UNIVERSITY OF CALIFORNIA

Los Angeles

Structure Learning of Bayesian Networks: Group Regularization and Divide-and-Conquer

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Statistics

by

Jiaying Gu

2018

ABSTRACT OF THE DISSERTATION

Structure Learning of Bayesian Networks:
Group Regularization and Divide-and-Conquer

by

Jiaying Gu
Doctor of Philosophy in Statistics
University of California, Los Angeles, 2018
Professor Qing Zhou, Chair

Bayesian networks, with structure given by a directed acyclic graph (DAG), are a popular class of graphical models. In this dissertation, we develop two structure learning methods for Bayesian networks.

First we propose a score-based algorithm for discrete data that can incorporate experimental intervention for causal learning. Learning Bayesian networks from discrete or categorical data is particularly challenging, due to the large parameter space and the difficulty in searching for a sparse structure. In this thesis, we develop a maximum penalized likelihood method to tackle this problem. Instead of the commonly used multinomial distribution, we model the conditional distribution of a node given its parents by multi-logit regression, in which an edge is parameterized by a set of coefficient vectors with dummy variables encoding the levels of a node. To obtain a sparse DAG, a group norm penalty is employed, and a blockwise coordinate descent algorithm is developed to maximize the penalized likelihood subject to the acyclicity constraint of a DAG. When interventional data are available, our method constructs a causal network, in which a directed edge represents a causal relation. We apply our method to various simulated and real data sets. The results show that our method is very competitive, compared to many existing methods, in DAG estimation from both interventional and high-dimensional observational data. In addition, an **R** package **discretecdAlgorithm** for this algorithm is published on CRAN, with a user friendly interface.

The second method we propose is a framework for fast learning of extremely large Bayesian networks. The size of the DAG space grows super-exponentially in the number of nodes, and it is challenging to develop an efficient structure learning method for massive networks. We propose a three step divide-and-conquer framework for Gaussian observational data: 1) partition the full DAG into several sub-networks (P-step), 2) estimate each disconnected sub-network individually (E-setp), 3) fuse sub-networks by adding edges between them (F-step). To partition the full DAG into some sub-networks, a modified hierarchical clustering with average linkage is proposed to automatically choose the number of sub-networks. For the estimation step, users can apply any proper structure learning algorithm; in our implementation we choose to use the Concave penalized Coordinate Descent with reparameterization (CCDr) algorithm (Aragam and Zhou, 2015) as an example. Finally we develop a hybrid fusion step that uses both conditional independence tests and a penalized likelihood scoring function to recover the structure of the full DAG. The simulation results show that our three step framework had significant improvement in both speed and accuracy compared to estimating the DAG as a whole with the algorithm used in the second estimation step.

The dissertation of Jiaying Gu is approved.

Adnan Youssef Darwiche

Yingnian Wu

Mark Stephen Handcock

Qing Zhou, Committee Chair

University of California, Los Angeles

2018

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

VITA

2009–2013	B.S., Department of Mathmatics, Major in Information and Computing Sciense, Fudan University, Shanghai, China.
2013–present	Expected Ph.D., Major in Statistics, University of California-Los Angeles.

PUBLICATIONS

Aragam, B., Gu, J., and Zhou, Q. (2018). Learning large-scale Bayesian networks with the sparsebn package. *Journal of Statistical Software*, to appear.

Gu, J., Fu, F., and Zhou, Q. (2018). Penalized estimation of directed acyclic graphs from discrete data. *Statistics and Computing*, DOI: 10.1007/s11222-018-9801-y.

Aragam, B., Gu, J., Amini, A. A., and Zhou, Q. (2017). Learning high-dimensional DAGs: Provable statistical guarantees and scalable approximation. *NIPS Workshop on Advances in Modeling and Learning Interactions from Complex Data.*

CHAPTER 1

Introduction

1.1 Bayesian Networks

Bayesian networks are a class of probabilistic graphical models whose structure encodes the conditional independence relationships among a set of random variables. A Bayesian network can be graphically represented by a directed acyclic graph (DAG). It is constructed with nodes and directed edges. Each node represents a random variable and each edge can represent the causal influence between two nodes. Recent years have seen its popularity in the biological and medical sciences for inferring gene regulatory networks and cellular networks, partially attributed to the fact that it can be used for causal inference. Learning the structure of biological networks from data is a key to understanding their functions. In biological networks, each protein can be regarded as a node in a Bayesian network, and their causal relationships as edges. If there is no protein-protein interaction between two nodes of interest, they are not directly related and consequently there is no edge between them.

Due to the directionality, Bayesian networks are capable of representing causality among variables. There is a natural interpretation of the cause-effect relationship encoded in the structure of Bayesian networks, where parents can serve as "cause" and children as "effect". This dissertation aims to find efficient methods to recover the structure of Bayesian networks. Firstly, we develop a method for discrete networks to learn the causal relationship from datasets with help of interventions. We develop a coordinate descent algorithm that is able to take as input a mixture of observational and interventional data. We will discuss details of causal learning and interventions in the following Section 1.2.2. And then, we propose a method for fast learning of massive Gaussian Bayesian networks from observational data.

Mathematical representation of the structure of a Bayesian network for p random variables $X_1, \ldots, X_p$ is given by a DAG $\mathcal{G} = (V, E)$. The set of nodes $V = \{1, \ldots, p\}$ represents the set of random variables $\{X_1, \ldots, X_p\}$, and the set of edges is given by $E = \{(j, i) \in V \times V : j \to i\}$, where $j \to i$ is a directed edge in $\mathcal{G}$. Given the structure of $\mathcal{G}$, the joint probability density (mass) function of $(X_1, \ldots, X_p)$ can be factorized as

$$P(x_1, \ldots, x_p) = \prod_{i=1}^{p} P(x_i | \Pi_i^{\mathcal{G}}), \tag{1.1}$$

where $\Pi_i^{\mathcal{G}} = \{j \in V : (j, i) \in E\}$ is called the set of parents of X_i and $P(x_i | \Pi_i^{\mathcal{G}})$ denotes the conditional probability density (CPD) of X_i given $\Pi_i^{\mathcal{G}}$. Throughout this thesis, we use i and X_i interchangeably.

Let $A^{\mathcal{G}} \in \{0, 1\}^{p \times p}$ be the adjacency matrix for a DAG $\mathcal{G}$, it is defined as $A^{\mathcal{G}} = (a_{ij})_{p \times p}$, where

$$a_{ij} = \begin{cases} 1, & (i, j) \in E \\ 0, & (i, j) \notin E \end{cases}.$$

Due to its directed acyclic nature, every DAG has at least one topological sort. If an ordering $\sqsubset$ is a topological sort of a DAG $\mathcal{G}$, we say that $\mathcal{G}$ and $\sqsubset$ are compatible. Definition of topological sort is given as follow,

Definition 1 (Topological sorts)**.** A topological sort of a DAG $\mathcal{G}$ is a linear ordering $\sqsubset$ of its nodes such that $i \prec j$ in $\sqsubset$ if $i \in \Pi_j^{\mathcal{G}}$.

We can understand ordering with the help of permutation. Let π be a permutation on the index set of nodes $\{1, \ldots, p\}$ for a DAG $\mathcal{G}$, and P_π the operation that permutes the adjacency matrix $A^{\mathcal{G}}$ according to π. For each ordering $\sqsubset$ compatible with $\mathcal{G}$, there exists a permutation π such that $P_\pi(A^{\mathcal{G}})$ has column and row indexes as $\sqsubset$, and the resulting matrix becomes a strictly upper triangular matrix. This means a node X_i can only have nodes ahead of it in $\sqsubset$ as its parents. For a DAG $\mathcal{G}$ there might exist more than one topological sort, and therefore we may find multiple permutations to make $A^{\mathcal{G}}$ a strictly upper triangular matrix.

1.2 Background: Some Fundamental Concepts and Theorems

In this section, we will go through some fundamental concepts and theorems in structure learning of DAGs. We will discuss the linkage between causal learning with the conditional probability density (1.1) and the conditional independence among variables, and how all these are related to the structure of Bayesian networks. We first define the *conditional independence* for random variables.

Definition 2 (Conditional independence)**.** Let X, Y be two random variables and $\boldsymbol{Z}$ a set of random variables that follows a probability distribution P, then X are Y are conditionally independent given $\boldsymbol{Z}$ if $P(X = x, Y = y|\boldsymbol{Z} = \boldsymbol{z}) = P(X = x|\boldsymbol{Z} = \boldsymbol{z})P(Y = y|\boldsymbol{Z} = \boldsymbol{z})$, $\forall \boldsymbol{z}$ where $P(\boldsymbol{Z} = \boldsymbol{z}) > 0$. We denote this conditional independence between X and Y as $\mathcal{I}_P(X; Y|\boldsymbol{Z})$.

The *local Markov properties* can serve as an alternative definition for Bayesian Networks.

Definition 3 (Local Markov properties)**.** Let $\mathcal{G}$ be an directed acyclic graph, V the set of nodes for $\mathcal{G}$ and P the probability distribution over V that generated according to the structure of $\mathcal{G}$. We say P has the local Markov properties with respect to $\mathcal{G}$ if for all X in V, X is conditionally independent of all its non-descendant nodes in V given the parent set of X, $\Pi_X^{\mathcal{G}}$.

We can see from (1.1) that Bayesian networks are graphs equipped with probability density P such that the pair $(\mathcal{G}, P)$ satisfies the local Markov condition. If we examine the structure of $\mathcal{G}$, there exists equivalent class known as *Markov equivalence class*, and it is related to the phenomenon called *observational equivalence*. Section 1.2.1 will discuss Markov equivalence and Section 1.2.2 will discuss observational equivalence and show how experimental intervention can help distinguishing observational equivalent DAGs.

1.2.1 Markov equivalence

From the local Markov properties we know that a node i is conditional independent of all its non-descendants given its parent set $\Pi_i^{\mathcal{G}}$. As a matter of fact, more conditional independence information is encoded in the graphical structure of $\mathcal{G}$ and can be tested using *d-separation.*

d-separation describes independencies encoded in DAGs, which is based on the concept of *blocked path.* This graphical criterion gives us a way to read off conditional independence among random variables given a DAG structure. Note that the d-separation test is solely based on the DAG structure and is irrelevant of the probability distribution vertices follow. The following definition of v-structure, blocked paths, and d-separation are adapted from the definition for d-separation in Pearl (1995).

Definition 4 (v-structure)**.** A v-structure is a triplet $\{i, j, k\} \subset V$ of the form $i \to k \leftarrow j$, while i and j are not directly connected, and k is called a *collider.*

Definition 5 (Blocked path)**.** A *path* p between nodes X and Y is a succession of arcs from X to Y, regardless of their directions. A set of random variables $\boldsymbol{Z}$ is said to *block* p if there exists a node D on the path that satisfies any of the two conditions :

(i) D is not a collider, and $D \in \boldsymbol{Z}$.

(ii) D is a collider, and neither D nor any of its descendants are in $\boldsymbol{Z}$.

Definition 6 (d-separation)**.** If $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$ are three disjoint subsets of V in a DAG $\mathcal{G}$, and let p be any undirected path between $x \in \boldsymbol{X}$ and $y \in \boldsymbol{Y}$. Than $\boldsymbol{Z}$ d-separates $\boldsymbol{X}$ from $\boldsymbol{Y}$, denoted by $\mathcal{D}_{\mathcal{G}}(\boldsymbol{X};\boldsymbol{Y}|\boldsymbol{Z})$, if and only if every path from $\boldsymbol{X}$ to $\boldsymbol{Y}$ is blocked by $\boldsymbol{Z}$.

With the concept of d-separation, we can define Markov equivalent classes as follow:

Definition 7 (Markov equivalence)**.** Let $\mathcal{G}$ and $\mathcal{G}'$ be two DAGs with the same set of nodes V. They are *Markov equivalent* if for any three disjoint subsets $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z} \subseteq V$, the following equivalence holds:

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

Given the definition of Markov equivalence, Theorem 1 gives us a convenient way to examine if two DAGs are Markov equivalent.

Theorem 1 (Verma and Pearl (1990))**.** *Two DAGs $\mathcal{G}$ and $\mathcal{G}'$ are Markov equivalent if and only if they have the same skeletons and the same v-strucures.*

d-separation $\mathcal{D}_{\mathcal{G}}(\cdot)$ is a test for the network structure on a DAG $\mathcal{G}$ while conditional independence $\mathcal{I}_P(\cdot)$ describes relationship of random variables given a probability distribution P. An assumption of faithfulness (Spirtes et al, 1993) is required to build up the equivalence between these two concepts.

Definition 8 (Faithfulness)**.** Suppose $\mathcal{G}$ is a DAG equipped with joint probability distribution P, then $\mathcal{G}$ and P are *faithful* to each other if and only if all conditional independence relations implied by P is entailed to $\mathcal{G}$ based on the Markov condition. We can also say that $(\mathcal{G}, P)$ satisfies the faithfulness condition.

From the definition of faithfulness, it is easy to arrive at Theorem 2 (Pearl, 2014),

Theorem 2. *If a Bayesian network $\mathcal{G}$ and the joint probability distribution P on V are faithful to each other, then,*

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

where $X, Y \in V$ and $\boldsymbol{Z} \subseteq V$.

If $(\mathcal{G}, P)$ satisfies the faithfulness condition, we can use conditional independence test (CI-test) to test for the *d*-separation in $\mathcal{G}$. And Theorem 3 (Spirtes et al, 1993) provides a method to determine the existence of an edge using CI-test. Therefore faithfulness is required in many of the structure learning algorithms, especially constraint-based methods and hybrid methods (there will be a detailed review of structure learning methods in Section 1.3). However sometimes faithfulness condition can be hard to enforce in practice, and in this thesis we propose a score-based algorithm that does not require the faithfulness condition in Section 2.

Theorem 3. *Suppose $(\mathcal{G}, P)$ satisfies the faithfulness condition, then there is no edge between a pair of nodes $X, Y \in V$ if and only if there exists a subset $\boldsymbol{Z} \subseteq V$ such that $\mathcal{I}_P(X; Y|\boldsymbol{Z})$.*

Theorem 3 gives us some insights in studying the skeleton of Bayesian networks, but recovering the structure of a Bayesian network also require us to decide the direction of edges. So, given a dataset, to what extend can we recover the structure of a DAG? We will discuss this problem in the next subsection.

1.2.2 Causal learning and interventional data

Given a joint distribution, there may exist multiple factorizations of the form in (1.1), leading to different DAGs. DAGs encoding the same set of joint distributions form an equivalence class of a Bayesian network, and this is also known as *"distributionally equivalent class"*. And this leads to the observational equivalent phenomenon, where with only observational data, equivalent DAGs might not be distinguished even with an infinite number of observations. The distributionally equivalence and the Markov equivalence coincides for data sets generated from multivariate Normal distribution and multinomial distributions. In Chapter 4 we propose a framework for massive size Gaussian network for observational data set and we do not distinguish between this two equivalence in that chapter. When used for causal inference, equivalent DAGs do not have the same causal interpretation. Luckily, equivalent DAGs can be differentiated using experiments, causal learning with interventional data will be introduced in the remaining of this subsection.

In order to distinguish different causal interpretations among equivalent DAGs, there are methods for learning causal relations from a mix of observational and experimental data (Cooper and Yoo, 1999; Meganck et al, 2006; Ellis and Wong, 2008) and related work on inferring gene networks from perturbed expression data (Peér et al, 2001; Pournara and Wernisch, 2004). In our research, we use interventional data to further explore causal relationships in a DAG. Experimental interventions reveal causality among a set of variables by breaking down various connections in the underlying causal network, and it can help distinguish equivalent DAGs.

First, let us use a simple example to illustrate how experimental intervention works: Consider DAGs with only two nodes, then the following two DAGs are equivalent: $\mathcal{G}_1$: $X_1 \leftarrow X_2$ and $\mathcal{G}_2$: $X_1 \rightarrow X_2$. Joint distributions for both DAGs are the same $P(X_1|X_2)P(X_2) = P(X_1, X_2) = P(X_2|X_1)P(X_1)$ Suppose we have two sets of interventions, i) intervene on X_1 and draw it from a known distribution $u(X_1)$; ii) intervene on X_2 and draw it from a known distribution $u(X_2)$. For those observations under experiment i), the directed edge in $\mathcal{G}_1$ has been cut off by the intervention and the joint distribution becomes $u(X_1)P(X_2)$, while the network structure in $\mathcal{G}_2$ stays the same and the joint distribution is still $u(X_1)P(X_2|X_1)$. Similarly, for those observations under experiment ii), the network structure in $\mathcal{G}_1$ stays the same with joint distribution $u(X_2)P(X_1|X_2)$, while for $\mathcal{G}_2$ the directed edge has been cut off so the joint distribution becomes $u(X_2)P(X_1)$. From this example we see that using experimental intervention, one can distinguish equivalent DAGs.

We describe below how the joint distribution of a Bayesian network (1.1) can be modified to incorporate experimental data. Briefly, assuming X_i, $i \in \mathcal{M} \subset \{1, \ldots, p\}$, is under experimental intervention, the joint density in (1.1) becomes

$$P(x_1, \ldots, x_p) = \prod_{i \notin \mathcal{M}} P(x_i|\Pi_i^{\mathcal{G}}) \prod_{i \in \mathcal{M}} P(x_i|\bullet), \tag{1.2}$$

where $P(x_i|\bullet)$ specifies the distribution of X_i under intervention. Experimental data generated from $\mathcal{G}$ can therefore be considered as being generated from the DAG $\mathcal{G}'$ obtained by removing all directed edges in $\mathcal{G}$ pointing to the variables under intervention. It should be noted that (1.2) also applies to observational data for which $\mathcal{M}$ is simply empty, and in this case, (1.2) reduces to (1.1). This parameterization also makes it easy to incorporate interventional data and observational data, and in Section 2 we develop our method under the assumption that part of the data are generated under experimental intervention, while regarding purely observational data as the special case of $\mathcal{M} = \varnothing$ for all data points. In addition, since distribution of nodes under intervention is already known, $\prod_{i \in \mathcal{M}} P(x_i|\bullet)$ in (1.2) is a constant and can be ignored.

It is worth noticing that, even with interventional data, causal effects encoded in DAGs may not be identifiable because of *interventional Markov equivalence* among DAGs (Hauser and Bühlmann, 2012). Denote $\mathcal{G}^{\mathcal{M}}$ as the DAG obtained by experimentally fixing nodes in $\mathcal{M} \subset \{1, ..., p\}$ from $\mathcal{G}$. Let $\{\mathcal{M}_i\}_N$ be the set of interventions for N data points, obviously if a variable $X_j \in V$ is under intervention for all $\mathcal{M}_i, i = 1, ..., p$, there is no way for us to learn the causal relationship between X_j and its parents. Hauser and Bühlmann (2012) named the interventon set $\{\mathcal{M}_i\}_N$ to be *conservative* if for all $X_j \in V$, there exists at least one experiment $\mathcal{M} \in \{\mathcal{M}_i\}_N$ that $X_j \notin \mathcal{M}$. Theorem 4 discusses equivalent DAGs given interventional data.

Theorem 4. *Suppose $\mathcal{G}_1$ and $\mathcal{G}_2$ are DAGs on V, and $\{\mathcal{M}_i\}_N$ is a conservative intervention set. Then, the following four statements are equivalent:*

a). $\mathcal{G}_1$ and $\mathcal{G}_2$ are interventional Markov equivalent with respect to intervention set $\{\mathcal{M}_i\}_N$

b). $\forall \mathcal{M} \in \{\mathcal{M}_i\}_N$, $\mathcal{G}_1^{\mathcal{M}}$ and $\mathcal{G}_2^{\mathcal{M}}$ are Markov equivalent.

c). $\forall \mathcal{M} \in \{\mathcal{M}_i\}_N$, $\mathcal{G}_1^{\mathcal{M}}$ and $\mathcal{G}_2^{\mathcal{M}}$ have the same skeleton and v-structure.

d). $\mathcal{G}_1$ and $\mathcal{G}_2$ are Markov equivalent, and $\forall \mathcal{M} \in \{\mathcal{M}_i\}_N$, $\mathcal{G}_1^{\mathcal{M}}$ and $\mathcal{G}_2^{\mathcal{M}}$ have the same skeleton.

Eberhardt et al (2012) proposed a sufficient condition on the number of interventions needed to discover all causal relationships encoded in a Bayesian network. They stated that if any number of nodes are allowed to be simultaneously and independently intervened on in every data point, $\log_2(p) + 1$ experiments are sufficient and in the worst case necessary to recover all the causal relationships in a Bayesian network for all $p \geq 2$. Based on their work, Eberhardt (2012) proposed an algorithm to determine the optimal intervention sets $\mathcal{M}_i, i = 1, ..., N$ for N data points. In addition, if an upper limit of k_{max} interventions are allowed for each data point, Eberhardt et al (2012) suggested that for $k_{max} < \frac{p}{2}$, $\left(\frac{p}{k_{max}} - 1\right) + \frac{p}{2k_{max}} \log_2(k_{max})$ is the sufficient and in the worst case necessary number of experiments needed to recover all the causal relationships. This means that, if up to 1 variable can be intervened for every data point, $p - 1$ experiments are sufficient for a complete causal learning. In the simulation tests of our CD algorithm for interventional data in Section 2 we use p sets of experiments of single variable intervention, which is sufficient.

1.3 Rescent Developments in Structure Learning Methods

The various algorithms in the literature for structure learning of Bayesian networks fall into three main categories: constraint-based methods, score-based methods and hybrid methods.

Constraint-based methods

Constraint-based methods rely on repeated conditional independence tests in order to learn the structure of a network. The main idea is to determine which edges cannot exist in a DAG using statistical tests of independence, a procedure which is justified whenever the so-called *faithfulness* assumption holds. These algorithms first use independence tests to learn the skeleton of the network, and then orient v-structures along with the rest of the edges. Because of the existence of Markov equivalent DAGs, the direction of some edges may not be decided (for more details, see e.g., Koller and Friedman (2009)). The PC algorithm proposed by Spirtes et al (1993) and the MMPC algorithm proposed by Tsamardinos et al (2006) are two well-known examples. Another example is the Fast Causal Inference (FCI) algorithm (Spirtes et al, 1993; Colombo et al, 2012), which allows for latent variables in the network. The output of these algorithms is a partially directed graph, which means that there may be some undirected edges in the estimated graph. While the PC algorithm is a powerful method to learn Bayesian networks in low-dimensions with n very large, the performance of the PC algorithm is less competitive in high-dimensions compared to recent score-based methods (Aragam and Zhou, 2015).

Score-based methods

Score-based methods rely on scoring functions such as the log-likelihood or some other loss functions. The goal of these algorithms is to find a DAG that optimizes a given scoring function. Some popular scoring functions include several Bayesian Dirichlet metrics (Buntine, 1991; Cooper and Herskovits, 1992; Heckerman et al, 1995), Bayesian information criterion (Chickering and Heckerman, 1997), minimum description length (Bouckaert, 1993; Suzuki, 1993; Lam and Bacchus, 1994), and entropy (Herskovits and Cooper, 1990). One of the classic score-based methods is the *greedy hill climbing* (HC) algorithm (Russell and Norvig, 2016). This algorithm is very fast but one needs to carefully choose its scoring function, for BIC selects out too many edges in high-dimensional settings. For discrete networks, the K2 algorithm (Cooper and Herskovits, 1992) is another popular method, however, this method requires prior knowledge about the ordering of the network which is often unavailable in applications. There are also Monte Carlo methods (Ellis and Wong, 2008; Zhou, 2011; Niinimäki et al, 2016), which are quite accurate but also computationally demanding. This limits Monte Carlo methods to smaller networks with only tens of nodes.

With the rising interest in sparse statistical modeling, score-based methods seem particularly attractive since various sparse regularization techniques are potentially applicable to them. Assuming a given natural ordering among the nodes, Shojaie and Michailidis (2010) decomposed DAG estimation into a sequence of ℓ_1-penalized linear regression problems. Fu and Zhou (2013) recently developed an ℓ_1-penalized likelihood approach to structure estimation of sparse DAGs from Gaussian data without assuming a given ordering. This method has been further generalized to the use of concave penalties by Aragam and Zhou (2015). Other recent developments on penalized DAG estimation include the work of Schmidt et al (2007) and Xiang and Kim (2013). There is also theoretical development on score-based estimation of sparse high-dimensional DAGs under a multivariate Gaussian model (van de Geer and Bühlmann, 2013; Aragam et al, 2017a; Nandy et al, 2018).

Hybrid methods

Finally, there are hybrid methods which combine constraint-based and score-based methods. Hybrid methods first prune the search space by using a constraint-based search, and then learn an optimal DAG structure via score-based search (Tsamardinos et al, 2006; Perrier et al, 2008; Gámez et al, 2011). The max-min hill-climbing (MMHC) algorithm proposed by Tsamardinos et al (2006) is a powerful method of this kind. It first uses the MMPC algorithm to learn the skeleton of the Bayesian network, and then uses the HC algorithm to orient directions.

1.4 Outline and Overview

In this thesis, we propose two structure learning methods for sparse Bayesian networks. Remaining chapters of this thesis are structured as follow:

- Chapter 2 proposes a score-based method to learn causal discrete Bayesian networks based on regularized likelihood. The adaptive group lasso penalty is employed to encourage sparsity. A coordinate descent algorithm is implemented to solve the optimization problem. This chapter is an extension of a previous work of Fu and Zhou (2013) for continuous data.
- Chapter 3 is an introduction and manual for our **R** packages **discretecdAlgorithm** and **sparsebn**, where the **discretecdAlgorithm** contains the main program for the coordinate-descent (CD) algorithm described in Chapter 2, and **sparsebn** is a user-friendly wrapper package of several structure learning packages that users can easily interact with.
- Chapter 4 has extended our work to large graphs and propose a three-stage method to speed up the structure learning process of massive Bayesian networks. Since the possible number of DAGs on a variable set V grows super-exponentially in the number of nodes p, running time can be extremely long due to the computational complexity.
- Chapter 5 summarizes and discusses the two methods we have developed.

CHAPTER 2

Causal Learning with Categorical Data

Despite the recent fast developments on sparse regularization methods for learning Gaussian DAGs, a generalization to discrete data is highly nontrivial. First, each node now represents a factor coded by a group of dummy variables. In order to select a group of dummy variables together, we need to use a group norm penalty instead of penalizing individual coefficients. Second, the log-likelihood function for categorical data has more parameters, and development of an algorithm to maximize the penalized log-likelihood becomes much more challenging. In this chapter, we propose a principled generalization of the penalized likelihood methodology in our previous work (Fu and Zhou, 2013) to estimate sparse DAGs from categorical data without knowing the ordering among variables. To reduce the parameter space, we use a multi-logit regression to model the conditional distributions in a discrete Bayesian network. A blockwise coordinate descent (CD) algorithm is developed, which may take both observational and interventional data. Through extensive comparisons, we demonstrate that our method can outperform many competitors in learning discrete Bayesian networks from interventional data or from high-dimensional ($p > n$) observational data. Our algorithm has been implemented in the R package, **discretecdAlgorithm**, available on CRAN.

2.1 Discrete Bayesian Networks

In a discrete Bayesian network, each variable X_i is considered a factor with r_i levels, indexed by $\{1, \ldots, r_i\}$. The set of its parents $\Pi_i^{\mathcal{G}}$ has a total of $q_i = \prod_{j \in \Pi_i^{\mathcal{G}}} r_j$ possible joint states $\{\boldsymbol{\pi}_k : k = 1, \ldots, q_i\}$. Let $\Theta_{ijk} = P\left(X_i = j \mid \Pi_i^{\mathcal{G}} = \boldsymbol{\pi}_k\right)$. A discrete Bayesian network $\mathcal{G}$ may be parameterized by $\boldsymbol{\Theta} = \{\Theta_{ijk} \geq 0 : \sum_j \Theta_{ijk} = 1\}$ via a product multinomial model given the graph structure. The number of parameters in this product multinomial model is

$$N(\mathbf{\Theta}) = \sum_{i=1}^{p} r_i q_i = \sum_{i=1}^{p} r_i \prod_{j \in \Pi_i^{\mathcal{G}}} r_j.$$

If we assume that each variable has $\mathcal{O}(r)$ levels, then

$$N(\mathbf{\Theta}) = \mathcal{O}\left(\sum_{i=1}^{p} r^{1+|\Pi_i^{\mathcal{G}}|}\right), \tag{2.1}$$

which grows exponentially as the size of the parent set $|\Pi_i^{\mathcal{G}}|$ increases. To reduce the number of free parameters, we propose a multi-logit model for discrete Bayesian networks under which development of a penalized likelihood method is straightforward. For the same DAG structure, the number of parameters can be much smaller compared to the product multinomial model.

2.1.1 A multi-logit model

We encode the r_i levels of X_i, $i = 1, \dots, p$, by a group of $d_i = r_i - 1$ dummy variables, here one can choose an arbitrary level as a reference category. Let $\mathbf{x}_i = (x_{i1}, .., x_{id_i}) \in \{0,1\}^{d_i}$ be the group of dummy variables for X_i and $\mathbf{x} = (1, \mathbf{x}_1, \dots, \mathbf{x}_p)$ be a d-vector, where $d = 1 + \sum_{i=1}^{p} d_i$. Each $x_{i\ell}, j = 1, ..., d_i$ is an indicator of the ℓth level at the ith data point, where $x_{i\ell} = 1$ and $x_{ik} = 0, k \neq \ell$, if $X_i = \ell$. Note that for a node X_i taking the reference level, we have $\mathbf{x}_i = \mathbf{0}$.

For a discrete Bayesian network $\mathcal{G}$, we model the conditional distribution $[X_j | \Pi_j^{\mathcal{G}}]$, $j = 1, \dots, p$, by the following multi-logit regression model

$$\begin{aligned} P(X_j = \ell \mid \Pi_j^{\mathcal{G}}) &= \frac{\exp(\beta_{j\ell 0} + \sum_{i=1}^{p} \mathbf{x}_i^T \boldsymbol{\beta}_{j\ell i})}{\sum_{m=1}^{r_j} \exp(\beta_{jm0} + \sum_{i=1}^{p} \mathbf{x}_i^T \boldsymbol{\beta}_{jmi})} \\ &= \frac{\exp(\mathbf{x}^T \boldsymbol{\beta}_{j\ell\cdot})}{\sum_{m=1}^{r_j} \exp(\mathbf{x}^T \boldsymbol{\beta}_{jm\cdot})} \triangleq p_{j\ell}(\mathbf{x}), \end{aligned} \tag{2.2}$$

for $\ell = 1, \dots, r_j$, where $\beta_{j\ell 0}$ is the intercept, $\boldsymbol{\beta}_{j\ell i} \in \mathbb{R}^{d_i}$ is the coefficient vector for X_i to predict the ℓ^{th} level of X_j, and $\boldsymbol{\beta}_{j\ell\cdot} = \text{vec}(\beta_{j\ell 0}, \boldsymbol{\beta}_{j\ell 1}, \dots, \boldsymbol{\beta}_{j\ell p}) \in \mathbb{R}^d$. Note that in (2.2), $\boldsymbol{\beta}_{j\ell i} = \mathbf{0}$ for all ℓ if $i \notin \Pi_j^{\mathcal{G}}$. Thus, our model indeed defines a joint distribution for $X_1, \dots, X_p$ which factorizes according to the DAG $\mathcal{G}$. We choose to use a symmetric form of the multi-logit model here, as was done in Zhu and Hastie (2004) and Friedman et al (2010). To make this model identifiable, we impose the following constraints on the intercepts

$$\beta_{j10} = 0, \quad j = 1, \dots, p. \tag{2.3}$$

The nonidentifiability of other parameters can be resolved via regularization as demonstrated by Friedman et al (2010). The particular form of regularization we use leads to the following constraints

$$\sum_{m=1}^{r_j} \boldsymbol{\beta}_{jmi} = \mathbf{0}, \quad \forall\, i, j = 1, \dots, p. \tag{2.4}$$

Let $\boldsymbol{\beta} = (\boldsymbol{\beta}_{jmi})$, which is a four-way array, denote all the parameters. Given the structure of $\mathcal{G}$, the number of free parameters is

$$N(\boldsymbol{\beta}) = \sum_{j=1}^{p} \left[(r_j - 1) + r_j \sum_{i \in \Pi_j^{\mathcal{G}}} d_i \right]. \tag{2.5}$$

If we further assume that $r_i = \mathcal{O}(r)$ for all i, then

$$N(\boldsymbol{\beta}) = \mathcal{O}(r^2)|E| + \mathcal{O}(rp), \tag{2.6}$$

which grows linearly in the total number of edges $|E|$. This rate of growth is much slower than that of the product multinomial model (2.1). Note that these two models are not equivalent, and numerical comparisons in Section 2.5 confirm that the proposed multi-logit model often serves as a good approximation to the product multinomial model.

