A Kernel-based Approach for Learning Causal Graphs From Mixed Data Containing Missing Values

Teny Handhayani

Doctor of Philosophy

University of York

Computer Science

March 2021

Dedication

Dedicated to people who work for developing knowledge to make a better civilization.

Abstract

A causal graph can be generated from a dataset using a particular causal algorithm, for instance, the PC algorithm, Fast Causal Inference (FCI) or Really Fast Causal Inference (RFCI). This research provides two contributions for learning causal graphs: an easy way to handle mixed data so that it can be used to learn causal graphs using the PC algorithm/FCI/RFCI and a method to evaluate the learned graph structure when the true graph is unknown. This research proposes using kernel functions and kernel alignment to handle mixed data. The two main steps of this approach are computing a kernel matrix for each variable and calculating a pseudo-correlation matrix using kernel alignment. The kernel alignment matrix is used as a substitute for the correlation matrix, the main component used in computing a partial correlation for the conditional independence test for Gaussian data in the PC algorithm, FCI, and RFCI. The advantage of this idea is it is then possible to handle more data types when there is a suitable kernel function to compute a kernel matrix for an observed variable. The proposed method is successfully applied to learn a causal graph from mixed data containing categorical, binary, ordinal, and continuous variables. We also introduce the Modal Value of Edges Existence (MVEE) method, a new method to evaluate the structure of learned graphs represented by a Partial Ancestral Graph (PAG) when the true graph is unknown. MVEE produces an agreement graph as a proxy to the true graph to evaluate the structure of the learned graph. MVEE is successfully used for choosing the best-learned graph when the true graph is unknown.

Contents

List of Figures

List of Tables

Acknowledgements

This work was supported through a scholarship managed by Lembaga Pengelola Dana Pendidikan Indonesia (Indonesia Endowment Fund for Education).

I wish to extend my special thanks to.

1. Dr. James Cussens, my first supervisor, who always gives me awesome guidance and supports during my PhD.
2. Dr. Simon O'Keefe for the meaningfull discussions and suggestions for my research.
3. Dr. Marco Scutari for the meaningfull discussion and suggestions for my research.
4. Dr. Detlef Plump, my second supervisor, for his support during my PhD.
5. My beloved family who always give me the infinity love and support.
6. Dr. Ruifei Cui and Dr. Michael Tsagris who share their codes for my experiment.
7. Dr. Durdane Kocacoban, Dr. Alfa Yohannis, Dr. Nunung Qomariyah, Dr. Yudistira Permana and his wife, Yumechris Amekan, AI Group Research members, and PPI York members.

Declaration

I declare that the research described in this thesis is original work, which I undertook at the University of York during 2017 - 2021. Except where stated, all of the work contained within this thesis represents the original contribution of the author.

Some parts of this thesis have been published in conference proceedings; where the item was published jointly with collaborators, the author of this thesis is responsible for the material presented here. For each published item the primary author is the first listed author.

- T. Handhayani and J. Cussens. Kernel-based Approach for Learning Causal Graphs from Mixed Data. In *Proceedings of Probabilistic Graphical Model*, September 2020. [35].

Chapter 1

Introduction

1.1 Background and Motivation

Statistical learning can be used to infer dependence among random variables from observational data. Correlation measures a relationship between two variables but a correlation does not imply causation. This means that statistical properties alone do not determine causal structures [56]. Understanding two variables moving together is not enough to know whether one variable causes another.

A distribution function does not show how the distribution will be different if external conditions are changed. It happens because the laws of probability theory do not instruct how one variable should change when another variable is modified [54]. This problem can be solved by causal assumptions which identify relationships that remain the same when external conditions change. Causal learning and analysis help answer the cause-effect questions in some areas, for instance, medical treatment, health, economic, and politics. The studies in inferring causal graphs in health and medical treatments have been done in previous research [36] [38] [55] [68] [77]. Causal inference in economics appears in some publications [13] [73] [81]. Causal inference in political science has been studied by Blackwell [10].

A *causal graph* is a graphical model used to describe the cause-effect relationship between variables. Examples of algorithms for learning causal graphs from a dataset are the PC algorithm [62], Fast Causal Inference (FCI) [62] and Really Fast Causal Inference (RFCI) [20]. The PC, FCI, and RFCI algorithms use conditional independence tests to generate a causal graph from a dataset [62] [39]. A dataset that includes both discrete and continuous variables is called *mixed data*. Testing for conditional independence is more

complex when data is mixed than when it is either entirely discrete or entirely continuous. Conditional independence testing is even more complicated when the dataset is mixed data containing missing values.

The motivation to develop a method for causal learning from mixed data is that it would be useful for real datasets that possibly have different data types. The real datasets are recorded from real events, for instance, patient datasets. A patient's dataset might contain patient's id, age, gender, blood type, blood pressure, heart beat, disease, treatment, time for recovery and other details. Causal discovery from this dataset can be beneficial, for example, discovering the causal relationship between medicine, blood pressure, and recovery rate. In reality, it is possible to encounter the situation when we need to learn causal graphs from mixed data containing missing values. The *missing values* probably happen because of instrument error or human error during data collection. Missing data also creates a problem in causal learning.

After generating a graph from a dataset, it is necessary to evaluate the quality of the learned graph. Evaluating the learned graph is important to ensure that the learned graph has a certain quality for further use, i.e., data analysis, decision support system, etc. Evaluating the learned graph is easy when there is a true graph; for instance, one needs only to compare the structure of the true graph and the learned graph then analyze the mismatch. In a situation where the true graph is unknown, it is difficult to evaluate the quality of the learned graph. The *unknown true graph* means that there is a true graph but we do not know its identity. Developing a method to create a proxy of the true graph is useful to measure the quality of the learned graph.

Some applications were designed for learning causal graphs from mixed data. *deal* is an R package for learning the Bayesian networks from discrete and continuous variables restricted to conditionally Gaussian networks [12] [11]. It outputs a DAG and the limitation is this method works only for datasets without missing values and no latent variables. *Latent variables* are variables that are not measured or recorded [20]. *Selection variables* are unmeasured variables that determine whether or not a measured unit is included in the data sample [20]. *MXM* is an R package for conditional independence test for mixed data [71] [43]. MXM was successfully implemented to learn a Bayesian network using the PC algorithm [70] and in this paper, we call it the *PC MXM*. MXM is equipped with a simple method for missing values handling; if there are missing values in the dataset column wise imputation takes place. MXM applies median and mode for the imputation

procedure for continuous and categorical variables, respectively. It is a naive and not so clever method, so the user is encouraged to make sure their data contain no missing values. *Greedy search Hill-Climbing (HC)* is a score-based learning algorithm in R package *bnlearn* to explore the space of the directed acyclic graphs [60] [51] [50]. The algorithm explore the search space starting from the empty graph and adding, deleting, or reserving one arc at a time until the score can no longer be improved [51].

1.2 Research Question

The issues addressed in this research are learning causal graphs from mixed data containing missing values and evaluating the learned graph when the true graph is unknown. In this study, there are two research questions: (i) how to handle mixed data containing missing values in learning causal graphs (ii) how to evaluate the learned graph structure when the true graph is unknown. We propose kernel functions and kernel alignment to handle mixed data in a way that allows existing algorithms (e.g. PC, FCI, RFCI) to be used. In this research, we use the PC, FCI, and RFCI implementation from *pcalg* [40]. The second goal is evaluating the structure of a learned graph when the ground truth is unknown. This research introduces a new method called Modal Value of Edges Existence (MVEE) to evaluate different learned PAGs (Partial Ancestral Graphs). This research implements kernel functions to compute kernel matrices where we have a choice of kernel parameters. Different kernel parameter choices can produce different learned graphs. This research uses the MVEE method to choose between these various learned graphs, since in real applications we do not have the ground truth to help us make this choice.

1.3 Research Contributions

The contributions of this research are:

- Kernel Alignment PC (KAPC), Kernel Alignment FCI (KAFCI), and Kernel Alignment RFCI (KARFCI) are methods for learning causal graphs from mixed data containing missing values. Those methods implement kernel functions and kernel alignment to produce a kernel alignment matrix from a dataset. The kernel alignment matrix is used as a substitute correlation matrix for conditional independence tests in the PC, FCI, and RFCI.

- Modal Value of Edge Existence (MVEE) is a method to evaluate the structure of learned graphs represented by PAG when the true graph is unknown. This method is useful for choosing the best learned graph when the true graph is unknown.

1.4 Thesis Structure

This thesis contains several chapters:

Chapter 1 Introduction: this chapter contains the background and motivation, research questions, research contributions, and thesis structure.

Chapter 2 Literature Review: this chapter contains the literature review related to the probabilistic graphical model, causal algorithm, mixed data, and missing values data.

Chapter 3 Learning Causal Graphs from Mixed Data: this chapter describes related works in causal learning from mixed data in the presence of latent variables, the detailed explanation of kernel function and kernel alignment to handle mixed data for causal learning using the PC, FCI, and RFCI, generating mixed datasets, experimental results and discussion.

Chapter 4 Learning Causal Graphs from Mixed Data Containing Missing Values: this chapter describes related works in missing values handling, kernel extension to handle mixed data containing missing values and latent variables for causal learning, generating mixed data containing missing values, experimental results and discussion.

Chapter 5 Evaluating Graph Structure When The True Graph is Unknown: this chapter describes related work in measuring graph structure when the true graph is unknown, a new method Modal Value of Edge Existence (MVEE) to evaluate the graph structure represented by Partial Ancestral Graphs (PAGs) when the true graph is unknown, experimental result and discussion.

Chapter 6 Evaluation and Conclusion: this chapter describes the evaluation of the proposed methods, conclusion, and future work.

Chapter 2

Literature Review

2.1 Probabilistic Graphical Models

2.1.1 Probability Theory

The word *probability* refers to a degree of confidence that an event of an uncertain nature will occur. An example of probability is a weather forecast, for example, "there is a high probability of heavy snow in the afternoon". A *probability distribution* P over (Ω, S) is a mapping from events in S to real values that satisfies [42]:

- $P(\alpha) \geq 0$ for all $\alpha \in S$
- $P(\Omega) = 1$
- If $\alpha, \beta \in S$ and $\beta = \emptyset$ then $P(\alpha \cup \beta) = P(\alpha) + P(\beta)$

The probability values lie between 0 and 1 [42]. $P(\alpha) = 1$ means that there is certainty one of the outcomes in α occurs. $P(\alpha) = 0$ means that all of the outcomes are impossible. Probabilities can be viewed as subjective degrees of belief. $P(\alpha) = 0.3$ represents one's own degree of belief that the event α will occur.

The probability of two events happening together can be explained by *conditional probability*. Let the event α be "student with the jacket size M" and the event β be "all female students". We can consider the probability of $\alpha \cap \beta$ (set of female students who wear jacket size M). The *conditional probability* of β given α is defined as $P(\beta|\alpha) = \frac{P(\alpha \cap \beta)}{P(\alpha)}$ [42]. The conditional probability is not defined when $P(\alpha) = 0$.

The chain rule of conditional probabilities is defined by $P(\alpha \cap \beta) = P(\alpha)P(\beta|\alpha)$. If $\alpha_1, ..., \alpha_k$ are events then it can be written as $P(\alpha_1 \cap ... \cap \alpha_k) = P(\alpha_1)P(\alpha_2|\alpha_1)...P(\alpha_k \cap$

$\alpha_1 \cap ... = \cap\alpha_{k-1}$) [42]. The Bayes rule is a consequence of the definition of conditional probability [42]:

$$P(\alpha|\beta) = \frac{P(\beta|\alpha)P(\alpha)}{P(\beta)} \tag{2.1}$$

A *random variable* is defined by a function that associates with each outcome in Ω a value. For example, *Size* is defined by a function f_{size} that maps each person in Ω his or her jacket size (one of H, M, L). The event *Size* = M refers to the event $\{\omega \in \Omega : f_{size}(\omega) = M\}$. The random variable for the student's gender takes as values either "female" or "male". Random variables can take different sets of values. Discrete random variables take one of a finite number of values. Random variables can take infinitely many values (integer or real values); for instance, *Height* denotes a student's height [42]. Let X be a random variable and $Val(X)$ denote the set of values that a random variable X can take. The distribution over random variable X is denoted by $P(X)$.

The marginal distribution over *Gender* assigns a probability to specific events such as $P(Gender = female)$ and $P(Gender = male)$, as well as to the trivial event $P(Gender \in \{Female, Male\})$. Note that these probabilities are defined by the probability distribution over the original space, for instance, $P(Gender = female) = 0.4$ and $P(Gender = male) = 0.6$. In the same way, we can define the marginal distribution for *Size*: $P(Size = H) = 0.2$, $P(Size = M) = 0.5$, $P(Size = L) = 0.3$.

In a specific situation, we are interested in questions that involve the values of several random variables, for example, we might be interested in the event "*Size = M*" and "*Gender = female*". We need to consider the joint distribution over these two random variables. The *joint distribution* over a set $\mathcal{X} = \{X_1, ..., X_n\}$ of random variables is denoted by $P(X_1, ..., X_n)$. The joint distribution of two random variables has to be consistent with the marginal distribution, in that, $P(x) = \sum_y P(x, y)$ [42].

The notation $P(Gender|Size = M)$ denotes the conditional distribution over the events describable by *Gender* given the knowledge that the jacket' size is M. The notation $P(X|Y)$ represents a set of conditional probability distributions.

An event α is *independent* of event β in P, denoted $P \models (\alpha \perp \beta)$, if $P(\alpha|\beta) = P(\alpha)$ or if $P(\beta) = 0$. A distribution P satisfies $(\alpha \perp \beta)$ if and only if $P(\alpha \cap \beta) = P(\alpha)P(\beta)$. An event α is *conditionally independent* of event β given event γ in P, denoted $P \models (\alpha \perp \beta|\gamma)$, if $P(\alpha|\beta \cap \gamma) = P(\alpha|\gamma)$ or if $P(\beta \cap \gamma) = 0$. P satisfies $(\alpha|\beta \cap \gamma)$ if and only if $P(\alpha \cap \beta|\gamma) = P(\alpha|\gamma)P(\beta|\gamma)$ [42].

Let $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$ be sets of random variables. $\mathbf{X}$ is conditionally independent of $\mathbf{Y}$ given $\mathbf{Z}$ in a distribution P if P satisfies $(X = x \perp Y = y|Z = z)$ for all values $x \in Val(X)$, $y \in Val(Y)$ and $z \in Val(Z)$. The variables in the set $\mathbf{Z}$ are often said to be observed. If the set $\mathbf{Z}$ is empty, then it is written as $(\mathbf{X} \perp \mathbf{Y})$ and $\mathbf{X}$ and $\mathbf{Y}$ are said to be marginally independent. The distribution P satisfies $(\mathbf{X} \perp \mathbf{Y}|\mathbf{Z})$ if and only if $P(X, Y|Z) = P(X|Z)P(Y|Z)$ [42].

A random variable X has a *Gaussian distribution* with mean μ and variance σ^2, denoted $X \sim \mathcal{N}(\mu; \sigma^2)$, if it has the a *probability density function (PDF)* $p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ [42]. A function $p : \mathcal{R} \mapsto \mathcal{R}$ is a probability density function or (PDF) for X if it is a non negative integrable function such that $\int_{Val(X)} p(x)dx = 1$. A *standard Gaussian* is one with mean 0 and variance 1. A set of random variables is *independent and identically distributed (IID)* if each random variable has the same probability distribution and all are mutually independent. Let $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ be sets of continuous random variables with joint density $p(\mathbf{A}, \mathbf{B}, \mathbf{C})$. $\mathbf{A}$ is conditionally independent of $\mathbf{B}$ given $\mathbf{C}$ if $p(a|c) = p(a, b|c)$ for all a, b, c such that $p(c) > 0$.

2.1.2 Graphs

A *graph* is a data structure $\mathcal{K}$ containing nodes and edges. Formally, a graph is defined by $\mathcal{K} = (\mathcal{X}, \mathcal{E})$, where $\mathcal{X}$ is a set of nodes (vertices) $\mathcal{X} = \{X_1, X_2, ..., X_n\}$ and $\mathcal{E}$ is a set of edges $\mathcal{E} \subseteq \mathcal{X} \times \mathcal{X}$ [42] [39]. The set of edges $\mathcal{E}$ is a set of pairs, where each pair is one of $X_i \rightarrow X_j$, $X_j \rightarrow X_i$, or X_i—X_j, for $X_i, X_j \in \mathcal{X}, i < j$ [42]. Two nodes are said to be *adjacent* if there is an edge connecting them. *Adjacent edges* are edges that share a common node. A *directed edge* connects two nodes X_i and X_j and it has an arrowhead. Examples of a directed edge are $X_i \rightarrow X_j$ or $X_i \leftarrow X_j$. $X_i \leftrightarrow X_j$ is an example of a *bi-directed* edge. An *undirected edge* connects two nodes without an arrowhead. An example of an undirected edge is X_i—X_j. A graph is *directed* if all edges are either $X_i \rightarrow X_j$ or $X_i \leftarrow X_j$ [42]. A graph is *undirected* if all edges are X_i—X_j [42]. If a graph has directed and undirected edges, it is called a *partially directed graph*. An *acyclic graph* is a graph without a cycle. A graph containing all directed edges with no cycle is called *Directed Acyclic Graph* (DAG) [39].

Let $X_i \rightarrow X_j \in \mathcal{E}$, X_j is called the *child* of X_i in $\mathcal{K}$ and X_i is called the *parent* of X_j in $\mathcal{K}$. Suppose X_i—X_j, X_i is a *neighbor* of X_j in $\mathcal{K}$, and vice versa. X_i is called an *ancestor* of X_j and X_j is called a *descendant* of X_i if there is a directed path from X_i to X_j. Pa_X,

Ch_X, Nb_X, and An_X denote parent, children, neighbor and ancestor of X, respectively. A *directed cycle* occurs in $\mathcal{K}$ when $X_j \to X_i$ is in $\mathcal{K}$ and $X_i \in An_{X_j}$. An *almost directed cycle* occurs when $X_i \leftrightarrow X_j$ in $\mathcal{K}$ and $X_i \in An_{X_j}$ [78].

A *path* in $\mathcal{K}$ is a sequence of distinct nodes $\langle X_1, ..., X_n \rangle$ such that for $1 \leqslant i \leqslant n-1$, X_i and X_{i+1} are adjacent in $\mathcal{K}$ [78]. A *directed path* from X_1 to X_n in $\mathcal{K}$ is a sequence of distinct nodes $\langle X_1, ..., X_n \rangle$ such that for $1 \leqslant i \leqslant n-1$, X_i is a *parent* of X_{i+1} in $\mathcal{K}$ [78]. $X_1, .., X_n$ form a *trail* in $\mathcal{K}$ if for every $i = 1, .., n-1$, there is $X_i \rightleftharpoons X_{i+1}$ [42]. $X_i \rightleftharpoons X_j$ represents that X_i and X_j are connected via some edges, whether directed (in any direction) or undirected [42]. The node X_3 on the path $X_1 \to X_3 \leftarrow X_2$ is called an *unshielded collider* if X_1 and X_2 are not adjacent.

A graph can be used to represent the conditional independence relations, for example, those relations that are true for some probability distribution. *d-separation* is a test for conditional independence. *d-separation* only works for DAGs. Suppose, three nodes X_i, X_j, X_k then X_i and X_j are d-separated given X_k if there is no *active trail* between X_i and X_j given X_k. Let $X_1 \rightleftharpoons ... \rightleftharpoons X_n$ be a trail in graph $\mathcal{G}$ and Z be a subset of observed variables. The trail $X_1 \rightleftharpoons ... \rightleftharpoons X_n$ is active given Z if whenever there is a $v-structure$ $X_{i-1} \to X_i \leftarrow X_{i+1}$, then X_i or one of its descendants are in Z and no other node along the trail is in Z [42]. Figure 2.1 shows an example of *active trail* (a and b) and not active trail (c). Figure 2.2 shows the possible relationships of a trail: direct connection, common cause, common effect ($v-structure$), and indirect causal/evidential effect.

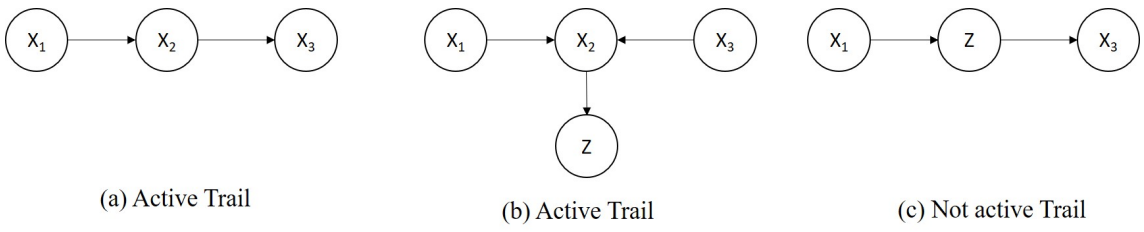


Figure 2.1: Example of active trail (a and b) and not active trail (c)

The *skeleton* of a DAG $\mathcal{G}$ is the undirected graph obtained from $\mathcal{G}$ by replacing the directed edges with undirected edges. Triple nodes X_i, X_j, X_k form a *v-structure* in DAG $\mathcal{G}$ contains directed edge $X_i \to X_j$ and $X_k \to X_j$, and X_i and X_k are not adjacent in $\mathcal{G}$ [39].

A *mixed graph* is a graph containing three kinds of edges: directed ($\to$), bi-directed ($\leftrightarrow$), and undirected (—), and at most one edge between any two vertices [78]. The two ends of an edge are called *marks* or *orientations*. Two kinds of marks are arrowhead (>)

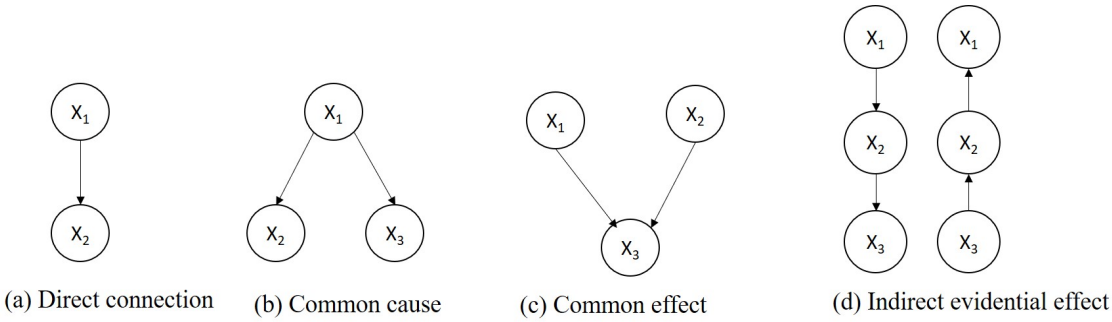


Figure 2.2: Possible relationships in a trail

and tails (-), for instance, a bi-directed edge has both arrowheads and an undirected edge has both tails.

2.1.3 Partially Directed Acyclic Graph (PDAG)

Directed Acyclic Graphs (DAGs) are used as statistical models and causal models [78]. The causal information between variables can be represented by a directed acyclic graph (DAG) in which the nodes represent random variables and the edges represent direct causal effects [20]. In a simple graph $X_1 \rightarrow X_2$, X_1 is a direct cause of X_2 [20]. Some DAGs can describe exactly the same conditional independence information and they are called *Markov equivalent*. Two DAGs are *Markov equivalent* if and only if they have the same adjacencies and the same unshielded colliders [1]. In other words, two DAGs are equivalent if and only if they have the same skeleton and the same v-structures [39]. Figure 2.3 shows an example of Markov equivalence class of DAGs because they imply the single conditional independence relationship "$X_1 \perp X_3|X_2$" (X_1 is conditionally independent of X_3 given X_2) [20]. The number of possible DAGs is super-exponential in the number of nodes so the estimation of DAG from data is difficult and computationally non-trivial [39]. A *completed partially directed acyclic graph* (CPDAG) describes Markov equivalence classes of DAGs.

A *partially directed acyclic graph* (PDAG) is a graph where some edges are directed and some are undirected and one cannot trace a cycle by following the direction of directed edges and any direction for undirected edges [39]. A PDAG is completed if it satisfies the following:

1. every directed edge exists also in every DAG belonging to the equivalence class of the DAG

2. for every undirected edge X_i—X_j there exists a DAG with $X_i \rightarrow X_j$ and a DAG

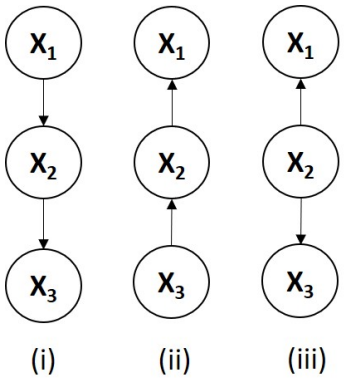


Figure 2.3: Markov equivalence class

with $X_i \leftarrow X_j$ in the equivalence class

A *completed partially directed acyclic graph* (CPDAG) is a unique equivalence class of *Directed Acyclic Graphs* (DAG) [42]. A CPDAG is a common tool for visualizing equivalence classes of DAGs [39]. There are two main parts for estimating the CPDAG: estimation of the skeleton and partial orientation of edges [39]. Figure 2.4 shows the example of a DAG and a CPDAG.

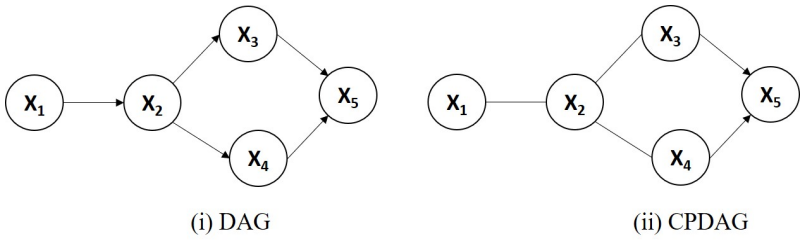


Figure 2.4: The example of (i) DAG and (ii) CPDG

2.1.4 Maximal Ancestral Graph (MAG)

Assuming there are no latent variables in a dataset, some causal algorithms work for learning causal graphs and their output is properly representable by a DAG. However, in practice, there maybe latent variables that might exist on the datasets. If we put an assumption that there are latent variables on the dataset, the causal structure might not be suitably representable by a DAG. *Ancestral graph* models were developed to represent the causal graph structure when these graphs are generated by the assumption that there are latent variables in the dataset. The nodes of an ancestral graph represent random variables and the graph is interpreted as encoding a set of conditional independence [78].

Figure 2.5 shows (i) a DAG with latent variables and (ii) the ancestral graph representing the latent variable using bi-directed between X_2 and X_4 [1].

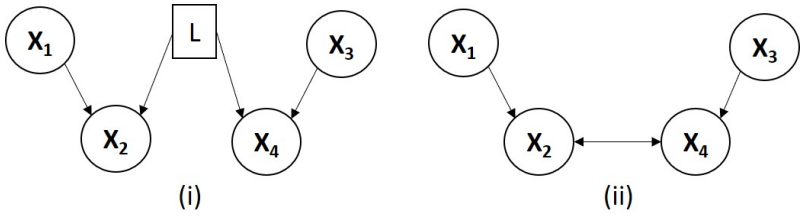


Figure 2.5: (i) A DAG with a latent variable L; (ii) The ancestral graph for (i)

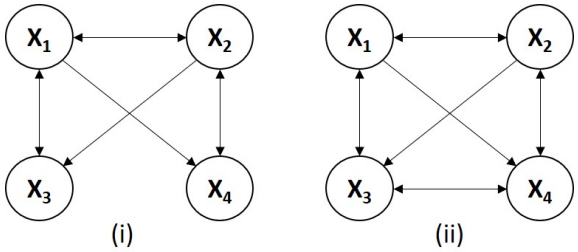


Figure 2.6: (i) An ancestral graph that is not maximal; (ii) A maximal ancestral graph

A mixed graph is ancestral if it satisfies this condition [78]:

- there is no directed cycle
- there is no almost directed cycle
- for any undirected edge X_1 — X_2, X_1 and X_2 have no parents or spouses

Figure 2.6 shows an example of ancestral graphs [78]. In a mixed graph, a non-endpoint node X on path p is called a *collider* if the two edges incident to X on p are both into X; otherwise X is called a *non-collider* on p. Figure 2.6 (i) shows X_2 is a collider on the path $\langle X_1, X_2, X_4 \rangle$, but it is a non-collider on the path $\langle X_3, X_2, X_4 \rangle$. An ancestral graph is maximal if and only if there is no *primitive inducing path* between any two non-adjacent nodes in the graph [78]. Let X_1 and X_2 be nodes and **L,S** be two disjoint sets of nodes not containing X_1 and X_2 in an ancestral graph. A path p between X_1 and X_2 is called an *inducing path* relative to **L,S** if every non-endpoint node on p is either in **L** or a collider, and every collider on p is an ancestor of either X_1, X_2, or a member of **S**. A *primitive inducing path* between X_1 and X_2 occurs when $\mathbf{L} = \mathbf{S} = \emptyset$. The path X_3, X_1, X_2, X_4 on Figure 2.6 (i) is a primitive inducing path between X_3 and X_4, so the graph is not maximal. Figure 2.6 (ii) shows a *maximal ancestral graph*. The meaning of three kinds of edges on a MAG can be explained as follows [78].

- $X_1 \rightarrow X_2$ means that X_1 is a cause of X_2 or of some selection variable, but X_2 is not a cause of X_1 or of any selection variable;
- $X_1 \leftrightarrow X_2$ means that X_1 is not a cause of X_2 or of any selection variable, and X_2 is not a cause of X_1 or of any selection variable;
- $X_1 - X_2$ means that X_1 is a cause of X_2 or of some selection variable, and X_2 is a cause of X_1 or of some selection variable.

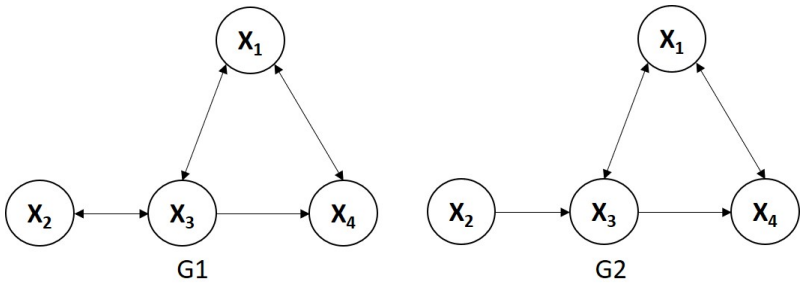


Figure 2.7: Two MAGs that show Markov equivalence

Every DAG with latent and selection variables can be represented by a unique MAG over the observed variables [20]. Some MAGs can describe exactly the same conditional independence relationship [1] [20]. These graphs are called Markov equivalent MAG [1]. Figure 2.7 shows two MAGs that have Markov equivalence. A *partial ancestral graph* (PAG) can be used to represent MAGs that form a Markov equivalence class [20].

2.1.5 Partial Ancestral Graph (PAG)

Let $[\mathcal{G}]$ be the Markov equivalence class of MAG $\mathcal{G}$. A *Partial Ancestral Graph* (PAG) for $[\mathcal{G}]$ is a graph $\mathcal{P}$ that possibly has three kinds of endpoints {circle ($\circ$), arrow ($>$),tail ($-$)}. These endpoints on the X_2 end of an edge between X_1 and X_2 have the following meanings [42]:

- An arrowhead $>$ implies that X_2 is not an ancestor of X_1 in any graph in $\mathcal{G}$.
- A straight end — implies that X_2 is an ancestor of X_1 in all graphs in $\mathcal{G}$.
- A circle $\circ$ implies that neither of the two previous cases hold.

The endpoints might form six kinds of edges $\circ\!\!\rightarrow$ (*partially directed*), $\leftrightarrow$ (*bi-directed*), $\circ\!\!-\!\!\circ$ (*nondirected*), $\rightarrow$ (*directed*), $\circ\!\!-$ (*partially undirected*), and — (*undirected*) [78]. $\mathcal{P}$ has

the same adjacencies as $\mathcal{G}$ and any member of $[\mathcal{G}]$, and every non-circle mark in $\mathcal{P}$ is an invariant mark in $[\mathcal{G}]$ [78]. The interpretation of the different edge types is as follows [42]:

- An edge $X_1 \rightarrow X_2$ has (almost) the standard meaning: X_1 is an ancestor of X_2 in all graphs in $\mathcal{G}$ and X_2 is not an ancestor of X_1 in any graph.
- $X_1 \leftrightarrow X_2$ means that X_1 is never an ancestor of X_2 and X_2 is never an ancestor of X_1.
- An edge $X_1 \circ\!\!\rightarrow X_2$ means that X_2 is not an ancestor of X_1 in any graph but X_1 is an ancestor of X_2 in some, but not all, graphs.

Figure 2.8 shows an example of a PAG and several members of the (infinite) equivalence class that it represents. All of the graphs in the equivalence class have one or more active trails between X_1 and X_2, none of which are directed from X_2 to X_1. A PAG is only a partial graph structure, and not a full model; thus, it cannot be used directly for answering causal queries [42].

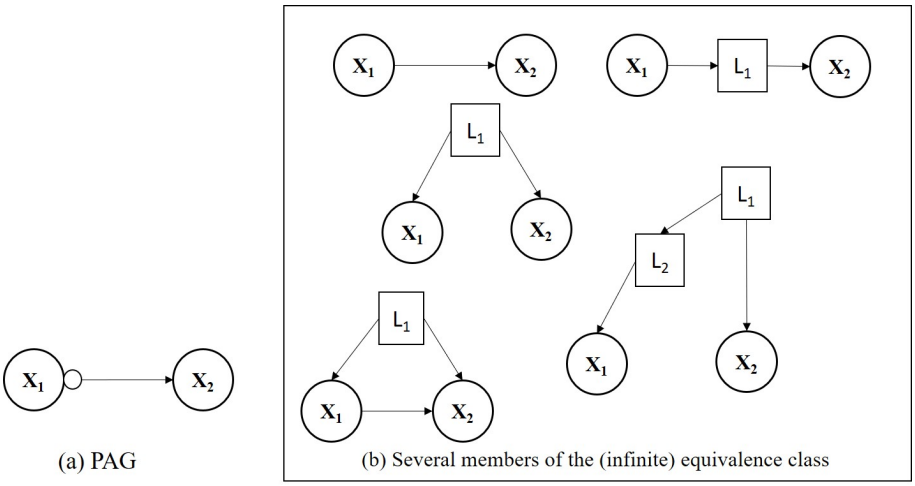


Figure 2.8: A PAG and several members of the (infinite) equivalence class that it represents

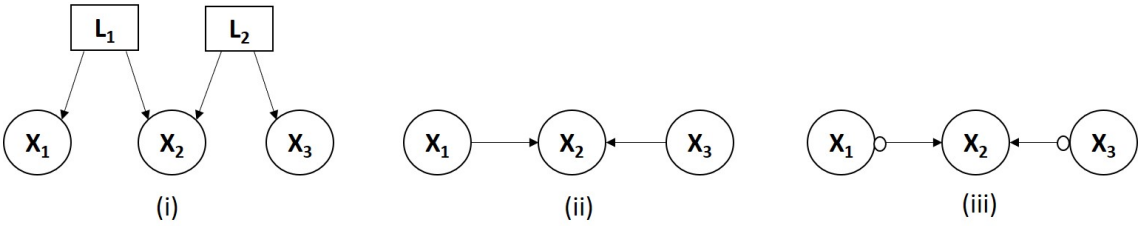


Figure 2.9: (i) DAG with latent variables; (ii) CPDAG; (iii) PAG [20]

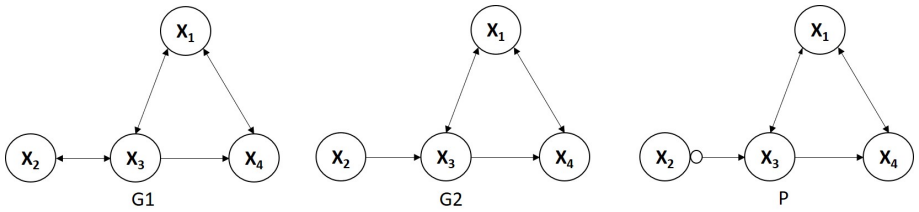


Figure 2.10: A PAG P represents two MAGs $G1$ and $G2$ that form Markov equivalence class

Figure 2.9 (i) shows a DAG contains three observed variables $X = \{X_1, X_2, X_3\}$ and two latent variables L_1 and L_2 [20]. Figure 2.9 (i) implies a single conditional independence relationship $X_1 \perp X_3$. Figure 2.9 (ii) shows a DAG representing the conditional independence relationship for $X_1 \perp X_3$. Figure 2.9 (ii) is an incorrect representation for DAG that has latent variables because there must be no directed path from X_1 to X_2 and from X_3 to X_2 [20]. Figure 2.9 (iii) gives a better representation of Figure 2.9 (i). Figure 2.9 (iii) shows that a PAG implies X_2 is not a cause of X_1, X_3 or a selection variable and this is relevant for conditional independence case $X_1 \perp X_3$ in Figure 2.9 (i). The circle marks at X_1 and X_3 in Figure 2.9 (iii) represent uncertainty about whether or not X_1 and X_3 are causes of X_2. Figure 2.9 (iii) describes the conditional independence relationship $X_1 \perp X_3$ from Figure 2.9 (ii) where X_1 and X_3 are causes of X_2 and it also represents $X_1 \perp X_3$ from Figure 2.9 (i) in which X_1 and X_3 are not causes of X_2 [20]. Figure 2.10 shows an example of two MAGs $G1$ and $G2$ that describe the same conditional independence relationships and a PAG P represents those MAGs.

2.2 Bayesian Network Representation

Bayesian Network is also known as a belief graph in which the structures are used to represent knowledge. Two common approaches to study Bayesian Network are parameter learning and structure learning. *Parameter learning* is applied when there exists a graph structure. *Structure learning* is used to build a Bayesian Network from datasets [42].

A *Bayesian Network* is a pair $\mathcal{B} = (\mathcal{G}, P)$, where P is a set of *conditional probability distributions* (CPD) associated with nodes in graph $\mathcal{G}$ [42]. Let $\mathcal{G}$ be a Bayesian Network graph over the variables $X_1, ..., X_n$. The chain rule of the Bayesian Network can be expressed using equation 2.2 [42]. The *conditional probability distribution* (CPD) or local probabilistic models is the individual factors $\prod_{i=1}^{n} P(X_i | Pa_{X_i}^{\mathcal{G}})$.

$$P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i | Pa_{X_i}^{\mathcal{G}}) \tag{2.2}$$

The Bayesian network representation is a directed acyclic graph (DAG) $\mathcal{G}$, where nodes represent the random variables and edges correspond to the direct influence between one node on another [42]. Graph $\mathcal{G}$ can be viewed as a data structure that uses a skeleton for representing a joint distribution and a compact representation for a set of conditional independence assumptions about a distribution.

Let X be a continuous variable, $U = \{U_1, ..., U_m\}$ be its discrete parents and $Y = \{Y_1, ..., Y_k\}$ be its continuous parents. X has a *conditional linear Gaussian* (CLG) CPD if, for every value $u \in Val(U)$, there is a set of $k+1$ coefficients $a_{u,0}, ..., a_{u,k}$ and a variance σ_u^2 such that $p(X|\mathbf{u}, \mathbf{y}) = \mathcal{N}\left(a_{u,0} + \sum_{i=1}^{k} a_{u,i} y_i; \sigma_u^2\right)$ [42]. A Bayesian network is called a CLG network if every discrete variable has only discrete parents and every continuous variable has a CLG CPD.

2.3 Parameter Learning

Parameter learning from the data is required because obtaining numerical parameters from a human expert is difficult. There are two approaches for parameter learning: maximum likelihood estimation and the Bayesian approach.

2.3.1 Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) is a method to estimate the parameter value that maximizes the likelihood. Let $L(\theta : D)$ be a likelihood function, Maximum Likelihood Estimation can be explained as follows. Given a dataset D, it chooses parameters $\hat{\theta}$ that satisfy $L(\hat{\theta} : D) = \max_{\theta \in \Theta} L(\theta : D)$ [42]. Suppose, a sequence data X is recorded from a simple experiment by throwing a thumbtack, $x = H$ if it shows *head* on the top and $x = T$ if it shows *tail* on top. Suppose we have random variables $X = (H, T, H, T, H, T, H, T, T, T)$ which are independent and identically distributed (*IID*). The probability a head (H) appears is θ and the probability tail (T) appears is $1 - \theta$. The probability of the sequence is $(P\langle H, T, H, T, H, T, H, T, T, T\rangle : \theta) = \theta(1-\theta)\theta(1-\theta)\theta(1-\theta)\theta(1-\theta)(1-\theta)(1-\theta) = \theta^4(1-\theta)^6$. This probability is dependent on the value θ. Different values of θ produce different probability for the sequence. The likelihood function can be defined as $(\theta : \langle H, T, H, T, H, T, H, T, T, T\rangle) = P(\langle H, T, H, T, H, T, H, T, T, T\rangle : \theta) = \theta^4(1-\theta)^6$.

Suppose, the dataset D of this observation contains $M[H]$ heads and $M[T]$ tails. The value $\hat{\theta}$ that maximizes the likelihood of θ relative to D can be estimated using equation 2.3.

$$\hat{\theta} = \frac{M[H]}{M[H] + M[T]} \tag{2.3}$$

This is a simple example of estimating parameters for a Bayesian network. Suppose, a simple network $X \rightarrow Y$ consists of two binary variables X and Y. For a single parameter, the goal of maximum likelihood estimation is to maximize the likelihood (or log-likelihood) function [42]. This network is parameterized by parameter vector θ. The parameterization consists of the following parameters: θ_{x^1} and θ_{x^0} specify the probability of the two values of X; $\theta_{y^1|x^1}$ and $\theta_{y^0|x^1}$ specify the probability of Y given that $X = x^1$; and $\theta_{y^1|x^0}$ and $\theta_{y^0|x^0}$ describe the probability of Y given that $X = x^0$. The notation $\theta_{Y|x^0}$ refers to the set $\{\theta_{y^1|x^0}, \theta_{y^0|x^0}\}$ and $\theta_{Y|X}$ refers to $\theta_{Y|x^1} \cup \theta_{Y|x^0}$.

Let each training instance be a tuple $\langle x[m], y[m] \rangle$ that describes a particular assignment to X and Y. The likelihood function is $L(\theta : D) = \prod_{m=1}^{M} P(x[m], y[m] : \theta)$. The network model specifies that $P(X, Y : \theta)$ has a product form, so the likelihood function can be written as $L(\theta : D) = \prod_m P(x[m] : \theta) P(y[m]|x[m] : \theta)$.

The likelihood function decomposes into a product of terms, one for each group of parameters in θ. The decomposability of the likelihood function can be explained using equation 2.4 and $\theta_{Y|x^0}$ can be maximized using equation 2.5 [42].

$$\prod_{m:x[m]=x^0} P(y[m]|x[m] : \theta_{Y|x^0}) = \theta_{y^1|x^0}^{M[x^0,y^1]} . \theta_{y^0|x^0}^{M[x^0,y^0]} \tag{2.4}$$

$$\theta_{y^1|x^0} = \frac{M[x^0, y^1]}{M[x^0, y^1] + M[x^0, y^0]} = \frac{M[x^0, y^1]}{M[x^0]} \tag{2.5}$$

The implementation of the likelihood function of a Bayesian network for learning the parameters for a Bayesian network with structure $\mathcal{G}$ and parameters θ can be explained as follows. The likelihood can be written as equation 2.6 [42].

$$
\begin{aligned}
L(\theta : D) &= \prod_{m} P_{\mathcal{G}}(\xi[m] : \theta) \\
&= \prod_{m} \prod_{i} P(x_i[m] | pa_{x_i}[m] : \theta) \\
&= \prod_{i} \left[\prod_{m} P(x_i[m] | pa_{x_i}[m] : \theta) \right]
\end{aligned}
\tag{2.6}
$$

The conditional likelihood of a particular variable given its parents in the network can be defined using equation 2.7. $\theta_{X_i}|Pa_{x_i}$ denotes the subset of parameters that determines $P(X_i|Pa_{X_i})$ in the model [42]. The local likelihood function for X_i can be defined by equation 2.8. Each local likelihood function can be maximized independently of the rest of the network and the solutions are combined to get an MLE solution. The decomposition of the global problem to independent sub-problems is an efficient way to solve the MLE problem.

$$
L(\theta : D) = \prod_{i} L_i(\theta_{X_i|Pa_{X_i}} : D) \tag{2.7}
$$

$$
L_i(\theta_{X_i|Pa_{X_i}} : D) = \prod_{m} P(x_i[m] | pa_{X_i}[m] : \theta_{X_i|Pa_{x_i}}) \tag{2.8}
$$

The issue in MLE is the number of experiments that affect the estimation parameter. For example, the thumbtack experiment produces 3 heads out of 10 and it gives the parameter $\theta = 0.3$. The value of parameter θ estimates the probability of the event coming out as heads. Moreover, 1,000,000 tosses of the thumbtack come out as 300,000 heads. It also estimates the parameter as 0.3. Maximum Likelihood Estimation does not give a distinction between observations from 10 to 1,000,000 experiments. Regarding this case, the Bayesian approach produces better parameter estimation [42].

