Online Causal Structure Learning in the Presence of Latent Variables

Durdane Kocaçoban

Doctor of Philosophy

University of York

Computer Science

October 2019

Abstract

In this thesis, we propose to use Causal Models, which play a central role in dealing with uncertainty in Artificial Intelligence (AI). Causal models can be created based on information, data, or both. Regardless of the source of information used to create the model, there may be inaccuracies, or the application area may vary. Therefore, the model needs constant improvement during use. Most of existing learning algorithms are batch. However, industrial companies store vast amounts of data every day in real-world. Existing batch methods cannot process the significant quantity of continuously incoming data in a reasonable amount of time and memory. Therefore, batch methods may become computationally expensive and infeasible for large dataset. In this way, we present three online causal structure learning algorithms to fill this gap. These algorithms can track changes in a causal structure and process data in a dynamic real-time manner. Standard causal structure learning algorithms assume that causal structure does not change during the data collection process, but in real-world scenarios, it does often change. The online causal structure learning algorithms we present here can revise correlation values without reprocessing the entire dataset and use an existing model to avoid re-learning the causal links in the prior model, which still fit data. The algorithms update the correlations of causes and effects with the weight estimation of each causal interaction. Proposed algorithms are tested on synthetic and real-world datasets. The online causal structure learning algorithms outperformed a well known batch algorithm (FCI) by a large margin in learning the changed causal structure correctly and efficiently when latent variables were present.

Contents

List of Tables

List of Tables

List of Figures

List of Figures

Acknowledgements

I am very thankful to the Ministry of National Education (Turkey) for providing funding for my studies.

On a more personal level, first of all, I would like to thank my supervisor, James Cussens and my assessor, Daniel Kudenko. Thank you for your knowledge, your time, and your resolve and ideas. I could not have done this study without them. I would like to thank AIG group members for discussions and hours spent ideas and valuable seminars. I would like to thank YCCSA members who are always being friendliness and shared the same floor with me for four years. Thank you to Burak Merdenyan, Elmira Esmaeili, Frances Drachenberg, Martin James Rusilowicz, Alan Millard, Teny Handhayani, Nunung Nurul Qomariyah and Lisa Sha Li, for the many teas, discussions, friendliness and your support. Thank you especially, to my fiance, Ibrahim, and my family; brothers, Goksel and Isa, sisters, Sultan and Buket and Father and Mother Mehmet and Emir Ayse, for their continued love and support. And finally, a big thank you to all for being in my life.

List of Figures

Declaration

I, Durdane Kocacoban, declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as references.

Some of the material contained in this thesis has appeared in the following published papers.

List of Publications

- Kocacoban, D. and Cussens, J. (2019). Online Causal Structure Learning in the Presence of Latent Variables. In: IEEE ICMLA 2019. Boca Raton, Florida, USA: IEEE.

- Kocacoban, D. and Cussens, J. (2019). Fast Online Learning in the Presence of Latent Variables. In: ISAAI'19. Munich, Germany: Springer Publishing Company.

List of Figures

Chapter 1

INTRODUCTION

1.1 Motivation and Problem statement

In real world scenarios, industrial companies store vast amounts of data in every day. Existing batch methods cannot process the significant quantity of continuously incoming data in a reasonable amount of time and memory. Batch structure learning algorithms may become computationally expensive and infeasible for large dataset. So, we need new methods which are able to learn a structure as online.

In this study, we aim to fill this gap via a machine learning approach. In this machine learning approach, Algorithms can do structure learning as online from a new data coming sequentially from a domain.

Bayesian Networks (BNs) are a type of a probabilistic graphical model that can be viewed as a Directed Acyclic Graph (DAG), where nodes represent uncertain variables and arcs represent dependency or causal relationship between variables. The structure of a BN can be learned from data, and there are three main classes of structure learning: constraint-based, score-based and hybrid learning. The first type relies on conditional independence tests to construct the skeleton and orient edges, whereas the second type searches over the space of possible graphs

and returns the graph that maximises a fitting score. Hybrid learning refers to algorithms that combine both constraint-based and score-based learning.

Bayesian Networks have a simple and easily understandable representation of data in comparison with other representation methods. Bayesian Networks have a simple and easily understandable representation of data in comparison with other representation methods. They are robust and consistency in representing and handling with relevant probabilistic relationships between variables. Its logical and straightforward structure makes it easier to see the connections between the data. Hence a Bayesian network (BN) has proven to be quite useful in representing various data analysing, and it is understandable easily visual form, such as for modelling probable conditional relationships between illnesses and symptoms. Using this model, the probability that a person has certain illnesses can be calculated when symptoms are seen in one person.

A very useful aspect of BNs is that there is no technically very little data. The minimum sample size is not required to perform the analysis and BNs take into account all available data [61]. Also, Kontkanen et al. [40] shows that Bayesian networks can show good predictive accuracy even at very small sample sizes. It is also possible to use data to learn the structure of BN. Furthermore, Bayesian network models have the advantage that they can easily include information from different accuracy and different sources in a mathematically consistent manner. Expert knowledge can be combined with data regarding variables on which no data exist.

Since BNs are analytically resolved, they can quickly respond to queries after the model is compiled. The compiled form of a BN contains a conditional probability distribution for each combination of variable values, and therefore, unlike simulation models, where the results need to be simulated, it can instantly deliver any very long distribution. Therefore, BNs have many benefits in comparison to

other machine learning methods.

In the literature, we had many kinds of causal structure learning algorithms which have been developed successfully and applied to many different areas [67, 81, 10, 33, 28]. One of them is the famous Fast Causal Inference (FCI) algorithm proposed by Spirtes et al. in 1999 [81] was one of the first algorithms that was able to validly infer causal relations from conditional independence statements in the large sample limit, even in the presence of latent and selection variables but ignores changing structure. Although all are successful structure learning algorithms, almost all these algorithms share an important feature. They assume that causal structure does not change during the data collection process. In real-world scenarios, a causal structure often changes [43]. To quickly identify these changes and then learn a new structure are both crucial. Therefore, it is not possible to determine these changes with existing batch-learning approaches; instead, the structure must be learned in an online manner.

"Online" does not necessarily imply fast or streaming; online means that information is processed as soon as it is received. Therefore, online learning algorithms should be able to handle data as soon as it is received without beginning from scratch and without reprocessing past data. There exist some online learning algorithms [46, 45, 73, 76] in the literature, which is capable of detecting changes. However, they have to begin from scratch when the algorithm detects a change. Therefore, there are online algorithms which have a capacity of learning causal model proposed as online algorithms but there is no such an online algorithm yet which works without beginning from scratch and in the presence of latent variables.

Then,

- *How to identify the causal structure change quickly and then learn the new causal structure in the presence of confounding factors as soon as new data is present?*

- *How to use the prior models while learning structure that changes with data streams?*

In this study, we present three heuristic online algorithms which aim to fill these gaps, which are constraint-based approaches. We believe that these algorithms will make a useful contribution to online structure learning in the presence of latent variables.

All algorithms are separated into three stages. The Online Covariance Matrix Estimator (OCME) receives each datapoint sequentially as input and estimates a covariance matrix to provide the raw materials for learning the causal structure. The Causal Model Change Detector (CMCD) tracks the divergence between recent data points and the estimated covariance matrix to detect changes in the structure, or significant errors in estimation so the unfitness between the current model and a new data point could refer a change or an error. It then uses that information to adjust the weights on previous data points. The Causal Model Learner (CML) takes the covariance matrix and learns the causal model then. The three algorithms proposed to begin to separate in CML part.

The first online causal structure algorithm we propose here is the Online Fast Causal Inference (OFCI). OFCI is an online version of the Fast Causal Inference (FCI) algorithm. This algorithm is modified using the FCI instead of the PC algorithm in DOCL algorithm proposed by Kummerfeld and Danks in 2012[46]. Therefore, the algorithm became an algorithm that can learn in the presence of hidden variables. Causal insufficiency is a common problem when learning BNs from data, where data fail to capture all the relevant variables. Variables not captured by data are referred to as latent variables (also known as unobserved or unmeasured variables). In the real world, latent variables are impossible to avoid either because data may not be available or simply because some variables are unknown unknowns for which we will never seek to record data. Therefore the

ignoring the existence of latent variables can be a problem in structure learning.

The OFCI asserts that when given a learned causal structure, as new data points arrive the correlations will be revised with the estimation of the weight of each causal interaction in OCME part, and the structure will be re-learned when data present evidence that the underlying structure has changed in CMCD part. The newer data points are considered more important than older data points therefore the newer data points are weighted more heavily after a change occurs. In particular, the method can estimate the causal structure even when its structure is changed multiple times and allows us the observe the model at any time we specified.

The second online causal structure algorithm we propose here is Fast Online Fast Causal Inference (FOFCI). FOFCI is a modified version of OFCI in a way to minimise the learning cost of the current model. In OFCI, structure learning is done only at change detection points, and the learned models are not stored or used for the next learning. However, FOFCI stores the previously learned model and check similarities between the updated correlation matrix and the previous model. If some relationships between variables in the previous model still fit the correlation matrix updated with the incoming data, the independence tests to learn these relationships are ignored. Thus, FOFCI minimises the learning cost of the current model. As the similarities between the current model and the incoming data increase, FOFCI learns much faster than OFCI and FCI. When the current model is completely changed, the performance of FOFCI is identical to the OFCI. That means FOFCI and OFCI outputs the same independence model for the same datasets. The advantage of FOFCI requires far less conditional independence tests to learn the same model with OFCI. The only disadvantage of FOFCI needs more memory than OFCI while learning, as it stores also learned models at change points. It may be an important disadvantage for big datasets.

OFCI and FOFCI have to perform a series of conditional independence tests which play an essential role in their complexity. In probability theory, two random events and are conditionally independent given a third event precisely if the occurrence of and the occurrence of are independent events in their conditional probability distribution given. The conditional independence tests performed by the algorithms increase exponentially with the number of variables in the data set so that these algorithms may become computationally infeasible for large graphs.

In the real world, we have always had a vast amount of data such as genetic datasets contain thousands of genes or neuroscience datasets contain tens of thousands of voxels. That means millions of conditional independence tests. This fact indicates that we need algorithms that are fast and flexible.

We, therefore, propose an alternative algorithm to these algorithms for one who wants to analyse large data sets in the best possible time by allowing them to have less informative results. The algorithm we offer here is Really Fast Online Fast Causal Inference (RFOFCI). RFOFCI is a modified version of FOFCI to minimise independence tests by ignoring some conditional independence tests given subsets of Possible-D-SEP sets (which is defined in [88]), which can become very large for sparse graphs. Therefore, RFOFCI uses dramatically fewer conditional independence testing than FOFCI. That makes RFOFCI faster than FOFCI for sparse graphs. Conversely, the output of RFOFCI can be less informative in some cases, most notably concerning conditional independence information. That means RFOFCI outputs may not be the same with FOFCI for the same datasets. However, FOFCI always outputs the same or more converging models by comparing RFOFCI.

The RFOFCI asserts that when given a learned causal structure, as new data points arrive the correlations will be revised with the estimation of the weight of each causal interaction in OCME part, and the structure will be re-learned

when data present evidence that the underlying structure has changed in CMCD part. The newer data points are considered more important than older data points therefore the newer data points are weighted more heavily after a change occurs. RFOFCI both updates the existing correlations in the light of new data and also uses the current causal structure in an attempt to speed up learning the new causal structure.In particular, the method can estimate the causal structure even when its structure is changed multiple times.A probabilistic scheduler which is optionally added algorithm, which allow us the observe the learning structure at any time we required.

1.2 Structure of thesis

The study covers the online causal structure learning algorithms which allow us to take data points sequentially as long as it is available, to update correlations between variables when data becomes available and to learn structure at each change point. Most of all, the algorithms use the previous model information in the learning process, saving us the extra computational cost of the re-learned structure at each change.

We believe that these algorithms will provide an optimal solution to the problem of learning a structure which fits data best. While new data is available, the learning process in batch-mode learning algorithms has to be started from scratch regardless of whether there are any changes in structure. Therefore, other problem arises from repeating learning process such as exponentially increasing computational cost. Through the algorithms we want to propound, we will be able to adapt the new data which is coming in sequential order to the current model without the need to repeat conditional dependency tests in each learning time. The proposed system is limited to continuous data under the assumption that all data are nor-

mally distributed. This problem is much harder for categorical/discrete variables or non-linear systems, as there will typically not be any compact representation of the sufficient statistics.

The rest of the thesis is structured as follows:

Chapter 2 covers Literature review. In this part, we are going to state the research work via preliminary information of research area we chose. Probability theory, probabilistic graphical models, Markov Networks, Bayesian networks, and learning Bayesian networks models will be tried to explain in detail, respectively.

Structure learning is separated into two categories. Firstly, score-based approaches working with scoring functions which measure the fitness data and network are given. Then secondly, constraint-based approaches and independence tests for learning structure of BNs are explained.

Next, we give the concept of causality, and the difference between Bayesian and Causal model are explained. We explained both learning the causal model structure without or in the presence of confounding factors, respectively. PC, FCI and RFCI algorithms are focused on, which are causal structure learning algorithms. The notions of interventions and latent variables are stated. Lastly, online learning which is the fundamentals of our study is presented.

For the first two parts (OCME and CMCD), in this study, we just described these parts and their functions. OCME and CMCD are identical in DOCL, OFCI and FOFCI and proposed by Kummerfeld and Danks in 2012 [46]. These parts will be given in the next chapter, but in-depth mathematical pieces of information such as properties, diligence, convergence and proofs can be found in Kummerfeld and Dank's works [44][46][45].

In **Chapter 3**, we will describe in detail the Online Covariance Matrix Estimator and Causal Model Change Detector proposed by Kummerfeld and Danks in 2012 [46]. Next, we will give the structure learning (CML) part, which is the part

that separates Kummerfeld and the three algorithms (OFCI, FOFCI and ROFCI) to be presented here. We continue with detailed descriptions and structural differences of three online structure learning algorithms which are OFCI, FOFCI and RFOFCI, respectively.

In **Chapter 4**, The experimental results of these algorithms will be presented as synthetic and real-world data results. The algorithms proposed here performed on a different type of data by including learning performances and learning time, which is small, average and large scale.

Chapter 2

LITERATURE REVIEW

2.1 Probabilistic Graphical Models

In this part, we provide a general overview of probabilistic graphical models (PGM) and a detailed overview of Bayesian networks. First, we give some standards and definitions to create basic information about probability and graph theory. The purpose of this part is to review some basic concepts and introduce some notations later in this text. Probabilistic graphical models provide a graphical presentation for compact encoding of a complex distribution in a high dimensional space [39]. A PGM is a compact representation of a joint probability distribution in which we can obtain marginal and conditional probabilities [89].