Suppose that we have a data set $\mathcal{X} = (\mathcal{X}_{hi})_{n \times p}$ generated from a causal discrete Bayesian network $\mathcal{G}$, where $\mathcal{X}_{hi}$ is the level of X_i in the h^{th} data point, $h = 1, \dots, n$, coded by dummy variables $\mathbf{x}_{h,i} \in \{0,1\}^{d_i}$. Let $\mathcal{I}_j$ be the index set of rows where X_i is under intervention and $\mathcal{O}_j = \{1, \dots, n\} \backslash \mathcal{I}_j$ be the index set of rows in which X_j is observational. Note that $\mathcal{I}_j$ are not necessarily mutually exclusive, which means there can be more than one node under intervention for a data point. Under the multi-logit model (2.2), the log-likelihood function $\ell(\boldsymbol{\beta})$ can be written according to the factorization (1.2) as

$$\begin{aligned} \ell(\boldsymbol{\beta}) &= \sum_{j=1}^{p} \sum_{h \in \mathcal{O}_j} \log \left[p(\mathcal{X}_{hj} | \mathbf{x}_{h,i}, i \in \Pi_j^{\mathcal{G}}) \right] \\ &= \sum_{j=1}^{p} \sum_{h \in \mathcal{O}_j} \left[\sum_{\ell=1}^{r_j} y_{hj\ell} \mathbf{x}_h^T \boldsymbol{\beta}_{j\ell\cdot} - \log \left\{ \sum_{m=1}^{r_j} \exp(\mathbf{x}_h^T \boldsymbol{\beta}_{jm\cdot}) \right\} \right], \end{aligned} \tag{2.7}$$

where $y_{hj\ell} = I(\mathcal{X}_{hj} = \ell)$ are indicator variables and $\boldsymbol{\beta}_{j\ell k} = \mathbf{0}$ for $k \notin \Pi_j^{\mathcal{G}}$.

Remark 1. Although we have assumed the availability of experimental data, it is easy to see that the log-likelihood (2.7) applies to observational data as well: If there are no experimental data for X_j, then $\mathcal{O}_j = \{1 , \ldots , n\}$ in (2.7).

2.1.2 Group norm penalty

Define $\boldsymbol{\beta}_{j\cdot i} = \mathrm{vec}(\boldsymbol{\beta}_{j1i}, \ldots, \boldsymbol{\beta}_{jr_ji}) \in \mathbb{R}^{d_i r_j}$ to be the vector of coefficients representing the influence of X_i on X_j and $\boldsymbol{\beta}_{j\cdot 0} = (\beta_{j10}, \ldots, \beta_{jr_j0}) \in \mathbb{R}^{r_j}$ to be the vector of intercepts for predicting X_j. The structure of $\mathcal{G}$ is coded by the sparsity of $\boldsymbol{\beta}_{j\cdot i}$ as

$$\boldsymbol{\beta}_{j\cdot i} = \mathbf{0} \qquad \Longleftrightarrow \qquad i \notin \Pi_j^{\mathcal{G}}. \tag{2.8}$$

In order to learn a sparse DAG from data, we estimate $\boldsymbol{\beta}$ via a penalized likelihood approach. It can be seen from (2.8) that, for discrete Bayesian networks, the set of parents of X_j is given by the set $\{i : \boldsymbol{\beta}_{j\cdot i} \neq \mathbf{0}\}$. The regular ℓ_1 penalty is inappropriate for this purpose since it penalizes each component of $\boldsymbol{\beta}$ separately. We instead penalize the vector $\boldsymbol{\beta}_{j\cdot i} \in \mathbb{R}^{d_i r_j}$ as a whole to obtain a sparse DAG via the use of a group norm penalty. Group norm penalties have been used in the group lasso and its generalizations (Yuan and Lin, 2006; Meier et al, 2008). Let $\mathcal{G}_{\boldsymbol{\beta}}$ denote the graph induced by $\boldsymbol{\beta}$ so that $\Pi_j^{\mathcal{G}_{\boldsymbol{\beta}}} = \{i : \boldsymbol{\beta}_{j\cdot i} \neq \mathbf{0}\}$ for $j = 1, \ldots, p$. We define our group norm penalized estimator for a discrete Bayesian network by the following optimization program:

$$f_\lambda(\boldsymbol{\beta}) \triangleq -\ell(\boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} \sum_{i=1}^{p} \|\boldsymbol{\beta}_{j\cdot i}\|_2, \tag{2.9}$$

$$\hat{\boldsymbol{\beta}}_\lambda = \underset{\boldsymbol{\beta}:\mathcal{G}_{\boldsymbol{\beta}} \text{ is a DAG}}{\arg\min} f_\lambda(\boldsymbol{\beta}), \tag{2.10}$$

where $\lambda > 0$ is a tuning parameter. See Section 2.2.3 for choosing the parameter λ. The feasible set of (2.10) is a DAG space, which imposes a highly nonconvex constraint. This is a major challenge for our optimization algorithm. Hereafter, we call $\boldsymbol{\beta}_{j\cdot i}$ a (component) group of $\boldsymbol{\beta}$.

2.2 Algorithm

Structure learning for discrete Bayesian networks is computationally demanding because of the nonlinear nature of the multi-logit model (2.2). We develop in this section a blockwise coordinate descent algorithm to solve (2.10). Coordinate descent algorithms have been proved successful in various settings (Fu, 1998; Friedman et al, 2007; Wu and Lange, 2008) and their implementations are relatively straightforward.

2.2.1 Single coordinate descent step

We first consider minimizing $f_\lambda(\boldsymbol{\beta})$ (2.9) with respect to $\boldsymbol{\beta}_{j\cdot i}$ while holding all the other parameters constant. We define

$$\begin{aligned} f_{\lambda,j}(\boldsymbol{\beta}_{j\cdot\cdot}) &= -\sum_{h\in\mathcal{O}_j}\left[\sum_{\ell=1}^{r_j} y_{hj\ell}\mathbf{x}_h^T\boldsymbol{\beta}_{j\ell\cdot} - \log\left\{\sum_{m=1}^{r_j}\exp\left(\mathbf{x}_h^T\boldsymbol{\beta}_{jm\cdot}\right)\right\}\right] + \lambda\sum_{k=1}^{p}\|\boldsymbol{\beta}_{j\cdot k}\|_2 \\ &\triangleq -\ell_j\left(\boldsymbol{\beta}_{j\cdot\cdot}\right) + \lambda\sum_{k=1}^{p}\|\boldsymbol{\beta}_{j\cdot k}\|_2, \end{aligned} \tag{2.11}$$

where $\boldsymbol{\beta}_{j\cdot\cdot} = (\boldsymbol{\beta}_{j\cdot 0}, \boldsymbol{\beta}_{j\cdot 1}, \ldots, \boldsymbol{\beta}_{j\cdot p})$. Considering the problem of minimizing $f_{\lambda,j}(\cdot)$ over $\boldsymbol{\beta}_{j\cdot i}$, we write $f_{\lambda,j}$ and ℓ_j as $f_{\lambda,j}(\boldsymbol{\beta}_{j\cdot i})$ and $\ell_j(\boldsymbol{\beta}_{j\cdot i})$, respectively.

Following the approach of Tseng and Yun (2009) and Meier et al (2008), we form a quadratic approximation to $\ell_j(\boldsymbol{\beta}_{j\cdot i})$ using the second-order Taylor expansion at $\boldsymbol{\beta}_{j\cdot i}^{(t)}$, the current value of $\boldsymbol{\beta}_{j\cdot i}$. Adding the penalty, the quadratic approximation is

$$\begin{aligned} Q_{\lambda,j}^{(t)}(\boldsymbol{\beta}_{j\cdot i}) &= -\left\{\left(\boldsymbol{\beta}_{j\cdot i} - \boldsymbol{\beta}_{j\cdot i}^{(t)}\right)^T \nabla\ell_j\left(\boldsymbol{\beta}_{j\cdot i}^{(t)}\right)\right. \\ &\quad \left. + \frac{1}{2}\left(\boldsymbol{\beta}_{j\cdot i} - \boldsymbol{\beta}_{j\cdot i}^{(t)}\right)^T \mathbf{H}_{ji}^{(t)}\left(\boldsymbol{\beta}_{j\cdot i} - \boldsymbol{\beta}_{j\cdot i}^{(t)}\right)\right\} \\ &\quad + \lambda\|\boldsymbol{\beta}_{j\cdot i}\|_2, \end{aligned} \tag{2.12}$$

up to an additive term that does not depend on $\boldsymbol{\beta}_{j\cdot i}$. The gradient of the log-likelihood function $\ell_j(\cdot)$ is

$$\nabla\ell_j(\boldsymbol{\beta}_{j\cdot i}^{(t)}) = \sum_{h\in\mathcal{O}_j}\begin{pmatrix} \left(y_{hj1} - p_{j1}^{(t)}(\mathbf{x}_h)\right)\mathbf{x}_{h,i} \\ \vdots \\ \left(y_{hjr_j} - p_{jr_j}^{(t)}(\mathbf{x}_h)\right)\mathbf{x}_{h,i} \end{pmatrix}, \tag{2.13}$$

where $p_{j\ell}^{(t)}(\mathbf{x})$, $\ell = 1, \ldots, r_j$, defined in (2.2) are evaluated at the current parameter values. To give a reasonable quadratic approximation, we use a negative definite matrix $\mathbf{H}_{ji}^{(t)} = h_{ji}^{(t)}\mathbf{I}_{d_i r_j}$ in (2.12) to approximate the Hessian of $\ell_j(\cdot)$, where the scalar $h_{ji}^{(t)} < 0$ and $\mathbf{I}_{d_i r_j}$ is the identity matrix of size $d_i r_j \times d_i r_j$. We choose

$$h_{ji}^{(t)} = h_{ji}(\boldsymbol{\beta}^{(t)}) \stackrel{\Delta}{=} -\max\{\text{diag}(-\mathbf{H}_{\ell_j}(\boldsymbol{\beta}_{j\cdot i}^{(t)})), b\}, \tag{2.14}$$

where $\mathbf{H}_{\ell_j}$ is the Hessian of the log-likelihood function $\ell_j(\cdot)$ and b is a small positive number used as a lower bound to help convergence. Note that it is not necessary to recompute $h_{ji}^{(t)}$ every iteration (Meier et al, 2008). See Section 2.2.3 for more details.

It is not difficult to show the following proposition, which is a direct consequence of the Karush-Kuhn-Tucker (KKT) conditions for minimizing (2.12).

Proposition 1. *Let $\mathbf{H}_{ji}^{(t)} = h_{ji}^{(t)}\mathbf{I}_{d_i r_j}$ for some scalar $h_{ji}^{(t)} < 0$ and $\boldsymbol{d}_{ji}^{(t)} = \nabla \ell_j(\boldsymbol{\beta}_{j\cdot i}^{(t)}) - h_{ji}^{(t)}\boldsymbol{\beta}_{j\cdot i}^{(t)}$. Then, the minimizer of $Q_{\lambda,j}^{(t)}(\boldsymbol{\beta}_{j\cdot i})$ in (2.12) is*

$$\bar{\boldsymbol{\beta}}_{j\cdot i}^{(t)} = \begin{cases} \mathbf{0} & \text{if } \|\boldsymbol{d}_{ji}^{(t)}\|_2 \le \lambda, \\ -\dfrac{1}{h_{ji}^{(t)}}\left[1 - \dfrac{\lambda}{\|\boldsymbol{d}_{ji}^{(t)}\|_2}\right]\boldsymbol{d}_{ji}^{(t)} & \text{otherwise.} \end{cases} \tag{2.15}$$

In order to achieve sufficient descent, an inexact line search by the Armijo rule is performed when $\bar{\boldsymbol{\beta}}_{j\cdot i}^{(t)} \neq \boldsymbol{\beta}_{j\cdot i}^{(t)}$, following the procedure in Meier et al (2008). Put $\boldsymbol{s}_{ji}^{(t)} = \bar{\boldsymbol{\beta}}_{j\cdot i}^{(t)} - \boldsymbol{\beta}_{j\cdot i}^{(t)}$, and let $\Delta^{(t)}$ be the change in $f_{\lambda,j}$ when the log-likelihood is linearized at $\boldsymbol{\beta}_{j\cdot i}^{(t)}$, i.e.,

$$\Delta^{(t)} = -(\boldsymbol{s}_{ji}^{(t)})^T \nabla \ell_j(\boldsymbol{\beta}_{j\cdot i}^{(t)}) + \lambda\|\bar{\boldsymbol{\beta}}_{j\cdot i}^{(t)}\|_2 - \lambda\|\boldsymbol{\beta}_{j\cdot i}^{(t)}\|_2.$$

Pick $\eta, \delta \in (0, 1)$ and $\alpha_0 > 0$, and let $\alpha^{(t)}$ be the largest value in the sequence $\{\alpha_0 \eta^k\}_{k \ge 0}$ such that

$$f_{\lambda,j}(\boldsymbol{\beta}_{j\cdot i}^{(t)} + \alpha^{(t)}\boldsymbol{s}_{ji}^{(t)}) \le f_{\lambda,j}(\boldsymbol{\beta}_{j\cdot i}^{(t)}) + \delta\alpha^{(t)}\Delta^{(t)}.$$

Then set

$$\boldsymbol{\beta}_{j\cdot i}^{(t+1)} = \boldsymbol{\beta}_{j\cdot i}^{(t)} + \alpha^{(t)}\boldsymbol{s}_{ji}^{(t)}, \tag{2.16}$$

which completes one iteration for updating $\boldsymbol{\beta}_{j\cdot i}$. In our implementation, we choose $\eta = 0.5$, $\delta = 0.1$ and $\alpha_0 = 1$ following the suggestion by Meier et al (2008).

It follows from Proposition 1 with $\lambda = 0$ that for the unpenalized intercepts,

$$\boldsymbol{\beta}_{j\cdot 0}^{(t+1)} = \bar{\boldsymbol{\beta}}_{j\cdot 0}^{(t)} = -\boldsymbol{d}_{j0}^{(t)}/h_{j0}^{(t)}. \tag{2.17}$$

In addition, some of the parameters are always constrained to zero, e.g., $\boldsymbol{\beta}_{j\cdot j}$ and β_{j10} for all j.

2.2.2 Blockwise coordinate descent

Our CD algorithm consists of two layers of iterations. In the outer loop, we cycle through all pairs of nodes to update the active set of edges, including their directions. In the inner loop, we only cycle through the active edge set to update the parameter values.

We first describe the outer loop. Due to the acyclicity constraint in (2.10), we know *a priori* that $\boldsymbol{\beta}_{i\cdot j}$ and $\boldsymbol{\beta}_{j\cdot i}$ cannot simultaneously be nonzero for $i \neq j$. This suggests performing the minimization in blocks, minimizing over $\{\boldsymbol{\beta}_{i\cdot j}, \boldsymbol{\beta}_{j\cdot i}\}$ simultaneously. In order to enforce acyclicity, we use a simple heuristic (Fu and Zhou, 2013): For each block $\{\boldsymbol{\beta}_{i\cdot j}, \boldsymbol{\beta}_{j\cdot i}\}$, we check if adding an edge from $i \to j$ induces a cycle in the estimated DAG. If so, we set $\boldsymbol{\beta}_{j\cdot i} = \mathbf{0}$ and minimize with respect to $\boldsymbol{\beta}_{i\cdot j}$. Alternatively, if the edge $j \to i$ induces a cycle, we set $\boldsymbol{\beta}_{i\cdot j} = \mathbf{0}$ and minimize with respect to $\boldsymbol{\beta}_{j\cdot i}$. If neither edge induces a cycle, we minimize over both parameters simultaneously. The cycle check is implemented by a breath-first search algorithm. We outline below (Algorithm 1) the complete blockwise CD algorithm for discrete Bayesian networks. In the algorithm, $\boldsymbol{\beta}_{j\cdot i} \Leftarrow \mathbf{0}$ is used to indicate that $\boldsymbol{\beta}_{j\cdot i}$ must be set to zero due to the acyclicity constraint given the current estimates of the other parameters. Minimization of $f_{\lambda,j}(\cdot)$ with respect to $\boldsymbol{\beta}_{j\cdot i}$ is done with the single CD step with line search (2.16).

Let $\boldsymbol{\beta}^{(t)}$ denote the parameter value after one cycle of the outer loop (after line 19 in Algorithm 1). Denote its active edge set by $E^{(t)} = \{(i,j) : \boldsymbol{\beta}_{j\cdot i}^{(t)} \neq \mathbf{0}\}$. The inner loop solves the following problem:

$$\min_{\boldsymbol{\beta}} f_\lambda(\boldsymbol{\beta}), \text{ subject to } \operatorname{supp}(\boldsymbol{\beta}) \subset E^{(t)}, \tag{2.18}$$

where f_λ is defined in (2.9). We use $\boldsymbol{\beta}^{(t)}$ as the initial value and cycle through $\boldsymbol{\beta}_{j\cdot i}$ for

$(i, j) \in E^{(t)}$. In particular, the direction of an edge will not be reversed but edges may be deleted if their parameters $\boldsymbol{\beta}_{j \cdot i}$ are updated to zero. See Algorithm 2 for an outline of the inner loop.

By construction, $E^{(t)}$ satisfies the acyclicity constraint and thus the feasible region in (2.18) is simply a Euclidean space. Since f_λ itself is convex, the CD algorithm for the inner loop has nice convergence properties. In analogy to Proposition 2 in Meier et al (2008), we arrive at the following convergence result.

Proposition 2. *Suppose that the sequence* $\{\boldsymbol{\beta}^{(k)}\}$ *is generated by the inner loop. If the matrix* $\mathbf{H}_{ji}^{(k)}$ *is chosen according to* (2.14)*, then every limit point of the sequence* $\{\boldsymbol{\beta}^{(k)}\}$ *is a minimizer of problem* (2.18)*.*

However, since the search space for the outer loop is the full DAG space, which is highly nonconvex, rigorous theory on its convergence is yet to be established. Therefore, a practical stopping criterion is employed. After the convergence of an inner loop, we obtain the current active set. If one more iteration of the outer loop does not change the active set, we then stop Algorithm 1. On the other hand, we also set a maximum number of iterations for the outer loop. For all the examples we have tested, our CD algorithm has shown no problem in convergence. For all the examples we have tested, the stopping criterion is met within five iterations of the outer loop, and our CD algorithm has shown no problem in convergence.

As an illustration, Figure 2.1 plots $\|\boldsymbol{\beta}^{(t+1)} - \boldsymbol{\beta}^{(t)}\|_\infty$ against t for a dataset generated from a small-world network with $p = 50$, where t indexes iterations in both the inner and the outer loops. In this figure there are three peaks, each appearing after the algorithm finishes an outer loop. The curve between two neighboring peaks demonstrates the convergence of the inner loop. The heights of the three peaks decay very fast, suggesting convergence of the outer loop. Similar patterns are observed for data generated from other types of DAGs.

This empirical observation is in line with recent theoretical work by Lee et al (2016) who have established that gradient descent converges to a local minimizer of a nonconvex objective function for almost all initial values and have suggested similar behavior for coordinate descent.

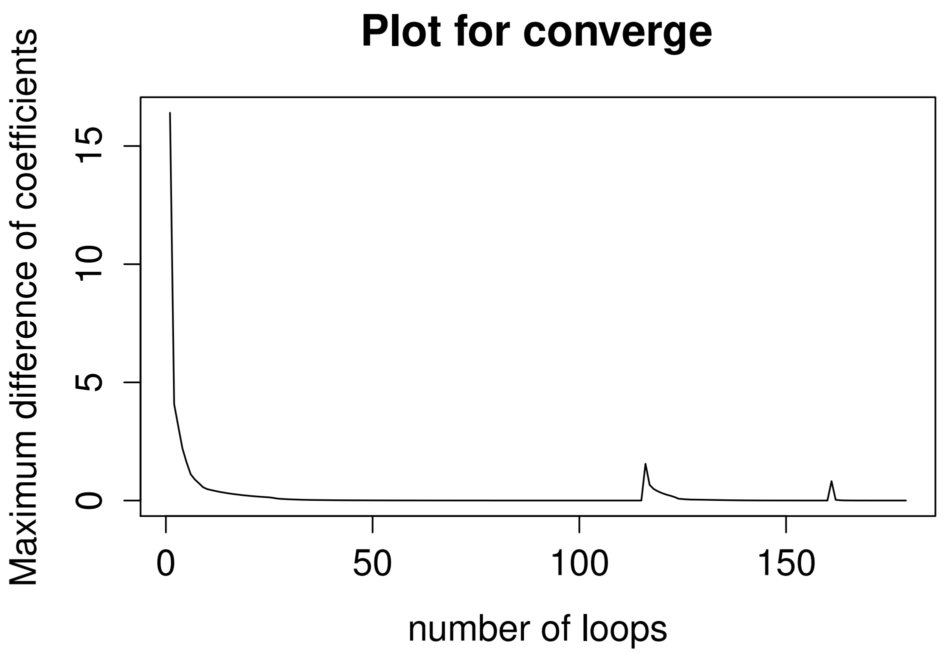


Figure 2.1: A typical convergence plot

2.2.3 Solution path

We use Algorithm 1 to compute $\hat{\boldsymbol{\beta}}_\lambda$ (2.10) over a grid of J values for the tuning parameter, $\lambda_1 > \ldots > \lambda_J > 0$, where at λ_1 every parameter other than the intercepts is estimated as zero. It follows from the KKT conditions for (2.9) that

$$\lambda_1 = \max_{1\le i,j\le p} \|\nabla \ell_j(\boldsymbol{\beta}_{j\cdot i})|_{\boldsymbol{\beta}_{j\cdot i}=\mathbf{0}}\|_2, \qquad (2.19)$$

in which $\boldsymbol{\beta}_{j\cdot 0}$ is set to the MLE of the intercept assuming all $\boldsymbol{\beta}_{j\cdot i}$, $i = 1, \ldots, p$, are zero.

The solution $\hat{\boldsymbol{\beta}}_{\lambda_m}$ is used as a warm start for estimating $\hat{\boldsymbol{\beta}}_{\lambda_{m+1}}$, $m = 1, \ldots, J-1$. To save computational time, we set $h_{ji}^{(t)} = h_{ji}(\hat{\boldsymbol{\beta}}_{\lambda_m})$ (2.14) in the CD algorithm for $\hat{\boldsymbol{\beta}}_{\lambda_{m+1}}$, instead of updating $h_{ji}^{(t)}$ every iteration.

Traditional model selection criteria such as BIC do not work well for the purpose of estimating DAGs from data. In our simulation results, the hill-climbing (HC) algorithm (Gámez et al, 2011), which uses BIC as the scoring function, always selects too many edges.

There are also numerical studies (Scutari, 2016) in which BIC tends to select too few edges on a different set of DAGs, showing that BIC could be sensitive and unstable. In order to select a suitable tuning parameter, we use an empirical model selection criterion proposed by Fu and Zhou (2013). Let $\hat{\mathcal{G}}_{\lambda_m}$ be the DAG induced by $\hat{\boldsymbol{\beta}}_{\lambda_m}$ and e_{λ_m} be the number of edges in $\hat{\mathcal{G}}_{\lambda_m}$. We reestimate $\boldsymbol{\beta}$ by the maximizer $\boldsymbol{\beta}^{\dagger}_{\lambda_m}$ of the log-likelihood $\ell(\boldsymbol{\beta})$ (2.7) given $\mathcal{G} = \hat{\mathcal{G}}_{\lambda_m}$ using the R package **nnet** (Venables and Ripley, 2002). We define the difference ratio between two estimated DAGs $\hat{\mathcal{G}}_{\lambda_m}$ and $\hat{\mathcal{G}}_{\lambda_{m+1}}$ by $dr_{(m,m+1)} = \Delta\ell_{(m,m+1)}/\Delta e_{(m,m+1)}$, where $\Delta\ell_{(m,m+1)} = \ell(\boldsymbol{\beta}^{\dagger}_{\lambda_{m+1}}) - \ell(\boldsymbol{\beta}^{\dagger}_{\lambda_m})$ and $\Delta e_{(m,m+1)} = e_{\lambda_{m+1}} - e_{\lambda_m}$, if $\Delta e_{(m,m+1)} \geq 1$. Otherwise, we set $dr_{(m,m+1)} = dr_{(m-1,m+1)}$. The selected tuning parameter is indexed by

$$\begin{aligned} m^* = \sup \quad \{ \quad & 2 \leq m \leq J : dr_{(m-1,m)} \\ & \geq \alpha \cdot \max\{dr_{(1,2)}, \ldots, dr_{(J-1,J)}\}\}. \end{aligned} \tag{2.20}$$

According to this criterion, an increase in model complexity, measured by the number of predicted edges, is accepted only if there is a substantial increase in the log-likelihood. We choose $\alpha = 0.3$ for all the result in this work.

2.3 Asymptotic Theory

In this section, we establish asymptotic theory for the DAG estimator $\hat{\boldsymbol{\beta}}_{\lambda}$ (2.10) assuming that p is fixed and $n \to \infty$. By rearranging and relabeling individual components, we rewrite $\boldsymbol{\beta}$ as $\boldsymbol{\phi} = (\boldsymbol{\phi}_{(1)}, \boldsymbol{\phi}_{(2)})$, where $\boldsymbol{\phi}_{(1)} = \text{vec}(\boldsymbol{\beta}_{1\cdot 1}, \ldots, \boldsymbol{\beta}_{1\cdot p}, \ldots\ldots, \boldsymbol{\beta}_{p\cdot 1}, \ldots, \boldsymbol{\beta}_{p\cdot p})$ is the parameter vector of interest and $\boldsymbol{\phi}_{(2)} = \text{vec}(\boldsymbol{\beta}_{1\cdot 0}, \ldots, \boldsymbol{\beta}_{p\cdot 0})$ denotes the vector of intercepts. Hereafter, we denote by ϕ_j the j^{th} group of $\boldsymbol{\phi}$, such that $\phi_1 = \boldsymbol{\beta}_{1\cdot 1}$, $\phi_2 = \boldsymbol{\beta}_{1\cdot 2}$, $\ldots, \phi_{p^2} = \boldsymbol{\beta}_{p\cdot p}$, and so on. We say $\boldsymbol{\phi}$ is acyclic if the graph $\mathcal{G}_{\boldsymbol{\phi}}$ induced by $\boldsymbol{\phi}$ (or the corresponding $\boldsymbol{\beta}$) is acyclic.

Define $\boldsymbol{\phi}_{[k]}$ $(k \in \{1, \ldots, p\})$ to be the parameter vector obtained from $\boldsymbol{\phi}$ by setting $\boldsymbol{\beta}_{k\cdot i} = \mathbf{0}$ for $i = 1, \ldots, p$. In other words, the DAG $\mathcal{G}_{\boldsymbol{\phi}_{[k]}}$ is obtained by deleting all edges pointing to the k^{th} node in $\mathcal{G}_{\boldsymbol{\phi}}$; see (2.8). We assume the data set $\mathcal{X}$ consists of $(p+1)$ blocks, denoted by $\mathcal{X}^j$ of size $n_j \times p$, $j = 1, \ldots, p+1$. The node X_j is experimentally fixed in $\mathcal{X}^j$ for the first p blocks, while the last block contains purely observational data. Let $\mathcal{I}_j$ be the set of row indices of $\mathcal{X}^j$. As demonstrated by (1.2), we can model interventional data in the k^{th} block of the data matrix $\mathcal{X}^k$ as *i.i.d.* observations from a joint distribution factorized according to $\mathcal{G}_{\boldsymbol{\phi}_{[k]}}$. Denote the corresponding probability mass function by $p(\mathbf{x}|\boldsymbol{\phi}_{[k]})$, where $\mathbf{x} = (x_1, \ldots, x_p)$ and $x_j \in \{1, \ldots, r_j\}$ for $j = 1, \ldots, p$. To simplify our notation, denote the parameter for the $(p+1)$th block by $\boldsymbol{\phi}_{[p+1]} = \boldsymbol{\phi}$. Then the log-likelihood of $\mathcal{X}$ is

$$L(\boldsymbol{\phi}) = \sum_{k=1}^{p+1} L_k(\boldsymbol{\phi}_{[k]}) = \sum_{k=1}^{p+1} \log p(\mathcal{X}^k \mid \boldsymbol{\phi}_{[k]}), \tag{2.21}$$

where $\log p(\mathcal{X}^k|\boldsymbol{\phi}_{[k]}) = \sum_{h \in \mathcal{I}_k} \log(p(\mathcal{X}_{h\cdot}|\boldsymbol{\phi}_{[k]}))$ and $\mathcal{X}_{h\cdot} = (\mathcal{X}_{h1}, \ldots, \mathcal{X}_{hp})$. The penalized log-likelihood function with a tuning parameter $\lambda_n > 0$ is

$$\begin{aligned} R(\boldsymbol{\phi}) &= L(\boldsymbol{\phi}) - \lambda_n \sum_{j=1}^{p^2} \|\phi_j\|_2 \\ &= \sum_{k=1}^{p+1} L_k(\boldsymbol{\phi}_{[k]}) - \lambda_n \sum_{j=1}^{p^2} \|\phi_j\|_2, \end{aligned} \tag{2.22}$$

where the component group ϕ_j $(j = 1, \ldots, p^2)$ represents the influence of one variable on another. Let $\Omega = \{\boldsymbol{\phi} : \mathcal{G}_{\boldsymbol{\phi}} \text{ is a DAG}\}$ be the parameter space. A penalized estimator $\hat{\boldsymbol{\phi}}$ is obtained by maximizing $R(\boldsymbol{\phi})$ in Ω.

Though interventional data help distinguish equivalent DAGs, the following notion of natural parameters is needed to completely establish identifiability of DAGs for the case where each variable has interventional data. We say that i is an ancestor of j in a DAG $\mathcal{G}$ if there exists at least one path from i to j. Denote the set of ancestors of j by $\text{an}(j)$.

Definition 9 (Natural parameters)**.** We say that $\boldsymbol{\phi} \in \Omega$ is natural if $i \in \text{an}(j)$ in $\mathcal{G}_{\boldsymbol{\phi}}$ implies that j is not independent of i under the joint distribution given by $\boldsymbol{\phi}_{[i]}$ for all $i, j = 1, \ldots, p$.

For a causal DAG, a natural parameter implies that the effects along multiple causal paths connecting the same pair of nodes do not cancel. This is a reasonable assumption for many real-world problems, and is much weaker than the faithfulness assumption. Under the faithfulness assumption, all conditional independence restrictions can be read off from d-separations in the DAG. If nodes i and j are independent in $\boldsymbol{\phi}_{[i]}$, then by faithfulness the nodes i and j must be separated by empty set and thus $i \notin \text{an}(j)$ in $\mathcal{G}_{\boldsymbol{\phi}_{[i]}}$. This implies that

$i \notin \mathrm{an}(j)$ in $\mathcal{G}_{\boldsymbol{\phi}}$ as well, by the construction of $\mathcal{G}_{\boldsymbol{\phi}_{[i]}}$. Indeed, we see that the faithfulness assumption implies the natural parameter assumption.

To establish asymptotic properties of our penalized likelihood estimator, we make the following assumptions:

(A1) The true parameter $\boldsymbol{\phi}^*$ is natural and an interior point of Ω.

(A2) The parameter $\boldsymbol{\theta}_j$ of the conditional distribution $[X_j|\Pi_j^{\mathcal{G}}; \boldsymbol{\theta}_j]$ is identifiable for each $j = 1, \ldots, p$. The log-likelihood function $\ell_j(\boldsymbol{\theta}_j) = \log p(x_j|\Pi_j^{\mathcal{G}}; \boldsymbol{\theta}_j)$ is strictly concave and continuously three times differentiable for any interior point.

Recall that the k^{th} block of our data, $\mathcal{X}^k$, can be regarded as an *i.i.d.* sample of size n_k from the distribution $p(\mathbf{x}|\boldsymbol{\phi}^*_{[k]})$ for all k, while we define $\boldsymbol{\phi}^*_{[p+1]} = \boldsymbol{\phi}^*$ for the last block of observational data.

Theorem 5. *Assume (A1) and (A2). If $p(\mathbf{x}|\boldsymbol{\phi}_{[k]}) = p(\mathbf{x}|\boldsymbol{\phi}^*_{[k]})$ for all possible $\mathbf{x}$ and all $k = 1, \ldots, p$, then $\boldsymbol{\phi} = \boldsymbol{\phi}^*$. Furthermore, if $n_k \gg \sqrt{n}$ for all $k = 1, \ldots, p$, then for any $\boldsymbol{\phi} \neq \boldsymbol{\phi}^*$,*

$$P(L(\boldsymbol{\phi}^*) > L(\boldsymbol{\phi})) \to 1 \quad \text{as } n \to \infty. \tag{2.23}$$

Theorem 6. *Assume (A1) and (A2). If $\lambda_n/\sqrt{n} \to 0$ and $n_k \gg \sqrt{n}$ for all $k = 1, \ldots, p$, then there exists a global maximizer $\hat{\boldsymbol{\phi}}$ of $R(\boldsymbol{\phi})$ such that $\|\hat{\boldsymbol{\phi}} - \boldsymbol{\phi}^*\|_2 = O_p(n^{-1/2})$.*

Before we show the proofs of the two theorems, recall a topological sort $\sqsubset$ for a DAG is a linear ordering, and it implies the ancient-descendant relationship in a Bayesian network. Due to its directed acyclic nature, every DAG has at least one topological sort. If an ordering $\sqsubset$ is a topological sort of a DAG $\mathcal{G}$, we say that $\mathcal{G}$ and $\sqsubset$ are compatible. Let $\mathcal{S}(\mathcal{G})$ denote the set of all topological sorts of a DAG $\mathcal{G}$.