2.3.2 Bayesian Parameter Estimation

In the *Bayesian Parameter Estimation* approach, the prior knowledge about parameter θ is encoded with a probability distribution. The distribution represents how likely we are a priori to believe the different choices of parameters. In the experiment of throwing thumbtacks, assume that the tosses are conditionally independent given θ. Parameter estimation using the Bayesian approach implements the concept of the prior and posterior [42]. *Prior* is knowledge before doing experiments and *posterior* is knowledge after doing

experiments. $P(\theta)$ is a prior distribution over the value of θ.

Suppose, a set of data D consists of M outcomes: $X[1], X[2], ..., X[M]$. A joint distribution $P(D, \theta)$ over the data and the parameters can be written as $P(D, \theta) = P(D|\theta)P(\theta)$. The prior distribution over the possible values in Θ captures the initial uncertainty about the parameters and the previous experience before starting the experiment. After specifying the likelihood function and the prior, the data can be used to derive the posterior distribution over the parameters. The posterior can be derived by the Bayes rule $P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$ [42]. The term $P(D)$ is the marginal likelihood of the data $P(D) =$

$$\int_{\Theta} P(D|\theta)P(\theta)d\theta$$

The MLE approach estimates the parameters $\hat{\theta}$ in parameter space Θ that are 'best' given the data. The Bayesian approach estimates the parameters by keeping track of 'beliefs' about the value of θ.

2.4 Structure Learning

There are two main approaches for structure learning in a Bayesian Network: constraint-based and score based approach. *Constraint-based structure learning* methods observe the Bayesian Network as a representation of interdependencies [42]. The graph is built from the data based on conditional independence tests. This method works well for learning a graph from the data for a simple graph. If the graph is complex, it involves a huge number of variables, and many tests are required to learn the structure, so there will a fair number of erroneous test results. Constraint-based structure learning methods produce an almost correct structure and work efficiently if there are few variables and a large sample size. The PC Algorithm, FCI, and RFCI are examples of constraint-based algorithms. The PC Algorithm, FCI, and RFCI are explained in Chapter 2.5.

Score-based structure learning methods learn a Bayesian Network by specifying a statistical model and choose the model that fits the observed data [42]. It uses the hypothesis space of possible graph structures, then a scoring function method to find the best graph. The goal of the score-based structure learning method is to find the highest-scoring graph structure. The most important thing in the score-based structure learning method is choosing the proper scoring function. An example of score-based learning method is Greedy Equivalence Search (GES). GES is explained in Chapter 2.5. Learning the optimal

structure using the Bayesian scoring criterion is an NP-hard problem [16]. The searching problem is usually an NP-hard problem that makes it an inefficient solution [42]. A study shows that causal model discovery is not an NP-hard problem for sparse graphs bounded by node degree k [18]. It leads to a theoretical worst-case running time $O(N^{2(k+2)})$ in the number of independence tests [18].

Both constraint-based and score-based learning can be used individually, but recent research shows that the hybrid approach possibly improves causal discovery. An example of the hybrid algorithms of constraint-based and score-based learning is Greedy Search for Maximal Ancestral Graph [69].

The *Bayesian score* is a scoring function that is based on a Bayesian perspective. The Bayesian score defines a structure prior $P(\mathcal{G})$ that puts a prior probability on different graph structures, and a parameter prior $P(\theta_{\mathcal{G}}|\mathcal{G})$, that puts a probability on a different choice of parameters once the graph is given. In the implementation of Bayesian rule, it applies $P(\mathcal{G}|D) = \frac{P(D|\mathcal{G})P(\mathcal{G})}{P(D)}$. The Bayesian score can be defined as $SCORE_B(\mathcal{G} : D) = \log P(D|\mathcal{G}) + \log P(\mathcal{G})$ [42].

The ability to credit a prior over structures gives us an opportunity of preferring some structures over others. $P(D|\mathcal{G})$ takes into consideration the uncertainty over the parameters $P(D|\mathcal{G})] = \int_{\Theta_{\mathcal{G}}} P(D|\theta_{\mathcal{G}}, \mathcal{G})P(\theta_{\mathcal{G}}|\mathcal{G})d\theta_{\mathcal{G}}$, where $P(D|\theta_{\mathcal{G}}, \mathcal{G})$ is the likelihood of the data given the network $\langle \mathcal{G}, \theta_{\mathcal{G}} \rangle$ and $P(\theta_{\mathcal{G}}|\mathcal{G})$ is the prior distribution over different parameter values for the network $\mathcal{G}$ [42]. $P(D|\mathcal{G})$ is called the marginal likelihood of the data given the structure. The marginal likelihood is the average value of this function, where we average based on the prior measure $P(\theta_{\mathcal{G}}|\mathcal{G})$. By integrating $P(D|\theta_{\mathcal{G}}, \mathcal{G})$ over the different choices of parameters $\theta_{\mathcal{G}}$, we are measuring the expected likelihood, averaged over different possible choices of $\theta_{\mathcal{G}}$.

The relationship between speed and accuracy of structure learning algorithms is not simple. A recent study has revealed the comparison of speed and accuracy among score-based, constraint-based and hybrid algorithms [59]. Different algorithms have different computational complexities, so their speed might be different between large and small graphs [59].

2.5 Causal Algorithms

2.5.1 Greedy Equivalence Search

Greedy Equivalence Search (GES) is a score-based algorithm for Bayesian Network structure learning. GES has two main steps : Forward Equivalence Search (FES) and Backward Equivalence Search (BES) [17].

The Forward Equivalence Search (FES) is started by an empty CPDAG. In this step, all vertices (nodes) have no edges. The algorithm applies GES *insert* operator greedily until no operator has a positive score. After GES reaches a local maximum, it applies a Backward Equivalence Search (BES). In this step, the algorithm applies a *delete* operator until no operator has a positive score. The procedure of *Insert* and *Delete* can be defined as follows [16].

Insert (X,Y,**T**):

For non-adjacent nodes X and Y in P^c, and for any subset **T** of the neighbors of Y that are not adjacent to X, the $Insert$(X,Y,**T**) operator modifies P^c by:

(1) inserting the directed edge $X \rightarrow Y$

(2) for each $T \in \mathbf{T}$, directing the previously undirected edge between T and Y as $T \rightarrow Y$

Delete (X,Y,**H**):

For adjacent nodes X and Y in P^c connected either as X—Y or $X \rightarrow Y$, and for any subset **H** of the neighbours of Y that are adjacent to X, the $Delete$(X,Y,**H**) operator modifies P^c deleting the edge between X, and for each $H \in \mathbf{H}$:

(1) directing the previously undirected edge between Y and H as $Y \rightarrow H$

(2) directing any previously undirected edge between X and H as $X \rightarrow H$

The '**T**' refers to the set-argument in the Insert operator. Every node of **T** becomes a *tail* node in a new v-structure as an outcome of the operator. The '**H**' in Delete operator means every node in this list becomes a *head* node in a new v-structure.

2.5.2 PC algorithm

The *PC algorithm* is a constraint-based structure learning method introduced by Peter Spirtes and Clark Glymour [62]. Generally, the PC algorithm consists of two main stages: generating a skeleton and orienting the edges [62]. The detailed procedure PC algorithm can be explained by Algorithm 1 and Algorithm 2 (see Appendix A.1) [19]. Figure 2.11 shows the illustration to generate a skeleton in the PC. Suppose, a dataset X has 5

variables and n data points. The first step is generating a complete undirected network containing p nodes, where p is the number of variables in the dataset. For every triplet node, it runs a conditional independence test. Suppose, it tests three variables A, B, C, "Is A conditionally independent to C given B?", if the test result "$A \perp C|B = true$", then the edge $A - C$ is deleted and B is saved as a *separation set* **Sepset(A,C)**. It repeats for each ordered pair of adjacent nodes. The output of this step is an undirected graph that is called the skeleton $\mathcal{S}$. The information of the conditional independence test in step one will be applied to orient the edges. Figure 2.12 shows the procedure in the second step to orient the unshielded triplets in the skeleton $\mathcal{S}$ based on the information in the *separation set*. Suppose, for triplets C, D, E, the pair (C, E) is adjacent in $\mathcal{S}$ but the pair (C, D) is not adjacent in $\mathcal{S}$. It orients $C - E - D$ as $C \rightarrow E \leftarrow D$ if and only if E is not in **Sepset(C,D)**. Let X_i, X_j, X_k be the triplet, the algorithm applies R_1, R_2, R_3 to orient the remaining undirected edges [19].

R_1 : orient $X_i - X_k$ as $X_i \rightarrow X_k$ whenever there is a directed edge $X_i \rightarrow X_j$ such that X_i and X_k are not adjacent (otherwise a new v-structure is created);

R_2 : orient $X_i - X_j$ as $X_i \rightarrow X_j$ whenever there is a chain $X_i \rightarrow X_k \rightarrow X_j$ (otherwise a directed cycle is created);

R_3 : orient $X_i - X_j$ as $X_i \rightarrow X_j$ whenever there are two chains $X_i - X_k \rightarrow X_j$ and $X_i - X_l \rightarrow X_j$ such that X_k and X_l are not adjacent (otherwise a new v-structure or a directed cycle is created).

The PC algorithm identifies the equivalence class of a Bayesian Network in polynomial time if the graph structure is a Directed Acyclic Graph (DAG) and each node has a limited degree [17]. The output of the PC algorithm is represented by a Completed Partially Directed Acyclic Graph (CPDAG) [39]. The PC algorithm is a causal algorithm that can be used to learn causal graphs by assuming there are no latent variables in the dataset.

2.5.3 Fast Causal Inference (FCI)

Fast Causal Inference (FCI) is a constraint-based structure learning method. The early steps of FCI includes generating a skeleton graph then orienting the edges. The advantage of FCI is that it allows the presence of the latent variables. The FCI procedure can be found in Algorithms 3, 4, 5, and 6 (see Appendix A.2) [20]. FCI produces a causal graph that is represented by a Partial Ancestral Graph (PAG) [20]. A PAG has three different marks to form six types of edges: $\circ\!\!\rightarrow$, $\leftrightarrow$, $\circ\!\!-\!\!\circ$, $\rightarrow$, $\circ\!\!-$, and$-$ [78].

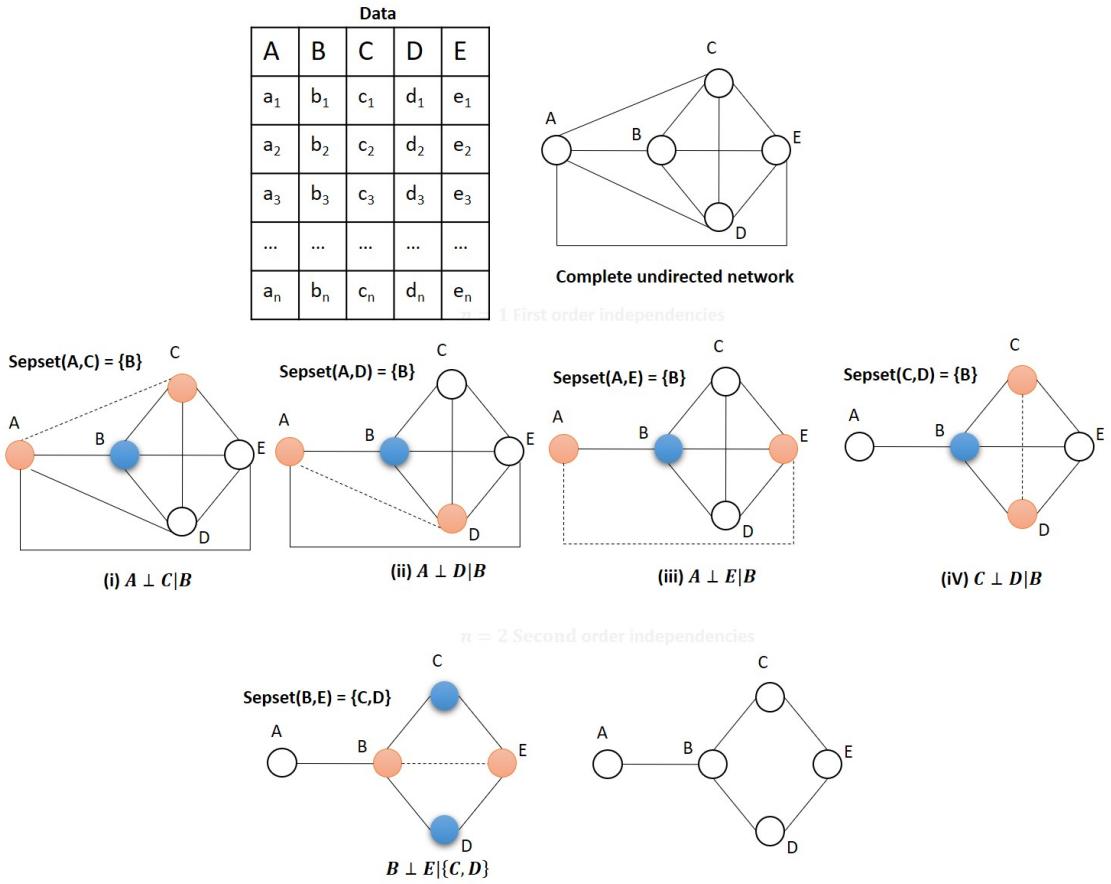


Figure 2.11: PC Algorithm Step 1: Creating Skeleton

Figure 2.13 illustrates the rule for orienting the edges in FCI. FCI orientation procedure applies the rules developed by Zhang [78]. There are some terms used in the rules that can be defined as follows. A path $p = \langle V_0, ..., V_n \rangle$ is said to be uncovered if for every $1 \leq i \leq n-1$, V_{i-1}, and V_{i+1} are not adjacent, i.e., if every consecutive triple on the path is unshielded. A path $p = \langle V_0, ..., V_n \rangle$ is said to be potentially directed (p.d.) from V_0 to V_n if for every $0 \leq i \leq n-1$, the edge between V_i and V_{i+1} is not into V_i or out of V_{i+1}. A special case of a p.d. path is where every edge on the path is of the form $\circ\!\!-\!\!\circ$, this path is called a circle path. FCI orientation rules can be explained as follows [78]:

R_0: For each unshielded triple $\langle \alpha, \gamma, \beta \rangle$ in P, orient it as a collider $\alpha *\!\!\rightarrow \gamma \leftarrow\!\!* \beta$ if and only if γ is not in $\mathbf{Sepset}(\alpha, \beta)$.

R_1: If $\alpha *\!\!\rightarrow \beta \circ\!\!-\!\!* \gamma$, and α and γ are not adjacent, then orient the triple as $\alpha *\!\!\rightarrow \beta \rightarrow \gamma$.

R_2: If $\alpha \rightarrow \beta *\!\!\rightarrow \gamma$ or $\alpha *\!\!\rightarrow \beta \rightarrow \gamma$, and $\alpha *\!\!-\!\!\circ \gamma$, then orient $\alpha *\!\!-\!\!\circ \gamma$ as $\alpha *\!\!\rightarrow \gamma$.

R_3: If $\alpha *\!\!\rightarrow \beta \leftarrow\!\!* \gamma$, $\alpha *\!\!-\!\!\circ \theta \circ\!\!-\!\!* \gamma$, α and γ are not adjacent, and $\theta *\!\!-\!\!\circ \beta$, then orient $\theta *\!\!-\!\!\circ \beta$ as $\theta *\!\!\rightarrow \beta$.

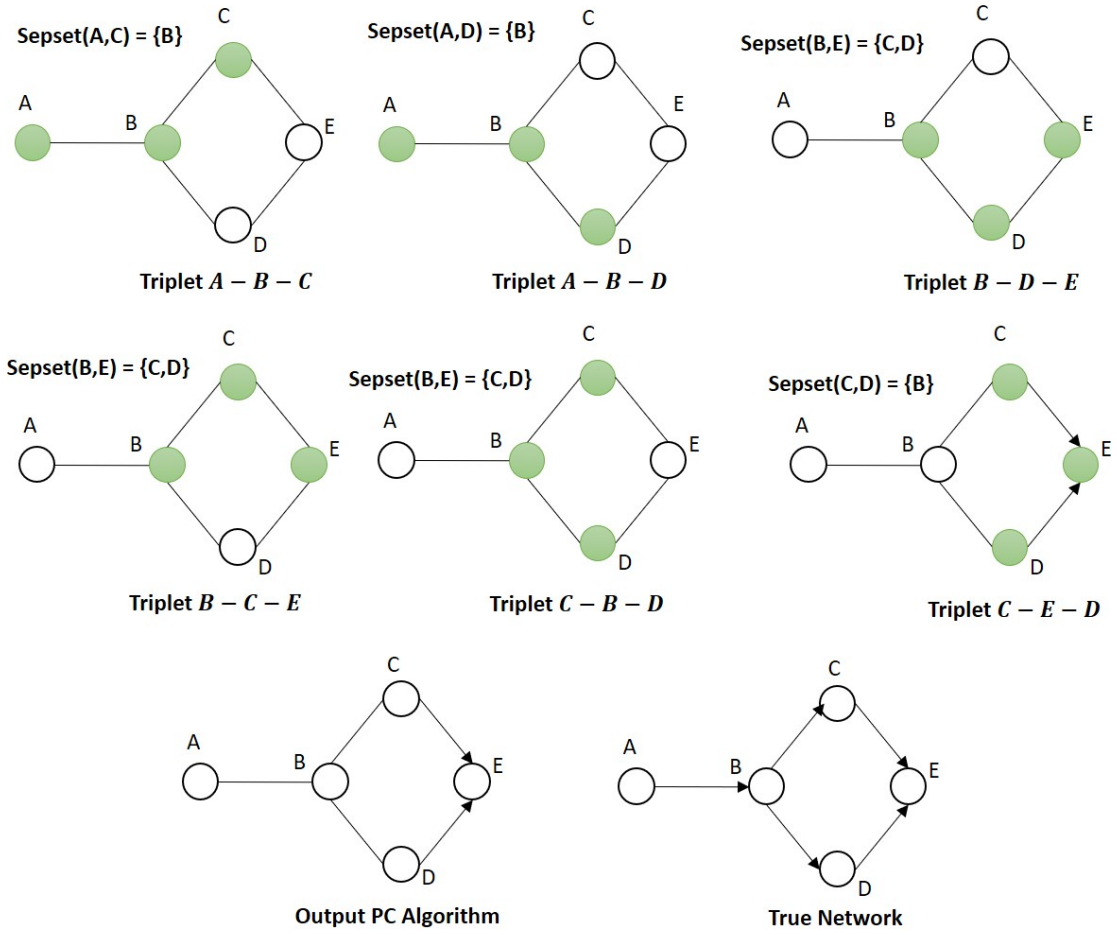


Figure 2.12: PC Algorithm Step 2: Orienting the skeleton

R_4: if $u = \langle \theta, ..., \alpha, \beta, \gamma \rangle$ is a discriminating path between θ and γ for β, and $\beta \circ\!\!-\!\!* \gamma$; then if $\beta \in \mathbf{Sepset}(\theta, \gamma)$, orient $\beta \circ\!\!-\!\!* \gamma$ as $\beta \rightarrow \gamma$; otherwise orient the triple $\langle \alpha, \beta, \gamma \rangle$ as $\alpha \leftrightarrow \beta \leftrightarrow \gamma$.

R_5: For every (remaining) $\alpha \circ\!\!-\!\!\circ \beta$, if there is an uncovered circle path $p = \langle \alpha, \gamma, ..., \theta, \beta \rangle$ between α and β s.t. α, θ are not adjacent and β, γ are not adjacent, then orient $\alpha \circ\!\!-\!\!\circ \beta$ and every edge on p as undirected edges ($-$).

R_6: If $\alpha -\!\!\circ \beta \circ\!\!-\!\!* \gamma$ (α and γ may or may not be adjacent), then orient $\beta \circ\!\!-\!\!* \gamma$ as $\beta -\!\!* \gamma$.

R_7: If $\alpha -\!\!\circ \beta \circ\!\!-\!\!* \gamma$ and α, γ are not adjacent, then orient $\beta \circ\!\!-\!\!* \gamma$ as $\beta -\!\!* \gamma$.

R_8: If $\alpha \rightarrow \beta \rightarrow \gamma$ or $\alpha -\!\!\circ \beta \rightarrow \gamma$, and $\alpha \circ\!\!\rightarrow \gamma$, orient $\alpha \circ\!\!\rightarrow \gamma$ as $\alpha \rightarrow \gamma$.

R_9: If $\alpha \circ\!\!\rightarrow \gamma$, and $p = \langle \alpha, \beta, \theta, ..., \gamma \rangle$ is an uncovered p.d. path from α to γ such that γ and β are not adjacent, then orient $\alpha \circ\!\!\rightarrow \gamma$ as $\alpha \rightarrow \gamma$.

R_{10}: Suppose $\alpha \circ\!\!\rightarrow \gamma$, $\beta \rightarrow \gamma \leftarrow \theta$, p_1 is an uncovered p.d. path from α to β, and p_2 is an uncovered p.d. path from α to θ. Let μ be the vertex adjacent to α on p_1 (μ could be β),

and ω be the vertex adjacent to α on p_2 (ω could be θ). If μ and ω are distinct, and are not adjacent, then orient $\alpha \circ\!\!\rightarrow \gamma$ as $\alpha \rightarrow \gamma$.

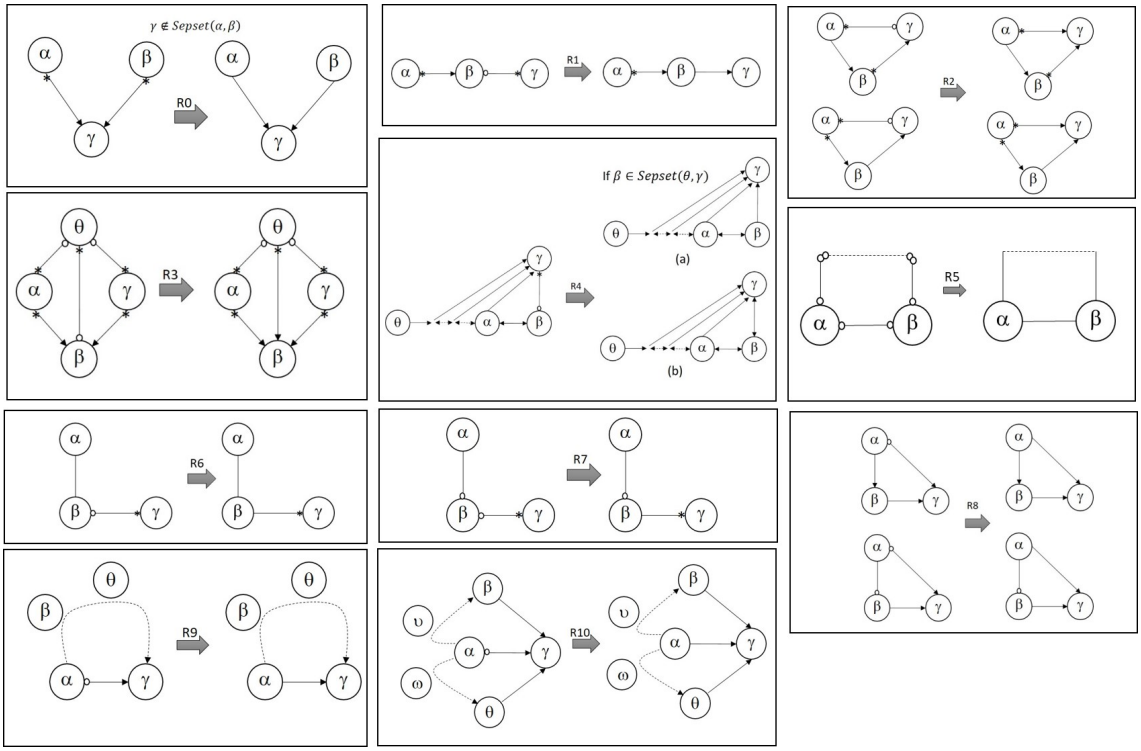


Figure 2.13: FCI Orientation Rule

2.5.4 Really Fast Causal Inference (RFCI)

Really Fast Causal Inference (RFCI) is a constraint-based structure learning algorithm for learning PAGs. RFCI is much faster than FCI because it applies fewer conditional independence tests than FCI [20]. RFCI applies the conditional independence tests on a smaller number of variables but its output may be less informative. The detailed procedure of RFCI can be explained by Algorithms 7, 8, 9, and 10 (see Appendix A.3) [20].

Figure 2.14 (a) shows a DAG consisting of observed variables $X = \{X_1, X_2, X_3, X_4, X_5, X_6\}$, latent variables $L = \{L_1, L_2\}$, and no selection variables $S = \emptyset$ [20]. Suppose, all conditional independence relationships over X that can be read off from this DAG are used as input for FCI and RFCI. Figure 2.14 (b) shows the initial skeleton C_1 and Figure 2.14 (c) shows the output of FCI and RFCI. The edge $X_1 \circ\!\!-\!\!\circ X_5$ exists in the skeleton in Figure 2.14 (b) but is absent in the final output (see Figure 2.14 (c)). Figure 2.14 (c) shows $X_1 \perp X_5 | \{X_2, X_3, X_5\}$.

Figure 2.15 (a) shows a DAG consisting of observed variables $X = \{X_1, X_2, X_3, X_4, X_5\}$,

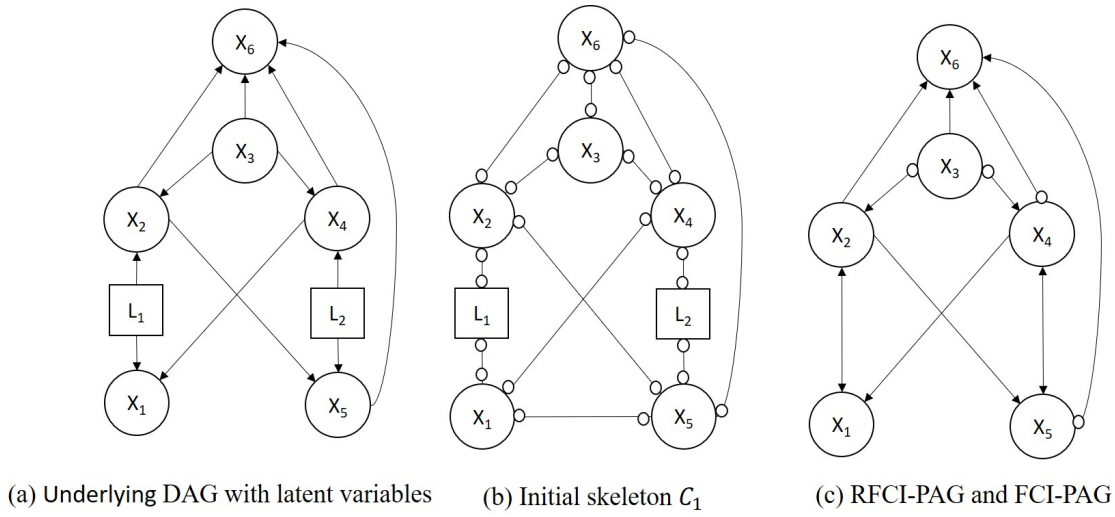


Figure 2.14: The example where outputs of FCI and RFCI are identical

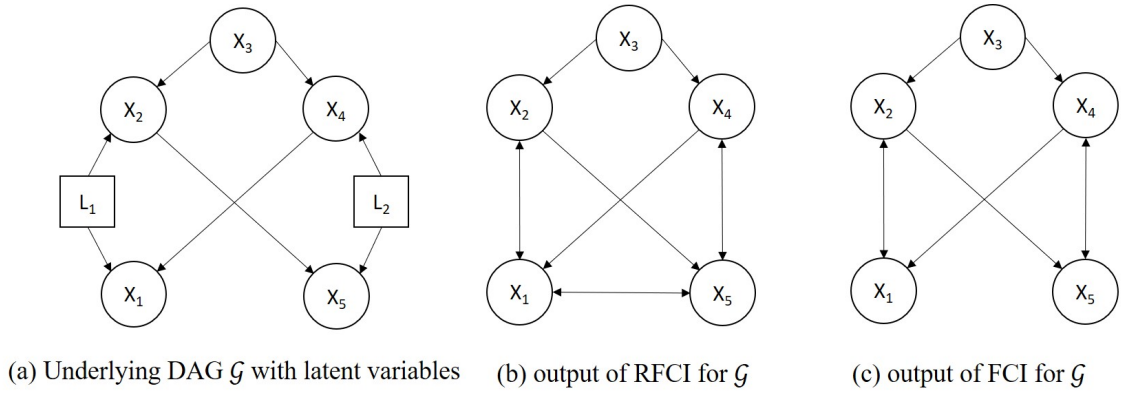


Figure 2.15: The example where outputs of FCI and RFCI are not identical

latent variables $L = \{L_1, L_2\}$, and no selection variables $S = \emptyset$ [20]. Suppose, all conditional independence relationships over X that can be read off from this DAG are used as input for FCI and RFCI. Figure 2.15 (b) shows the output of RFCI and Figure 2.15 (c) shows the output of FCI. The output of RFCI has an extra edge $X_1 \leftrightarrow X_5$. The RFCI-PAG in Figure 2.15 (b) describes two Markov equivalence classes.

Let FCI-PAG be a PAG output from the FCI algorithm and FCI-PAG be a PAG output from the RFCI algorithm. Every FCI-PAG is an RFCI-PAG. Different RFCI-PAGs for the same underlying DAG may have different skeletons, while the FCI-PAG skeleton is unique. The RFCI-PAG skeleton is a supergraph of the FCI-PAG skeleton. An RFCI-PAG can correspond to more than one Markov equivalence class of DAGs.

2.6 Structural Hamming Distance

Structural Hamming Distance (SHD) is a method to evaluate the structure of a learned graph when we know the true graph. The SHD method computes the mismatch of the structure of a learned graph to the true graph. This paper uses some terms that can be defined as follows. A *true graph* is the ground truth and a *learned graph* is a graph generated from a dataset using a causal algorithm. A *true edge* is an edge in the learned graph that has exactly the same position and marks with the edge on the true graph. A *missing edge* is an edge that does not appear in the learned graph but exists in the true graph. An *extra edge* is an edge in the learned graph but originally it does not exist in the true graph. A *wrong marked edge* is an edge in the learned graph that exists in the true graph but has a different mark on one or both sides. The SHD score S is a summation of the number of extra edges, missing edges, and wrong orientation edges [72].

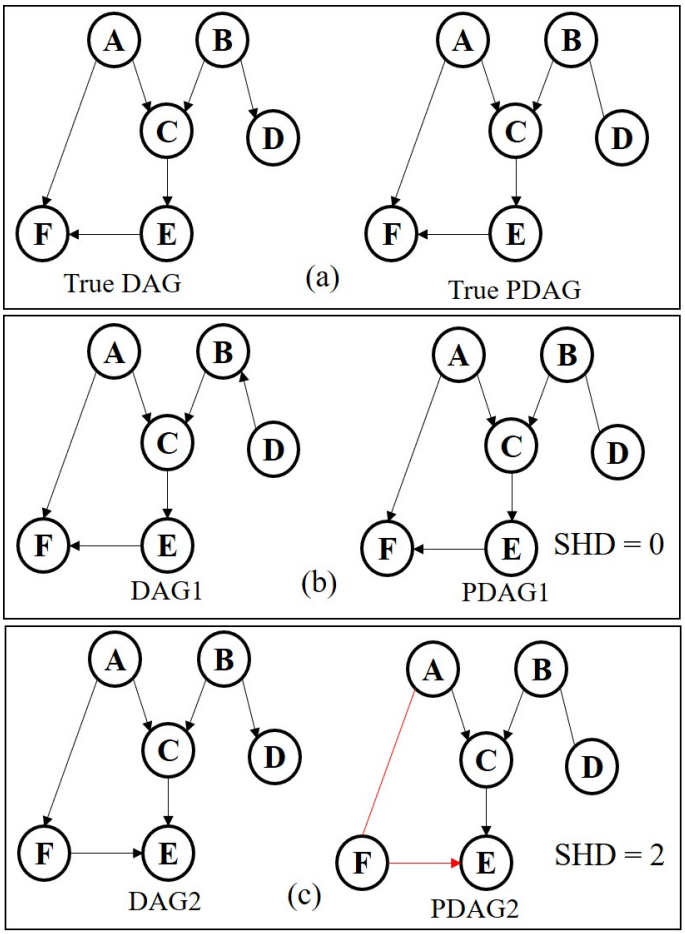


Figure 2.16: Structural Hamming Distance for PDAG

Figure 2.16 shows an example of how to compute the SHD score [72]. Figure 2.16 (a)

shows a hypothetical true DAG and its corresponding PDAG. Figure 2.16 (b) shows a learned DAG (from data sampled from the true DAG) and its corresponding PDAG with $SHD = 0$ (notice that, the actual DAGs are different). Figure 2.16 (c) shows PDAG2 has an $SHD = 2$; it can match the true PDAG by adding one edge direction and by reversing another.

$$S_{norm} = \frac{S}{\binom{n}{2}/2} \tag{2.9}$$

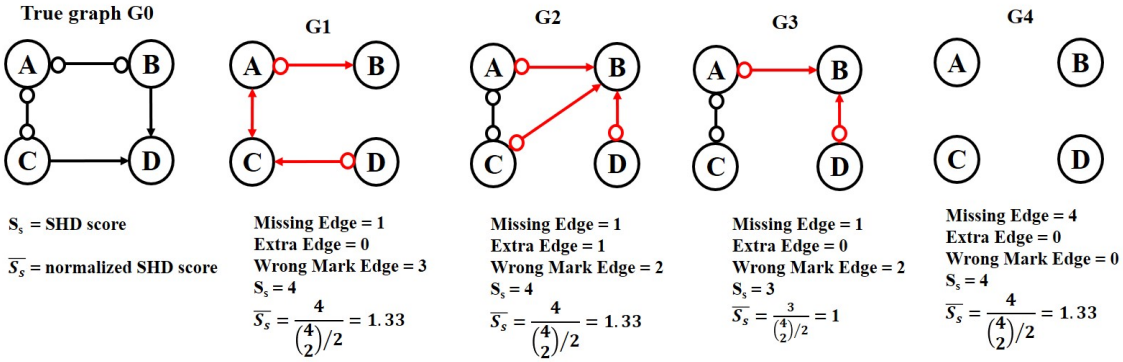


Figure 2.17: Structural Hamming Distance for PAG

The normalized SHD score S_{norm} can be computed using equation 2.9, where $\binom{n}{2}$ is the binomial coefficient and n is the number of observed variables in a dataset [46]. The lower SHD score means that the learned graph has less mismatch to the true graph. It implies that the learned graph has a high similarity to the true graph.

The original SHD was developed for PDAG. We implement SHD to evaluate the structure of the learned graph represented by PAG. Figure 2.17 shows an example of how to implement the SHD method to evaluate the PAGs. It applies the general concept of SHD by summing the number of extra edges, missing edges, and wrong orientation edges. Figure 2.17 shows that two different graph structures might have the same SHD score.

2.7 Mixed Data

Mixed data is defined as a dataset containing different data types. The data types refer to continuous and discrete (binary, ordinal, categorical).

A *continuous variable* is a variable which has infinite possible values. A continuous variable has a real number. A continuous variable can be measured on a continuous scale, for instance, time or human height and weight. The height of a person can be 154.376325635 centimeters, but sometimes it is unnecessary to use that degree of accuracy. Table 2.1 shows a dataset consisting of three continuous variables: height in centimeters, weight in kilograms and BMI (Body Mass Index).

Table 2.1: A dataset contains continuous variables

No	**Student ID**	**Heigh**	**Weight**	**BMI (Body Mass Index)**
1	S01	165.78	50.55	20.8
2	S02	175.44	75.54	24.5
3	S03	150.23	45.12	19.9
4	S04	180.67	67.34	20.6
5	S05	150.67	40.31	17.7

A *discrete Variable* is a variable where values are countable. Binary, ordinal, and categorical variables are kinds of discrete variables. An *ordinal variable* is a discrete variable with values that can be ordered from smallest to largest [30]. For instance, the number of absences in a class during a term and the number of voluntary jobs in a year. Table 2.2 shows a dataset containing ordinal variables. In this example, the value of the number of absences and the number of voluntary jobs can be ordered.

Table 2.2: A dataset contains ordinal variables

No	**Student ID**	**Number of Absences**	**Number of Voluntary Jobs**
1	S01	0	1
2	S02	2	3
3	S03	1	2
4	S04	2	4
5	S05	3	3

A *binary variable* admits exactly two values [49]. Examples of binary values are "yes/no", "male/female", and "on/off". A *categorical variable* is a discrete variable. A categorical variable is called a nominal variable or quantitative variable. The average of the categorical variables has no purpose although they are coded by number [49]. An example of a categorical variable is human blood type. Human blood type has four values: A, B, O and AB. The value of a categorical variable shows $v_1 \neq v_2 \neq ... \neq v_n$. Originally, the categorical variables cannot be ordered as $v_1 \nleq v_2 \nleq ... \nleq v_n$. The ordering might be done by coding into a number, but it cannot absolutely replace the meaning of sequence. Table 2.3 shows an example of a dataset containing categorical variables. The dataset in

Table 2.3 consists of a binary variable (Gender) and categorical variables (Blood Type and Sport Activity).

Table 2.3: A dataset contains categorical variables

No	Student ID	Gender	Blood Type	Sport Activity
1	S01	Female	A	Badminton
2	S02	Male	O	Football
3	S03	Female	B	Basketball
4	S04	Male	AB	Football
5	S05	Female	A	Swimming

Table 2.4 shows an example of mixed data. The dataset consists of 5 variables: Height and Weight (continuous), Blood Type (categorical), Gender (binary), and Number of Absences (ordinal).

Table 2.4: An example of mixed data

No	Student ID	Height	Weight	Blood Type	Gender	Number of Absences
1	S01	165.78	50.55	A	Female	0
2	S02	175.44	75.54	O	Male	2
3	S03	150.23	45.12	B	Female	1
4	S04	180.67	67.34	AB	Male	2
5	S05	150.23	40.31	A	Female	3

2.8 Missing Values Data

Missing value data means that data are missing for some but not all variables and for some but not all cases [2]. Table 2.5 shows an example of mixed data containing missing values, where "*NA*" represents the missing values. The missing data can be categorized into several groups according to the distribution of the data absence. This is called a missing data mechanism: missing completely at random (MCAR), missing at random (MAR), and missing not at random (MNAR) [2] [41] [44]. Suppose, there are two variables X and Z, variable Z represents a variable containing missing data. The value of $R_Z = 1$ if Z is missing, otherwise $R_Z = 0$. The missing completely at random (MCAR) assumption can be defined as $P(R_Z = 1|X, Z) = P(R_Z = 1)$. MCAR happens if the probability of Z missing is not dependent either on the observed variable X or on the chance missing value of Z. Missing at random (MAR) assumption is expressed as $P(R_Z = 1|X, Z) = P(R_Z = 1|X)$. Missing at random (MAR) occurs if the absence of Z depends on variable X, but it does not depend on Z. If the assumption of the absence of MAR is violated, the missing

data can be categorized as missing not at random (MNAR).

Table 2.5: An example of mixed data containing missing values

No	Student ID	Height	Weight	Blood Type	Gender	Number of Absences
1	S01	*NA*	50.55	A	Female	0
2	S02	175.44	75.54	O	Male	2
3	S03	150.23	*NA*	B	Female	1
4	S04	180.67	67.34	AB	Male	2
5	S05	*NA*	40.31	*NA*	Female	3
6	S06	187.67	80.54	A	Male	*NA*

Missing Completely at Random (MCAR)

Suppose there are missing data for variable Y. These data are categorized as Missing Completely at Random (MCAR) if the probability of missing data on Y is unrelated to the value of Y itself or to the value of any other variables in the dataset [2]. MCAR allows for the probability that "missingness" on Y is related to "missingness" on some other variable X. The data could still be categorized as MCAR, for instance, if people who refuse to report their age always reject reporting their income. The MCAR assumption would be violated if people who reject reporting their income were younger or older than the average age who did report their income [2].

Missing at Random (MAR)

Data on Y can be categorized as MAR if the probability of missing data on Y is unrelated to the value of Y, after controlling for the other variables in the analysis. Suppose, there are two variables X and Y, where X is observed and Y sometimes missing. The MAR assumption is expressed as $P(Y_{missing}|X, Y) = P(Y_{missing}|X)$ [2]. It means that the conditional probability of missing data on Y, given Y and X is equal to the probability of missing data on Y given X [2]. Suppose, a dataset consists of two variables (income and marital status), the MAR assumption would be satisfied if the probability of missing data on income depends on a person's marital status, but within the marital status category, the probability of missing income was unrelated to income [2]. The missing data mechanism can be ignored if the data are MAR and the parameters that determine the missing data process are unrelated to the parameters to be estimated [2]. It means that there is no need to model the missing data mechanism as part of the estimation process [2].

Missing Not at Random (MNAR)

If the assumption of missingness of MAR is violated, the missing data can be categorized as missing not at random (MNAR). The missing data mechanism cannot be ignored

if the data are not MAR [2]. In this situation, the missing data mechanism must be modeled to get good estimates of good parameters of interest [2].

2.9 Measuring Dependence of Variables

2.9.1 Measuring Dependence using Partial Correlation

The *correlation coefficient* measures the relationship between two variables. Let X and Y be random variables with covariance σ_{XY} and standard deviations σ_X for variable X and σ_Y for variable Y. The correlation coefficient of X and Y is $\rho_{XY} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y}$ [75]. The covariance of variables X and Y for discrete and continuous variables can be computed using equations 2.10 and 2.11, respectively. The notations μ_X and μ_Y refer to the mean for X and Y, respectively. The correlation coefficient satisfies $-1 \leq \rho \leq 1$.

$$\sigma_{XY} = E[(X - \mu_X)(Y - \mu_Y)] = \sum_x \sum_y (x - \mu_x)(y - \mu_y) f(x, y) \tag{2.10}$$

$$\sigma_{XY} = E[(X - \mu_X)(Y - \mu_Y)] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - \mu_x)(y - \mu_y) f(x, y) dx dy \tag{2.11}$$

Formally, the *partial correlation* between two variables X_i and X_j given $Y = (Y_1, Y_2, \ldots, Y_k)$ is the correlation between residual $e_{(X_i)}$ resulting from the linear regression of X_i with Y and $e_{(X_j)}$ produced from the linear regression of X_j with Y. The regression X_i on Y^* is then defined as the conditional mean value $E(X_i \mid Y^*)$ [48]. Partial correlation has a range value [-1,1].