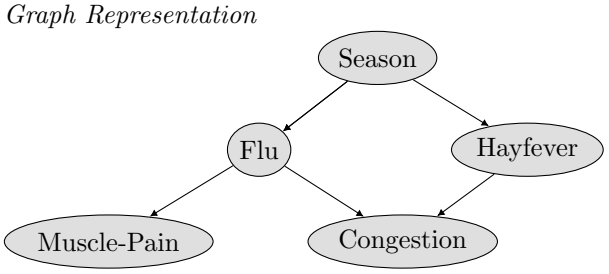


Figure 2.1: (a) A sample Bayesian Network. (b) A sample Markov Network. (example taken from [39])

We will give two graphical presentations of distributions which are called Markov networks (MNs) which use an undirected graph to represent relationships, and Bayesian Networks which use a directed graph to represent relationships, as shown in Figure-2.1 (a) and (b). In these representations, nodes represent the variables and edges represent direct probabilistic interactions between nodes.

2.1.1 Probability and Information Theory

The history of probability goes back to the seventeenth century. In those years, the gambler De Mere consulted to the famous mathematician Pascal to increase his chance in games. Then, Pascal entered into correspondence with another mathematician Fermat about it. These conversations led to the beginning of mathematical probability studies. During the 18^th century, this research was moved from games to science. These probability studies were continued by important researchers such as Huygens, Bernoulli, and DeMoivre [25]. The foundation

of the studies of contemporary probability theory laid to the 1930s.

Graphs are the best way to visualise relationships between random variables, such as family trees. Graphical models use graphs as a tool. In 2001, Kevin Murphy also declared that "Graphical models are a marriage between probability theory and graph theory". Probabilistic graphical models are used as a tool for dealing with uncertainty, independence, and complexity [39] For directly encoding complex distributions over the high-dimensional spaces, probabilistic models are useful tools, which is combining probabilities and independence constraints [39].

Probability theory became prominent. Therefore the academic works in this field have increased because probability has many applications in every research area. These are some important definitions in probability theory.

Definition 1. ***Random Variables***- *is thought that it is an outcome of a measurement process [98].*

Definition 2. ***Bayes theorem***- *In probability theory and statistics, Bayes' theorem defines the probability of an event based on previously acquired information about the event [98]. It is defined as mathematically;*

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} \tag{2.1}$$

where A and B are events and $P(Y) \neq 0$.

Definition 3. ***Probability and Conditional Probability***- *The probability distribution of X is expressed with $P(X)$. $P(X)$ is a list of probabilities and $x \in X$ is its possible values. Conditional probability is shown as $P(X|Y)$ that is the probability of X when given Y [98].*

$$P(X|Y = y) = \frac{P(X, Y = y)}{P(Y = y)} \tag{2.2}$$

Definition 4. ***Independence****-In probability, we say two events are independent if knowing one event occurred doesn't change the probability of the other event.Given* X *and* Y *are two random variables, when the joint probability of* X *and* Y *is equal the probability of* X *multiplied by the probability of* Y*, we can say that* X *and* Y *are* ***independent*** *written as* $X \amalg Y$

$$P(X,Y) = P(X).P(Y) \tag{2.3}$$

[87].

Definition 5. ***Conditional Independence****-In probability theory, two random events and are conditionally independent given a third event precisely if the occurrence of and the occurrence of are independent events in their conditional probability distribution given. Given that* X*,* Y *and* Z *are random variables, when* X *and* Y *are* ***independent*** *given* Z*, we can say that* X *and* Y *are* ***conditionally independent*** *on* Z *[87]. It is written in the form of* $X \amalg Y|Z$*.*

$$P(X,Y|Z) = P(X|Z).P(Y|Z) \tag{2.4}$$

2.1.2 Graph theory

In this study, we will study with directed graphs whose nodes represent random variables and edges represent relationships between them.

There are several different kinds of graphs in the literature, such as undirected and directed. Hence, all of them has a set of vertices and edges classically. According to the type of structure, graphs show differences.

Definition 6. ***Undirected graph*** *is a graph that the edges between its nodes are undirected, as shown in Figure-2.2 .*

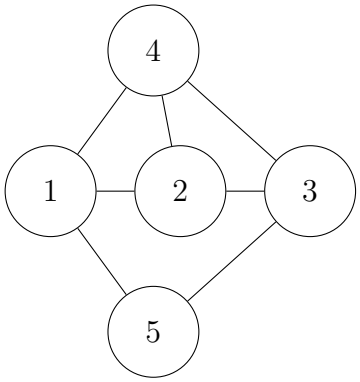


Figure 2.2: Undirected Graph (*adapted from* [87])

Definition 7. ***Directed Graph****- On the contrary to undirected graph, the edges of a directed graph are in a direction, as shown in Figure-2.3*

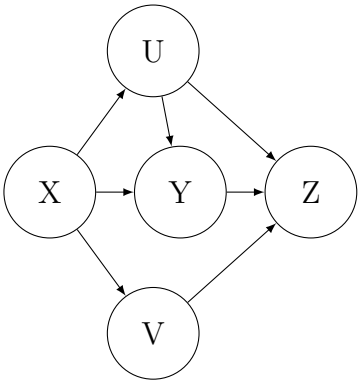


Figure 2.3: Directed Graph (*adapted from* [87])

Definition 8. ***Complete graph*** *is a graph whose vertices are connected to all other vertices [52], as shown in Figure-2.4.*

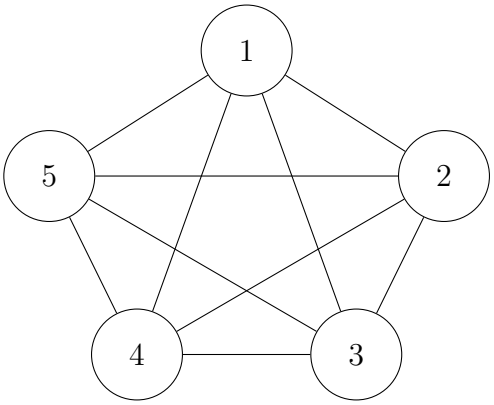


Figure 2.4: Complete Graph

Definition 9. *A vertex **separation set** is a minimal set of vertices whose removal from the graph makes the graph disconnected. More precisely, List of length vertices; each element of the list contains another list of length vertex. For example, the element **sepset[[x]][[y]]** contains the separation set that made the edge between **x** and **y** drop out. Each separation set is a vector with positions of variables in the adjacency matrix [12].*

Definition 10. ***Induced subgraph** of a graph is also graph constructed from a subset of the graph's nodes and all the edges joining the two nodes in that subset [52]. Given $G = (V, E)$ is a graph, an induced subgraph of G is determined by its node-set, an example is shown in Figure-2.5.*

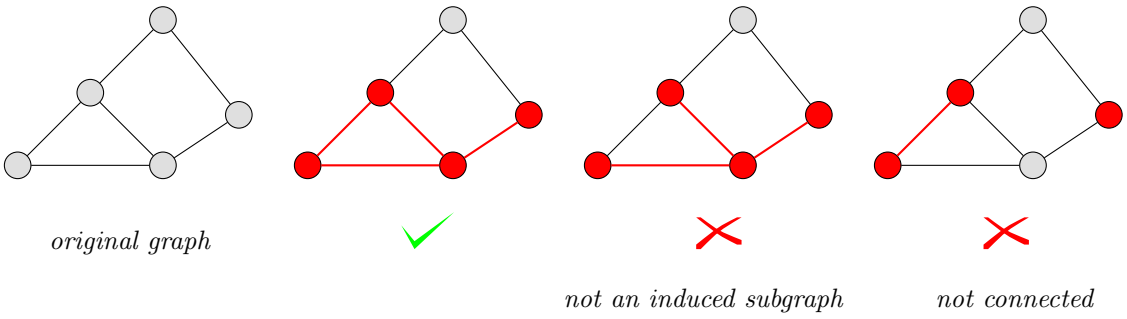


Figure 2.5: induced subgraphs

Definition 11. ***Clique** is a subset of nodes of an undirected graph, where its*

induced subgraph is complete [52]. Given G *is a graph, a clique of a graph* G *is a complete subgraph of* G*, and the clique of the largest possible size is referred to as a maximum clique. An example is shown in Figure-2.6.*

Figure 2.6: cliques

2.1.3 Directed Acyclic Graphs

A directed acyclic graph (DAG) is a kind of directed graph and includes edges and vertices having no graph cycles. DAGs are a good representation method of conditional independence relationships among random variables [82]. Suppose X and Y are random variables, If there is a directed edge from X to Y, we say that X is the ***Parent*** of Y and Y is ***Child*** of X and denote the set of all parents of a vertex X by $pa(X)$.

Definition 12. *Given a DAG* G*, directed acyclic graphs represent conditional independence implied by recursive decomposition, a joint distribution* $X_1, ..., X_n$ *is*

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i|pa_i) \tag{2.5}$$

such that $P(X_i|pa_i)$ *is conditional probability distribution* X_i *given* $pa(X_i)$ *[11].*

Definition 13. ***Markov Condition****- The Markov condition for a graph defines that any variable in a graph is independent of its non-descendants, given its parents. Given a directed acyclic graph* G *over* V *and a probability distribution* $P(V)$

satisfy ***Markov Condition*** *if and only if for any* $W \in V$ *is independent of* V *[87]. That is;*

$$W \perp V \backslash (Descendants(W) \cup Parents(W)) | Parents(W) \tag{2.6}$$

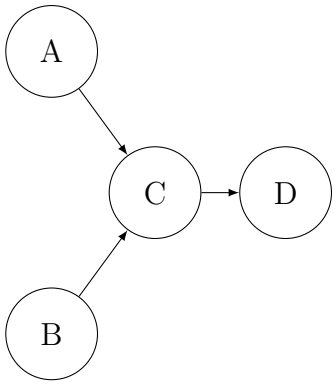


Figure 2.7: ***Markov Condition*** example

Given a directed acyclic graph as shown in FIGURE 2.7. The following conditional relationships satisfies Markov Condition.

$$A \perp B \tag{2.7}$$

$$D \perp \{A, B\} | C \tag{2.8}$$

Definition 14. ***Markov Blanket****-A Markov blanket (boundary) was first proposed by Pearl [65] in a Bayesian network, and is defined as that in a faithful Bayesian network. Given a directed acyclic graph* G*, a set of nodes* V*. For every node* $A \in V$*, its Markov blanket is the set of parents, children and spouses (parents of the children of* A*), as shown in Figure-2.8.*

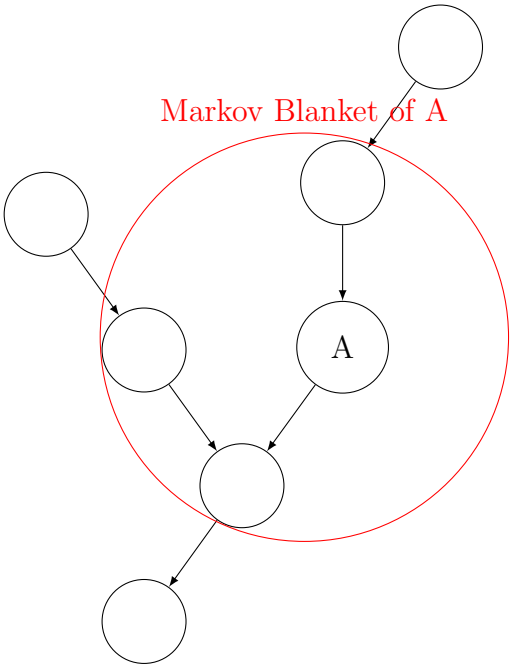


Figure 2.8: ***Markov Blanket*** of A = parents+children+children's other parent's example

2.2 Markov Networks (MNs)

A Markov network (or undirected graphical model) is a set of random variables which have a Markov property defined by an undirected graph. The joint probability distribution of the model can be factorized according to the cliques of the graph G as follow;

$$P(X = x) = \frac{1}{Z}. \prod_{C \in cl(G)} \phi(C) \tag{2.9}$$

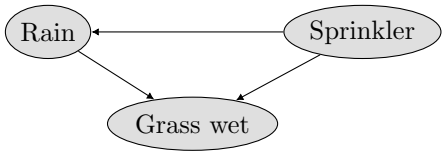
such that Z is a normalisation factor, $cl(G)$ is the set of cliques of G and the function $\phi(C)$ is known as factor or clique potential. Markov networks are useful in the domains where interaction between variables is symmetrical and the direction is not important [98].

2.3 Bayesian Networks(BNs)

"Bayesian" name comes originally from Bayes' theorem of Thomas Bayes. Bayes' rule is the fundamental point of approach to update and improve probabilities by considering new findings [66]. The Bayesian Network term started to be used by Judea Pearl after 1985. Pearl in 1988 and Neapolitan in 1989 defined and summarised Bayesian Networks field and properties in their books "Probabilistic Reasoning in Intelligent Systems" [65] and "Probabilistic Reasoning in Expert Systems" [62], respectively.

According to the definition of Koller and Friedman in "Probabilistic Graphical Model" book, Bayesian Networks are directed acyclic graphs (DAG), whose nodes represent random variables and whose edges correspond to probabilistic relationships between two nodes. This probabilistic model can either be evaluated as a data structure which uses the skeleton to compactly show joint probability distributions in a factorised way or compact representation of conditional independence set about a distribution [39], see example Figure-2.9 and Figure-2.10.

	Sprinkler	
Rain	T	F
F	0.4	0.6
T	0.01	0.99



Sprinkler	
T	F
0.2	0.8

		Grass wet	
Sprinkler-Rain		T	F
F	F	0.4	0.6
F	T	0.01	0.99
T	F	0.01	0.99
T	T	0.01	0.99

Figure 2.9: A simple Bayesian network and conditional probability table

The joint probability distribution of the BN Fig-2.9 is;

$$P(S,R,W) = P(S)P(R|S)P(W|S,R)$$

Let's;

$$P(W=t|S=t)=?$$

$$\begin{aligned} P(W=t|S=t) &= \sum_{R=t,f} \frac{P(S=t,R,W=t)}{P(S=t)} \\ &= \frac{P(S=t,R=t,W=t)}{P(S=t)} + \frac{P(S=t,R=f,W=t)}{P(S=t)} \\ &= \frac{P(S=t)P(R=t|S=t)P(W=t|S=t,R=t)}{P(S=t)} \\ &\quad + \frac{P(S=t)P(R=f|S=t)P(W=t|S=t,R=f)}{P(S=t)} \\ &= 0.0041 \end{aligned}$$

Figure 2.10: Conditional probability distribution example

2.4 Some Principles of BNs

2.4.1 D-separation

Dependence and independence are crucial to understanding the structure of a Bayesian network. Additionally, Independencies are essential to answer questions and reduce the cost of computation. As mentioned previously, Judea Pearl has a big contribution to Bayesian Networks. He extended the relationship between random variables to disjoint subsets of nodes via d-separation [75]. $d-separation$ (d stands for direction) is a simple graphical test rule introduced by Pearl to deduce conditional independence relationships from a directed acyclic graph [59]. Therefore, the necessary semantics to define the network can be obtained.