Lemma 7. *For any $\boldsymbol{\phi}^0 \in \Omega$, there is a $\delta(\boldsymbol{\phi}^0) > 0$ such that if $\boldsymbol{\phi} \in \Omega$ and $\|\boldsymbol{\phi} - \boldsymbol{\phi}^0\|_2 < \delta(\boldsymbol{\phi}^0)$ then $\mathcal{S}(\mathcal{G}_{\boldsymbol{\phi}}) \cap \mathcal{S}(\mathcal{G}_{\boldsymbol{\phi}^0}) \neq \varnothing$.*

Proof of Lemma 7. If $\mathcal{G}_{\boldsymbol{\phi}^0}$ is an empty graph which is compatible with any ordering, the statement holds trivially. Otherwise, let $\delta(\boldsymbol{\phi}^0) = \frac{1}{2}\min_{j:\phi_j^0 \neq \mathbf{0}} \|\phi_j^0\|_2 > 0$, where ϕ_j^0 is the

j^{th} component group of $\boldsymbol{\phi}^0$ (see Appendix). If $\|\boldsymbol{\phi} - \boldsymbol{\phi}^0\|_2 < \delta(\boldsymbol{\phi}^0)$ and $\phi_i^0 \neq \mathbf{0}$ for some $i \in \{1, \ldots, p^2\}$, then $\phi_i \neq \mathbf{0}$ as well, since otherwise $\|\boldsymbol{\phi} - \boldsymbol{\phi}^0\|_2 \geq \|\phi_i^0\|_2 > \delta(\boldsymbol{\phi}^0)$. This implies that every edge in $\mathcal{G}_{\boldsymbol{\phi}^0}$ is also an edge in $\mathcal{G}_{\boldsymbol{\phi}}$ and thus $\mathcal{G}_{\boldsymbol{\phi}}$ and $\mathcal{G}_{\boldsymbol{\phi}^0}$ have at least one common topological sort. $\square$

This lemma shows that every DAG in a sufficiently small neighborhood of $\boldsymbol{\phi}^0$, denoted by $\text{nb}(\boldsymbol{\phi}^0) \subset \Omega$, has a topological sort that is also compatible with $\mathcal{G}_{\boldsymbol{\phi}^0}$.

Proof of Theorem 5. We prove the first claim by contradiction. Suppose $\boldsymbol{\phi} \neq \boldsymbol{\phi}^*$ and $p(\mathbf{x}|\boldsymbol{\phi}_{[k]}) = p(\mathbf{x}|\boldsymbol{\phi}^*_{[k]})$ for $k = 1, \ldots, p$. There are two cases to consider depending on the topological sorts of $\mathcal{G}_{\boldsymbol{\phi}}$ and those of $\mathcal{G}_{\boldsymbol{\phi}^*}$.

Case 1: $\mathcal{S}(\mathcal{G}_{\boldsymbol{\phi}}) \cap \mathcal{S}(\mathcal{G}_{\boldsymbol{\phi}^*}) \neq \varnothing$. Let $\sqsubset \in \mathcal{S}(\mathcal{G}_{\boldsymbol{\phi}}) \cap \mathcal{S}(\mathcal{G}_{\boldsymbol{\phi}^*})$, i.e., an ordering compatible with both $\mathcal{G}_{\boldsymbol{\phi}}$ and $\mathcal{G}_{\boldsymbol{\phi}^*}$. Assume without loss of generality that in this ordering $i \prec j$ if $i < j$. Apparently, $\sqsubset$ is also compatible with $\mathcal{G}_{\boldsymbol{\phi}_{[k]}}$ and $\mathcal{G}_{\boldsymbol{\phi}^*_{[k]}}$ for $k = 1, \ldots, p$. Then we can write $p(\mathbf{x}|\boldsymbol{\phi}_{[k]}) = \prod_{i=1}^p p(x_i|x_1, \ldots, x_{i-1}, \boldsymbol{\phi}_{[k]}) = \prod_{i=1}^p p(x_i|\Pi_i^{\mathcal{G}_{\boldsymbol{\phi}_{[k]}}}, \boldsymbol{\phi}_{[k]})$ and $p(\mathbf{x}|\boldsymbol{\phi}^*_{[k]}) = \prod_{i=1}^p p(x_i|x_1, \ldots, x_{i-1}, \boldsymbol{\phi}^*_{[k]}) = \prod_{i=1}^p p(x_i|\Pi_i^{\mathcal{G}_{\boldsymbol{\phi}^*_{[k]}}}, \boldsymbol{\phi}^*_{[k]})$. Since $p(\mathbf{x}|\boldsymbol{\phi}_{[k]}) = p(\mathbf{x}|\boldsymbol{\phi}^*_{[k]})$ for all possible $\mathbf{x}$, it follows that $\Pi_i^{\mathcal{G}_{\boldsymbol{\phi}_{[k]}}} = \Pi_i^{\mathcal{G}_{\boldsymbol{\phi}^*_{[k]}}}$ for all i and thus $\mathcal{G}_{\boldsymbol{\phi}_{[k]}} = \mathcal{G}_{\boldsymbol{\phi}^*_{[k]}}$ for all k. However, since $\boldsymbol{\phi} \neq \boldsymbol{\phi}^*$, there exists some k such that $\boldsymbol{\phi}_{[k]} \neq \boldsymbol{\phi}^*_{[k]}$. Therefore, there exists a k such that the common probability mass function $p(\mathbf{x}|\boldsymbol{\phi}_{[k]}) = p(\mathbf{x}|\boldsymbol{\phi}^*_{[k]})$, factorized according to a common structure $\mathcal{G}_{\boldsymbol{\phi}_{[k]}} = \mathcal{G}_{\boldsymbol{\phi}^*_{[k]}}$, can be parameterized by two different parameters $\boldsymbol{\phi}_{[k]}$ and $\boldsymbol{\phi}^*_{[k]}$. This is impossible, since according to assumption (A2) $[X_j|\Pi_j^{\mathcal{G}_{\boldsymbol{\phi}}}]$ is identifiable.

Case 2: $\mathcal{S}(\mathcal{G}_{\boldsymbol{\phi}}) \cap \mathcal{S}(\mathcal{G}_{\boldsymbol{\phi}^*}) = \varnothing$, that is, none of the orderings of $\mathcal{G}_{\boldsymbol{\phi}^*}$ is compatible with $\mathcal{G}_{\boldsymbol{\phi}}$. In this case, there must exist a pair of indices (i, j) such that in $\mathcal{G}_{\boldsymbol{\phi}^*}$ $X_i \in \text{an}(X_j)$, but in $\mathcal{G}_{\boldsymbol{\phi}}$ X_j is a non-descendant of X_i. Then X_j is independent of X_i in the DAG of $\boldsymbol{\phi}_{[i]}$, since in $\mathcal{G}_{\boldsymbol{\phi}_{[i]}}$ X_i has no parents and X_j is a non-descendant of X_i. However, in $\mathcal{G}_{\boldsymbol{\phi}^*_{[i]}}$ we still have $X_i \in \text{an}(X_j)$. Since $\boldsymbol{\phi}^*$ is natural, X_i and X_j are not independent in $\boldsymbol{\phi}^*_{[i]}$. Therefore, there exists $1 \leq i \leq p$ such that $p(\mathbf{x}|\boldsymbol{\phi}_{[i]}) \neq p(\mathbf{x}|\boldsymbol{\phi}^*_{[i]})$, which contradicts our assumption.

So in both *case 1* and *case 2* we have a contradiction. Thus, the first claim holds.

To prove the second claim (25), note that by the law of large numbers,

$$
\begin{aligned}
\frac{1}{n}(L(\boldsymbol{\phi}) - L(\boldsymbol{\phi}^*)) &= \sum_{k=1}^{p+1} \frac{n_k}{n}\frac{1}{n_k}\sum_{h\in\mathcal{I}_k} \log\frac{p(\mathcal{X}_{h\cdot}|\boldsymbol{\phi}_{[k]})}{p(\mathcal{X}_{h\cdot}|\boldsymbol{\phi}^*_{[k]})} \\
&= \sum_{k=1}^{p+1} \alpha_k \left\{ \mathbf{E}_{\boldsymbol{\phi}^*_{[k]}}\left[\log\frac{p(\mathbf{Y}|\boldsymbol{\phi}_{[k]})}{p(\mathbf{Y}|\boldsymbol{\phi}^*_{[k]})}\right] + \mathcal{O}(n_k^{-\frac{1}{2}})\right\}, \qquad (2.24)
\end{aligned}
$$

where $\alpha_n = \frac{n_k}{n}$ and $\mathbf{Y}$ is a random vector with probability mass function $p(\mathbf{x}|\boldsymbol{\phi}^*_{[k]})$. Using Jensen's inequality, $\mathbf{E}_{\boldsymbol{\phi}^*_{[k]}}\left[\log\frac{p(\mathbf{Y}|\boldsymbol{\phi}_{[k]})}{p(\mathbf{Y}|\boldsymbol{\phi}^*_{[k]})}\right] = c_k \leq 0$ for all $k = 1, \ldots, p+1$. Furthermore, by the above arguments for the first claim of identifiability, there exists $j \in \{1, \ldots, p\}$ such that $c_j < 0$ is a negative constant. In order to guarantee the right hand side of the equation (2.24) to be negative, it is sufficient to have $\min_{k=1,\ldots,p} \alpha_k \gg n^{-\frac{1}{2}}$. This completes the proof. $\square$

Proof of Theorem 6 . Let $\mathbf{Y}$ be a random vector with probability mass function $p(\mathbf{x}|\boldsymbol{\phi})$ and

$$
\mathbf{I}(\boldsymbol{\phi}) = \mathbf{E}_{\boldsymbol{\phi}}\left\{\left[\frac{\partial}{\partial\boldsymbol{\phi}}\log p(\mathbf{Y}|\boldsymbol{\phi})\right]\left[\frac{\partial}{\partial\boldsymbol{\phi}}\log p(\mathbf{Y}|\boldsymbol{\phi})\right]^T\right\}
$$

be the Fisher information matrix. We consider two cases.

Case 1: Consider $\boldsymbol{\phi} \in \text{nb}(\boldsymbol{\phi}^*)$. By Lemma 7, $\mathcal{G}_{\boldsymbol{\phi}}$ and $\mathcal{G}_{\boldsymbol{\phi}^*}$ have a common compatible ordering. If we restrict to the lower dimensional space $\Omega_{[k]} = \left\{\boldsymbol{\phi}_{[k]} : \boldsymbol{\phi} \in \Omega\right\}$, the same argument applies to an arbitrarily small neighborhood of $\boldsymbol{\phi}^*_{[k]}$ in this space, that is, $\mathcal{G}_{\boldsymbol{\phi}_{[k]}}$ and $\mathcal{G}_{\boldsymbol{\phi}^*_{[k]}}$ share a compatible ordering. Then it follows from the arguments used in *Case 1* in the proof of Theorem 1 that, $p(\mathbf{x}|\boldsymbol{\phi}_{[k]}) \neq p(\mathbf{x}|\boldsymbol{\phi}^*_{[k]})$ for $\boldsymbol{\phi}_{[k]} \in \text{nb}(\boldsymbol{\phi}^*_{[k]})\backslash\{\boldsymbol{\phi}^*_{[k]}\}$ and some non-negligible set of $\mathbf{x}$. Thus,

$$
\mathbf{E}_{\boldsymbol{\phi}^*_{[k]}}\left[\log p(\mathbf{Y}|\boldsymbol{\phi}^*_{[k]})\right] > \mathbf{E}_{\boldsymbol{\phi}^*_{[k]}}\left[\log p(\mathbf{Y}|\boldsymbol{\phi}_{[k]})\right], \quad k = 1, \ldots, p+1,
$$

which implies that $\mathbf{I}(\boldsymbol{\phi}^*_{[k]})$ is positive definite for all k.

Let $\mathbf{u} \in \{\mathbf{u} : \boldsymbol{\phi}^* + a_n\mathbf{u} \in \Omega\}$ and u_j be its j^{th} component group defined in the same way as ϕ_j. Without loss of generality, we can assume that $\|\mathbf{u}\|_2 = M$ where M is a fixed positive constant. Further, let $\mathbf{u}_{[k]}$ be the vector defined similarly as $\boldsymbol{\phi}_{[k]}$ by setting $u_j = \mathbf{0}$ if the j^{th} group corresponds to an edge pointing to X_k. Note that $\sum_{k=1}^{p+1}\|\mathbf{u}_{[k]}\|_2^2 \geq \|\mathbf{u}\|_2^2$. Let $\delta^k_{\min} > 0$

be the minimal eigenvalue of $\mathbf{I}(\boldsymbol{\phi}^*_{[k]})$ and $\rho = \min_{k=1,\ldots,p+1}(\alpha_k \delta^k_{\min}/2)$. Then

$$\sum_{k=1}^{p+1} \frac{\alpha_k}{2} \mathbf{u}_{[k]}^T \mathbf{I}(\boldsymbol{\phi}^*_{[k]}) \mathbf{u}_{[k]} \geq \sum_{k=1}^{p+1} \frac{\alpha_k}{2} \delta^k_{\min} \|\mathbf{u}_{[k]}\|_2^2 \geq \rho \sum_{k=1}^{p+1} \|\mathbf{u}_{[k]}\|_2^2 \geq \rho \|\mathbf{u}\|_2^2. \tag{2.25}$$

Write $a_n = \omega(b_n)$ if $b_n^{-1} = O(a_n^{-1})$. Let $\omega(1/\sqrt{n}) = a_n = o(1)$. Recall that $\mathcal{B} = \{j : \phi_j^* \neq \mathbf{0}, 1 \leq j \leq p^2\}$. We have

$$\begin{aligned}
& R(\boldsymbol{\phi}^* + a_n \mathbf{u}) - R(\boldsymbol{\phi}^*) \\
& \leq L(\boldsymbol{\phi}^* + a_n \mathbf{u}) - L(\boldsymbol{\phi}^*) - \lambda_n \sum_{j \in \mathcal{B}} (\|\phi_j^* + a_n u_j\|_2 - \|\phi_j^*\|_2) \\
& \leq \sum_{k=1}^{p+1} \left[L_k(\boldsymbol{\phi}^*_{[k]} + a_n \mathbf{u}_{[k]}) - L_k(\boldsymbol{\phi}^*_{[k]}) \right] + \lambda_n a_n \sum_{j \in \mathcal{B}} \|u_j\|_2,
\end{aligned} \tag{2.26}$$

where the last line follows from the triangle inequality. By assumption (A2), Taylor expansion of L_k around $\boldsymbol{\phi}^*_{[k]}$ in (2.26) leads to

$$\begin{aligned}
& \sum_{k=1}^{p+1} \left[a_n \{\nabla L_k(\boldsymbol{\phi}^*_{[k]})\}^T \mathbf{u}_{[k]} - \frac{1}{2} n_k a_n^2 \mathbf{u}_{[k]}^T \mathbf{I}(\boldsymbol{\phi}^*_{[k]}) \mathbf{u}_{[k]} \{1 + o_p(1)\} \right] + \lambda_n a_n \sum_{j \in \mathcal{B}} \|u_j\|_2 \\
\leq & \sum_{k=1}^{p+1} \left[\sqrt{\alpha_k} \sqrt{n} a_n \frac{\{\nabla L_k(\boldsymbol{\phi}^*_{[k]})\}^T}{\sqrt{n_k}} \mathbf{u}_{[k]} \{1 + o_p(1)\} \right] - \rho n a_n^2 \|\mathbf{u}\|_2^2 \{1 + o_p(1)\} \\
& + \lambda_n n^{-1/2} \sqrt{n} a_n \sum_{j \in \mathcal{B}} \|u_j\|_2,
\end{aligned}$$

where we have used (2.25) and that $n_k/n = \alpha_k$. Now dividing both sides by $\sqrt{n} a_n$, we arrive at

$$\begin{aligned}
& (\sqrt{n} a_n)^{-1} \left[R(\boldsymbol{\phi}^* + a_n \mathbf{u}) - R(\boldsymbol{\phi}^*) \right] \\
\leq & \sum_{k=1}^{p+1} \left[\sqrt{\alpha_k} \frac{\{\nabla L_k(\boldsymbol{\phi}^*_{[k]})\}^T}{\sqrt{n_k}} \mathbf{u}_{[k]} \{1 + o_p(1)\} \right] - \rho \sqrt{n} a_n \|\mathbf{u}\|_2^2 \{1 + o_p(1)\} \\
& + \lambda_n n^{-1/2} \sum_{j \in \mathcal{B}} \|u_j\|_2.
\end{aligned} \tag{2.27}$$

Note that $n_k^{-1/2} \|\nabla L_k(\boldsymbol{\phi}^*_{[k]})\|_2 = O_p(1)$ for all k by the central limit theorem, and $\lambda_n/\sqrt{n} = o(1)$. Therefore, the second order term in (2.27) dominates the first and the third terms uniformly if $\sqrt{n} a_n$ is sufficiently large. Hence, for any $\varepsilon > 0$, there exists a constant $C < \infty$

such that

$$P\left[\sup_{a_n\|\mathbf{u}\|_2\geq C/\sqrt{n}} R(\boldsymbol{\phi}^* + a_n\mathbf{u}) < R(\boldsymbol{\phi}^*)\right] \geq 1-\varepsilon, \tag{2.28}$$

when n is sufficiently large.

Case 2: Consider $\boldsymbol{\phi} \in \Omega\backslash \mathrm{nb}(\boldsymbol{\phi}^*)$. Let $\sqsubset_i$ $(1 \leq i \leq p!)$ be an ordering of the p variables. Since $R(\boldsymbol{\phi})$ is a strictly concave function in the subspace $\Omega_i = \{\boldsymbol{\phi} \in \Omega : \sqsubset_i$ is compatible with $\mathcal{G}_{\boldsymbol{\phi}}\}$ by assumption (A2), there exists a unique local maximizer $\hat{\boldsymbol{\phi}}^i$ of $R(\boldsymbol{\phi})$ in Ω_i. The pointwise convergence in probability of the random concave function $n^{-1}R(\boldsymbol{\phi})$ implies that $\hat{\boldsymbol{\phi}}^i \to_p \boldsymbol{\phi}^i$, where $\boldsymbol{\phi}^i \in \Omega_i$ is the unique maximizer of its limiting function; see Andersen and Gill (1982) and Pollard (1991). Let $\mathcal{M} \triangleq \{\boldsymbol{\phi}^i : \boldsymbol{\phi}^i \neq \boldsymbol{\phi}^*\}$ which is a finite set. For any $\boldsymbol{\phi}^i \in \mathcal{M}$ we have

$$\begin{aligned}\frac{1}{n}R(\boldsymbol{\phi}^i) - \frac{1}{n}R(\boldsymbol{\phi}^*) &\leq \frac{1}{n}L(\boldsymbol{\phi}^i) - \frac{1}{n}L(\boldsymbol{\phi}^*) - \frac{1}{n}\lambda_n \sum_{j\in\mathcal{B}}(\|\phi^i_j\|_2 - \|\phi^*_j\|_2)\\ &\to_p \sum_{k=1}^{p+1}\alpha_k \mathbf{E}_{\boldsymbol{\phi}^*_{[k]}}\left[\log\frac{p(\mathbf{Y}|\boldsymbol{\phi}^i_{[k]})}{p(\mathbf{Y}|\boldsymbol{\phi}^*_{[k]})}\right] - \frac{1}{n}\lambda_n O(1), \end{aligned}\tag{2.29}$$

by (2.24). The second term is $o(n^{-\frac{1}{2}})$ since $\lambda_n = o(n^{\frac{1}{2}})$, while the first term is negative and on the order of $\alpha_j \gg n^{-\frac{1}{2}}$ for some $j \in \{1,\ldots,p\}$, as shown in (2.24). Thus, the last line will be negative when n is large. Since $|\mathcal{M}|$ is finite, this implies that for any $\varepsilon > 0$,

$$P\left[\sup_{\mathcal{M}} R(\boldsymbol{\phi}^i) < R(\boldsymbol{\phi}^*)\right] \geq 1-\varepsilon, \tag{2.30}$$

when n is sufficiently large.

Combining (2.28) and (2.30), we have shown that there is a global maximizer $\hat{\boldsymbol{\phi}}$ of $R(\boldsymbol{\phi})$ such that $\|\hat{\boldsymbol{\phi}} - \boldsymbol{\phi}^*\|_2 = O_p(n^{-\frac{1}{2}})$. □

Theorem 5 confirms that the causal DAG model is identifiable with interventional data assuming a natural parameter. Theorem 6 implies that there is a $\sqrt{n}$-consistent global maximizer of $R(\boldsymbol{\phi})$ with the group norm penalty. Note that Assumption (A2) does not specify a particular choice of model for the conditional distribution $[X_j|\Pi^{\mathcal{G}}_j]$ and thus these theoretical results apply to a large class of DAG models for discrete data. In particular, the multi-logit regression model (2.2) satisfies (A2).

Remark 2. The assumption on the sample size of interventional data, $n_k \gg \sqrt{n}$, imposes a lower bound on how fast the fraction $\alpha_k = n_k/n \gg n^{-1/2}$ can approach zero for $k = 1, \ldots, p$. Although this allows the observational data to dominate when $\alpha_k \to 0$, the fractions of interventional data must be larger than the typical order $O_p(n^{-1/2})$ of statistical errors so that (2.23) can hold to establish identifiability of the true causal DAG parameter $\boldsymbol{\phi}^*$. This guarantees that the global maximizer $\hat{\boldsymbol{\phi}}$ will locate in a neighborhood of $\boldsymbol{\phi}^*$ with high probability. Once in this neighborhood, the convergence rate of $\hat{\boldsymbol{\phi}}$ then depends on the size n of all data, both interventional and observational. Therefore, increasing the size of observation data will lead to more accurate estimate $\hat{\boldsymbol{\phi}}$ as long as we keep $\alpha_k \gg n^{-1/2}$ for $k = 1, \ldots, p$.

Remark 3. It is interesting to generalize the above asymptotic results to the case where $p = p_n$ grows with the sample size n, say, by developing nonasymptotic bounds on the ℓ_2 estimation error $\|\hat{\boldsymbol{\phi}} - \boldsymbol{\phi}^*\|_2$. However, in order to estimate the causal network consistently, sufficient interventional data are needed for each node, i.e., n_k must approach infinity, and thus $p/n \to 0$ as $n \to \infty$. This limits us to the low-dimensional setting with $p < n$. Suppose we have a large network with $p \gg n$. One may first apply some regularization method on observational data to screen out independent nodes and to partition the network into small subgraphs that are disconnected to one another. Then for each small subgraph, we can afford to generate enough interventional data for every node and apply the method in this paper to infer the causal structure. Our asymptotic theory provides useful guidance for the analysis in the second step.

For purely observational data, the theory becomes more complicated due to the existence of equivalent DAGs and parameterizations. It is left as future work to establish the consistency of a global maximizer for high-dimensional observational data. See Aragam and Zhou (2015) for related analysis on Gaussian Bayesian networks.

2.4 Simulation Studies

We evaluate the CD algorithm on simulated data sets. As stated in Remark 1, the log-likelihood (2.7) applies to observational data as well. Therefore, we apply the CD algorithm on both interventional data and observational data. In order to assess the accuracy and efficiency of the CD algorithm, we compare it with a few competing methods. For interventional data, we compare our CD algorithm with the PC algorithm (Kalisch and Bühlmann, 2007), the greedy interventional equivalent search (GIES) algorithm (Hauser and Bühlmann, 2015) and the equi-energy sampler (EE sampler) (Kou et al, 2006). For observational data we compare the CD algorithm with the hill-climbing (HC) algorithm (Gámez et al, 2011), the max-min hill-climbing (MMHC) algorithm (Tsamardinos et al, 2006), and the PC algorithm. Among these competitors, the PC algorithm is a constraint-based method, the MMHC is a hybrid method and the others are all score-based.

Details about data generation and parameter choices will be discussed in Section 2.4.1. In Section 2.4.2, we compare DAGs estimated from interventional data. Section 2.4.3, on the other hand, presents results on high-dimensional observational data. The comparison of running times is provided in Section 2.4.4.

2.4.1 Experimental setup

Four types of networks are used to compare the methods: the bipartite graph, the scale-free network, the small-world network, and random DAGs. In each setting, we consider the combination of three main parameters: (n, p, s_0), where n is the sample size, p is the number of nodes, and s_0 is the number of true edges.

We generated bipartite graphs, scale-free networks, and small-world networks with the R package **igraph** (Csárdi and Nepusz, 2006). The bipartite graphs were generated by the Erdős-Rényi model (Renyi and Erdos, 1959). Each bipartite graph in our datasets had $0.2p$ top nodes, $0.8p$ bottom nodes, and $s_0 = p$ directed edges from the top to the bottom nodes. The structure of a scale-free network was generated using the Barabási-Albert model

(Barabási and Albert, 1999). These networks had $s_0 = p-1$ directed edges. The small-world networks were generated by the Watts-Strogatz model (Watts and Strogatz, 1998). A graph initially generated by the model was undirected. To convert it to a DAG, edge directions were chosen according to a randomly generated topological sort. In this way, a small-world network had $s_0 = 2p$ directed edges. Random DAGs were sampled using the R package **pcalg** (Kalisch et al, 2012), and each DAG had $s_0 \approx p$ edges.

In all the simulation studies, each variable was assumed to be binary, i.e., $r_j = 2$ for all j. In this case, each group of parameters $\boldsymbol{\beta}_{j\cdot i} = (\beta_{j1i}, \beta_{j2i}) \in \mathbb{R}^2$. If $\Pi_j = \varnothing$, X_j would be sampled from its two levels with equal probability. Otherwise, the parameters $\boldsymbol{\beta}_{j\cdot 0}$ and $\boldsymbol{\beta}_{j\cdot i}$, $i \in \Pi_j$, were chosen such that

$$p_{j\ell}(\mathbf{x}_h) = \frac{\exp(2\sum_{i\in\Pi_j} y_{hi\ell})}{\exp(2\sum_{i\in\Pi_j} y_{hi1}) + \exp(2\sum_{i\in\Pi_j} y_{hi2})}$$

for $\ell = 1, 2$, where $y_{hi\ell} = I(\mathcal{X}_{hi} = \ell)$. The value of a variable under intervention was randomly fixed to one of its levels regardless of its parents.

For each dataset, we input to our CD algorithm a sequence of 40 values of λ, starting from λ_1 (2.19) and ending at $0.01\lambda_1$. Since we assume the graphs are sparse, we stop a solution path when the number of predicted edges exceeds $3p$. Consequently, a sequence of DAGs is estimated and one of them will be picked by our model selection criterion (2.20) with $\alpha = 0.3$. To avoid any potential bias in the estimation, we pick a random order to cycle through all the blocks in the outer loop of Algorithm 1.

The EE sampler is used for comparisons on interventional data. Its implementation for DAG estimation was done as in Zhou (2011). We will call it the EE-DAG sampler hereafter. In each run, we simulate 10 chains with a 0.1 chance of equi-energy jumps. To obtain an estimated DAG, we threshold the average graph from the target chain (the 10th chain). More precisely, an edge will be predicted if its posterior inclusion probability is greater than 0.5.

The HC algorithm is a standard greedy method, while the MMHC algorithm is a hybrid method. For these two algorithms, we use the R package **bnlearn** (Scutari, 2010, 2017). These methods are designed specifically for observational data and thus are compared with our method only on observational data. For both the HC and the MMHC algorithm, we have the option to limit the number of parents per node, but since this algorithm can estimate a reasonable number of edges, we did not set an upper limit. It is worth noticing that, in our simulation results the HC algorithm predicts much more edges than the number of true edges most of the time. It is an indication that the regular BIC might not serve as a good model selection criterion in high-dimensional case.

The PC algorithm is a popular constraint-based algorithm for learning Bayesian networks, with an efficient implementation in the R package pcalg (Kalisch et al, 2012; Hauser and Bühlmann, 2012). However, it may not produce a DAG for every data set, and instead its output is a completed partially directed acyclic graph (CPDAG), which contains both directed and undirected edges. To make a fair comparison, we distinguish undirected edges from directed ones in our calculation of various performance metrics, with details provided later. The tuning parameter for the PC algorithm is the significance level for conditional independence tests, which is chosen as $\alpha = 0.01$ for all data.

The GIES algorithm is an algorithm designed specifically to learn Bayesian networks from a mixture of observational and interventional data, by searching over the so-called interventional equivalence classes. Like for the PC algorithm, this method also might not produce a DAG but a partial directed graph (a graph in the interventional Markov equivalent family). Note that given a pure observational data set, the interventional Markov equivalent classes decrease to Markov equivalent class, and the algorithm should be producing a CPDAG just like the PC algorithm. Only with enough sets of experimental intervention, the GIES algorithm is able to produce a DAG, we have discussed details for the condition of "enough experiments" in Chapter 1. Under our experimental setting in Section 2.4.2, the GIES algorithm should be able to recover the DAG structure. Therefore we compare our CD algorithm with GIES algorithm for interventional data. However, the implementation of this algorithm in the **pcalg** package, the only implementation we found, can only take continuous data as input. So we generated continuous data from the simulated discrete data. We convert our discrete data to continuous data using the following procedure: for a discrete variable X_j with r_j levels at the hth data point, the node takes level $\mathcal{X}_{hj}$. We generated continuous data $\widetilde{\mathcal{X}}_{hj}$ by sampling from $Unif[\mathcal{X}_{hj}, \mathcal{X}_{hj} + 1]$, $\mathcal{X}_{hj} \in \{0, 1, ..., r_j - 1\}$.

2.4.2 Results for interventional data

For each type of network, we generated graphs with $p = 50$ and $p = 100$. For each node X_j, we generated n_j data points where the node is under intervention, so that the sample size $n = \sum_{j=1}^{p} n_j$ for interventional data. We chose $n_j \in \{1, 5\}$ for all $j = 1, \ldots, p$ to test the performance of the algorithms given different amount of intervention. In particular, when $n_j = 1$ we have $n = p$, which lies on the boundary between low- and high-dimensional settings. In combination, our choices of the data size were $(n, p) \in \{(50, 50), (250, 50), (100, 100), (500, 100)\}$. For each combination of (n, p), we generated 20 data sets.

We compare the DAGs estimated by four algorithms, the CD algorithm, the EE-DAG sampler, the PC algorithm, and the GIES algorithm. For an estimated DAG, we distinguish between expected edges, which are estimated edges in the true skeleton with the correct direction, and reversed edges, which are in the true skeleton but with a reversed direction. Let P, E, R, and FP denote, respectively, the numbers of predicted edges, expected edges, reversed edges, and false positive edges (excluding the reversed ones) in an estimated DAG, and recall that s_0 is the number of edges in the true graph. Then the number of missing edges is $\mathrm{M} = s_0 - \mathrm{E} - \mathrm{R}$. The accuracy of DAG estimation is measured by the true positive rate (TPR), the false discovery rate (FDR), the structural Hamming distance (SHD) (Tsamardinos et al, 2006), and the Jaccard index (JI), defined as $\mathrm{TPR} = \mathrm{E}/s_0$, $\mathrm{FDR} = (\mathrm{R} + \mathrm{FP})/\mathrm{P}$, $\mathrm{SHD} = (\mathrm{R} + \mathrm{M} + \mathrm{FP})$, and $\mathrm{JI} = \mathrm{E}/(\mathrm{P} + s_0 - \mathrm{E})$. Note that SHD was originally defined for CPDAGs, and our definition here measures the Hamming distance between two DAGs. Both SHD and JI are single performance metrics for DAG estimation. We mark in boldface results with the optimum SHD and JI scores in the subsequent tables.

Results reported in Table 2.1 are the comparisons between our CD algorithm and the PC algorithm, while in Table 2.2 are the comparisons between our CD algorithm and the EE-DAG sampler. Results are averages over 20 data sets for each setting (n, p, s_0). For our CD algorithm, we report two results for each setting, (i) result with the smallest SHD along the solution path; (ii) result using our model selection criterion (2.20) with $\alpha = 0.3$. In Table 2.1, in order to make a clear comparison, we report lower and upper bounds of the SHD and the Jaccard index for CPDAGs estimated by the PC algorithm. Counting all undirected edges in a CPDAG that are in the true skeleton as expected edges, we will have a lower bound for the SHD and an upper bound for the Jaccard index. Counting these undirected edges as reversed edges will give us an upper bound for the SHD and a lower bound for the Jaccard index.