Computing the partial correlation from the correlation coefficient can be explained as follows. Let ρ be the correlation coefficient and r be partial correlation. The partial correlation of zero order corresponds to the paired correlation coefficient. The partial correlation coefficient of the first order $r_{1,i(j)}$ can be defined from the paired correlation coefficients between residuals $\varepsilon_j = X_1 - E(X_1 \mid Y_j)$ and the residuals $\kappa_j = X_i - E(X_i \mid Y_j)$. The partial correlation of the first order $r_{1,i(j)}$ can be computed using equation 2.12 [48].

$$r_{1,i(j)} = \frac{\rho_{1i} - \rho_{1j}\rho_{ij}}{\sqrt{\left(1 - \rho_{1i}^2\right)\left(1 - \rho_{ij}^2\right)}} \tag{2.12}$$

The partial correlation coefficients of the second order $r_{1,i(j,k)}$ can be defined as the

paired correlation coefficients of residuals $\varepsilon_{j,k} = X_1 - E(X_1 \mid (Y_j, Y_k))$ and the residuals $\kappa_{j,k} = X_i - E(X_i \mid (Y_j, Y_k))$ and can be computed using equation 2.13 [48].

$$r_{1,i(j,k)} = \frac{\rho_{1i(j)} - \rho_{1j(k)}\rho_{ij(k)}}{\sqrt{\left(1 - \rho^2_{1j(k)}\right)\left(1 - \rho^2_{ij(k)}\right)}} \tag{2.13}$$

The partial correlation coefficient of the $(m-1)th$ order $r_{1,i(2,3,...,m)}$ corresponds to the paired correlation coefficient between residuals $\varepsilon_{2,..,m} = X_1 - E(X_1 \mid Y^*)$ and residuals $\kappa_{2,..,m} = X_i - E(X_i \mid Y^*)$, where Y^* contains the components $Y_2, Y_3, ..., Y_{i-1}, Y_{i+1}, ...Y_m$ [48].

The partial correlation coefficient of the higher orders are estimated according to a recursive formula $r_{1,j(2,3,...,j-1)} = \frac{A-BC}{\sqrt{(1-B^2)(1-C^2)}}$, where $A = \rho_{1,j(2,3,...,j-2)}$, $B = \rho_{1,j-1(2,3,...,j-2)}$, and $A = \rho_{j,j-1(2,3,...,j-2)}$. Let ρ be the correlation matrix and $\rho_{i,j}$ be the matrix formed by leaving out the i^{th} row and the j^{th} column of the correlation matrix. Computing the partial correlation coefficients using matrix notation can be defined by the equation 2.14 [48].

$$r_{1,i(2,3,...,m)} = \frac{\det(\rho_{1,i})}{\sqrt{\det(\rho_{1,1})\det(\rho_{i,i})}} \tag{2.14}$$

Let X, Y, Z be random variables and ρ be the correlation coefficient between two variables. The partial correlation r can be computed from the correlation matrix using the equation 2.15, where Z is a single variable.

$$r_{XY.Z} = \frac{\rho_{XY} - \rho_{XZ}\rho_{ZY}}{\sqrt{(1 - \rho^2_{XZ})(1 - \rho^2_{ZY})}} \tag{2.15}$$

A *conditional independence test* for Gaussian data can be done using equation 2.16 [39] [24]. It tests the question "is a variable Z_u conditionally independent of Z_v given Z_S?", where n is the number of samples, S is the separation set, $\hat{\rho}$ is partial correlation, α is a significance level, and $\Phi(.)$ denotes the cumulative distribution function (cdf) of $\mathcal{N}(0,1)$ [24]. Assume, the distribution P of the random variables X is multivariate normal. For $i \neq j \in \{1, ..., p\}, k \subseteq \{1, ..., p\} \backslash \{i, j\}$ and the partial correlation between X_i and X_j given $\{X_r; r \in k\}$ is denoted by $\rho_{i,j|k}$. $\rho_{i,j|k} = 0$ if and only if X_i and X_j are conditionally independent given $\{X_r; r \in k\}$. The estimated partial correlation $\hat{\rho}_{i,j|k}$ can be computed via the regression, inversion of part of the covariance matrix or recursively. Fisher's z-transform is used to test whether the partial correlation is equal to zero or not. We reject

the null hypothesis $H_0(i, j \mid k) : \rho_{i,j|k} = 0$ against the two-sided alternative $H_A(i, j \mid k) : \rho_{i,j|k} \neq 0$ if $\sqrt{n- \mid k \mid -3} \mid Z(i, j \mid k) \mid > \Phi^{-1}(1 - \alpha/2)$ [39].

$$Z_u \perp Z_v \mid Z_S \Leftrightarrow \sqrt{n- \mid S \mid -3} \left| \frac{1}{2} \log \left(\frac{1 + \hat{\rho}_{uv|S}}{1 - \hat{\rho}_{uv|S}} \right) \right| \leq \Phi^{-1}(1 - \alpha/2). \tag{2.16}$$

2.9.2 Measuring Dependence using Distance Correlation

A *Distance correlation* is a method to measure the dependence between random variables [66]. Similar to the product-moment correlation ρ, a Distance correlation has characteristics of a true dependence measure. The Distance correlation satisfies $0 \leq \mathcal{R} \leq 1$ and $\mathcal{R} = 0$ if X and Y are independent. The $\mathcal{R}$ is a function of ρ and $\mathcal{R}(X, Y) \leq |\rho(X, Y)|$ with equality when $\rho = \pm 1$ in the bivariate normal case. Suppose, f_X and f_Y are characteristic functions of X and Y, and $f_{X,Y}$ is a joint function of X and Y, then Distance covariance, Distance variance, and Distance correlation can be computed using equations 2.17, 2.18, and 2.19, respectively [66]. For the observed random variables $(X, Y) = (X_k, Y_k); k = 1, 2, ..., n$ from the joint distribution of random vector X in $\mathbb{R}^p$ and Y in $\mathbb{R}^q$ where p and q are positive integers, we can define $a_{kl} = |X_k - X_l|_p$, $\bar{a}_{k.} = \frac{1}{n} \sum_{l=1}^{n} a_{kl}$, $\bar{a}_{.l} = \frac{1}{n} \sum_{k=1}^{n} a_{kl}$, $\bar{a}_{..} = \frac{1}{n^2} \sum_{k,l=1}^{n} a_{kl}$, and $A_{kl} = a_{kl} - \bar{a}_{k.} - \bar{a}_{.l} + \bar{a}_{..}$, for $k, l = 1, 2, ..., n$. It also defines $b_{kl} = |Y_k - Y_l|_q$ and $B_{kl} = b_{kl} - \bar{b}_{k.} - \bar{b}_{.l} + \bar{b}_{..}$, for $k, l = 1, 2, ..., n$. Thus, the empirical distance covariance and variance can be defined by equations 2.20 and 2.21, respectively [66]. The Distance correlation is more suitable for a dataset containing outliers rather than classic correlation.

$$\mathcal{V}^2(X, Y) = ||f_{X,Y}(t, s) - f_X(t) f_Y(s)||^2 \tag{2.17}$$

$$\mathcal{V}^2(X) = \mathcal{V}^2(X, X) = ||f_{X,X}(t, s) - f_X(t) f_X(s)||^2 \tag{2.18}$$

$$\mathcal{R}^2(X, Y) = \begin{cases} \frac{\mathcal{V}^2(X,Y)}{\sqrt{\mathcal{V}^2(X)\mathcal{V}^2(Y)}}, & \mathcal{V}^2(X)\mathcal{V}^2(Y) > 0 \\ 0, & \mathcal{V}^2(X)\mathcal{V}^2(Y) = 0 \end{cases} \tag{2.19}$$

$$\mathcal{V}^2(X, Y) = \frac{1}{n^2} \sum_{k,l=1}^{n} A_{kl} Bkl \tag{2.20}$$

$$\mathcal{V}^2(X) = \mathcal{V}^2(X, X) = \frac{1}{n^2} \sum_{k,l=1}^{n} A_{kl}^2 \tag{2.21}$$

Suppose $T(X, Y, \alpha, n)$ is the test that rejects independence if $\frac{n\mathcal{V}_n^2(X,Y)}{S_2} > (\Phi^{-1}(1 - \alpha/2))^2$. The $\Phi(.)$ denotes the standard normal cumulative distribution function, $\alpha(X, Y, n)$ denote the achieved significance level of $T(X, Y, \alpha, n)$, and $S_2 = \frac{1}{n^2} \sum_{k,l=1}^{n} |X_k - X_l|_p \frac{1}{n^2} |Y_k - Y_l|_q$ [66].

2.9.3 Measuring Dependence using Hilbert-Schmidt Norms

Gretton et al. proposed an independence criterion based on the eigenspectrum of covariance operators in reproducing kernel Hilbert spaces [33]. Given separable reproducing kernel Hilbert spaces (RKHSs), $\mathcal{F}, \mathcal{G}$ and a joint measure p_{xy} over $(\mathcal{X} \times \mathcal{Y}, \Gamma \times \Lambda)$, the Hilbert-Schmidt Independence Criterion (HSIC) is defined as the squared HS-norm of the associated cross-covariance operator C_{xy} : $HSIC(p_{xy}, \mathcal{F} \times \mathcal{G}) = \|C_{xy}\|^2$. Γ be the Borel sets on $\mathcal{X}$ and Λ the Borel sets on $\mathcal{Y}$. Borel sets are the sets that can be created from open or closed sets by repeatedly taking countable unions and intersections. Let $Z := \{(x_1, y_1), ..., (x_m, y_m)\} \subseteq \mathcal{X} \times \mathcal{Y}$ be a series of m independent observations drawn from p_{xy}. The estimator of Hilbert-Schmidt Independence Criterion (HSIC) can be defined by equation 2.22 [33].

$$HSIC(Z, F, G) = (m - 1)^{-2} tr(KHLH) \tag{2.22}$$

where $H, K, L \in \mathbb{R}^{m \times m}, K_{ij} = k(x_i, x_j), L_{ij} = l(y_i, y_j), H_{ij} = \delta_{ij} - m^{-1}$, and k, l are kernel functions. $\|C_{xy}\|_{HS} = 0$ if and only if x and y are independent [33]. It is guaranteed to detect any existing dependence with a high probability as the sample size m is increased [33]. Other estimators of HSIC (the block-based estimator, the Nyström estimator and the random Fourier feature (RFF) estimator) have been studied in [79].

2.10 Summary

Causal inference is one of the data analysis methods to solve causal relationship problems by generating a causal graph from the dataset then analyzing the causal relations among variables. The main goal of learning causal graph from the datasets is finding as much as possible the true causal relations among variables from the equivalence graph. The

two main elements for learning causal graphs are the causal algorithms and the datasets. Structure learning can be implemented for learning causal graphs from datasets. Structure learning has two approaches: score-based learning and constraint-based learning.

Score-based learning methods, for example, Greedy Equivalence Search (GES), generate an equivalence causal graph from the data and compute a score optimizing the learned graphs. This method finds an equivalence graph that fits the data based on the optimum score. The challenge of this method is defining the best scoring function [42]. The GES is developed for learning causal graphs from datasets which have no latent variable. GES produces graphs represented by CPDAG.

Constraint-based learning methods generate causal graphs from the datasets using conditional independence tests [42]. Examples of constraint-based learning algorithms are the PC algorithm, FCI, and RFCI. The PC algorithm is a constraint-based causal algorithm that can be used to learn causal graphs from datasets with the assumption that there is no latent variable. The PC algorithm produces graphs represented by a CPDAG. FCI and RFCI can be applied to learn causal graphs from datasets that allow the presence of latent variables. They output learned graphs represented by a PAG. Theoretically, constraint-based structure learning can be applied to learn causal graphs from the datasets if there is an appropriate conditional independence test. Structural Hamming Distance is a method used to measure the mismatch between two graphs originally developed for PDAGs.

A dataset containing discrete and continuous variables is called mixed data. A dataset containing no missing values is complete data. Missing value datasets (incomplete datasets) are datasets that have some missing values. Datasets containing missing values can be categorized as MAR, MCAR, and MNAR.

The conditional independence test for Gaussian data uses partial correlation. Partial correlation can be computed from a correlation matrix. This conditional independence test applies Fisher's z-transform. Distance correlation and HSIC are other methods to measure the dependence between random variables.

Chapter 3

Learning Causal Graphs from Mixed Data

3.1 Introduction

Learning causal graphs from mixed data is an interesting issue. Developing a new causal algorithm for mixed data needs more attention. Methods have been developed to learn causal graphs from mixed data using the existing algorithms (e.g. PC algorithm and FCI). The first approach is developing a method to compute a matrix that looks like a correlation matrix from mixed data then uses this matrix as an input for a conditional independence test. This idea was successfully implemented for Rank PC [53] and adopted in the Copula PC [24]. Harris and Drton improved the PC algorithm to non-parametric Gaussian (nonparanormal) models [53]. Let $f = (f_v)_{v \in V}$ be a collection of strictly increasing functions of function $f_v : \mathbb{R} \to \mathbb{R}$ and let $\Sigma \in \mathbb{R}^{V \times V}$ be a positive definite correlation matrix. The nonparanormal distribution NPN (f, Σ) is the distribution of the random vector $(f_v(Z_v))_{v \in V}$ for $(Z_v)_{v \in V} \sim N(0, \Sigma)$. Harris and Drton applied the PC algorithm by replacing the Pearson correlation matrix with rank-based measures of correlation; their method is called Rank PC (RPC). RPC works well for normal data and considerably better for non-normal data. The second approach is developing a conditional independence test for mixed data. This approach was successfully implemented in [58] and [70].

The PC algorithm and FCI are computationally feasible. PC and FCI are not restricted for specific kinds of datasets as long as there is a suitable conditional independence test for those datasets. Hence, it is beneficial to extend those algorithms so that they can be used to learn causal graphs from mixed data. One of the challenges of learning causal

graphs from mixed data is how to treat different data types without discriminating one or another.

This study proposes to use kernel alignment to compute a kernel alignment matrix from mixed data so that this matrix can be an input for the conditional independence test in the PC, FCI, and RFCI. This approach is similar to the method used in the Copula PC [24] [26]. The goal of this research is to develop an easy method to handle mixed data so that it can be used to learn causal graphs using PC, FCI, and RFCI. The improvement of the proposed method from the Copula PC algorithm is to treat categorical variables as continuous, binary, and ordinal variables. The kernel-based approach is not restricted by the type of the data distribution. The kernel-based approach is a general model that can be applied to any data type as long as there exists a suitable kernel function for the related data type.

3.2 Related Works in Learning Causal Graphs

3.2.1 Measuring Dependence using Kernel-based Approach

Kernel methods have been used to measure conditional (in)dependence. Kernel Generalized Variance (KGV) was developed to measure conditional dependence and it can be used to learn a hybrid network from discrete and continuous variables [7]. KGV allows discrete and continuous variables to be treated as Gaussians obtained from Mercer kernels in a feature space. Kernel Canonical Correlation (KCC) was proposed as a measure of independence [6]. Gretton et al. proposed Kernel Mutual Information (KMI) to measure the degree of independence of random variables [34]. Sun et al. developed a causal learning method for discrete and continuous variables by measuring the strength of statistical dependencies in terms of the Hilbert-Schmidt norm of kernel-based cross-covariance operators and used incomplete Cholesky decomposition [65]. Fukumizu et al. proposed a method to measure conditional dependence using kernels based on a normalized cross-covariance operator on reproducing kernel Hilbert spaces [29]. The kernel PC (kPC) implements a kernel-based conditional dependence measure similar to HSIC in the first step to identify the Markov equivalence class [67]. The centered kernel target alignment (KTA) coefficient has been used to construct nonlinear canonical variables for multivariate functional data and is useful to investigate the dependency between two sets of variables [32].

3.2.2 Causal Algorithms for Mixed Data

Methods have been developed to handle mixed data for learning causal graphs. Conditional Gaussian (CG) score, Mixed Variable Polynomial (MVP) score, and Degenerate Gaussian (DG) score are score-based methods for learning DAG from mixed continuous and discrete variables [3] [4]. The other methods are constraint-based structure learning for mixed data. Tsagris et al. proposed the likelihood-ratio test based on an appropriate regression model then derived symmetric conditional independence tests [70]. This test only works well when certain conditions hold. Raghu et al. proposed a conditional independence test based on linear and logistic regression to handle mixed data for causal discovery [58]. It was implemented together with the modification of FCI for the causal discovery of latent variables from mixed data. Raghu et al. assumed linear and logistic regression were accurate models of the interactions between continuous and discrete variables, but it is not clear how well these assumptions will hold in certain continuous nonlinear cases [58]. The Copula PC algorithm is a method for causal discovery from mixed data with an assumption that the data is drawn from a Gaussian copula model [24]. The Copula PC algorithm implements Gibbs sampling based on the extended rank likelihood, then estimates a scale matrix and degrees of freedom from the Gibbs samples. The scale matrix substitutes for a correlation matrix and the degrees of freedom acts as the effective number of data points for the conditional independence test. The Copula PC algorithm can be used to learn causal graphs from mixed data containing binary, ordinal, and continuous variables but is not appropriate for non-binary categorical variables whose values cannot be ranked, such as blood type.

3.3 Kernel Function and Kernel Alignment for Learning Causal Graphs from Mixed Data

In this research, we use the standard conditional independence test for Gaussian data which requires *partial correlations* [39]. Partial correlations can be computed from a correlation matrix [48]. We propose an easy way to compute a correlation matrix from mixed data using kernels. This approach is inspired by learning graphical models using KGV [7]. The KGV method was proposed by Bach and Jordan who map data into a feature space using a set of Mercer kernels, with different kernels for different data types [7]. They then treat all data equally as Gaussian in the feature space. Suppose $X_1, ..., X_m$ are m

random variables with values in space $\mathcal{X}_1, ..., \mathcal{X}_m$. A Mercer kernel k_i is assigned to each input space $\mathcal{X}_i$, with feature space F_i and feature map Φ_i. The random vectors of feature images $\phi = (\phi_1, ..., \phi_m) \triangleq ((\Phi_1(X_1), ..., \Phi_m(X_m)))$ have a covariance matrix C defined by blocks. Block C_{ij} is the covariance matrix between $\phi_i = \Phi_i(X_i)$ and $\phi_j = \Phi_j(X_j)$. Suppose $\phi^G = (\phi_1^G, ..., \phi_m^G)$ denotes a jointly Gaussian vector with the same mean and covariance as $\phi = (\phi_1, ..., \phi_m)$. The vector ϕ^G will be used as the random vector on which the learning of the graphical model structure is based. KGV is a general framework that can be applied to any type of variable [7].

Computing sample covariances using the kernel trick need high computation, and for efficient implementation it uses incomplete Cholesky decomposition. Our idea is to transform each variable in the mixed data using a suitable kernel function for each data type into a *single* Gaussian variable in the feature space. We then use *kernel alignment* to compute pairwise 'covariances' and thus construct a pseudo-covariance matrix. In this way we obtain a pseudo-correlation matrix which can be used for conditional independence tests in the normal way. The prefix 'pseudo-' is used to emphasize that the resulting matrix is not a correlation matrix for the original variables.

3.3.1 Kernel Function

A *kernel* is a function κ that for all $x, z \in X$ satisfies $\kappa(x, z) = \langle \phi(x), \phi(z) \rangle$, where ϕ is a mapping from X to an (inner product) feature space F, $\phi : x \to \phi(x) \in F$ [61]. Given a set of vectors $S = \{x_1, ..., x_\ell\}$, the *Gram matrix* is an $\ell \times \ell$ matrix G whose entries are $G_{ij} = \langle x_i, x_j \rangle$. Using a kernel function κ to evaluate the inner products in a feature space with feature map ϕ, the associated Gram matrix has entries $G_{ij} = \langle \phi(x_i), \phi(x_j) \rangle = \kappa(x_i, x_j)$. In this case the matrix is often referred to as the *kernel matrix*. The example of kernel functions are the RBF kernel and the Categorical kernel. RBF Kernel and Categorical kernel can be computed using equation 3.1 [61] and equation 3.2 [8], respectively.

RBF Kernel:

$$\kappa(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), \sigma > 0 \tag{3.1}$$

Categorical Kernel:

$$\kappa(z_i, z_j) = \begin{cases} h_\theta(P_Z(z_i)) & \text{if } z_i = z_j \\ 0 & \text{if } z_i \neq z_j. \end{cases} \tag{3.2}$$

P is probability and $h_\theta(.)$ is a function that depends on the parameter θ. It is defined as $h_\theta(z) = \left(1 - z^\theta\right)^{1/\theta}, \theta > 0$ [8]. Note that our method is not restricted to these particular choices of kernel.

The kernel matrix consists of the scalar product of the data points in feature space. In the feature space, the matrix K is called 'ill-condition' if it has about the same value because the origin is far away from the convex hull of the data [47]. Centering data is applied to solve the 'ill-condition' problem. Centering the data in the feature space can be done by centering the kernel matrix [61]. The goal of centering data is transferring the origin of the feature space to the center of mass of the samples. The centered kernel matrix $\tilde{K}$ can be computed using equation 3.3, where K is kernel matrix size $\ell \times \ell$ and j is the whole 1st vector [61].

$$\tilde{K} = K - \frac{1}{\ell}\sum_{i=1}^{\ell} jj'K - \frac{1}{\ell}Kjj' + \frac{1}{\ell^2}(j'Kj)jj' \tag{3.3}$$

In the experiment, we use the RBF kernel and Categorical kernel if the dataset has no missing values. The RBF kernel is one of the common and popular kernel functions in the Machine Learning area. Another alternative kernel function that popular to use is the Polynomial kernel [61]. There are not many choice kernel functions for categorical variables, the Categorical kernel function is reasonable and easy to implement. The proposed method is not restricted to the choices of kernel functions. The combination of RBF kernel and Categorical kernel is not mandatory. Further investigation in learning graphs from mixed datasets containing missing values (see Chapter 4) implements kernel extension.

3.3.2 Positive Semi-definite Matrix

A *square matrix* is a matrix with the same number of rows and columns. A *symmetric matrix* M is a square matrix whose value $M_{ij} = M_{ji}$, thus the transpose matrix $M^T = M$. A *symmetric matrix* is *positive semi-definite* if it has eigenvalues that are all non-negative, equivalently $v'Mv \geq 0, v \neq 0$ [61]. Eigenvalues are a set of scalars associated with a linear transformation.

3.3.3 Kernel Alignment

Given a sample $S = \{x_1, ..., x_m\}$, the inner product between two kernel matrices is $\langle K_1, K_2 \rangle = \sum_{i,j=1}^{m} K_1(x_i, x_j) K_2(x_i, x_j)$. The alignment between kernel k_1 and k_2 of the sample S is defined as equation 3.4 where K_i is the kernel matrix of the sample S using kernel function k_i [23] [22].

$$\hat{A}(S, k_1, k_2) = \frac{\langle K_1, K_2 \rangle}{\sqrt{\langle K_1, K_1 \rangle \langle K_2, K_2 \rangle}} = \frac{\langle K_1, K_2 \rangle}{\| K_1 \| \| K_2 \|} \tag{3.4}$$

The centered kernel alignment was introduced by Cortes et al. (2012) [21]. Let $K_1 \in \mathbb{R}^{n \times n}$ and $K_2 \in \mathbb{R}^{n \times n}$ be two kernel matrices such that $\|\tilde{K_1}\|_F \neq 0$ and $\|\tilde{K_2}\|_F \neq 0$. The centered kernel alignment between K_1 and K_2 is defined by equation 3.5, where $\tilde{K_1}$ and $\tilde{K_2}$ are the centered kernel matrices [32].

$$A_c(K_1, K_2) = \frac{\langle \tilde{K_1}, \tilde{K_2} \rangle}{\|\tilde{K_1}\|_F \|\tilde{K_2}\|_F} \tag{3.5}$$

We use kernel alignment as follows. Figure 3.1 shows the kernel alignment approach to compute a kernel alignment matrix from the data. Suppose, a mixed dataset consists of two variables (X and Y) and ℓ data point, i.e., $X = \{x_1, x_2, ..., x_\ell\}$ and $Y = \{y_1, y_2, ..., y_\ell\}$. We compute kernel matrices K_X and K_Y using kernel functions k_1 and k_2, respectively. Kernel matrices K_X and K_Y correspond to variables X and Y, respectively. $K_X(i, j)$ can be thought of as the similarity between x_i and x_j, the ith and jth observed values of X. It is also the inner product of the x_i and x_j in the feature space. In the same way K_Y encodes similarities between observed values of Y and contains inner products in the feature space for Y.

The alignment $A(K_X, K_Y)$ is built from the inner product between K_X and K_Y (and so is a (normalised) inner product of inner products). Equation 3.6 defines kernel alignment.

$$\begin{aligned} A(K_X, K_Y) &= \frac{\langle K_X, K_Y \rangle}{\sqrt{\langle K_X, K_X \rangle \langle K_Y, K_Y \rangle}} \\ &= \frac{\langle K_X, K_Y \rangle}{\|K_X\| \|K_Y\|} \\ &= \frac{\sum_{i,j=1}^{n} k_1(x_i, x_j) k_2(y_i, y_j)}{\sqrt{(\sum k_1(x_i, x_j) k_1(x_i, x_j))(\sum k_2(y_i, y_j) k_2(y_i, y_j))}} \end{aligned} \tag{3.6}$$

Suppose a dataset $D = \{V_1, .., V_p\}$ consists of p variables and a set of kernel matrices $K = \{K_1, ..., K_p\}$ is computed from those variables. The kernel alignment matrix A is a $p \times p$ Gram matrix where each entry $A(s, t)$ is an inner product of the two vectors

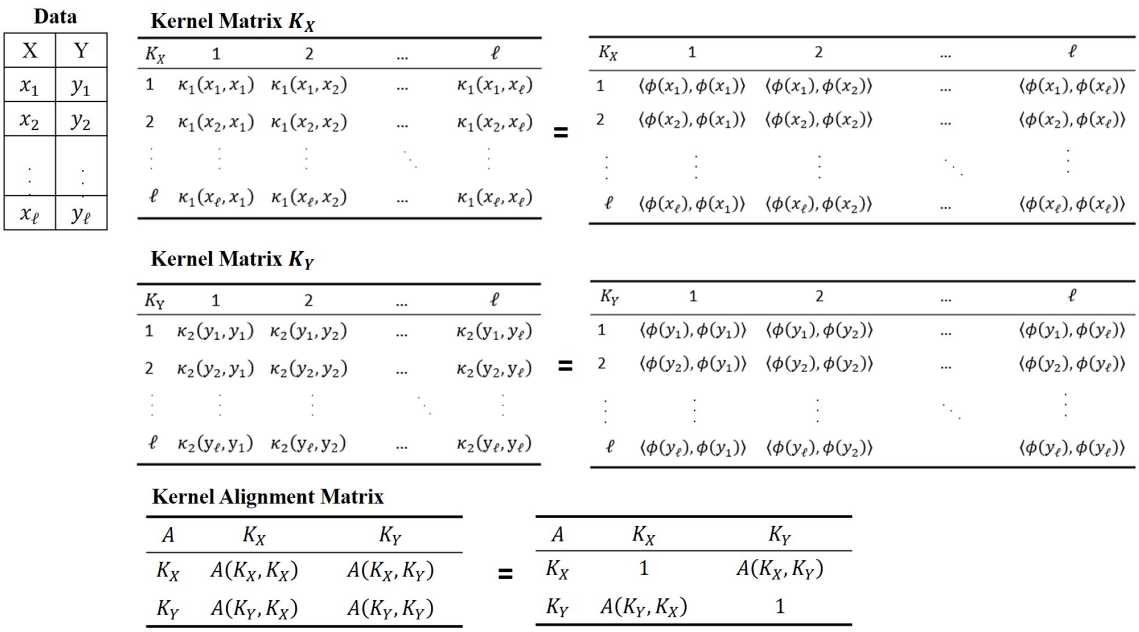
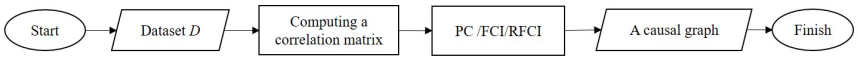


Figure 3.1: Kernel Alignment Approach

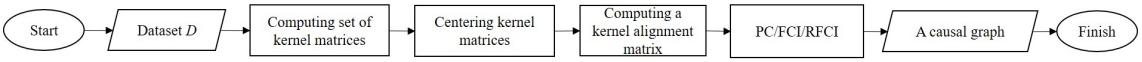
produced by flattening the kernel matrices K_s and K_t. A Gram matrix is a positive semi-definite matrix [61]. Every positive semi-definite matrix is a covariance matrix, so a kernel alignment matrix A is a covariance matrix. Given any covariance matrix, it is possible to construct a Gaussian distribution with that covariance matrix. Thus the use of kernels in this proposed method can be viewed as a way of (implicitly) generating a Gaussian distribution whose covariance matrix is the kernel alignment matrix. This proposed method differs from [7] in that the (implicitly constructed) Gaussian always has the same dimension as the original data (since unlike them we use kernel alignment), but in common with Bach and Jordan there is a separate kernel for each of the original variables. Like Bach and Jordan in [7], we do not view this treating-variables-separately approach as unduly restrictive. (Note that for Copula PC the mapping is in the other direction: the observed variables are viewed as the result of a mapping from some latent Gaussian distribution.) The alignment between a kernel matrix and itself is $A(s,s) = A(K_s,K_s) = \frac{\langle K_s,K_s\rangle}{\sqrt{\langle K_s,K_s\rangle\langle K_s,K_s\rangle}} = 1$, so that $\sigma_s = \sqrt{1} = 1$. $A(s,s)$ and $A(t,t)$ can be viewed as variance σ_s^2 and σ_t^2, respectively. Thus, the entry of the correlation matrix can be defined as $\rho_{st} = \frac{\sigma_{st}}{\sigma_s\sigma_t} = \frac{A(K_s,K_t)}{A(K_s,K_s)A(K_t,K_t)} = \frac{A(K_s,K_t)}{(1)(1)} = A(K_s,K_t)$. Hereafter, this correlation matrix is called a pseudo-correlation matrix A. Note that the entries in any kernel alignment matrix are in the range $[0,1]$ [61].

3.4 Kernel Alignment Substitutes Pearson Correlation for Conditional Independence Test

A correlation matrix and the number of data points are inputs for the conditional independence test in PC, FCI, and RFCI. Figure 3.2 (A) shows a flowchart for learning causal graphs using the original PC algorithm/FCI/RFCI. The original PC algorithm, FCI, and RFCI computes the correlation matrix from the data then uses this matrix as an input to generate causal graphs. Figure 3.2 (B) describes the procedure to learn causal graphs using KAPC/KAFCI/KARFCI. In KAPC, KAFCI, and KARFCI, a kernel matrix is computed for each variable using a suitable kernel function. We apply a *centering kernel matrix* using equation 3.3 to compute a centered kernel matrix. The kernel alignment matrix is computed from the centered kernel matrices. The kernel alignment matrix is implemented to substitute the correlation matrix and it is used as an input for conditional independence test in the PC, FCI, and RFCI. The kernel alignment matrix substitutes the correlation matrix then the partial correlation is computed from the kernel alignment matrix.



(A) Learning a causal graph from a dataset using PC algorithm/FCI/RFCI



(B) Learning a causal graph from a dataset using Kernel Alignment PC algorithm/FCI/RFCI

Figure 3.2: (A) The original PC algorithm/FCI/RFCI and (B) KAPC/KAFCI/KARFCI

3.5 The Connection of Kernel Alignment, HSIC, Pearson Correlation, Distance Correlation, and Cosine Similarity

Suppose, X and Y are two random vectors, then the *inner product* $\langle X, Y \rangle = \sum_i x_i y_i$. Using the procedure to find the angle θ between two vectors X and Y, the uncentered correlation coefficient is identical with the cosine similarity and it can be computed using equation 3.7. Meanwhile, Pearson correlation ρ_r of variables X and Y can be computed using equation 3.8. The Pearson correlation ρ_r is a cosine between X and Y centered by their mean. Therefore the Pearson correlation is a centered correlation coefficient.

The alignment satisfies $-1 \leq A \leq 1$ because it can be viewed as the cosine of the angle between ℓ^2-dimensional vectors of $\ell \times \ell$ matrices [61]. Figure 3.3 illustrates cosine similarity between two vectors V_{K_X} and V_{K_Y}, where V_{K_X} and V_{K_Y} are vectors formed by flattening the kernel matrix K_X and K_Y, respectively. K_X and K_Y are positive semi-definite matrices, therefore the value of $\langle K_X, K_Y \rangle \geq 0$ [61]. As a consequence, kernel alignment has a range of values $0 \leq A(K_X, K_Y) \leq 1$. According to equation 3.6, kernel alignment $A(K_X, K_Y)$ is equivalent to $\cos(K_X, K_Y)$. Hence, kernel alignment can be considered a correlation coefficient. Rearranging equation 2.19 when $\mathcal{V}^2(X)\mathcal{V}^2(Y) > 0$ [66], we can define the empirical Distance correlation $\mathcal{R}_n^2$ as a cosine similarity function in equation 3.9. Pearson Correlation ρ_r, empirical Distance Correlation $\mathcal{R}_n$, and kernel alignment A can be defined as cosine similarity function: $\rho_r(X,Y) \equiv \mathcal{R}_n^2(X,Y) \equiv A(K_X, K_Y) \equiv \cos(X,Y)$.

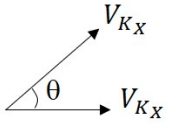
$$\cos(X,Y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}} = \frac{\langle X, Y \rangle}{\|X\|\|Y\|} \tag{3.7}$$

$$\begin{aligned} \rho_r(X,Y) &= \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2}\sqrt{\sum (y_i - \bar{y})^2}} \\ &= \frac{\langle x - \bar{x}, y - \bar{y} \rangle}{\|x - \bar{x}\|\|y - \bar{y}\|} = \cos(x - \bar{x}, y - \bar{y}) \end{aligned} \tag{3.8}$$

$$\begin{aligned} \mathcal{R}_n^2 &= \frac{\frac{1}{n^2}\sum_{k,l=1}^n A_{kl} B_{kl}}{\sqrt{\frac{1}{n^2}\sum_{k,l=1}^n A_{kl}^2}\sqrt{\frac{1}{n^2}\sum_{k,l=1}^n B_{kl}^2}} \\ &= \frac{\frac{1}{n^2}\sum_{k,l=1}^n A_{kl} B_{kl}}{\left(\frac{1}{n}\sqrt{\sum_{k,l=1}^n A_{kl}^2}\right)\left(\frac{1}{n}\sqrt{\sum_{k,l=1}^n B_{kl}^2}\right)} \\ &= \frac{\sum_{k,l=1}^n A_{kl} B_{kl}}{\left(\sqrt{\sum_{k,l=1}^n A_{kl}^2}\right)\left(\sqrt{\sum_{k,l=1}^n B_{kl}^2}\right)} \\ &= \frac{\langle A_{kl}, B_{kl} \rangle}{\|A_{kl}\|\|B_{kl}\|} = \cos(A_{kl}, B_{kl}) \end{aligned} \tag{3.9}$$

$$K_X = \begin{bmatrix} m_{11} & \cdots & m_{1\ell} \\ \vdots & \ddots & \vdots \\ m_{\ell 1} & \cdots & m_{\ell\ell} \end{bmatrix} \quad \text{as vector } V_{K_X} = [m_{11} m_{12} m_{13} \ldots m_{\ell\ell}]$$

$$K_Y = \begin{bmatrix} n_{11} & \cdots & n_{1\ell} \\ \vdots & \ddots & \vdots \\ n_{\ell 1} & \cdots & n_{\ell\ell} \end{bmatrix} \quad \text{as vector } V_{K_Y} = [n_{11} n_{12} n_{13} \ldots n_{\ell\ell}]$$



$$A\langle K_X, K_Y \rangle \equiv \cos\theta$$

Figure 3.3: Kernel Alignment as cosine of two vectors

The kernel alignment matrix has the same range of values as Distance correlation. In our case, kernel alignment A between two kernel matrices is similar to the distance between two vectors resulting from flattening those kernel matrices (see equation 3.6). However, kernel alignment values are connected to the angle between pairs of vectors resulting from flattening kernel matrices (see equation 3.6). In contrast, the Distance correlation between two variables is related to their Euclidean distance apart.

Suppose, K and L are two kernel matrices. The transpose matrix K^T of the kernel matrix K has the same values as itself. The $tr(KL)$ is obtained by removing H and $(m-1)^{-2}$ from equation 2.22 (see Chapter 2.9.3). The $tr(KL) = \langle K^T, L \rangle$ implies the connection of the HSIC and the kernel alignment.

We implement kernel alignment to approximate the estimated values of HSIC. The greater the sample size m, the closer the kernel alignment value is to the estimated value HSIC and to the true values of HSIC. The computation cost of HSIC is expensive and a low-rank decomposition of the Gram matrices via incomplete Cholesky decomposition is needed to compute an accurate approximation to HSIC [33]. It means that in the implementation we never actually use equation 2.22. Implementing the incomplete Cholesky factorization for efficient implementation is not numerically stable [67]. The kernel alignment offers a faster computation to approximate the HSIC.

3.6 Generating Mixed Data

We use two different methods to generate mixed datasets. The first method is Conditional Linear Gaussian (CLG) models. The other method is generating continuous datasets, then discretizing some variables. The mixed datasets for simulation in learning causal graphs are generated using forward sampling from a hybrid network and we apply Conditional Linear Gaussian (CLG) models [42]. First we generate random DAGs containing 10, 20 and 30 nodes. Each variable is given a data type. For each different group of nodes, there are 10 different graphs and each graph produces five different datasets. We generate two groups of datasets according to the kind of data types: mixed dataset 1 (binary, ordinal and continuous variables) and mixed dataset 2 (binary, ordinal, categorical and continuous variables). The mixed datasets generated by discretizing some variables from continuous datasets are used for conditional independence test experiments.

3.7 Experimental Design for Learning Causal Graphs from Mixed Data

The experiment is divided into 3 different groups: experiment using benchmark datasets, experiment using mixed data 1, and experiment using mixed data 2. KAPC is run to generate the learned graphs from benchmark datasets. The goal of this experiment is to ensure that the proposed method works well to learn graphs from homogeneous data as well as mixed data. In the experiment using mixed data 1, it uses datasets containing binary, ordinal and continuous variables. In the experiment using mixed data 2, it uses mixed data containing categorical, binary, ordinal and continuous variables. One variable is deleted from datasets of 10 variables. Four and eight variables are deleted from data sets of 20 and 30 variables, respectively. Those deleted variables represent latent variables. After deleting some variables from the datasets, the remaining variables are used to learn causal graphs using the KAFCI and KARFCI.

The experiments use the PC algorithm, FCI, and RFCI in *pcalg* [40]. Structural Hamming Distance (SHD) is used to measure the quality of the learned graphs. The SHD score is obtained by computing the mismatch between the true graph and the learned graph, so the lower SHD score means that the learned graph has less mismatch. The normalized SHD score is computed using equation 2.9 [46] (see Chapter 2.6). The normalized SHD has the same meaning as the original SHD where the lower SHD score means that the learned graph has less mismatch to the true graph. The normalized SHD score is used to make a fair comparison of quality learned graphs with a different number of nodes.

The experiments implement a kernel alignment matrix and the number of data points as the input for conditional independence test in the PC, FCI and RFCI. We name our method Kernel Alignment PC (KAPC), Kernel Alignment FCI (KAFCI), and Kernel Alignment RFCI (KARFCI). The number of data points refers to the data points in each variable, where all variables in a dataset have the same data points. All datasets for this experiment have no missing values. We also run some existing applications: Copula PC [24], greedy search Hill-Climbing from bnlearn [60], *deal* [11], and PC MXM [70].

3.8 Experimental Results

3.8.1 Kernel Alignment Matrix for Conditional Independence Test

The goal of the experiment of conditional independence test (CI test) using the kernel alignment matrix is to analyze the impact of choosing kernel parameter values for the CI test result. We apply the kernel alignment matrix for conditional independence tests for simple graphs. The datasets for simulations are generated based on the simple graphs in Figure 3.4. The datasets are grouped into 7 different cases. The reason for creating 7 different cases for 3 graphs is to create various types of datasets. The various datasets are beneficial to test our proposed method in different conditions. The 7 cases consist of datasets containing homogeneous variables (all discrete (Case 1) and all continuous (Case 4)) and mixed datasets (Case 2, 3, 5, 6, and 7). The mixed datasets are distinguished into two groups: mixed data from graphs where no discrete child has continuous parents (graph G1 and G2 Case 2, 3 and G3 Case 2, 3, 7) and where a discrete child has a continuous parent (graph G1 and G2 Case 5, 6, 7 and G3 Case 5, 6). Table 3.1 shows the 7 cases of datasets based on the data type of each variable. Figure 3.5 shows the graphs with data types for each variable. The concern of this experiment is using the kernel alignment to substitute the correlation matrix for conditional independence test using equation 2.16 (see Chapter 2.9.1). Therefore, the conditional independence tests are run to test three different things:

- $X \perp Y \mid Z$.
- $X \perp Z \mid Y$.
- $Y \perp Z \mid X$.

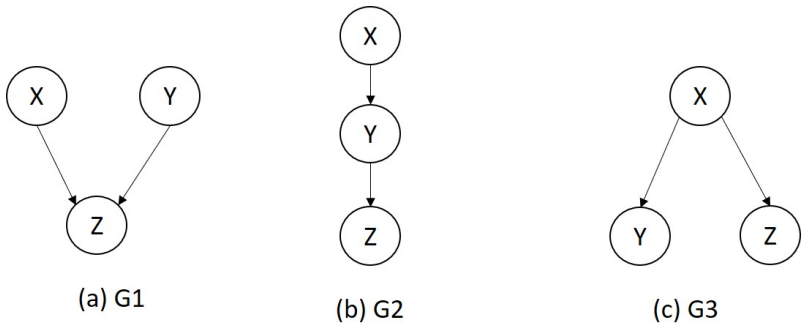


Figure 3.4: Simple graphs for case study

Table 3.1: Data types for each variable in the datasets

No	Cases	X	Y	Z
1	Case 1	Discrete	Discrete	Discrete
2	Case 2	Discrete	Discrete	Continuous
3	Case 3	Discrete	Continuous	Continuous
4	Case 4	Continuous	Continuous	Continuous
5	Case 5	Continuous	Discrete	Discrete
6	Case 6	Continuous	Continuous	Discrete
7	Case 7	Discrete	Continuous	Discrete

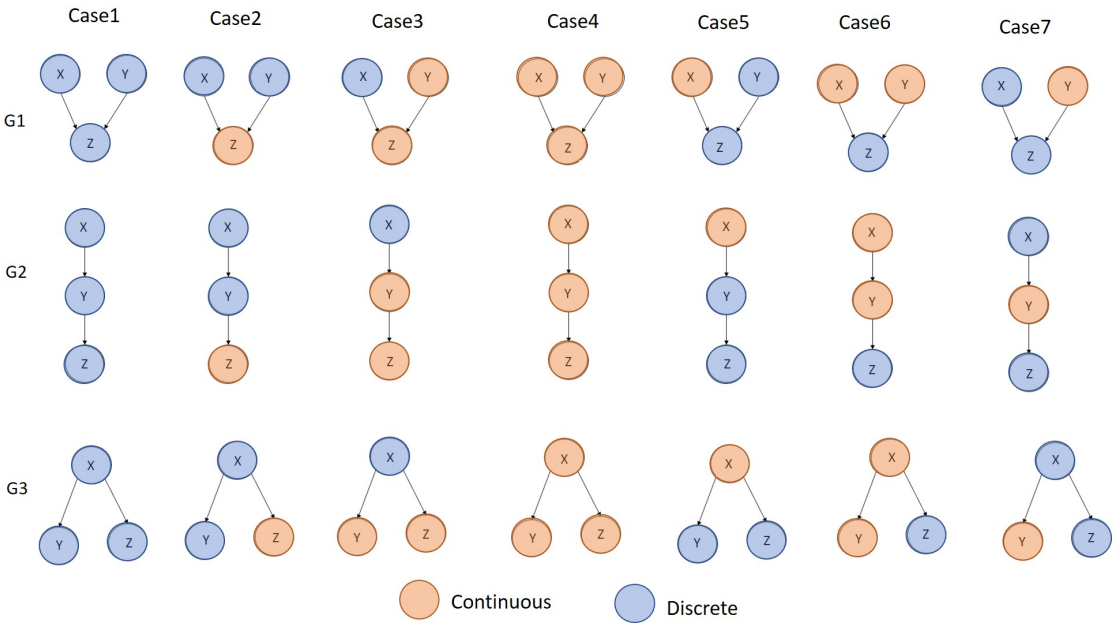


Figure 3.5: The three graphs with their data type for each node

For each case of each graph, we generate 10 different datasets. Forward sampling from the hybrid network is used to generate datasets when a discrete variable has no continuous parents. The datasets of G1, G2, G3 for Case 1, Case 2, Case 3, and G3 for Case 7 are generated using the forward sampling from a hybrid network and it implements Conditional Linear Gaussian models. We define discrete variables $D = \{d_1, d_2, ..., d_n\}$ with the probability $p = \{p_1, p_2, ..., p_n\}$, where $\Sigma_i^n p_i = 1$. Let X and Y be continuous variables, the CPD for $P(X|D)$ and $P(Y|D, X)$ can be defined by equation 3.10 and 3.11, respectively, where $b \neq 0$ and e is a Gaussian coefficient.

$$P(X|D) = \begin{cases} \mathcal{N}(\mu_i; \sigma_i) & D = d_i \\ ... & \\ \mathcal{N}(\mu_n; \sigma_n) & D = d_n; \end{cases} \tag{3.10}$$

$$P(Y|D,X) = \begin{cases} \mathcal{N}(b_i X + e_i; \sigma_i) & D = d_i \\ ... & \\ \mathcal{N}(b_n X + e_n; \sigma_n) & D = d_n; \end{cases} \tag{3.11}$$

In a condition where the discrete variable has a continuous parent and/or continuous and discrete parents, we generate datasets consisting of all continuous variables then discretize the particular variable. Figure 3.6 shows the formulae to generate continuous datasets for each graph, where μ = mean, σ = deviation standard, $b \neq 0$, and e = Gaussian coefficient. The method to discretize the variables can be explained as follows. Suppose $X = \{x_1, x_2, ..., x_n\}$ is a continuous variable, we generate a binary variable Z with entries:

$$Z_i = \begin{cases} v_1 & \text{if } X_i \leq Q_2(X) \\ v_2 & \text{otherwise} \end{cases}$$

We generate a non-binary variable Z with entries:

$$Z_i = \begin{cases} v_1 & \text{if } X_i \leq Q_1(X) \\ v_2 & \text{if } X_i > Q_1(X) \text{ and } X_i \leq Q_2(X) \\ v_3 & \text{if } X_i > Q_2(X) \text{ and } X_i \leq Q_3(X) \\ v_4 & \text{otherwise} \end{cases}$$

The $Q_1(X), Q_2(X), Q_3(X)$ refer to the first quartile, median and the third quartile of variable X, respectively. We use quantiles as thresholds when discretizing the variable to minimize destroy the probabilistic structure of the data. For instance, in graph G2 Case 5 (see Figure 3.5), first, we generate a dataset consisting of all continuous variables, then discretize variables Y and Z separately. We create two different types of discrete variables: binary and non-binary. The motivation to produce binary and non-binary is to ensure that our datasets are varied. The true distribution of the generated mixed data using this method is the distribution of the one that the dataset is sampling from.