Definition 15. ***D-separation****-(definition adapted from Pearl, 1995 [66]); Given that* G *is a directed acyclic graph and* V_i *random variables* $(i = 1, 2, ..., n)$*.* A*,* B *and* C *are three disjoints subsets of* V_i*.* p *is a path (any series of edges, regardless of direction) between any node* i *in* A *and any node* j *in* B*. If there is a node* w *on* p *and it satisfies these conditions;*

- p *has a chain* $i \rightarrow w \rightarrow j$ *or a fork* $i \leftarrow w \rightarrow j$*, where middle node* w *is in* C *or*
- p *has an inverted fork (or collider)* $i \rightarrow w \leftarrow j$ w*, where middle node* w *is not in* C *and no descendant of* w *is in* C*.*

It can be said that C *blocks* p*. If and only if* C *blocks every path from a node in* A *to a node in* B*,* C *is* $d-separated$ *from* A *from* B*. [54].*

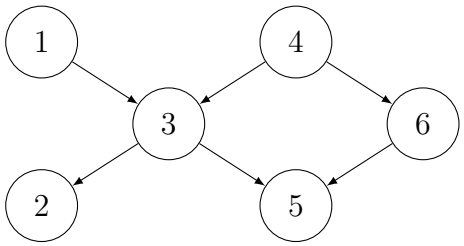


Figure 2.11: a BN for $d-separation$ example

To make it more clear, let investigate it on a simple example. Suppose we have a BN as shown in example Figure-2.11. Some relationships are;

- $1 \perp\!\!\!\perp 6$, all paths between 1 and 6 are blocked because there are two paths between them, which are $1 \rightarrow 3 \rightarrow 4 \rightarrow 6$ and $1 \rightarrow 3 \rightarrow 5 \rightarrow 6$. They are blocked because 3 and 5 are blocked, respectively.
- $1 \not\perp\!\!\!\perp 6$—3, because both 3 and 4 are unblocked.3 does not meet second and 4 does not meet the first condition.

- $1 \not\perp\!\!\!\perp 6$—2, because both 3 and 4 are unblocked. 2 is a descendant of 3, and it does not meet second.

- $1 \not\perp\!\!\!\perp 6$—$\{3, 4\}$, because although 3 is unblocked, 4 is now blocked. Therefore, one of these condition is satisfied.

- $2 \not\perp\!\!\!\perp 6$, because 3 and 4 are unblocked. They are not in $\emptyset$ therefore, the first condition is not satisfied.

- $2 \perp\!\!\!\perp 6$—4, because of 4 satisfied first condition.

- $2 \not\perp\!\!\!\perp 6$—$\{4,5\}$, because 5 does not satisfied the second condition.

2.4.2 Faithfulness

In a DAG, the Markov condition gives a set of independence relations, and these relationships may entail further relationships as well as those given by the Markov condition. For instance, a distribution over the graph in FIGURE 2.12 satisfy Markov Condition. 1 and 4 might be independent in the distribution although the graph does not require this independence.

Definition 16. ***Faithfulness****-Given a directed acyclic graph* G *distributed over the set of vertices* V*, and a probability distribution* P *are faithful to one another* x*. if every one and all independence relations valid in* P *are those entailed by the Markov condition on* G *[87] [81], see example Figure-2.12.*

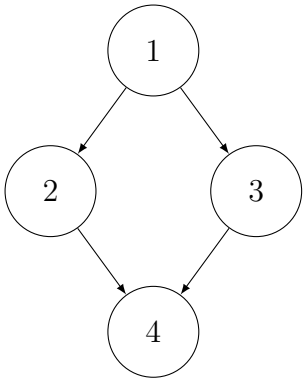


Figure 2.12: ***Faithfulness*** example

2.4.3 Markov Equivalence Class

Many DAGs might define the exactly the same conditional independence information. These DAGs are called Markov equivalent and form a Markov equivalence class. For example, consider DAGs on the variables $\{X_1, X_2, X_3\}$. Then $X_1 \rightarrow X_2 \rightarrow X_3$, $X_1 \leftarrow X_2 \leftarrow X_3$ and $X_1 \leftarrow X_2 \rightarrow X_3$ form a Markov equivalence class, since they all refer the single conditional independence relationship $X_1 \amalg X_3 | X_2$, that is, X_1 is conditionally independent of X_3 given X_2. Another Markov equivalence class is given by a single DAG $X_1 \rightarrow X_2 \leftarrow X_3$, since this is the only DAG that implies the conditional independence relationship $X_1 \amalg X_3$ alone. ***Markov equivalence classes*** of DAGs can be described uniquely by a completed partially directed acyclic graph (CPDAG) [14].

2.4.4 Exact and Approximate Inference

There are many ***exact inference algorithms*** in Bayesian Networks. However, they just provide an effective solution when worked with networks that have small cliques [98]. As inference in BNs are NP-hard problem [16], ***approximate inference algorithms*** are typically used rather than exact inference algorithms

[98]. Well-known approximate inference algorithms are importance sampling, loopy belief propagation, generalised belief propagation, and variational methods.

2.5 Learning Bayesian Networks

In this part, we will give a brief and general introduction to Bayesian network learning. Bayesian network learning algorithms aim to find the network that best encodes the joint probability distribution in the data [7]. There are two main learning tasks: estimating the parameters of a model and learning the structure of a network. First, we introduce the parameter learning problem and then structure learning. In this study, we will only consider data that is fully observed (no missing value). It can be found out more about learning with missing data in [48].

2.5.1 Learning the Parameters

One of the important part of BNs learning process is the parameter learning. In the parameter estimation, when the structure of the network is given, the parameters are determined from data. For the given structure, the parameters indicates the conditional probability distributions [77]. Maximum likelihood estimation and Bayesian estimates are two main approaches in parameter learning [98].

Maximum likelihood estimation (MLE)

A well-known approach for finding data generating parameters is ***Maximum likelihood estimation*** [77]. Given observed values $X_1 = x_1$, $X_2 = x_2$, . . . ,$X_n = x_n$, the likelihood of θ is the function

$$lik(\theta) = f_\theta(x_1, x_2, ..., x_n) = f(x_1, x_2, ..., x_n|\theta) \tag{2.10}$$

considered as a function of θ. In a nutshell, the maximum likelihood estimate of θ is that value of θ that maximises $lik(\theta)$.

If the distribution is discrete, the maximum likelihood is the principle of predicting the probability distribution of the parameter that best fit the data [98]. The probability of parameters is predicted by using their frequency in the observational data [98]. If the X_i are iid, then the likelihood simplifies to

$$lik(\theta) = \sum_{i=1} nlog(f(x_i|\theta) \tag{2.11}$$

.

Bayesian estimation (BE)

MLE views θ as quantities whose values are unknown but fixed parameter. In ***Bayesian estimation***, θ is assumed to be a random variable which has some known prior distribution. In other words, θ is a quantity whose variation can be described by the prior probability distribution $P(\theta)$ [98].

2.5.2 Learning the Structure: Score-based methods

One of the most studied ways of Bayesian Network structure learning is score-based techniques. It addresses the problem as a model selection. The score based methods of Bayesian Networks searches highest scored directed acyclic graph (DAG), where a certain score function measures the fitness data and model. This is an NP-hard problem and subject is an advanced research topic. Even using the latest theoretical advances [18], the method is impractical even when you set a limit for learning where the best DAG has no more than two parents nodes.

Given the dataset D, the score of a possible structure is

$$Score(G, D) = Pr(G/D) = \frac{Pr(D/G)Pr(G)}{Pr(D)} \tag{2.12}$$

Akaike Information Criterion (AIC) proposed in 1973 [2] is considered as one of the most well-known score function. AIC provides a simple and effective means for the selection of the best-approximating model to the true model [6]. The basic formula is defined as:

$$AIC = -2/N * LL + 2 * k/N \tag{2.13}$$

Where N is the number of examples in the training dataset, LL is the log-likelihood of the model on the training dataset, and k is the number of parameters in the model. Concerning general linear models, AIC is known to perform relatively well for small samples, however, the criterion does not tend to select the true model in large samples. Nevertheless, the form of this expression is very similar to BIC (below). We see that the penalty for AIC is less than for BIC. This causes AIC to pick more complex models. However, this can result in better predictive accuracy.

Another well-known score function is considered as Bayesian Information Criterion (BIC), which is a method for scoring and selecting a model. Its name is derived from the field of study which is Bayesian probability and inference. Like AIC, it is appropriate for models to fit under the maximum likelihood estimation framework. The basic formula is defined as:

$$BIC = -2 * LL + log(N) * k \tag{2.14}$$

Where log() has the base-e called the natural logarithm, LL is the log-likelihood of the model, N is the number of examples in the training dataset, and k is the number of parameters in the model. The quantity calculated is different from AIC,

although can be shown to be proportional to the AIC. Unlike the AIC, the BIC penalizes the model more for its complexity, meaning that more complex models will have a worse (larger) score and will, in turn, be less likely to be selected.

Another well-known score function is the Minimum Description Length (MDL). Jorma Rissanen proposed MDL in 1978. MDL is simply negative of BIC. Although, MDL and BIC are proposed in the same year. They are independent studies.

As the number of variables is large, direct searching could be intractable, so network searching space grows exponentially depending on the number of variables. For n variables, there are $n(n-1)$ possible directed edges and possible structures and $2^{n(n-1)}$ possible structures for every subsets of these edges. Searching the all possible structures is mindless and instead, heuristic methods are used.

Well-known heuristic score based algorithm is hill-climbing. Hill Climbing search is a greedy search algorithm. Its idea is to produce a model step-by-step by enabling maximum improvement in an objective quality function at every step. As you see in Fig 2.13, in *hill-climbing* search, the first step might be an empty, full or a random network. The parameters of the local probability distribution functions are estimated by Probability Tables given a BN structure. Generally, *maximum-likelihood estimation* is used. The loop in the algorithm is that;

- Trying each possible *single-edge* addition, removal or reverse
- Making a network increasing the graph score the highest
- Iterate until the process stop.

The process stops until there is no single edge change increasing the score., see example Figure-2.13.

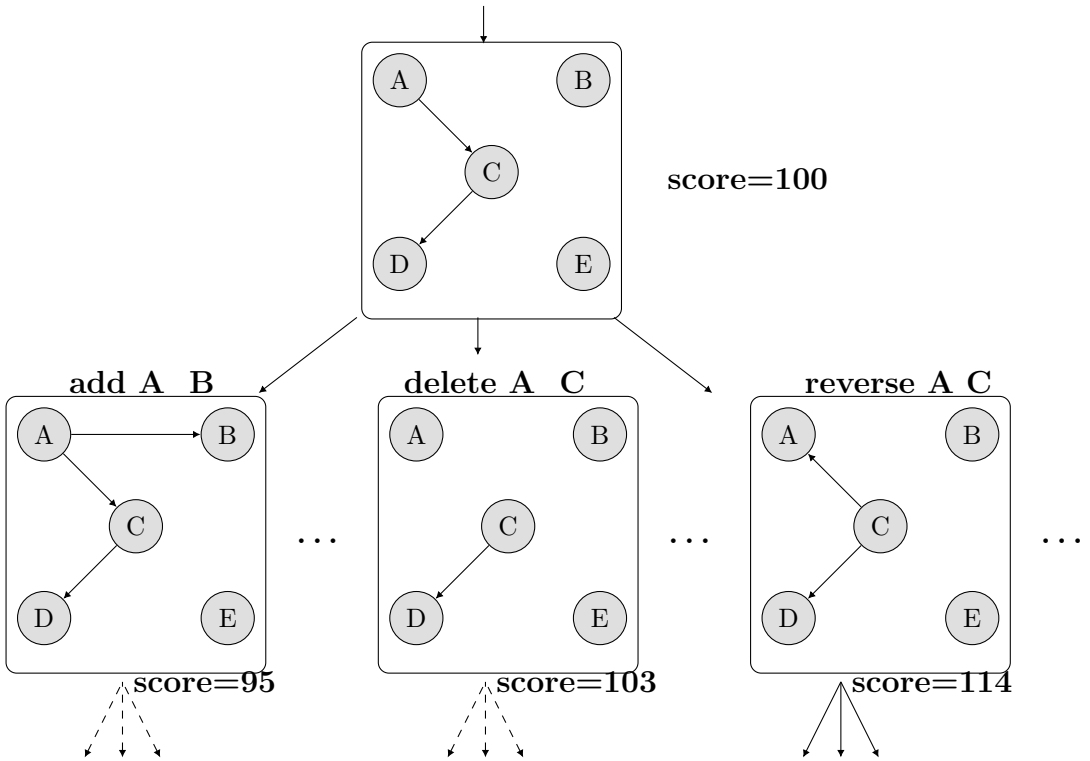


Figure 2.13: Hill-Climbing example taken from [57]

This is a maximum likelihood estimation of the probability entries from the dataset. Algorithm attempts every possible single-edge addition, removal, or reversal, which increases the score of the network, and iterates. The process stops when there is no single edge change that increases the score. Hill-climbing does not guarantee good results every time. The algorithm may not settle at a global maximum every time. Hill Climbing algorithm can get stuck in local minima. If you get bed results. There available more advances methods such as Tabu Search, Simulated annealing and Genetic algorithm. When the initial searching condition does not properly get, Score-based structure learning algorithms may get stuck in a local maximum.

TABU search is a slight modification to Hill Climbing search. As Hill Climbing search may get stuck in a local minimum, TABU search solves this problem by

maintaining a TABU list which is the list of previously visited states. TABU search will not allow any addition or removal of the edge which makes the network go to the state that is already in TABU list.

Genetic Algorithms (GAs) are probably the most popular evolutionary algorithms with a wide variety of applications [96]. GAs are often viewed as function optimizers and solve the vast majority of known optimization problems [96]. GAs stochastically transform candidate solution sets into new to find one solution that suitably solves the problem at hand. solution groups to find a solution that solves the existing problems appropriately. The quality of each candidate solution is expressed by using a user-defined objective function called fitness.

Simulated annealing is based on the metal annealing processing [34]. Unlike gradient-based methods and other deterministic search methods, the main advantage of simulated annealing is that it can avoid getting stuck in local optima [96].

As the initial structure is known with some edges in advance, the probability of score-based algorithms to settle down on the global maximum will increase. Several learning algorithms in this approach have been developed. However, the Bayesian network score-based learning is adversely affected by exponential time and NP-hard problems. Consequently, the SB approach makes it difficult to apply to a large network.

2.5.3 Learning the Structure: Constraint-based methods

Constraint-based algorithms, which is also known as conditional independence test-based, approaches [50], tackle BNs as a representation of independence. The constraint-based algorithms learn the network structure by investigating the probabilistic relationships entailed by the Markov property of Bayesian networks by using conditional independence tests. After that, a graph is structured, which satisfies d-separation statements [75]. These methods aim to test conditional de-

pendence and independence in the data and then obtain the best network, which represents these dependence/independence [39].

Let explain by an example.