It is obvious from Table 2.1 that in majority of the cases our CD algorithm outperformed the PC algorithm. The SHD score is smaller than the lower bound of the PC algorithm, and the Jaccard index is higher than the upper bound of the PC algorithm. Only for random DAGs and small-world networks with $n = 100$, the SHD of our CD algorithm is slightly higher than the lower bound of the SHD while the Jaccard index is slightly lower than the upper bound of the PC algorithm, showing that our algorithm was close to the best performance one could hope for the PC algorithm. Note that when calculating the TPRs and FDRs in the table, the undirected edges are counted as expected ones, which clearly favors the PC algorithm.

We were only able to test the EE-DAG sampler on small graphs with $p = 50$ because its computing time was too long for graphs with $p = 100$. For the cases of $n = p = 50$ (Table 2.2), we see that the smallest SHD our CD algorithm can achieve along a solution path is 20% lower than the EE-DAG sampler for bipartite graphs, random DAGs and scale-free networks, and 10% lower for small-world networks. The EE-DAG sampler predicted much more false positive edges (FP) but slightly fewer reversed edges (R) than our CD algorithm. For the cases of $n = 5p$ (Table 2.3), the EE-DAG sampler had an outstanding performance, with a lower SHD and a higher Jaccard index for all cases. These observations are largely in agreement with our expectation. The EE-DAG sampler searches for DAGs by sampling from a posterior distribution under the product multinomial model with a conjugate Dirichlet prior, which is close to ℓ_0 regularization when n is large. Thus, it is expected to have good performance when p is small and n is large. However, the search is combinatorial in nature, which makes it impractical for even moderately large networks (such as the graphs with $p = 100$ here). On the contrary, our CD algorithm showed no problem in estimating graphs with hundreds of nodes and can obtain comparable or better results in the cases of

$n = p = 50$.

We also did a comparison between the CD algorithm and the GIES algorithm, reported in Table 2.4. The results seem to suggest that the GIES algorithm often selects too many edges. When $n = p = 100$, the number of edges the GIES algorithm predicted was around $3s_0$ in most of the cases. Consequently, it showed a much higher FDR as well as a larger SHD. Recall that the available package for the GIES algorithm can only take continuous data, which were generated by taking a transformation of the simulated discrete data. As a result, this comparison could be confounded by the use of different although related data sets, and thus is only intended to illustrate how the two algorithms would work.

In order to show how interventional data improve the accuracy, we did more experiments. Figure 2.2 shows how the SHD decreases when adding intervention to an observational data set, for all four types of graphs with $n = 500$ and $p = 50$. We started with a purely observational data set, and replaced m observational data points by m interventional data points for each node, for $m = 1, 2, ..., 10$. The sample size was fixed as $n = 500$. Therefore, we would finally have a data set with 10 interventions for each node. Figure 2.2 shows the average of 20 experiments, with a very clear downward trend in all plots. The curve for the bipartite graph is not as smooth as the curves for the other types of networks. This is because the improvement for bipartite graphs is not as significant as the other networks.

2.4.3 Results for high-dimensional observational data

In this section, we apply our CD algorithm to high-dimensional observational data and compare its performance with the PC algorithm, the MMHC algorithm, and the HC algorithm.

Metrics for estimation accuracy in this section are modified from those for interventional data. Since equivalent DAGs cannot be distinguished with observational data, we define reversed edges with regard to CPDAGs. A CPDAG is a partially directed graph that has all compelled (directed) edges in the equivalent class of a DAG. We calculate CPDAGs for both an estimated DAG and the true DAG. A reversed edge (R) refers to a predicted edge that satisfies the following two conditions: i) Its direction in the estimated DAG is wrong

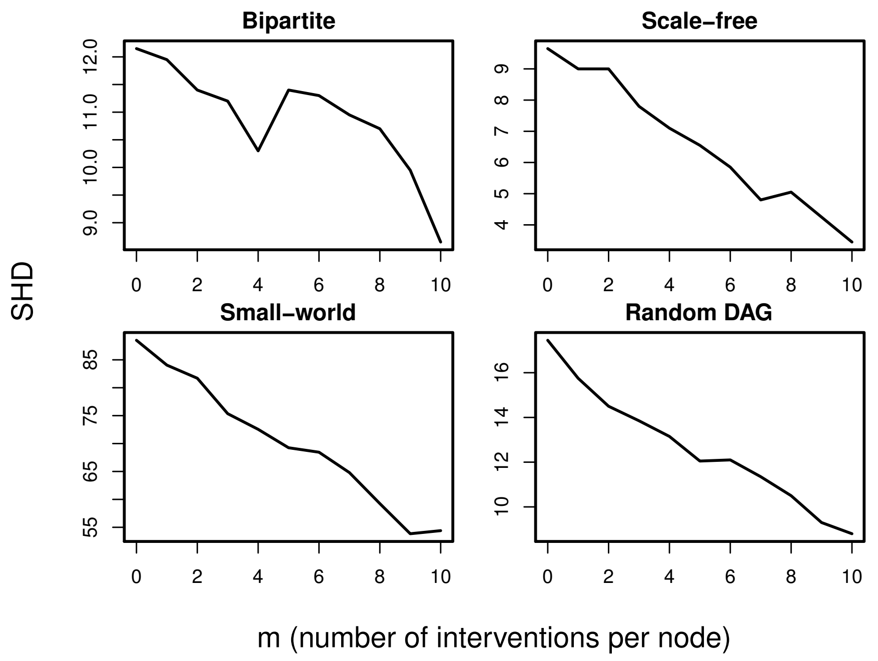


Figure 2.2: The effect of interventions in terms of the SHD, where each node has m interventional data points while the total sample size n is fixed

compared to the true DAG. ii) The direction of this edge is inconsistent between the CPDAGs of the true and estimated DAGs, including the case where the edge is directed in one CPDAG but undirected in the other. Likewise, we define reversed edges in a CPDAG predicted by the PC algorithm as edges in the true skeleton that have an inconsistent direction with the true CPDAG. The number of expected edges (E) is the number of estimated edges in the true skeleton excluding those reversed ones.

For high-dimensional data, we generated graphs with $p = 200$ for each type of the networks. We chose $n = 50$ and the number of true edges s_0 ranged from 190 to 400 for these graphs. Again, 20 data sets were generated for each combination of (n, p).

Table 2.5 summarizes the comparison results. One sees that our model selection criterion works quite well: The SHDs of CD and CD* in Table 2.5 are very close. Our CD or CD* algorithm has the highest Jaccard index for all networks, and the SHD is also quite low compared to other algorithms for most of the networks. Only for the case of small-world networks, the lowest SHD our CD algorithm can achieve is slightly higher than the MMHC algorithm. We can see that when running the HC algorithm with the default setting, it tends to predict too many edges that for the more sparse networks like bipartite graphs, scale-free networks and random DAGs, the number of edges it predicted is almost twice the number of true edges. This makes the HC algorithm had a much higher TPR as well as a higher FDR and larger SHD than all the other algorithms. The PC algorithm predicted too few edges in scale-free networks and small-world networks, which led to a substantially lower TPRs and JIs, and the FDRs were quite high for bipartite networks. The MMHC algorithm, on the other hand, predicted a comparable number of edges as our CD algorithm in most cases. However its performance was much worse than our CD algorithm except for the small-world case. Our CD algorithm presents a clear advantage over all other algorithms in these high-dimensional cases.

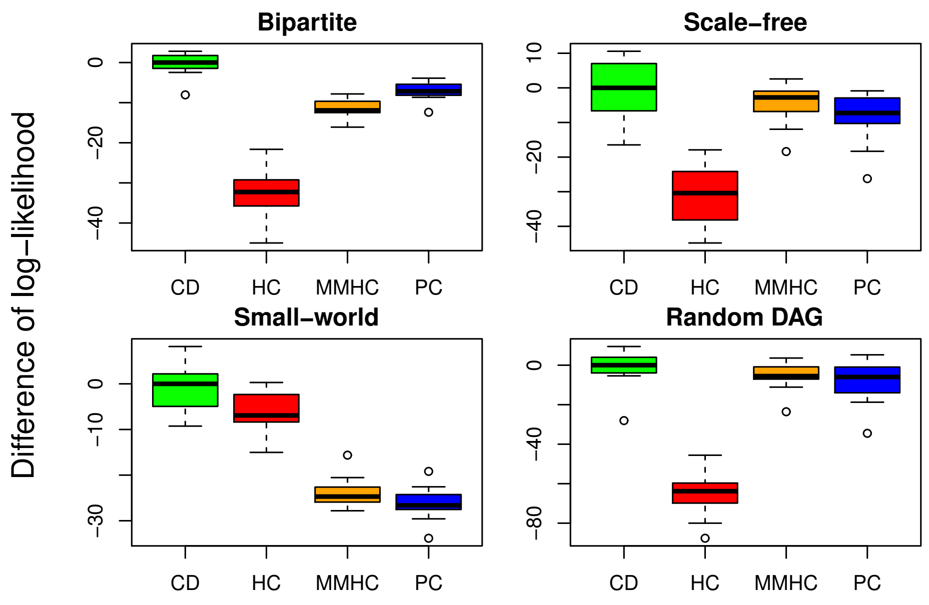


Figure 2.3: Box-plot of test data log-likelihood for four algorithms with log-likelihood scaled by the sample size $n = 50$

To further evaluate the quality of estimated networks, we computed test data log-likelihood to compare the predictive power. We generated 500 test data sets of the same size ($n = 50$) for each DAG with $p = 200$. We used each estimated graph to calculate the total log-likelihood of a test data set. Note that the output graph of the PC algorithm is a CPDAG, for which we cannot directly calculate the test data log-likelihood. However, since the likelihood for any data set under every DAG in an equivalence class is the same, we converted a CPDAG output by the PC algorithm to an arbitrary DAG in the equivalence class and then calculated its test data log-likelihood. Figure 2.3 is the box-plot of test data log-likelihood for the four types of graphs in terms of the difference from the median of the test data log-likelihood of the CD algorithm. DAGs estimated by the CD algorithm were chosen by our model selection criterion.

It is seen from Figure 2.3 that our CD algorithm has the highest test data log-likelihood for all types of networks. Note that even though for small-world networks case where our CD algorithm do not have the best SHD, it has the best predictive power. We can see that for small-world networks, our CD algorithm and the HC algorithm have comparable SHD and Jaccard index. Both Jaccard indexes is much higher than the MMHC algorithm and the PC algorithm and both algorithm has a much higher skeleton true positive rate, and that is why they have higher log-likelihood for small-world networks. Figure 2.3 shows that our method also has a very good predictive power in high-dimensional cases. We see that the HC algorithm has a much lower test data log-likelihood for most cases, which suggests overfitting given the observation that this algorithm often predicts too many edges. These results demonstrate the critical role of sparsity not only in structure estimation but also in predictive modeling.

2.4.4 Timing comparison

We comment briefly on the comparison of running time among the algorithms. The running time for generating a whole solution path for the interventional data (results in Tables 2.1 and 2.2) was within 40 seconds for all graphs. The PC algorithm was about two times faster than our CD algorithm for bipartite graphs and random DAGs, but it did not scale well for scale-free networks and small-world networks. For these two types of networks, running time of the PC algorithm was much longer for $n = 500$ and $p = 100$. For the high-dimensional data in Table 2.5, it took between 2 and 20 seconds for our method to compute the entire solution path. The speed of the PC algorithm was quite comparable to our CD algorithm on these data sets. The MMHC algorithm was faster which took at most 5 seconds for all data sets. The fastest algorithm was the HC algorithm, however, its accuracy in learning Bayesian networks was too bad so we will not go into details for this algorithm. Our method gives a principled way to incorporate interventional data and is often more accurate than the other competitors. These merits in performance justify its utility. In addition, there is room for a more efficient implementation of our algorithm which may improve its speed substantially.

2.5 Applications to Real Networks

In this section, we apply our CD algorithm to real networks. In Section 2.5.1, we examine how the proposed multi-logit model compares to the product multinomial model by comparing our CD algorithm to the K2 algorithm (Cooper and Herskovits, 1992). We will then apply our method to a real data set in Section 2.5.2.

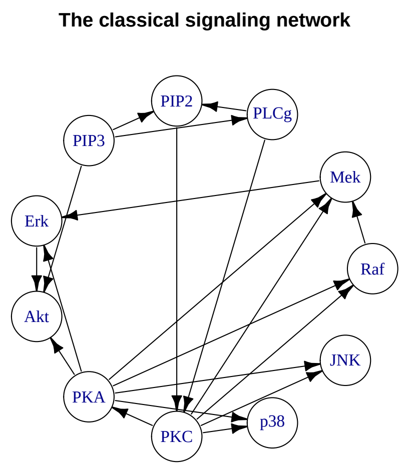
2.5.1 Comparison with the K2 Algorithm

The K2 algorithm is a well-known method for learning discrete Bayesian networks based on a product multinomial model. However, it requires an input ordering of the nodes. A wrong ordering can severely damage the quality of the estimated graph. Therefore, we provide the K2 algorithm with an ordering that is compatible with the true DAG to obtain the best estimation. In order to conduct a fair comparison, we also input the same ordering to our CD algorithm by only running the inner loop of Algorithm 2, which is equivalent to a sequence of $p-1$ penalized multi-logit regression problems. With a known ordering, a main difference between the two algorithms is the underlying model, the multi-logit model for our CD algorithm and the multinomial model for the K2 algorithm. We used a Matlab package **K2** (Bielza et al, 2011) to run the algorithm. The K2 algorithm also requires an upper bound for the maximum number of parents for each node. In our experiments, we set the upper bound to be 4. We chose 8 real networks provided by the **bnlearn** package, where p ranges from 8 to 441. Observational data were simulated from these networks, and for each DAG, 20 data sets were generated independently according to a product multinomial model. This comparison will demonstrate how well our proposed multi-logit model approximates the multinomial model.

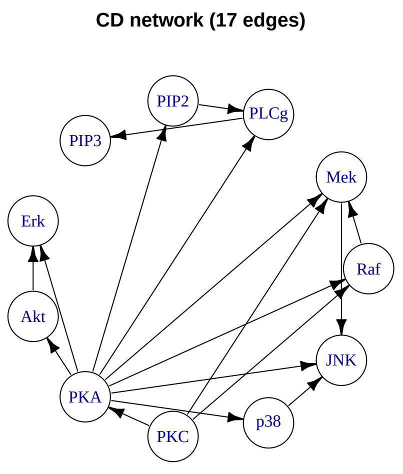
Summary of the comparison for the 8 networks is provided in Table 2.6. Since a correct ordering is given, there will not be any reversed edges, and thus, in this table, we only report P, TPR, FDR, SHD, and JI. Here we matched the number of predicted edges of our CD algorithm with the K2 algorithm. It can be seen that for most graphs the SHD for our CD algorithm is lower than that of the K2 algorithm, while the JI is higher, except the networks asia and hailfinder. Since the data sets were simulated by product multinomial models, this result confirms that our proposed multi-logit model serves as a good approximation to the full multinomial model. This comparison also suggests that the group norm regularization in our method may be more efficient than using an upper bound on the parent size as in the K2 algorithm.

2.5.2 Application to flow cytometry data

We consider in this section applying the CD algorithm to a real data set that has been extensively studied. The data set was generated from a flow cytometry experiment conducted by Sachs et al (2005), who studied a well-known signaling network in human primary CD4+ T-cells of the immune system. This chosen network was perturbed by various stimulatory and inhibitory interventions. Each interventional condition was applied to an individual component of the network. Simultaneous measurements were taken on $p = 11$ proteins and phospholipids of this network from individual cells under each condition. Since three interventions were targeted at proteins that were not measured, samples collected under these conditions were observational. Among the 11 measured components, five proteins and phospholipids were perturbed. The data set contains measurements for $n = 5,400$ cells. Each variable has three levels (high, medium and low), and consequently, the size of a component group of $\boldsymbol{\beta}$ is 6 for this data set.



(A)



(B)

Figure 2.4: (A) The consensus signaling network in human immune system cells, (B) DAG estimated by the CD algorithm

Figure 2.4A is a plot for the known causal interactions among the 11 components of this signaling network. These causal relationships are well-established, and no consensus has been reached on interactions beyond those present in the network. This network structure is often used as the benchmark to assess the accuracy of an estimated network. Therefore, we call it the consensus model. Our estimated network by the CD algorithm with the smallest SHD along the solution path is shown in Figure 2.4B. The DAG is qualitatively close to the consensus model. More detailed performance measures are reported in Table 2.7, including both results for the DAG with the smallest SHD (CD algorithm) and the one selected by our model selection criterion (CD algorithm*). As a comparison, we include the DAGs estimated by three competing methods, the order-graph sampler (Ellis and Wong, 2008), the EE-DAG sampler, and the PC algorithm. Our CD algorithm showed a very competitive performance, predicting more or comparable number of expected edges and fewer reversed edges than the other three methods. Our method also had the least number of false positive edges among all the methods. All these led to the lowest SHD and highest Jaccard index for our CD algorithm. Note that for the PC algorithm, we counted all 3 undirected edges in the true skeleton as expected edges in the calculation of the SHD and JI. Yet our CD algorithm still outperformed it. Markov chain Monte Carlo (MCMC) methods for DAG estimation often have good performance when the number of nodes p is small, but they do not scale well. Thus, it is comforting to see that our method, which can handle larger networks, outperforms MCMC methods on this relatively small network.

In addition, we have also compared our CD algorithm and the K2 algorithm on this data set. Given a true ordering, our CD algorithm predicted exactly the same number of expected edges with 1 less false positive edge than the K2 algorithm, as reported in Table 2.7. This implies that the proposed multi-logit model fits this data set as well as the product multinomial model.

2.6 Discussions

We have developed a maximum penalized likelihood method for estimating sparse discrete Bayesian networks under a multi-logit model. In order to avoid penalizing separately individual dummy variables for a factor, a group norm penalty is utilized to encourage sparsity at the factor level. A blockwise coordinate descent algorithm is developed where each coordinate descent step is solved by iteratively applying a quadratic approximation. The acyclicity constraint imposed on the structure of Bayesian networks can be enforced in a natural way by the coordinate descent algorithm. Our method has been evaluated on simulated graphs and real-world networks, with both interventional and observational data. We have demonstrated that the CD algorithm outperforms many existing methods, particularly when $n \leq p$. We have also performed an analysis of a flow cytometry data set generated from a signaling network in human immune system cells. The DAG estimated by the CD algorithm is close to the consensus model. Since the true network is not available, the estimated edges provide candidate causal interactions that could be tested in future experiments.

Computation for estimating discrete Bayesian networks is demanding due to the size of the parameter space and the nonlinear nature of the multi-logit model. There is room for improving the efficiency of the CD algorithm. For example, one may incorporate the idea of stochastic gradient descent in the quadratic approximation step, which will reduce significantly the computation. Moreover, since our search space is non-convex, introducing such components of stochastic optimization may also increase the chance of finding a global minimizer of the penalized loss function. Other future directions include studying the consistency of our penalized estimator when the number of nodes $p = p_n$ grows with the sample size n and investigating the use of group concave penalties.

Algorithm 1 CD algorithm for estimating discrete Bayesian networks

```
1: Initialize $\boldsymbol{\beta}$ such that $\mathcal{G}_{\boldsymbol{\beta}}$ is acyclic
2: for $i = 1, \ldots, p-1$ do
3:     for $j = i+1, \ldots, p$ do
4:         if $\boldsymbol{\beta}_{j\cdot i} \Leftarrow \mathbf{0}$ then
5:             $\boldsymbol{\beta}_{i\cdot j} \leftarrow \arg\min_{\boldsymbol{\beta}_{i\cdot j}} f_{\lambda,i}(\cdot), \quad \boldsymbol{\beta}_{j\cdot i} \leftarrow \mathbf{0}$
6:         else if $\boldsymbol{\beta}_{i\cdot j} \Leftarrow \mathbf{0}$ then
7:             $\boldsymbol{\beta}_{i\cdot j} \leftarrow \mathbf{0}, \quad \boldsymbol{\beta}_{j\cdot i} \leftarrow \arg\min_{\boldsymbol{\beta}_{j\cdot i}} f_{\lambda,j}(\cdot)$
8:         else
9:             $S_1 \leftarrow \min_{\boldsymbol{\beta}_{i\cdot j}} f_{\lambda,i}(\cdot) + f_{\lambda,j}(\cdot)|_{\boldsymbol{\beta}_{j\cdot i}=\mathbf{0}}$
10:            $S_2 \leftarrow f_{\lambda,i}(\cdot)|_{\boldsymbol{\beta}_{i\cdot j}=\mathbf{0}} + \min_{\boldsymbol{\beta}_{j\cdot i}} f_{\lambda,j}(\cdot)$
11:            if $S_1 \leq S_2$ then
12:                $\boldsymbol{\beta}_{i\cdot j} \leftarrow \arg\min_{\boldsymbol{\beta}_{i\cdot j}} f_{\lambda,i}(\cdot), \quad \boldsymbol{\beta}_{j\cdot i} \leftarrow \mathbf{0}$
13:            else
14:                $\boldsymbol{\beta}_{i\cdot j} \leftarrow \mathbf{0}, \quad \boldsymbol{\beta}_{j\cdot i} \leftarrow \arg\min_{\boldsymbol{\beta}_{j\cdot i}} f_{\lambda,j}(\cdot)$
15:            end if
16:        end if
17:    end for
18: end for
19: Update intercepts $\boldsymbol{\beta}_{j\cdot 0}$ for $j = 1, \ldots, p$
20: Inner loop given the active edge set (Algorithm 2)
21: Repeat step 2 to 20 until some stopping criterion is met
```

Algorithm 2 Inner loop

```
1: Input $E^{(t)}$ and initialize $\boldsymbol{\beta} = \boldsymbol{\beta}^{(t)}$
2: for $(i, j) \in E^{(t)}$ do
3:     $\boldsymbol{\beta}_{j\cdot i} \leftarrow \arg\min_{\boldsymbol{\beta}_{j\cdot i}} f_{\lambda,j}(\cdot)$
4: end for
5: Repeat step 2 to step 4 until convergence.
```

Table 2.1: Comparison between our CD algorithm and the PC algorithm on simulated interventional data

Graph	(n, p, s_0)	Method	P	E	R	FP	TPR	FDR	SHD	JI
Bipartite	$(100, 100, 100)$	CD	98.2	63.0	18.6	16.5	0.630	0.355	**53.5**	**0.466**
		CD*	60.5	41.2	14.5	4.7	0.412	0.316	63.5	0.345
		PC	50.0	5.6(18.3)	21.9	4.2	0.239	0.085	(80.3, 98.7)	(0.039, 0.191)
	$(500, 100, 100)$	CD	104.2	81.7	17.1	5.5	0.816	0.217	**23.8**	**0.666**
		CD*	86.3	68.8	15.8	1.7	0.688	0.203	32.9	0.586
		PC	80.5	29.2(27.1)	20.5	3.8	0.562	0.047	(47.5, 74.6)	(0.193, 0.454)
Scale-free	$(100, 100, 99)$	CD	99.2	74.8	17.9	6.5	0.756	0.245	**30.8**	0.610
		CD*	103.3	76.3	18.1	8.8	0.771	0.260	31.5	**0.611**
		PC	59.1	4.9(49.5)	2.8	1.9	0.549	0.032	(46.5, 96.0)	(0.032, 0.525)
	$(500, 100, 99)$	CD	98.8	85.0	13.4	0.4	0.859	0.140	**14.5**	**0.758**
		CD*	105.2	85.5	13.6	6.2	0.863	0.186	19.8	0.726
		PC	74.0	1.2(71.0)	0.0	1.8	0.729	0.023	(28.6, 99.5)	(0.007, 0.717)
Small-world	$(100, 100, 200)$	CD	145.2	69.1	49.1	26.9	0.346	0.523	157.9	0.250
		CD*	121.1	59.1	42.7	19.2	0.296	0.511	160.1	0.226
		PC	67.7	11.7(45.4)	7.7	3.0	0.285	0.045	(**146.0**, 191.3)	(0.046, **0.271**)
	$(500, 100, 200)$	CD	168.8	98.8	53.0	17.0	0.494	0.412	**118.2**	**0.367**
		CD*	135.1	82.0	46.6	6.5	0.410	0.393	124.5	0.324
		PC	117.0	43.0(26.2)	46.2	1.5	0.346	0.0130	(132.2, 158.4)	(0.158, 0.283)
Random DAG	$(100, 100, 101.5)$	CD	91.8	56.6	24.7	10.4	0.556	0.384	55.4	0.414
		CD*	61.0	38.5	18.9	3.5	0.381	0.365	66.5	0.311
		PC	66.2	22.4(28.7)	12.5	2.6	0.506	0.040	(**53.0**, 81.8)	(0.156, **0.441**)
	$(500, 100, 102.0)$	CD	103.4	76.2	23.4	3.8	0.746	0.263	29.8	0.591
		CD*	85.8	64.2	19.8	1.8	0.636	0.253	39.6	0.524
		PC	96.7	60.5(24.2)	10.6	1.3	0.837	0.0140	(**18.6**, 42.9)	(0.438, **0.755**)

CD is the result of our CD algorithm with the smallest SHD along the solution path; CD* is the result of our CD algorithm using our model selection criterion; The number in parentheses in column E for the PC algorithm reports the number of predicted undirected edges in the true skeleton

Table 2.2: Comparison between our CD algorithm and the EE-DAG sampler on simulated interventional data

Graph	(n, p, s_0)	Method	P	E	R	FP	TPR	FDR	SHD	JI
Bipartite	$(50, 50, 50)$	CD	30.8	19.4	6.5	4.8	0.389	0.362	**35.4**	0.316
		CD*	29.6	17.8	6.5	5.3	0.355	0.398	37.6	0.286
		EE	53.9	27.4	4.8	21.6	0.548	0.486	44.2	**0.362**
Scale-free	$(50, 50, 49)$	CD	48.5	30.2	6.8	11.4	0.617	0.376	**30.1**	0.452
		CD*	51.0	31.0	6.8	13.2	0.633	0.387	31.2	**0.454**
		EE	60.6	32.9	3.1	24.6	0.670	0.453	40.8	0.434
Small-world	$(50, 50, 100)$	CD	70.2	35.0	25.9	9.2	0.350	0.498	**74.2**	**0.260**
		CD*	39.5	21.2	15.8	2.5	0.212	0.458	81.2	0.181
		EE	62.4	30.6	16.1	15.7	0.306	0.507	85.0	0.234
Random DAG	$(50, 50, 46.8)$	CD	34.1	20.4	8.3	5.5	0.441	0.400	**31.9**	0.340
		CD*	26.1	16.4	6.8	2.9	0.361	0.363	33.2	0.296
		EE	50.0	25.2	7.0	17.8	0.547	0.492	39.3	**0.358**

CD is the result of our CD algorithm with the smallest SHD along the solution path; CD* is the result of our CD algorithm using our model selection criterion

Table 2.3: Comparison between our CD algorithm and the EE-DAG sampler on simulated interventional data

Graph	(n, p, s_0)	Method	P	E	R	FP	TPR	FDR	SHD	JI
Bipartite	$(250, 50, 50.0)$	CD	53.2	39.4	7.7	6.2	0.787	0.259	16.9	0.617
		CD*	36.1	28.3	6.0	1.8	0.566	0.215	23.6	0.488
		EE	51.8	47.9	1.8	2.1	0.958	0.073	**4.2**	**0.892**
Scale-free	$(250, 50, 49.0)$	CD	48.9	44.5	4.0	0.3	0.908	0.089	4.8	0.837
		CD*	50.6	44.6	4.2	1.9	0.910	0.118	6.3	0.815
		EE	50.6	48.4	0.6	1.6	0.988	0.0430	**2.2**	**0.946**
Small-world	$(250, 50, 100.0)$	CD	84.6	49.5	28.4	6.8	0.495	0.414	57.2	0.366
		CD*	44.2	28.6	15.3	0.4	0.286	0.351	71.8	0.247
		EE	81.7	70.5	7.0	4.2	0.705	0.136	**33.6**	**0.635**
Random-DAG	$(250, 50, 49.9)$	CD	49.5	36.5	9.6	3.4	0.743	0.261	16.2	0.592
		CD*	38.9	29.8	7.6	1.6	0.607	0.237	21.1	0.515
		EE	49.6	45.8	2.6	1.2	0.927	0.076	**4.8**	**0.864**

CD is the result of our CD algorithm with the smallest SHD along the solution path; CD* is the result of our CD algorithm using our model selection criterion

Table 2.4: Comparison between our CD algorithm and the GIES algorithm on simulated interventional data

Graph	(n, p, s_0)	Method	P	E	R	FP	TPR	FDR	SHD	JI
Bipartite	$(100, 100, 100)$	CD	98.2	63.0	18.6	16.5	0.630	0.355	**53.5**	**0.466**
		CD*	60.5	41.2	14.5	4.7	0.412	0.316	63.5	0.345
		GIES	322.5	82	13.8	226.7	0.820	0.745	244.7	0.242
	$(500, 100, 100)$	CD	104.2	81.7	17.1	5.5	0.816	0.217	**23.8**	**0.666**
		CD*	86.3	68.8	15.8	1.7	0.688	0.203	32.9	0.586
		GIES	187.9	92.4	7.6	88.0	0.924	0.507	95.5	0.474
Scale-free	$(100, 100, 99)$	CD	99.2	74.8	17.9	6.5	0.756	0.245	**30.8**	0.610
		CD*	103.3	76.3	18.1	8.8	0.771	0.260	31.5	**0.611**
		GIES	276.9	82.1	15.3	179.5	0.829	0.703	196.4	0.281
	$(500, 100, 99)$	CD	98.8	85.0	13.4	0.4	0.859	0.140	**14.4**	**0.758**
		CD*	105.2	85.5	13.6	6.2	0.863	0.186	19.8	0.726
		GIES	173.8	93.3	5.7	74.8	0.943	0.461	80.4	0.522
Small-world	$(100, 100, 200)$	CD	145.2	69.1	49.1	26.9	0.346	0.523	**157.8**	**0.250**
		CD*	121.1	59.1	42.7	19.2	0.296	0.511	160.1	0.226
		GIES	316.2	89.0	51.4	175.9	0.445	0.717	286.9	0.210
	$(500, 100, 200)$	CD	168.8	98.8	53.0	17.0	0.494	0.412	**118.2**	0.367
		CD*	135.1	82.0	46.6	6.5	0.410	0.393	124.5	0.324
		GIES	284.1	152.2	34.1	97.9	0.761	0.464	145.8	**0.459**
Random DAG	$(100, 100, 101.5)$	CD	91.8	56.6	24.7	10.4	0.556	0.384	**55.4**	**0.414**
		CD*	61.0	38.5	18.9	3.5	0.381	0.365	66.5	0.311
		GIES	318.2	77.2	20.6	220.6	0.761	0.757	244.9	0.226
	$(500, 100, 101.5)$	CD	103.4	76.2	23.4	3.8	0.746	0.263	**29.8**	**0.591**
		CD*	85.8	64.2	19.8	1.8	0.636	0.253	39.6	0.524
		GIES	192.9	93.0	8.7	91.2	0.911	0.518	100.2	0.461

CD is the result of our CD algorithm with the smallest SHD along the solution path.

CD* is the result of our CD algorithm using model selection criterion with $\alpha = 0.3$.