We compute the kernel matrix for each variable in a dataset using a suitable kernel function for each data type. It implements RBF kernel for continuous data type and Categorical kernel for discrete data type. Tables B.1, B.2, and B.3 show a list of kernel parameter values σ for RBF kernel and θ for Categorical kernel. The kernel parameter

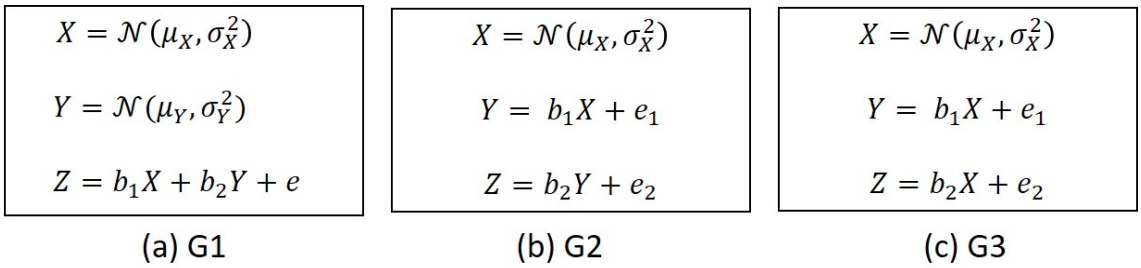


Figure 3.6: The formulae for generating continuous dataset

values are positive numbers, $\sigma > 0$ and $\theta > 0$, chosen randomly and we use 6 kernel parameter values for each case. The kernel alignment matrix A is computed from the kernel matrices. The kernel alignment matrix A and the number of data points are used as inputs for the conditional independence test. The number of data points that are used in this study is $N = \{1000, 3000, 5000\}$.

Table 3.2: Ground Truth

No	**Test Case**	**G1**	**G2**	**G3**
1	$X \perp Y \mid Z$	FALSE	FALSE	FALSE
2	$X \perp Z \mid Y$	FALSE	TRUE	FALSE
3	$Y \perp Z \mid X$	FALSE	FALSE	TRUE

		Actual class	
		TRUE	FALSE
Predicted	TRUE	True Positives	False Positives
	FALSE	False Negatives	True Negatives

(a)

		Actual class	
		TRUE	FALSE
Predicted	TRUE	True Positive Rate (TPR) $\frac{\Sigma \text{ True Positive}}{\Sigma \text{ Condition Postive}}$	False Positive Rate (FPR) $\frac{\Sigma \text{ False Positive}}{\Sigma \text{ Condition Negative}}$
	FALSE	False Negative Rate(FNR) $\frac{\Sigma \text{ False Negative}}{\Sigma \text{ Condition Postive}}$	True Negative Rate (TNR) $\frac{\Sigma \text{ True Positive}}{\Sigma \text{ Condition Negatie}}$

(b)

Figure 3.7: Confusion Matrix

The conditional independence tests implement Fisher's z-transform (see equation 2.16 in Chapter 2.9.1). The tests are run for $\alpha = \{0.01, 0.05, 0.1, 0.5, 0.7\}$. In a common situation, we usually use the values $\alpha = \{0.01, 0.05, 0.1\}$. We run the experiment using

$\alpha = \{0.5, 0.7\}$ to test the behaviour of our proposed method. The test result returns one of two classes $\{TRUE, FALSE\}$. For instance, in graph G3, it is true that $Y \perp Z \mid X$, so the actual class is TRUE. In graph G3, $X \not\perp Z \mid Y$ so the actual class for the test case "$X \perp Z \mid Y$" is FALSE. Table 3.2 shows the list of the actual classes. Given a generated dataset from graph G3, we run a test 'is $X \perp Z \mid Y$?'; the result might return TRUE or FALSE. The test result is called the predicted class. Figure 3.7 (a) shows the confusion matrix and it is utilized to analyze the experimental results. Figure 3.7 (b) shows the equations to compute True Positive Rate (TPR) and False Positive Rate (FPR). The graph G1 has no actual class TRUE because it has no variables that are conditionally independent to others, so there is no TPR value. It uses a confusion matrix to summarize the number of true positives, false positives, false negatives, and true negatives. To clarify the meaning of table 3.7 (a) and (b), we explain those terms as follows. The *True Positive* is defined as the actual class is TRUE and the test result is TRUE. The *False Positive* is defined as the actual class is FALSE and the test result is TRUE. The *False Negative* is defined as the actual class is TRUE and the test result is FALSE. The *True Negative* is defined as the actual class is FALSE and the test result is FALSE. The condition positive is defined as the number of the actual classes that are TRUE and the condition negative refers to the number of the actual classes that are FALSE.

The experimental result of the conditional independence tests using kernel alignment can be explained as follows. Tables B.4 - B.12 show the values of True Positive Rate (TPR) and False Positive Rate (FPR) for each experiment. P1-P6 in Tables B.4 - B.12 and Figures 3.8, 3.9, and 3.10 represent kernel parameter values (see Tables B.1, B.2, B.3). The ROC Curves in Figures 3.8, 3.9, and 3.10 show the performance of the kernel parameter values for each case. The line in the upper left in the ROC curve shows the suitable kernel parameter values because it represents these kernel parameter values producing a high TPR and low FPR. In general, the results shows that $\alpha = \{0.01, 0.05, 0.1\}$ produce a high True Positive Rate (TPR) and a low False Positive Rate (FPR). Meanwhile, the $\alpha = \{0.5, 0.7\}$ produce a low True Positive Rate (TPR) and a high False Positive Rate (FPR). The performance of the kernel-based approach is affected by the choice of kernel parameter values.

Tables B.4, B.5, and B.6 show the detailed TPR and FPR for graph G1. In graph G1, there is no 'TRUE' class for conditional independence among variables, so the TPR value is not available. We prefer to put (-) and do not write 0 to fill up the value of TPR in

the table of graph G1 in order to avoid the misleading perception. Figure 3.8 shows the ROC curve for graph G1. The CI test results show there are no kernel parameter values used in the experiments that produce FPR > 0.5 for Case 1 and Case 7 when $\alpha \leq 0.1$. At least one of six kernel parameter produce the FPR < 0.5 in Case 2, 3, 4, 5, and 6 when $\alpha \leq 0.1$. It means that there exist kernel parameter values suitable for the datasets so the kernel alignment matrix can be used to catch dependencies among variables. The ROC curve for graph G1 presents the best kernel parameter values when it produces the lowest FPR. The best kernel parameter values are displayed by the line at the bottom left.

Tables B.7, B.8, and B.9 show the detailed TPR and FPR for graph G2. The worst kernel parameter values produce zero TPR and higher FPR, for instance, P6 in Case 4. The different datasets on graph G2 need different kernel parameter values to produce precise CI test results. There exist at least one of 6 kernel parameter values that produce TPR > 0.5 and FPR < 0.5 when $\alpha \leq 0.1$ for each case in Graph 2, except Case 7. The high TPR and low FPR indicates that the kernel alignment is successfully applied to measure dependencies. Figure 3.9 shows the suitable kernel parameter values for each case. The lines in the upper left perform the suitable kernel parameter for the dataset.

Tables B.10, B.11, and B.12 show the detailed TPR and FPR for graph G3. The kernel parameter values used in the experiment of Graph G3 mostly produce TPR > 0.5 and FPR < 0.5 when $\alpha \leq 0.1$ except for Case 5. The optimum kernel parameter values are different for each dataset. Figure 3.10 shows the ROC curve and it visualizes the performance of kernel parameter values for Graph 3. The best kernel parameter values are displayed by the line in the upper left.

Implementing $\theta < 1$ for the Categorical kernel might produce very small values in the kernel matrix, almost zero, and/or zero. It can be used as a reason that $\theta < 1$ is less recommended. KAPC using the Categorical kernel with kernel parameter $\theta \geq 2$ for the datasets containing all discrete variables always produces the same learned graphs. The experimental result shows that the proper parameter values for the Categorical kernel are $\theta \geq 2$. Different kernel parameter values might generate different test results. If it uses the proper kernel parameter values, the test result is correct. The choices of kernel parameters that only small toy datasets have been considered in this research and the results obtained may not generalize to bigger datasets such as those found in real-world applications.

The more data points used in the experiment reduces the FPR. The experimental results for Case 7 graph G1 and G2 and Case 5 graph 3 produce higher FPR than other

cases. The similarity of those cases is the dataset for the experiments contain two variables discretized from continuous variables. We also run the experiment using the original continuous data and obtain a high TPR and low FPR. It shows that coarse discretization possibly makes it hard to detect conditional dependencies.

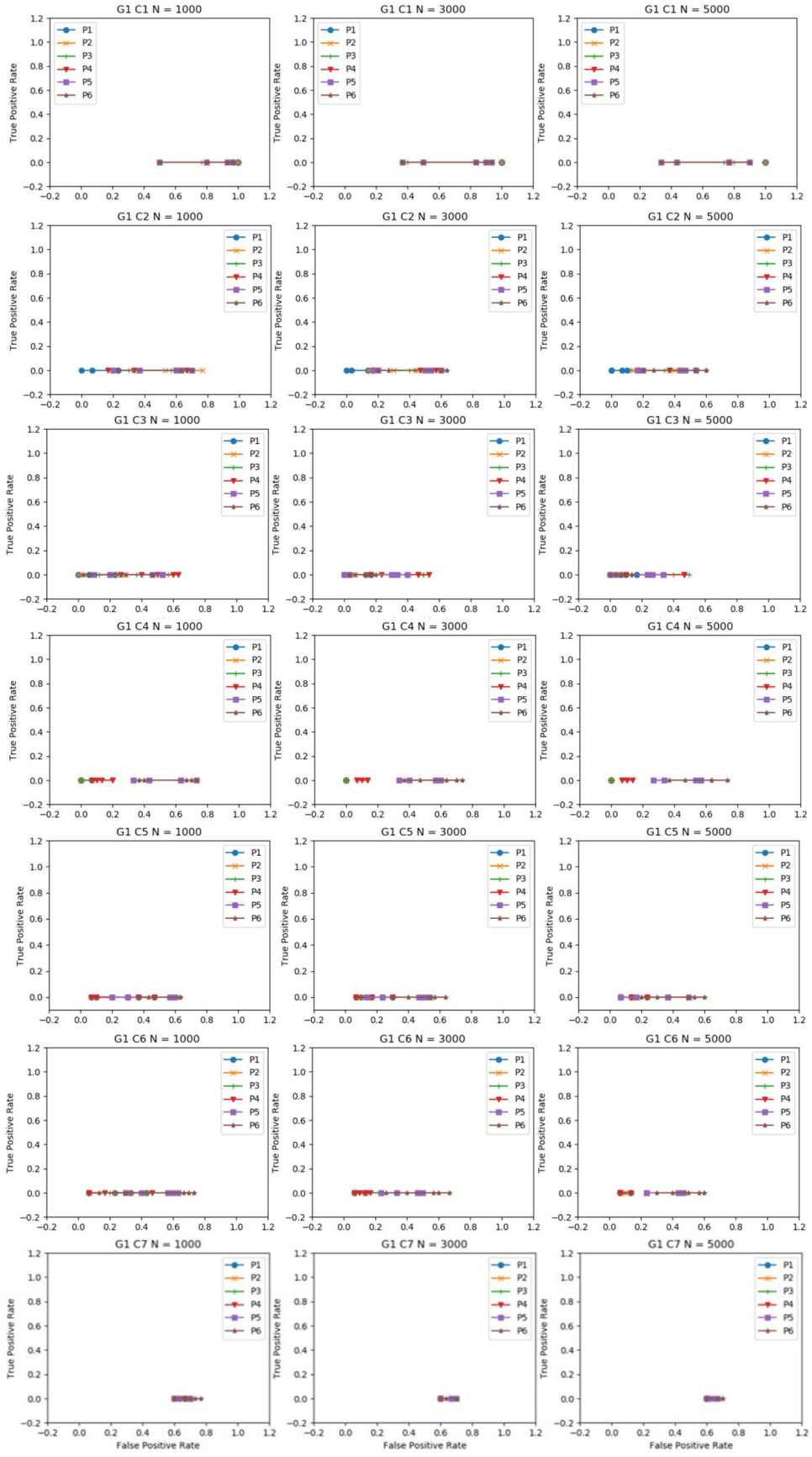


Figure 3.8: The ROC curve for Graph G1

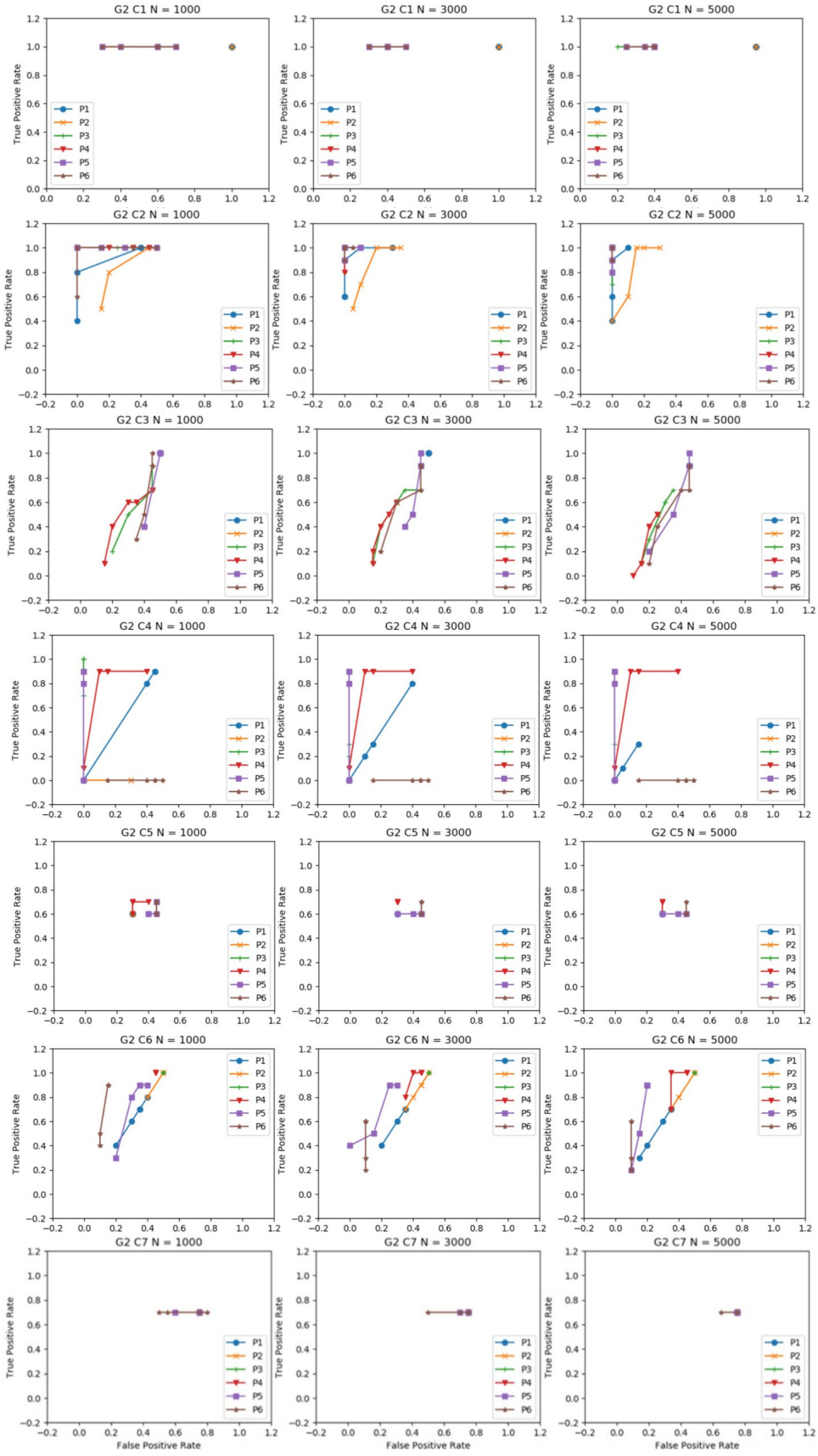


Figure 3.9: The ROC curve for Graph G2

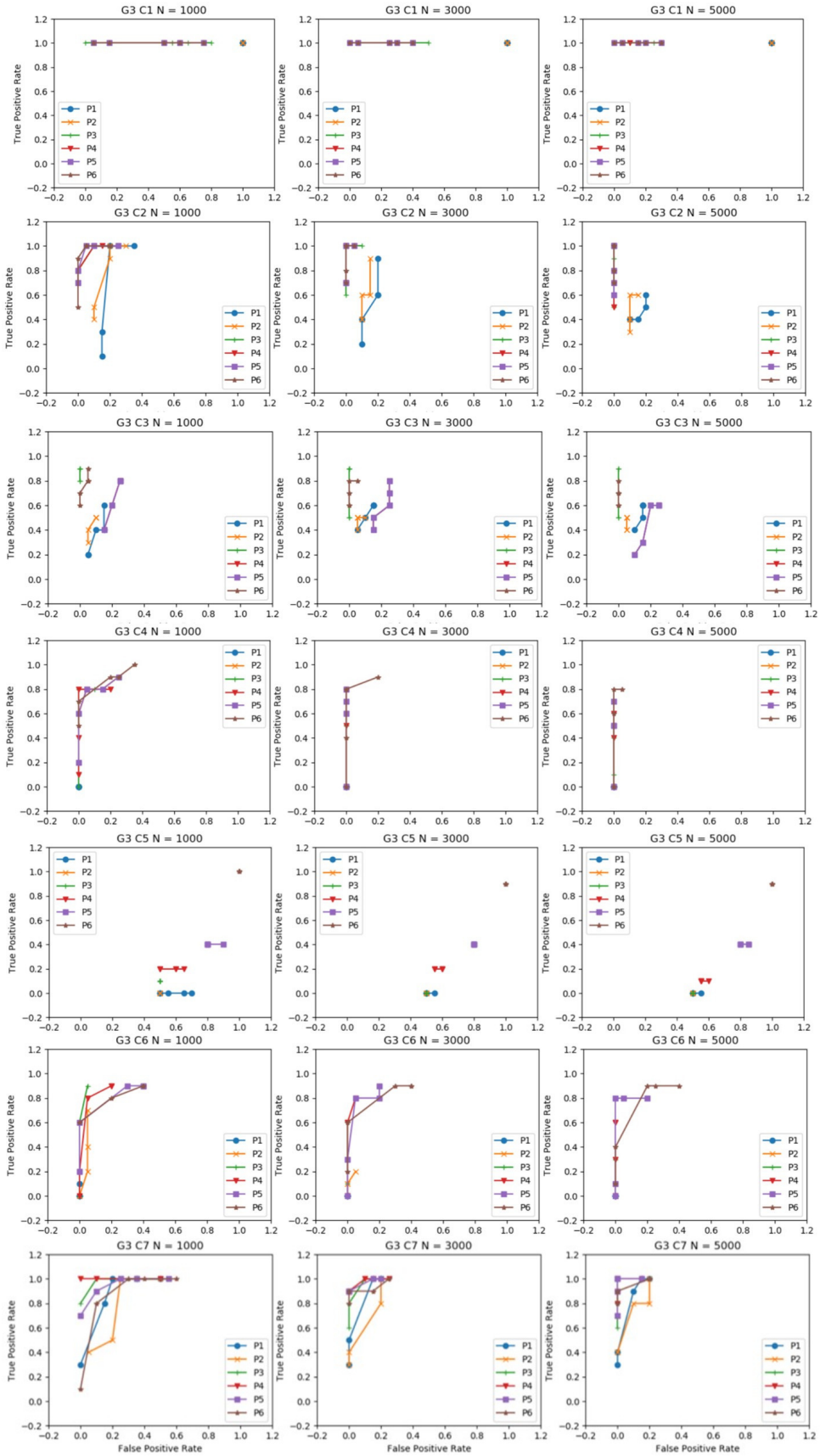


Figure 3.10: The ROC curve for Graph G3

3.8.2 Learning Causal Graphs from Benchmark Datasets

To further investigate the performance of the proposed method, we also ran experiments using the datasets *gaussian.test* (sampled from a Gaussian) and *clgaussian.test* from the R package bnlearn, and *gmD* (discrete variables) and *gmG* (Continuous variables) from the R package pcalg. The distribution from which *clgaussian.test* was sampled contains one Gaussian variable, 4 discrete variables (2 binary and 2 categorical variables) and 3 conditional Gaussian variables.

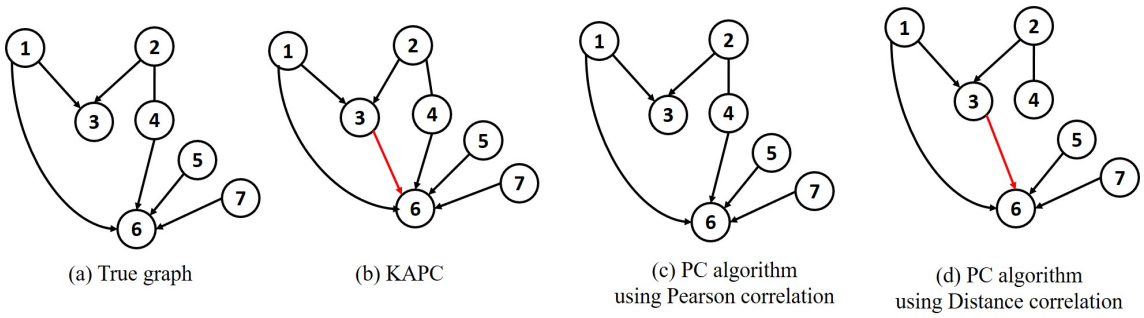


Figure 3.11: The true graph and learned graphs generated from *gaussian.test* dataset

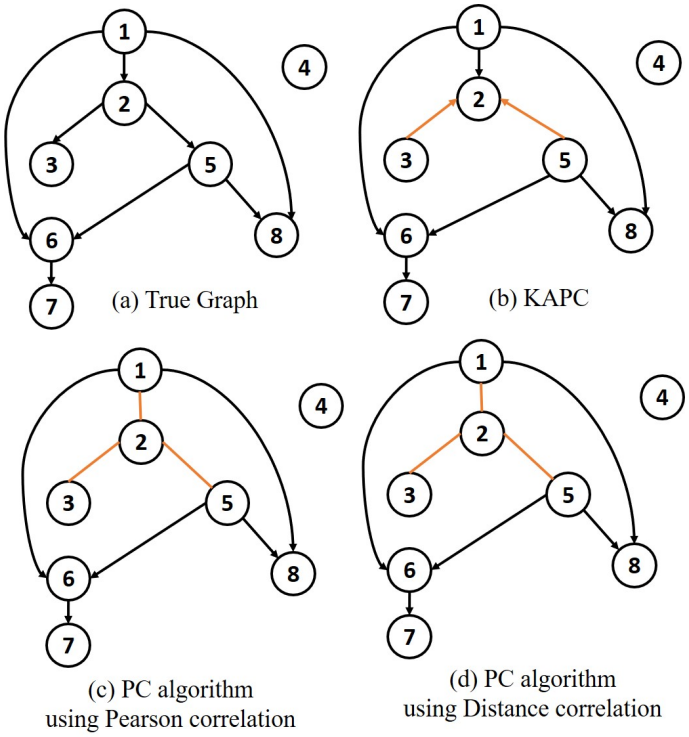


Figure 3.12: The true graph and learned graphs generated from *gmG* dataset

Figure 3.11 shows the learned graphs generated by the PC algorithm and KAPC using *gaussian.test* at $\alpha = 0.05$. KAPC implements the RBF kernel. KAPC successfully

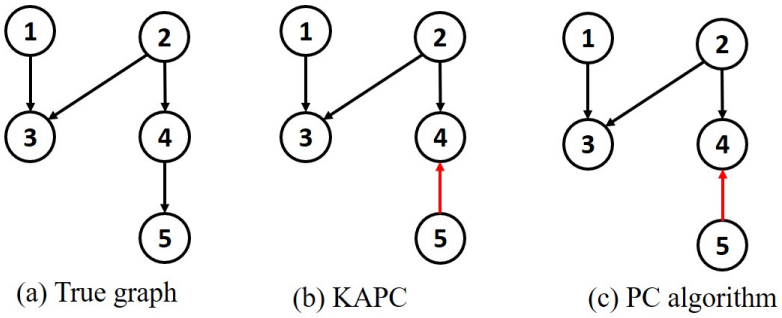


Figure 3.13: The true graph and learned graphs from *gmD* dataset

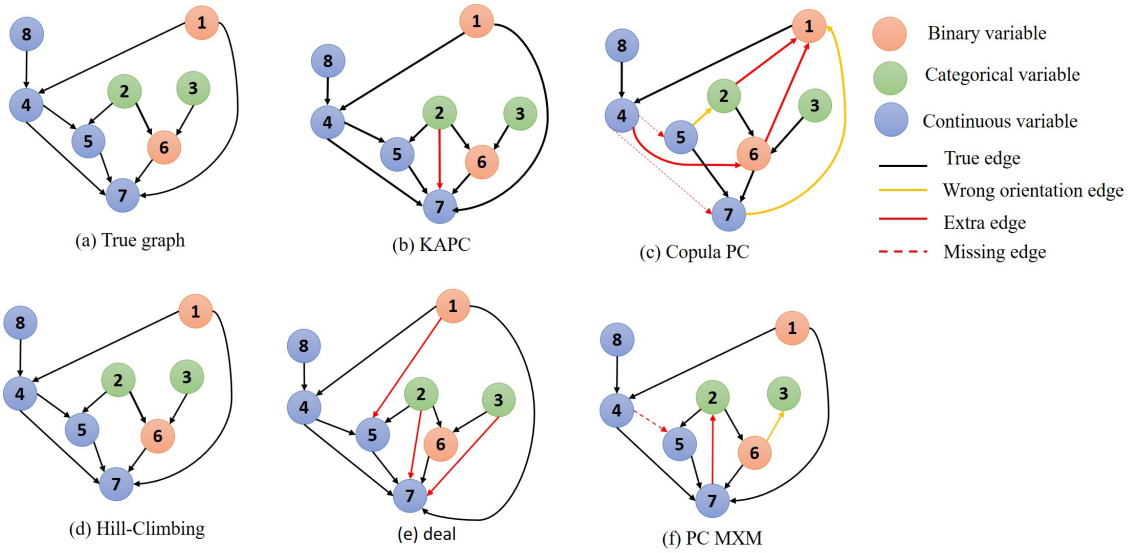


Figure 3.14: The true graph of *clgaussian.test* and learned graphs

generates all true edges but it produces one extra edge(red edge). The original PC using Pearson correlation produces the learned graph that exactly matches the true graph. The learned graphs obtained by KAPC and the PC using Distance correlation have the same graph structure. Figure 3.12 shows the true graph and learned graphs using dataset *gmG* at $\alpha = 0.01$. PC algorithm using Pearson correlation, PC algorithm using Distance correlation and KAPC generate the learned graphs which have the same skeleton to the skeleton true graph. Figure 3.13 shows the true graph and learned graphs generated from the *gmD* dataset at $\alpha = 0.1$. The original PC algorithm and KAPC produce the same graph skeleton as the true graph skeleton but they have one wrongly marked edge. KAPC creates a learned graph from *gmD* data using Categorical kernel (equation 3.2) with kernel parameter $\theta = 3$. Figure 3.14 shows the true graph and learned graphs generated from *clgaussian.test* data using KAPC, Copula PC, PC MXM at $\alpha = 0.05$, *deal* and Hill-Climbing (HC). KAPC implements the RBF kernel for continuous variables and

Categorical kernel for categorical variables. KAPC generates a learned graph containing 10 true edges and 1 extra edge. Meanwhile, Copula PC algorithm generates a learned graph containing 6 true edges, 2 wrong marked edges, 3 extra edges, and 2 missing edges. The experiment using *clgaussian.test* shows that Copula PC algorithm does not work well for mixed data containing categorical variables. Hill-Climbing (HC) generates the learned graph containing no error. *deal* generates a learned graph which has 10 true edges and 3 extra edges. PC MXM generated a learned graph containing 8 true edges, 1 missing edge, 1 extra edge and 1 wrong orientation edge.

3.8.3 Kernel Alignment PC Algorithm/FCI/RFCI for Binary, Ordinal, and Continuous Variables

We run KAPC, Copula PC [24], *deal* [11], PC MXM [70], and Hill-Climbing [60] for learning causal graphs from generated mixed data containing binary, ordinal, and continuous variables (mixed data 1). This experiment implements kernel parameters for RBF kernel σ and Categorical kernel θ, $P = \{P_1(\sigma = 0.001, \theta = 0.5), P_2(\sigma = 0.01, \theta = 1), P_3(\sigma = 0.001, \theta = 1), P_4(\sigma = 0.01, \theta = 1.5), P_5(\sigma = 0.001, \theta = 1.5)$. The experiments using PC, FCI and RFCI implement $\alpha = 0.05$. We use the default option for Hill-Climbing in the bnlearn. The default options that are used in the bnlearn are very similar to the *gaussCItest* function by pcalg implementing Fisher's Z test at $\alpha = 0.05$ [51]. Figure 3.15 shows the SHD score of learned graphs generated from mixed data containing binary, ordinal, and continuous variables (KAPC using different kernel parameters (P1-P5), Copula PC (C), PC MXM (M), Hill-Climbing (H), and *deal* (D)). Copula PC produces lower SHD scores for graphs with 10 nodes. KAPC using kernel parameter P1, P3, and P5 slightly outperforms Copula PC for graphs with 20 and 30 nodes. Copula PC successfully generates the learned graphs from 92% of all datasets. Hill-Climbing method produces the highest SHD score for graphs containing 10 nodes, but it outperforms Copula and KAPC for graphs with 20 and 30 nodes. KAPC always outperforms *deal*. Some learned graphs produced by KAPC have lower SHD scores than learned graphs generated using PC MXM.

We delete some variables to represent latent variables, then use the remaining variables to learn graphs using KAFCI, KARFCI and Copula FCI. Figure 3.16 shows the SHD scores of learned graphs produced by KAFCI using different kernel parameters (P1-P5), KARFCI using different kernel parameters (R1-R5) and Copula FCI (C). KAFCI and KARFCI produce lower SHD scores than Copula FCI. KAFCI and KARFCI generate

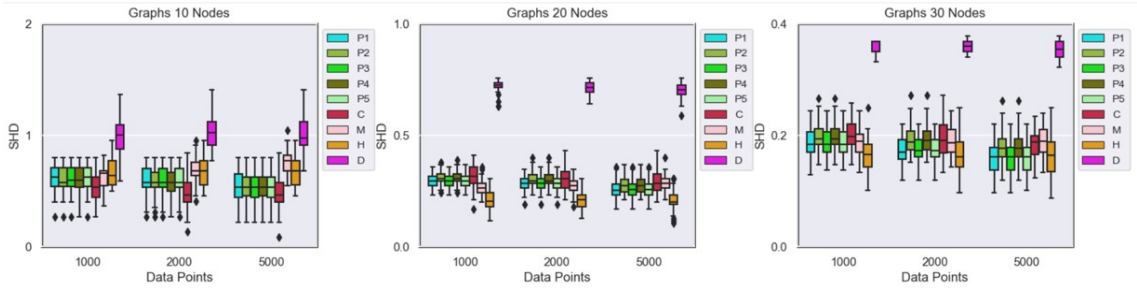


Figure 3.15: SHD score of learned graphs generated from mixed dataset (binary, ordinal, continuous data types)

graphs that have no significant difference for SHD scores. The more data points used to learn causal graphs, the more similar the resulting learned graph to the true graph. Copula FCI successfully generates the learned graphs from 98.67% of total datasets. Copula PC and Copula FCI fail to generate the learned graphs from a few datasets; meanwhile KAPC, KAFCI, and KARFCI successfully generate the learned graphs from all datasets used in the experiments. We do not run MXM for FCI and RFCI due to no ready-to-use application.

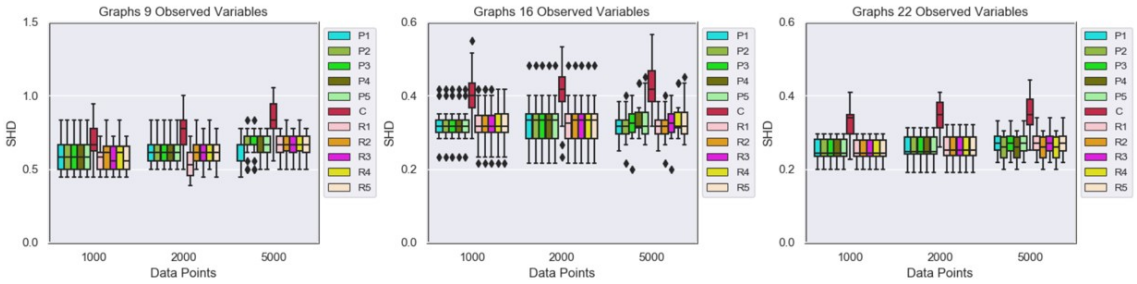


Figure 3.16: SHD score of learned graphs generated from mixed dataset (binary, ordinal, continuous data types) containing latent variables

3.8.4 Kernel Alignment PC Algorithm/FCI/RFCI for Categorical, Binary, Ordinal, and Continuous Variables

The experiments using mixed data containing categorical, binary, ordinal and continuous variables (mixed data 2) apply kernel parameters for RBF kernel $\sigma = \{1, 0.1, 0.01, 0.001, 0.0001\}$ and Categorical kernel $\theta = 2$. The experiments using PC, FCI and RFCI implement $\alpha = 0.05$.

Figure 3.17 shows the SHD score of learned graphs generated from mixed datasets containing categorical, binary, ordinal and continuous variables. P1-P5 refer to KAPC using different kernel parameters, C refers to Copula PC, M refers to PC MXM, D is *deal*,

and H is Hill-Climbing. We run Hill-Climbing using default options. Generally, KAPC outperforms Copula PC and *deal*. Copula PC works well for some datasets. There is an error in the implementation of Copula PC where non-positive definite matrices are being sampled as correlation matrices. This error happened randomly for some datasets. Copula PC successfully generates learned graphs only from 32.67% of all datasets used in this experiment. Hill-Climbing outperforms KAPC in the graphs with 10 and 30 nodes. The SHD scores of PC MXM for the 10 nodes dataset are slightly lower than the SHD score for KAPC. However, PC MXM failed to generate learned graphs from datasets with 20 and 30 nodes due to a software error. *deal* runs very slowly for datasets with 30 nodes, so we decided to terminate the experiment. The experimental results from graphs with 10 and 20 nodes show KAPC outperforms *deal*.

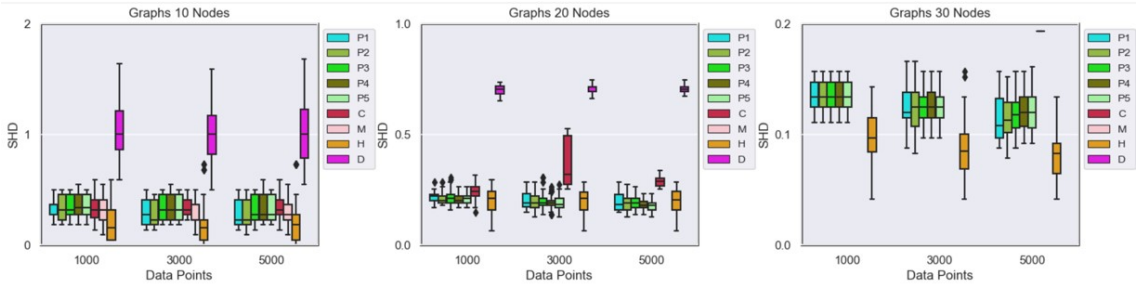


Figure 3.17: SHD score of learned graphs generated from mixed dataset (categorical, binary, ordinal, continuous data types)

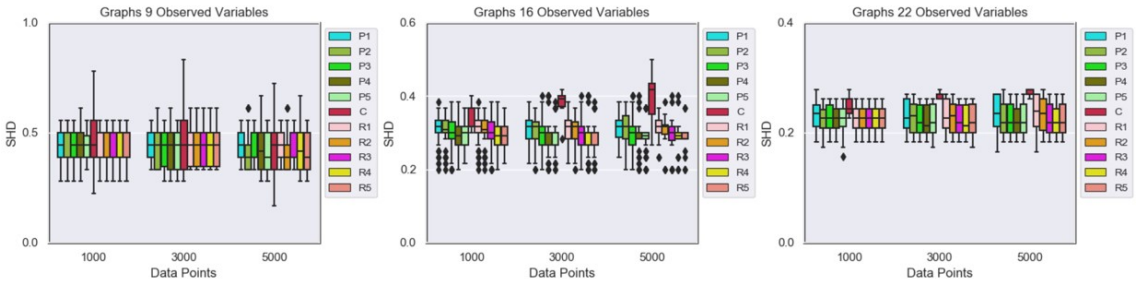


Figure 3.18: SHD score of learned graphs generated from mixed dataset (categorical, binary, ordinal, continuous data types) containing latent variables

We delete some variables to represent latent variables and use the remaining variables to learn graphs using KAFCI, KARFCI and Copula FCI. Figure 3.18 shows SHD scores of KAFCI using different kernel parameter values (P1-P5), KARFCI using different kernel parameter values (R1-R5), and Copula FCI (C) for mixed data containing categorical, binary, ordinal and continuous variables. The kernel matrices and kernel alignment matrix are computed from the remaining variables. The SHD scores of KAFCI and KARFCI from

graphs with 9 observed variables are not significantly different but P2(R2) generates the lower SHD score for sample size $N = 5000$. The SHD scores of graphs with 16 observed variables show that the values of kernel parameter P4/R4 and P5/R5 produce lower SHD scores than others. In graphs with 22 observed variables, P1(R1) produces a higher SHD score among P1-P5(R1-R5). The more sample data used to learn the graphs, decreases the SHD score. In general, this experimental result shows that KAFCI and KARFCI outperforms Copula FCI. Copula FCI only successfully generates the learned graphs from 35.33% of total datasets. KAPC, KAFCI, and KARFCI successfully generates the learned graphs from all datasets used in the experiments. There is no significant difference between the KAFCI SHD score and the KARFCI SHD score. The learned graphs generated using KAFCI from this experiment are used as input for MVEE. MVEE will be discussed in Chapter 5.

3.8.5 Running Time

The kernel-based approach computes the kernel matrix from each variable in the dataset and computes the kernel alignment matrix from those kernel matrices, then uses the kernel alignment matrix as an input for a conditional independence test. The Copula method refers to Copula PC computing the scale matrix and the degree of freedom from the dataset, then using the scale matrix to substitute the correlation matrix and degree of freedom as the number of data points for the input of the conditional independence test [24]. The running time refers to the time spent by the Copula method and our kernel-based method to process the mixed data into the outputs ready to be used as inputs for the conditional independence test. Running time for Hill-Climbing and PC MXM is how long the algorithms generate a graph from a dataset. It estimates how long the process to generate graphs from the datasets in the simulation. Note that this is limited to the datasets in the simulation so it does not lead to generalized cases. Figure 3.19 A and B shows the comparison running time of the kernel-based approach, the Copula approach, PC MXM, Hill-Climbing, and *deal* for mixed data 1 and mixed data 2, respectively. In a condition when the dataset has no missing values, the kernel-based approach is faster than the Copula approach. The algorithm runs more slowly if the dataset contains more variables and data points. Hill-Climbing and PC MXM run faster than the kernel-based approach. *deal* runs faster than the kernel based approach except for 30 nodes datasets containing categorical, binary, ordinal, and continuous variables. *deal* runs very slowly on

large numbers of variables.

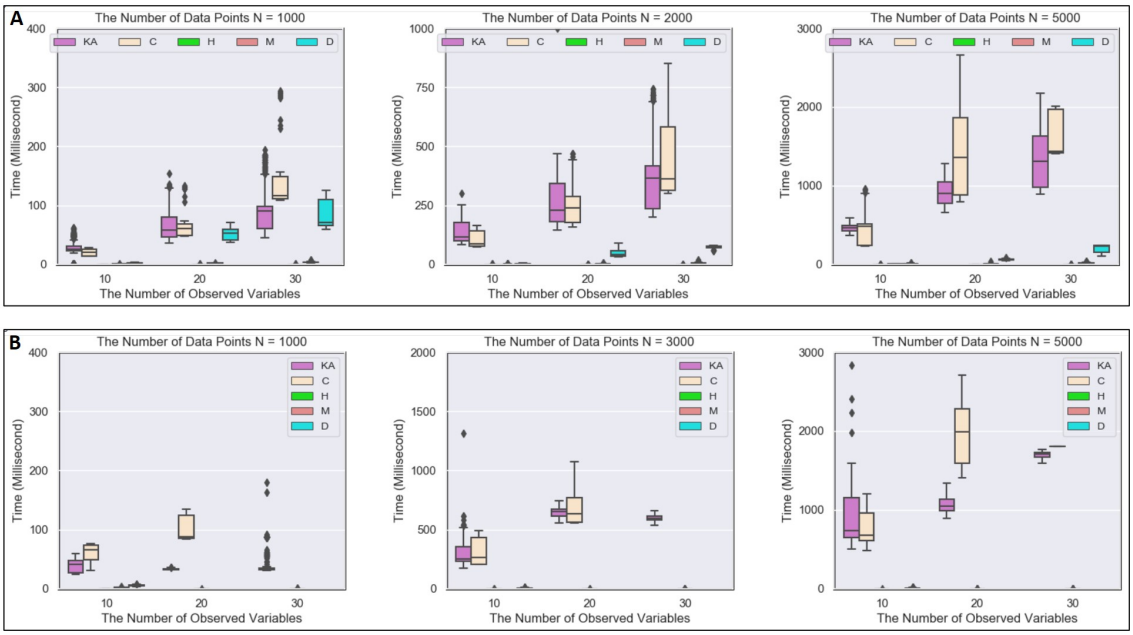


Figure 3.19: Running time for datasets having no missing value

3.9 Discussion

The reason to develop the kernel-based approach is that it provides a procedure to treat categorical variables similarly to the binary, ordinal and continuous variable in learning a causal graph from mixed data using PC, FCI and RFCI. Copula PC algorithm computes the scale matrix and degree of freedom then uses them as input for a conditional independence test [24]. Copula PC algorithm uses around 80% of the actual number of data points. The kernel-based approach uses n data points for conditional independence test, where n is the number of data points in the dataset. Having more data typically leads to more accurate learned graphs.