Example 1. *Suppose we have X and Y variables and S is a subset of variables. The test is to check $X \amalg Y|S$ or not [41]. The results of the test are the constraints, and the graph has to satisfy them.*

Constraint-based algorithms have a similar structure learning for causal models. *These algorithm consists of three main steps [75]:*

first the skeleton of the network (the undirected graph underlying the network structure) is learned. Since a comprehensive search (independence test between all nodes) is not computationally unfeasible for all other than the simplest datasets, all learning algorithms use some kind of optimization, such as limiting the search to the Markov blanket of each node (which includes the parents, the children and all the nodes that share a child with that particular node).Identify the direction of all edges having a ***v-structure*** (A v-structure is an ordered triplet of nodes (XYZ) such that there is an edge from X to Y and from Z to Y, but no edge between X and Z, $X_j \rightarrow X_i \leftarrow X_k$). The rest of the edges must satisfy acyclicity constraint.

Well-known constraint-based algorithms Grow-Shrink [57], Incremental Association [91], Fast Incremental association [97], Interleaved incremental association [91], Max-min parents and children[92], Chow-Liu [15], Boundary DAG [9], Inductive causation [67], SGS algorithm [9], Wermuth-Lauritzen algorithm [9] and PC-algorithm [9].

2.5.4 Independence Tests

The algorithms based on independence tests perform a quantitative study of the dependence and independence relationships between the variables in the domain. In simple terms, the main idea in BNs is to find conditional independence between data variables and to reduce probability calculations by using these independencies. Suppose we have two variables X and Y. The question, which is for determining whether X and Y are independent or not. $P(X,Y) =^{?} P(X).P(Y)$, where $P(X)$ and $P(Y)$ are stated as marginal probability distributions, $P(X,Y)$ is joint probability distribution.

Many conditional independence tests are available for using in constraint-based learning algorithms. Indeed, these tests must be specified to the test argument such as discrete or continuous data.

In general, Pearson correlation ρ does not represent the independence or dependence, but a linear relationship between between two continuous variables. If we assume that paired two random variables are either independent or just linearly related, then Pearson correlation can be used to measure independency. It is known as the best method of measuring the association between variables of interest because it is based on the method of covariance. It gives information about the magnitude of the association, or correlation, as well as the direction of the relationship.

$$H_0 : \rho = 0 \rightarrow there\ is\ no\ relationship\ between\ variables \tag{2.15}$$

$$H_1 : not\ H_0 \rightarrow there\ is\ a\ relation \tag{2.16}$$

Let n be the number of observations. Exact t test for Pearson's correlation

coefficient ρ.

$$t = \rho\sqrt{\frac{n-2}{1-\rho^2}} \tag{2.17}$$

is approximately distributed as $t(n-2)$ under H_0.

Let n be the number of observations. Fisher's Z test,

$$z = \frac{\sqrt{n-3}}{2} log(\frac{1+\rho}{1-\rho}) \tag{2.18}$$

is approximately distributed as $N(0,1)$ under H_0.

We looked at how you can calculate linear dependencies between two continuous variables with covariance and correlation. Both methods use the means of the two variables in their calculations. However, mean values and other population moments make no sense for categorical (nominal) variables.

For instance, if you denote "Clerical" as 1 and "Professional" as 2 for an occupation variable, what does the average of 1.5 signify? You have to find another test for dependencies—a test that does not rely on numeric values. You can use contingency tables and the chi-squared test. Let's take a closer look at these two methods.

Contingency tables are used to examine the relationship between subject scores for two qualitative or categorical variables. They show the actual and expected distributions of cases in a cross-tabulated format for the two variables, see table 2.1.

If the columns are not contingent on the rows, then the row and column frequencies are independent. The test of whether columns are contingent on rows is called the chi-squared test of independence. The null hypothesis for this test is that there is no relationship between row and column frequencies—in other words, that the variables are independent. Therefore, there should be no difference between the observed (O) and expected (E) frequencies.

Table 2.1: Contingency table example

		Death Penalty	
Victim's Race	Defendant's Race	Yes	No
White	White	19	132
	Black	11	52
Black	White	0	9
	Black	6	97

The chi-squared test simply calculates the sum of the squares of differences between observed and expected frequencies divided by expected frequencies. This formula is also referred to as the Pearson chi-squared formula.

$$X^2 = \frac{1}{n} * \sum_{i=1}^{n} \frac{(O-E)^2}{E} \tag{2.19}$$

There exist premade tables that list the critical points for chi-squared distributions. If your calculated chi-square value is greater than a critical value in such a table for the defined degrees of freedom and a specific confidence level, you can reject the null hypothesis with that confidence. For a table with R rows and C columns, the degree of freedom is calculated as the following product:

$$DF = (C-1) * (R-1) \tag{2.20}$$

In conclusion, we have to be aware that we may not see big differences between these independence tests results because as it is seen, they have similar formulations.

2.5.5 Learning the Structure: Hybrid methods

Hybrid methods try to achieve the best of both learning methods: learn a skeleton with a Constraint-based approach and constrain on the DAGs considered

in the Score based. Both approaches have their advantages and disadvantages.

Constraint-based approaches are relatively fast, deterministic and have a well-defined stopping criterion; however, they rely on an arbitrary level of severity to test dependency, and they may be unstable as an error at the start of the search can have a domino effect that causes many errors to be found in the final network.

Score based approaches have the advantage of flexibly involving the background information of users on structures in the form of previous possibilities and can also deal with missing records in the database (e.g. EM technique). Score based methods are preferred when dealing with small data sets, but they are slow to converge and the finding optimal BN structures are often impossible due to computational complexity for larger groups of variables. The computing load becomes obstructive.

Therefore, the key idea is to restrict search locally around the target variable by using the advantage of CB methods over SB methods. They can create a local graph around the target node without having to build the entire BN first. Tsamardinos et al. proposed the Max-Min Hill Climbing (MMHC) [92]. Their study shows that MMHC outperformed both in terms of time efficiency and quality to many algorithms such as the PC algorithm [81], the sparse candidate algorithm [24], Greedy equivalence search [10], and the greedy hill-climbing search on a variety of networks.

Perrier et al. proposed [72] a hybrid algorithm that can learn an optimal BN when an undirected graph is given as a structural constraint. They defined this undirected graph as a super-structure. This algorithm can learn optimal BNs containing up to 50 vertices when the average degree of the super-structure is around two, that is, a sparse structural constraint is assumed.

To extend to the feasibility of BN with a few hundred of vertices and an average degree up to four, Kojima et al. proposed [38] to divide the superstructure into

several clusters and perform an optimal search on each of them to scale up to larger networks. Despite interesting improvements in terms of score and structural hamming distance on several benchmark BNs, they report running times about 103 times longer than MMHC on average, which is still prohibitive. Therefore, there is a great deal of interest in hybrid methods capable of improving the structural accuracy of both CB and SB methods on graphs containing up to thousands of vertices. However, they make the strong assumption that the skeleton contains at least the edges of the true network and as small as possible extra edges.

2.5.6 Evaluating Structural Accuracy

How can we measure the accuracy of a structure learning algorithm? One usual solution is choosing an existing network, gold standard (or randomly) and generate the dataset from its joint probability distribution. Further, the algorithm is applied upon generated dataset and learned the structure of the network. For evaluation purposes, structural accuracy of learned networks can be measured with a variety of different metrics that compare the structure of both the learned and true models. We use a very basic method by comparing manually (with simple Matlab code) the learned BN with the initial one. The first metric is the accuracy of edges in the learned model and second is directions.

In the literature, there exist different techniques based upon one of the methods, e.g. the Bayesian Dirichlet equivalent uniform (BDeu), Bayesian information criterion (BIC), Kullback-Leibler divergence (KL) and Structural Hamming Distance (SHD) based along with sensitivity and specificity based methods [63]. Each method has pluses and cons, some of them are complex to compute, but they take into account Markov equivalent classes, e.g. KL-divergence based methods. Furthermore, others are simple but do not consider Markov equivalent class e.g. sensitivity and specificity based methods.

BDeu score proposed by Heckerman et al [27]. The scoring function used in the learning algorithms, measures how likely the network is given the data. BIC can be regarded as the likelihood of the learned structure after having seen the data with a penalty term of model complexity measured by the number of parameters. Both BDeu and BIC have the limitation that they are only reasonable under certain assumptions. In BDeu, parameter independence is violated when data is missing.

To directly measure how close the true network and the learned network, Kullback-Leibler (KL) divergence can be used. KL (also known as cross-entropy) [42] is a non-symmetric measure that quantifies divergence between the joint probability distributions associated respectively with the true network and the learned network. KL divergence rewards equally all statistically equivalent models, but it does not take into account causal relationships which distinguish between observationally equivalent models.

BDeu and KL-div do not rely on the true structure of the original as it is unknown. They just rely on the dataset by assuming this data represents the true structure. However, BN structure learning algorithms are often evaluated against networks created by experts.

One of the most popular methods is the structural hamming distance (SHD) proposed by Tsamardinos [92], based on directly the true network and the learned model. the raw counts of errors in the learned model. The SHD of a model is a type of graph edit distance and is equal to the number of edge deviations between the model and the true model. Its use is fully oriented toward discovery, rather than inference. the SHD is composed of the sum of five sub-error measures: extra edge, missing edge, extra direction, missing direction, wrong direction. The Structural Hamming Distance considers directed acyclic graphs (DAGs) and partially directed acyclic graphs (PDAGs) and counts how many edges do not coincide. However, it has limitations partially ancestral graphs (PAGs) which we will work.

2.5.7 Correlation and Causation

intuitive description

Association is the statistical dependence between events or characteristics. Positive association means a direct relationship and negative is opposite. However, these relations do not always imply causation. To repeat, Pearl emphasises that the purpose of the many sciences is that understanding mechanisms through variables and values they take and estimating the values of these variables if naturally occurring mechanisms are exposed to external manipulations [85].

For example, Epidemiologists collect data on dietary habits and life expectancy in the general population. They aim to find out which nutritional factors affect people's life expectancy and estimate the effects of recommending people to change their diet. [85]. Therefore, the finding answers to queries about the mechanisms are causal inference [85].

Let emphasise with a simple and good example inspired by the internet. Chief in a clothing store decides to rearrange the inventory on his floor. He arranges the athletic wear and shoes in a notable spot in the store, the swimwear next to the first register and the business wear to a less visible spot. Over the next few weeks, he recognises some changes in his employees. They are active, eat healthier and take walks on their breaks. Could the athletic wear in a prominent spot cause the employees to have the motivation to be healthier? He tries to be sure by exchanging the athletic and business wears amongst themselves. Over the next few weeks, he cannot see any change. Therefore, he asks them what the reason that caused them to suddenly want to work out and live a healthier lifestyle was. Was it the athletic wear? However, employees said no. It was the swimsuits by the front register reminding them that spring break was coming.

mathematical description

The main aim of standard statistical analysis is to estimate parameters in a distribution under conditions that remain the same, which lets researchers estimate the probabilities of future events [70]. Causal analysis is a more general version of this analysis because it allows us to make deductions under conditions that change [70].

Correlation and causation are different things. We can tell by looking at the experiences we have encountered in life that correlation does not imply causation. Nevertheless, they have a strong relationship. Correlation measures the strength of the relationships between variables which would be negative or positive. The correlation between two variables can be calculated quantitatively via the coefficient of correlation [95]. The most popular one is Pearson's coefficient of correlation;

$$r_{xy} = \frac{cov(x, y)}{\sigma_x . \sigma_y} \tag{2.21}$$

where x and y are samples and $cov(x, y)$ is covariance showing how much x and y change together.However, during this process, the relationship among variables are ignored, but we have an experimental ground to believe that some factors are direct causes of others or other pairs are related to a common cause [95]. Causation indicates an observed action which causes the second action. Causation is an abstract term going to show the progress on of the world. Therefore, we may need an analysis method that combines the correlation coefficient and causation.

2.6 Non-Causal and Causal Models

A BN only contains statistical information. Meaning that anything you can infer from the joint probability table you can infer from the directed probabilistic

relationship, nothing more, nothing less.

A causal relationship is something else entirely. A Causal Bayesian Network must specify what happens under any variable intervention. A causal Bayesian network is a directed acyclic graph (DAG), in which each node corresponds to a distinct variable V_i in the domain, and each edge corresponds to causal effect from the parent node to the child node. The parent node of an edge is the node at the tail of the edge, and the child node is the node at the head of the edge [49].

In causal Bayesian networks, the meaning of ***causality*** is as follows: When we change the parent variable by fixing its state to different values, we can observe the change in the probability distribution of the child variable. If there is no causal effect from variable V_x to variable V_y , there will be no edge from variable V_x to variable V_y in the causal Bayesian network [67] [49].

While the DAGs and the probability theory form the computational part of BNs, Causality between variables is the most philosophical part. Therefore the causality is required to be studied carefully and considered deeply. The important point is that although BNs are mostly used to represent causal relationships, it does not always have to be. If a BN is a causal model, the relationship between nodes must be causal.

The evolution in BNs studies in parallel with those of causality. In 1921, Sewall Wright used for the first time Causal models for model genetic inheritance in his seminal study is "Correlation and Causation" [95]. Additionally, he used causal models via directed graphs in 1934. Another important contributor to Causal models is Judea Pearl. He has the studies for causal and counterfactual inference. Causal inference is a process of estimating the causal quantities. The counterfactual inference is an important part of causal inference.Counterfactual inference simply is to determine the probability that the event y would not have occurred ($y = 0$) had the event x not occurred ($x = 0$), given the fact that event

x did occur ($x = 1$) and event $y$vdid happen ($y = 1$), which can be represented as $P(y_{x=0} = 0|x = 1, y = 1)$, where $y|x = 0$ is a counterfactual notion, which denotes the value of y when the setting is $x = 0$ and the fixing effects of other variables are unchanged, so it is different from the conditional probability ($Py|x = 0$).

In the 1990s, the studies were done by essential researchers such as Spirtes, Glymour, Scheines [87] and Pearl [71] had a vital role for viewing BNs as causal graphs.

After that, Pearl did a fundamental study which is "Causal diagrams for empirical research". The way of using graphical models as a mathematical language was shown to combine statistical and subject information [66]. He defined a symbolic calculus ($do(x)$) allowing us to quantify causal effects from experimental data. In the causal model, the do-calculus [69] simulates the physical interventions that force some variables X to take certain constants x. Formally, the intervention that sets the values of X to x is denoted by $do(X = x)$. The intervention $do(X = x)$ manipulates the causal graph. Although, it is complicated to define fully, the basic idea of do-calculus is as defined below.