Table 2.5: Comparison among our CD algorithm and other algorithms on simulated observational data

Network	(n, p, s_0)	Method	P	E	R	FP	TPR	FDR	SHD	JI
Bipartite	$(50, 200, 200.0)$	CD	108.7	69.6	20.6	18.6	0.348	0.357	**148.9**	**0.290**
		CD*	90.2	59.6	17.8	12.8	0.298	0.333	153.2	0.258
		PC	75.7	26.9	34.2	14.6	0.134	0.643	187.7	0.108
		MMHC	101.0	34.9	11.8	54.2	0.174	0.655	219.3	0.131
		HC	393.0	108.7	31.2	253.1	0.544	0.723	344.4	0.225
Scale-free	$(50, 200, 199.0)$	CD	139.6	83.8	15.8	39.9	0.421	0.402	**155.1**	0.326
		CD*	201.8	106.8	21.6	73.4	0.537	0.470	165.6	**0.365**
		PC	99.5	46.5	23.1	30.0	0.234	0.532	182.5	0.185
		MMHC	97.8	43.3	8.2	46.2	0.218	0.556	201.9	0.171
		HC	349.1	118.1	25.7	205.3	0.593	0.662	286.2	0.275
Small-world	$(50, 200, 400.0)$	CD	88.2	28.9	36.2	23.0	0.072	0.533	394.1	0.058
		CD*	296.5	84.6	110.8	101.1	0.212	0.714	416.5	**0.138**
		PC	70.2	7.3	54.4	8.6	0.018	0.898	401.2	0.016
		MMHC	87.5	35.4	29.3	22.9	0.088	0.595	**387.5**	0.078
		HC	259.6	79.2	80.8	99.5	0.198	0.695	420.4	0.137
Random DAG	$(50, 200, 203.6)$	CD	113.0	68.8	28.1	16.1	0.339	0.386	**150.9**	**0.278**
		CD*	101.2	63.7	25.1	12.4	0.315	0.364	152.3	0.265
		PC	97.3	47.1	37.0	13.2	0.233	0.515	169.7	0.187
		MMHC	112.5	54.0	22.4	36.0	0.267	0.520	185.6	0.207
		HC	388.1	99.7	46.0	242.5	0.491	0.743	346.4	0.203

CD* is the result of our CD algorithm using our model selection criterion

Table 2.6: Comparison between our CD algorithm and the K2 Algorithm

		CD Algorithm					K2 Algorithm				
Network	(n, p, s_0)	P	TPR	FDR	SHD	JI	P	TPR	FDR	SHD	JI
asia	$(250, 8, 8)$	10.2	0.719	0.430	6.7	0.469	10.1	0.838	0.331	**4.7**	**0.597**
sachs	$(250, 11, 17)$	14.5	0.732	0.133	**6.6**	**0.659**	14.7	0.538	0.374	13.3	0.408
child	$(250, 20, 25)$	31.1	0.656	0.469	**23.3**	**0.416**	30.1	0.602	0.500	25.0	0.376
insurance	$(250, 27, 52)$	50.4	0.473	0.511	**53.2**	**0.316**	51.1	0.414	0.578	60.0	0.265
alarm	(250, 37, 46)	60.8	0.664	0.497	**45.6**	**0.401**	60.8	0.618	0.531	49.9	0.364
hailfinder	$(250, 56, 66)$	80.2	0.525	0.558	76.9	0.313	79.1	0.546	0.542	**73.0**	**0.331**
hepar2	$(250, 70, 123)$	137.6	0.269	0.756	**194.5**	**0.146**	139.9	0.236	0.792	204.8	0.124
pigs	$(250, 441, 592)$	773.65	0.863	0.334	**343.5**	**0.600**	788.8	0.704	0.472	547.4	0.432

Table 2.7: Comparison on the flow cytometry data set

Method	P	E	R	M	FP	SHD	JI
CD algorithm	17	10	3	7	4	**14**	**0.370**
CD algorithm*	14	8	3	9	3	15	0.308
PC algorithm	17	5(3)	4	8	5	17	0.276
Order-graph sampler	20	8	4	8	8	20	0.250
EE-DAG sampler	26	9	6	5	11	22	0.243

Results given a correct ordering

Method	P	E	R	M	FP	SHD	JI
CD algorithm	28	18	0	2	10	12	0.600
K2 algorithm	29	18	0	2	11	13	0.581

The order-graph sampler result comes from the mean graph (Figure 11 in Ellis and Wong, 2008)

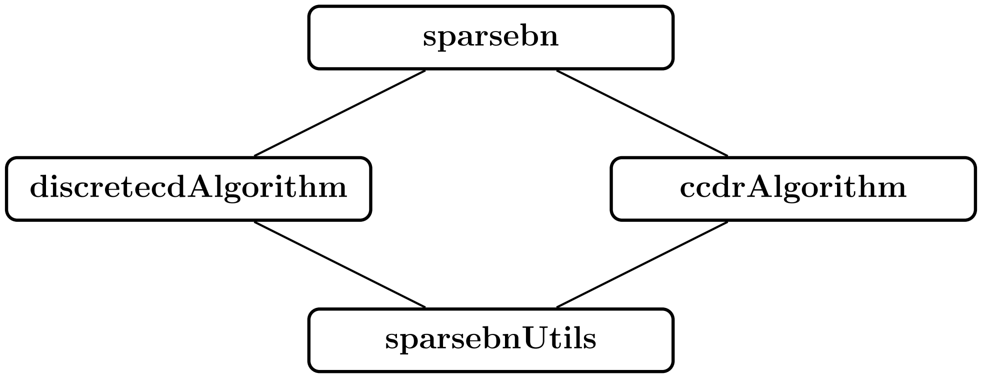
CHAPTER 3

R Packages: discretecdAlgorithm and sparsebn

We have developed an **R** package **discretecdAlgorithm** based on the algorithm described in Chapter 2. The package is now available on **CRAN**. It aims to handle discrete data sets, and it is able to take as input observational data, interventional data, or a mixture of both. In this chapter, we will go through the design of this package and show some examples.

3.1 Introduction and Related Packages

The **discretecdAlgorithm** belongs to a family of **R** packages, it imports some of the utility functions from the package **sparsebnUtils**, and exports to the package **sparsebn** (Aragam et al, 2017b) which is a wrapper of several structure learning packages. Structure of the family of our packages is shown as follow:



Note that the **ccdrAlgorithm** (Aragam et al, 2017b) package is another structure learning package for continuous Bayesian networks. Users can access our discrete CD algorithm 1 via either the **discretecdAlgorithm** package by `cd.run` or the **sparsebn** package by `estimate.dag`. We recommend users to interact with the **spasrsebn** package because it is more user-friendly and is compatible with several commonly used packages for graphs: **graph** (Gentleman et al, 2016), **network** (Butts, 2008), and **igraph** (Csárdi and Nepusz, 2006). Please refer to Aragam et al (2017b) for more details.

3.2 Using the sparsebn Package for Structure Learning of Discrete Bayesian Networks

In this section, we will introduce the estimation methods for discrete data in **sparsebn** package. The main function is `estimate.dag`, which can be called as follows:

```
R> estimate.dag(data, lambdas = NULL, lambdas.length = 20,
+    whitelist = NULL, blacklist = NULL, error.tol = 1e-04,
+    max.iters = NULL, edge.threshold = NULL, concavity = 2,
+    weight.scale = 1, convLb = 0.01, upperbound = 100,
+    adaptive = FALSE, verbose = FALSE)
```

The main arguments are `data`, `lambdas`, and `lambdas.length`. By default, the `lambdas` argument is `NULL` and a standard sequence of $J = 20$ regularization parameters is generated. If desired, the user can pre-compute a vector of regularization parameters to be used instead, in which case this vector should be passed through the `lambdas` argument. If there is prior knowledge of (directed) edges that are known to be present in the network, these can be specified via the `whitelist` argument. Similarly, if there is prior knowledge of (directed) edges that are known to be absent from the network, these can be specified via the `blacklist` argument. The rest of the arguments control the convergence of the internal algorithms, and are intended for advanced users. This method returns a `sparsebnPath` object described in Section 3.2.1 which stores the solution path.

The objects returned by `estimate.dag` are graphs, and in particular they do not include estimates of model parameters such as edge weights or conditional variances. To obtain these parameters, **sparsebn** includes the `estimate.parameters` method as follows:

```
R > estimate.parameters(fit, data, ...)
```

where `fit` is the output of `estimate.dag` and `data` is the data to be used for parameter estimation.

3.2.1 Data Structures

sparsebn uses three different S3 classes in order to represent data (`sparsebnData`), graphs (`sparsebnFit`), and solution paths (`sparsebnPath`). For each of these classes, the usual generics are defined such as `print`, `summary`, and `plot`.

The `sparsebnData` class is used to represent both continuous and discrete data with experimental interventions, and to use the discrete CD algorithm we should specify `type = "discrete"`. Observational data corresponds to the degenerate case where our data set does not contain any interventions, and is treated as such by the **sparsebn** package. The slots are:

- `data`: This is the original data as a data frame with n observations and p variables.
- `type`: `"discrete"` or `"continuous"`.
- `levels`: A list of levels for each variable. This is a list of length p whose jth component is a vector containing the levels of the jth variable.
- `ivn`: The list of interventions for each observations. This is a list of length n whose ith component is a vector of node names (or indices) that are under intervention for the ith observation. Default is `NULL` for observational data.

The `sparsebnPath` class represents a solution path, which is the output of the main function `estimate.dag`. Internally, this is a `list` of `sparsebnFit` objects whose jth component corresponds to the jth value of λ in the solution path, $\lambda_{\max} > \lambda_1 > \cdots > \lambda_{\min}$. Since this class is essentially a wrapper for this list, it has no named slots. The `sparsebnFit` class represents an individual graph estimate from a DAG learning algorithm. The graph itself is stored as an `edgeList` object in the `edges` slot, which is an internal implementation of a child-parent edge list.

3.2.2 Installation

sparsebn is an open-source package and is made freely available through CRAN. To install the latest stable version in **R**,

```
R> install.packages("sparsebn")
```

For advanced users, the development versions can be downloaded directly from GitHub. Using **devtools**, the entire suite of packages can be installed via

```
R> devtools::install_github(c("itsrainingdata/sparsebnUtils/dev",
+   "itsrainingdata/ccdrAlgorithm/dev",
+   "gujyjean/discretecdAlgorithm/dev",
+   "itsrainingdata/sparsebn/dev"))
```

Note that by installing the **sparsebn** package, users have automatically installed packages **discretecdAlgorithm**, **ccdrAlgorithm** and **sparsebnUtils**.

3.3 An Example of Discrete Cytometry Data

To illustrate the use of the **sparsebn** package on discrete data, we use the flow cytometry data set in Chapter 2 Section 2.5.2 as an example in this section. First, we load the data from the **sparsebn** package:

```
R> library("sparsebn")
R> data("cytometryDiscrete")
R> names("cytometryDiscrete")
  [1] "dag"  "data" "ivn"
```

Note that this is not a `data.frame`, but instead a list of **R** objects that will be useful for this specific example. Each component of this list stores an important part of the experiment:

- `dag` is the consensus network with 11 nodes and 17 edges, as described above.

- `data` is raw data collected from these experiments.

- `ivn` is a list of interventions for each observation in the dataset.

The remaining of this section will go through the main steps for structure learning of this discrete flow cytometry data set: (1) Converting `cytometryDiscrete` to a `sparsebnData` object, (2) Running the structure learning algorithm, (3) Incorporating with prior knowledge, (4) Checking the solution path, (5) Estimating the parameters, (6) Selecting regularization parameter.

3.3.1 Getting `sparsebnData` object

In order to distinguish different types of data—namely, experimental versus observational and continuous versus discrete—we use the `sparsebnData` class which wraps a `data.frame` into an object containing this auxiliary information. All of the methods implemented in **sparsebn** expect input as `sparsebnData`.

To use this class, we need two important pieces of information: The raw data as a `data.frame`, and a list of interventions for each observation in the dataset. If the dataset does not contain any interventions, then the latter can be omitted. In order to create a `sparsebnData` object from the cytometry data, we first extract the necessary objects from `cytometryDiscrete`:

```
R> cyto.raw <- cytometryDiscrete$data
R> cyto.ivn <- cytometryDiscrete$ivn
```

Finally, for discrete data, we may wish to manually specify the levels of each variable, which can be done using the `levels` argument. When this argument is omitted, we attempt to automatically infer the levels. Note that in doing so, however, levels which are missing from the data will not be recognized by the `sparsebnData` constructor. We can manually set the `levels` argument as follows:

```
R> cyto.levels <- lapply(1:ncol(cyto.raw),
+   function(x){c(0, 1, 2)})
R> names(cyto.levels) <- colnames(cyto.raw)
```

Now we can create the required `sparsebnData` object:

```
R> cyto.data <- sparsebnData(data = cyto.raw,
+   type = "discrete", ivn = cyto.ivn, levels = cyto.levels)
```

Notice that we need to explicitly specify that the data is discrete, and the last argument `levels` can be omitted since there is no missing levels in this case.

```
> print(cyto.data)
        raf  mek  plc  pip2  pip3  erk  akt  pka  pkc  p38  jnk
   1:    0    0    0     1     2    1    0    2    0    1    0
   2:    0    0    0     0     2    2    1    2    0    1    0
   3:    0    0    1     1     2    1    0    2    1    0    0
   4:    0    0    0     0     2    1    0    2    0    2    0
   5:    0    0    0     0     2    1    0    2    0    0    0
  ---
5396:    0    0    0     0     1    1    0    1    1    0    0
5397:    0    0    0     0     0    1    1    0    0    1    1
5398:    0    0    0     0     1    1    0    1    1    0    0
5399:    0    1    0     0     0    0    0    1    1    0    0
5400:    1    1    0     0     1    1    0    1    1    1    0

5400 total rows (5390 rows omitted)
Discrete data w/ interventions on 3600/5400 rows.
```

This is an example of a mixture of observational and interventional data set where 3600 out of 5400 data points were under intervention.

3.3.2 Structure Learning

To learn the structure of a Bayesian network from this data we use the `estimate.dag()` method, which runs Algorithm 1 when we set `type = "discrete"`. To call this method using the default parameter settings, use:

```
R> cyto.learn <- estimate.dag(data = cyto.data)
R> print(cyto.learn)
sparsebn Solution Path
 11 nodes
 5400 observations
 8 estimates for lambda in [1.791, 9.7712]
 Number of edges per solution: 0-6-9-13-15-21-30-38

R> summary(cyto.learn)
sparsebn Solution Path
 11 nodes
 5400 observations
 8 estimates for lambda in [1.791, 9.7712]
 Number of edges per solution: 0-6-9-13-15-21-30-38

     lambda nedge
1  9.771153     0
2  7.668010     6
3  6.017547     9
4  4.722330    13
5  3.705896    15
6  2.908238    21
7  2.282269    30
8  1.791033    38
```

In addition to `data`, there are several optional parameters that can be passed to function `estimate.dag`. The main arguments of interest are `lambdas` and `lambdas.length`, which allow the user to adjust the grid of regularization parameters $\lambda_{\max} > \lambda_1 > \cdots > \lambda_{\min}$ in (2.9).

By default, `estimate.dag` produces a solution path of 20 estimates with the grid chosen adaptively to the data. This grid can be shortened or lengthened by specifying the parameter `lambdas.length`:

```
R> estimate.dag(data = cyto.data, lambdas.length = 50)
```

For even more fine-tuning, the `lambdas` argument allows the user to explicitly input their own grid. For convenience we have included the `generate.lambdas` method, which provides a mechanism for generating grids of arbitrary lengths on either a linear or log scale. To generate a grid with a linear scale, use `scale = "linear"`:

```
R> cyto.lambdas <- generate.lambdas(lambda.max = 10,
+    lambdas.ratio = 0.001, lambdas.length = 10,
+    scale = "linear")
R> cyto.lambdas
[1] 10.00  8.89  7.78  6.67  5.56  4.45  3.34  2.23  1.12  0.01
```

To use a log scale, use `scale = "log"`:

```
R> cyto.lambdas <- generate.lambdas(lambda.max = 10,
+    lambdas.ratio = 0.001, lambdas.length = 10, scale = "log")
R> cyto.lambdas
[1] 10.000  4.642  2.154  1.000  0.464  0.215  0.100  0.046
    0.022  0.010
```

If we run the CD algorithm with a user specified lambda sequence:

```
R> estimate.dag(data = cyto.data, lambdas = cyto.lambdas)
sparsebn Solution Path
 11 nodes
 5400 observations
 4 estimates for lambda in [1, 10]
 Number of edges per solution: 0-13-31-42
```

Note that there are only 4 estimates because for discrete CD algorithm we have set the default upper limit on the number of edges estimated to be `alpha = 3` times the number of nodes, that is 33 in this case. `alpha` is an argument in `cd.run` of **discretecdAlgorithm** package. We can justify the value of `alpha` to get denser estimations:

```
R> library("discretecdAlgorithm")
R> cd.run(indata = cyto.data, lambdas = cyto.lambdas,
+   alpha = 10)
sparsebn Solution Path
 11 nodes
 5400 observations
 10 estimates for lambda in [0.01, 10]
 Number of edges per solution: 0-13-31-42-53-55-55-55-55-55
```

This feature has not been exported to the **sparsebn** package yet.

3.3.3 Prior Knowledge

In some contexts, users may have prior knowledge regarding edges that must be present or absent from the network. For example, it may already be known that PIP3 regulates PIP2 (see Figure 2.4 (A)). In this case, estimation of the underlying network can be substantially improved by incorporating this information into the estimation procedure. With the **sparsebn** package, this can be done via *whitelists* and *blacklists*, which specify edges that must be present and absent, respectively.

For example, to specify a known relationship between PIP3 and PIP2, we can create a whitelist as follows:

```
R> whitelist <- matrix(c("pip3", "pip2"), nrow = 1)
R> estimate.dag(cyto.data, whitelist = whitelist)
sparsebn Solution Path
 11 nodes
 5400 observations
 8 estimates for lambda in [1.791, 9.7712]
 Number of edges per solution: 1-7-10-14-15-21-28-37
```

We can see that now instead of the `NULL` graph, the solution path starts with a DAG with 1 edge. Thus, this whitelist ensures that the edge PIP3→PIP2 will be present in the final estimates. The `whitelist` argument should be a two-column matrix, where the first column stores parents and the second stores children (i.e., a from-to adjacency list):

```
R> whitelist
     [,1]   [,2]
[1,] "pip3" "pip2"
```

Similarly, we can specify a blacklist, which stores edges that are known to be absent. For example, we can forbid any edges between RAF and MEK as follows:

```
R> blacklist <- rbind(c("raf", "jnk"), c("jnk", "raf"))
R> estimate.dag(cyto.data, blacklist = blacklist)
```

As with the whitelist, the blacklist should be a two-column matrix. Note that we specify *both* directions RAF→MEK and MEK→RAF. If it is known that the direction can only go in one direction, then a single direction may be specified instead.

Blacklists are useful for specifying known root and leaf nodes in a Bayesian network. In the cytometry network, PIP3 is a root node (i.e., it has no parents). Thus, we can forbid any edges pointing *into* PIP3. Similarly, JNK, P38, and AKT are leaf nodes (i.e., they have no children), so we can forbid any edges pointing *away* from all three nodes. To specify this, we make use of the `specify.prior` function, which automatically builds an appropriate blacklist given the names of the root and leaf nodes. Any number of root and/or leaf nodes can be specified.

```
R> blacklist <- specify.prior(roots = "pip3",
+   leaves = c("jnk", "p38", "akt"),
+   nodes = names(cyto.data$data))
R> estimate.dag(cyto.data, blacklist = blacklist)
```

Finally, whitelists and blacklists can be combined arbitrarily, as long as they are consistent in the sense that no edge appearing in the whitelist appears in the blacklist, and vice versa. This allows for a powerful specification of prior knowledge in learning networks from data.

3.3.4 Solution paths

The output of `estimate.dag` is a `sparsebnPath` object, which stores the entire solution path that is returned by the method, and each estimate is stored as the internal class `sparsebnFit`. Since `sparsebnPath` objects also inherit from the `list` class in base **R**, we can inspect the first solution using ordinary **R** indexing. Note that for `sparsebnFit` objects, the `print` and `summary` methods are identical, so the output below is shown only once.

```
R> print(cyto.learn[[1]])
R> summary(cyto.learn[[1]])
CCDr estimate
5400 observations
lambda = 9.77115285854563

DAG:
<Empty graph on 11 nodes.>
```

The first estimate will always be the empty graph, which is a consequence of how we employ warm starts in the solution path. The third estimate, for example, shows a bit more structure:

```
R> print(cyto.learn[[3]])
R> summary(cyto.learn[[3]])
CCDr estimate
5400 observations
lambda = 6.01754700798802

DAG:
```

```
[raf]   pka
[mek]   pka   raf
[plc]   pip2  pka
[pip2]
[pip3]
[erk]   akt   pka
[akt]   pka
[pka]
[pkc]
[p38]   pka
[jnk]
```

Each row in the output above corresponds to a child node—indicated by the square brackets—with its parents listed to the right without brackets. Formally, this is an adjacency list ordered by children. For large graphs, explicit output of the parental structure as shown here is omitted by default, however, this behaviour can be overridden via the `maxsize` argument. Alternatively, we can retrieve the adjacency matrix for this estimate:

```
R> get.adjacency.matrix(cyto.learn[[3]])
11 x 11 sparse Matrix of class "dgCMatrix"
   [[ suppressing 11 column names 'raf', 'mek, 'plc' ... ]]

raf  . 1 . . . . . . . . .
mek  . . . . . . . . . . .
plc  . . . . . . . . . . .
pip2 . . 1 . . . . . . . .
pip3 . . . . . . . . . . .
erk  . . . . . . . . . . .
akt  . . . . . 1 . . . . .
pka  1 1 1 . . 1 1 . . 1 .
```

```
pkc  . . . . . . . . . . .
p38  . . . . . . . . . . .
jnk  . . . . . . . . . . .
```

Note the use of the **Matrix** package (Bates and Maechler, 2018), which reduces the memory footprint on large graphs. Finally, for large graphs, it may be desirable to inspect a subset of nodes, which can be done using the `show.parents` method:

```
R> show.parents(cyto.learn[[3]], c("raf", "pip2"))
[raf]   pka
[pip2]
```

3.3.5 Parameter estimation

It is important to note that the output of `estimate.dag` is a sequence of *graphs*, i.e., no parameters (edge weights, etc.) have been estimated at this stage. The next step is to estimate the values of the parameters associated with the underlying joint distribution. This is easy to do:

```
R> cyto.param <- estimate.parameters(cyto.learn,
+   data = cyto.data)
```

The output is a list of parameters in our multi-logit model (2.2), for example the parameters for the third estimation is:

```
R> cyto.param[[3]]
$raf
  (Intercept)      pka_1     pka_2
1   0.4219942 -0.9457959 -2.282747
2   1.8258925 -3.6595227 -5.034717

$mek
  (Intercept)     raf_1      raf_2     pka_1     pka_2
```

```
1     -2.12277  1.951786 -0.0136459  1.096985 -2.494880
2    -13.34661 13.228964 14.2988204 -3.373844 -4.733759

$plc
  (Intercept)   pip2_1   pip2_2        pka_1        pka_2
1    -2.204715 2.394603 1.886452   -0.5895796   -0.1304795
2    -1.724715 4.910626 6.774940  -18.8440374  -15.3974901

$pip2
integer(0)

$pip3
integer(0)

$erk
  (Intercept)       akt_1      akt_2    pka_1    pka_2
1   -0.4927002 -0.1564742   9.716911 2.605460 2.757640
2   -4.1401389  3.0175745  19.423297 3.319035 4.235309

$akt
  (Intercept)       pka_1      pka_2
1   0.00321718 -0.7273075   -1.184633
2   0.31229882 -6.0676902  -13.011249

$pka
integer(0)

$pkc
integer(0)
```

```

$p38
  (Intercept)      pka_1      pka_2
1   -1.2242978 -0.4220145 -0.8272241
2    0.3761702 -7.0447429 -2.2011545

$jnk
integer(0)
```

All these parameters are estimated using the **R** package **nnet**.

3.3.6 Model selection

Unlike existing methods, the output of `estimate.dag` is a solution path with multiple estimates of increasing complexity, indexed by the regularization parameter. Thus, it is important to be able to pick out estimates for inspection and further exploration. For *ad hoc* exploration, the `select` method is useful: This allows one to select an estimate from a `sparsebnPath` object based on the number of edges, the regularization parameter λ, or the index j. When selecting by the number of edges or by λ, fuzzy matching is used by default so that the closest match is returned to within a given tolerance. Selecting by index is equivalent to subsetting as usual with the subset operator '`[[]]`'. To save space, the output of the following code is suppressed:

```
R> select(cyto.learn, edges = 9)
R> select(cyto.learn, edges = 10)
```

In the first line above, an exact match is returned. In the second line, the closest match is returned since there is no graph with exactly 10 edges in the solution path.

```
R> select(cyto.learn, lambda = 9.75)
R> select(cyto.learn, lambda = 9.7)
```

In both of the above examples, the closest match is returned.

```
R> select(cyto.learn, index = 4)
R> cyto.learn[[4]]
```

In the both lines above, an exact match is returned. Note that the second line is equivalent to the first.

For practical applications, one is often concerned with selecting an optimal value of λ. We implemented the empirical model selection criterion described in Chapter 2 Section 2.2.3 in **sparsebn** via the method `select.parameter`:

```
R> selected.lambda <- select.parameter(cyto.learn, cyto.data,
+   alpha = 0.3)
R> selected.lambda
[1] 5
```

Our model selection method selected out the 5th DAG with 15 edges along the solution path. Note that the default setting for the argument `alpha` is 0.1, this value works well for continuous data sets. As for discrete data sets the empirical value for `alpha` should be set as 0.3.

3.4 Discussions

In this chapter, we went through some examples for learning discrete Bayesian networks with **sparsebn** package, which imports the main algorithm for structure learning of discrete data from **discretecdAlgorithm** package. Implementation for our discrete CD algorithm includes the following features:

- Learning from a mixture of interventional and observational data set.
- Having competitive performance with high-dimensional data.
- Accepting prior knowledge: black-list and white-list.
- Providing an empirical model selection method to determine the tuning parameter λ

along the solution path.

In our future updates we can provide the users an option to input a known ordering of the graph, in addition to black-list and white-list users might also have a prior knowledge on the topological ordering of the network. Also in our current implementation of the discrete CD algorithm, we store the DAG structure in matrix form, to save memory and speed up our algorithm we can implement it in a sparse matrix form like what the authors did for the **ccdrAlgorithm** package. Moreover, we might further scale our algorithm by introducing stochastic gradient descent when the number of observations is big.

To sum up, we have developed a user friendly **R** package so that users can have convenient access to our algorithm. And we will keep maintaining and updating our packages in the future.

CHAPTER 4

Learning Massive Gaussian Bayesian Networks

4.1 Motivation and Outline

The DAG space grows super-exponentially in the number of nodes p. The number of DAGs when $p = 1, 2, 3, 4, 5$ is $1, 3, 25, 543, 29281$, respectively. Although there is no closed-form formula for counting the number of DAGs given p, Robinson (1977) showed the following recursive equation to count the number of directed acyclic graphs,

$$\begin{cases} a_p = \sum_{k=1}^{p} (-1)^{k+1} \binom{p}{k} 2^{k(p-k)} a_{p-k} \\ a_0 = 1 \end{cases},$$

where a_p is the number of DAGs with p nodes. There exist some effective algorithms to learn the structure of DAGs, but as the number of nodes increases it is still very challenging to scale the algorithm, and the amount of time it takes for structure learning of large Bayesian networks might be hours or even days. In this chapter, we propose a three step divide-and-conquer framework that integrates various structure learning algorithms for massive networks. Our proposed method is able to significantly reduce the running time without losing much accuracy.

Many real-world networks show block structure to some degree, with weak or no connections between subgraphs. We can speed up the structure learning process by the following steps:

1. Break down the full DAG into several disconnected small networks using a clustering algorithm (Partition/P-step).

2. Use a structure learning algorithm like the CD algorithm proposed in Chapter 2 to learn the DAG structure for each subgraph (Estimation/E-step).

3. Finally recover edges among disconnected subgraphs (Fusion/F-step).

We will call this three step framework the PEF method hereafter.

This chapter focuses on the case of continuous observational data. All technical details, computer implementations and simulation results are for observational continuous data. The remaining of this Chapter is organized as follows: Section 4.2 introduces the statistical model and features for the PEF method. Section 4.3 describes our PEF method in detail. Section 4.4 demonstrates our simulation results when applying the PEF method to real networks with comparisons to other DAG learning algorithms. Section 4.5 summarizes our contribution in this chapter and future directions.

4.2 A Gaussian Model for Continuous Data and Assumptions

In this chapter we consider the case of continuous networks, where the variables $X_1, \ldots, X_p$ jointly follow a multivariate normal distribution. In addition, we assume the data sets to be purely observational. Discrete networks and interventional data sets will be left as future work. Furthermore, since state-of-the-art algorithms can efficiently recover structures of DAGs with hundreds of nodes, our algorithm is mainly designed for networks with thousands of nodes or even bigger. Also, in many applications only high-dimensional data is available for massive networks, so we will focus on the high-dimensional scenario in this chapter. Another assumption that motivates the partition step is that the full DAG can be roughly separated into subgraphs. And finally faithfulness is assumed for the PEF method. Detailed discussion on the assumptions will be in Section 4.2.2.

4.2.1 Gaussian model for continuous data

Gaussian Bayesian network is a Bayesian network where all its variables $X_1, \ldots, X_p$ are continuous and all the conditional probability densities (CPD) in Equation 1.1 are linear

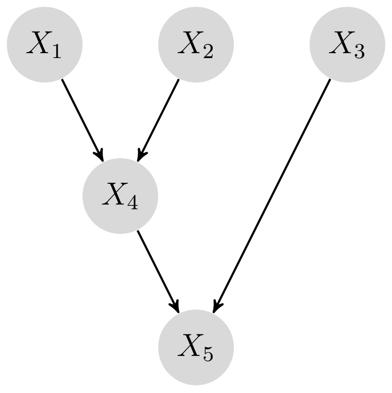
Gaussian. The CPD can be written as the following linear structural equation model,

$$X_j = \sum_{i=1}^{p} \beta_{ij} X_i + \epsilon_j, \quad \epsilon_j \sim N(0, \sigma_j^2), \quad j = 1, ..., p. \tag{4.1}$$

Note that β_{ij} represents the influence of X_i on X_j, therefore if $X_i \notin \Pi_j^{\mathcal{G}}$ then $\beta_{ij} = 0$. So Equation 4.1 can also be written as,

$$X_j = \sum_{i \in \Pi_j^{\mathcal{G}}} \beta_{ij} X_i + \epsilon_j, \tag{4.2}$$

where all $\beta_{ij} \neq 0$. If we let $B = (\beta_{ij}) \in \mathbb{R}^{p \times p}$ be the parameter matrix, it can also be regarded as the weighted adjacency matrix, where the weight for edge $(i, j) \in E$ is the coefficient β_{ij}. For example, suppose $\mathcal{G}$ is a DAG with 5 nodes $V = \{1, 2, 3, 4, 5\}$, and the DAG structure is illustrated in Figure 4.1 (a).



(a)

The Gaussian model for DAG in (a) should be:

$$\begin{cases} X_1 = \epsilon_1 \\ X_2 = \epsilon_2 \\ X_3 = \epsilon_3 \\ X_4 = \beta_{14} X_1 + \beta_{24} X_2 + \epsilon_4 \\ X_5 = \beta_{45} X_4 + \beta_{35} X_3 + \epsilon_5 \end{cases}$$

(b)

Figure 4.1: An example on the linear structural equation model.

Recall in Chapter 2, $\boldsymbol{\beta}_{j \cdot i}$ is the influence of X_i has on X_j which is a matrix, and for the Gaussian case it reduces to a scaler β_{ij}. Therefore the parameter matrix B for the discrete model is a four-way array. In our discrete CD algorithm, we learn the structure of a DAG via the sparsity pattern of $\boldsymbol{\beta}_{j \cdot i}$, similarly we can learn the DAG structure via the sparsity pattern of β_{ij}. Similar to Equation 2.8, we have,

$$\beta_{ij} = 0 \qquad \Longleftrightarrow \qquad i \notin \Pi_j^{\mathcal{G}}.$$

If we rewrite Equation 4.1 in matrix form, it should be

$$X = BX + E, \qquad E \sim N(0, \Sigma) \tag{4.3}$$

where $\Sigma = \text{diag}(\sigma_1^2, ..., \sigma_p^2)$. Let data set generated from a continuous Bayesian network $\mathcal{G}$ be $\mathcal{X} = (\mathcal{X}_1| \dots |\mathcal{X}_p)_{n\times p}$, and let B_j be the jth column of B. Then the log-likelihood for this Gaussian model becomes,

$$\ell(B, \Sigma) = \sum_{j=1}^{p} \left[-\frac{n}{2} \log(\sigma_j^2) - \frac{1}{2\sigma_j^2} \|\mathcal{X}_j - \mathcal{X} B_j\|^2 \right]. \tag{4.4}$$

4.2.2 Features and assumptions

The major challenge we wish to conquer in this chapter is the computational challenge for massive size networks with thousands or even ten thousands of nodes. The PEF framework should have the following features:

i. Fast learning of massive networks with thousands or even ten thousands of nodes.

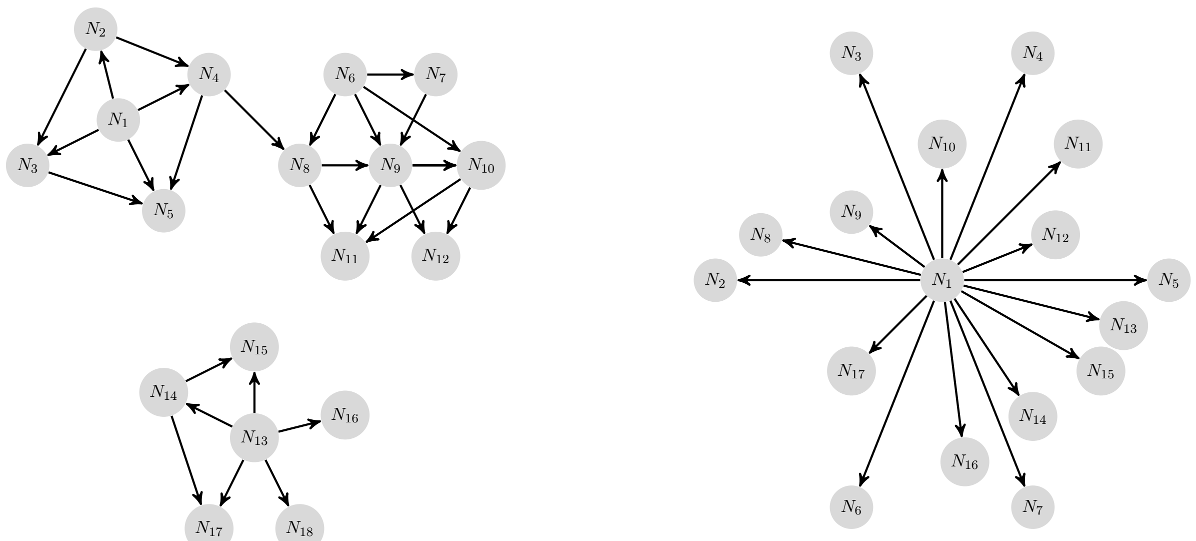
ii. Works well on high-dimensional data.

iii. Do not assume a prior knowledge on ordering of nodes.