The main goal of learning causal graphs from the data is producing the learned graphs similar to the true graphs. The advantage of KAPC is it works for categorical variables as well as binary, ordinal, and continuous variables, so it is a step further than Copula PC. When the dataset has no missing values, the kernel-based approach is running faster than the Copula model. KAPC, KAFCI, and KARFCI successfully generate learned graphs from mixed data when the other methods (Copula PC/FCI, PC MXM and *deal*) could not do for some datasets. The kernel-based approach gives a simple solution for learning causal graphs from mixed data using PC, FCI, and RFCI. The experiments using different

values of parameters show that KAPC, KAFCI, KARFCI produce graphs with a lower SHD score when given proper kernel parameter values. This proposed method might generate the wrong graph when we use the wrong kernel parameter. All experiments in Chapter 3 use the datasets without missing values. This proposed method is possible to implement for mixed data containing missing values (see Chapter 4). In general, Hill-Climbing generates learned graphs with an average lower SHD score than KAPC. Some learned graphs produced by KAPC have lower SHD scores than learned graphs produced by PC MXM. Based on the experimental results, KAPC always outperforms *deal*. Greedy search Hill-Climbing and *deal* are score-based methods to learn graphs from mixed data assuming there is no latent variable in the datasets.

FCI and RFCI allow the presence of latent variables. The advantage of the kernel-based approach is it can be implemented for FCI and RFCI, so KAFCI/KARFCI can be used to learn graphs from a dataset that has latent variables. Copula PC and PC MXM do not work for some datasets due to a bug in the software. We do not carry out a further investigation because it is not our main concern. Meanwhile, KAPC, KAFCI and Hill-Climbing succeed in generating the learned graphs for all datasets used in the simulation.

In the situation when learning causal graphs from the datasets but the ground truth (true graph) is unknown, it is difficult to choose the best kernel function. The learned graphs structure cannot be compared to the true graph. We propose a Modal Value of Edges Existence (MVEE) method to measure the quality of the learned graph's structure when the true graph is unknown. MVEE can be used to choose the proper values of the kernel parameters or to choose the best kernel function. MVEE will be explained in Chapter 5.

3.10 Summary

The kernel-based approach is a method proposing to handle mixed data so that it can be used to learn causal graphs using PC, FCI and RFCI. Two main steps of the kernel-based approach are computing the kernel matrix from each variable and computing the kernel alignment matrix. The kernel alignment between two variables is computed from kernel matrices from those variables. The kernel alignment matrix is used to substitute the normal correlation matrix for conditional independence test in PC, FCI, and RFCI.

The datasets for simulation of learning graphs are generated using forward sampling

from the hybrid network. It implements the Conditional Linear Gaussian (CLC). First, it generates random DAGs that contain 10, 20, and 30 nodes, then each node is given a data type. The second step is to generate the mixed data from those graphs. The data types are categorical, binary, ordinal and continuous. This experiment applies the RBF kernel and the Categorical kernel for continuous and discrete variables, respectively. The experiments using benchmark datasets are run to ensure that the proposed method is reasonable to learn causal graphs. The quality of the learned graph's structure is measured using the SHD score. The SHD score is obtained by comparing the learned graph to the true graph. The lower SHD score means that the learned graph has a high similarity to the true graph. The outputs of our kernel-based approach are compared to the output of Copula method [24], PC MXM [70], *deal* [11], and greedy search Hill-Climbing [60]. Based on the experimental results using generated datasets, KAPC and KAFCI/KARFCI outperform Copula PC and Copula FCI. KAPC also outperforms *deal*. KAPC shows a slightly better performance than PC MXM. However, greedy search Hill-Climbing shows a better performance than KAPC. The advantage of the kernel-based approach compared to Hill-Climbing and *deal* is it can be used for FCI and RFCI that allow the presence of latent variables. There is no significant difference in SHD scores from learned graphs produced by KAFCI and KARFCI. The kernel-based approach is successful in generating learned graphs from all generated datasets. The kernel-based approach is a promising method for learning a causal graph from mixed data containing categorical, binary, ordinal, and continuous variables using the PC, FCI, and RFCI. The kernel-based approach offers better treatment for the categorical variable in mixed data which cannot be handled properly using the Copula model [24].

Chapter 4

Learning Causal Graphs from Mixed Data Containing Missing Values

4.1 Introduction

Missing value data might occur during data collection because of instrumental error or human error. Missing value data is a common problem in data analysis because conventional statistical methods and software were developed for fully measured data. Missing data also has a significant effect on the conclusion produced from the data [41]. Algorithms for data analysis were designed for data matrices with no missing values [31]. Missing data causes some problems: reducing the sample size might affect the performance of algorithms and statistical methods, mislead the experimental results, and disturb significantly the outcome of the research study.

KAPC, KAFCI, and KARFCI are successful for learning causal graphs from mixed data and they are not restricted to the specific datasets and choices of kernel functions. It is an advantage to choose the kernel functions suitable for the dataset, i.e., a dataset containing missing values. If some kernel functions (e.g. RBF kernel and Categorical kernel) are not working when the dataset contains missing values, it is possible to change the kernel functions suitable for the dataset's condition. The goal of this experiment is to learn causal graphs from mixed data containing missing values using a kernel-based approach. We run two different approaches to learn graphs from mixed data containing

missing values: apply the kernel function for missing values data and apply the imputation method. The strategies extend the performance of the kernel-based approach for learning causal graphs not only from mixed data but also when the dataset contains missing values.

4.2 Related Work in Causal Learning with Missing Values Data

Several methods were developed to handle missing data, e.g. listwise deletion, pairwise deletion, dummy-variable adjustment, imputation, and maximum likelihood [2]. *Listwise deletion* is removing cases with missing data, then analyzing the remaining data. It enables analyzing cases with available data on each variable. *Listwise deletion* method is a simple method but it might lead to a loss of more information. Strobl et al. implemented test-wise deletion which refers to the process of listwise deletion, then using those datasets for learning causal graphs using FCI and RFCI [64]. The experimental result shows that FCI and RFCI with test-wise deletion outperform their list-wise deletion and imputation on MNAR datasets. However, the deletion procedure also eliminates good samples that contain only a few missing values. It drives much information loss. *Pairwise deletion* removes the data only if the specific data point required to test a particular assumption vanishes. It enables analyzing all cases in which the variables of interest exist. A *dummy variable* is created for each predictor with missing data to indicate the presence of missingness. A constant value (e.g. mean) is given to the cases with missing data on a predictor. This method uses all available data but the result is biased. The *imputation method* is guessing and estimating a particular value to substitute for the missing value, for example, substitute the missing value with the mean or median of the remaining data for continuous/ordinal variables and mode for categorical variables. This is not smart imputation, especially for categorical variables because it produces the 'new data' dominated by majority values. Some research has been done to develop the imputation method for missing data handling [80] [63] [57]. Two examples of imputation methods are regression imputation and multiple imputations. Multiple imputation replaces each missing value with two or more admissible values that represent a distribution of possibilities [44]. It leads to uncertainty because there are many different ways to do multiple imputations [2]. Westreich et al. applied multiple imputations and the parametric g-formula [76]. The imputation method works by creating a copy of the full dataset where the missing values are

predicted using an imputation model. *Maximum likelihood* estimates the value which is most possible to have appeared in the observed data. This method works well if there are large amounts of data but it works poorly in a small sample. Copula PC algorithm used Gibbs sampling to draw correlation matrix samples from mixed data under missingness at random (MAR) [25] [26]. These samples are translated into an average correlation matrix and an effective sample size. Copula PC estimates a more accurate correlation matrix and causal structure under MCAR and MAR. PC MXM applies a simple imputation method to replace the missing data using mode/median value [70].

4.3 Kernel Extension for Missing Values Data

Kernel extension was developed to compute a kernel matrix from missing value data [52]. Kernel extension works for discrete and continuous variables. The advantages of applying a kernel extension is to reduce information loss because there is no removing a sample when it has missing values. The other benefit is there is no need to apply the preprocessing method for missing values data (e.g. imputation or deletion).

Let $X = \{x_1, x_2, ..., x_n\}$ be a continuous variable containing missing values and let $H \in \mathbb{R}$ be any bounded subset, then denote $b = \sup_{x,y \in H} |x-y|$ and $a = \inf_{x,y \in H} |x-y|$. The kernel extension using uniform Kernel Density Estimation (KDE) for continuous variables can be computed using equation 4.1, where $\mathcal{M}$ is missing value, g_1 and G_1 are defined by equations 4.2 and 4.3, respectively.

$$\hat{K}_1(x_i, x_j) = \begin{cases} 1 - \frac{|x_i - x_j|}{b-a}, & \text{if } x_i, x_j \neq \mathcal{M} \\ g_1(x_i), & \text{if } x_i \neq \mathcal{M} \text{ and } x_j = \mathcal{M} \\ g_1(x_j), & \text{if } x_i = \mathcal{M} \text{ and } x_j \neq \mathcal{M} \\ G_1, & \text{if } x_i = x_j = \mathcal{M} \text{ and } i \neq j \\ 1, & \text{if } x_i = x_j = \mathcal{M} \text{ and } i = j \end{cases} \tag{4.1}$$

$$
\begin{aligned}
g_1(z) =& \int_{-\infty}^{\infty} \hat{f}(x)(1 - \tfrac{|x-z|}{b-a})dx \\
=& \int_{-\infty}^{\infty} \tfrac{1}{nh} \textstyle\sum_{i=1}^{n} \varphi(\tfrac{x-x_i}{h})(1 - \tfrac{x-z}{b-a})dx \\
=& \tfrac{1}{nh} \textstyle\sum_{i=1}^{n} \int_{-\infty}^{\infty} \varphi(\tfrac{x-x_i}{h})(1 - \tfrac{x-z}{b-a})dx \\
=& \tfrac{1}{2nh} \textstyle\sum_{i=1}^{n} \tfrac{1}{2} \int_{x_i-h}^{x_i+h} (1 - \tfrac{|x-z|}{b-a})dx \\
=& \tfrac{1}{2nh} \textstyle\sum_{i=1}^{n} \alpha_i(z)
\end{aligned}
\tag{4.2}
$$

Where $\alpha_i(z)$ can be defined as

$$
\alpha_i(z) = \begin{cases}
\frac{2h(b-z+x_i-a)}{b-a}, & if\ z > x_i + h \\
\frac{2h(b-a)-(x_i-z)^2-h^2}{b-a}, & if\ x_i - h \leq z \leq x_i + h \\
\frac{2h(b-x_i+z-a)}{b-a}, & if\ z < x_i - h
\end{cases}
$$

$$
\begin{aligned}
G_1 =& \int_{-\infty}^{\infty} \hat{f}(z) g_1(z) dz \\
=& \tfrac{1}{2nh} \textstyle\sum_{i=1}^{n} \int_{-\infty}^{\infty} \tfrac{1}{nh} \sum_{j=1}^{n} \varphi(\tfrac{z-x_j}{h}) \alpha_i(z) dz \\
=& (\tfrac{1}{2nh})^2 \textstyle\sum_{i=1}^{n} \sum_{j=1}^{n} \int_{x_j-h}^{x_j+h} \alpha_i(z) dz \\
=& (\tfrac{1}{2nh})^2 \textstyle\sum_{i=1}^{n} \sum_{j=1}^{n} \beta_{ij}
\end{aligned}
\tag{4.3}
$$

Where β_{ij} can be defined as

$$
\beta_{ij} = \begin{cases}
\frac{4h^2(b-x_j+x_i-a)}{b-a}, & if\ x_i + h < x_j - h \\
\frac{12(b-a)h^2-(x_i-x_j)^3-2h(4h^2+3(x_i-x_j)^2)}{3(b-a)}, & if\ x_j - h \leq x_i + h < x_j + h \\
\frac{4h^2(3(b-a)-2h)}{3(b-a)}, & if\ x_j = x_i \\
\frac{12(b-a)h^2+(x_i-x_j)^3-2h(4h^2+3(x_i-x_j)^2)}{3(b-a)}, & if\ x_j - h < x_i + h \leq x_j + h \\
\frac{4h^2(b-x_i+x_j-a)}{b-a}, & if\ x_j + h < x_i - h
\end{cases}
$$

Suppose a categorical variable has a set of finite values $\mathcal{V} = \{v_1, ..., v_l\}$. The kernel extension for discrete variables can be computed using equation 4.4, where $\mathcal{M}$ is missing value.

$$\hat{K}_2(v_i, v_j) = \begin{cases} \mathbb{I}_{\{v_i = v_j\}}, & \text{if } v_i, v_j \neq \mathcal{M}; \\ g_2(v_i), & \text{if } v_i \neq \mathcal{M} \text{ and } v_j = \mathcal{M} \\ g_2(v_j), & \text{if } v_i = \mathcal{M} \text{ and } v_j \neq \mathcal{M} \\ G_2, & \text{if } v_i = v_j = \mathcal{M} \text{ and } i \neq j \\ 1, & \text{if } v_i = v_j = \mathcal{M} \text{ and } i = j \end{cases} \tag{4.4}$$

The $g_2(z) = \sum_{i=1}^{l} f(v_i)\mathbb{I}_{\{v_i = v_j\}} = f(z)$ and $G_2 = \sum_{i=1}^{l} f(v_i)^2$, is a kernel in $\mathcal{V} \cup \{\mathcal{M}\}$.

After computing the kernel matrices, it computes the kernel alignment matrix from these kernel matrices. The kernel alignment matrix A is used as an input for conditional independence test in PC, FCI and RFCI.

4.4 Experimental Design for Learning Causal Graphs from Incomplete Mixed Data

We run two strategies for learning causal graphs from mixed data containing missing values using the kernel-based approach. The first way is to implement the kernel functions that can be used to compute a kernel matrix from missing values data. The second approach is to apply the preprocessing method to handle missing values data. The preprocessing methods might apply imputation or deletion. We prefer to apply the imputation method to keep the dataset and reduce the information loss.

There are only a few kernel functions that can be used to compute kernel matrix from a dataset containing missing values. This experiment implements kernel extension proposed by Troyano and Munoz [52] to compute kernel matrices from variables containing missing values. We implement kernel extension using uniform KDE for continuous variables and kernel extension for categoric features for discrete variables. The reason for implementing the kernel extension is it is available for discrete and continuous variables. In the situation when a dataset containing some variables has missing values and other variables have no missing values, the kernel extension is used to compute a kernel matrix for a variable containing missing values. We implement kernel extension for continuous variables using the kernel parameter value $h = 0.0001$. For the other variables which have no missing values, we use the RBF kernel and the Categorical kernel for continuous and discrete variables, respectively.

The function *imputeFAMD* from R package *missMDA* is an imputation method developed based on a principal components method for the factorial analysis for mixed data (FAMD) [5] [37]. It applies the iterative FAMD algorithm or the regularised iterative FAMD algorithm to impute the missing entries of mixed data. We run function *imputeFAMD* from R package *missMDA* to create the complete datasets from missing values data, then apply the RBF and the Categorical kernel to compute the kernel matrix from the continuous and discrete variables, respectively. The kernel parameters values are set for RBF kernel $\sigma = \{1, 0.1, 0.01, 0.001, 0.0001\}$ and Categorical kernel $\theta = 2$.

The centered kernel matrices are computed using equation 3.3 (see Chapter 3.3.1). We apply kernel alignment to compute the kernel alignment matrix from the centered kernel matrices. Hereafter, the kernel alignment matrix is applied to substitute the correlation matrix for the conditional independence test in PC, FCI, and RFCI. The experiments use three different data point amounts $N = \{1000, 3000, 5000\}$. The experiments for the kernel-based approach, the Copula method, and PC MXM are run at $\alpha = 0.05$. We run Hill-Climbing using the default option.

We use the mixed data 2 (the dataset containing categorical, binary, ordinal, and continuous variables) previously employed for the experiment in Chapter 3.8.4. The missing values datasets (incomplete datasets) for simulation are generated from mixed data 2 containing categorical, binary, ordinal and continuous data types. We employ three kinds of missing values datasets according to the missingness: MAR, MCAR, and MNAR. The missing values datasets are generated using 'function *ampute*' from the R package *mice* [14] [15]. It generates three groups of datasets based on the missingness methods: MAR, MCAR, and MNAR. The incomplete datasets contain 10-15% missing values. Some variables are removed from the mixed data to represent latent variables and the remaining variables are used to learn graphs using KAFCI, KARFCI and Copula FCI.

The proportion of 15% missing data is common in educational and psychological studies [28]. The missing rate of 5% or less is possibly inconsequential [27]. The results of statistical analysis might be biased when the dataset containing more than 10% missing values [9] [27]. The results should only be considered as hypothesis-generating when the dataset containing more than 40% missingness [45]. KAPC/KAFCI/KARFCI possibly generate more accurate learned graphs from datasets containing a small amount of missingness ($< 10\%$) than when the proportion of missing data is increased ($> 15\%$).

4.5 Experimental Results and Discussion

The kernel-based approach, the Copula method [24], PC MXM [70], greedy search Hill-Climbing [60], and *deal* [11] are run to learn graphs from mixed data containing missing values. The Copula method [26] and PC MXM [70] can be used to learn causal graphs from mixed data containing missing values [26]. Hill-Climbing (HC) and *deal* (D) are not working for datasets containing missing values. In the experiment using Hill-Climbing and *deal*, we implement the imputation method, function *imputeFAMD*, from R package *missMDA* [37].

Figures 4.1, 4.2 and 4.3 show the SHD scores of graphs generated from a mixed dataset containing missing values. The prefix *i* in iKE/iC/iHC/iM/iD/iP refers to 'incomplete' and it is used to identify the output from algorithms for mixed data containing missing values (incomplete datasets). iHC, iC, iM, and iD refer to Hill-Climbing, Copula method, PC MXM, and *deal*, respectively. Greedy search Hill-Climbing using the imputation method outperforms KAPC in the graphs of 10 and 30 nodes. Note that Hill-Climbing is a score-based learning method. Meanwhile, KAPC, PC MXM, and Copula PC are constraint-based learning methods. KAPC shows better performance than *deal*, Copula PC and PC MXM. We run *deal* to learn graphs only from datasets with 10 and 20 nodes because *deal* runs very slowly for datasets with 30 nodes.

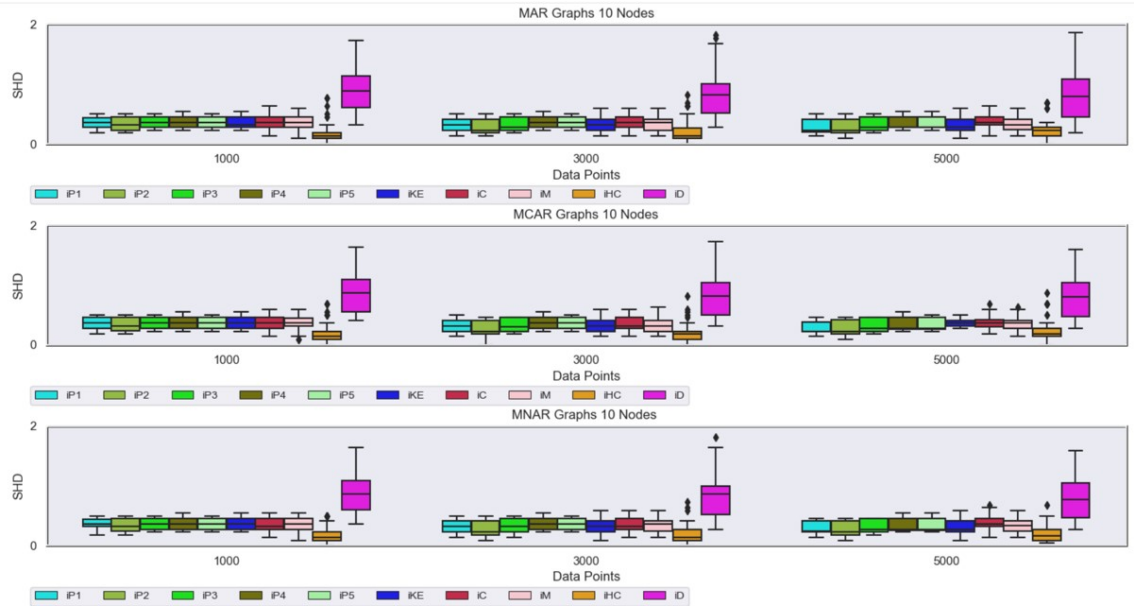


Figure 4.1: The SHD scores of graphs learned from mixed data containing missing values for graphs with 10 nodes

Figures 4.4, 4.5 and 4.6 show the SHD score of graphs generated from a mixed dataset

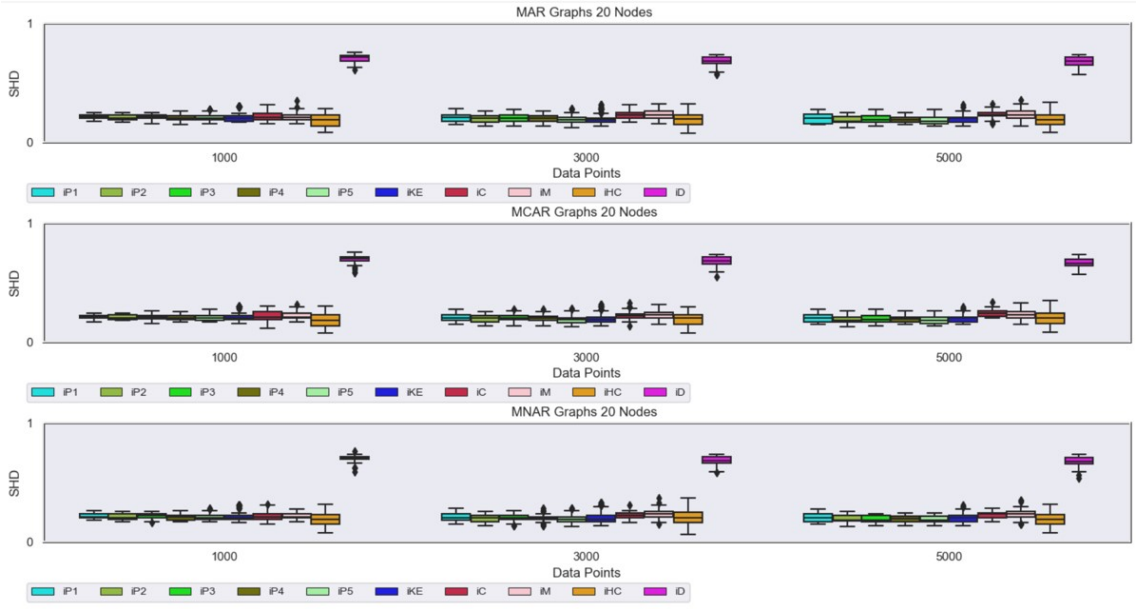


Figure 4.2: The SHD scores of graphs learned from mixed data containing missing values for graphs with 20 nodes

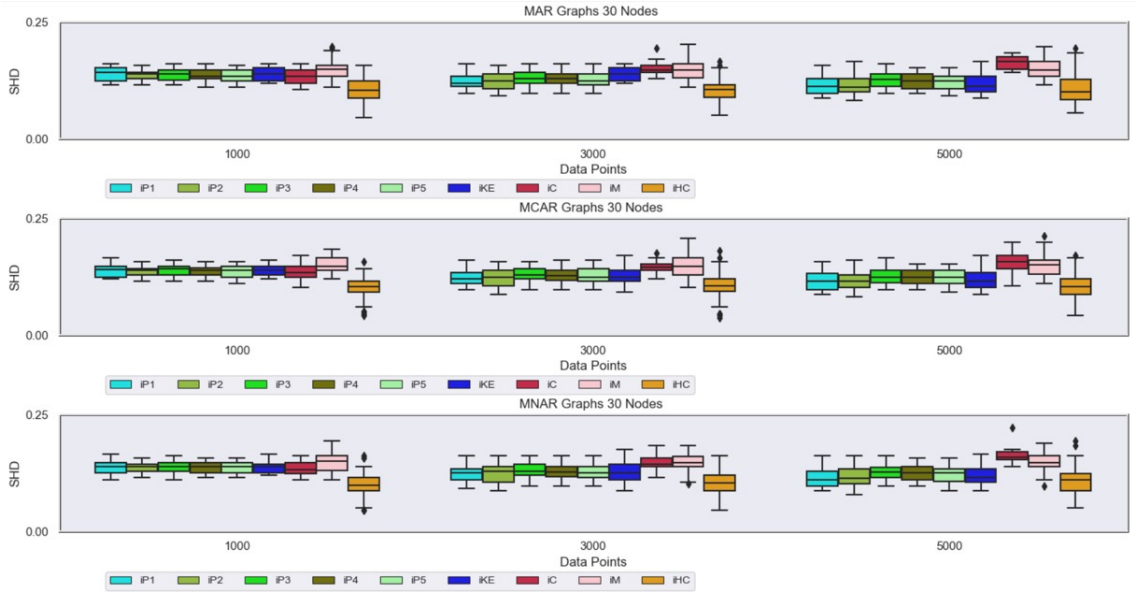


Figure 4.3: The SHD scores of graphs learned from mixed data containing missing values for graphs with 30 nodes

containing missing values and latent variables. iKEF and iKER refer to KAFCI and KARFCI using kernel extension. The kernel-based approach using the imputation method applies the RBF and the Categorical kernel. KAFCI(iP1-iP5) and KARFCI(iR1-iR5) implement different kernel parameter values for the RBF kernel and Categorical kernel. iC refers to Copula FCI. There is no significant difference between the SHD scores of KAFCI and KARFCI. The experimental results show that KAFCI and KARFCI outperform Cop-

ula FCI.

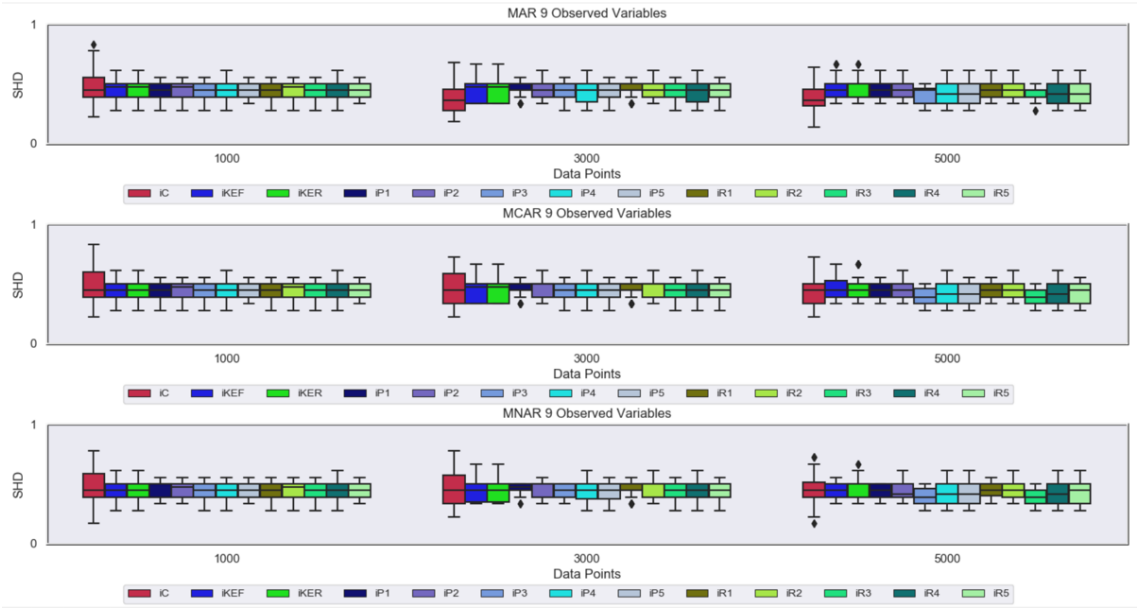


Figure 4.4: The SHD scores of graphs learned from mixed data containing missing values and latent variables for graphs with 9 observed variables

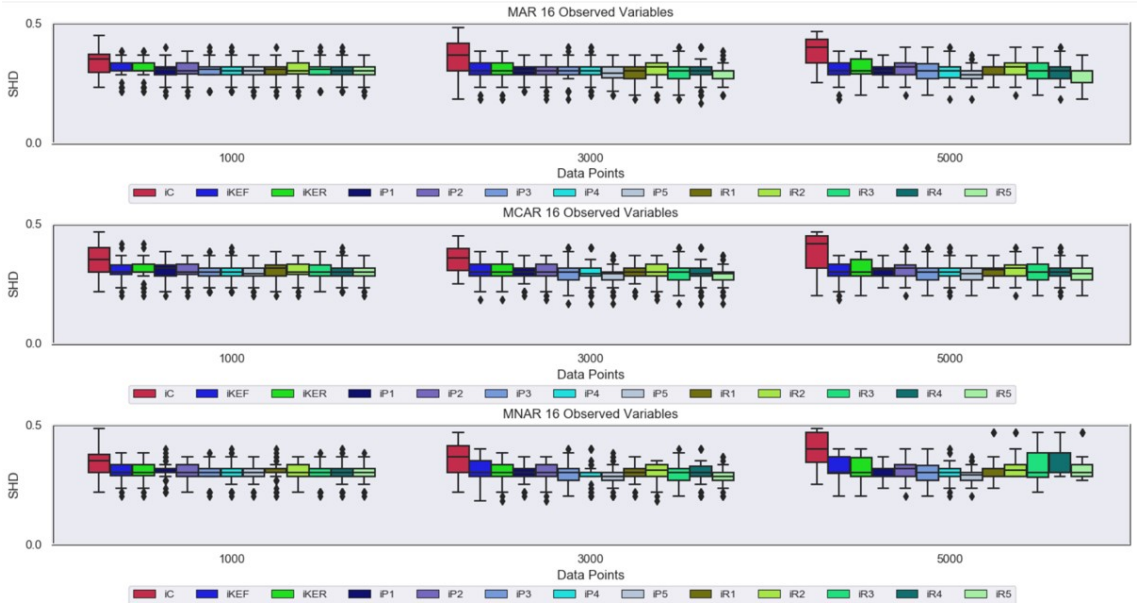


Figure 4.5: The SHD scores of graphs learned from mixed data containing missing values and latent variables for graphs with 16 observed variables

The performance of KAPC and KAFCI/KARFCI using the kernel extension and imputation shows no significant difference. KAPC, KAFCI, and KARFCI produce learned graphs which have no significantly different SHD scores for MAR, MCAR, and MNAR datasets. The experimental results show that the performance of the kernel-based ap-

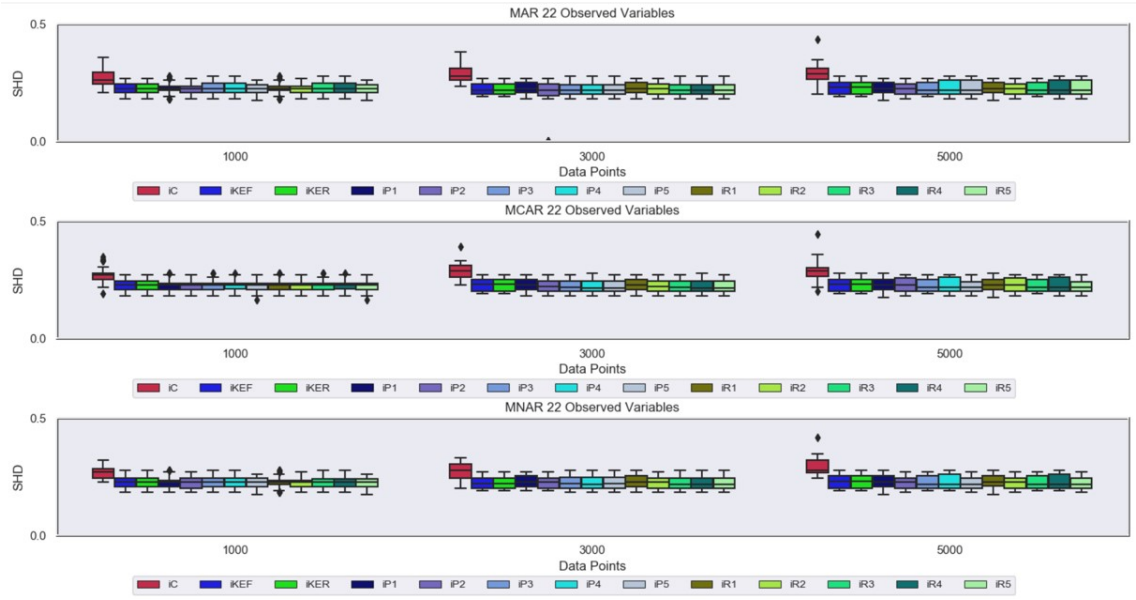


Figure 4.6: The SHD scores of graphs learned from mixed data containing missing values and latent variables for graphs with 22 observed variables

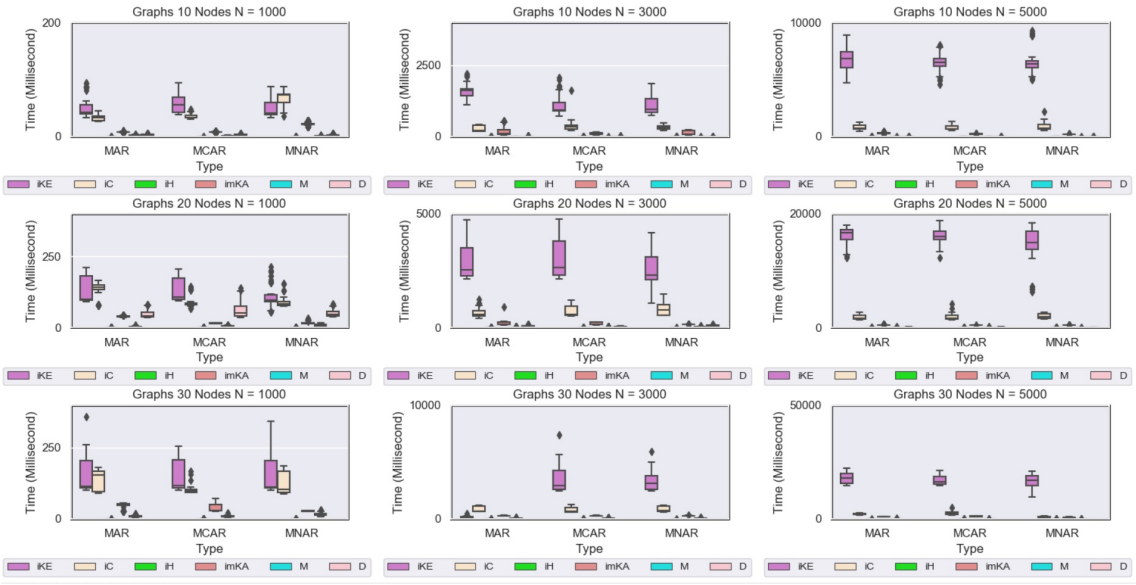


Figure 4.7: Running time for missing values data

proach for missing values data is not restricted to the specific missing data. The more data points used in the experiment, the more similar the outputs of the learned graph from the kernel-based approach to their true graph. KAPC, KAFCI, and KARFCI produce a high quality learned graph when they are given the suitable kernel parameter values.

PC, greedy search Hill-Climbing and *deal* are methods to learn graphs assuming the dataset has no latent variables. In the condition when the mixed datasets contain missing values and have no latent variables, Hill-Climbing outperforms KAPC. The advantage of

the kernel-based approach is it can be implemented for FCI and RFCI which allows the presence of latent variables. The experimental result shows that when the mixed datasets contain missing values and latent variables, KAFCI and KARFCI outperform Copula FCI when KAFCI and KARFCI are given suitable kernel parameter values. KAPC, KAFCI, and KARFCI work for the incomplete data with a small number of missing values as well as with complete data. Note that these experiments use the datasets containing a small number of missing values (10-15% missing values). If the datasets have more missing values (more than 15%) the results might be different.

KAPC, KAFCI, and KARFCI with the kernel extension use the remaining data points from the dataset without applying the deletion or imputation method to fill up the missing values. KAPC, KAFCI, and KARFCI using kernel extension offer a solution to learn the graphs from missing values data when we want to maintain the original data and avoid any modifications (deletion or imputation). KAPC, KAFCI, and KARFCI using the imputation methods are another way to handle the datasets containing missing values. The imputation methods play an important role in generating high-quality learned graphs. The sophisticated imputation method supports the performance of KAPC, KAFCI and KARFCI. However, a very simple imputation method (e.g. replace the missing values using mean or mode values) might cause KAPC, KAFCI, and KARFCI to generate a poor learned-graph.

The weakness of KAPC, KAFCI, and KARFCI using kernel extension is they run more slowly than when they implement the RBF kernel and the Categorical kernel. Figure 4.7 shows the running time for missing values data. iKE refers to the kernel-based approach using kernel extension and imKA refers to the kernel-based approach using imputation. The kernel extension has more complex computation than the RBF kernel and the Categorical kernel. The Copula method draws samples from the datasets using the same procedure no matter whether the dataset has missing values or not. As a consequence, the Copula method's running time remains the same for a complete and incomplete dataset. Hill-Climbing (iH), PC MXM (M), and *deal* (D) run much faster than the kernel-based approach using kernel extension and the Copula method (iC).

4.6 Summary

The kernel-based approach applies two different ways to handle missing values data. The first is to apply the kernel extension for missing values and the second is to implement the

imputation method. KAPC, KAFCI, and KARFCI using kernel extension are methods that can be used to learn causal graphs from mixed data containing missing values. The kernel extension is applied to compute the kernel matrix from the variables that have missing values. The kernel extension works for discrete and continuous variables. KAPC, KAFCI, and KARFCI implement the function *imputeFAMD* to handle missing values data, then apply the RBF kernel and the Categorical kernel to compute the kernel matrices from the dataset. After computing the kernel matrix from each variable in the dataset, we compute the kernel alignment matrix A. The kernel matrix A is used as an input for conditional independence test in PC, FCI, and RFCI.

We use three groups of missing value data related to the mechanism of missingness: MAR, MCAR, and MNAR. The experimental results show that KAPC, KAFCI, and KARFCI work well to learn causal graphs from mixed data containing missing values for MAR, MCAR, and MNAR datasets. Assuming the mixed datasets containing missing values have no latent variable, KAPC outperforms *deal*, Copula PC, and PC MXM, but the greedy search Hill-Climbing outperforms KAPC. The advantage of a kernel-based approach is it can be implemented for FCI and RFCI which allow the presence of latent variables. KAFCI and KARFCI generate the learned graphs which have a lower SHD score than the learned graphs generated using Copula FCI.

Chapter 5

Evaluating The Graph Structure When The True Graph is Unknown

5.1 Introduction

Structural learning in the context of causal graph inference is learning a graph from a dataset. The learned graph is generated from the dataset using a particular causal algorithm, for instance, the PC algorithm, FCI or RFCI. The learned graph output of the PC is represented by a Completed Partially Directed Acyclic Graph (CPDAG). There are some algorithms that can be used to generate causal graphs that allow the presence of latent variables, for instance, FCI and RFCI. FCI and RFCI produce a causal graph represented by a Partial Ancestral Graph (PAG).

Suppose KAFCI with different kernel parameters can be viewed as different algorithms. The different algorithms might produce different graph structures from the same dataset. KAFCI with different kernel parameters are applied to learn causal graphs from the same dataset and they output the same type of graphs that are represented by an FCI-PAG. Suppose we have competing algorithms that are used to generate learned graphs from the same dataset and they output the same type of graph (e.g. PAG). Figure 5.1 shows 4 different FCI-PAGs generated from the same dataset using 4 different algorithms. It is difficult to choose the best-learned graph when the true graph is unknown. In this situation, we assume that an edge (resp. non-edge) that exists in *most* of the learned graphs

has a high possibility of being a true edge (resp. non-edge). Based on this assumption, we propose a method based on the modal value of the edge type for any node pair to create a proxy to the true graph when the true graph is unknown.

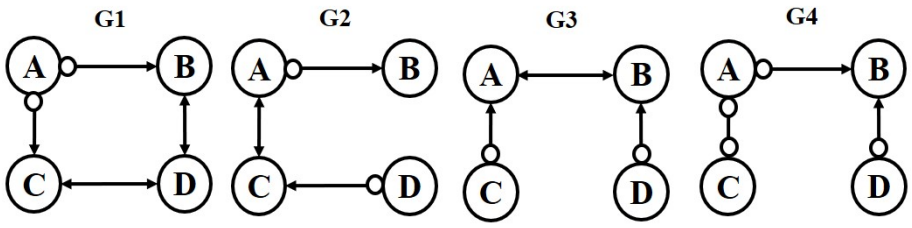


Figure 5.1: Four PAGs generated from a dataset using 4 different algorithms.

The goal of this research was to evaluate the learned graphs when the true graph is unknown. The unknown true graph means that there is a true graph we do not know. We introduced a new method called a Modal Value of Edges Existence (MVEE) to evaluate graph structure represented by a PAG. MVEE adopts the idea of the agreement graph from InterVal [74] and uses the agreement graph as a proxy for the true graph. MVEE can be used to choose the best-learned graph among several learned graphs.

5.2 Related Work

Intersection-validation (*InterVal*) is a method for evaluating CPDAG learning algorithms when the ground truth is unknown [74]. The basic idea is to generate an *agreement graph* from the CPDAGs learned by different algorithms from the same dataset. The agreement graph is then used as a proxy for the ground truth.

Let $G = (V, E)$ be a graph with node set V and edge set $E \subseteq V \times V$. Denote a node pair (u, v) by uv and say that its type in G (or, in E) is bidirected, forward, backward, or non adjacent if, respectively, both uv and vu, only uv, only vu, or neither belongs to E. Agreement graphs are *partial graphs* where a *partial graph* on a set of node pairs $S \subseteq V \times V$ is a pair (S, E) where $E \subseteq S$ [74]. An ordinary graph on V is obtained as a special case with $S = V \times V$.

Suppose, a set of CPDAGs $G = \{G_1, G_2, ..., G_k\}$ are learned from a dataset using algorithms $Alg = \{A_1, A_2, ..., A_k\}$ as input graphs. Figure 5.2 (a) shows the original InterVal method to generate an agreement graph, which is a partial graph, from two CPDAGs (dashed lines connect excluded node pairs not in S) [74]. In this example, the agreement graph has the following node pairs $S = \{(A, B), (A, C), (B, C)\}$ and only one

edge $E = \{(A, C)\}$.

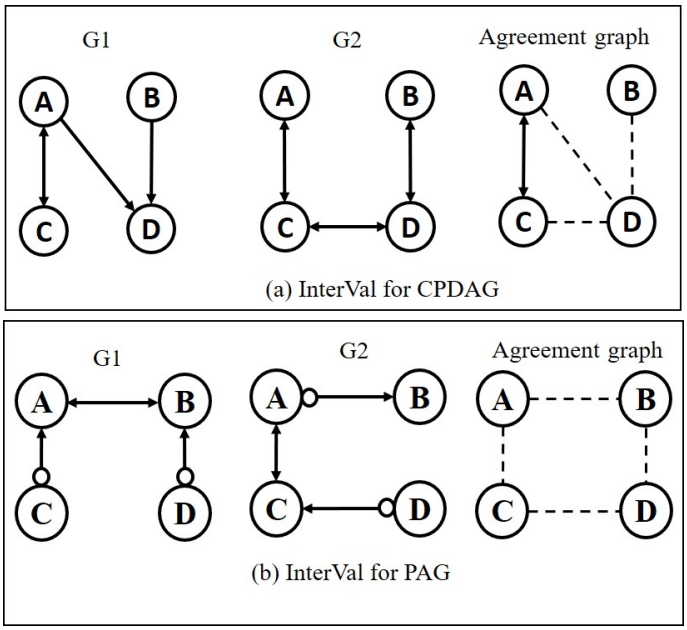


Figure 5.2: Intersection-Validation on CPDAGs and PAGs

InterVal applies a strict rule where it only includes a node pair in the agreement graph when all input graphs agree on that node pair. InterVal might produce an agreement graph containing no node pairs when there is no exact matching node pair type from all input graphs, especially when we create an agreement graph from PAGs. A PAG has three kinds of marks that form six different types of edges ($\circ\!\!\rightarrow$, $\leftrightarrow$, $\circ\!\!-\!\!\circ$, $\rightarrow$, $\circ\!\!-$, and $-$), so to get an exact match for all input graphs is difficult. Figure 5.2 (b) shows that the agreement graph of two PAGs has two node pairs (A, D) and (B, C). InterVal was designed to choose between CPDAGs and MVEE to choose between PAGs which contain more edge types than CPDAGs.