Definition 17. ***do-calculus*** *(definition adapted from Pearl, 2012 [69]); Given $\boldsymbol{G}$ is a directed acyclic graph associated with a causal model and V_i is a set of random variables ($i = 1, 2, ...n$). X, Y, Z, and $W \subset V_i$ are any disjoints four subsets of nodes. The union of them equals V_i. Three rules are valid for every interventional distribution compatible with $\boldsymbol{G}$;*

- ***Rule 1.*** *(Addition or Removing of Observations) If $Y \amalg Z|X, W$*

$$P(y|do(x), do(z), w) = P(y|do(x), z, w) \tag{2.22}$$

- ***Rule 2.*** *(Action or Observation exchange) If $Y \amalg Z|X, W$*

$$P(y|do(x), z, w) = P(y|do(x), w) \tag{2.23}$$

- ***Rule 3.*** *(Addition or Removing of Actions) If* $Y \amalg Z|X, W$

$$P(y|do(x), do(z), w) = P(y|do(x), w) \tag{2.24}$$

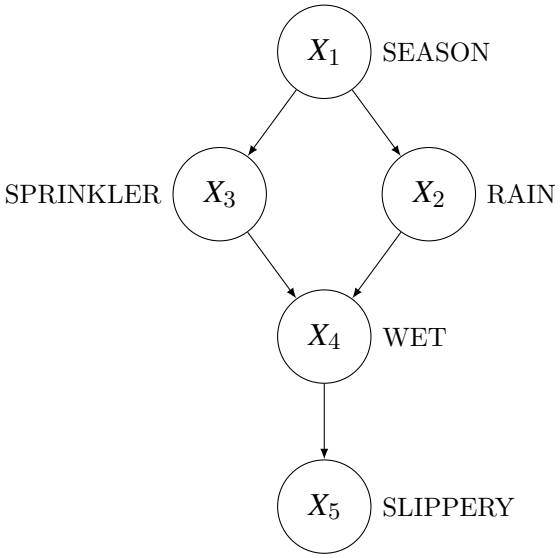


Figure 2.14: ***do-calculus example*, Figure 1.2, Pearl 2000**

$$P_{x_3=on}(x_1, x_2, x_4, x_5) = P(x_1)P(x_2|x_1)P(x_4|x_1) * P(x_4|x_2, X_3 = on)P(x_5|x_4) \tag{2.25}$$

This probability comes from Bayesian conditioning. It is an observation $X_3 = on$. After observing that the sprinkler is on, we wish to infer that the season is dry, that it probably did not rain, and so on; no such inferences should be drawn in evaluating the effects of a contemplated action "turning the sprinkler on.

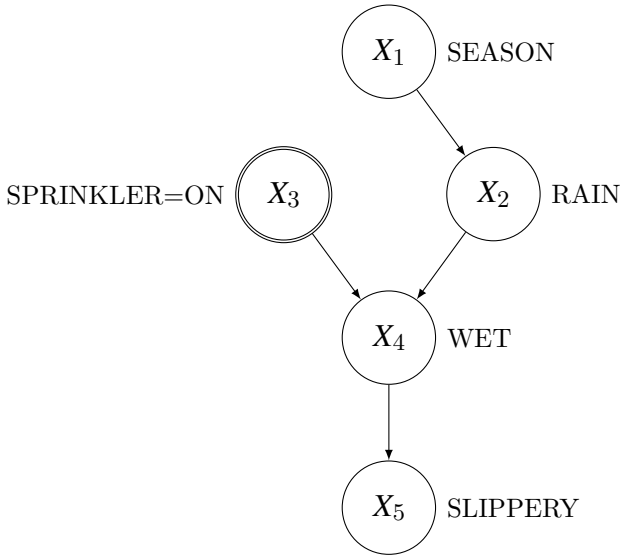


Figure 2.15: ***do-calculus example*, Figure 1.4, Pearl 2000**

$$P_M(y|do(x)) \tag{2.26}$$

This is the causal action, "turning the sprinkler On". The system must "respond to interventions in accordance with the principle of autonomy."

2.7 Learning Causal Models

In the early 1990s, researchers developed constrained algorithms such as IC [71] and PC [81] to learn the structure of causal graph models. Significant research advances have been made over the last few decades, allowing the structure and parameters of causal graph models to be learned from observational and interventional data.

First, we consider a situation where there are no confounding factors so that all relevant variables are observed in the data. We then discuss the challenges associated with latent variables and approaches for dealing with them. We finally

move to the task of learning in the presence of latent variables.

2.7.1 Learning Causal Models without Confounding Factors

In this stage, we approach the learning of Causal Bayesian networks from data. In this kind of learning systems, we suppose that the causal knowledge about all independence relationships is sufficient, that means no unmeasured hidden variables and no unmeasured selection variables [14]. The causal relationships are expressed via directed acyclic graphs. Its vertices are random variables and edges are direct causal effects. More precisely, A and B are vertices, if there is a directed edge from A to B that is, $A \rightarrow B$, it is expressed as A is a direct cause of B, and A is a parent of B. Else if there is a path from A to B, it is expressed that A is an indirect cause of B and A is an ancestor of B [67][68].

Definition 18. ***Causal Markov Condition (Assumption)*** *says that a phenomenon is conditionally independent of the others (non-descendants) when given its direct causes [26]. Therefore, a node is conditionally independent of the whole network when given its Markov blanket (see fig 2.8) [26].*

If the structure of a Bayesian network accurately depicts causality, the Markov (see fig 2.7 and the Causal Markov conditions are equivalent. However, a network can accurately include the Markov condition without representing causality, in this case, it should not be assumed to include the causal Markov condition. While the difference between the Markov and the Causal Markov conditions might appear purely syntactic, it is fundamental from a philosophical perspective [39]. The Markov conditions for Bayesian networks state properties that a particular distribution has. The causal Markov condition makes a statement about the world: If we relate variables by the causal relationship, these independence assumptions

will indicate the experimental distribution we observe in the world [39].

The well-known algorithm for the learning of causal structure without confounding factors is PC-algorithm, which is based on the current causal structure learning algorithms.

2.7.2 PC Algorithm

Peter Spirtes and Clark Glymour proposed a constrained based causal structure learning algorithm in 1990. In this algorithm, independence constraints are used to find out the causal structure by assuming that causal knowledge about all independence relations is sufficient [21]. Roughly, the PC-algorithm consists of two main steps. Step 1 starts with setting up the initial skeleton, which is a graph with a link between all its nodes. After that, the adjacencies between variables are searched. And then, all independencies according to the size of the conditioning set is decided [21]. Firstly, the pair of variables are checked, and independents edges are deleted. It continues until all possible relationships have been solved. Then, step 2 starts, which is a decisive step for the direction of edges. Firstly, unshielded triples are to subjected to a collider test. For example, A, B, C vertices, if A, B and B, C are dependent and B is not conditioning set for, the direction of edges are done as $A \rightarrow B \leftarrow C$, see example Figure-2.16. Otherwise, nothing is done.

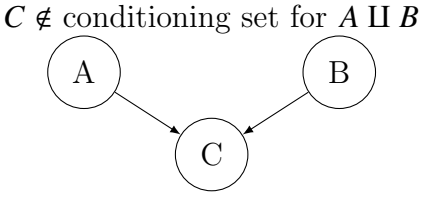


Figure 2.16: unshielded triples example

After that, the four rules that are proposed by Christopher Meek in 1995 are

applied for orienting edges in this way, as is shown in Figure-2.17;

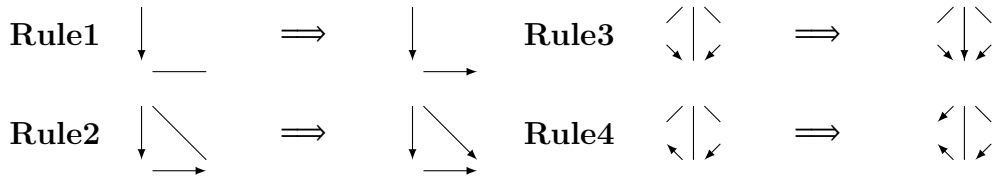


Figure 2.17: Meek [58] orientation rules

Although, this algorithm is asymptotically reliable and computationally efficient, it takes many risks in all procedure (causation prediction search). For example, to decide the undirected edge between two variables the process is to test every subset of the adjacency set of these two variables. However, the dependence or independence of these two variables might be completely irrelevant for causal relations (causation prediction search). It just provides a graph under assumption in the set of variables that are causally sufficient by ignoring the existence latent and selection variables.

2.7.3 Hidden Variables and Confounders

Hidden variables, also known as latent, are unobserved variables, as opposed to observable variables, that are not directly observed but are rather inferred from other variables that are observed such as Gravity fields, subatomic particles, antibodies [94]. Sometimes there was great evidence that hidden variables are real physical entities, such as quarks and sometimes is abstract like psychological stress. The data we can observe does not always provide all the information we need to model the system adequately we can use latent variables to give us more expressive power.

Selection variables are hidden variables that are in a specific state for each record in the observed data, such as a selection variable, for example, in a mail

survey, it can correspond to "the person who completed and sent the survey"; the presence of these variables can lead to biased results, since respondents to the survey may not represent the population as a whole.

The purpose of statistical analysis, usually done with hidden variables, is to reduce the dimensionality of the data [78]. Although this is a practical requirement in many cases, it is sometimes a challenging idea to discover the truth, especially when the truth concerns the causal relationship between hidden variables [78]. For example, there are several methods of achieving effective size reduction, assuming that the latent workings are independent. Since complete independence between random variables is a firm assumption, models derived from such methods may never correspond to exact causal mechanisms, even if such models fit the data good [78]. When the number of hidden variables is uncertain as to which variables measure them, Or, the researcher, who aims to make a causal explanation that the measured variables affect other measured variables, faces the most challenging and difficult discovery problem of the existing methods [78].

Hidden variables represent data that was not observed. We can, however, use a learning algorithm to determine the relationships between the observed variables and unobserved (latent) variables such as FCI and RFCI.

2.7.4 Learning Causal Models in the Presence of Confounding Factors

In this part, we will mention the graphical representations, including latent and selection variables. Therefore we need to sufficient representation for modelling hidden variables and uncertainty [93].

To understand causal structure learning algorithms including hidden variables, we need to give basic definitions and concepts. We introduce notation and terminology to describe independence models and graphs. Given $G = (V, E)$ is a graph

such that V is a set of vertices in the form of $\{V_1, ..., V_p\}$ and E is a set of edges. The vertices show random variables and edges show independence and ancestral relationships between variables [14]. In the structure learning part, differently from previous part, we will have many different type of edge style like;

- ***directed*** $\rightarrow$ shows direct cause
- ***bi-directed*** $\leftrightarrow$ shows two-way relationship between two vertices
- ***undirected*** $-$ shows there is latent variable
- ***non-directed*** $\circ\!\!-\!\!\circ$ shows arbitrary means could be any type of edge but data is not enough to make an assumption
- ***partially undirected*** $\circ\!\!-$
- ***partially directed*** $\circ\!\!\rightarrow$

$\circ$ symbol will be used to refer the arbitrary edge mark. As you know, directed graphs contains directed edges. If a graph can contain *directed, undirected and bi-directed* edge, it is said as ***mixed graph*** [14]. If there exists an edge between two variables, they are ***adjacent***. If all pair of variables are adjacent each other denoted by $adj(G, X_i)$, it is said that the graph is ***complete***. If there is and edge between X_i and X_j which are two random variables in G and they are adjacent, then it forms a ***cycle***. If there is a directed path from X_i to X_j and they have edge in between, then $X_i \rightarrow X_j$ forms ***directed cycle*** and $X_i \leftrightarrow X_j$ forms ***almost directed cycle*** [14]. If there is a directed path from X_i to X_j, it is said that X_i is an ***ancestor*** of X_j and X_j is an ***descendant*** of X_i.

Definition 19. ***Ancestral Graphs*** *(AGs) are mixed graphs used with three kinds of edges: directed edges* $\rightarrow$ *, bidirected edges* $\leftrightarrow$*, and undirected edges* $-$ *[99]. It is required to satisfy some additional constraints:*

- *If there is an edge from a vertex u to another vertex v, with an arrowhead at v that is* $u \rightarrow v$ *or* $u \leftrightarrow v$ *), then there does not exist a path from v to u consisting of undirected edges and/or directed edges oriented consistently with the path.*

- *If a vertex v is an endpoint of an undirected edge, then it is not also the endpoint of an edge with an arrowhead at v.*

Definition 20. *Suppose* p *is a path in a graph* G*,* V_x *is a vertex and non-end point on* p*. If the two edges intersect to* V_x *on* p *are both into* V_x*,* V_x *is a* ***collider*** *otherwise* ***non-collider*** *[99].*

Definition 21. ***Maximal Ancestral Graph*** *(MAG) is both an ancestral and maximal graph which has no any inducing path between any two non-adjacent nodes in the graph. It includes directed, undirected, and bi-directed edges [99], see example Figure-2.18.*

Example 2.

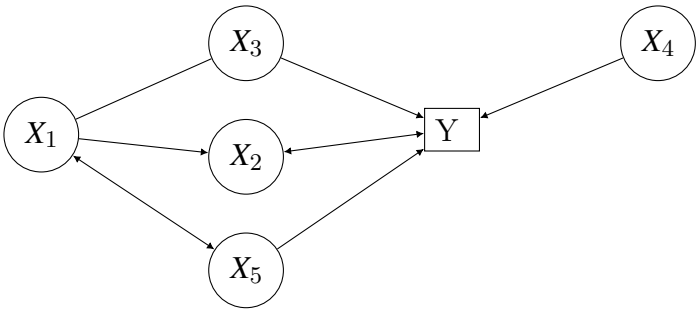


Figure 2.18: MAG example

Where X_1, X_2, X_3, X_4, X_5 are vertices and Y is a latent variable. MAGs allow to represent the directed acyclic graphs over a set of vertices including latent variables.

Definition 22. ***Partial Ancestral Graph*** *(PAG) represents a Markov equivalence class of MAGs. It includes directed, undirected, partially directed, and bi-directed edge [56], see example Figure-2.19.*

Example 3.

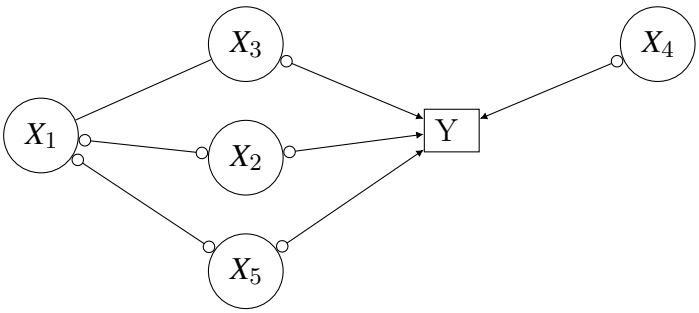


Figure 2.19: PAG example

Where X_1, X_2, X_3, X_4, X_5 are vertices and Y is a latent variable. We introduced PAGs in this part so the algorithms we will propose here search for a PAG. So, the PAGs we will consider (excluding the possibility of selection variables) can have the following edges: $\rightarrow$, $\circ\!\!\rightarrow$, $\circ\!\!-\!\!\circ$, and $\leftrightarrow$.

2.7.5 FCI Algorithm

In 1999, Spirtes et al. [83] proposed the Fast Causal Inference (FCI) algorithm. FCI is a modified and extended version of the PC algorithm, allowing arbitrarily hidden and selection variables [13]. It accepts the existence of hidden and selection variables and has been designed to show conditional independence and causal information between random variables [13].