Remark 4. Our PEF framework is specifically designed for massive networks, so we recommend researchers use this algorithm for large networks rather than small ones with less than a few hundreds of nodes. This is because the way we formulate our algorithm might sacrifice some accuracy for speed. When the network is too small the advantage of our algorithm in speed will not be that obvious or necessary. So feature (i.) is not only a feature of our PEF method but also an assumption. In addition, our main focus is on high-dimensional setting, which is a very common scenario for massive networks.

There are two assumptions we need for the PEF method. We will discuss them in detail in the remaining of this section.

This assumption is needed in the first P-step. As mentioned in Section 4.1, we use a clustering algorithm to breakdown the full DAG into small disconnected sub-networks in the P-step. An implicit assumption is that the full DAG has clustered structure where edges are denser within clusters than between clusters.



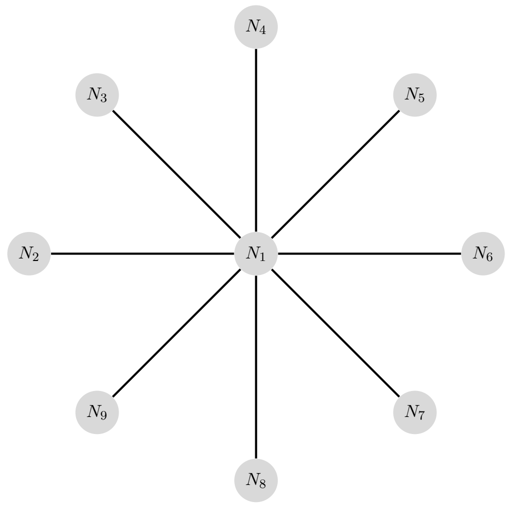
(a) $\mathcal{G}_1$

(b) $\mathcal{G}_2$

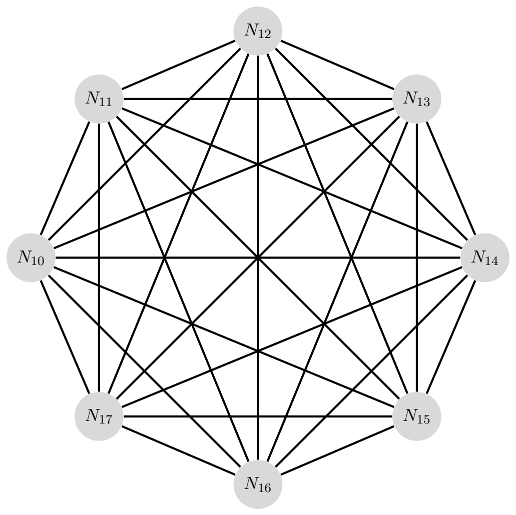
Figure 4.2: Example for DAGs with and without clusters

As we can see, there are 3 clusters in Figure 4.2 (a), where $C_1 = \{N_1, N_2, N_3, N_4, N_5\}$, $C_2 = \{N_6, N_7, N_8, N_9, N_{10}, N_{11}, N_{12}\}$, and $C_3 = \{N_{13}, N_{14}, N_{15}, N_{16}, N_{17}, N_{18}\}$. And we can see C_1 and C_2 are connected while C_3 is isolated from the first two clusters. We can break down $\mathcal{G}_1$ into three clusters by cutting off the edge $4 \to 8$. On the other hand, if we look at Figure 4.2 (b), $\mathcal{G}_2$ just has one big cluster. It does not make sense no matter how we partition the DAG into more than one sub-network. And if we force the DAG into two sub-networks, there might be some serious consequences when we run the estimation step. For example, if we partition the DAG into two clusters $C_1 = \{N_1, N_2, N_3, N_4, N_5, N_6, N_7, N_8, N_9\}$ and $C_2 = \{N_{10}, N_{11}, N_{12}, N_{13}, N_{14}, N_{15}, N_{16}, N_{17}, N_{18}\}$. And if ignoring directions, in the second E-step the graph we would recover is shown in Figure 4.3.

If we compare Figure 4.3 with Figure 4.2 (b), it is easy to see that we can recover the



(a) $\mathcal{G}_{C_1}$



(b) $\mathcal{G}_{C_2}$

Figure 4.3: Example for DAGs with and without clusters

true conditional dependency for cluster C_1 where node $\{N_2, N_3, N_4, N_5, N_6, N_7, N_8, N_9\}$ are all connected to N_1. However, without the hub node N_1, $\mathcal{G}_2$ becomes a clique.

Faithfulness

Definition for faithfulness can be found in Chapter 1. With this assumption, *d*-separation and conditional independence in the joint distribution coincides. This assumption is needed in our fusion step, when we refine DAG structure using a sequence of conditional independence tests. It may seem like that the faithfulness assumption can be quite restrictive, but Meek (1995) has proved for Gaussian networks, with respect to Lebesgue measure, the set of distributions that are unfaithful to a DAG G has measure 0. So our assumption of faithfulness is reasonable here.

4.3 A Divide-and-Conquer Framework

In this section we propose a framework for learning massive size sparse Bayesian networks, and all technical details discussed are specifically for observational Gaussian data. We describe the three steps: Partition, Estimation, Fusion in the following three subsections.

4.3.1 Partition

Assume we have a DAG $\mathcal{G}$ constructed with several disconnected or weakly connected subnetworks, we want to break it into subgraphs using clustering techniques. Let $C_i, i = 1, ..., k$, be the k clusters we get in the P-step, $S_i = |C_i|$ the size of the ith cluster, s_w the number of total edges within clusters, and s_b the number of edges between different clusters. Alternatively, we can think of s_b as the number of edges we "cut-off" in the clustering process. And those cut-off edges may be recovered in the third F-step of our algorithm. We wish to have the following three desirable parameters in the clustering step,

- k as large as possible: The larger the k the faster the second E-step and the more running time we can save.
- s_b as small as possible: We want to breakdown the full DAG without too much damage on the DAG structure. This will help our second step to recover the majority number of edges.
- $\max(S_i)$ as small as possible: Running time of the E-step will be dominated by the largest cluster, so the more uniform the size of the clusters the more time we can save.

We propose a modified version of hierarchical clustering with average linkage that automatically chooses the number of clusters k. Generally, hierarchical clustering falls into two categories: agglomerative method and divisive method. Suppose there are N nodes $Z_i, i = 1, \ldots, N$ to be clustered.

- *Agglomerative method*: It is also known as *Agnes*. It is a bottom-up method that starts with N clusters where each cluster is a single node Z_i. In every step, two most similar clusters merges into a bigger cluster. The algorithm stops until every nodes merges into one big cluster $\{Z_1, \ldots, Z_N\}$.
- *Divisive method*: It is also known as *Diana*. It is a top-down method that starts with a cluster with all nodes $\{Z_1, \ldots, Z_N\}$. In each step, the most heterogeneous cluster will

be divided into two sub-clusters. The algorithm stops until there are N nodes where each single node Z_i is a cluster.

Both methods will return a tree with N levels, and divisive methods are like the inverse of the agglomerative methods. The agglomerative method is more intuitive, distance between two clusters is easy to calculate, and there are varieties of definition for distance among clusters. On the other hand, for divisive method after selects out the most heterogeneous cluster, there are $2^n - 1$ ways of splitting a cluster with n nodes. Therefore divisive methods need other heuristics to split the cluster. **R** applications on the hierarchical clustering are mainly agglomerative methods, `hclust` in package **stats** and `agnes` in package **cluster** are two examples, they have some different implementations for the linkage functions. A divisive method available in **R** is `diana` in package **cluster**, and there is only one implementation available. In the partition step, we choose to use agglomerative clustering.

Our choice of distance

Let X and Y be two vectors in $\mathbb{R}^n$, the most common distance between X and Y defined for clustering algorithms is the Euclidean distance $\mathbf{d}_E(X,Y) = \|X - Y\|_2$. If X, Y are centered and scaled to have variance 1, $\mathbf{d}_E(X,Y) = \sqrt{2 - 2r_{XY}}$, where $r_{XY} = corr(X,Y)$. It is obvious that if X and Y are perfectly positively correlated, $\mathbf{d}_E(X,Y)$ is the smallest, 0. And when they are perfectly negatively correlated, $\mathbf{d}_E(X,Y)$ is the largest, 2. We can see the Euclidean distance does not match our intuition for distance between nodes in Bayesian networks. Two nodes should have the largest distance when their correlation is 0, and the smallest distance when they are perfectly correlated. So $|r_{XY}|$ can be a measure of similarity between X an Y. We define the distance between nodes of a Bayesian network as

$$\mathbf{d}_B(X,Y) = 1 - |r_{XY}|, \tag{4.5}$$

where $\mathbf{d}_B(X,Y) \in [0,1]$.

Remark 5. With this choice of distance, the regular K-means algorithm is not suitable our problem. There is a kernel K-means algorithm that allow users to define the similarity matrix, but it is not proper in our case because generally the term "center" does not really make sense for Bayesian networks. And like the regular K-means algorithm, the kernel K-means algorithm requires observations in a cluster C_i be close to its center in the projected space. For Bayesian networks, two nodes may not have very high correlation to be in one cluster; on the other hand, the logic is more like "a friend of my friend should also be my friend". With a proper choice of linkage function for agglomerative method, hierarchical clustering can achieve this property.

Our choice of the linkage function

There are various ways to define distance between two clusters. Some popular methods are but not restricted to: single linkage, complete linkage, average linkage, centroid linkage, Ward's method. Among those, the first three methods allow users to define their own dissimilarity matrix, and the clustering result relies largely on the user defined distance. Let two clusters be C_A and C_B, and $D(A, B)$ the distance between C_A and C_B. $D(A, B)$ for the first three methods are defined as follows:

- Single linkage. $D_s(C_A, C_B) = \min\limits_{X \in C_A, Y \in C_B} \mathbf{d}(X, Y)$.
- Complete linkage. $D_c(C_A, C_B) = \max\limits_{X \in C_A, Y \in C_B} \mathbf{d}(X, Y)$.
- Average linkage. $D_a(C_A, C_B) = \frac{1}{|C_A||C_B|} \sum_{X \in C_A} \sum_{Y \in C_B} \mathbf{d}(X, Y)$.

Single linkage is related to the minimum spanning tree (MST), where MST is a collection of edges with the minimum total weight that connects all the nodes. Gower and Ross (1969) showed that the single linkage cluster is obtained by removing edges from the MST from the edge with the largest weight to the smallest. And as Hartigan (1981) pointed out, the K-nearest neighbors is an extension of the single linkage method. Single linkage is famous for its chaining effect, since for single linkage the merging logic is strictly local, clusters tends to have very long chains. For Bayesian networks, single linkage excels in clustering DAGs with disconnected clusters. But when clusters are weakly connected in a sense that there are a few edges connecting different clusters, single linkage will not be able to detect them. In my simulation, when clustering a DAG with disconnected clusters, agglomerative clustering using single linkage always has adjusted rand index (ARI) close to 1. However, when I add a few edges between clusters, single linkage tends to identify a few big clusters or even a unimodal.

Complete linkage is the opposite of the single linkage. According to the definition of complete linkage, it requires any two nodes in a cluster to have quite high absolute value of correlation. When the signal-to-noise ratio is small and the chain in DAG is long, it is almost an impossible requirement, the influence of the root diminishing as information pass along and can hardly reach to the leaf.

Average linkage is our choice of the linkage function. It is in between of complete linkage and single linkage, the merging technique is not strictly local but still it considers local structure, so we can have more averaged sub-clusters (smaller $sd(S_i)$). Compared to single linkage, s_b can be slightly larger for the average linkage but is acceptable.

A modified hierarchical clustering method

A natural question is, how should we choose the number of clusters k? Hartigan (1981) suggests that it is only of interest to examine clusters with at least 5% of the number of nodes p. He called them clusters containing a positive fraction. We will refer to clusters with at least 5% nodes as positive fractions hereafter. We propose a method to automatically choose k based on this suggestion.

Since only clusters with at least $0.05p$ nodes are of interest, there can be at most 20 clusters for a DAG. Let $k_{\max}$ be the maximum number of clusters, we know that $k_{\max} \leq 20$. Let $\mathcal{C}_h$ be the set of clusters formed at the hth step of the agglomerative method, $h = 0, 1, ..., p-1$. Therefore $\mathcal{C}_0 = \{\{X_1\}, \{X_2\}, ..., \{X_p\}\}$ and $\mathcal{C}_{p-1} = \{X_1, ..., X_p\}$. Let k_i be the number of positive fractions of $\mathcal{C}_i$. We choose k using the following equation

$$k = \min(k_{\max}, \max(k_i)), i = 0, ..., p-1. \tag{4.6}$$

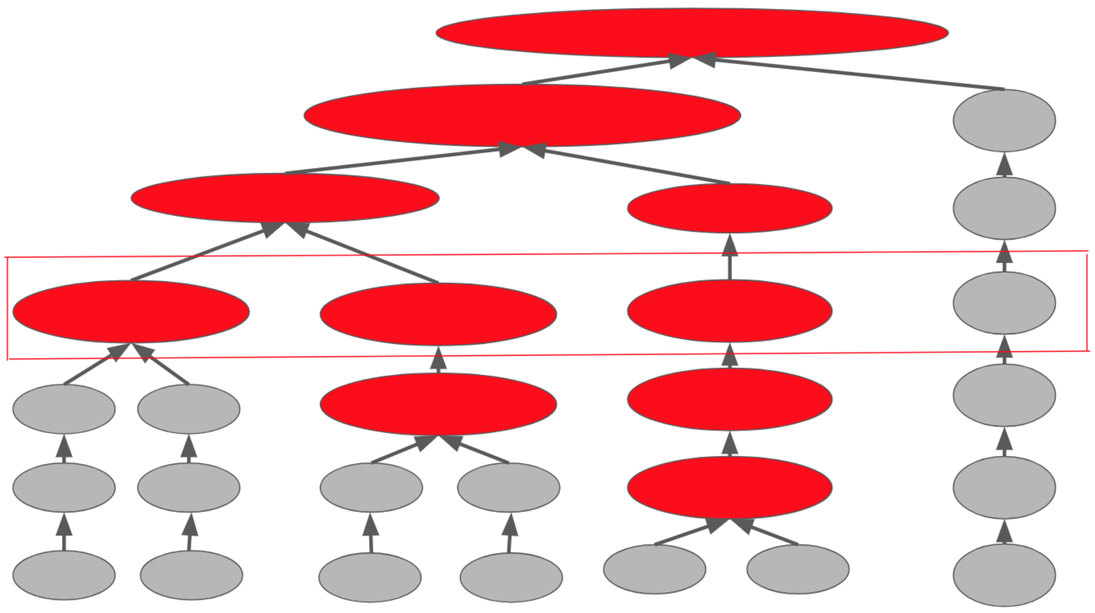


Figure 4.4: Example on choosing k and l_k. This is an upper portion of the clutering tree. Red clusters are clusters with more than $0.05p$ nodes, and the grey ones are small clusters. The level l_k is marked in the red box, and in this case $k = 3$.

Let l_k be the highest level with k positive fractions,

$$l_k = \underset{i=0,\ldots,p-1}{\operatorname{argmax}} \{i : k_i = k\}. \tag{4.7}$$

Figure 4.4 is an example on how we choose k and l_k.

Relabel clusters in $\mathcal{C}_{l_k}$ so that $S_1 \geq S_2 \ldots \geq S_{p-l_k}$. Then the first k clusters are positive fractions, and we assign the rest of the small clusters into the first k clusters in the following way: We will keep merging the closest clusters, but if the closest clusters are two positive fractions, where at least one cluster is not a positive fraction. An outline of our modified algorithm is in Algorithm 3.

In Section 4.4, we will show detailed simulation results on the performance of this clustering method.

Algorithm 3 Modified hierarchical clustering

```
1: Input dissimilarity matrix $D$ to the hierarchical clustering algorithm.
2: Get the output tree structure $T_D$.
3: Choose $k$ by Equation (4.6), $l_k$ by Equation (4.7), and set $\mathcal{C} = \mathcal{C}_{l_k}$ .
4: Relabel clusters in $\mathcal{C}$ so that $S_1 > S_2 > ... > S_{p-l_k}$.
5: while $|\mathcal{C}| > k$ do
6:     $i, j = \underset{i,j: i<j \text{ and } (i>k \text{ or } j>k)}{\operatorname{argmin}} D_s(C_i, C_j)$.
7:     Merge $C_j$ into $C_i$.
8: end while
9: Return $\mathcal{C} = \{C_1, C_2, ..., C_k\}$
```

Note: in Line 6, the distance we use is single linkage, this is mainly for the speed purpose.

4.3.2 Estimation

In the E-step we learn the structure of each sub-network individually. In this step we can use any structure learning algorithm to estimate the sub-networks. We choose the CCDr algorithm (Aragam and Zhou, 2015) in the **R** package **sparsebn** (Aragam et al, 2017b) as an example in this chapter. There are two main reasons: 1) the CCDr algorithm has competitive performance in accuracy for structure learning of large networks with high-dimensional data, 2) The way it is formulated and coded makes it much faster compared to many benchmark algorithms like the MMHC algorithm and the PC algorithm. Figure 4.5 is a plot to compare the running time of the CCDr algorithm, the PC algorithm and the MMHC algorithm, this simulation is done by Aragam et al (2017b), for more details please refer to their paper. According to their data, when $p = 1090, n = 50$, the CCDr algorithm is 17 times faster than the PC algorithm and 52 times faster than the MMHC algorithm.

Note that since running time of most structure learning methods grows on the order of $\omega(p)$, the total running time of learning the small networks in E-step will be much shorter than estimating the full DAG as a whole. Furthermore, if we have multiple cores, we can distribute the estimation step. Suppose in the partition step we have divided the full DAG $\mathcal{G}$ into k sub-networks $\mathcal{G}_1, \ldots \mathcal{G}_k$, and the running time for $\mathcal{G}_i$ is t_i. Running k sub-networks

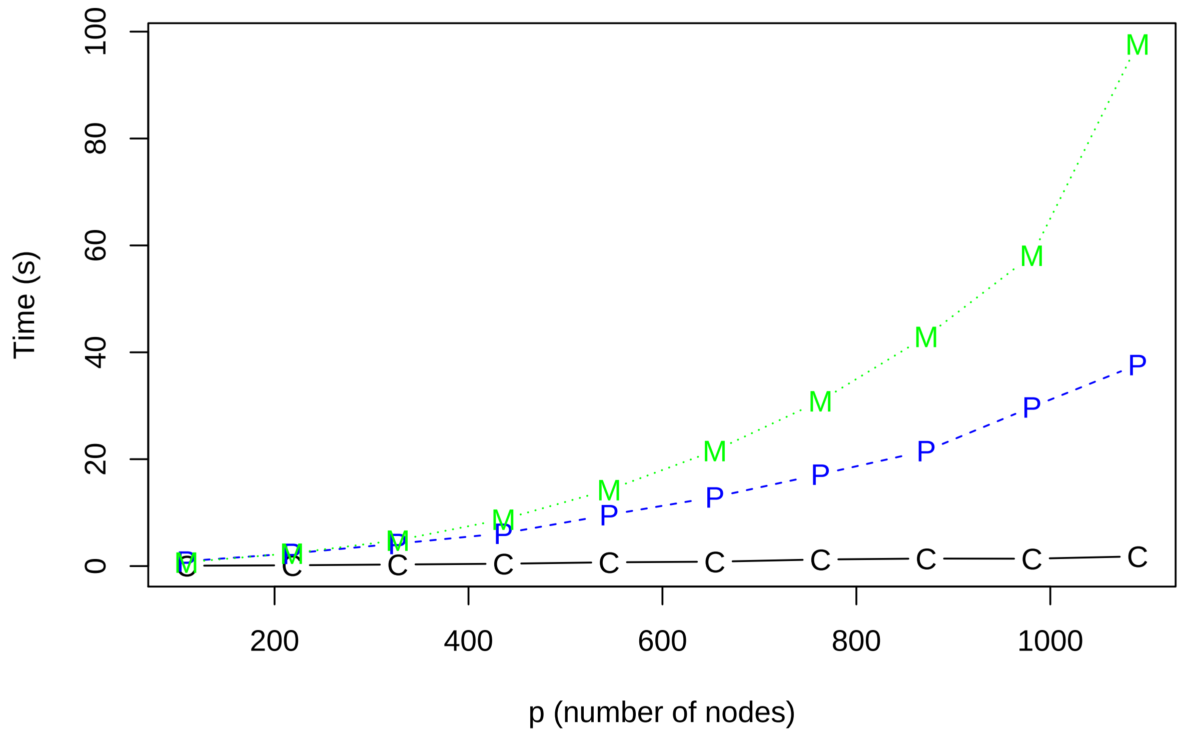


Figure 4.5: Time comparison of the CCDr algorithm, the MMHC algorithm and the PC algorithm. C is for the CCDr algorithm, P is for the PC algorithm, M is for the MMHC algorithm.

on k cores simultaneously will reduce time for the E-step to $\max(t_i), i = 1, \ldots, k$. And the running time of the E-step will be dominated by the largest sub-network. According to our discussion in the previous subsection, if we only consider clusters with at least 5% of the nodes, there will be at most 20 sub-networks, and computing resource with 20 cores is reasonable nowadays. Therefore the E-step is where we can save the majority of time.

A review for the CCDr algorithm

The CCDr algorithm is a score-based algorithm for observational continuous data. It uses the linear structural equation model in (4.1), and their method seek to find the optimizer of the following problem,

$$f_\lambda(B,\Sigma) \triangleq -\ell(B,\Sigma) + n\sum_{i=1}^{p}\sum_{j=1}^{p}\rho_\lambda(|\beta_{ij}|), \tag{4.8}$$

$$\hat{B},\hat{\Sigma} = \underset{\mathcal{G}_B \text{ is a DAG}}{\operatorname{argmin}} f_\lambda(B,\Sigma) \tag{4.9}$$

where the penalty term ρ_λ is chosen as the minimax concave penalty (MCP) (Zhang et al, 2010).

Similar to our discrete CD algorithm, the CCDr algorithm output a DAG structure. It uses a block-wise coordinate descent algorithm to optimize the scoring function, an outline of the CCDr algorithm is in Algorithm 4.

Algorithm 4 CCDr algorithm

```
1: Input data matrix $\mathcal{X}$, initialize $B$ such that $\mathcal{G}_B$ is acyclic.
2: for all pairs $(i,j)$, where $i \neq j$ do
3:     if $\beta_{ij} \Leftarrow 0$ then
4:         update $\beta_{ji}$.
5:     else if $\beta_{ji} \Leftarrow 0$ then
6:         update $\beta_{ij}$.
7:     else
8:         update $\beta_{ij}$ and $\beta_{ji}$ that leads to a smaller (4.8).
9:     end if
10: end for
11: Repeat 2 to 10 until some stopping criterion.
```

4.3.3 Fusion

In this Fusion-step, we propose a method to fuse sub-networks from the second step. First, we use a sequence of conditional independence tests to restrict the search space, and then we use a score-based method to learn the DAG structure.

Conditional independence test

For Gaussian data, we can test conditional independence based on partial correlation (Baba et al, 2004). Let $\boldsymbol{Z}$ be a vector of random variables $(Z_1, ..., Z_n)$. Suppose $(X, Y, \boldsymbol{Z})$ follows a multivariate Gaussian distribution. Then X and Y are conditional independent given $\boldsymbol{Z}$ if the partial correlation between X and Y given $\boldsymbol{Z}$ is 0.

The partial correlation between X and Y given $\boldsymbol{Z}$, denoted by $\rho_{XY\cdot\boldsymbol{Z}}$, can be obtained by calculating the correlation of residuals after projecting X and Y onto the space spanned by $\boldsymbol{Z}$. Let R_X and R_Y be the residuals from linear regression of X onto $\boldsymbol{Z}$ and Y onto $\boldsymbol{Z}$,

$$\rho_{XY\cdot\boldsymbol{Z}} = corr(R_X, R_Y). \tag{4.10}$$

To simplify the calculation, we do not need to calculate the residuals, but use precision matrix instead. Let Σ be the covariance matrix of $(X, Y, \boldsymbol{Z})$, and $\Omega = (\omega_{ij})_{(n+2)\times(n+2)} = \Sigma^{-1}$ be the precision matrix. Then the partial correlation can be written as

$$\rho_{XY\cdot\boldsymbol{Z}} = -\frac{\omega_{12}}{\sqrt{\omega_{11}\omega_{22}}}, \tag{4.11}$$

and we have the following equivalence,

$$\mathcal{I}_P(X; Y|\boldsymbol{Z}) \qquad \Longleftrightarrow \qquad \rho_{XY\cdot Z} = 0. \tag{4.12}$$

Restrict the search space

Given the local Markov properties, we know that a node i is independent of all its non-descendants given its parent set $\Pi_i^{\mathcal{G}}$. So, for two nodes i and j where $i \prec j$ in an ordering $\sqsubset$ compatible with $\mathcal{G}$, then i is not a descendant of j, and therefore X_i and X_j should be independent given $\Pi_j^{\mathcal{G}}$. Without knowing the topological ordering, if X_i and X_j are non-adjacent then they are independent given the union of their parent sets $\Pi_i^{\mathcal{G}} \cup \Pi_j^{\mathcal{G}}$. In addition, recall that Theorem 3 provides us a method to decide if an edge exists using conditional independence tests. We can arrive at the following corollary:

Corollary 8. *If $(\mathcal{G}, P)$ satisfies the faithfulness condition (Definition 8), there is no edge between i and j if $\mathcal{I}_P(X_i; X_j|\Pi_i^{\mathcal{G}} \cup \Pi_j^{\mathcal{G}})$.*

In light of Corollary 8, we propose a way to produce a set of candidate edges (A) between the sub-networks obtained in the E-step. Let $\mathcal{G}_1, ..., \mathcal{G}_k$ be the k disconnected sub-networks estimated from the E-step, V_i the node set for $\mathcal{G}_i$, and Z_i the cluster label of node i. Suppose $i \in V_{Z_i}$ and $j \in V_{Z_j}$, where $Z_i \neq Z_j$ and $Z_i, Z_j = 1, ..., k$, be two nodes from two distinct sub-networks, it is sufficient to show that there is no edge between i, j if they are conditional independent given the union of their parent sets:

$$\mathcal{I}_P(X_i; X_j | \Pi_i^{\mathcal{G}_{Z_i}} \cup \Pi_j^{\mathcal{G}_{Z_j}}) \qquad \Longrightarrow \qquad (i,j) \notin A. \tag{4.13}$$

In this way, we can tune the search space based on the sub-networks we got from the E-step. Note that if we check the conditional independence between i, j for all possible pairs between sub-networks, the number of precision matrices we need to calculate is on the order of $\mathcal{O}(p^2)$. To save some calculation, we approximate $\rho_{ij \cdot \Pi_i^{\mathcal{G}_{Z_i}} \cup \Pi_j^{\mathcal{G}_{Z_j}}}$ by $\tilde{\rho}_{ij} = corr(\tilde{R}_i, \tilde{R}_j)$, where $\tilde{R}_i$ is the residual from projecting X_i onto its parents $\Pi_i^{\mathcal{G}_{Z_i}}$ in $\mathcal{G}_{Z_i}$. Consequently, we can get an approximated active set $\tilde{A} = \{(i,j) : |\tilde{\rho}|_{ij}$ is significantly greater than $0\}$. After getting $\tilde{A}$ we can further tune the search space to get rid of more false positives. Algorithm 5 Line 8-Line 15 describes a recursive way to obtain a more refined candidate edge set A between sub-networks. We propose to go through each node pair (i,j) in $\tilde{A}$ and run the conditional independence test given the union of all their possible parents, P_{ij} and finally get an active set A. In tth iteration, $P_{ij}^{(t)}$ is defined as follow,

$$P_{ij}^{(t)} = \{k : (k,i) \text{ or } (k,j) \in A^{(t)}\}. \tag{4.14}$$

Note that node pair (i,j) does not imply direction, and $(p,i) \in A$ means (p,i) or $(i,p) \in A$. The method to tune the search space is given in Algorithm 5.

Remark 6. The way we add edges to active set A is greedy and will be affected by the order we go through $\tilde{A}$. In our implementation we sort the node pairs in $\tilde{A}$ so that we can first go through node pairs with smaller p-value in test if $\tilde{\rho}_{ij}$ is 0. In this way, edges that we have more confidence in have higher priority. Similarly, after getting A, we also sort it according to the p-value obtained in the test for (4.13).

Algorithm 5 Tune the search space

```
1: Input data matrix $\mathcal{X}$ and $\mathcal{G}_1, ..., \mathcal{G}_k$. Set $\tilde{A} = \emptyset$.
2: for all pairs $i \in V_{Z_i}, j \in V_{Z_j}$, where $Z_i \neq Z_j$ and $Z_i, Z_j = 1, ...k$ do
3:     if $|\tilde{\rho}_{ij}|$ is not significantly greater than 0 then
4:         add $(i, j)$ to $\tilde{A}$.
5:     end if
6: end for
7: Set $A^{(0)} = \emptyset$.
8: for all $(i, j)_t \in \tilde{A}$, $t = 1, ..., T$ do
9:     Let $\boldsymbol{Z} = \Pi_i^{\mathcal{G}_{Z_i}} \cup \Pi_j^{\mathcal{G}_{Z_j}} \cup P_{ij}^{(t-1)}$, where $P_{ij}^{t-1}$ is defined in (4.14)
10:    if not $\mathcal{I}_P(X_i; X_j|\boldsymbol{Z})$ then
11:        $A^{(t)} = A^{(t-1)} \cup (i, j)_t$.
12:    else
13:        $A^{(t)} = A^{(t-1)}$.
14:    end if
15: end for
16: Return $A = A^{(T)}$.
```

RIC for model selection

For every candidate edge in A, we calculate log-likelihood to decide the direction and use regularization to enforce sparsity. In low dimensional setting BIC is a popular method, and it is defined in (4.15)

$$\text{BIC} = -2\ell + \pi\lambda \tag{4.15}$$

where π is the number of parameters, $\lambda = \log(n)$ and n is the number of observations. In the high-dimensional setting, BIC defined in (4.15) will introduce too many false positive edges, so instead we use RIC (Foster and George, 1994) where $\lambda = 2\log(p)$ to achieve enough penalty.

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

$$\begin{cases} M_0: & \text{no edge between } i \text{ and } j \\ M_1: & i \text{ is parent of } j \\ M_2: & j \text{ is parent of } i. \end{cases} \tag{4.16}$$

Consider two linear models. Let $\boldsymbol{X}_Z$ be the vector of variables for $Z \subset V$ and $\boldsymbol{\beta}_{Zi}$ be the vector of coefficients for Z onto i. So we have

$$X_i = \beta_{ji} X_j + \boldsymbol{\beta}^T_{\{\Pi_i^{\mathcal{G}} \setminus j\}i} \boldsymbol{X}_{\{\Pi_i^{\mathcal{G}} \setminus j\}} + \epsilon_i \tag{4.17}$$

$$X_j = \beta_{ij} X_i + \boldsymbol{\beta}^T_{\{\Pi_j^{\mathcal{G}} \setminus i\}j} \boldsymbol{X}_{\{\Pi_j^{\mathcal{G}} \setminus i\}} + \epsilon_j \tag{4.18}$$

M_0 is equivalent to $\beta_{ij} = \beta_{ji} = 0$, M_1 equivalent to $\beta_{ij} \neq 0$ and $\beta_{ji} = 0$, M_2 equivalent to $\beta_{ij} = 0$ and $\beta_{ji} \neq 0$. Note that for either M_1 or M_2 (there exist an edge between i, j), if we run the linear regression, both β_{ij} and β_{ji} will be significant. Let RIC(M) be the RIC score for model M, we only add an edge between i, j if

$$\max(\text{RIC}(M_1), \text{RIC}(M_2)) < \text{RIC}(M_0). \tag{4.19}$$

We can enforce acyclicity similar to the CD algorithm and the CCDr algorithm.