5.3 Modal Value of Edges Existence (MVEE)

Modal Value of Edges Existence (MVEE) is modified from the original InterVal concept proposed by Viinikka et al. [74]. Like InterVal MVEE adopts the idea of an agreement graph that is generated from input graphs and uses the agreement graph as a proxy for the true graph. Specifically, we consider the PAG output from the FCI algorithm. For a given pair of a graph $\mathcal{G}$ and a distribution P faithful to it, there may be two different FCI-PAGs that represent graph $\mathcal{G}$ but they will have the same skeleton [20], so 'true' PAGs have the same skeleton. Since they will all share the same skeleton, it is useful to

use a skeleton agreement graph as a proxy for the skeleton of the true PAG and use it to choose the 'best' graph from the output of competing PAG-learning algorithms. MVEE finds a skeleton agreement graph using a majority vote from the skeletons of a set of input graphs. This differs from InterVal where (i) all graphs must agree on a node pair for that node pair to be included in the agreement graph and where (ii) the agreement graph is not just a skeleton (see Fig 5.2(a)).

5.3.1 MVEE From Two Input Graphs

Generating an agreement graph from two input graphs can be explained as follows. An agreement graph is built by taking the edges that exist in both input graphs. Figure 5.3 shows two input FCI-PAGs (G1 and G2), input graph skeletons (S1 and S2), and an agreement graph skeleton. The skeletons (S1 and S2) are made from G1 and G2 by removing the marks on the edges. An agreement graph is produced by taking the edges that exist in both input skeletons, so in this example the agreement graph has two edges (A—B and A—C). The edges B—D and C—D are not added to the agreement graph because they only exist in one of the input skeletons. The agreement graph has node pairs: A—B, A—C, A—D, and B—C and two edges A—B and A—C.

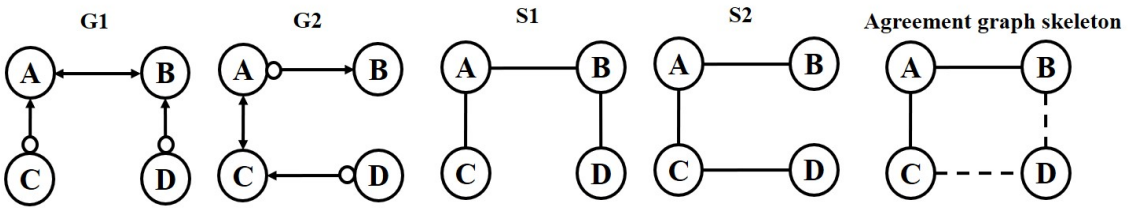


Figure 5.3: Two PAGs and their agreement graph created using MAEE

5.3.2 MVEE From More Than Two Input Graphs

The detailed implementation of MVEE for more than two PAGs can be explained as follows. MVEE creates a skeleton by taking the majority edges that appear in the input graphs. Figure 5.4 shows the input graphs represented by PAGs, the skeleton of input graphs and the skeleton agreement graph. In this example, a tie happens: two graphs have an edge between node C and D and two other graphs do not. In this situation, MVEE decides to abstain and so does not include (C, D) in the node pairs for MVEE's skeleton agreement graph (dashed line connects excluded node pair). If every node pair on input

graphs results in a tie, the MVEE agreement graph has no node pairs. Figure 5.5 shows a case where most node pairs on input graphs are a tie (dashlines connect excluded node pairs). If there are no ties then S, the set of node pairs in the MVEE agreement graph is all possible pairs. Figure 5.6 shows an example of a case of no tie in each node pair. If all input graphs were empty, the MVEE agreement graph is (S, E) where S is $V \times V$ and E is the empty set.

The 'ties' imply that the threshold for including an edge in MVEE is that it appears in 50% of the input graphs. It is possible to use other thresholds (less or more than 50%), e.g. greater threshold might create a sparse agreement graph.

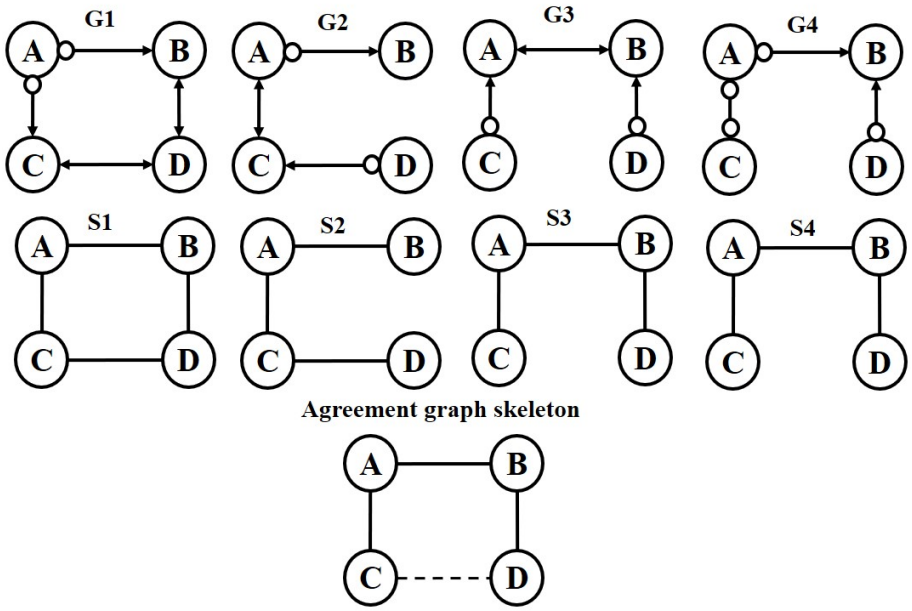


Figure 5.4: Four PAGs and their agreement graph created using MAEE

After generating an agreement graph, it is used as a proxy of the true graph. The next step is computing the mismatch between each input graph and the agreement graph. *Partial Hamming Distance* (PHD) is used to measure the mismatch between a CPDAG and a partial graph. Partial Hamming Distance between two partial graphs $P = (S, E)$ and $P' = (S, E')$ denoted by $PHD(P, P')$ is the number of node pairs in S whose types are different in P and P' [74]. Partial Hamming Distance is a metric in the set of partial graphs on a fixed set of node pairs. In this case, we use the Partial Hamming Distance (PHD) method to compute the mismatch between two skeletons, and the score is called a *Partial Skeleton Error* score. The Partial Skeleton Error (PSE) score is computed by comparing the skeleton of the input graph to the skeleton agreement graph. Figure 5.7 shows an

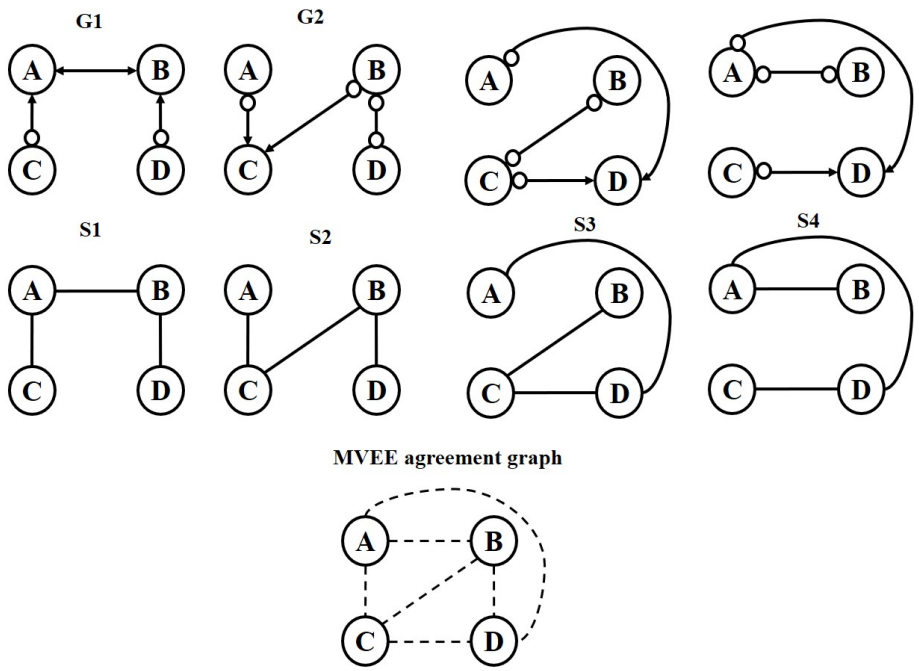


Figure 5.5: MVEE agreement graph when every node pair in the input graphs ties

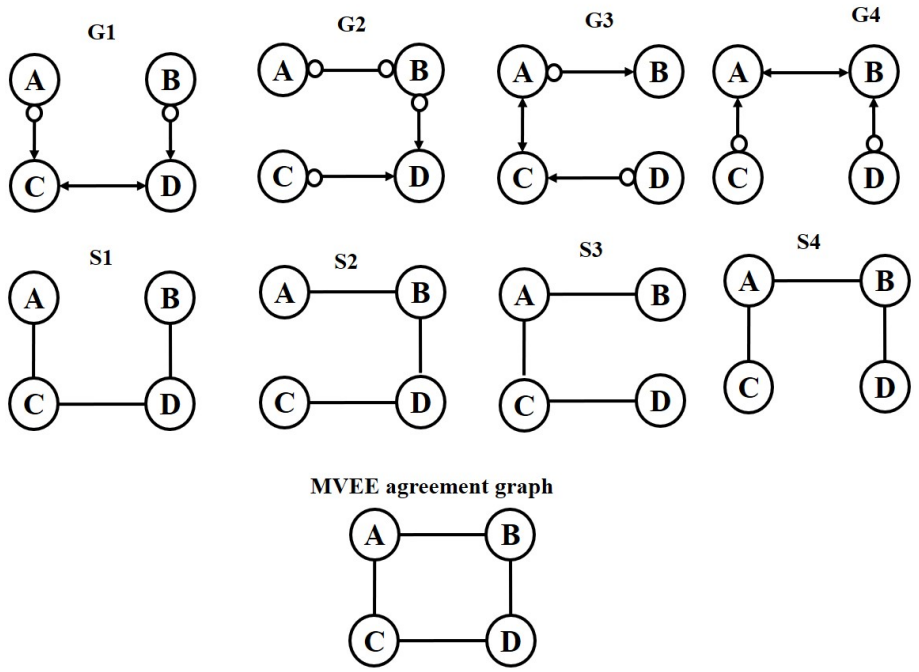


Figure 5.6: MVEE agreement graph with no tie in the input graphs

example of computing the PSE score. The $PSE(Si, S_0)$ computes the mismatch of the skeleton Si to the skeleton agreement graph S_0. For instance, the skeleton of agreement graph S_0 has 5 node pairs. The PSE observes the edges that exist in the agreement graph

skeleton and computes the mismatch node pairs. The $PSE(S1, S_0)$ and $PSE(S3, S_0)$ score is 0 because $S1$ and $S3$ have 5 node pairs matching the S_0 and PSE ignores the edges that do not exist in the agreement graph. The $PSE(S2, S_0)$ score is 1 because $S2$ does not have an edge B—D that exists in the agreement graph. The $PSE(S4, S_0)$ score is 2 and it is obtained from missing edge A—B and extra edge C—B.

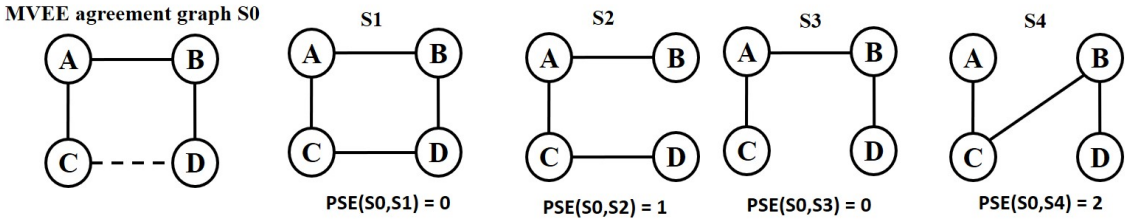


Figure 5.7: The example of computing Partial Skeleton Error (PSE) Score

The agreement graph can be used to measure the quality of the graph structure of the input graphs. The input graph which has the lowest mismatch to the agreement graph can be viewed as the best learned graph. It also shows that the best-learned graph is produced by KAFCI using the appropriate kernel parameters. The skeleton $S4$ (see Figure 5.7) is a worse graph than the three others because it has the highest PSE score $= 2$. Suppose, those input graphs $G1, G2, G3, G4$ are generated by KAFCI using different kernel parameters $P1, P2, P3, P4$, respectively. It implies that the kernel parameter values $P1$ and $P3$ are more precise than $P2$ and $P4$. Implementing the Partial Skeleton Error score to measure the mismatch between the learned graph and the agreement graph then using this score to choose the best-learned graph might provide multiple outputs. Suppose there are two learned graphs that have the lowest PSE score but their graph structures are different. In this situation, both learned graphs are the best-learned graphs. For instance, the graphs $S1$ and $S3$ have different skeletons but they produce the same PSE score.

5.4 Experimental Results and Discussion

We use MVEE to estimate the best learned graph when the true graph is unknown. In this experiment we use mixed data 2 (see section 3.6), the dataset containing categorical, binary, ordinal and continuous variables, and we delete some variables to represent latent variables. First, we generate learned graphs using KAFCI with different kernel parameter values. An agreement graph is generated from the skeletons of the learned graphs from

the same dataset.

The InterVal approach uses subsamples of the data to analyze how close the resulting graph is to the agreement graph. Our goal is different from InterVal, so we do not use exactly the same approach. Our goal is to use MVEE to estimate the best learned graph produced by KAFCI with different kernel parameters when the true graph is unknown. Suppose, $G = \{G_1, G_2, ..., G_n\}$ is a set of learned graphs generated from the same dataset using a different kernel parameter $P = \{P1, P2, ..., Pn\}$. KAFCI using different kernel parameter values can be viewed as different algorithms. These learned graphs are used as input for MVEE to produce an agreement graph S_0. We apply Partial Hamming Distance (PHD) proposed by Viinikka et al. [74]. We call the score a Partial Skeleton Error (PSE) score. The learned graph with the lowest PSE score is considered the best learned graph. Figure 5.8 (A) shows an example of how to choose the ground truth of the best-learned graph. In this case, the ground truth of the best-learned graph is G_4 because it has the lowest SHD score. Figure 5.8 (B) shows how to choose the best learned-graph using the MVEE agreement graph. The best-learned graph is the learned-graph that has the lowest PSE score. The SHD and PHD methods allow two or more learned graphs with different structures to have the same score. As a consequence, the best-learned graph might not be unique. In this example, the best-learned graphs are G_1, G_3, and G_4. MVEE is successful in choosing the best-learned graph if it finds at least one learned-graph that is a member of a ground-truth list of the best-learned graph. Finding the best-learned graph implies that we also discover suitable kernel parameter values. In the example in Figure 5.8, the highest PSE score is produced by graph S2 and it indicates that the kernel parameter values P2 to generate graph G2 are worse than kernel parameters P1, P3, and P4.

To evaluate MVEE, we first find which learned graphs have a minimal (i.e., best) SHD score when compared to the true graph; this is the set of optimal learned graphs. We generate an MVEE agreement graph from the learned graphs generated using KAFCI with different values of kernel parameters from the same dataset. We estimate a best learned graph by computing the PSE score between the learned graphs and the agreement graph; the estimated best learned graph is some learned graph with a minimal PSE score. In our evaluation of MVEE we have found that MVEE identifies an optimal graph 91.56% of the time.

We analyze the agreement graphs to understand their performance as a proxy for the true graph. First, we compute the mismatch between the skeleton of the agreement graph

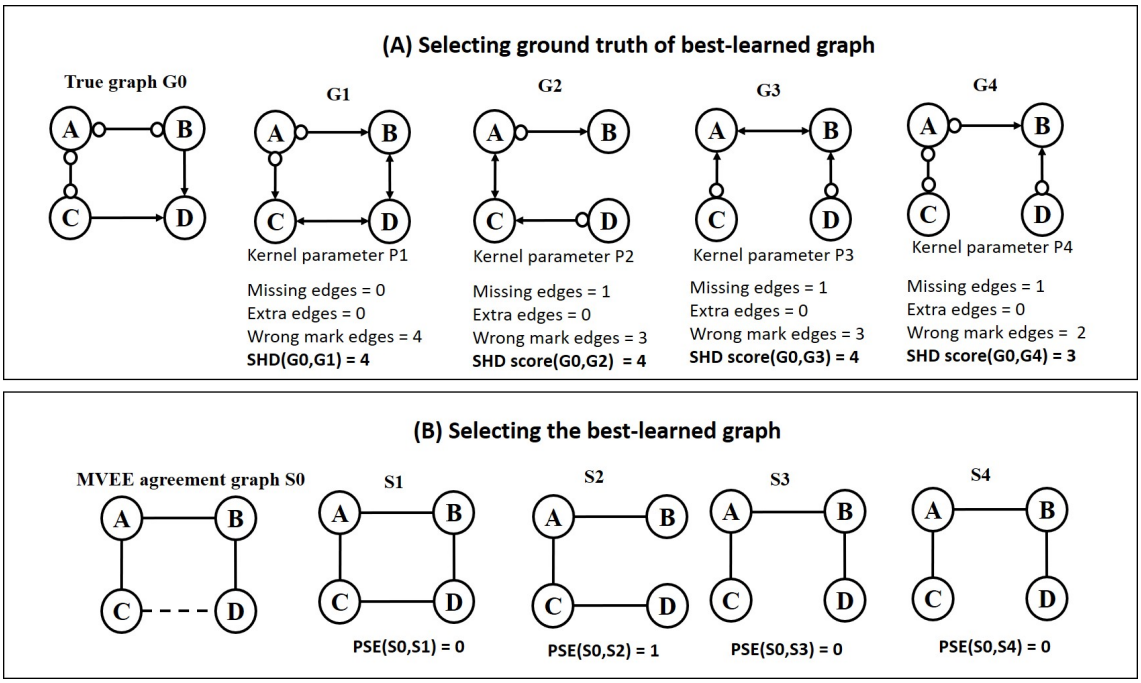


Figure 5.8: Selecting the best-learned graph

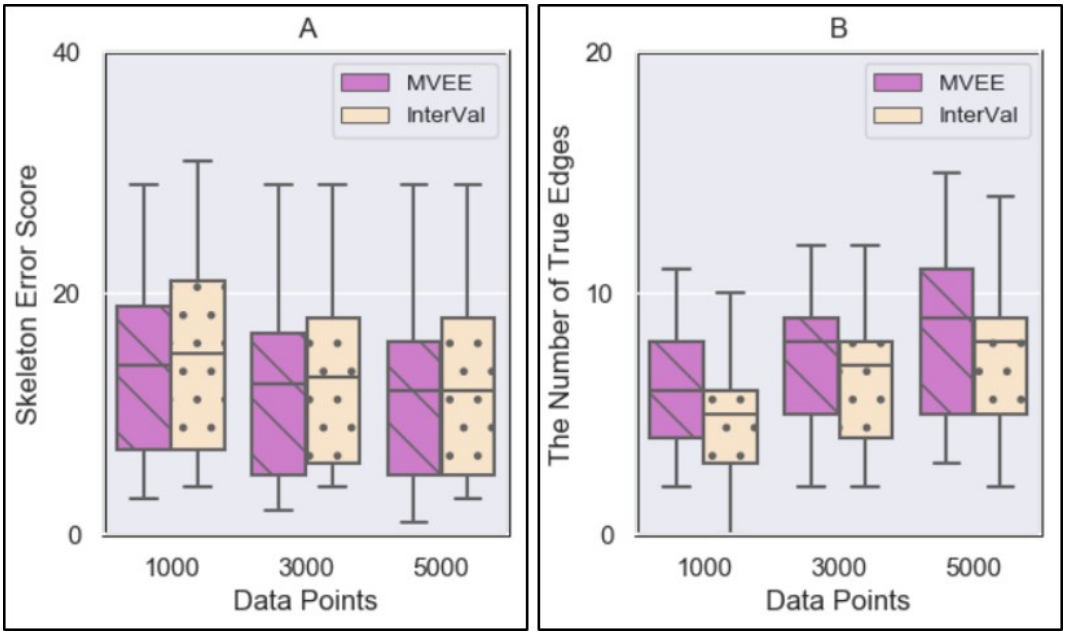


Figure 5.9: MVEE Evaluation

and the skeleton of the true graph. This score is named the skeleton error score. Figure 5.9 (**A**) shows the MVEE skeleton agreement graph has slightly less mismatch than the InterVal skeleton agreement graph. Figure 5.9 (**B**) shows MVEE skeleton agreement graphs have slightly more true edges than Interval skeleton agreement graphs. Unsurprisingly

both methods are more accurate as of the number of data points increases.

5.5 Summary

MVEE is a new method to evaluate the structure of a learned graph when the true graph is unknown. MVEE slightly outperforms InterVal to generate agreement graphs which have more true edges and less skeleton error score. The experimental result shows that MVEE produces agreement graphs that have a higher number of edges than Interval. The MVEE agreement graph is more confident as a proxy of the true graph. The MVEE agreement graph is successfully used as a proxy to the true graph and produces accuracy of 91.56%. Finding the best-learned graph implies we also find the suitable kernel-parameter values for the dataset, because this graph is generated using these kernel parameter values. MVEE is helpful in choosing the value of the kernel parameter for KAFCI if the true graph is unknown.

The proposed method is only based on the structure of the graph without analyzing the statistical meaning. However, it is easy to implement. For future work, it is possible to combine MVEE with the statistical based evaluation method to develop a sophisticated evaluation method when the true graph is unknown.

Chapter 6

Evaluation and Conclusions

6.1 Evaluation of Kernel-based Approach for Learning Causal Graphs from Mixed Data Containing Missing Values

The kernel-based approach is a promising method to learn causal graphs from mixed data containing binary, ordinal, categorical, and continuous variables when the datasets are complete or incomplete. Table 6.1 shows the comparison of the kernel-based approach and other methods. The kernel-based approach (KAPC, KAFCI, and KARFCI), Copula PC [24], PC MXM [70], greedy search Hill-Climbing [60] [50], and *deal* [11] are methods that can be used to learn causal graphs from mixed data. Those methods work for mixed data containing missing values except *deal* and greedy search Hill-Climbing. Table 6.2 shows the list of data types that can be handled by each algorithm.

Table 6.1: Comparison of the kernel-based approach to other methods

Algorithm	**Mixed Data**	**Missing Values**
KAPC	✓	✓
KAFCI and KARFCI	✓	✓
Copula PC	✓	✓
PC MXM	✓	✓
Greedy search based on Hill-Climbing	✓	✗
deal	✓	✗

The kernel-based approach offers better treatment for categorical variables than the Copula method. *deal* is a method for learning causal graphs from mixed data and it is restricted for conditionally Gaussian networks. PC MXM is a method for learning graphs from mixed data and it implements the column-wise imputation method based on median and mode for the datasets that contain missing values. It is not a smart method and

Table 6.2: List of data types that fit for each algorithm

Algorithm	**Binary**	**Ordinal**	**Categorical**	**Continuous**
KAPC	✓	✓	✓	✓
KAFCI/KARFCI	✓	✓	✓	✓
Copula PC	✓	✓	✗	✓
PC MXM	✓	✓	✓	✓
Greedy search Hill-Climbing	✓	✓	✓	✓
deal	✓	✓	✓	✓

might produce misleading data when the 'real' missing value is not a modal value.

In the experiments to learn graphs from mixed data, we compare the performance of the kernel-based approach to the Copula method [24], PC MXM [70], *deal* [11], and greedy search Hill-Climbing [60]. KAPC outperforms *deal*. The SHD scores of KAPC are mostly lower than the SHD score of Copula PC and PC MXM. In general, the SHD scores of KAFCI and KARFCI are lower than the SHD score of Copula FCI. The experimental results confirm that KAPC, KAFCI, and KARFCI work well for complete and incomplete mixed data. When there are no missing values in the datasets, the kernel-based approach runs faster than the Copula method. However, when the datasets contain missing values, the kernel-based approach using kernel extension runs more slowly than the Copula method. The slow running time of the kernel-based approach is caused by matrix computation using kernel extension function and is time-consuming. The greedy search Hill-Climbing outperforms KAPC. However, greedy search Hill-Climbing can not be applied directly to missing values data. The missing values dataset must be pre-processed to treat the missingness before it is implemented to learn graphs using Hill-Climbing and *deal*. The advantage of the kernel-based approach compared to greedy search Hill-Climbing and *deal* is the kernel-based approach can be implemented for FCI and RFCI that allow the presence of latent variables.

Figures 6.1, 6.2, and 6.3 show the comparison of the SHD KAPC for complete data and incomplete data. Figures 6.4, 6.5, and 6.6 show the comparison of the SHD KAFCI for complete data and incomplete data. The **P1, P2, P3, P4, P5** refer to the kernel-based method using different kernel parameter values for complete data, **iKE** refers to the kernel-based approach using kernel extension and **iP1, iP2, iP3, iP4, iP5** refers to the kernel-based approach using different kernel parameters for datasets after imputation. The comparison of SHD scores of the learned graphs generated from complete and incomplete datasets using KAPC and KAFCI indicates that KAPC and KAFCI work for incomplete

data with a small number of missing values as good as complete data. There are no significant SHD scores of KAPC and KAFCI from complete and incomplete data.

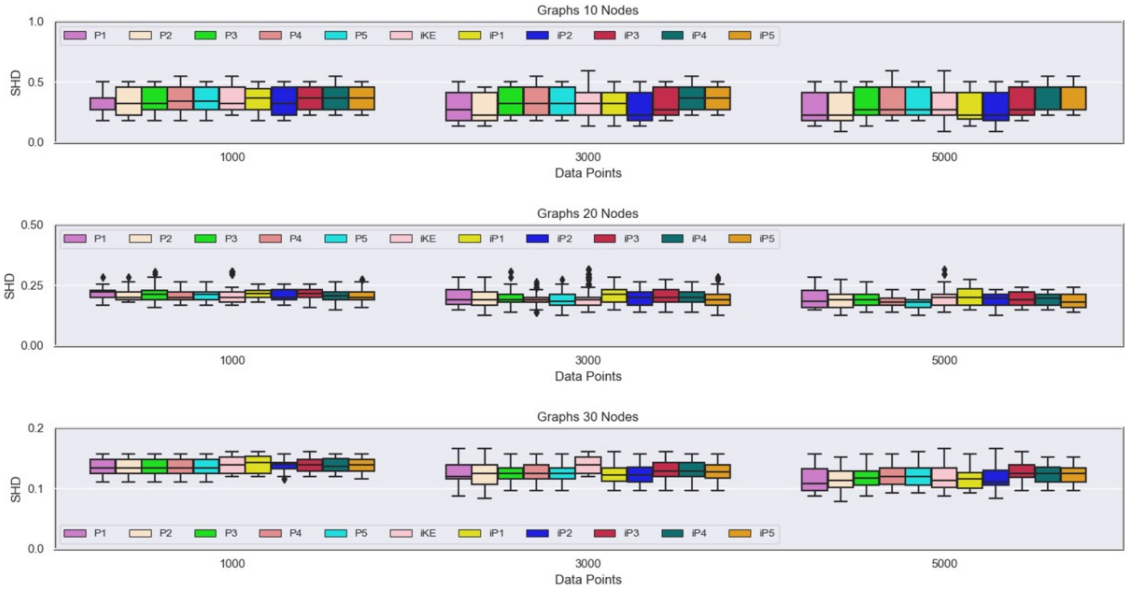


Figure 6.1: Comparison of SHD score KAPC for complete datasets and MAR datasets

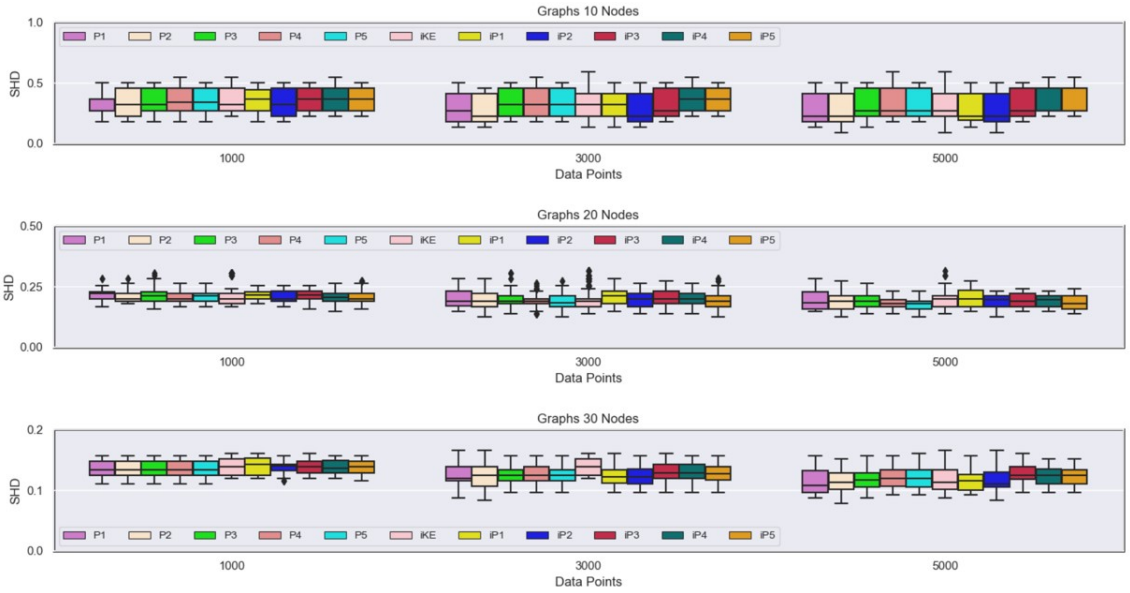


Figure 6.2: Comparison of SHD score KAPC for complete datasets and MCAR datasets

The kernel-based approach is a framework to handle mixed data containing missing values for learning causal graphs when there exists a kernel function that works for this condition. The kernel-based approach is not restricted to specific data distribution. The advantage of the kernel-based approach is this method can be applied to any data type as long as there is a suitable kernel function for the related data type. The kernel-based

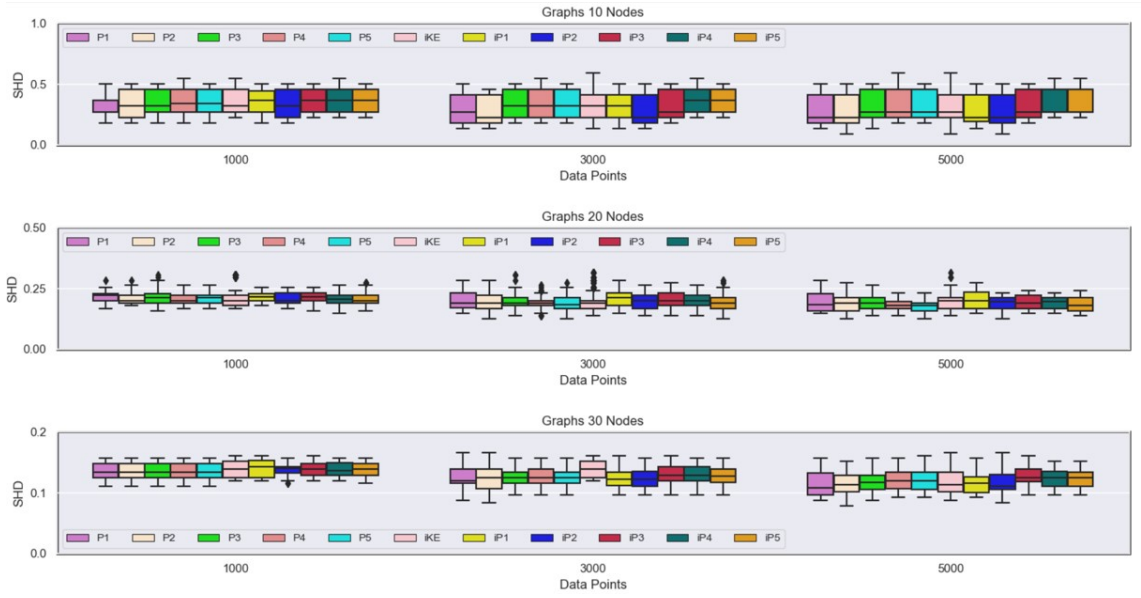


Figure 6.3: Comparison of SHD score KAPC for complete datasets and MNAR datasets

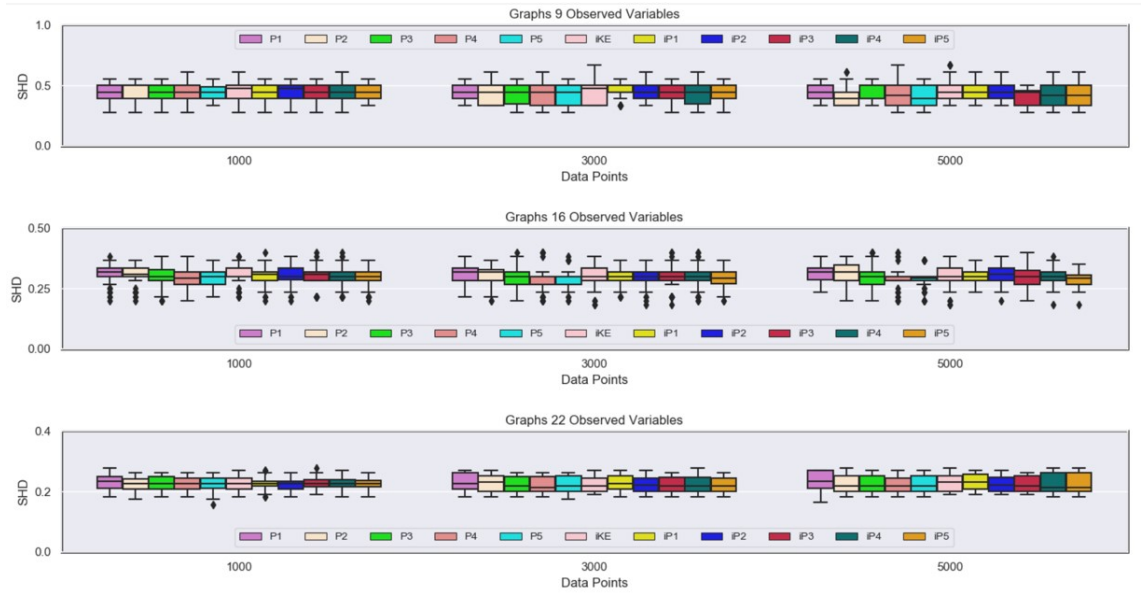


Figure 6.4: Comparison of SHD score KAFCI for complete datasets and MAR datasets

approach depends on the availability of the kernel function for the data type and its condition (e.g. the dataset has missing values). Our investigation to use the kernel alignment to substitute the correlation matrix for the conditional independence test shows that the kernel-based approach can be used to learn causal graphs from mixed datasets not only from datasets under the Conditional Linear Gaussian model but also datasets under circumstances when a graph contains a discrete child with continuous parent. The kernel-based approach can handle more data types than the Copula method proposed by

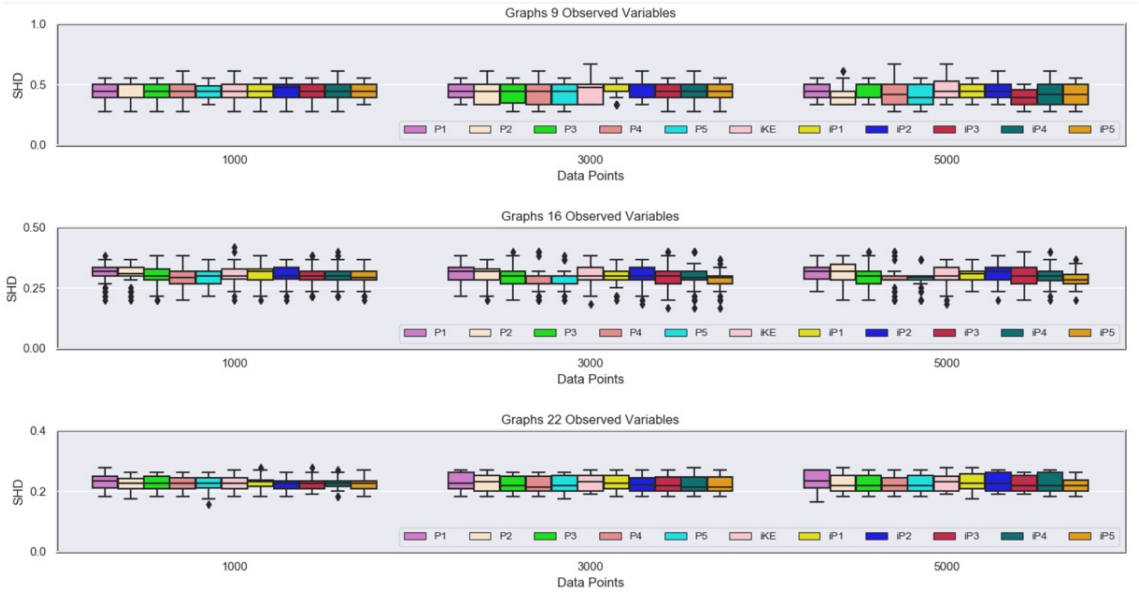


Figure 6.5: Comparison of SHD score KAFCI for complete datasets and MCAR datasets

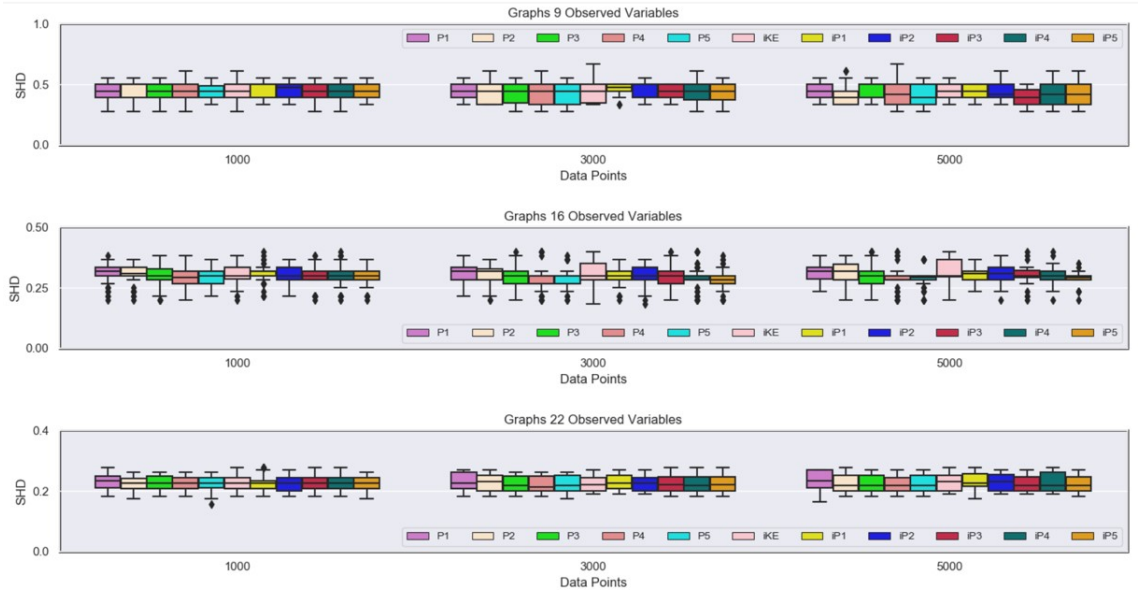


Figure 6.6: Comparison of SHD score KAFCI for complete datasets and MNAR datasets

Cui et al. [24]. The other advantage of the kernel-based approach is not only can it be implemented for PC but also FCI/RFCI that allows the presence of latent variables.

The choice of the kernel parameter values affects the result of the learned graph. The kernel-based approach might produce different learned graphs when it is given different kernel parameter values. Finding the proper kernel parameter values is tricky. We implement MVEE to choose the best-learned graphs from a set of learned graphs output from KAFCI using different kernel parameter values when the ground truth is unknown.

Implementing the Structural Hamming Distance (SHD) method to measure the quality graph structure represented by PAG is acceptable but it is not the best choice. Figure 6.7 shows a true graph **G0** and a learned graph **G1**. There is neither a missing edge nor extra edge in graph **G1** but it has the wrong marked edges. A PAG has six different types of edges, where each edge has two marks on an edge (say at the start and end of the edge). The SHD method was originally developed for PDAG only caring about the wrong orientation edge. It does not include a procedure to handle the edge which has two different marks on each point. For instance, the edges between node $A - B$ and $A - C$ on graph **G1** are assigned as the wrong orientation edges because they don't have exactly the same marks as the true graph **G0** even though they only have one wrong mark. As a consequence, two graphs with different skeletons, the learned graph **G1** and the learned graph **G2**, have the same SHD score. It sounds unfair because the learned graph **G1** has more true edges than the learned graph **G2** based on the skeleton's structure.

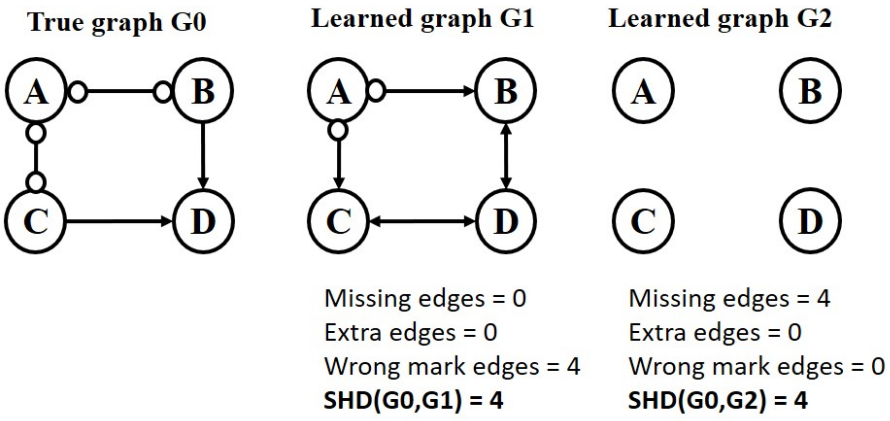


Figure 6.7: Structural Hamming Distance for PAG

6.2 Evaluation of MVEE as a Proxy of the True Graph

Modal Value of Edge Existence (MVEE) is a method to choose the best-learned graph from a set of learned graphs generated from the same dataset when the true graph is unknown. MVEE is developed to create a proxy of the true graph from a set of input graphs represented by FCI-PAG. It is an easy method to evaluate the learned graph when we do not know the true graph. MVEE is developed only based on the graph structure without considering the statistical meaning from the graph structure. The limitation of MVEE is this method is developed for FCI-PAG, where it is possible two different FCI-

PAGs represent a DAG but they have the same skeleton. Although MVEE procedures can be implemented to generate an agreement graph from RFCI-PAGs, this agreement graph has no theoretical support. Unlike FCI-PAG, different RFCI-PAGs for the same underlying DAG may have different skeletons and an RFCI-PAG can correspond to more than one Markov equivalence class of DAGs [20]. It is more difficult to create a proxy of the true graph for RFCI-PAG using a simple method.