As all constraint-based algorithms, FCI has two main parts which are independence test and orientation of edges. The FCI algorithm has the first part that is identical to the first part of PC [30]. The FCI starts with a complete undirected

graph and estimates the initial skeleton. The edges in the initial skeleton are presented as $\circ\!\!-\!\!\circ$ rather than a line $-$ contrary to the PC algorithm. The reason is that the subsets of adjacency the set of variables are no longer enough for deciding of dependencies between variables due to the existence of hidden variables. Therefore, we may have extra edges in the initial skeleton stage [30]. After, the orientation part, which begins with $v-structures$ check, is passed. The algorithm orients unshielded triples $A \bullet\!\!-\!\!\circ C \circ\!\!-\!\!\bullet B$ as $A \bullet\!\!\rightarrow C \leftarrow\!\!\bullet B$ if and only if C is not in conditioning set for $A \perp\!\!\!\perp B$ and $B \perp\!\!\!\perp A$.

After this stage, by comparing to the PC algorithm, the computational complexity of FCI starts to increase [14]. In this step, the Possible-D-SEP sets are calculated, which is defined as follow.

Definition 23. *Possible-D-SEP- Suppose G is a graph, which contains $\circ\!\!-\!\!\circ, \circ\!\!\rightarrow, \leftrightarrow$ edge styles and V_x is a vertex in G. Possible-D-SEP(V_x, G) function computes as follows: V_y vertex is in Possible-D-SEP(V_x, G), if and only if there is a path p between V_x and V_y in G such that for every subpath $< a, b, c >$ of p, b is a $collider$ on this subpath or it is a triangle in G [30].*

We denote obtained graph as G_1 and compute *Possible-D-SEP* sets. Next, we reorient all edges as $\circ\!\!-\!\!\circ$ and then update skeleton and information in separation sets.

Finally, orientation rules are applied for doing directed to many circles in the graph, which are proposed by Zhang in 2008 [100], see example Figure-2.20.

Example 4.

$1 \amalg 5|\{2,3,4\}$

$3 \notin adj(1)$,

$3 \notin adj(5)$,

L_1 and $L_2 \rightarrow$ latent

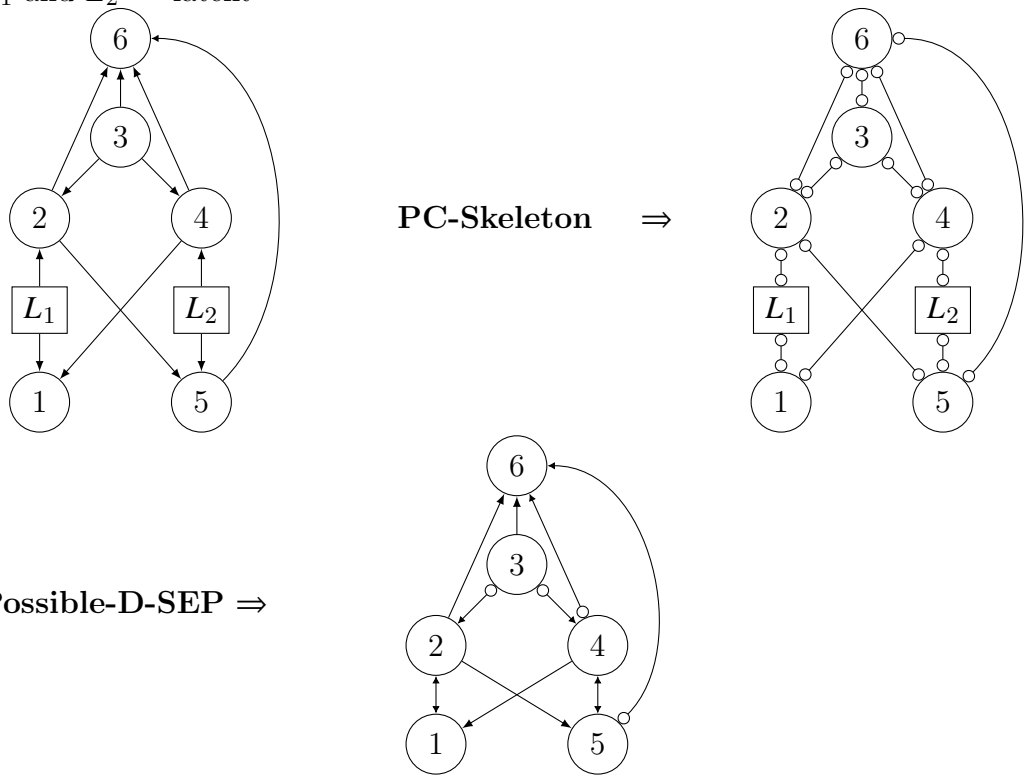


Figure 2.20: Possible-D-SEP example

After possible d-separation sets are computed for every variable in the graph, a conditional independence test is applied to decide whether an edge between two variables should be removed. Subsequently, *v-structures* are newly determined and oriented on the final skeleton. Finally, as many as possible, undetermined edge marks are determined using the ten orientation rules given by Zhang [100]. FCI has many stages and takes a big space for giving all steps. The algorithms proposed here use FCI for learning in the final part. We only gave an oracle version algorithm (Algorithm 1) here.

Algorithm 1 FCI algorithm (oracle version)

Require: Conditional independence information among all variables in variables $X_1, X_2, ..., X_r$

1: Form a complete graph G on the set of variables, where there is an edge ∘–∘ between each varibles pair.
2: Find first skeleton with independence tests and separations sets.
3: Orient unshielded triples in the skeleton based on separation sets.
4: Find Possible d separation sets as defined in definition 1 to find the final skeleton. Update graph and separation sets.
5: Orient unshielded triples in the skeleton based on separation sets.
6: Use rules ($R1$)-($R10$) of Zhang [100] to orient as many edge marks possible.
7: **return** PAG, P, conditional dependencies of $X_1, X_2, ..., X_r$

As is known, the structure learning for Bayesian Networks (BNs) is an NP-hard problem, therefore proposed algorithms for structure learning are optimisation algorithms. As in other optimisation algorithms, FCI has some deficiencies. It is computationally impossible to use the FCI algorithm when it has to work with large graphs [13]. For dealing with this problem, Colombo et al. propounded an algorithm named as the Really Fast Causal Inference (RFCI) in 2011 [13] by removing possible d-separation sets and adding some independence tests instead.

2.7.6 RFCI algorithm

RFCI algorithm is a modified version of FCI. FCI algorithm is weak for dealing with a graph have a large number of variables because of some process of learning structure. The RFCI algorithm is faster than FCI because it reduces conditional independence tests process. In RFCI, *Possible-D-SEP* sets are not computed; therefore, conditioning tests are reduced [29]. Colombo et al. modified the orientation rules for v-structures and discriminating path in this algorithm [29].
The first part of RFCI is identical FCI and PC. As it was mentioned, RFCI does not compute *Possible-D-SEP* sets. Contrary to FCI, RFCI uses a different method

and adds additional tests before the orientation part to ensure soundness [12]. The first of these additional tests are as follows;

Definition 24. *(Unshielded triple rule). Suppose given a graph $G = (V, E)$ is composed of a set of vertices $V = X \bigcup L \bigcup S$ such that L, S and E are latent, selection variables and a set of edges, respectively. Suppose that* ***(i)*** *S_{ik} is a minimal separating set for X_i and X_j when given S,* ***(ii)*** *X_i and X_j are conditionally dependent when given $(S_{ik} \backslash \{X_j\}) \bigcup S$. Then X_i is in ancestor subset X_i and X_k if and only if X_j is in S_{ik} [30].*

On the other hand, it may be detected a relationship like $X \amalg Y$ when given separation set for X and Z. This situation may occur when X and Y are not d-separated when given adjacency subsets of X and Y respectively but may not be m-separated when adjacency subset of X and Z [12]. At that case, the edge between X and Y are removed. Nevertheless, this edge removing step can cause new unshielded triples [30]. For solving this problem, it is worked with lists defined in detail in the study of Colombo et al. in 2012 [14].

Before passing the orientation part we have one more conditional independence test part.

Lemma 1. ***Discriminating path rule*** *(taken from [13] and it will be new rule* ***iv****). Let the distribution of $V = X \bigcup L \bigcup S$ be faithful to a DAG G. Let $\Pi_{ik} =< X_i, ..., X_l, X_j, X_k >$ be a sequence of at least four vertices that satisfy: $(a1)$ X_i and X_k are conditionally independent given $S_{ik} \bigcup S$, $(a2)$ any two successive vertices X_h and X_{h+1} on Π_{ik} are conditionally dependent given $(Y' \backslash \{X_h, X_{h+1}\}) \bigcup S$ for all $Y' \subseteq S_{ik}$, $(a3)$ all vertices X_h between X_i and X_j (not including X_i and X_j satisfy $X_h \in an(G, X_k)$ and $X_h \notin an(G, \{X_{h-1}, X_{h+1}\} \bigcup S)$, where X_{h-1} and X_{h+1} denote the vertices adjacent to X_h on Π_{ik}. Then the following hold: $(b1)$ if $X_j \in S_{ik}$, then $X_j \in an(G, X_k \bigcup S)$ and $X_k \notin an(G, X_j \bigcup S)$, and $(b2)$ if $X_j \notin S_{ik}$, then $X_j \notin an(G, X_l, X_k \bigcup S)$ and $X_k \notin an(G, X_j \bigcup S)$.*

For each triple $a <\!\!-\!\!\bullet\ b\ \circ\!\!-\!\!\bullet\ c$ with $a- > c$, it is searched for a discriminating path $p =< d, ..., a, b, c >$ for b of minimal length, and controlled vertices in every pair on p are conditionally dependent when given all subsets of separation set for d and c. If there is no conditional independence relationships, the path $a <\!\!-\!\!\bullet\ b\ \circ\!\!-\!\!\bullet\ c$ is oriented as in rule (R4) proposed by Zhang in 2008 [100]. Otherwise, if there is one or more conditional independence relationships, matching edges are removed and their d-separation sets are held. RFCI uses fewer conditional independence tests than FCI, and its tests condition on a smaller number of variables [13]. As a result, RFCI is much faster than FCI and its output tends to be more reliable for small samples, since conditional independence tests of high order have low power [13]. In some situations the output of RFCI is slightly less informative, in particular with respect to conditional independence information [13].

We are inspired by RFCI and computed the computational cost of the possible d-separations of the algorithms. Then we analysed that if we remove the possible d-separations from the algorithms, how much our graphs will be less informative. So we presented a version (RFOFCI) which does not include the possible d-separations.

2.8 Online Learning

This section includes the main contributions to online learning algorithms for BNs developed up to now. All these algorithms aim to change or evolve what is already known structure when new data incomes. Nevertheless, there is no recognition of what is considered to be an online algorithm in the field of learning BNs, nor is there a widely accepted definition of these algorithms. [73].

Until now, we had supposed we have stationary domains. After this point, we are going to touch on data which are changing over time because this conjecture

is not reasonable in real-world online applications. For example, unobservable impacts may suddenly change, or unknown events may arise.

Online learning is a learning paradigm in which the training data points are placed in an ordered sequence [8]. The current model is quickly updated to generate the best model up to now when a new data point incomes. Online learning has the same aim as classic structure learning, and it also aims to optimise the performance of the given learning task [8]. Batch learning is costly if a new data point arrives, and if all the available data are used again. Therefore, it causes unnecessary memory and running time efficient and is not suitable for a real-world scenario. For example, predicting stock market trends and weather forecasting is real-world scenario samples of sequential prediction problems. Additionally, unlike statistical machine learning, these algorithms do not make stochastic assumptions about the observed data.

Nevertheless, Online Structure Learning for Probabilistic Graphical models is not a straightforward task because the new data is not always following the learned model from the previous data. Therefore, it needs to be modified such as with weights. One solution to this problem is to be weighted more heavily to the new data points. So the poor fit between a new data point and the current model may indicate that the structure is changed. These weights given are determined by the distance between the new data and the current model.

Why we are studying with Online Machine Learning approach;

- In a growing number of machine learning applications, one must make online, real-time decisions and continuously improve performance with the sequential incoming data.

- The number of conditional independence tests performed by the algorithm grows exponentially with the number of variables in the dataset. This affects both the speed and the accuracy of the algorithm on small samples

so conditional independence tests on large numbers of variables have little power.

- A trial and error approach often leads to too large a test problem to handle computationally, and online learning is a powerful and popular way of dealing with problems.

- An online learning algorithm monitors a sample stream and makes a forecast for each item in the stream

- The algorithm immediately gets feedback about each estimate and uses it to increase the accuracy on subsequent feedbacks.

- Maintaining the privacy of sensitive data that different parties have is often a critical question. However, in many practical applications, BNs require a gradual acquisition of data at different timescales, where conventional collective learning algorithms are not appropriate or not implemented [74].

- The goal is to fix with the functionality of online learning so that the next iteration of production contains fewer or less serious inferential errors [17].

An online learning algorithm should meet some constraints which are a short fixed time per recording is required, it can create a model using up to one data scan, and regardless of the total number of records shown, a fixed amount of main memory must be used [73]. More simply and clearly, learning algorithms to use the information in the learning process whenever they want.

2.8.1 Online Parameter Learning

In previous offline learning part, the learning parameter for BNs was explained. In this part, it will be exemplified online versions of some well-known parameter

learning algorithms. We will start with a popular one of methods for learning parameters, called Expectation Maximisation.

1977, Dempster et. al proposed ***Expectation Maximization*** (EM) algorithm [19]. The EM algorithm is a recursive method to detect the maximum probability or maximum posterior that estimates parameters in statistical models where the model is dependent on unobserved hidden variables. It has two steps called E and M symbolising the first letters of *Expectation Maximization*. In step E, a function is created for the expectation of log-likelihood. In M step, parameters are computed for maximising the expected log-likelihood found on the E step.

EM tries to find parameters $\hat{\Theta}$ that maximise log probability $\log P(x;\Theta)$ of observed data. The advantages of EM are its simplicity and ease of implementation. It works efficiently when we have fewer missing values and works with data sets which have not too many variables [64]. Bilmess propounded a study on EM algorithm and its application to Hidden Markov Models (HMM) in 1998 [4]. In 2008, Mongillo et. al has developed an online version of this method [60]. Another important online parameter learning algorithm for HMM are propounded by Feti et. al in 2014, called ***Stochastic variational inference (SVI)*** which is a mini-batch based variational Bayes method [22]. Another algorithm proposed by Omar in 2016 can also be given as the most recent example. The ***Moment Matching Algorithm*** which is an online technique which means that updating the model parameters requires a certain amount of time after each new observation is taken [64].

2.8.2 Online Score-based Approaches

The earliest studies on online structure learning for BNs are ***Buntine's approach*** (1991) [5], ***Lam and Bacchus's approach*** (1994) [47], ***Friedman and Goldszmidth's approach*** (1997) [23]. It is enough to explain the first two

approach because other algorithms follow the same idea.