Example in Figure 4.3 shows that partition the full DAG into sub-networks might not only introduce false negative but also false positives in the estimation step. In Figure 4.3 (b), without the hub node N_1 all the disconnected nodes in $\mathcal{G}_{C_2}$ form a clique. By cutting off some of the edges in the P-step, we have changed the structure of sub-networks, and therefore structure learning algorithm in the E-step might not recover the true graph structure. To fix this problem, after we go through the active set A we will revisit all edges learned from the E-step and correct the DAG structure based on new edges added between sub-netwroks. Define $\mathcal{G}$ be the DAG consisting of disconnected sub-DAGs learned from the E-step and $SK(\mathcal{G})$ the undirected edge set of $\mathcal{G}$. We propose the following method to add edges between sub-networks and fix sub-DAGs learned from the E-step: After we have obtained candidate edge set A between sub-networks, we attache all edges in $\mathcal{G}$ to the end of A. For all node pairs (i,j) in A, we first perform a conditional independence test $\mathcal{I}_P(X_i, X_j | \Pi_i^{\mathcal{G}} \cup \Pi_j^{\mathcal{G}})$ to further tune the search space. Then, we check the RIC of M_1, M_2, and M_0, for those node pairs satisfying Equation 4.19 we use the heuristic for our discrete CD algorithm in Algorithm 1 to enforce acyclicity. If an edge from i to j induces a cycle, we will add an edge $j \to i$, and if none of the direction induces a cycle we choose the model with a smaller RIC. The full algorithm for the fusion step is shown in Algorithm 6.

This final fusion step is a hybrid method where we use conditional independence tests to restrict the search space and RIC to learn the DAG structure. The framework proposed in Algorithm 6 shares similar logic with the CD algorithm in Chapter 2 and uses the same heuristic to enforce acyclicity. A major difference here is that the CD algorithm searches in the parameter space while in the fusion step we search in the DAG space.

4.4 Applications to Real Networks

In this section, we test our PEF method on simulated Gaussian observational data sets from real networks and compare its performance with the CCDr algorithm. All network structures are downloaded from repository of the **R** package **bnlearn**. The networks used in this section is: PATHFINDER, ANDES, DIABETES, PIGS, LINK, and the full network of 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 massive networks with clustered structure we replicate the networks for k times and randomly add some edges between clusters. To simplify our future references, let Net $\in$ { PATHFINDER, ANDES, DIABETES, PIGS, LINK, MUNIN}, define Net(k, α) as the network fused by k replicates of Net, and with αs_{sub} edges added between sub-networks, $\alpha \geq 0$ is some constant. And finally let Net(k) refer to the collection of networks fused by k replicates of Net with some edges added between clusters. We have three network generation schemes:

i. Net $\in$ {PATHFINDER, ANDES, DIABETES, PIGS, LINK}: For the first four types of networks, we replicate Net for $k = 5$ times, and get a network with 5 identical disconnected subgraphs. Define s_w to be the number of within cluster edges, $s_w = 5s_{\text{sub}}$. Then, we randomly add $s_b = \alpha s_w$ edges between sub-networks, where $\alpha =$

$0, 0.01, 0.02, 0.05, 0.1$. The network LINK is very large, and we only test it for the case where there is no between cluster edges, $\alpha = 0$. So for Net = LINK, only LINK$(5, 0)$ is generated. The number of true edges for each network is $s_0 = s_w + s_b$. In total, there are $4 \times 5 + 1 = 21$ networks generated for this scheme.

ii. Mixed network: We fuse networks PATHFINDER, ANDES, DIABETES, PIGS, LINK to get a DAG with $k = 5$ different subgraphs. Similar to scheme (i.), we randomly add $s_b = \alpha s_w$ edges between clusters, $\alpha = 0, 0.01, 0.02, 0.05, 0.1$. In total, there are 5 networks generated for this scheme. We will refer to these networks as Mix$(5, \alpha)$ for it is constructed with 5 sub-networks.

iii. MUNIN: It is the largest network available on the **bnlearn** repository, we replicate the network for $k = (1, 2, 3, 4, 5, 6, 7, 8, 9)$ times without adding any edges between sub-networks. So the number of true edges for each DAG is $s_0 = s_w = ks_{\text{sub}}$. In total, 9 networks are generated by this scheme.

Data generation

Each data set was generated according to the linear structural equation model in (4.2). We sampled β_{ij} uniformly from $[-1, -0.5] \cup [0.5, 1]$ if $(i, j) \in E$ and set $\beta_{ij} = 0$ otherwise. The error variance $\sigma_i^2, i = 1, ..., p$ were chosen so that all data columns had the same standard deviation.

The number of observations for all simulated data sets were set to be $n = 1000$. For all networks generated in scheme (i.) and (ii.), we generated 10 data sets. And for each network generated by scheme (iii.), we generated one data set. We use them to test the limit of the CCDr algorithm, so only one data set was generated for each DAG.

4.4.1 Comparison with the CCDr algorithm

Since in our implementation, the algorithm we use for the E-step is the CCDr algorithm, we show in this section the comparison of performance between the CCDr algorithm and the

PEF method on massive networks. We will first show the improvement in speed of our PEF framework over the CCDr algorithm on the full DAGs. Then we will show that the PEF framework actually improves the accuracy of the CCDr algorithm. For all the experiments, we ran the CCDr algorithm provided in the **R** package **sparsebn** with default setting. Like our discrete CD algorithm, the CCDr algorithm outputs a solution path, and we always choose the DAG where the number of predicted edges is the closest to the number of edges in the true graph.

Timing comparison

Table 4.1 and Table 4.2 show the improvement in time of our PEF method over the CCDr algorithm. Table 4.1 reports how the two methods scale when the size of the sub-networks increases, networks tested are $\mathrm{Net}(5,0)$ where $\mathrm{Net} \in \{$ PATHFINDER, ANDES, DIABETES, LINK, MUNIN$\}$. While Table 4.2 reports how the two methods scale when the number of sub-networks increases, networks tested are $\mathrm{MUNIN}(k,0)$ where $k = 1, ..., 8$. Both tables report the total running time of the CCDr algorithm and the PEF method (T) as well as the running time for each step of the PEF method (P, E, F). Figure 4.6 and Figure 4.7 are plots for the $\log_{10}$ of total time reported in the tables. For the second E-step in our PEF method, we report the time if we parallel the estimation of multiple sub-networks. For the running time without parallelization, please refer to the supplemental material. Finally, Figure 4.8 is a plot of $\log_{10}\left(\frac{T_C}{T_P}\right)$ versus p for all networks generated, where T_C is the running time for the CCDr algorithm and T_P is the running time for our PEF method.

From Table 4.1 we see that when the number of sub-networks stayed the same and the size of the sub-networks became larger, the running time of the PEF method increased monotonically. The scalability of the second E-step depends on the CCDr algorithm. Therefore we expect the running time of the second E-step of our PEF method to increase as the size of the sub-networks increases on the same scale of the CCDr algorithm. As we can see for $\mathrm{MUNIN}(5,0)$, the PEF method is 26 times faster than the CCDr algorithm.

In Table 4.2, we can see that after $k = 4$ the running time of the PEF method did not

Table 4.1: Timing comparison in munites for networks with 5 identical sub-networks.

		CCDr	PEF			
Network	p	T	T	P	E	F
PATHFINDER	545	1.13	0.17	0.01	0.04	0.12
ANDES	1115	5.26	0.35	0.02	0.09	0.24
DIABETES	2065	38.32	1.08	0.06	0.39	0.63
LINK	3620	86.22	3.17	0.17	1.19	1.81
MUNIN	5205	210.27	7.97	0.37	3.91	3.69

Note: p is the number of nodes, T is the total running time, P is the running time for the P-step, E is the running time for the E-step, F is the running time for the F-step.

increase much. The number of groups our PEF method identified ($\hat{k}$) is shown in the last column of Table 4.2. We see that the PEF method identified the correct number of sub-networks for $k \geq 4$, and therefore the running time of the second E-step stayed comparable to the running time for a single MUNIN network with $k = 1$. And since we did not add any between cluster edges, the running time of the Fusion-step did not increase much. This example shows the advantage of our PEF method when the full DAG is constructed with multiple sub-networks. From the timing data in Table 4.2, we can see that when $k = 4$ the PEF method is 11 times faster than the PEF method, and when the number of sub-networks increases to $k = 8$, the PEF method is 94 times faster than the CCDr algorithm. When k increases to 9, the CCDr algorithm took more than 24 hours to run and we stopped it before it finished, so there is no data for the CCDr algorithm when $k \geq 9$. Our PEF method, on the other hand, took only 9.25 minutes to run MUNIN(9, 0).

In Figure 4.8, we can see that as p increased, the log-ratio of running time of the two methods increased almost linearly, which means our PEF method is about 10^{cp} times faster than the CCDr algorithm, where c is some positive constant. And for the largest network, our PEF method reached two orders of magnitude improvement.

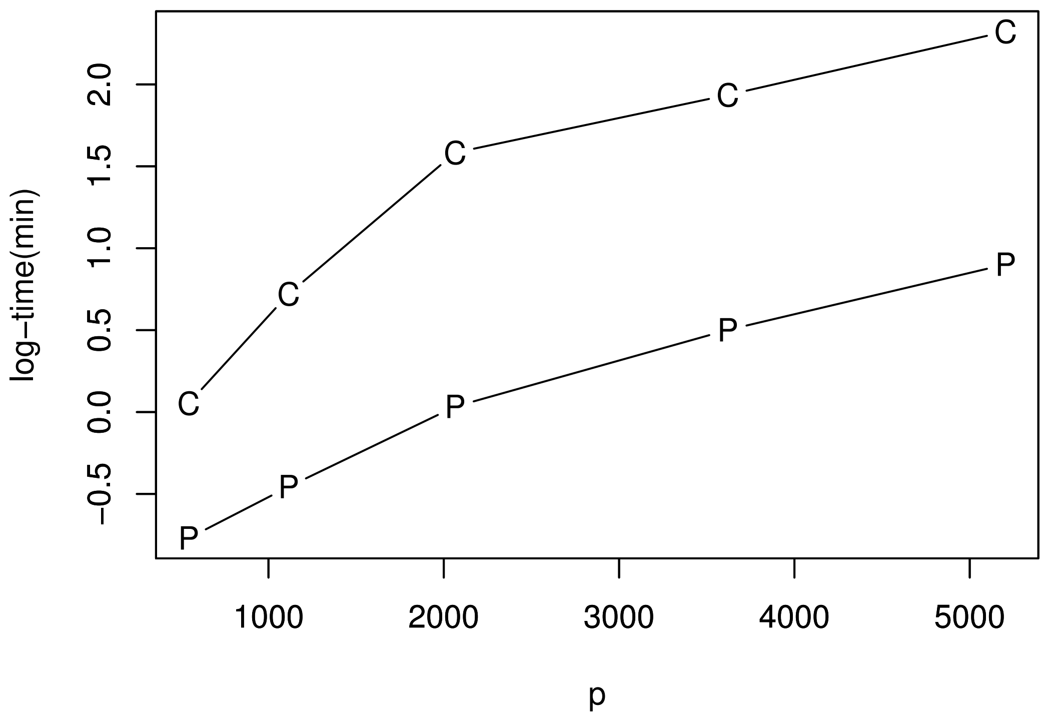


Figure 4.6: $\log_{10}$ time for networks with 5 identical sub-networks. The line with -C- is for the CCDr algorithm, and the line with -P- is for the PEF method. From left to right, networks are formed with 5 replicates of PATHFINDER ($p = 109$), ANDES ($p = 223$), DIABETES ($p = 413$), LINK ($p = 724$), MUNIN ($p = 1041$).

Accuracy comparison

In this section, we show the comparison of accuracy between our PEF method and the CCDr algorithm in Table 4.3 - Table 4.5. In each table, we report the summary of accuracy for both structure learning methods. Since we were using pure observational data, metrics for accuracy are defined the same way as in Chapter 2, Section 2.4.3 for discrete observational data, taking v-structure and compelled edges into account. We can see that, for all cases, the SHD of the PEF method is much smaller than the CCDr algorithm, and the JI is higher

Table 4.2: Timing comparison in minutes for network with increasing number of sub-netwroks (MUNIN), .

		CCDr	PEF				
k	p	T	T	P	E	F	$\hat{k}$
1	1041	3.26	1.23	0.02	1.01	0.20	6
2	2082	29.97	4.16	0.07	2.81	1.28	3
3	3123	94.98	5.63	0.17	3.43	2.03	4
4	4164	109.57	9.70	0.24	4.15	5.31	4
5	5205	210.27	7.97	0.37	3.91	3.69	5
6	6246	330.03	8.34	0.56	3.23	4.55	6
7	7287	587.18	10.95	0.75	3.11	7.09	7
8	8328	793.62	8.42	0.99	3.63	3.80	8

Note: k is the number of sub-networks, and other metrics are the same with Table 4.1.

than the CCDr algorithm.

For PATHFINDER(5) the number of observations ($n = 1000$) is greater than the number of nodes ($p = 545$), the advantage of our PEF method is not as obvious as the rest of the big networks. The number of expected edges of our PEF method is comparable to the CCDr algorithm, but the reversed edges and the false positive edges are much fewer than the CCDr algorithm. Overall, the SHD of the estimated DAGs by our PEF method is over 15% lower and the JI is over 15% higher than the CCDr algorithm.

For all other networks where $p > n$, the number of expected edges of our PEF method increased more than 15% compared to the CCDr algorithm in most of the cases, while the reversed edges and the false positives decreased more than 35% . The overall metric SHD decreased more than 20% and the JI increased over 30% for all cases.

The accuracy of the PEF method was not affected much by the number of edges added between clusters. This is because when we fuse all the sub-networks we also correct the sub-DAG structures obtained in the E-step. Therefore, even if we cut off some edges in the

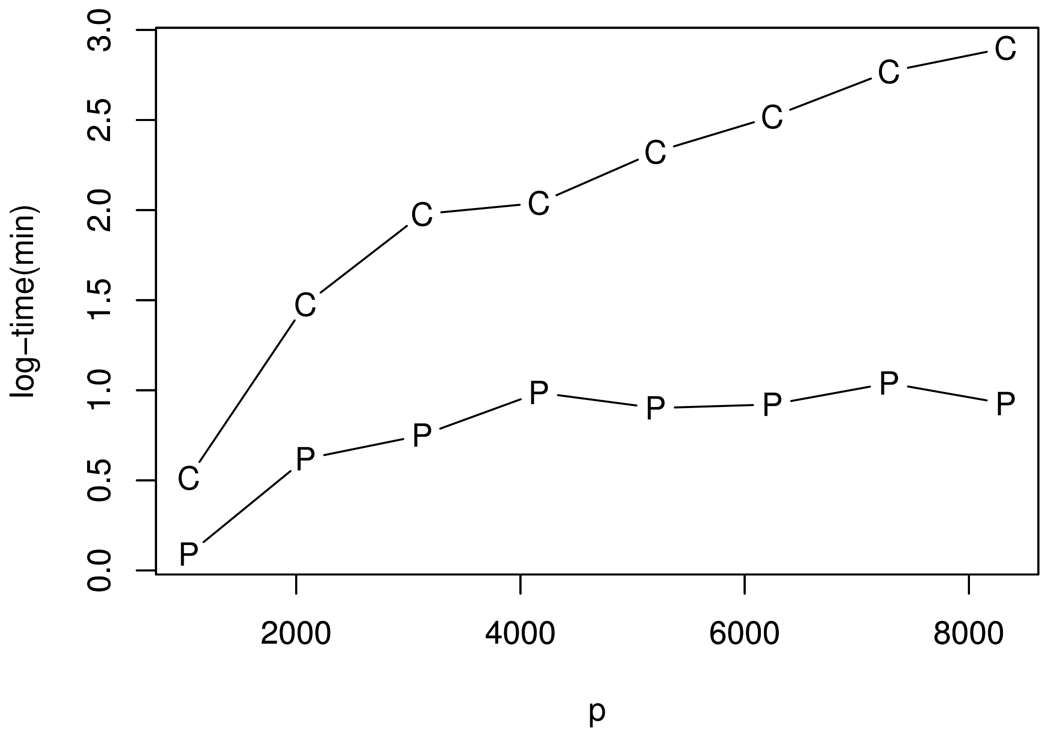


Figure 4.7: $\log_{10}$ time for networks with increasing number of sub-networks. The line with -C- is for the CCDr algorithm, and the line with -P- is for the PEF method. From left to right, each network is k replicates of MUNINs, where $k = 1, 2, 3, 4, 5, 6, 7, 8$.

P-step, which may alter the sub-DAG structures, we can still correct sub-networks in the F-step 4.3.3. Our PEF method has some tolerance for the errors in the partition step, so even if the full DAG does not have a clear cluster structure and many edges may be cut in the P-step, our PEF method can still recover a reasonable amount of these edges. We will show in the next section how our clustering algorithm works with more detailed data.

Note that performance of the PEF method is related to the structure learning algorithm plugged-in in the second E-step. In the final F-step, we can only remove within-subgraph edges, so the missing edges within sub-networks introduced in the E-step will never be added in the fusion step. In addition, the learned subgraph structure may affect our choice

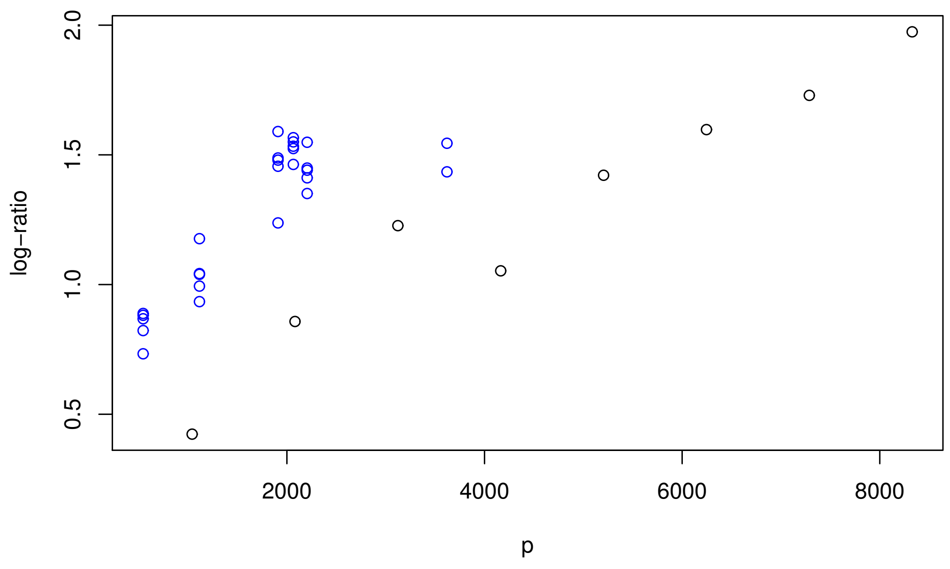


Figure 4.8: $\log_{10}\left(\frac{T_C}{T_P}\right)$ for all networks. Dots in blue are for networks generated by scheme (i.) and (ii.); Dots in black are for networks generated by scheme (iii.)

for the active set A and the final accuracy. Finally, we also tried to compare the timing and accuracy performance with some other benchmark structure learning algorithms like the MMHC algorithm and the PC algorithm. However, for both algorithms, even with the smallest network PATHFINDER$(5, 0)$ where $n = 1000$ no algorithm finished running within a day and I had to stop the algorithms before they output anything.

4.4.2 Performance of the clustering step

Table 4.6 reports the number of estimated clusters $\hat{k}$, the number of edges cut off in the first P-step $\hat{s}_b$ and the size of the largest sub-networks ($S_{\max}$) as well as the smallest sub-networks ($S_{\min}$). We focus on the behavior of the clustering step on different networks. All networks studied in this section are constructed with 5 sub-networks.

Recall the three goals for the clustering algorithm: 1) large $\hat{k}$, 2) small $\hat{s}_b$, 3) small $\max(S_i)$, $i = 1, ..., k$. In our implementation, we set $k_{\max} = 20$ which is the maximum number of clusters we can get when the positive fraction is defined to have at least 5% nodes of V. And since for all networks we fuse 5 sub-networks, the PEF method is expected to find at least 5 clusters when the number of between cluster edges added is small. Among all the networks in Table 4.6, only for PATHFINDER(5) our partition step detected 5 clusters for all s_b, and the sizes of the sub-networks were the most even. The sub-network PATHFINDER looks more like the star shaped example in Figure 4.2 (b). It has a hub node directly connected to most of the other nodes. Therefore, a clustering algorithm is unlikely to partition PATHFINDER into multiple sub-networks. As for ANDES(5), the partition step identified the most number of clusters. It appears that the graph of ANDES has two to three blocks.

Having larger $\hat{k}$ means the sub-networks are smaller and the more time we can save in the second E-step. On the other hand, a larger $\hat{k}$ also means more edges are cut off and the structure learned in the E-step could be less accurate. When adding more edges between sub-networks, $\hat{s}_b$ became larger: when $s_b = 0.1s_w$ the number of edges cut off ($\hat{s}_b$) is around twice the number of edges added between clusters (s_b), that is $0.2s_0$.

To sum up, our partition step does not aim to find the correct clusters, but to find a reasonable way to partition the nodes to speed up our algorithm without losing much accuracy. And our partition Algorithm 3 works fine for this purpose.

Remark 7. We did not use adjusted rand index (ARI) as a metric to assess the partition step for the following reasons: 1) We are not sure how many blocks are there in each Net $\in$ {PATHFINDER, ANDES, DIABETES, PIGS, LINK, MUNIN}. And after adding between sub-network edges, there is no clear block structure. 2) As long as the number of edges cut in the clustering step is much smaller than the number of edges within sub-networks, we consider the clustering step to work well. 3) We would prefer to partition a cluster into smaller clusters as long as it can improve the speed and we are able to recover majority of the cut-off edges in the last F-step.

4.4.3 Recovery rate of the fusion step

In this section, we compare the final estimated DAG obtained from our full PEF method with the DAG estimated from the first two steps, the partition step and the estimation step, called the PE method. Table 4.7-4.9 report the comparison of the PEF method and the PE method. The row in method PE is the summary for this two-step method. The row in method PEF reports the percentage increase for each metric. We can see from the simulation results that the fusion step always improves the structure of an estimated DAG with increased E, JI and decreased R, FP, SHD.

Table 4.6 shows that the number of edges cut-off ($\hat{s_b}$) increased as we added more between cluster edges (s_b). Correspondingly, Table 4.7-4.9 show that the number of edges predicted by the PE method decreased as s_b changed from 0 to $0.1s_w$. Therefore the number of expected edges from the first two steps decreased as s_0 of the full DAG increased. Consequently, the number of edges we expect to recover in the fusion step increased. From the simulation results we can see that as s_b increased, the fusion step indeed recovered increasing number of expected edges. The number of expected edges recovered in the fusion step can reach 60% of the number of expected edges recovered in the first two steps. This demonstrates the critical role of the fusion step.

In addition, we see that our fusion Algorithm 6 can not only recover expected edges, it is also able to remove reversed and false positive edges. The percentage decrease in reversed edges stays roughly the same when we increase s_b while the percentage decrease in false positives becomes smaller. For all cases, the F-step reduces 15% to 40% FPs and 15% to 55% Rs, which substantially improves the structure learning accuracy.

All these improvements in accuracy demonstrate that our fusion Algorithm 6 works well on the simulation data. Since it may recover more expected edges when we cut off more edges in the partition step, our fusion step is quite flexible with the output of the first two steps and it may handle networks with a moderate number of between subgraph edges.

4.4.4 Influence of the number of clusters k

In all our previous experiments we did not set any limit on the number of clusters we might get, $k_{\max}$. In this subsection we will set $k_{\max} = 5$ and this setting will force the partition step to partition the full DAG into at most 5 sub-networks. In this way, we will in general obtain fewer clusters in this subsection's results, and we can check how the number of clusters affects the performance of our PEF method.

The summaries of accuracy for the PEF method are reported in Table 4.10-Table 4.11. Note that we exclude results for PATHFINDER(5) because the number of clusters identified when $k_{\max} = 20$ is the same when we set $k_{\max} = 5$. Comparing simulation results in this section with Table 4.7-4.9 in Section 4.4.3, the accuracy after the first two steps tends to be higher if we limit $k_{\max} = 5$, especially for ANDES(5) and Mix(5). This is because when we limit the number of clusters, less edges will be cut off in the partition step, which will lead to a better accuracy in the first two steps. Meanwhile, by comparing our simulation results with Table 4.3-4.5 in Section 4.4.1, we can see that the accuracy of our PEF method is comparable between the two choices of $k_{\max}$.

The summary of running time when we set $k_{\max} = 5$ is reported in Table 4.12. Compared to the results in Table 4.13-4.15, we can see that the PEF method is faster when $k_{\max} = 20$. The E-step is faster when there are more sub-networks, as the largest sub-network tends to be smaller.

From the simulation results in this section, we see that the number of sub-networks $\hat{k}$ chosen in the partition step does not affect the accuracy of our PEF method, and having more sub-networks can in fact speed up the PEF method. This result again shows the usefulness of the fusion step, and it also suggests that our clustering algorithm does partition the full DAG into a reasonable number of sub-networks.

4.5 Discussions

In this chapter we developed a three step divide-and-conquer framework for massive networks constructed with some weakly connected sub-networks. We focused on the high-dimensional observational Gaussian data, and implemented the PEF method in **R** and **Rcpp**. Our simulation results suggest that our modified hierarchical clustering can partition the DAG into some reasonable size subgraphs without cutting off too many edges. In addition, our fusion step can correct and fix the DAG structure damaged by partitioning the full network, so the overall accuracy of the PEF method is comparable or even better than the structure learning algorithm used in the estimation step. Finally, our PEF method runs all the simulated data set within 10 minutes while the CCDr algorithm run more than 24 hours for large network.

There are some possible generalizations for this PEF framework. The algorithm we currently plugged in produces a DAG. There are some other efficient structure learning algorithms that produce a CPDAG like the PC algorithm and the GES algorithm (Chickering, 2002). We describe in the remaining of this chapter our proposal to modify the fusion step when the network returned in the E-step is a CPDAG, or a skeleton.

E-step returns k CPDAGs

In the fusion step when we first tune the search space, we regress X_i onto its parent set $\Pi_i^{\mathcal{G}}$, where $\Pi_i^{\mathcal{G}}$ is well defined for a DAG. For algorithms that return a CPDAG like the PC algorithm, we can define neighbors of a node i as $N_i = \{j : j \to i \in E \text{ or } (i, j) \in E\}$, where E is the edge set of the CPDAG output from the first two steps, $i \to j$ is a directed edge, and (i, j) means an undirected edge between i, j. For this case, we can regress X_i onto N_i.

After we get A, Algorithm 6 go through each node pair $(i, j) \in A$ to decide if an edge exists and also the direction. Recall that in order to enforce sparsity, the direction of an edge partially depends on the current DAG structure, therefore if the output of the estimation step is a CPDAG we can treat neighbors as parents in RIC calculation in Algorithm 6. So we can still estimate a DAG if the estimation step outputs CPDAGs. In addition, we may revise our fusion step to also output a CPDAG, but that will require more investigation.

E-step returns k skeletons

If it is the skeleton that we get from the second E-step, when we tune the search space we can regress each node i onto all its neighbors $\text{nb}(i) = \{x : (x, i) \in E\}$, that is all its possible parent set.

Let E_i be the undirected edge set for all subgraphs from the second step, and define $+$ the operator to concatenate two sets: $\{a_1, ..., a_l\} + \{b_1 ..., b_h\} = \{a_1, ..., a_l, b_1, ..., b_h\}$. Then after we get the active set A, we can combine all $E_i, i = 1, ..., k$ and A by appending all E_i ahead of A: $A = E_1 + ... + E_k + A$ so that when we go though the active set all the undirected edges found within clusters from our estimation step have priority.

All these generalization will be in our future work so that this PEF framework can work with more structure learning algorithms. Also, 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 the dissimilarity matrix, and when the number of nodes becomes really large, the machine might run out of memory. To solve this problem we might use subsample clustering (Marchetti and Zhou, 2016). 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 and the machine runs out of memory to store the correlation matrix, we can implement the algorithm by taking as input the data matrix and calculate correlations whenever needed.

4.6 Supplemental Materials

We report timing data for each network in Table 4.13-4.15, the row for PEF is obtained assuming we have enough cores and paralleling the E-step. And the PEF* row is for the case when we run the E-step sequentially.

