Artificial Intelligence*, pages 220–227, 1990.

Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

Jing Xiang and Seyoung Kim. A* lasso for learning a sparse bayesian network structure for continuous variables. In *Advances in Neural Information Processing Systems*, pages 2418–2426, 2013.

Yiping Yuan, Xiaotong Shen, Wei Pan, and Zizhuo Wang. Constrained likelihood for reconstructing a directed acyclic Gaussian graph. *Biometrika*, 106:109–125, 2019.

Yi-feng Zeng and Kim-leng Poh. Block learning bayesian network structure from data. In *Fourth International Conference on Hybrid Intelligent Systems (HIS'04)*, pages 14–19. IEEE, 2004.

Xun Zheng, Bryon Aragam, Pradeep K Ravikumar, and Eric P Xing. Dags with no tears: Continuous optimization for structure learning. In *Advances in Neural Information Processing Systems*, pages 9472–9483, 2018.