Buntine's approach is a generalization of the ***K2*** algorithm [98]. In the study of Buntine [5], any empirical assessment to verify the effectiveness of the approach are not provided. The approach used the score-based Bayesian network structure learning technique; therefore, the algorithm has limitation and not scalable for high-dimensional domains.

In Lam and Bacchus's approach, firstly a partial network is learned and then the *Minimum description length* principle is used to refine the existing network. It is also scored based method and has the same limitation as to other score-based methods.

All these algorithms share the same idea. They are setting up a network with what they have seen so far, and the learning algorithm is triggered when new data arrives. Then, the algorithm searches for the current network location.

2.8.3 Online Constraint-based Approaches

Unfortunately, there is not enough source on online algorithm uses constraint-based Bayesian network structure learning techniques. Some of the existing sources assume that there is no latent variable. The critical problem with learning cause and effect from observational (as opposed to interventional) data is the presence of hidden confounders. In practice, it is difficult to know whether all confounders have been taken into account. So such methods should be used only as guides for identifying possible cause and effect. Therefore, an algorithm uses constraint-based Bayesian network structure learning techniques and takes into account latent, and selection variables would give much more effective outputs.

As we mentioned, there are a few sources in literature in this way.

In 2012, Kummerfeld and Danks proposed an algorithm that is ***Dynamic Online Causal Learning*** (DOCL) [46]. It is a new causal structure learning

algorithm and processes data in a dynamic and real-time functioning [46]. This algorithm is divided into three parts which are the *Online Covariance Matrix Estimator* (OCME), the *Causal Model Change Detector* (CMCD) and the *Causal Model Learner* (CML).

The OCME takes each data point in sequence and predicts a covariance matrix to provide "raw materials" to learn to be causal [46]. The CMCD monitors the difference between the new data points and the predicted covariance matrix to discover changes in the environment or significant errors. This information is then used to adjust the weights on the previous data points [46]. The CML takes the covariance matrix and learns the causal model at that point in time [46]. We will give in detail these parts in proposed algorithms chapter. This relationships are expressed with following diagram, see example Figure-2.21.

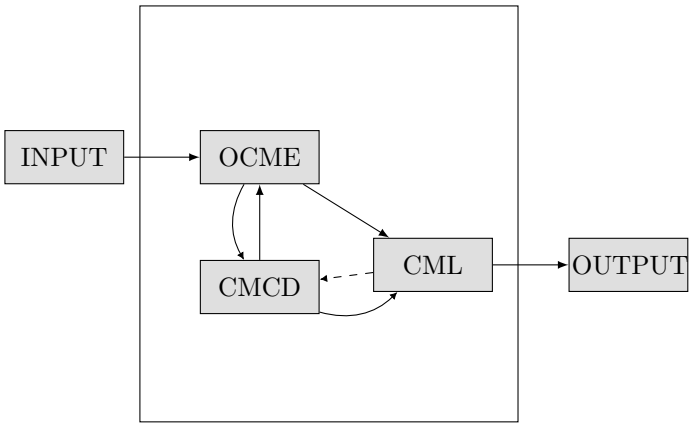


Figure 2.21: ***DOCL*** algorithm diagram

In 2013, Kummerfeld and Danks again proposed a similar algorithm, called as ***Locally Stationary Structure Tracker*** (LoSST) algorithm [45]. It also uses constraint-based Bayesian network structure learning techniques. Working process is so similar to *DOCL*. In *DOCL*, they assume that the data are generated independently of the underlying causal model, however, they do not assume that this causal model is stationary over time [46]. In contrast to *DOCL*, in*LoSST*, they

assume only that the generating process is locally stationary [45]. Both algorithm are compared with PC algorithm and got more effective results.

2.9 The Contributions and Limitations in the Related Work

Bayesian network structure learning has attracted great attention in recent years. In a nutshell, structure learning can be defined as finding DAGs, PDAGs, or PAGs that fit the data. Our main study is on causal models which is represented by PAGs, but importantly not that not all algorithms assume that the direction of the edges represent causation.

The critical point for the current research is the assumption that both of the causal structure learning approaches assume that the data comes from a single generating causal structure, and therefore these methods cannot be used directly for learning when a causal structure changes during the data collection process. Both types of approaches are not able to keep up with systems in a developing and changing world. Therefore, we need new tools to handle this, which are capable of giving results in a reasonable amount of time. Nevertheless, they only require sufficient statistics as input data and can, therefore, provide part of the solution to this problem. They need a mechanism which can detect changes, respond to it, and then learn the new causal model.

In the literature, there exist two main methods for online tracking of some feature in a structure, which are temporal difference learning (TDL) [90] and Bayesian change-point detection (CPD) [1]. Nevertheless, both methods have not been applied to detect changes in a causal structure, so they need some modifications to do this.

The standard TDL algorithm provides a dynamic estimate of a univariate ran-

dom variable using a simple update rule. In this update, the error in the current estimate is updated with a learning rate coefficient. Therefore, the static learning rate plays an important role and controls how quickly or slowly, a model learns a problem. If it is chosen too small, the TDL algorithm converges slowly. If it is chosen too large, the algorithm will be so sensitive even when the environment is stable. TDL algorithm can detect slow change but not high stability or dramatic changes. That feature is essential for causal structure learning as causal structures often have non-deterministic connections.

In contrast, CPD algorithms are useful for dramatic changes that indicate breaks between periods of stability [1]. CPD algorithms must store large parts of input data. These algorithms assume that the model only has a dramatically changing environment separated by periods of stability. Both algorithm types have not been applied to tracking causal structure. To do so, they need the necessary modifications.

Talih and Hengartner [55] do other related work. In their work, the data sets are taken sequentially as input and divided into a fixed number of data intervals, each with an associated undirected graph that differentiates one edge from its neighbours. In contrast to our work, in their work, they focus on a particular type of graphical structure change (a single edge added or removed), only work in batch mode and use undirected graphs instead of directed acyclic graphical models. Next, Siracusa [79] uses a Bayesian approach to find posterior uncertainty on possible directed edges at different points in a time series. Our work differs from their work because we use frequentist methods instead of Bayesian methods, and we can work in real-time in an incoming data stream.

Some methods aim to estimate the time-varying causal model. The DOCL algorithm proposed by Kummerfeld and Danks is applied to tracking causal structure [46]. They demonstrated the adequate performance of algorithms in tracking

changes in structure. It is important to note that we use their algorithm for change detection. Therefore, their work is the source of inspiration for our work.

On the other hand, the work of Kummerfeld and Danks [46] differs from ours in two ways. First of all, DOCL does not allow for the possibility of latent and selection variables. However, the critical problem with learning cause and effect from observational (as opposed to interventional) data is the presence of hidden confounders. Next, DOCL runs the learning algorithm whenever there is a change in the structure, and this learned graph is used only to display the changing relationships between variables. However, in our algorithm, the relationships in the learned graphs are saved for the next change point and take an active part in learning the next change point.

Another related method is proposed by Bendtsen [3], which is the regime aware learning algorithm to learn a sequence of Bayesian networks that model a system with regime changes. These methods are not able to cope with real-world data as they suffer from a large number of statistical tests and ignore the existence of confounding factors. We present a new approach which is capable of detecting changes even multiple times and learning structure in the light of sequentially incoming data in the presence of confounding factors.

Chapter 3

PROPOSED ALGORITHMS

In this chapter, we will propose three novel algorithms which can track changes in a causal structure and process data in a dynamic real-time manner. In a nutshell, these three algorithms take sequentially data as input, update covariance matrix in the light of sequentially incoming data and detect fitness between structure and data, and finally outputs PAGs when detecting outliers between data and structure. Until the algorithms detect outliers, they work same and then they differentiate after from this point.

Therefore, we will give first the joint part of three algorithms and then pass to algorithms description.

3.1 Problem Definition

Given a set of continuous variables V, we assume that we have a true underlying causal model over V at each moment in time. We specify a causal model by a pair $\langle G, F \rangle$, where G denotes a DAG over V, and F is a set of linear equations. These kinds of causal models are also known as recursive Causal Structural Equation Models (SEMs) [46]. We assume that the data are independently generated from

the true underlying causal model at each moment in time, though we do not assume that this causal model is stationary through time.

In a nutshell, the online causal structure learning algorithms proposed here take a new data point as input at each time step and outputs a graphical model (PAG). We have separated the proposed algorithm into three distinct parts which are online covariance matrix estimation (OCME), causal model change detection (CMCD) and causal structure learner (CML) parts, respectively. OCME and CMCD parts are identical for three algorithms; therefore, we will start with these parts and then will give algorithms in CML part. We had mentioned shortly OCME, CMCD and CML parts in the previous section.

Table 3.1: Online Algorithms Module Details

Sum of weights:	$b_r = \sum_{k=1}^{r} a_k$
Weighted mean:	$\mu_i^r = \sum_{k=1}^{r} \frac{a_k}{b_r} X_i^k$
Updating the mean:	$\mu_i^{r+1} = \frac{b_r}{b_{r+1}} \mu_i^r + \frac{a_{r+1}}{b_{r+1}} X_i^{r+1}$
Weighted covariance:	$C_{V_i,V_j}^r = \sum_{k=1}^{r} \frac{a_k}{b_r} (X_i^r - \mu_i^r)(X_j^r - \mu_j^r)$
Updating the Weighted covariance:	$C_{X_i,X_j}^{r+1} = \frac{1}{b_{r+1}} = [b_r C_{X_i,X_j}^r + b_r \delta_i \delta_j ...$ $... + a_{r+1}(X_i^{r+1} - \mu_i^{r+1})(X_j^{r+1} \mu_j^{r+1})]$
Correction term:	$\delta_i = \mu_i^{r+1} - \mu_i^r = \frac{a_{r+1}}{b_{r+1}} (X_i^{r+1} - \mu_i^r)$
Mahalanobis distance:	$D_r = (X^r - \vec{\mu})(C^r)^{-1}(X^r - \vec{\mu})^T$
p-value of the distance:	$T^2(x > D_{r+1} \mid p = N, m = S^r - 1)$
Weighted pooled p-value:	$\rho_r = \Phi(\sum_{i=1}^{r} a_i \Phi^{-1}(p_i, 1), \sqrt{\sum a_i^2})$
Weight for next data point:	$a_{r+1} = \begin{cases} a_r, & if\ \rho_r \geq T \\ \frac{a_r \gamma T}{\gamma T + \rho_r - T}, & otherwise \end{cases}$

The table 3.1 which shows all notations and the set of linear equations and next we will explain OCME, CMCD and CML parts in detail.

3.2 OCME

Kummerfeld and Danks propose the Online Covariance Matrix Estimator (OCME) in 2012 [46], which is the first stop of the algorithm. OCME starts with sequentially taking each datapoint that is available online as input [45, 46]. Algorithms have a "burn-in" period. Burning is intended to give the algorithms time to achieve equilibrium distribution. Typically, the first examples are not fully valid, so they are not enough to give an idea us about the structure. The burn-in samples allow us to discard these first samples that are not yet stationary. Burn-in is a colloquial term we used that describes the practice of throwing away some iterations at the beginning of the online algorithms will be proposed here. We determined the length of the burn-in period with 10 data samples, but any number of data points can be specified as a starting covariance matrix C over the variables V. During the burn-in period, the individual p-values for each data point is 0.05. Let a_r be the weight on the r^{th} datapoint is 1.

As OCME does not store any of the incoming new data points, its memory needs only $O(N^2)$ for the estimated covariance matrix,where N is the number of variables. In batch algorithms, this memory is $O(NM+N^2)$, where M is the sample size. That is the batch mode algorithms require the memory both all data-samples and the estimated covariance matrix [45, 46]. Thus, the proposed algorithms have a substantial memory advantage compared to batch mode learning algorithms.

In particular, let X^r be r^{th} multivariate datapoint and let X_i^r be the value of V_i for that data point. The data points are weighted distinctly for tracking possible structure change. As we do not assume a stationary causal model, the data points should be weighted differently in a way to weight more recent datapoints more heavily after a change occurs and reduce confidence in previous data points. These weights are determined by the CMCD part given in detail next part depending on the distance between the new data point and the current model.

Let a_r be the weight on the r^{th} datapoint where $a_i \in (0, \infty)$ and let $b_r = \sum_{k=1}^{r} a_k$ be the sum of weights on the each datapoint. The weighted average of V_i after datapoint r is:

$$\mu_i^r = \sum_{k=1}^{r} \frac{a_k}{b_r} X_i^k \tag{3.1}$$

As OCME is an online estimation method, Kummerfeld and Danks [46] translated it into means update equation:

$$\mu_i^{r+1} = \frac{b_r}{b_{r+1}} \mu_i^r + \frac{a_{r+1}}{b_{r+1}} X_i^{r+1} \tag{3.2}$$

The weighted covariance between V_i and V_j after datapoint r is computed with:

$$\mathrm{C}_{V_i,V_j}^r = \sum_{k=1}^{r} \frac{a_k}{b_r} (X_i^r - \mu_i^r)(X_j^r - \mu_j^r) \tag{3.3}$$

As OCME is an online estimation method, Kummerfeld and Danks [46] translated it into an update equation. The covariance matrix update equation as online is:

$$\mathrm{C}_{X_i,X_j}^{r+1} = \frac{1}{b_{r+1}} = [b_r \mathrm{C}_{X_i,X_j}^r + b_r \delta_i \delta_j + a_{r+1}(X_i^{r+1} - \mu_i^{r+1})(X_j^{r+1} \mu_j^{r+1})] \tag{3.4}$$

where $\delta_i = \mu_i^{r+1} - \mu_i^r = \frac{a_{r+1}}{b_{r+1}}(X_i^{r+1} - \mu_i^r)$. If $a_k = c$ for all k and some constant $c>0$, then the estimated covariance matrix using this method is identical to the batch mode estimated covariance matrix.

If $a_r = \alpha b_r$, then the algorithm acts like $TD(0)$ [90] learning for each covariance with a learning rate of α.

Definition 25. *The classic Temporal difference learning (TDL) algorithm TD(0) [90], provides a dynamic estimate $\mu_t(X)$ of a univariate random variable X using*

a simple update rule:

$$E_{t+1}(X) \leftarrow (1-\alpha)E_t(X) + \alpha X_t \tag{3.5}$$

where E is expected value (mean), X_t is the value X at time t. That is, one updates the estimate by α times the error in the current estimate.

Therefore, the fitness between the new incoming data and covariance matrix determines their weight in an estimated covariance matrix update equation. If the covariance matrix is fitting the data, a similar weight is used to preserve the structure of the covariance matrix. If it is not fitting, the weight of the recent datapoint is weighted more heavily after a change occurs and reduce confidence in previous data, so this indicates that the structure has changed.