Table 6.3: Comparison of InterVal and MVEE

	InterVal	**MVEE**
Graph's type	CPDAG	FCI-PAG
Input	Graphs as whole	Graph's skeleton
Output	Graphs with marks on each edge	Graph's skeleton

MVEE applies the Partial Hamming Distance (PHD) method to compute the mismatch of the learned-graphs to the agreement graph. The PHD method enables more than two different graph structures to have the same PHD score. Suppose, we generate learned graphs from a dataset using KAFCI with three different kernel parameter values $P1, P2, P3$ and we get the three different learned graphs. We generate a proxy of the true graph from those three learned graphs and then compute the skeleton error score. It is possible all learned graphs have the same skeleton error score. In this case, all learned graphs might potentially be the best-learned graphs. In other words, we can not choose a unique graph as the best learned-graph.

6.3 Conclusion

In conclusion, KAPC, KAFCI, and KARFCI are a promising methods for learning causal graphs from mixed data containing missing values. KAPC works for mixed datasets, assuming the datasets have no latent variables. KAFCI and KARFCI allow the presence of latent variables in the mixed data. KAPC, KAFCI, and KARFCI employ the kernel function to compute a kernel matrix from each variable in the mixed data. The kernel matrices represent variables in the feature space. KAPC and KAFCI implement the kernel alignment method to compute the kernel alignment matrix from those kernel matrices. The kernel alignment matrix is a covariance matrix and thus the correlation matrix is then produced. The generated correlation matrix is used as an input for the conditional independence test for the PC, FCI, and RFCI. KAPC, KAFCI, and KARFCI are not restricted to the choices of kernel functions, so they can be implemented to learn causal

graphs from mixed data containing missing values as long as there is a kernel function suitable for the data.

In the experiment using the RBF kernel for continuous variables and the Categorical for discrete variables, the performance of KAPC, KAFCI, and KARFCI are affected by the values of the kernel parameters. KAPC, KAFCI, and KARFCI generate the best-learned graph when they are given the proper kernel parameter values. KAPC, KAFCI, and KARFCI might generate different learned graphs or the same learned graphs from the same dataset when they are given different parameter values.

In a situation when the true graph is unknown, it is difficult to choose the best-learned graphs from a set of learned graphs from the same dataset. MVEE is a method for selecting the best-learned graph from a set of learned graphs generated from the same dataset when the learned graphs are represented by FCI-PAGs. MVEE generates an agreement graph from a set of skeleton input graphs based on the modal values of the edge existence. The agreement graph is used as a proxy of the true graph. The MVEE agreement graph is a graph's skeleton. Each skeleton of the learned graph is compared to their agreement graph and their mismatch is recorded. The number of mismatches is named skeleton error score. The best-learned graph is the learned graph that has the lowest skeleton error score. When MVEE finds the best-learned graph, MVEE also identifies the suitable kernel parameter values for the dataset. The best-learned graph is generated using the particular kernel parameter values, so the kernel parameter values used for this graph are the best among other values for other graphs. Two different skeleton graphs might have the same skeleton error score so that the best learned-graph is not unique. The ground truth of the best-learned graphs is selected from the set of learned graphs generated from the same dataset. First, we compute the SHD score by comparing a set of the learned graphs to the true graph. The ground truth of the best-learned graph is the learned graph which has the lowest SHD score. In the experiment to choose the best-learned graph, MVEE produces an accuracy of 91.56%.

6.4 Future Work

6.4.1 Future Work to Improve the Kernel-based Approach

Two main concerns that need to be improved in the kernel-based approach are how to speed-up running time and how to choose proper kernel parameter values automatically.

Although MVEE can be used to choose the best learn-graph among several input graphs, we still need to generate the learned graphs among a huge choice of kernel parameter values. For future work, the improvement of the kernel-based approach will be focused on two major issues:

6.4.1.1 Decrease the Running Time of the Kernel-based Approach using Parallel Computing

Parallel computing is a computation method where many calculations or executions of the process can be done simultaneously. Parallel computing is a common method applied for matrix computation. The kernel-based approach involves matrix computation in which it is possible to apply parallel computing to increase the running time. Figure 6.8 shows the schema for computing a kernel alignment matrix from a dataset. Suppose a dataset consists of p variables and n data points for each variable. The first step is computing kernel matrices $K = \{K_1, K_2, K_3, .., K_p\}$ and each kernel matrix corresponds to each variable. In sequential computing, computing kernel matrices $K = \{K_1, K_2, K_3, .., K_p\}$ will be done one by one from K_1 to K_p and it is time consuming. To increase the running time of the kernel-based method, computation of kernel matrices can be parallelized, so the computation of K can be done at the same time. It does not affect the output because each kernel matrix K can be computed independently. Parallel computing also can be implemented to compute a kernel alignment matrix.

6.4.1.2 Automatic Tuning Parameter Values

The issue in the kernel-based approach for learning causal graphs from mixed data containing missing values is how to choose the best kernel function for each variable and how to choose the proper values of the kernel parameter. The kernel-based approach is not limited to the choice of kernel functions. There are a lot of kernel functions that are possible to be implemented for particular variables. It also opens the opportunity to create new kernel functions for causal learning from incomplete mixed data.

Unlike implementing a kernel function for classification problems, tuning kernel parameters for causal learning problem is more challenging. Moreover, the difficulties of choosing the kernel parameter values are increased when we learn a graph from the real data where the true graph is unknown. In this research, we introduce MVEE to choose the best kernel parameter values. However, MVEE is not run automatically because it still

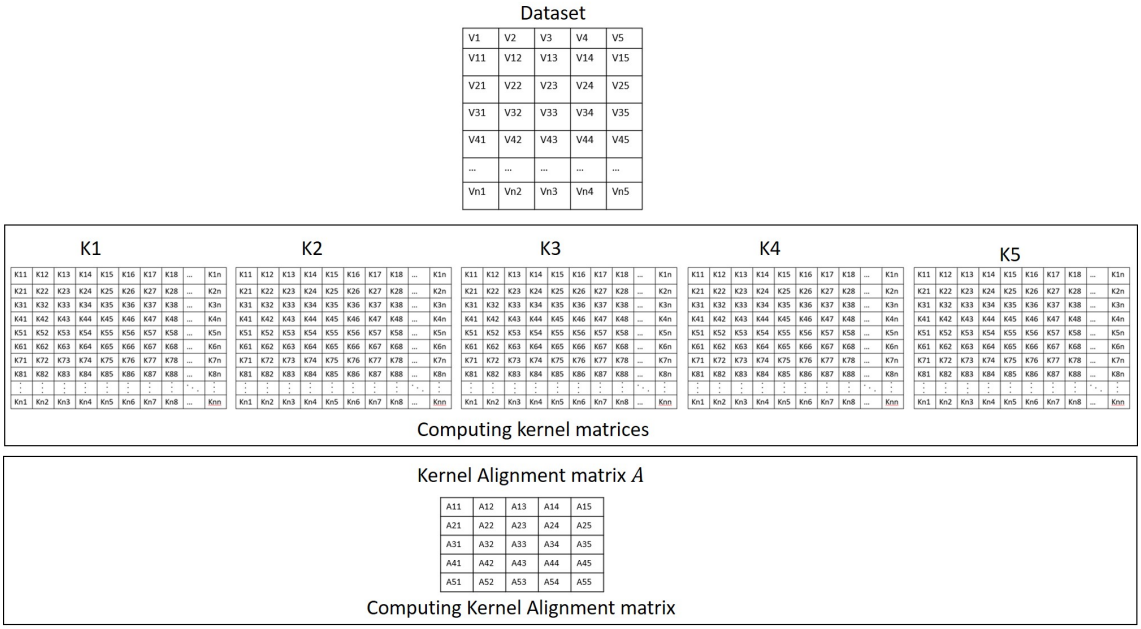


Figure 6.8: The schema for computing kernel alignment matrix

requires inputting kernel parameter values from the user. For future research, it is possible to apply a hybrid of searching method and MVEE to find the best kernel-parameter values in the range of possible values $[a, b]$.

6.4.1.3 Creating New Kernel Functions for Missing Values Data

There are not many choices of kernel functions that work for missing value datasets. It opens the opportunity to develop a new kernel function for missing value data. The new kernel functions are expected to support the kernel-based approach as a framework for learning causal graphs from mixed data containing missing values.

6.4.2 Improving the Performance of MVEE to Evaluate the Graph Structure when the True Graph is Unknown

Developing a method to evaluate graph structure when the true graph is unknown is one of the interesting topics in learning graphical models. It is very useful when learning graphs from real datasets where the true graphs are often unknown. MVEE is a promising graph evaluation method when the true graph is unknown. MVEE implements Partial Hamming Distance to compute the mismatch between a proxy of the true graph and the learned graph and it allows more than two different graphs to produce the same score. As a consequence, MVEE might output all input graphs as the best learned graphs even

though they have different graph structures when they have the same score as their proxy graph. Structural Hamming Distance is less appropriate to implement compared to two PAGs. Suppose, three edges $E1 : A \leftrightarrow B$, $E2 : A \circ\!\!\rightarrow B$ and $E3 : A \leftarrow B$, the SHD score between $E1$ and $E3$ is the same as the SHD score between $E2$ and $E3$. To improve MVEE, firstly, a new standard measurement needs to be created for comparing two PAGs when considering the types of edges. The MVEE agreement graph is only built based on the modal values of the present edges in the input graphs. Secondly, it is useful to implement a scoring function to compute a score that measures the statistical relationship. The best-learned graphs are chosen based in the statistical score and structural score.

In this research, MVEE is limited for FCI-PAG. The MVEE might be improved to create a proxy graph from RFCI-PAGs and/or FCI-PAGs and RFCI-PAGs.

Appendix A

The first appendix

A.1 PC Algorithm

Algorithm 1 PC algorithm

Require: Conditional Independence information among all variables in V, and an ordering order (V) on the variables

Ensure: Return output graph (C) and separation set (sepset)

1: Find the skeleton C and separation sets using Algorithm 2
2: Orient unshielded triplets in the skeleton C based on the separation sets
3: In orient as many of the remaining undirected edges as possible by repeated application of rule R1-R3

Algorithm 2 Creating Skeleton on PC algorithm

Require: Conditional independence information among all variables in V, and ordering order (V) on the variables

Ensure: C, Sepset

1: Form the complete undirected graph C on the vertex set V
2: Let ℓ=-1
3: **repeat**
4: Let $\ell = \ell + 1$
5: **repeat**
6: Select a (new) ordered pair of vertices (X_i, X_j) that are adjacent in C and satisfy $|adj(C, X_i) \backslash \{X_j\}| \geq \ell$ using order V
7: **repeat**
8: Chose a(new) set $S \subseteq adj(C, X_i) \setminus \{X_j\}$ with $|S| = \ell$, using order V
9: **if** X_i and X_j are conditionally independent given S **then**
10: Delete edge $X_i - X_j$ from C
11: Let $Sepset(X_i, X_j) = Sepset(X_j, X_i) = S$
12: **end if**
13: **until** X_i and X_j are no longer adjacent in C or all $S \subseteq adj(C, X_i) \setminus \{X_j\}$ with $|S| = \ell$ have been considered
14: **until** all ordered pairs of adjacent vertices (X_i, X_j) in C with $|adj(C, X_i) \setminus \{X_j\}| \geq \ell$ have been considered
15: **until** all pairs of adjacent vertices (X_i, X_j) in C satisfy $|adj(C, X_i) \setminus \{X_j\}| \leq \ell$

A.2 Fast Causal Inference (FCI)

Algorithm 3 FCI algorithm

Require: Conditional Independence information among all variables in X given S
Ensure: C, Sepset
1: Finding an initial skeleton C using algorithm 4, separation sets (sepset) and unshielded triple list $\mathcal{M}$
2: Orienting v-structures using algorithm 5 (update C)
3: Finding final skeleton using algorithm 6 (update C and sepset)
4: Orienting v-structures using algorithm 5 (update C)
5: Orienting as many as edge marks as possible using rule (R1-R10) (update C)

Algorithm 4 Obtaining an initial skeleton for FCI

Require: Conditional independence information among all variables in X given S
Ensure: C, Sepset, $\mathcal{M}$
1: Form the complete graph C on the vertex set X with edges $\circ\!\!-\!\!\circ$
2: Let $\ell = -1$
3: **repeat**
4: Let $\ell = \ell + 1$
5: **repeat**
6: **for** all vertices X_i in C **do**
7: Compute $adj(C, X_i)$
8: **end for**
9: Select a (new) ordered pair vertices (X_i, X_j) that are adjacent in C and satisfy $|adj(C, X_i) \setminus \{X_j\}| \geq \ell$
10: **repeat**
11: Choose a (new) set $Y \subseteq adj(C, X_i) \setminus \{X_j\}$ with $|Y| = \ell$
12: **if** X_i and X_j are conditionally independent given $Y \cup S$ **then**
13: Delete edge $X_i \circ\!\!-\!\!\circ X_j$ from C
14: Let $Sepset(X_i, X_j) = Sepset(X_j, X_i) = Y$
15: **end if**
16: **until** X_i and X_j are no longer adjacent in C or all $Y \subseteq adj(C, X_i)$ X_j with $|Y| = \ell$ have been considered
17: **until** all pairs of adjacent vertices (X_i, X_j) in C with $adj(C, X_i) \setminus \{X_j\}| \geq \ell$ have been considered
18: **until** all pairs of adjacent vertices (X_i, X_j) in C satisfy $|adj(C, X_i) \setminus \{X_j\}| \leq \ell$
19: Form a list $\mathcal{M}$ of all unshielded triplets $\langle X_k, ., X_m \rangle$ (i.e. the middle vertex is left unspecified) in C with $k < m$

Algorithm 5 Orienting v-structures in the FCI Algorithm

Require: Initial skeleton C, separation sets (sepset) and unshielded triple list $\mathcal{M}$
Ensure: C, Sepset
1: **for** all elements $\langle X_i, X_j, X_k \rangle$ of $\mathcal{M}$ **do**
2: **if** $X_i \notin Sepset(X_i, X_k)$ **then**
3: Orient $X_i *\!\!-\!\!\circ X_j \circ\!\!-\!\!* X_k$ as $X_i *\!\!\rightarrow X_j \leftarrow\!\!* X_k$ in C
4: **end if**
5: **end for**

Definition 1 [20]. Let C be a graph with any of the following edge types: $\circ\!\!-\!\!\circ$, $\circ\!\!\rightarrow$, $\leftrightarrow$. Possible-d-separation(X_i, X_j) in C, denoted in shorthand by pds(C, X_j, X_j), is defined as follows: $X_k \in pds(C, X_i, X_j)$ if and only if there is a path π between X_i and X_k in C such that for every subpath $\langle X_m, X_l, X_h \rangle$ of π, X_l is a collider on the subpath in C or $\langle X_m, X_l, X_h \rangle$ is a triangle in C.

Algorithm 6 Obtaining the final skeleton in the FCI Algorithm

Require: Partially oriented graph C and separation sets (sepset)
Ensure: C, Sepset, $\mathcal{M}$

for all vertices X_i in C **do**
  Compute pds$(C, X_i, .)$ as defined in Definition 1
  for all vertices $X_j \in adj(C, X_i)$ **do**
    Let $\ell = -1$
    repeat
      Let $\ell = \ell + 1$
      repeat
        Choose a (new) set $Y \subseteq pds(C, X_i, .) \setminus \{X_j\}$ with $|Y| = \ell$
        if X_i and X_j are conditional independent given $Y \cup S$ **then**
          Delete edge $X_i \ast\!\!-\!\!\ast X_j$ from C
          Let $Sepset(X_i, X_j) = Sepset(X_j, X_i) = Y$
        end if
      until X_i and X_j are no longer adjacent in C or all $Y \subseteq pds(C, X_i, .) \setminus \{X_j\}$ with $|Y| = \ell$ have been considered
    until X_i and X_j are no longer adjacent in C or $|pds(C, X_i, .) \setminus X_j| \leq \ell$
  end for
end for
Reorient all edges in C as $\circ\!\!-\!\!\circ$
Form a list $\mathcal{M}$ of all unshielded triplets $\langle X_k, ., X_m \rangle$ in C with $k < m$

A.3 Really Fast Causal Inference (RFCI)

Algorithm 7 RFCI algorithm

Require: Conditional Independence information among all variables in X given S
Ensure: C, Sepset
1: Finding an initial skeleton C using Algorithm 8 separation sets (sepset) and unshielded triple list($\mathcal{M}$)
2: Orient v-structures using algorithm 9 (update C and sepset)
3: Orient as many as edge marks as possible (update C and sepset)

Algorithm 8 Obtaining an initial skeleton for RFCI

Require: Conditional independence information among all variables in X given S
Ensure: C, Sepset, $\mathcal{M}$
1: Form the complete graph C on the vertex set X with edges $\circ\!\!-\!\!\circ$
2: Let $\ell = -1$
3: **repeat**
4: Let $\ell = \ell + 1$
5: **repeat**
6: **for** all vertices X_i in C **do**
7: Compute $adj(C, X_i)$
8: **end for**
9: Select a (new) ordered pair vertices (X_i, X_j) that are adjacent in C and satisfy $|adj(C, X_i) \setminus \{X_j\}| \geq \ell$
10: **repeat**
11: Choose a (new) set $Y \subseteq adj(C, X_i) \setminus X_j$ with $|Y| = \ell$
12: **if** X_i and X_j are conditionally independent given $Y \cup S$ **then**
13: Delete edge $X_i \circ\!\!-\!\!\circ X_j$ from C
14: Let $Sepset(X_i, X_j) = Sepset(X_j, X_i) = Y$
15: **end if**
16: **until** X_i and X_j are no longer adjacent in C or all $Y \subseteq adj(C, X_i) \setminus \{X_j\}$ with $|Y| = \ell$ have been considered
17: **until** all ordered pairs of adjacent vertices (X_i, X_j) in C with $adj(C, X_i) \setminus \{X_i\} \geq \ell$ have been considered
18: **until** all pairs of adjacent vertices (X_i, X_j) in C satisfy $adj(C, X_i) \setminus \{X_i\} \leq \ell$
19: Form a list $\mathcal{M}$ of all unshielded triplets $\langle X_k, ., X_m \rangle$ (i.e. the middle vertex is left unspecified) in C with $k < m$

Algorithm 9 Orienting v-structures in the RFCI Algorithm

Require: Initial skeleton C, separation sets (sepset) and unshielded triple list $\mathcal{M}$ from algorithm 8
Ensure: C, Sepset

Let $\mathcal{L}$ be an empty list
while $\mathcal{M}$ is non-empty **do**
  Choose an unshielded triple $\langle X_i, X_j, X_k \rangle$ from $\mathcal{M}$
  if both (X_i and X_j) and (X_j and X_k) are conditionally dependent given $(Sepset(X_i, X_k) \setminus \{X_j\} \cup S)$ **then**
    Add $\langle X_i, X_j, X_k \rangle$ to $\mathcal{L}$
  else

    for $r \in i, k$ **do**
      if X_r and X_j are conditionally independent given $(Sepset(X_i, X_k) \setminus \{X_j\} \cup S)$ **then**
        Find a minimal separating set $Y \subseteq Sepset(X_i, X_k) \setminus \{X_k\}$ for X_r and X_j
        Let $Sepset(X_r, X_j) = Sepset(X_j, X_r) = Y$
        Add to $\mathcal{M}$ all triplets $\langle X_{min(r,j),.,X_{max(r,j)}} \rangle$ that form a triangle in C
        Delete from $\mathcal{M}$ and $\mathcal{L}$ all triplets containing (X_r, X_j):$\langle X_r, X_j, . \rangle$,$\langle X_j, X_r, . \rangle$,$\langle ., X_j, X_r \rangle$, and $\langle ., X_r, X_j \rangle$
        Delete edge $X_r \ast\!\!-\!\!\ast X_j$ from C
      end if
    end for
  end if
  Remove $\langle X_i, X_j, X_k \rangle$ from $\mathcal{M}$
end while
for all elements $\langle X_i, X_j, X_k \rangle$ of $\mathcal{L}$ **do**
  if $X_j \notin Sepset(X_i, X_k)$ and both edges $X_i \ast\!\!-\!\!\ast X_j$ and $X_j \ast\!\!-\!\!\ast X_k$ are parent in C **then**
    Orient $X_i \ast\!\!-\!\!\circ X_j \circ\!\!-\!\!\ast X_k$ as $X_i \ast\!\!\rightarrow X_j \leftarrow\!\!\ast X_k$ in C
  end if
end for

Algorithm 10 Orientation rules for RFCI-algorithm

Require: Skeleton C, separation set(sepset) and unshielded triplet list $\mathcal{M}$ from Algorithm 9
Ensure: C, Sepset

1: **repeat**
2: Orient as many edge marks as possible in by applying rules(R1) - (R3) (see Chapter 2.5.3)
3: **while** a triangle between three vertices $\langle X_l, X_j, X_k \rangle$ exists such that $X_j \circ\!\!-\!\!* X_k$, $X_l \leftarrow\!\!* X_j$, and $X_l \rightarrow X_k$ in C **do**
4: Find a shortest discriminating path for $\langle X_l, X_j, X_k \rangle$
5: **if** a discriminating path π for $\langle X_l, X_j, X_k \rangle$ exists between X_k and say X_i **then**
6: **for** all pairs of vertices (X_r, X_q) that are adjacent on π **do**
7: $\ell = -1$
8: **repeat**
9: Let $\ell = \ell + 1$
10: **repeat**
11: Choose a (new) subset $Y \subseteq Sepset(X_i, X_k) \setminus \{X_r, X_q\}$ with $|Y| = \ell$
12: **if** X_r and X_q are conditionally independent given $Y \cup S$ **then**
13: Let $Sepset(X_r, X_q) = Sepset(X_q, X_r) = Y$
14: Create a list $\mathcal{M}$ of all triplets $\langle X_r, ., X_q \rangle$ with $r < q$ that form a triangle in C
15: Delete edge $X_r \ast\!\!-\!\!\ast X_q$ from C
16: Run Algorithm 9 with input $\{C, Sepset, \mathcal{M}\}$ and update C and sepset
17: **end if**
18: **until** X_r and X_q are no longer adjacent in C or all $Y \subseteq Sepset(X_i, X_k) \setminus \{X_r, X_q\}$ with $|Y| = \ell$ have been considered
19: **until** $|Sepset(X_i, X_k) \setminus \{X_r, X_q\}| < \ell$
20: **end for**
21: **if** If all of the edges between adjacent vertices on π are present in C then **then**
22: **if** $X_i \in Speset(X_i, X_k)$ **then**
23: Orient $X_j \circ\!\!-\!\!* X_k$ as $X_j \rightarrow X_k$ in C
24: **else**
25: Orient the triple $X_l \leftarrow\!\!* X_j \circ\!\!-\!\!* X_k$ as $X_l \leftrightarrow X_j \leftrightarrow X_k$ in C
26: **end if**
27: **end if**
28: **end if**
29: **end while**
30: Orient as many edge marks as possible in C by applying rules (R5)-(R10) (see Chapter 2.5.3)
31: **until** C remains unchanged throughout lines 2-30 above

Appendix B

The second appendix

Table B.1: Kernel parameter values for G1

		X	**Y**	**Z**
Case 1	P1	$\theta = 0.00001$	$\theta = 0.00001$	$\theta = 0.00001$
	P2	$\theta = 0.01$	$\theta = 0.01$	$\theta = 0.01$
	P3	$\theta = 1$	$\theta = 1$	$\theta = 1$
	P4	$\theta = 2$	$\theta = 2$	$\theta = 2$
	P5	$\theta = 5$	$\theta = 5$	$\theta = 5$
	P6	$\theta = 10$	$\theta = 10$	$\theta = 10$
Case 2	P1	$\theta = 2$	$\theta = 2$	$\sigma = 0.0001$
	P2	$\theta = 2$	$\theta = 2$	$\sigma = 0.01$
	P3	$\theta = 2$	$\theta = 2$	$\sigma = 2$
	P4	$\theta = 2$	$\theta = 2$	$\sigma = 5$
	P5	$\theta = 2$	$\theta = 2$	$\sigma = 10$
	P6	$\theta = 2$	$\theta = 2$	$\sigma = 100$
Case 3	P1	$\theta = 2$	$\sigma = 0.00001$	$\sigma = 0.00001$
	P2	$\theta = 2$	$\sigma = 0.01$	$\sigma = 0.01$
	P3	$\theta = 2$	$\sigma = 2$	$\sigma = 2$
	P4	$\theta = 2$	$\sigma = 0.00001$	$\sigma = 2$
	P5	$\theta = 2$	$\sigma = 2$	$\sigma = 0.00001$
	P6	$\theta = 2$	$\sigma = 2$	$\sigma = 0.1$
Case 4	P1	$\sigma = 0.000001$	$\sigma = 0.000001$	$\sigma = 0.000001$
	P2	$\sigma = 0.00001$	$\sigma = 0.00001$	$\sigma = 0.00001$
	P3	$\sigma = 0.0001$	$\sigma = 0.0001$	$\sigma = 0.0001$
	P4	$\sigma = 0.01$	$\sigma = 0.01$	$\sigma = 0.01$
	P5	$\sigma = 2$	$\sigma = 2$	$\sigma = 2$
	P6	$\sigma = 10$	$\sigma = 10$	$\sigma = 10$
Case 5	P1	$\sigma =$1E-14	$\theta = 2$	$\theta = 2$
	P2	$\sigma = 0.000001$	$\theta = 2$	$\theta = 2$
	P3	$\sigma = 0.00001$	$\theta = 2$	$\theta = 2$
	P4	$\sigma = 0.01$	$\theta = 2$	$\theta = 2$
	P5	$\sigma = 10$	$\theta = 2$	$\theta = 2$
	P6	$\sigma = 100$	$\theta = 2$	$\theta = 2$
Case 6	P1	$\sigma = 0.000001$	$\sigma = 0.000001$	$\theta = 2$
	P2	$\sigma = 0.0001$	$\sigma = 0.0001$	$\theta = 2$
	P3	$\sigma = 0.01$	$\sigma = 0.01$	$\theta = 2$
	P4	$\sigma = 0.1$	$\sigma = 0.1$	$\theta = 2$
	P5	$\sigma = 10$	$\sigma = 10$	$\theta = 2$
	P6	$\sigma = 100$	$\sigma = 100$	$\theta = 2$
Case 7	P1	$\theta = 2$	$\sigma = 0.00001$	$\theta = 2$
	P2	$\theta = 2$	$\sigma = 0.0001$	$\theta = 2$
	P3	$\theta = 2$	$\sigma = 0.01$	$\theta = 2$
	P4	$\theta = 2$	$\sigma = 0.1$	$\theta = 2$
	P5	$\theta = 2$	$\sigma = 10$	$\theta = 2$
	P6	$\theta = 2$	$\sigma = 100$	$\theta = 2$

Table B.2: Kernel parameter values for G2

		X	**Y**	**Z**
Case 1	P1	$\theta = 0.00001$	$\theta = 0.00001$	$\theta = 0.00001$
	P2	$\theta = 0.01$	$\theta = 0.01$	$\theta = 0.01$
	P3	$\theta = 1$	$\theta = 1$	$\theta = 1$
	P4	$\theta = 2$	$\theta = 2$	$\theta = 2$
	P5	$\theta = 5$	$\theta = 5$	$\theta = 5$
	P6	$\theta = 10$	$\theta = 10$	$\theta = 10$
Case 2	P1	$\theta = 2$	$\theta = 2$	$\sigma = 0.0001$
	P2	$\theta = 2$	$\theta = 2$	$\sigma = 0.01$
	P3	$\theta = 2$	$\theta = 2$	$\sigma = 2$
	P4	$\theta = 2$	$\theta = 2$	$\sigma = 5$
	P5	$\theta = 2$	$\theta = 2$	$\sigma = 10$
	P6	$\theta = 2$	$\theta = 2$	$\sigma = 100$
Case 3	P1	$\theta = 2$	$\sigma =$1E-13	$\sigma =$1E-13
	P2	$\theta = 2$	$\sigma =$1E-11	$\sigma =$1E-11
	P3	$\theta = 2$	$\sigma =$1E-12	$\sigma =$1000
	P4	$\theta = 2$	$\sigma =$1E-11	$\sigma =$100
	P5	$\theta = 2$	$\sigma =$1E-14	$\sigma =$100000
	P6	$\theta = 2$	$\sigma =$1E-15	$\sigma =$10000
Case 4	P1	$\sigma = 0.00001$	$\sigma = 0.00001$	$\sigma = 100$
	P2	$\sigma = 0.0001$	$\sigma = 100$	$\sigma = 0.00001$
	P3	$\sigma = 0.001$	$\sigma = 10$	$\sigma = 100$
	P4	$\sigma = 0.01$	$\sigma = 100$	$\sigma = 10$
	P5	$\sigma = 0.1$	$\sigma = 10$	$\sigma = 10$
	P6	$\sigma = 10$	$\sigma = 0.00001$	$\sigma = 100$
Case 5	P1	$\sigma = 0.00001$	$\theta = 2$	$\theta = 2$
	P2	$\sigma = 0.01$	$\theta = 2$	$\theta = 2$
	P3	$\sigma = 1$	$\theta = 2$	$\theta = 2$
	P4	$\sigma = 10$	$\theta = 2$	$\theta = 2$
	P5	$\sigma = 100$	$\theta = 2$	$\theta = 2$
	P6	$\sigma = 1000$	$\theta = 2$	$\theta = 2$
Case 6	P1	$\sigma =$1E-07	$\sigma =$1E-12	$\theta = 2$
	P2	$\sigma =$1E-08	$\sigma =$1E-12	$\theta = 2$
	P3	$\sigma =$1E-12	$\sigma =$1E-12	$\theta = 2$
	P4	$\sigma =$1E-14	$\sigma =$1E-11	$\theta = 2$
	P5	$\sigma =$1E-15	$\sigma = 0.000000001$	$\theta = 2$
	P6	$\sigma =$1E-16	$\sigma = 0.00000001$	$\theta = 2$
Case 7	P1	$\theta = 2$	$\sigma = 0.00001$	$\theta = 2$
	P2	$\theta = 2$	$\sigma = 0.01$	$\theta = 2$
	P3	$\theta = 2$	$\sigma = 0.9$	$\theta = 2$
	P4	$\theta = 2$	$\sigma = 10$	$\theta = 2$
	P5	$\theta = 2$	$\sigma = 100$	$\theta = 2$
	P6	$\theta = 2$	$\sigma = 1000$	$\theta = 2$

Table B.3: Kernel parameter values for G3

		X	**Y**	**Z**
Case 1	P1	$\theta = 0.00001$	$\theta = 0.00001$	$\theta = 0.00001$
	P2	$\theta = 0.01$	$\theta = 0.01$	$\theta = 0.01$
	P3	$\theta = 1$	$\theta = 1$	$\theta = 1$
	P4	$\theta = 2$	$\theta = 2$	$\theta = 2$
	P5	$\theta = 5$	$\theta = 5$	$\theta = 5$
	P6	$\theta = 10$	$\theta = 10$	$\theta = 10$
Case 2	P1	$\theta = 2$	$\theta = 2$	$\sigma = 0.0001$
	P2	$\theta = 2$	$\theta = 2$	$\sigma = 0.01$
	P3	$\theta = 2$	$\theta = 2$	$\sigma = 2$
	P4	$\theta = 2$	$\theta = 2$	$\sigma = 5$
	P5	$\theta = 2$	$\theta = 2$	$\sigma = 10$
	P6	$\theta = 2$	$\theta = 2$	$\sigma = 100$
Case 3	P1	$\theta = 2$	$\sigma = 0.00001$	$\sigma = 0.00001$
	P2	$\theta = 2$	$\sigma = 0.01$	$\sigma = 0.01$
	P3	$\theta = 2$	$\sigma = 2$	$\sigma = 2$
	P4	$\theta = 2$	$\sigma = 0.00001$	$\sigma = 2$
	P5	$\theta = 2$	$\sigma = 2$	$\sigma = 0.00001$
	P6	$\theta = 2$	$\sigma = 2$	$\sigma = 0.1$
Case 4	P1	$\sigma = 0.00001$	$\sigma = 0.00001$	$\sigma = 0.00001$
	P2	$\sigma = 10$	$\sigma = 10$	$\sigma = 10$
	P3	$\sigma = 300$	$\sigma = 300$	$\sigma = 300$
	P4	$\sigma = 500$	$\sigma = 500$	$\sigma = 500$
	P5	$\sigma = 1000$	$\sigma = 1000$	$\sigma = 1000$
	P6	$\sigma = 3000$	$\sigma = 3000$	$\sigma = 3000$
Case 5	P1	$\sigma = 10$	$\theta = 2$	$\theta = 2$
	P2	$\sigma = 0.00001$	$\theta = 2$	$\theta = 2$
	P3	$\sigma =$1E-15	$\theta = 2$	$\theta = 2$
	P4	$\sigma =$1E-16	$\theta = 2$	$\theta = 2$
	P5	$\sigma =$1E-17	$\theta = 2$	$\theta = 2$
	P6	$\sigma =$1E-18	$\theta = 2$	$\theta = 2$
Case 6	P1	$\sigma = 30$	$\sigma = 30$	$\theta = 2$
	P2	$\sigma = 300$	$\sigma = 300$	$\theta = 2$
	P3	$\sigma = 3000$	$\sigma = 3000$	$\theta = 2$
	P4	$\sigma = 30000$	$\sigma = 30000$	$\theta = 2$
	P5	$\sigma = 300000$	$\sigma = 30000$	$\theta = 2$
	P6	$\sigma = 3000000$	$\sigma = 3000000$	$\theta = 2$
Case 7	P1	$\theta = 2$	$\sigma = 0.00001$	$\theta = 2$
	P2	$\theta = 2$	$\sigma = 0.1$	$\theta = 2$
	P3	$\theta = 2$	$\sigma = 2$	$\theta = 2$
	P4	$\theta = 2$	$\sigma = 5$	$\theta = 2$
	P5	$\theta = 2$	$\sigma = 10$	$\theta = 2$
	P6	$\theta = 2$	$\sigma = 100$	$\theta = 2$

Table B.4: True Positive Rate (TPR) and False Positive Rate (FPR) Graph G1 N = 1000

		P1		P2		P3		P4		P5		P6	
		TPR	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**
Case1	$\alpha = 0.01$	-	1	-	1	-	0.97	-	0.97	-	0.97	-	0.97
	$\alpha = 0.05$	-	1	-	1	-	0.93	-	0.97	-	0.97	-	0.97
	$\alpha = 0.1$	-	1	-	1	-	0.93	-	0.93	-	0.93	-	0.93
	$\alpha = 0.5$	-	1	-	1	-	0.76	-	0.8	-	0.8	-	0.8
	$\alpha = 0.7$	-	1	-	1	-	0.5	-	0.5	-	0.5	-	0.5
Case2	$\alpha = 0.01$	-	0.23	-	0.76	-	0.67	-	0.67	-	0.7	-	0.7
	$\alpha = 0.05$	-	0.23	-	0.63	-	0.63	-	0.63	-	0.63	-	0.66
	$\alpha = 0.1$	-	0.2	-	0.53	-	0.56	-	0.6	-	0.6	-	0.63
	$\alpha = 0.5$	-	0.06	-	0.36	-	0.3	-	0.33	-	0.36	-	0.33
	$\alpha = 0.7$	-	0	-	0.33	-	0.2	-	0.16	-	0.2	-	0.23
Case3	$\alpha = 0.01$	-	0.23	-	0.3	-	0.57	-	0.63	-	0.53	-	0.47
	$\alpha = 0.05$	-	0.23	-	0.27	-	0.53	-	0.6	-	0.53	-	0.3
	$\alpha = 0.1$	-	0.2	-	0.23	-	0.5	-	0.5	-	0.47	-	0.27
	$\alpha = 0.5$	-	0.06	-	0.03	-	0.37	-	0.4	-	0.2	-	0.06
	$\alpha = 0.7$	-	0	-	0	-	0.13	-	0.27	-	0.1	-	0.03
Case4	$\alpha = 0.01$	-	0.06	-	0.06	-	0.06	-	0.2	-	0.73	-	0.73
	$\alpha = 0.05$	-	0	-	0	-	0	-	0.2	-	0.63	-	0.7
	$\alpha = 0.1$	-	0	-	0	-	0	-	0.13	-	0.63	-	0.66
	$\alpha = 0.5$	-	0	-	0	-	0	-	0.1	-	0.43	-	0.4
	$\alpha = 0.7$	-	0	-	0	-	0	-	0.06	-	0.33	-	0.36
Case5	$\alpha = 0.01$	-	0.46	-	0.46	-	0.46	-	0.46	-	0.6	-	0.63
	$\alpha = 0.05$	-	0.36	-	0.36	-	0.36	-	0.36	-	0.56	-	0.63
	$\alpha = 0.1$	-	0.3	-	0.3	-	0.3	-	0.3	-	0.56	-	0.63
	$\alpha = 0.5$	-	0.1	-	0.1	-	0.1	-	0.1	-	0.3	-	0.43
	$\alpha = 0.7$	-	0.06	-	0.06	-	0.06	-	0.06	-	0.2	-	0.36
Case6	$\alpha = 0.01$	-	0.43	-	0.43	-	0.43	-	0.46	-	0.63	-	0.73
	$\alpha = 0.05$	-	0.33	-	0.33	-	0.3	-	0.3	-	0.6	-	0.7
	$\alpha = 0.1$	-	0.23	-	0.23	-	0.2	-	0.16	-	0.56	-	0.66
	$\alpha = 0.5$	-	0.06	-	0.06	-	0.06	-	0.06	-	0.4	-	0.3
	$\alpha = 0.7$	-	0.06	-	0.06	-	0.06	-	0.06	-	0.3	-	0.13
Case7	$\alpha = 0.01$	-	0.66	-	0.66	-	0.66	-	0.7	-	0.7	-	0.76
	$\alpha = 0.05$	-	0.66	-	0.66	-	0.66	-	0.66	-	0.7	-	0.73
	$\alpha = 0.1$	-	0.63	-	0.63	-	0.63	-	0.63	-	0.7	-	0.7
	$\alpha = 0.5$	-	0.6	-	0.6	-	0.6	-	0.6	-	0.63	-	0.66
	$\alpha = 0.7$	-	0.6	-	0.6	-	0.6	-	0.6	-	0.6	-	0.6

Table B.5: True Positive Rate (TPR) and False Positive Rate (FPR) Graph G1 N = 3000

		P1		P2		P3		P4		P5		P6	
		TPR	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**
Case1	$\alpha = 0.01$	-	1	-	1	-	0.93	-	0.93	-	0.93	-	0.93
	$\alpha = 0.05$	-	1	-	1	-	0.9	-	0.9	-	0.9	-	0.9
	$\alpha = 0.1$	-	1	-	1	-	0.83	-	0.83	-	0.83	-	0.83
	$\alpha = 0.5$	-	1	-	1	-	0.5	-	0.5	-	0.5	-	0.5
	$\alpha = 0.7$	-	1	-	1	-	0.4	-	0.36	-	0.36	-	0.36
Case2	$\alpha = 0.01$	-	0.16	-	0.5	-	0.56	-	0.56	-	0.6	-	0.63
	$\alpha = 0.05$	-	0.16	-	0.46	-	0.43	-	0.53	-	0.53	-	0.6
	$\alpha = 0.1$	-	0.13	-	0.43	-	0.4	-	0.46	-	0.5	-	0.6
	$\alpha = 0.5$	-	0.03	-	0.3	-	0.2	-	0.2	-	0.2	-	0.26
	$\alpha = 0.7$	-	0	-	0.13	-	0.16	-	0.16	-	0.16	-	0.2
Case3	$\alpha = 0.01$	-	0.16	-	0.13	-	0.5	-	0.53	-	0.4	-	0.2
	$\alpha = 0.05$	-	0.16	-	0.06	-	0.5	-	0.46	-	0.33	-	0.13
	$\alpha = 0.1$	-	0.13	-	0.06	-	0.5	-	0.46	-	0.3	-	0.13
	$\alpha = 0.5$	-	0.03	-	0.03	-	0.16	-	0.23	-	0.03	-	0.06
	$\alpha = 0.7$	-	0	-	0	-	0.13	-	0.16	-	0	-	0.03
Case4	$\alpha = 0.01$	-	0	-	0	-	0	-	0.13	-	0.6	-	0.73
	$\alpha = 0.05$	-	0	-	0	-	0	-	0.13	-	0.56	-	0.7
	$\alpha = 0.1$	-	0	-	0	-	0	-	0.1	-	0.56	-	0.63
	$\alpha = 0.5$	-	0	-	0	-	0	-	0.06	-	0.4	-	0.46
	$\alpha = 0.7$	-	0	-	0	-	0	-	0.06	-	0.33	-	0.36
Case5	$\alpha = 0.01$	-	0.3	-	0.3	-	0.3	-	0.3	-	0.53	-	0.63
	$\alpha = 0.05$	-	0.23	-	0.23	-	0.23	-	0.23	-	0.5	-	0.56
	$\alpha = 0.1$	-	0.16	-	0.16	-	0.16	-	0.16	-	0.46	-	0.53
	$\alpha = 0.5$	-	0.1	-	0.1	-	0.1	-	0.06	-	0.23	-	0.4
	$\alpha = 0.7$	-	0.06	-	0.06	-	0.06	-	0.06	-	0.13	-	0.3
Case6	$\alpha = 0.01$	-	0.23	-	0.23	-	0.23	-	0.16	-	0.5	-	0.66
	$\alpha = 0.05$	-	0.13	-	0.13	-	0.13	-	0.13	-	0.46	-	0.6
	$\alpha = 0.1$	-	0.13	-	0.13	-	0.13	-	0.1	-	0.46	-	0.56
	$\alpha = 0.5$	-	0.06	-	0.06	-	0.06	-	0.06	-	0.33	-	0.4
	$\alpha = 0.7$	-	0.06	-	0.06	-	0.06	-	0.06	-	0.23	-	0.26
Case7	$\alpha = 0.01$	-	0.6	-	0.6	-	0.6	-	0.6	-	0.7	-	0.7
	$\alpha = 0.05$	-	0.6	-	0.6	-	0.6	-	0.6	-	0.66	-	0.7
	$\alpha = 0.1$	-	0.6	-	0.6	-	0.6	-	0.6	-	0.66	-	0.7
	$\alpha = 0.5$	-	0.6	-	0.6	-	0.6	-	0.6	-	0.6	-	0.63
	$\alpha = 0.7$	-	0.6	-	0.6	-	0.6	-	0.6	-	0.6	-	0.6

Table B.6: True Positive Rate (TPR) and False Positive Rate (FPR) Graph G1 N = 5000