Algorithm 6 Fuse all sub-networks

```
1: Input data matrix $\mathcal{X}$ and $\mathcal{G}_1, ..., \mathcal{G}_k$. Run Algorithm 5 to get active edge set $A$.
2: Initialize $\mathcal{G}$ be the DAG fused by $k$ distinct sub-networks $\mathcal{G}_k$.
3: Attache $(i, j) \in SK(\mathcal{G})$ to the end of $A$.
4: for all $(i, j) \in A$ do
5:     if $i, j$ are connected in $\mathcal{G}$ then
6:         Remove the edge from $\mathcal{G}$.
7:     end if
8:     if $\mathcal{I}_P(X_i; X_j | \Pi_i^{\mathcal{G}} \cup \Pi_j^{\mathcal{G}})$ then
9:         Remove $(i, j)$ from $A$.
10:    else
11:        Calculate $\mathrm{RIC}_{\max} = \max(\mathrm{RIC}(M_1), \mathrm{RIC}(M_2))$.
12:        if $\mathrm{RIC}_{\max} < \mathrm{RIC}(M_0)$ then
13:            if Adding edge $i \to j$ induces a cycle then
14:                $\beta_{ij} \Leftarrow 0$, add $j \to i$ to $\mathcal{G}$.
15:            else if Adding edge $j \to i$ induces a cycle then
16:                $\beta_{ji} \Leftarrow 0$, add $i \to j$ to $\mathcal{G}$.
17:            else
18:                Choose the direction that leads to a smaller RIC.
19:            end if
20:        end if
21:    end if
22: end for
23: Repeat 4 to 22 until structure of $\mathcal{G}$ does not change and return $\mathcal{G}$.
```

Table 4.3: Comparison between the CCDr algorithm and the PEF method

PATHFINDER(5), $p = 545$

(s_0, s_b)	Method	P	E	R	FP	SHD	JI
$(975, 0)$	CCDr	948.8	269.6	170.8	508.4	1213.8	0.164
	PEF	717.4	275.6	136.0	305.8	1005.2	0.196
$(985, 10)$	CCDr	963.5	303.5	163.3	496.7	1178.2	0.186
	PEF	734.1	302.0	135.2	296.9	979.9	0.215
$(995, 20)$	CCDr	972.8	285.8	161.9	525.1	1234.3	0.171
	PEF	728.8	288.4	132.1	308.3	1014.9	0.203
$(1024, 49)$	CCDr	1030.2	365.0	161.8	503.4	1162.4	0.216
	PEF	766.6	364.2	128.2	274.2	934.0	0.256
$(1073, 98)$	CCDr	1090.3	376.5	169.1	544.7	1241.2	0.211
	PEF	797.7	374.7	126.7	296.3	994.6	0.251

ANDES(5), $p = 1115$

(s_0, s_b)	Method	P	E	R	FP	SHD	JI
$(1690, 0)$	CCDr	1586.0	931.4	447.0	207.6	966.2	0.397
	PEF	1517.8	1164.5	222.9	130.4	655.9	0.570
$(1707, 17)$	CCDr	1616.1	961.8	433.4	220.9	966.1	0.408
	PEF	1542.3	1193.7	213.5	135.1	648.4	0.582
$(1724, 34)$	CCDr	1615.5	973.9	434.3	207.3	957.4	0.412
	PEF	1561.7	1210.8	212.9	138.0	651.2	0.584
$(1775, 85)$	CCDr	1709.8	1002.9	455.0	251.9	1024.0	0.404
	PEF	1635.6	1253.4	218.8	163.4	685.0	0.582
$(1859, 169)$	CCDr	1721.8	1051.6	452.1	218.1	1025.5	0.416
	PEF	1693.4	1349.8	192.0	151.6	660.8	0.613

Table 4.4: Comparison between the CCDr algorithm and the PEF method

DIABETES(5), $p = 2065$							
(s_0, s_b)	Method	P	E	R	FP	SHD	JI
$(3010, 0)$	CCDr	3166.3	1327.3	1067.9	771.1	2453.8	0.274
	PEF	2677.5	1530.2	759.0	388.3	1868.1	0.368
$(3041, 31)$	CCDr	3210.8	1347.8	1074.7	788.3	2481.5	0.275
	PEF	2712.9	1545.2	761.5	406.2	1902.0	0.367
$(3071, 61)$	CCDr	3235.8	1369.9	1071.1	794.8	2495.9	0.278
	PEF	2755.2	1595.9	746.8	412.5	1887.6	0.377
$(3161, 151)$	CCDr	3353.6	1470.4	1052.5	830.7	2521.3	0.292
	PEF	2837.8	1694.4	715.9	427.5	1894.1	0.394
$(3311, 301)$	CCDr	3354.7	1560.1	1033.2	761.4	2512.3	0.306
	PEF	3004.0	1885.7	681.5	436.8	1862.1	0.426

PIGS(5), $p = 2205$							
(s_0, s_b)	Method	P	E	R	FP	SHD	JI
$(2960, 0)$	CCDr	2990.2	1609.9	802.2	578.1	1928.2	0.371
	PEF	2635.6	1825.2	536.6	273.8	1408.6	0.484
$(2990, 30)$	CCDr	3029.9	1623.9	810.8	595.2	1961.3	0.369
	PEF	2696.7	1866.0	536.7	294.0	1418.0	0.488
$(3020, 60)$	CCDr	3050.5	1643.4	818.6	588.5	1965.1	0.371
	PEF	2719.2	1891.9	533.2	294.1	1422.2	0.492
$(3108, 148)$	CCDr	3139.3	1741.2	789.7	608.4	1975.2	0.386
	PEF	2812.0	2010.8	497.2	304.0	1401.2	0.515
$(3256, 296)$	CCDr	3262.5	1874.0	800.8	587.7	1969.7	0.404
	PEF	2996.1	2183.2	483.2	329.7	1402.5	0.537

Table 4.5: Comparison between the CCDr algorithm and the PEF method

Mix(5), $p = 1910$							
(s_0, s_b)	Method	P	E	R	FP	SHD	JI
$(2852, 0)$	CCDr	2855.4	1415.2	760.3	679.9	2116.7	0.330
	PEF	2523.7	1630.0	518.9	374.8	1596.8	0.436
$(2881, 29)$	CCDr	2803.2	1393.8	746.5	662.9	2150.1	0.325
	PEF	2537.9	1633.9	498.9	405.1	1652.2	0.432
$(2910, 58)$	CCDr	2913.5	1427.4	773.4	712.7	2195.3	0.325
	PEF	2579.5	1673.3	495.3	410.9	1647.6	0.439
$(2995, 143)$	CCDr	2964.0	1510.4	750.2	703.4	2188.0	0.340
	PEF	2684.5	1784.4	475.4	424.7	1635.3	0.458
$(3138, 286)$	CCDr	3058.8	1595.7	785.6	677.5	2219.8	0.347
	PEF	2851.5	1933.6	487.0	430.9	1635.3	0.477

Table 4.6: Partition step summary

Network	(s_0, s_b)	$\hat{k}$	$\hat{s}_b$	$S_{\max}$	$S_{\min}$
PATHFINDER(5)	$(975, 0)$	5.0	3.4	110.1	108.2
	$(985, 10)$	5.0	19.5	111.8	106.7
	$(995, 20)$	5.0	33.7	111.4	106.6
	$(1024, 49)$	5.0	78.9	113.8	104.1
	$(1073, 98)$	5.0	157.2	116.6	102.1
ANDES(5)	$(1690, 0)$	9.5	135.8	178.0	75.5
	$(1707, 17)$	9.2	155.3	196.6	62.9
	$(1724, 34)$	9.3	190.1	193.2	70.0
	$(1775, 85)$	9.1	275.5	200.3	67.7
	$(1859, 169)$	8.4	370.9	216.2	65.6
DIABETES(5)	$(3010, 0)$	8.1	113.9	418.9	120.3
	$(3041, 31)$	8.2	186.9	416.9	124.4
	$(3071, 61)$	7.9	230.6	420.8	123.2
	$(3161, 151)$	7.7	382.7	449.5	116.6
	$(3311, 301)$	8.2	621.2	420.6	124.1
PIGS(5)	$(2960, 0)$	5.8	84.6	472.8	242.3
	$(2990, 30)$	5.7	133.4	489.8	235.1
	$(3020, 60)$	5.5	173.9	585.3	228.4
	$(3108, 148)$	5.4	307.2	497.0	288.6
	$(3256, 296)$	6.2	518.7	496.3	222.2
Mix(5)	$(2852, 0)$	8.6	195.9	391.5	115.0
	$(2881, 29)$	8.4	258.8	455.6	113.7
	$(2910, 58)$	7.9	288.3	461.3	123.9
	$(2995, 143)$	7.7	428.9	453.4	119.8
	$(3138, 286)$	7.2	550.4	598.5	118.7

Table 4.7: Comparison between the PEF method and the PE method

PATHFINDER(5), $p = 545$

(s_0, s_b)	Method	P	E	R	FP	SHD	JI
$(975, 0)$	PE	962.4	270.0	171.9	520.5	1225.5	0.163
	PEF(%)	-25	2	-21	-41	-18	20
$(985, 10)$	PE	966.2	296.1	164.2	505.9	1194.8	0.180
	PEF(%)	-24	2	-18	-41	-18	19
$(995, 20)$	PE	958.9	269.4	163.1	526.4	1252.0	0.161
	PEF(%)	-24-	7	-19	-41	-19	26
$(1024, 49)$	PE	947.1	317.2	155.1	474.8	1181.6	0.192
	PEF(%)	-19	15	-17	-42	-21	33
$(1073, 98)$	PE	915.6	286.8	149.7	479.1	1265.3	0.169
	PEF(%)	-13	31	-15	-38	-21	49

ANDES(5), $p = 1115$

(s_0, s_b)	Method	P	E	R	FP	SHD	JI
$(1690, 0)$	PE	1522.5	858.1	444.0	220.4	1052.3	0.365
	PEF(%)	0	36	-50	-41	-38	56
$(1707, 17)$	PE	1530.8	876.6	428.4	225.8	1056.2	0.372
	PEF(%)	1	36	-50	-40	-39	56
$(1724, 34)$	PE	1519.4	857.4	434.7	227.3	1093.9	0.359
	PEF(%)	3	41	-51	-39	-40	63
$(1775, 85)$	PE	1485.8	827.9	433.9	224.0	1171.1	0.340
	PEF(%)	10	51	-50	-27	-42	71
$(1859, 169)$	PE	1468.4	822.0	429.9	216.5	1253.5	0.328
	PEF(%)	15	64	-55	-30	-47	87

Table 4.8: Comparison between the PEF method and the PE method

DIABETES(5), $p = 2065$							
(s_0, s_b)	Method	P	E	R	FP	SHD	JI
$(3010, 0)$	PE	2919.1	1243.1	1008.1	667.9	2434.8	0.265
	PEF(%)	-8	23	-25	-42	-23	39
$(3041, 31)$	PE	2838.7	1213.3	996.1	629.3	2457.0	0.260
	PEF(%)	-4	27	-24	-35	-23	41
$(3071, 61)$	PE	2848.9	1221.4	986.5	641.0	2490.6	0.260
	PEF(%)	-3	31	-24	-36	-24	45
$(3161, 151)$	PE	2726.5	1206.2	936.1	584.2	2539.0	0.258
	PEF(%)	4	40	-24	-27	-25	53
$(3311, 301)$	PE	2596.2	1172.7	911.2	512.3	2650.6	0.248
	PEF(%)	16	61	-25	-15	-30	72
PIGS(5), $p = 2205$							
(s_0, s_b)	Method	P	E	R	FP	SHD	JI
$(2960, 0)$	PE	2909.1	1544.0	794.3	570.8	1986.8	0.357
	PEF(%)	-9	18	-32	-52	-29	36
$(2990, 30)$	PE	2904.8	1522.8	803.3	578.7	2045.9	0.348
	PEF(%)	-7	23	-33	-49	-31	40
$(3020, 60)$	PE	2878.5	1507.7	806.6	564.2	2076.5	0.343
	PEF(%)	-6	25	-34	-48	-32	43
$(3108, 148)$	PE	2836.6	1509.1	764.6	562.9	2161.8	0.340
	PEF(%)	-1	33	-35	-46	-35	51
$(3256, 296)$	PE	2783.0	1477.0	769.3	536.7	2315.7	0.324
	PEF(%)	8	48	-37	-39	-39	66

Table 4.9: Comparison between the PEF method and the PE method

			Mix(5), $p = 1910$				
(s_0, s_b)	Method	P	E	R	FP	SHD	JI
$(2852, 0)$	PE	2629.6	1288.3	734.1	607.2	2170.9	0.307
	PEF(%)	-4	27	-29	-38	-26	42
$(2881, 29)$	PE	2581.5	1238.5	717.7	625.3	2267.8	0.293
	PEF(%)	-2	32	-30	-35	-27	47
$(2910, 58)$	PE	2583.9	1238.5	726.1	619.3	2290.8	0.291
	PEF(%)	0	35	-32	-34	-28	51
$(2995, 143)$	PE	2545.8	1242.5	700.6	602.7	2355.2	0.289
	PEF(%)	5	44	-32	-30	-31	58
$(3138, 286)$	PE	2601.7	1263.4	735.2	603.1	2477.7	0.282
	PEF(%)	10	53	-34	-29	-34	69

Table 4.10: Accuracy of the PEF method and the PE step when limiting $k_{\max} = 5$.

ANDES(5), $p = 1115$

(s_0, s_b)	Method	P	E	R	FP	SHD	JI
$(1690, 0)$	PE	1586.7	909.1	450.0	227.6	1008.5	0.384
	PEF	1468.7	1135.0	226.5	107.2	662.2	0.561
$(1707, 17)$	PE	1616.6	931.9	436.6	248.1	1023.2	0.390
	PEF	1510.0	1172.5	217.4	120.1	654.6	0.574
$(1724, 34)$	PE	1575.6	912.1	434.9	228.6	1040.5	0.382
	PEF	1525.1	1184.8	218.0	122.3	661.5	0.574
$(1775, 85)$	PE	1538.1	872.4	439.2	226.5	1129.1	0.358
	PEF	1591.7	1233.9	216.2	141.6	682.7	0.579
$(1859, 169)$	PE	1511.1	867.3	429.8	214.0	1205.7	0.347
	PEF	1662.5	1327.7	191.8	143.0	674.3	0.606

DIABETES(5), $p = 2065$

(s_0, s_b)	Method	P	E	R	FP	SHD	JI
$(3010, 0)$	PE	3033.6	1286.7	1037.2	709.7	2433.0	0.271
	PEF	2680.6	1531.3	766.7	382.6	1861.3	0.368
$(3041, 31)$	PE	2956.7	1262.0	1020.5	674.2	2453.2	0.267
	PEF	2708.8	1540.7	765.4	402.7	1903.0	0.366
$(3071, 61)$	PE	2943.5	1260.2	1008.5	674.8	2485.6	0.265
	PEF	2744.6	1598.4	745.6	400.6	1873.2	0.379
$(3161, 151)$	PE	2840.2	1245.0	964.4	630.8	2546.8	0.262
	PEF	2837.1	1701.4	717.4	418.3	1877.9	0.396
$(3311, 301)$	PE	2696.0	1230.3	921.4	544.3	2625.0	0.258
	PEF	2979.4	1861.6	689.0	428.8	1878.2	0.420

Table 4.11: Accuracy of the PEF method and the PE step when limiting $k_{\max} = 5$.

PIGS(5), $p = 2205$

(s_0, s_b)	Method	P	E	R	FP	SHD	JI
$(2960, 0)$	PE	2923.9	1557.9	794.0	572.0	1974.1	0.360
	PEF	2628.7	1826.3	536.8	265.6	1399.3	0.486
$(2990, 30)$	PE	2915.7	1536.1	801.8	577.8	2031.7	0.352
	PEF	2686.9	1866.0	534.8	286.1	1410.1	0.490
$(3020, 60)$	PE	2889.9	1516.1	807.3	566.5	2070.4	0.345
	PEF	2719.7	1890.1	534.7	294.9	1424.8	0.491
$(3108, 148)$	PE	2848.2	1516.0	765.6	566.6	2158.6	0.341
	PEF	2812.4	2005.9	500.7	305.8	1407.9	0.513
$(3256, 296)$	PE	2813.4	1511.6	766.0	535.8	2280.2	0.332
	PEF	2991.3	2178.5	485.9	326.9	1404.4	0.536

Mix(5), $p = 1910$

(s_0, s_b)	Method	P	E	R	FP	SHD	JI
$(2852, 0)$	PE	2712.5	1342.6	754.4	615.5	2124.9	0.318
	PEF	2462.8	1604.4	517.7	340.7	1588.3	0.433
$(2881, 29)$	PE	2684.6	1305.7	729.5	649.4	2224.7	0.307
	PEF	2509.2	1622.1	493.7	393.4	1652.3	0.431
$(2910, 58)$	PE	2676.2	1291.7	741.4	643.1	2261.4	0.301
	PEF	2539.8	1658.2	492.6	389.0	1640.8	0.438
$(2995, 143)$	PE	2647.3	1296.6	718.1	632.6	2331.0	0.298
	PEF	2652.8	1760.0	480.6	412.2	1647.2	0.453
$(3138, 286)$	PE	2699.3	1329.5	742.7	627.1	2435.6	0.295
	PEF	2819.7	1917.5	483.7	418.5	1639.0	0.475

Table 4.12: Running time of the PEF method when $k_{\max} = 5$

ANDES(5), $p = 1115$						DIABETES(5), $p = 2065$					
(s_0, s_b)	Method	T	P	E	F	(s_0, s_b)	Method	T	P	E	F
$(1690, 0)$	PEF	0.42	0.01	0.16	0.25	$(3010, 0)$	PEF	1.67	0.04	0.43	1.20
	PEF*	0.90	0.01	0.64	0.25		PEF*	3.10	0.04	1.86	1.20
$(1707, 17)$	PEF	0.42	0.01	0.15	0.26	$(3041, 31)$	PEF	1.39	0.04	0.46	0.89
	PEF*	0.91	0.01	0.64	0.26		PEF*	2.77	0.04	1.84	0.89
$(1724, 34)$	PEF	0.35	0.01	0.16	0.18	$(3071, 61)$	PEF	1.83	0.04	0.43	1.36
	PEF*	0.83	0.01	0.64	0.18		PEF*	3.18	0.04	1.78	1.36
$(1775, 85)$	PEF	0.48	0.01	0.20	0.27	$(3161, 151)$	PEF	2.07	0.04	0.59	1.44
	PEF*	0.95	0.01	0.67	0.27		PEF*	3.35	0.04	1.87	1.44
$(1859, 169)$	PEF	0.71	0.01	0.30	0.40	$(3311, 301)$	PEF	2.52	0.04	0.76	1.72
	PEF*	1.07	0.01	0.66	0.40		PEF*	3.64	0.04	1.88	1.72
PIGS(5), $p = 2205$						Mix(5), $p = 1910$					
(s_0, s_b)	Method	T	P	E	F	(s_0, s_b)	Method	T	P	E	F
$(2960, 0)$	PEF	1.51	0.05	0.51	0.95	$(2852, 0)$	PEF	1.59	0.04	1.09	0.46
	PEF*	3.02	0.05	2.02	0.95		PEF*	2.52	0.04	2.02	0.46
$(2990, 30)$	PEF	1.56	0.05	0.58	0.93	$(2881, 29)$	PEF	1.53	0.04	0.86	0.63
	PEF*	3.03	0.05	2.05	0.93		PEF*	2.65	0.04	1.98	0.63
$(3020, 60)$	PEF	1.78	0.06	1.00	0.72	$(2910, 58)$	PEF	1.81	0.04	0.88	0.89
	PEF*	3.14	0.06	2.36	0.72		PEF*	3.00	0.04	2.07	0.89
$(3108, 148)$	PEF	1.46	0.06	0.67	0.73	$(2995, 143)$	PEF	2.17	0.04	0.96	1.17
	PEF*	2.86	0.06	2.07	0.73		PEF*	3.32	0.04	2.11	1.17
$(3256, 296)$	PEF	1.84	0.06	0.74	1.04	$(3138, 286)$	PEF	3.02	0.04	2.15	0.83
	PEF*	3.25	0.06	2.15	1.04		PEF*	3.94	0.04	3.07	0.83

Table 4.13: Time comparison wetween the CCDr algorithm and the PEF method.

PATHFINDER(5), $p = 545$

(s_0, s_b)	Method	T	Method	T	P	E	F
$(975, 0)$	CCDr	1.13	PEF	0.17	0.01	0.04	0.12
			PEF*	0.31	0.01	0.18	0.12
$(985, 10)$	CCDr	1.37	PEF	0.18	0.01	0.04	0.13
			PEF*	0.32	0.01	0.18	0.13
$(995, 20)$	CCDr	1.19	PEF	0.22	0.01	0.04	0.17
			PEF*	0.36	0.01	0.18	0.17
$(1024, 49)$	CCDr	1.47	PEF	0.19	0.01	0.04	0.14
			PEF*	0.34	0.01	0.19	0.14
$(1073, 98)$	CCDr	1.55	PEF	0.21	0.01	0.04	0.16
			PEF*	0.35	0.01	0.18	0.16

ANDES(5), $p = 1115$

(s_0, s_b)	Method	T	Method	T	P	E	F
$(1690, 0)$	CCDr	5.26	PEF	0.35	0.02	0.09	0.24
			PEF*	0.62	0.02	0.36	0.24
$(1707, 17)$	CCDr	4.27	PEF	0.39	0.02	0.10	0.27
			PEF*	0.67	0.02	0.38	0.27
$(1724, 34)$	CCDr	4.74	PEF	0.43	0.02	0.10	0.31
			PEF*	0.71	0.02	0.38	0.31
$(1775, 85)$	CCDr	5.72	PEF	0.58	0.02	0.10	0.46
			PEF*	0.86	0.02	0.38	0.46
$(1859, 169)$	CCDr	5.07	PEF	0.59	0.02	0.11	0.46
			PEF*	0.87	0.02	0.39	0.46

Table 4.14: Time comparison wetween the CCDr algorithm and the PEF method.

DIABETES(5), $p = 2065$

(s_0, s_b)	Method	T	Method	T	P	E	F
$(3010, 0)$	CCDr	38.32	PEF	1.08	0.06	0.39	0.63
			PEF*	2.21	0.06	1.52	0.63
$(3041, 31)$	CCDr	31.67	PEF	0.86	0.05	0.37	0.44
			PEF*	1.85	0.05	1.36	0.44
$(3071, 61)$	CCDr	34.11	PEF	1.02	0.06	0.38	0.58
			PEF*	2.00	0.06	1.36	0.58
$(3161, 151)$	CCDr	31.66	PEF	1.09	0.05	0.48	0.56
			PEF*	2.00	0.05	1.39	0.56
$(3311, 301)$	CCDr	33.12	PEF	0.97	0.05	0.40	0.52
			PEF*	1.78	0.05	1.21	0.52

PIGS(5), $p = 2205$

(n, p, s_b)	Method	T	Method	T	P	E	F
$(2960, 0)$	CCDr	40.31	PEF	1.14	0.07	0.51	0.56
			PEF*	2.52	0.07	1.89	0.56
$(2990, 30)$	CCDr	36.43	PEF	1.32	0.07	0.54	0.71
			PEF*	2.69	0.07	1.91	0.71
$(3020, 60)$	CCDr	35.45	PEF	1.58	0.07	1.04	0.47
			PEF*	2.92	0.07	2.38	0.47
$(3108, 148)$	CCDr	39.94	PEF	1.42	0.07	0.6	0.75
			PEF*	2.82	0.07	2.00	0.75
$(3256, 296)$	CCDr	37.39	PEF	1.45	0.07	0.59	0.79
			PEF*	2.69	0.07	1.83	0.79

Table 4.15: mix: Time (min) for dags with no between cluster edges.

Mix(5), $p = 1910$							
(s_0, s_b)	Method	T	Method	T	P	E	F
$(2852, 0)$	CCDr	31.12	PEF	0.80	0.06	0.32	0.42
			PEF*	1.60	0.06	1.12	0.42
$(2881, 29)$	CCDr	29.53	PEF	0.96	0.06	0.49	0.41
			PEF*	1.70	0.06	1.23	0.41
$(2910, 58)$	CCDr	27.72	PEF	0.97	0.07	0.50	0.40
			PEF*	1.74	0.07	1.27	0.40
$(2995, 143)$	CCDr	28.37	PEF	0.94	0.06	0.45	0.43
			PEF*	1.79	0.06	1.30	0.43
$(3138, 286)$	CCDr	34.22	PEF	1.98	0.06	1.07	0.85
			PEF*	2.70	0.06	1.79	0.85

CHAPTER 5

Summary and Discussion

In this dissertation, we focused on structure learning of sparse Bayesian networks, and we have developed two methods for this purpose. The first method is a generalization of a previous work by Fu and Zhou (2013) on continuous data to discrete case. The generalization from continuous to discrete was nontrivial for there are more parameters and our score function for the discrete model has no closed-form solution. The coordinate descent algorithm we developed can incorporate interventional and observational data. We have shown in our simulation section the improvement in accuracy after incorporating interventional data. Also, the CD algorithm works well with high-dimensional observational data, both our experiments on simulation data and real data showed that our CD algorithm has a competitive performance compared to several other structure learning algorithms. In addition, we have developed an **R** package **discretecdAlgorithm** for the CD algorithm so other researchers can easily access our algorithm.

The second structure learning method we proposed is a framework for massive size sparse Bayesian networks. It is a divide-and-conquer method that first **partition** the full DAG into several sub-networks, then **estimate** each sub-network individually, and finally **fuse** all sub-networks together by adding edges between clusters. One feature of our PEF framework is that we can plug-in any structure learning algorithm in the second E-step, so the second E-step is like a black box, and the performance of our PEF method largely depends on the second step. Users have the flexibility to choose a proper structure learning algorithm for the E-step based on their experience with their data set. Our current implementation is only for observational Gaussian data, and we plugged in the CCDr algorithm in the second step as an example. Our simulation results showed a significant improvement in speed as well as accuracy compared to the CCDr algorithm.

Future work

There are a lot of generalizations we can implement for the PEF framework. As mentioned in the discussion part of Chapter 4, we can adapt the method so the fusion step can not only accept a sequence of DAGs but also CPDAGs and skeletons. And therefore more structure learning algorithms will be available to be used for the E-step, like the PC algorithm and the MMPC algorithm.

In addition, we can further generalize the method to discrete data. For discrete data, one can still use our clustering method for the partition step, and plug in a structure learning algorithm for discrete data in the estimation step. As for the fusion step, the conditional independence test is no longer testing the partial correlation, instead we can use the G^2 test for discrete data. Finally we can substitute the linear regression with the multinomial logistic regression.

Moreover, we can further adapt the framework to incorporate interventional data. Given a data set with k blocks where for each block a set of nodes are under intervention, certain edges will be cut off due to experimental interventions. A problem is that how can we code this information in the partition step and the fusion step.

For the partition step, we can modify the definition of distance $\mathbf{d}_B(X, Y)$ to only use observational data for both X and Y. That is saying $r_{XY} = corr(X_O, Y_O)$, where O is the index set of selected rows of X and Y, $\forall i \in O, i \notin \mathcal{M}_X, i \notin \mathcal{M}_Y$, and $\mathcal{M}_X$ is the set of data points where variable X is under intervention.

As for the fusion step, when we do the conditional independence test $X \perp Y | \boldsymbol{Z}$, we can also exclude interventional data points for X and Y. Similarly, when we do the regression of $i \sim \Pi_i^{\mathcal{G}}$, we can exclude interventional data points for node i.

Finally, our current implementation of the fusion step is implemented with an **Rcpp** package `Armadillo`, if we can code it in pure C++, we might further improve the speed of the fusion step.

Bibliography

Andersen PK, Gill RD (1982) Cox's regression model for counting processes: a large sample study. The Annals of Statistics 10(4):1100–1120

Aragam B, Zhou Q (2015) Concave penalized estimation of sparse Bayesian networks. Journal of Machine Learning Research 16:2273-2328

Aragam B, Amini AA, Zhou Q (2017a) Learning directed acyclic graphs with penalized neighbourhood regression ArXiv preprint arXiv:1511.08963

Aragam B, Gu J, Zhou Q (2017b) Learning large-scale Bayesian networks with the sparsebn package. Journal of Statistical Software to appear, arXiv preprint arXiv:1703.04025

Baba K, Shibata R, Sibuya M (2004) Partial correlation and conditional correlation as measures of conditional independence. Australian & New Zealand Journal of Statistics 46(4):657–664

Barabási AL, Albert R (1999) Emergence of scaling in random networks. Science 286:509–512

Bates D, Maechler M (2018) Matrix: Sparse and Dense Matrix Classes and Methods. URL `https://CRAN.R-project.org/package=Matrix`, r package version 1.2-14

Bielza C, Li G, Larranaga P (2011) Multi-dimensional classification with Bayesian networks. International Journal of Approximate Reasoning 52(6):705–727

Bouckaert RR (1993) Probabilistic network construction using the minimum description length principle. In: Symbolic and Quantitative Approaches to Reasoning and Uncertainty: European Conference ECSQARU '93, Lecture Notes in Computer Science, vol 747, Springer, pp 41–48

Buntine W (1991) Theory refinement on Bayesian networks. In: Proceedings of the Seventh Annual Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, pp 52–60

Butts CT (2008) network: a package for managing relational data in r. Journal of Statistical Software 24(2), URL `http://www.jstatsoft.org/v24/i02/paper`

Chickering DM (2002) Optimal structure identification with greedy search. Journal of machine learning research 3(Nov):507–554

Chickering DM, Heckerman D (1997) Efficient approximations for the marginal likelihood of Bayesian networks with hidden variables. Machine Learning 29:181–212

Colombo D, Maathuis MH, Kalisch M, Richardson TS (2012) Learning high-dimensional directed acyclic graphs with latent and selection variables. The Annals of Statistics 40(1):294–321

Cooper GF, Herskovits E (1992) A Bayesian method for the induction of probabilistic networks from data. Machine Learning 9:309–347

Cooper GF, Yoo C (1999) Causal discovery from a mixture of experimental and observational data. In: Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence, Morgan Kaufmann Publishers Inc., pp 116–125

Csárdi G, Nepusz T (2006) The igraph software package for complex network research. InterJournal Complex Systems:1695, URL `http://igraph.org`

Eberhardt F (2012) Almost optimal intervention sets for causal discovery. arXiv preprint arXiv:12063250

Eberhardt F, Glymour C, Scheines R (2012) On the number of experiments sufficient and in the worst case necessary to identify all causal relations among n variables. arXiv preprint arXiv:12071389

Ellis B, Wong WH (2008) Learning causal Bayesian network structures from experimental data. Journal of the American Statistical Association 103:778–789

Foster DP, George EI (1994) The risk inflation criterion for multiple regression. The Annals of Statistics 22(4):1947–1975

Friedman J, Hastie T, Höfling H, Tibshirani R (2007) Pathwise coordinate optimization. The Annals of Applied Statistics 1:302–332

Friedman J, Hastie T, Tibshirani R (2010) Regularization paths for generalized linear models via coordinate descent. Journal of Statistical Software 33:1–22

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

Fu W (1998) Penalized regressions: The bridge versus the lasso. Journal of Computational and Graphical Statistics 7:397–416

Gámez JA, Mateo JL, Puerta JM (2011) 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

van de Geer S, Bühlmann P (2013) ℓ_0-penalized maximum likelihood for sparse directed acyclic graphs. The Annals of Statistics 41(2):536–567

Gentleman R, Whalen E, Huber W, Falcon S (2016) graph: A package to handle graph data structures. R package version 1.50.0

Gower JC, Ross GJ (1969) Minimum spanning trees and single linkage cluster analysis. Applied statistics 18(1):54–64

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

Hauser A, Bühlmann P (2012) Characterization and greedy learning of interventional markov equivalence classes of directed acyclic graphs. The Journal of Machine Learning Research 13(1):2409–2464

Hauser A, Bühlmann P (2015) Jointly interventional and observational data: estimation of interventional markov equivalence classes of directed acyclic graphs. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 77(1):291–318

Heckerman D, Geiger D, Chickering DM (1995) Learning Bayesian networks: The combination of knowledge and statistical data. Machine Learning 20:197–243

Herskovits E, Cooper G (1990) Kutató: An entropy-driven system for construction of probabilistic expert systems from databases. In: Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence, pp 54–62

Kalisch M, Bühlmann P (2007) Estimating high-dimensional directed acyclic graphs with the pc-algorithm. The Journal of Machine Learning Research 8:613–636

Kalisch M, Mächler M, Colombo D, Maathuis MH, Bühlmann P (2012) Causal inference using graphical models with the r package pcalg. Journal of Statistical Software 47(11):1–26

Koller D, Friedman N (2009) Probabilistic graphical models: principles and techniques. MIT press

Kou S, Zhou Q, Wong WH (2006) Equi-energy sampler with applications in statistical inference and statistical mechanics (with discussion). The Annals of Statistics 34:1581–1652

Lam W, Bacchus F (1994) Learning Bayesian belief networks: An approach based on the MDL principle. Computational Intelligence 10:269–293

Lee JD, Simchowitz M, Jordan MI, Recht B (2016) Gradient descent only converges to minimizers. vol 49, pp 1–12

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

Meek C (1995) Strong completeness and faithfulness in bayesian networks. In: Proceedings of the Eleventh conference on Uncertainty in artificial intelligence, Morgan Kaufmann Publishers Inc., pp 411–418

Meganck S, Leray P, Manderick B (2006) Learning causal bayesian networks from observations and experiments: A decision theoretic approach. In: Modeling Decisions for Artificial Intelligence, Springer, pp 58–69

Meier L, van de Geer S, Bühlmann P (2008) The group Lasso for logistic regression. Journal of the Royal Statistical Society Series B 70:53–71

Nandy P, Hauser A, Maathuis MH (2018) High-dimensional consistency in score-based and hybrid structure learning. The Annals of Statistics to appear, arXiv preprint arXiv:1507.02608

Niinimäki T, Parviainen P, Koivisto M (2016) Structure discovery in bayesian networks by sampling partial orders. The Journal of Machine Learning Research 17(1):2002–2048

Pearl J (1995) Causal diagrams for empirical research. Biometrika 82(4):669–688

Pearl J (2014) Probabilistic reasoning in intelligent systems: networks of plausible inference. Elsevier

Peér D, Regev A, Elidan G, Friedman N (2001) Inferring subnetworks from perturbed expression profiles. Bioinformatics 17(suppl 1):S215–S224

Perrier E, Imoto S, Miyano S (2008) Finding optimal bayesian network given a superstructure. Journal of Machine Learning Research 9(Oct):2251–2286

Pollard D (1991) Asymptotics for least absolute deviation regression estimators. Econometric Theory 7:186–199

Pournara I, Wernisch L (2004) Reconstruction of gene networks using bayesian learning and manipulation experiments. Bioinformatics 20(17):2934–2942

Renyi A, Erdos P (1959) On random graphs. Publicationes Mathematicae 6(290-297):5

Robinson RW (1977) Counting unlabeled acyclic digraphs. In: Combinatorial mathematics V, Springer, pp 28–43

Russell SJ, Norvig P (2016) Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited

Sachs K, Perez O, Pe'er D, Lauffenburger DA, Nolan GP (2005) Causal protein-signaling networks derived from multiparameter single-cell data. Science 308:523–529

Schmidt M, Niculescu-Mizil A, Murphy K, et al (2007) Learning graphical model structure using l1-regularization paths. In: AAAI, vol 7, pp 1278–1283

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

Scutari M (2016) An empirical-bayes score for discrete bayesian networks. In: Conference on Probabilistic Graphical Models, pp 438–448

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

Shojaie A, Michailidis G (2010) Penalized likelihood methods for estimation of sparse high-dimensional directed acyclic graphs. Biometrika 97(3):519–538

Spirtes P, Glymour C, Scheines R (1993) Causation, Prediction, and Search. Springer-Verlag

Suzuki J (1993) A construction of Bayesian networks from databases based on an MDL principle. In: Proceedings of the Ninth Annual Conference on Uncertainty in Artificial Intelligence, pp 266–273

Tsamardinos I, Brown LE, Aliferis CF (2006) The max-min hill-climbing bayesian network structure learning algorithm. Machine learning 65(1):31–78

Tseng P, Yun S (2009) A coordinate gradient descent method for nonsmooth separable minimization. Mathematical Programming B 117:387–423

Venables WN, Ripley BD (2002) Modern Applied Statistics with S, 4th edn. Springer, New York, URL `http://www.stats.ox.ac.uk/pub/MASS4`, iSBN 0-387-95457-0

Verma T, Pearl J (1990) Equivalence and synthesis of causal models. In: Sixth Annual Conference on Uncertainty in Artificial Intelligence, pp 220–227

Watts DJ, Strogatz SH (1998) Collective dynamics of ‘small-world’ networks. Nature 393:440–442

Wu T, Lange K (2008) Coordinate descent procedures for lasso penalized regression. The Annals of Applied Statistics 2:224–244

Xiang J, Kim S (2013) A* lasso for learning a sparse bayesian network structure for continuous variables. In: Advances in Neural Information Processing Systems, pp 2418–2426

Yuan M, Lin Y (2006) Model selection and estimation in regression with grouped variables. Journal of the Royal Statistical Society Series B 68:49–67

Zhang CH, et al (2010) Nearly unbiased variable selection under minimax concave penalty. The Annals of statistics 38(2):894–942

Zhou Q (2011) Multi-domain sampling with applications to structural inference of Bayesian networks. Journal of the American Statistical Association 106:1317–1330

Zhu J, Hastie T (2004) Classification of gene microarrays by penalized logistic regression. Biostatistics 5:427–443