		P1		P2		P3		P4		P5		P6	
		TPR	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**
Case1	$\alpha = 0.01$	-	1	-	1	-	0.9	-	0.9	-	0.9	-	0.9
	$\alpha = 0.05$	-	1	-	1	-	0.8	-	0.77	-	0.77	-	0.77
	$\alpha = 0.1$	-	1	-	1	-	0.73	-	0.77	-	0.77	-	0.77
	$\alpha = 0.5$	-	1	-	1	-	0.43	-	0.43	-	0.43	-	0.43
	$\alpha = 0.7$	-	1	-	1	-	0.33	-	0.33	-	0.33	-	0.33
Case2	$\alpha = 0.01$	-	0.16	-	0.46	-	0.43	-	0.53	-	0.53	-	0.6
	$\alpha = 0.05$	-	0.1	-	0.4	-	0.36	-	0.43	-	0.46	-	0.6
	$\alpha = 0.1$	-	0.06	-	0.36	-	0.33	-	0.36	-	0.43	-	0.53
	$\alpha = 0.5$	-	0	-	0.13	-	0.16	-	0.16	-	0.2	-	0.26
	$\alpha = 0.7$	-	0	-	0.13	-	0.16	-	0.16	-	0.16	-	0.2
Case3	$\alpha = 0.01$	-	0.16	-	0.06	-	0.5	-	0.46	-	0.33	-	0.13
	$\alpha = 0.05$	-	0.1	-	0.06	-	0.46	-	0.46	-	0.26	-	0.06
	$\alpha = 0.1$	-	0.06	-	0.06	-	0.4	-	0.46	-	0.23	-	0.06
	$\alpha = 0.5$	-	0	-	0.03	-	0.13	-	0.23	-	0.03	-	0.03
	$\alpha = 0.7$	-	0	-	0	-	0.1	-	0.1	-	0	-	0
Case4	$\alpha = 0.01$	-	0	-	0	-	0	-	0.13	-	0.56	-	0.73
	$\alpha = 0.05$	-	0	-	0	-	0	-	0.1	-	0.56	-	0.63
	$\alpha = 0.1$	-	0	-	0	-	0	-	0.1	-	0.53	-	0.63
	$\alpha = 0.5$	-	0	-	0	-	0	-	0.06	-	0.33	-	0.46
	$\alpha = 0.7$	-	0	-	0	-	0	-	0.06	-	0.26	-	0.36
Case5	$\alpha = 0.01$	-	0.23	-	0.23	-	0.23	-	0.23	-	0.5	-	0.6
	$\alpha = 0.05$	-	0.13	-	0.13	-	0.13	-	0.13	-	0.36	-	0.53
	$\alpha = 0.1$	-	0.13	-	0.13	-	0.13	-	0.13	-	0.36	-	0.5
	$\alpha = 0.5$	-	0.06	-	0.06	-	0.06	-	0.06	-	0.16	-	0.3
	$\alpha = 0.7$	-	0.06	-	0.06	-	0.06	-	0.06	-	0.06	-	0.2
Case6	$\alpha = 0.01$	-	0.13	-	0.13	-	0.13	-	0.13	-	0.46	-	0.6
	$\alpha = 0.05$	-	0.13	-	0.1	-	0.1	-	0.06	-	0.46	-	0.56
	$\alpha = 0.1$	-	0.06	-	0.06	-	0.06	-	0.06	-	0.43	-	0.5
	$\alpha = 0.5$	-	0.06	-	0.06	-	0.06	-	0.06	-	0.23	-	0.4
	$\alpha = 0.7$	-	0.06	-	0.06	-	0.06	-	0.06	-	0.23	-	0.3
Case7	$\alpha = 0.01$	-	0.6	-	0.6	-	0.6	-	0.6	-	0.66	-	0.7
	$\alpha = 0.05$	-	0.6	-	0.6	-	0.6	-	0.6	-	0.63	-	0.7
	$\alpha = 0.1$	-	0.6	-	0.6	-	0.6	-	0.6	-	0.6	-	0.66
	$\alpha = 0.5$	-	0.6	-	0.6	-	0.6	-	0.6	-	0.6	-	0.6
	$\alpha = 0.7$	-	0.6	-	0.6	-	0.6	-	0.6	-	0.6	-	0.6

Table B.7: True Positive Rate (TPR) and False Positive Rate (FPR) Graph G2 N = 1000

		P1		P2		P3		P4		P5		P6	
		TPR	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**
Case1	$\alpha = 0.01$	1	1	1	1	1	0.7	1	0.7	1	0.7	1	0.7
	$\alpha = 0.05$	1	1	1	1	1	0.6	1	0.6	1	0.6	1	0.6
	$\alpha = 0.1$	1	1	1	1	1	0.6	1	0.6	1	0.6	1	0.6
	$\alpha = 0.5$	1	1	1	1	1	0.4	1	0.4	1	0.4	1	0.4
	$\alpha = 0.7$	1	1	1	1	1	0.3	1	0.3	1	0.3	1	0.3
Case2	$\alpha = 0.01$	1	0.5	1	0.5	1	0.5	1	0.45	1	0.5	1	0.5
	$\alpha = 0.05$	1	0.4	1	0.5	1	0.35	1	0.35	1	0.3	1	0.35
	$\alpha = 0.1$	1	0.4	1	0.45	1	0.25	1	0.2	1	0.15	1	0.15
	$\alpha = 0.5$	0.8	0	0.8	0.2	1	0	1	0	1	0	1	0
	$\alpha = 0.7$	0.4	0	0.5	0.15	1	0	1	0	1	0	0.6	0
Case3	$\alpha = 0.01$	1	0.5	1	0.5	0.9	0.45	0.7	0.45	1	0.5	1	0.45
	$\alpha = 0.05$	1	0.5	1	0.5	0.9	0.45	0.6	0.35	1	0.5	0.9	0.45
	$\alpha = 0.1$	1	0.5	1	0.5	0.7	0.45	0.6	0.3	1	0.5	0.9	0.45
	$\alpha = 0.5$	1	0.5	1	0.5	0.5	0.3	0.4	0.2	0.4	0.4	0.5	0.4
	$\alpha = 0.7$	1	0.5	1	0.5	0.2	0.2	0.1	0.15	0.4	0.4	0.3	0.35
Case4	$\alpha = 0.01$	0.9	0.45	0	0.3	1	0	0.9	0.4	0.9	0	0	0.5
	$\alpha = 0.05$	0.9	0.45	0	0	1	0	0.9	0.15	0.9	0	0	0.45
	$\alpha = 0.1$	0.8	0.4	0	0	1	0	0.9	0.1	0.8	0	0	0.45
	$\alpha = 0.5$	0	0	0	0	0.9	0	0.1	0	0	0	0	0.4
	$\alpha = 0.7$	0	0	0	0	0.7	0	0	0	0	0	0	0.15
Case5	$\alpha = 0.01$	0.6	0.3	0.6	0.3	0.6	0.3	0.7	0.4	0.7	0.45	0.7	0.45
	$\alpha = 0.05$	0.6	0.3	0.6	0.3	0.6	0.3	0.7	0.3	0.7	0.45	0.7	0.45
	$\alpha = 0.1$	0.6	0.3	0.6	0.3	0.6	0.3	0.7	0.3	0.6	0.45	0.7	0.45
	$\alpha = 0.5$	0.6	0.3	0.6	0.3	0.6	0.3	0.7	0.3	0.6	0.4	0.6	0.45
	$\alpha = 0.7$	0.6	0.3	0.6	0.3	0.6	0.3	0.6	0.3	0.6	0.4	0.6	0.45
Case6	$\alpha = 0.01$	0.8	0.4	1	0.5	1	0.5	1	0.45	0.9	0.4	0.9	0.15
	$\alpha = 0.05$	0.7	0.35	1	0.5	1	0.5	1	0.45	0.9	0.35	0.9	0.15
	$\alpha = 0.1$	0.7	0.35	1	0.5	1	0.5	1	0.45	0.9	0.35	0.9	0.15
	$\alpha = 0.5$	0.6	0.3	0.8	0.4	1	0.5	1	0.45	0.8	0.3	0.5	0.1
	$\alpha = 0.7$	0.4	0.2	0.8	0.4	1	0.5	1	0.45	0.3	0.2	0.4	0.1
Case7	$\alpha = 0.01$	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.8
	$\alpha = 0.05$	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75
	$\alpha = 0.1$	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75
	$\alpha = 0.5$	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.55
	$\alpha = 0.7$	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.6	0.7	0.5

Table B.8: True Positive Rate (TPR) and False Positive Rate (FPR) Graph G2 N = 3000

		P1		P2		P3		P4		P5		P6	
		TPR	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**
Case1	$\alpha = 0.01$	1	1	1	1	1	0.5	1	0.5	1	0.5	1	0.5
	$\alpha = 0.05$	1	1	1	1	1	0.4	1	0.4	1	0.4	1	0.4
	$\alpha = 0.1$	1	1	1	1	1	0.4	1	0.4	1	0.4	1	0.4
	$\alpha = 0.5$	1	1	1	1	1	0.4	1	0.4	1	0.4	1	0.4
	$\alpha = 0.7$	1	1	1	1	1	0.3	1	0.3	1	0.3	1	0.3
Case2	$\alpha = 0.01$	1	0.3	1	0.35	1	0.1	1	0.1	1	0.1	1	0.05
	$\alpha = 0.05$	1	0.1	1	0.3	1	0	1	0	1	0	1	0
	$\alpha = 0.1$	0.9	0	1	0.2	1	0	1	0	1	0	1	0
	$\alpha = 0.5$	0.6	0	0.7	0.1	0.9	0	0.9	0	0.9	0	0.9	0
	$\alpha = 0.7$	0.6	0	0.5	0.05	0.8	0	0.8	0	0.8	0	0.8	0
Case3	$\alpha = 0.01$	1	0.5	0.6	0.3	0.7	0.45	0.6	0.3	1	0.45	0.9	0.45
	$\alpha = 0.05$	1	0.5	0.6	0.3	0.7	0.35	0.5	0.25	0.9	0.45	0.9	0.45
	$\alpha = 0.1$	1	0.5	0.6	0.3	0.6	0.3	0.4	0.2	0.9	0.45	0.7	0.45
	$\alpha = 0.5$	1	0.5	0.6	0.3	0.4	0.2	0.2	0.15	0.5	0.4	0.6	0.3
	$\alpha = 0.7$	1	0.5	0.6	0.3	0.1	0.15	0.1	0.15	0.4	0.35	0.2	0.2
Case4	$\alpha = 0.01$	0.8	0	0	0	0.9	0	0.9	0.4	0.9	0	0	0.5
	$\alpha = 0.05$	0.3	0	0	0	0.9	0	0.9	0.15	0.9	0	0	0.45
	$\alpha = 0.1$	0.2	0	0	0	0.9	0	0.9	0.1	0.8	0	0	0.45
	$\alpha = 0.5$	0	0	0	0	0.3	0	0.1	0	0	0	0	0.4
	$\alpha = 0.7$	0	0	0	0	0.2	0	0	0	0	0	0	0.15
Case5	$\alpha = 0.01$	0.6	0.3	0.6	0.3	0.6	0.3	0.7	0.3	0.6	0.45	0.7	0.45
	$\alpha = 0.05$	0.6	0.3	0.6	0.3	0.6	0.3	0.7	0.3	0.6	0.45	0.7	0.45
	$\alpha = 0.1$	0.6	0.3	0.6	0.3	0.6	0.3	0.7	0.3	0.6	0.45	0.6	0.45
	$\alpha = 0.5$	0.6	0.3	0.6	0.3	0.6	0.3	0.7	0.3	0.6	0.4	0.6	0.45
	$\alpha = 0.7$	0.6	0.3	0.6	0.3	0.6	0.3	0.7	0.3	0.6	0.3	0.6	0.45
Case6	$\alpha = 0.01$	0.7	0.35	1	0.5	1	0.5	1	0.45	0.9	0.3	0.6	0.1
	$\alpha = 0.05$	0.7	0.35	0.9	0.45	1	0.5	1	0.45	0.9	0.25	0.6	0.1
	$\alpha = 0.1$	0.7	0.35	0.9	0.45	1	0.5	1	0.45	0.9	0.25	0.6	0.1
	$\alpha = 0.5$	0.6	0.3	0.8	0.4	1	0.5	1	0.4	0.5	0.15	0.3	0.1
	$\alpha = 0.7$	0.4	0.2	0.7	0.35	1	0.5	0.8	0.35	0.4	0	0.2	0.1
Case7	$\alpha = 0.01$	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75
	$\alpha = 0.05$	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75
	$\alpha = 0.1$	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75
	$\alpha = 0.5$	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.7
	$\alpha = 0.7$	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.7	0.7	0.5

Table B.9: True Positive Rate (TPR) and False Positive Rate (FPR) Graph G2 N = 5000

		P1		P2		P3		P4		P5		P6	
		TPR	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**
Case1	$\alpha = 0.01$	1	0.95	1	0.95	1	0.4	1	0.4	1	0.4	1	0.4
	$\alpha = 0.05$	1	0.95	1	0.95	1	0.4	1	0.4	1	0.4	1	0.4
	$\alpha = 0.1$	1	0.95	1	0.95	1	0.4	1	0.4	1	0.4	1	0.4
	$\alpha = 0.5$	1	0.95	1	0.95	1	0.35	1	0.35	1	0.35	1	0.35
	$\alpha = 0.7$	1	0.95	1	0.95	1	0.2	1	0.25	1	0.25	1	0.25
Case2	$\alpha = 0.01$	1	0.1	1	0.3	1	0	1	0	1	0	1	0
	$\alpha = 0.05$	0.9	0	1	0.2	1	0	1	0	1	0	1	0
	$\alpha = 0.1$	0.9	0	1	0.15	1	0	1	0	1	0	1	0
	$\alpha = 0.5$	0.6	0	0.6	0.1	0.9	0	0.9	0	0.0	0	0.9	0
	$\alpha = 0.7$	0.4	0	0.4	0	0.7	0	0.8	0	0.8	0	0.9	0
Case3	$\alpha = 0.01$	0.9	0.45	0.5	0.25	0.7	0.35	0.5	0.25	1	0.45	0.9	0.45
	$\alpha = 0.05$	0.9	0.45	0.5	0.25	0.6	0.3	0.4	0.2	0.9	0.45	0.7	0.45
	$\alpha = 0.1$	0.9	0.45	0.5	0.25	0.6	0.3	0.4	0.2	0.9	0.45	0.7	0.4
	$\alpha = 0.5$	0.9	0.45	0.5	0.25	0.3	0.2	0.1	0.15	0.5	0.35	0.4	0.25
	$\alpha = 0.7$	0.9	0.45	0.5	0.25	0.1	0.15	0	0.1	0.2	0.2	0.1	0.2
Case4	$\alpha = 0.01$	0.3	0.15	0	0	0.9	0	0.9	0.4	0.9	0	0	0.5
	$\alpha = 0.05$	0.1	0.05	0	0	0.9	0	0.9	0.15	0.9	0	0	0.45
	$\alpha = 0.1$	0	0	0	0	0.8	0	0.9	0.1	0.8	0	0	0.45
	$\alpha = 0.5$	0	0	0	0	0.3	0	0.1	0	0	0	0	0.4
	$\alpha = 0.7$	0	0	0	0	0	0	0	0	0	0	0	0.15
Case5	$\alpha = 0.01$	0.6	0.3	0.6	0.3	0.6	0.3	0.7	0.3	0.6	0.45	0.7	0.45
	$\alpha = 0.05$	0.6	0.3	0.6	0.3	0.6	0.3	0.7	0.3	0.6	0.45	0.6	0.45
	$\alpha = 0.1$	0.6	0.3	0.6	0.3	0.6	0.3	0.7	0.3	0.6	0.4	0.6	0.45
	$\alpha = 0.5$	0.6	0.3	0.6	0.3	0.6	0.3	0.6	0.3	0.6	0.3	0.6	0.45
	$\alpha = 0.7$	0.6	0.3	0.6	0.3	0.6	0.3	0.6	0.3	0.6	0.3	0.6	0.45
Case6	$\alpha = 0.01$	0.7	0.35	1	0.5	1	0.5	1	0.45	0.9	0.2	0.6	0.1
	$\alpha = 0.05$	0.7	0.35	1	0.5	1	0.5	1	0.45	0.9	0.2	0.6	0.1
	$\alpha = 0.1$	0.6	0.3	0.8	0.4	1	0.5	1	0.45	0.9	0.2	0.3	0.1
	$\alpha = 0.5$	0.4	0.2	0.8	0.4	1	0.5	1	0.35	0.5	0.15	0.2	0.1
	$\alpha = 0.7$	0.3	0.15	0.7	0.35	1	0.5	0.7	0.35	0.2	0.1	0.2	0.1
Case7	$\alpha = 0.01$	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75
	$\alpha = 0.05$	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75
	$\alpha = 0.1$	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75
	$\alpha = 0.5$	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75
	$\alpha = 0.7$	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.75	0.7	0.65

Table B.10: True Positive Rate (TPR) and False Positive Rate (FPR) Graph G3 N = 1000

		P1		P2		P3		P4		P5		P6	
		TPR	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**
Case1	$\alpha = 0.01$	1	1	1	1	1	0.8	1	0.75	1	0.75	1	0.75
	$\alpha = 0.05$	1	1	1	1	1	0.65	1	0.6	1	0.6	1	0.6
	$\alpha = 0.1$	1	1	1	1	1	0.55	1	0.5	1	0.5	1	0.5
	$\alpha = 0.5$	1	1	1	1	1	0.15	1	0.15	1	0.15	1	0.15
	$\alpha = 0.7$	1	1	1	1	1	0	1	0.05	1	0.05	1	0.05
Case2	$\alpha = 0.01$	1	0.35	1	0.3	1	0.25	1	0.25	1	0.25	1	0.2
	$\alpha = 0.05$	1	0.2	1	0.2	1	0.2	1	0.15	1	0.1	1	0.05
	$\alpha = 0.1$	1	0.2	0.9	0.2	1	0.1	1	0.1	1	0.05	1	0.05
	$\alpha = 0.5$	0.3	0.15	0.5	0.1	0.8	0	0.8	0	0.8	0	0.9	0
	$\alpha = 0.7$	0.1	0.15	0.4	0.1	0.7	0	0.7	0	0.7	0	0.5	0
Case3	$\alpha = 0.01$	0.6	0.15	0.5	0.1	0.9	0	0.8	0.25	0.8	0.25	0.9	0.05
	$\alpha = 0.05$	0.4	0.15	0.5	0.1	0.9	0	0.8	0.25	0.8	0.25	0.8	0.05
	$\alpha = 0.1$	0.4	0.1	0.5	0.1	0.9	0	0.8	0.25	0.8	0.25	0.8	0.05
	$\alpha = 0.5$	0.2	0.05	0.4	0.05	0.8	0	0.6	0.2	0.6	0.2	0.7	0
	$\alpha = 0.7$	0.2	0.05	0.3	0.05	0.8	0	0.4	0.15	0.4	0.15	0.6	0
Case4	$\alpha = 0.01$	0	0	0.9	0.25	0.8	0.1	0.8	0.2	0.9	0.25	1	0.35
	$\alpha = 0.05$	0	0	0.8	0.15	0.8	0.05	0.8	0.05	0.8	0.15	0.9	0.25
	$\alpha = 0.1$	0	0	0.8	0.05	0.8	0	0.8	0	0.8	0.05	0.9	0.2
	$\alpha = 0.5$	0	0	0.6	0	0.2	0	0.4	0	0.6	0	0.7	0
	$\alpha = 0.7$	0	0	0.2	0	0	0	0.1	0	0.2	0	0.5	0
Case5	$\alpha = 0.01$	0	0.7	0	0.5	0.1	0.5	0.2	0.65	0.4	0.9	1	1
	$\alpha = 0.05$	0	0.65	0	0.5	0.1	0.5	0.2	0.6	0.4	0.8	1	1
	$\alpha = 0.1$	0	0.55	0	0.5	0.1	0.5	0.2	0.6	0.4	0.8	1	1
	$\alpha = 0.5$	0	0.5	0	0.5	0.1	0.5	0.2	0.5	0.4	0.8	1	1
	$\alpha = 0.7$	0	0.5	0	0.5	0.1	0.5	0.2	0.5	0.4	0.8	1	1
Case6	$\alpha = 0.01$	0.1	0	0.7	0.05	0.9	0.05	0.9	0.2	0.9	0.4	0.9	0.4
	$\alpha = 0.05$	0	0	0.4	0.05	0.6	0	0.9	0.2	0.9	0.4	0.9	0.4
	$\alpha = 0.1$	0	0	0.2	0.05	0.6	0	0.8	0.05	0.9	0.3	0.9	0.4
	$\alpha = 0.5$	0	0	0	0	0	0	0.2	0	0.6	0	0.8	0.2
	$\alpha = 0.7$	0	0	0	0	0	0	0	0	0.2	0	0.6	0
Case7	$\alpha = 0.01$	1	0.5	1	0.5	1	0.5	1	0.5	1	0.55	1	0.6
	$\alpha = 0.05$	1	0.35	1	0.35	1	0.35	1	0.35	1	0.35	1	0.4
	$\alpha = 0.1$	1	0.2	1	0.25	1	0.3	1	0.25	1	0.25	1	0.3
	$\alpha = 0.5$	0.8	0.15	0.5	0.2	1	0.1	1	0.1	0.9	0.1	0.8	0.1
	$\alpha = 0.7$	0.3	0	0.4	0.05	0.8	0	1	0	0.7	0	0.1	0

Table B.11: True Positive Rate (TPR) and False Positive Rate (FPR) Graph G3 N = 3000

		P1		P2		P3		P4		P5		P6	
		TPR	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**
Case1	$\alpha = 0.01$	1	1	1	1	1	0.5	1	0.4	1	0.4	1	0.4
	$\alpha = 0.05$	1	1	1	1	1	0.3	1	0.3	1	0.3	1	0.3
	$\alpha = 0.1$	1	1	1	1	1	0.3	1	0.25	1	0.25	1	0.25
	$\alpha = 0.5$	1	1	1	1	1	0.05	1	0.05	1	0.05	1	0.05
	$\alpha = 0.7$	1	1	1	1	1	0	1	0	1	0	1	0
Case2	$\alpha = 0.01$	0.9	0.2	0.9	0.15	1	0.1	1	0.05	1	0.05	1	0.05
	$\alpha = 0.05$	0.6	0.2	0.6	0.15	1	0	1	0	1	0	1	0
	$\alpha = 0.1$	0.6	0.2	0.6	0.1	1	0	1	0	1	0	1	0
	$\alpha = 0.5$	0.4	0.1	0.4	0.1	0.7	0	0.7	0	0.7	0	0.8	0
	$\alpha = 0.7$	0.2	0.1	0.4	0.1	0.6	0	0.7	0	0.7	0	0.7	0
Case3	$\alpha = 0.01$	0.6	0.15	0.5	0.1	0.9	0	0.8	0.25	0.8	0.25	0.8	0.05
	$\alpha = 0.05$	0.6	0.15	0.5	0.05	0.9	0	0.7	0.25	0.7	0.25	0.8	0
	$\alpha = 0.1$	0.6	0.15	0.5	0.05	0.9	0	0.6	0.25	0.6	0.25	0.7	0
	$\alpha = 0.5$	0.5	0.1	0.5	0.05	0.5	0	0.5	0.15	0.5	0.15	0.6	0
	$\alpha = 0.7$	0.4	0.05	0.4	0.05	0.5	0	0.4	0.15	0.4	0.15	0.6	0
Case4	$\alpha = 0.01$	0	0	0.8	0	0.6	0	0.8	0	0.8	0	0.9	0.2
	$\alpha = 0.05$	0	0	0.7	0	0.5	0	0.6	0	0.7	0	0.8	0
	$\alpha = 0.1$	0	0	0.6	0	0.5	0	0.5	0	0.6	0	0.8	0
	$\alpha = 0.5$	0	0	0	0	0	0	0	0	0	0	0.4	0
	$\alpha = 0.7$	0	0	0	0	0	0	0	0	0	0	0	0
Case5	$\alpha = 0.01$	0	0.55	0	0.5	0	0.5	0.2	0.6	0.4	0.8	0.9	1
	$\alpha = 0.05$	0	0.5	0	0.5	0	0.5	0.2	0.6	0.4	0.8	0.9	1
	$\alpha = 0.1$	0	0.5	0	0.5	0	0.5	0.2	0.6	0.4	0.8	0.9	1
	$\alpha = 0.5$	0	0.5	0	0.5	0	0.5	0.2	0.55	0.4	0.8	0.9	1
	$\alpha = 0.7$	0	0.5	0	0.5	0	0.5	0.2	0.55	0.4	0.8	0.9	1
Case6	$\alpha = 0.01$	0	0	0.2	0.05	0.6	0	0.8	0.05	0.9	0.2	0.9	0.4
	$\alpha = 0.05$	0	0	0.1	0	0.3	0	0.6	0	0.8	0.2	0.9	0.4
	$\alpha = 0.1$	0	0	0.1	0	0.1	0	0.6	0	0.8	0.05	0.9	0.3
	$\alpha = 0.5$	0	0	0	0	0	0	0	0	0.3	0	0.6	0
	$\alpha = 0.7$	0	0	0	0	0	0	0	0	0	0	0.2	0
Case7	$\alpha = 0.01$	1	0.2	1	0.2	1	0.25	1	0.25	1	0.2	1	0.25
	$\alpha = 0.05$	1	0.2	1	0.2	1	0.15	1	0.15	1	0.15	0.9	0.15
	$\alpha = 0.1$	1	0.15	0.8	0.2	1	0.1	1	0.1	0.9	0	0.9	0
	$\alpha = 0.5$	0.5	0	0.4	0	0.8	0	0.9	0	0.9	0	0.9	0
	$\alpha = 0.7$	0.3	0	0.3	0	0.6	0	0.9	0	0.9	0	0.8	0

Table B.12: True Positive Rate (TPR) and False Positive Rate (FPR) Graph G3 N = 5000

		P1		P2		P3		P4		P5		P6	
		TPR	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**
Case1	$\alpha = 0.01$	1	1	1	1	1	0.3	1	0.3	1	0.3	1	0.3
	$\alpha = 0.05$	1	1	1	1	1	0.25	1	0.2	1	0.2	1	0.2
	$\alpha = 0.1$	1	1	1	1	1	0.1	1	0.1	1	0.15	1	0.15
	$\alpha = 0.5$	1	1	1	1	1	0	1	0.05	1	0.05	1	0.05
	$\alpha = 0.7$	1	1	1	1	1	0	1	0	1	0	1	0
Case2	$\alpha = 0.01$	0.6	0.2	0.6	0.15	1	0	1	0	1	0	1	0
	$\alpha = 0.05$	0.5	0.2	0.6	0.1	1	0	1	0	1	0	1	0
	$\alpha = 0.1$	0.4	0.15	0.4	0.1	0.9	0	0.8	0	0.8	0	0.8	0
	$\alpha = 0.5$	0.4	0.1	0.4	0.1	0.6	0	0.7	0	0.7	0	0.7	0
	$\alpha = 0.7$	0.4	0.1	0.3	0.1	0.5	0	0.5	0	0.5	0	0.5	0
Case3	$\alpha = 0.01$	0.6	0.15	0.5	0.05	0.9	0	0.6	0.25	0.6	0.25	0.8	0
	$\alpha = 0.05$	0.6	0.15	0.5	0.05	0.9	0	0.6	0.25	0.6	0.25	0.7	0
	$\alpha = 0.1$	0.6	0.15	0.5	0.05	0.7	0	0.6	0.2	0.6	0.2	0.7	0
	$\alpha = 0.5$	0.5	0.15	0.4	0.05	0.5	0	0.3	0.15	0.3	0.15	0.6	0
	$\alpha = 0.7$	0.4	0.1	0.4	0.05	0.5	0	0.2	0.1	0.2	0.1	0.6	0
Case4	$\alpha = 0.01$	0	0	0.7	0	0.5	0	0.6	0	0.7	0	0.8	0.05
	$\alpha = 0.05$	0	0	0.5	0	0.4	0	0.5	0	0.5	0	0.8	0
	$\alpha = 0.1$	0	0	0.5	0	0.1	0	0.4	0	0.5	0	0.6	0
	$\alpha = 0.5$	0	0	0	0	0	0	0	0	0	0	0	0
	$\alpha = 0.7$	0	0	0	0	0	0	0	0	0	0	0	0
Case5	$\alpha = 0.01$	0	0.55	0	0.5	0	0.5	0.1	0.6	0.4	0.85	0.9	1
	$\alpha = 0.05$	0	0.5	0	0.5	0	0.5	0.1	0.55	0.4	0.85	0.9	1
	$\alpha = 0.1$	0	0.5	0	0.5	0	0.5	0.1	0.55	0.4	0.8	0.9	1
	$\alpha = 0.5$	0	0.5	0	0.5	0	0.5	0.1	0.55	0.4	0.8	0.9	1
	$\alpha = 0.7$	0	0.5	0	0.5	0	0.5	0.1	0.55	0.4	0.8	0.9	1
Case6	$\alpha = 0.01$	0	0	0.1	0	0.3	0	0.6	0	0.8	0.2	0.9	0.4
	$\alpha = 0.05$	0	0	0	0	0.1	0	0.6	0	0.8	0.05	0.9	0.25
	$\alpha = 0.1$	0	0	0	0	0	0	0.3	0	0.8	0	0.9	0.2
	$\alpha = 0.5$	0	0	0	0	0	0	0	0	0.1	0	0.4	0
	$\alpha = 0.7$	0	0	0	0	0	0	0	0	0	0	0.1	0
Case7	$\alpha = 0.01$	1	0.2	1	0.2	1	0.15	1	0.15	1	0.15	1	0.2
	$\alpha = 0.05$	1	0.15	0.8	0.2	1	0	1	0	1	0	0.9	0
	$\alpha = 0.1$	0.9	0.1	0.8	0.1	1	0	1	0	1	0	0.9	0
	$\alpha = 0.5$	0.4	0	0.4	0	0.8	0	0.9	0	0.9	0	0.9	0
	$\alpha = 0.7$	0.3	0	0.4	0	0.6	0	0.8	0	0.7	0	0.8	0

Abbreviations

BES BES Equivalence Search

BMI Body Mass Index

CFD Cumulative Distribution Function

CPD Conditional Probability Distribution

CPDAG Completed Partially Directed Acyclic Graph

CLC Conditional Linear Gaussian

DAG Direted Acyclic Graph

FCI Fast Causal Inference

FES Forward Equivalence Search

FNR False Negative Rate

FPR False Positive Rate

GES Greedy Equivalence Search

IID Independent and Identically Distributed

InterVal Intersection Validation

KAFCI Kernel Alignment Fast Causal Inference

KAPC Kernel Alignment PC Algorithm

KGV Kernel Generalized Variance

KMI Kernel Mutual Information

MAG Maximal Ancestral Graph

MAR Missing At Random

MCAR Missing Completely At Random

MLE Maximum Likelihood Estimation

MMCI Max-Min Hill Climbing

MNAR Missing Not At Random

MVEE Modal Value of Edges Existence

MXM Mens eX Machina

PAG Partial Ancestral Graph

PDAG Partially Directed Acyclic Graph

PDF Probability Density Function

PHD Partial Hamming Distance

PSE Partial Skeleton Error

RBF Radial Basis Function

ROC Receiver Operating Characteristic

RFCI Really Fast Causal Inference

SHD Structural Hamming Distance

TNR True Negative Rate

TPR True Positive Rate

References

[1] R.A. Ali, T. S. Richardson, and P. Spirtes. Markov equivalence for ancestral graphs. *The Annals of Statistics*, 37(5B):2808–2837, 2009.

[2] P.D. Allison. *Missing Data*. Sage Publications, Inc, 2001.

[3] B. Andrews, J. Ramsey, and G. F. Cooper. Scoring Bayesian networks of mixed variables. *International Journal of Data Science and Analytics*, 6:3–18, 2018.

[4] B. Andrews, J. Ramsey, and G. F. Cooper. Learning High Dimensional Directed Acyclic Graphs with Mixed Data Types. In *Proceedings of Machine Learning Research*, pages 4–21, 2019.

[5] V. Audigier, F. Husson, and J. Josse. A principal components method to impute missing values for mixed data. *Advances in Data Analysis and Classification*, 10:5 – 26, 2013.

[6] F. R. Bach and M. I. Jordan. Kernel independent component analysis. *Journal of Machine Learning Research*, 3:1 – 48, 2002.

[7] F. R. Bach and M. I. Jordan. Learning Graphical Models with Mercer Kernels. In *Proceedings of the 15th International Conference on Neural Information Processing Systems*, pages 1033–1040, 2002.

[8] L. A. Belanche and M. A. Villegas. Kernel functions for categorical variables with application to problems in the life sciences. In *The 16 International Conference of the Catalan Association of Artificial Intelligence*, pages 171–180, 2013.

[9] D.A. Bennet. How can I deal with missing data in my study? *Aust N Z J Public Health*, 25(5):464–469, 2001.

[10] M. Blackwell. A framework for dynamic causal inference in political science. *American Journal of Political Science*, 57(2):504–519, 2013.

[11] S. G. Boettcher and C. Dethlefsen. deal: A package for learning bayesian networks. *Journal of Statistical Software*, 8(20):1–40, 2003.

[12] S. G. Bottcher and C. Dethlefsen. deal: Learning Bayesian networks with mixed variables.

[13] K. H. Brodersen, F. Gallusser, J. Koehler, N. Remy, and S. L. Scott. Inferring causal impact using Bayesian structural time-series models. *The Annals of Applied Statistics*, 9(1):247–274, 2015.

[14] S. Buuren, K. G. Oudshoorn, G. Vink, R. Schouten, A. Robitzsch, L. Doove, S. Jolani, M. Moreno-Betancur, I. White, P. Gaffert, F. Meinfelder, B. Gray, and V. A. Bundock. mice: Multivariate imputation by chained equations.

[15] S. V. Buuren and K.Groothuis-Oudshoorn. mice: Multivariate Imputation by Chained Equations in R. *Journal of Statistical Software*, 45(3):1–67, 2011.

[16] D. M. Chickering. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3:507–554, 2002.

[17] D. M. Chickering and C. Meek. Selective Greedy Equivalence Search: Finding optimal Bayesian Networks using a polynomial number of score evaluations. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, pages 211–219, Arlington, Virginia, United States, 2015. AUAI Press.

[18] T. Claassen, J.M Mooij, and T. Heskes. Learning sparse causal models is not NP-hard. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence*, pages 172–181, 2013.

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

[20] D. Colombo, M. H. Maathuis, M. Kalisch, and T. S. Richardson. Learning High-Dimensional Directed Acyclic Graphs with Latent and Selection Variables. *The Annals of Statistics*, 40:294–321, 2012.

[21] C. Cortes, M. Mohri, and A. Rostamizadeh. Algorithms for learning kernels based on centered alignment. *Journal of Machine Learning Research*, 13:795–828, 2012.

[22] N. Cristianini, J. Kandola, A. Elisseeff, and J. Shawe-Taylor. On kernel-target alignment. In D. E. Holmes and L. C. Jain, editors, *Innovations in Machine Learning. Studies in Fuzziness and Soft Computing*, volume 194, pages 205–256, Berlin, Heidelberg, 2006. Springer.

[23] N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. Kandola. On kernel-target alignment. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Neural Information Processing Systems 14 (NIPS 2001)*, pages 367–373. MIT Press, 2001.

[24] R. Cui, P. Groot, and T. Heskes. Copula PC algorithm for causal discovery from mixed data. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 377–392, Riva del Garda, 2016. Springer.

[25] R. Cui, P. Groot, and T. Heskes. Robust estimation of gaussian copula causal structure from mixed data with missing values. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 835–840. IEEE, 2017.

[26] R. Cui, P. Groot, and T. Heskes. Learning causal structure from mixed data with missing values using Gaussian copula models. *Statistics and Computing*, 29:311–333, 2019.

[27] Y. Dong and C-Y. J. Peng. Principled missing data methods for researchers. *Springerplus*, 2:1–17, 2013.

[28] C.K. Enders. Using the expectation maximization algorithm to estimate coefficient alpha for scales with item-level missing data. *Psychological Methods*, 8:322–337, 2003.

[29] F. Fukumizu, A. Gretton, X. Sun, and B. H. Schölkopf. Kernel measures of conditional dependence. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, pages 489–496, 2007.

[30] P. I. Good. *Introduction to Statistics Through Resampling Methods and R.* John Wiley and Sons, New Jersey, 2013.

[31] J. W. Graham. *Missing Data: Analysis and Design.* Springer, 2012.

[32] T. Górecki, M. Krzysko, and W. Wołynski. Independence test and canonical correlation analysis based on the alignment between kernel matrices for multivariate functional data. *Artificial Intelligence Review*, 53:475–499, 2020.

[33] A. Gretton, O. Bousquet, A. Smola, and B. Scholkopf. Measuring Statistical Dependence with Hilbert-Schmidt Norms. Technical report, UMax Planck Institute for Biological Cybernetics, 05 2005.

[34] A. Gretton, R. Herbrich, A. Smola, O. Bousquet, and B. Schölkopf. Kernel methods for measuring independence. *Journal of Machine Learning Research*, 6:2075–2129, 2005.

[35] T. Handhayani and J. Cussens. Kernel-based approach for learning causal graphs from mixed data. In *Proceedings of The 10th International Conference on Probabilistic Graphical Models*, pages 1–12, 2020.

[36] S. M. Hill, L. M. Heiser, T. Cokelaer, M. Unger, N. K. Nesser, D. Carlin, Y. Zhang, A. Sokolov, E. O. Paull, C. K. Wong, K. Graim, A. Bivol, H. Wang, F. Zhu, B. Afsari, L. V. Danilova, A. V. Favorov, W. S. Lee, HPN-DREAM Consortium, G. B. Mils, J. W. Gray, M. Kellen, T. Norman, S. Friend, M. A. Qutub, E. J. Fertig, Y. Guan, M. Song, J. M. Stuart, P. T. Spellman, H. Koeppl, G. Stolovitzky, J. Saez-Rodriguez, and S. Mukherjee. Inferring causal molecular networks: empirical assessment through a community-based effort. *Nature Methods*, 13(4):310–318, 2016.

[37] F. Husson and J. Josse. missMDA: Handling missing values with multivariate data analysis.

[38] J. L. Kaiser, C. Bland, and D. J. Klinke. Identifying causal networks linking cancer processes and anti-tumor immunity using Bayesian network inference and metagene constructs. *Biotechnol Prog*, 32(2):470–479, 2016.

[39] M. Kalisch and P. Buhlmann. Estimating high-dimensional directed acyclic graphs with the PC algorithm. *Journal of Machine Learning Research*, 8:613–636, 2007.

[40] M. Kalisch, M. Machler, D. Colombo, M. H. Maathuis, and P. Buhlmann. Causal Inference using Graphical Models with the R Package pcalg. *Journal of Statistical Software*, 47:1–26, 2012.

[41] H. Kang. The prevention and handling of the missing data. *Korean J Anesthesiol*, 64(5):402–406, 2013.

[42] D. Koller and N. Friedman. *Probabilistic Graphical Models Principles and Techniques*. The MIT Press, 2009.

[43] V. Lagani, G. Athineou, A. Farcomeni, M. Tsagris, and I. Tsamardinos. Feature selection with the r package MXM: Discovering statistically equivalent feature subsets. *Journal of Statistical Software*, 80(7):1–25, 2017.

[44] R. J. A. Little and D. B. Rubin. *Statistical Analysis with Missing Data, 2nd edition*. John Wiley & Sons, Inc., 2002.

[45] P. Madley-Dowd, R. Hughes, K. Tilling, and J. Heron. The proportion of missing data should not be used to guide decisions on multiple imputation. *Journal of Clinical Epidemiology*, 110:63–73, 2019.

[46] B. Malone, M. Järvisalo, and P. Myllymäki. Impact of Learning Strategies on The Quality of Bayesian Networks: an Empirical Evaluation. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, pages 562–571, Amsterdam, Netherlands, 2015. AUAI Press.

[47] M. Meila. Data centering in feature space. In *AISTATS*, 2003.

[48] M. Meloun and J. Militký. *Statistical Data Analysis A Practical Guide*. India PVT. LTD., 2011.

[49] B. Mirkin. *Core Concept in Data Analysis: Summarization, Correlation, and Visualization*. Springer, London, 2011.

[50] M.Scutari and J-B. Denis. *Bayesian Networks With Examples in R*. CRC Press, 2015.

[51] R. Nagarajan, M. Scutari, and S. Lebre. *Bayesian Network in R with Applications in Systems Biology*. Springer, 2013.

[52] G. Nebot-Troyano and L.A. Belanche-Munoz. A kernel extension to handle missing data. In M. Bramer, R. Ellis, and M. Petridis, editors, *The Twenty-ninth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 165–178. Springer, 2009.

[53] N.Harris and M. Drton. PC algorithm for nonparanormal graphical models. *Journal of Machine Learning Research*, 14:3365–3383, 2013.

[54] J. Pearl. An introduction to causal inference. *The International Journal of Biostatistics*, 6(2):1–59, 2010.

[55] C. H. Peng, Y. Z. Jiang, A. S. Tai, C. B. Liu, S. C. Peng, C. T. Liao, T. C. Yen, and W. P. Hsieh. Causal inference of gene regulation with subnetwork assembly from genetical genomics data. *Nucleic Acids Research*, 42(5):2803–2819, 2014.

[56] J. Peters, D. Janzing, and B. Schölkopf. *Elements of causal Inference*. The MIT Press, 2017.

[57] J. Poulos and R. Valle. Missing data imputation for supervised learning. *Applied Artificial Intelligence*, 32:186–196, 2018.

[58] V. K. Raghu, J. D. Ramsey, A. Morris, D. V. Manatakis, P. Sprites, P. K. Chrysanthis, C. Glymour, and P. V. Benos. Comparison of strategies for scalable causal discovery of latent variable models from mixed data. *International Journal of Data Science and Analytics*, 6:33–45, 2018.

[59] M. Scutari, C.E. Graafland, and J.M Gutiérrez. Who learns better Bayesian network structures: Accuracy and speed of structure learning algorithms. *International Journal of Approximate Reasoning*, 115:235–253, 2019.

[60] M. Scutari and R. Ness. bnlearn: Bayesian Network structure learning, parameter learning and inference.

[61] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004.

[62] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search.* The MI Press, New York, 2001.

[63] D. J. Stekhoven and P. Bühlmann. Missforest—non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28:112–118, 2012.

[64] E. V. Strobl, S. Visweswaran, and P. L. Spirtes. Fast causal inference with non-random missingness by test-wise deletion. *International Journal of Data Science and Analytics*, 6:47–62, 2018.

[65] X. Sun, D. Janzing, B. H. Scholkopf, and K. Fukumizu. A kernel-based causal learning algorithm. In *The 24th international conference on Machine learning*, pages 855–862, 2007.

[66] G. J. Székely, M. L. Rizzo, and N. K. Bakirov. Measuring and testing dependence by correlation of distances. *The Annals of Statistics*, 35(6):2769–2794, 2007.

[67] R. E. Tillman, A. Gretton, and P. Spirtes. Nonlinear directed acyclic structure learning with weakly additive noise models. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, pages 1847–1855, 2009.

[68] S. Triantafillou, V. Lagani, C. Heinze-Deml, A. Schmidt, J. Tegner, and I. Tsamardinos. Predicting causal relationships from biological data: Applying automated causal discovery on mass cytometry data of human immune cells. *Nature*, 2017.

[69] S. Triantafillou and I. Tsamardinos. Score based vs constraint based causal learning in the presence of confounders. In *Conference on Uncertainty in Artificial Intelligence*, pages 59–67, New York City, 2016.

[70] M. Tsagris, G. Borboudakis, V. Lagani, and I. Tsamardinos. Constraint-based causal discovery with mixed data. *International Journal of Data Science and Analytics*, 6(1):19–30, 2018.

[71] M. Tsagris, I. Tsamardinos, V. Lagani, G. Athineou, G. Borboudakis, and A. Roumpelaki. MXM: Feature selection (including multiple solutions) and Bayesian networks.

[72] I. Tsamardinos, L. E. Brown, and C. F. Aliferis. The Max-Min Hill-Climbing Bayesian Network Structure Learning Algorithm. *Machine Learning*, pages 31–78, 2006.

[73] H. R. Varian. Causal inference in economics and marketing. *PNAS*, 113(27):7310–7315, 2016.

[74] J. Viinikka, R. Eggeling, and M. Koivisto. Intersection-validation: A Method for Evaluating Structure Learning without Ground Truth. In *The 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1–5, Lanzarote, Spain, 2018. PMLR.

[75] R. E. Walpole, R. H. Myers, S. L. Myers, and K. Ye. *Probability & statistics for engineers & scientists*. Prentice Hall, 2011.

[76] D. Westreich, J. K. Edwards, S. R. Cole, R. W. Platt, S. L. Mumford, and E. F. Schisterman. Imputation approaches for potential outcomes in causal inference. *International Journal of Epidemiology*, pages 1731–1737, 2015.

[77] J. Yang, Z. Li, X. Fan, and Y. Cheng. Drug disease association and drug repositioning predictions in complex diseases using causal inference probabilistic matrix factorization. *Journal of Chemical Information and Modeling*, 54(9):2562–2569, 2014.

[78] J. Zhang. On the Completeness of Orientation Rules for Causal Discovery in the Presence of Latent Confounders and Selection Bias. *Artificial Intelligence*, 172:1873–1896, 2008.

[79] Q. Zhang, S. Filippi, A. Gretton, and D. Sejdinovic. Large-scale kernel methods for independence testing. *Statistics and Computing*, 28:113–130, 2018.

[80] S. Zhang, X. Wu, and M. Zhu. Efficient missing data imputation for supervised learning. In *Proceeding of The 9th IEEE International Conference on Cognitive Informatics (ICCI'10)*, pages 672–679, Beijing, 2010.

[81] X. Zhang, Y. Hu, K. Xie, S. Wang, E. W. T. Ngai, and M. Liu. A causal feature selection algorithm for stock prediction modeling. *Neurocomputing*, 142:48–59, 2014.