Any causal structure learning algorithm needs a sample size with an estimated covariance matrix. In the proposed algorithms, we also need to update the sample size in the light of incoming data point and its weight. Various data points can receive different weights. We compute sample size, which is called an effective sample size [46], by adjusting the previous the effective sample size based on datapoint's relative weight. Therefore, we first need to update the learning rate to track the weighted sum of mean error values. And then, we need to compare this against a distribution which depends on the effective sample size and data size. We assume that every new data point contributes 1 to the sample size, and the effective sample size is updated according to weight.

More specifically, let S^r be the effective sample size at time r. We assume the incoming datapoint contributes 1 to the effective sample size and update accordingly:

$$S^{r+1} = \frac{a_r}{a_{r+1}} S^r \tag{3.6}$$

The causal model change detector (CMCD) detects whether the structure has changed or not by looking at the fitness of the current structure with the incoming data and give weight accordingly. If there is unfitness, it indicates that the structure has changed and this is a change in the new incoming data direction. So, CMCD assures that $a_{r+1} > a_r$ for all r. Therefore, the effective sample size does not necessarily have to be equal to the true sample size (which should be less than or equal to the actual sample size). If the structure is not changed, that means datapoints weights are constant, and then the effective sample size equals the true sample size.

Therefore, we no longer need to remember previous data points, so OCME provides us with the sufficient statistics μ^{r+1}, C^{r+1}, and S^{r+1} (covariance matrix, sample size and mean). They are enough to remember for information about previous data points.

3.3 CMCD

Kummerfeld and Danks propose the Causal Model Change Detector (CMCD) in 2012 [46]. In OCME, the fitness between the current estimated covariance matrix and the input data to detect the changes in the underlying causal model is tracked by the CMCD for adjusting to the previous and new datapoints' relative weight [45, 46]. The Mahalanobis distance [53] gives the fit between each incoming data point and the current estimated covariance matrix.

More precisely, as we assumed that the data has a multivariate Gaussian distribution, the *mahalanobis distance* between the incoming datapoint X^r and the current estimated covariance matrix C^r with the current estimate of the means $\vec{\mu}$

is given by Mahalanobis distance D_r [53]:

$$D_r = (X^r - \vec{\mu})(\mathrm{C}^r)^{-1}(X^r - \vec{\mu})^T \tag{3.7}$$

A large Mahalanobis distance for any particular data point can merely indicate an outlier; consistently large Mahalanobis distances over multiple datapoints state that the current estimated covariance matrix fits poorly to the underlying causal model. Therefore, the new data points should be weighted more heavily [45, 46].

The approach is to first calculate the individual p-values for each data point for Mahalanobis distance. Next, a weighted pooling method to aggregate those each p-values into a pooled p-value by using Liptak's method [51] is used.

More precisely, the Mahalanobis distance of a $V-$dimensional datapoint from a covariance matrix estimated from a sample of size N is distributed T^2. The $p-$value for the Mahalanobis distance D_{r+1}

$$p_{r+1} = T^2(x > D_{r+1} | p = N, m = S^r - 1) \tag{3.8}$$

where T^2 is Hotelling's $T-$squared distribution, S^r is the effective sample size and $p = V$ and $m = N - 1$ are parameters.

A big Mahalanobis distance could indicate an outlier. The several large Mahalanobis distances signify that the current estimated covariance matrix has a poor fit to the underlying causal model, so new data points are required to be weighted more heavily. These $p-$values establish the likelihood of X^r given $\vec{\mu}$ and C^r, but what we need is the likelihood of the weighted data points X incoming sequentially given $\vec{\mu}$ and C^r. The distribution of a sum of weighted chi-square variables D_r is analytically intractable, and so we cannot use the D_r values directly. Instead, Liptak's method [51] for weighted pooling of individual $p-$values is used. This method, also known as weighted Z-test, was generalised by Liptak (1958) to give

different weights to each study according to their power.

Let $\Phi(x, y)$ be the cumulative distribution function of a Gaussian with mean 0 and variance y evaluated at x. Then the pooled, weighted $p-$value is:

$$\rho_r = \Phi(\sum_{i=1}^{r} a_i \Phi^{-1}(p_i, 1), \sqrt{\sum a_i^2}) \tag{3.9}$$

Finally we need to determine the weight of the next point a_{r+1} given the the pooled $p-$value ρ_r. There are many ways to convert the pooled $p-$value ρ_r into a weight a_{r+1}. The simple strategy is used here is: if ρ_r is greater than some threshold T (i.e., the data sequence is sufficiently likely given the current model), then keep the weight constant; if ρ_r is less than T, then increase a_{r+1} linearly and inversely reducing the weight of all previous datapoints by some constant factor. Mathematically, this information is:

$$a_{r+1} = \begin{cases} a_r, & if\ \rho_r \geq T \\ \frac{a_r \gamma T}{\gamma T + \rho_r - T}, & otherwise \end{cases} \tag{3.10}$$

3.4 CML

The parts up until now are to track the changes in structure. The last part is the Causal Model Learner (CML) [46] which learns the causal model from the estimated (from weighted data) sufficient statistics (covariances, sample size and means) provided in OCME. Kummerfeld and Danks's [46] algorithm uses the PC algorithm [81] as a standard constraint-based causal structure learning algorithm. Instead, alternative causal structure learning algorithms could be used. Therefore, a system that can detect this process before learning is required. In this point, CMCD comes into play and tracks changes in the structure in the light of sequentially incoming data. When CMCD detects changes in the environments,

the causal model structure learning algorithm learns the structure.

Additionally, Kummerfeld and Danks developed a probabilistic re-learning scheduler which utilises the pooled $p-$values calculated by the CMCD module to determine when to re-learn the causal graph [45, 46]. In Kummerfeld and Danks works [45, 46], CML uses PC algorithm [84]. By contrast them, CML uses FCI and two different modified FCI algorithms instead of PC [84] in three algorithms proposed. By proposing these three algorithms, we are aiming for three things. First algorithm OFCI we will propose to allow us to learn in the presence of latent variables. However, when OFCI detects changes and makes new learning, the previous model is ignored and learned again. Second algorithm FOFCI we will propose, it is also the most important algorithm of this study, allow us to use previously learned structure while learning new structure. When FOFCI detects changes and makes new learning, the previous model is saved and used to reduce the cost for learning the edges that do not change in the new model. Third algorithm RFOFCI we will propose, it is an alternative algorithm to the second algorithm, allow us a fast causal structure learning algorithm to deal with large networks. RFOFCI ignores some independence tests to make learning faster. However, it's outputs mostly are less convergent (informative) to the true graph than OFCI and FOFCI.

3.5 Online Fast Causal Inference (OFCI)

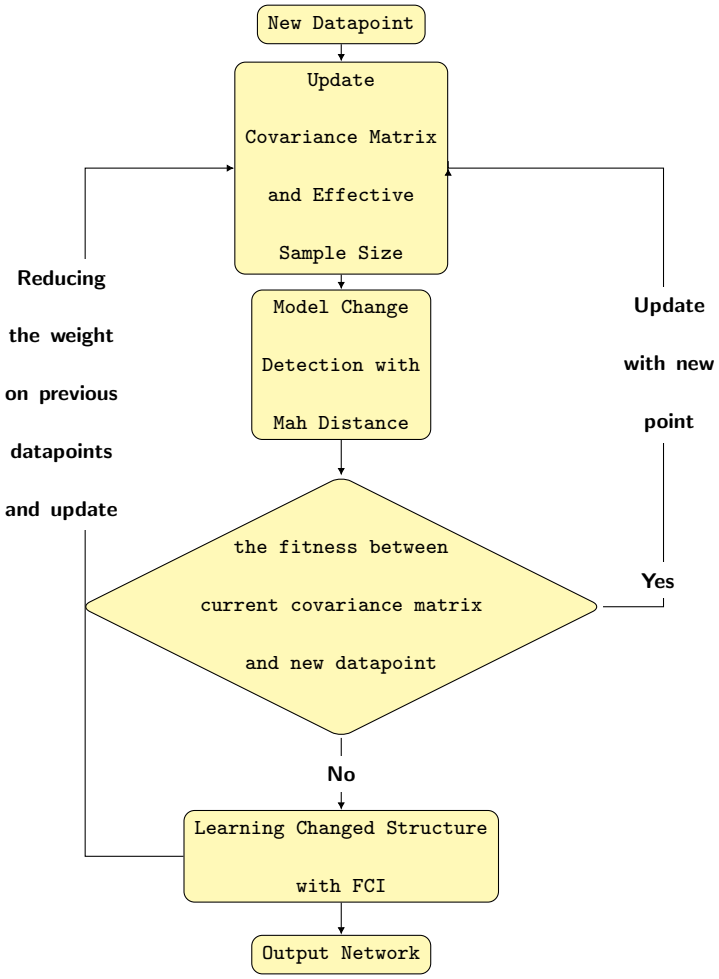
In Kummerfeld and Danks's work, OCME takes the new data point sequentially as input and estimates current covariance matrix with weights determined by CMCD section by looking distance the new data point and current covariance matrix. A final, if the algorithm detects an anomaly then CML learns the structure with the PC algorithm. PC algorithm assumes that there are no unmeasured

confounders and the true causal structure is known. Unfortunately, in almost all real-world problems the actual underlying causal structure is not known. Spirtes et al. [86] proved that the FCI algorithm is sound in the presence of arbitrarily many latent variables. Spirtes et al. [80] extended the soundness proof to allow for selection variables as well. Therefore they proposed an algorithm which is a suitable study for solving real-world problems. In contrast with the method of Kummerfeld and Danks's work, CML uses FCI instead of PC in OFCI. Just like Kummerfeld and Danks's work, OFCI re-learns the causal model after a change occurs. In Kummerfeld and Danks work [46], CML uses the effective sample size.

In Kummerfeld and Danks's work, in all stages of the algorithm is used the effective sample size including the PC algorithm. In OFCI, we replaced the PC algorithm with FCI and we obtained better results for actual sample size rather than the effective sample size. However, the actual sample size is just used for the FCI algorithm, other states continue to use the effective sample size like Kummerfeld and Danks's work.

In a nutshell, OCME sequentially takes each datapoint as input and then updates the estimated covariance matrix to provide inputs for the structure learning algorithm. Although the algorithm appears to have three distinct components, OCME and CMCD work simultaneously. OCME updates estimated covariance matrix and effective sample size in the light of the inputs provided by CMCD. This simple Figure 3.1 represents a process of OFCI.

Figure 3.1: Basic flowchart of OFCI algorithm



3.6 Fast Online Fast Causal Inference (FOFCI)

Kummerfeld and Danks's work and OFCI algorithms detect and respond to change points but their work learns the structure with updated covariance matrix from scratch. Although it is essential to identify these change points, re-learning from scratch each time increases the cost because we know some parts will be stable after changes. So if we only search for the changing parts of the model, then learn model in the light of changes may not meet some constraints to say online learning.

Domingos and Hulten stated desirable properties of online learning [20]. It states that we may able to gather new data every day and that it would be interesting to revise the current model in the light of this new data without spending an unreasonable amount of time and memory [20]. FOFCI allow us to revise the current model in light of this new data without spending an unreasonable amount of time and memory.

Learning graphical model structure is computationally expensive, and so one should balance the accuracy of the current model against the computational cost of re-learning. For these reasons, we propose FOFCI to reduce the computational cost of re-learning and make the proposed algorithms more online [35]. Although OFCI is an online algorithm and does active learning, it ignores all of its previous outputs. According to us, to speak of a real online learning mechanism, all parts of the algorithm must be actively involved in learning at every change point. That makes the algorithm more online. The FOFCI algorithm differs from both OFCI and Kummerfeld and Danks's work [46] for causal model learning the part. In those two algorithms, the OCME and CMCD parts continue to update sufficient statistics as long as only the new data point is available. However, the CML part has no other role than to learn the updated information.

In contrast, FOFCI uses a modified version of the FCI algorithm. We modified the FCI algorithm in a way to use the information of the previous model while learning a new model to reduce learning complexity. Modified FCI has three main parts which are the check of prior model information, independence test and orientation of edges.

Definition 26. *The **Modified FCI** takes the sepset of the prior model as input. If conditional independence is found between two variables, an edge between them, and the set responsible for this conditional independence is saved in sepset. The separation set is the set that carries the information of the skeleton of a model.*

Therefore, sepset is vital in examining the joint parts between the new covariance matrix and the current model. The algorithm starts with the analysing of the sepset of the previous model. First, the algorithm makes independence test for every subset in separation set. Therefore, the modified FCI starts with a sparse graph, rather than a complete graph, compared to the classic FCI. The rest of this algorithm is identical to the classic FCI [81].

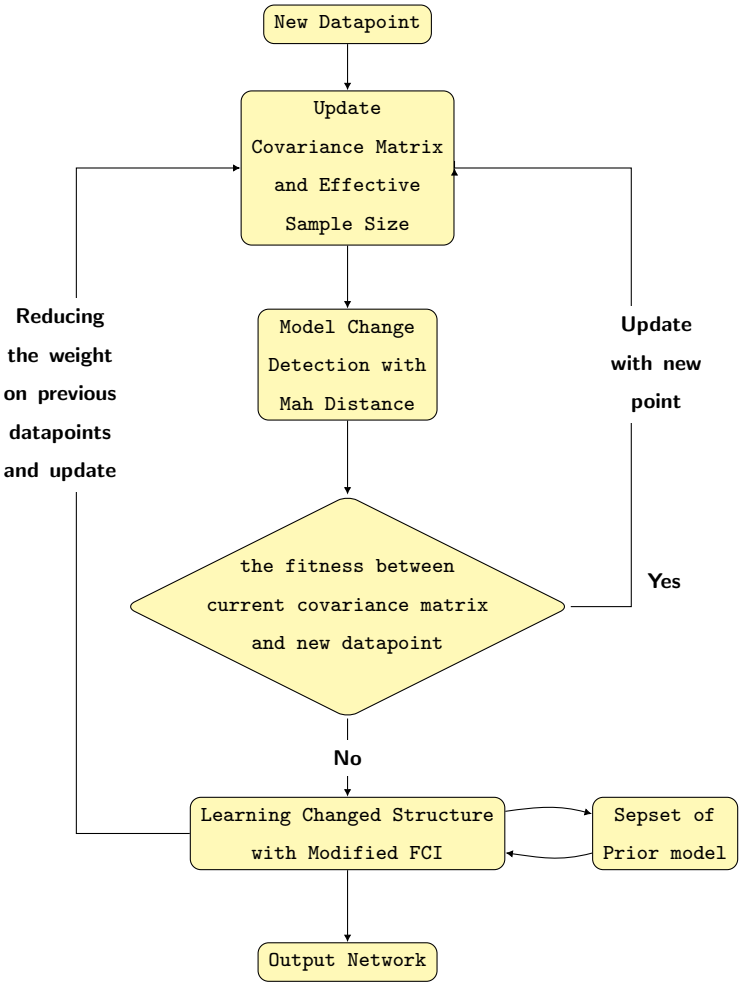
It continues to estimate the initial skeleton. The dependent edges in the initial skeleton are shown in $\circ\!\!-\!\!\circ$ rather than a line $-$. The reason is that the subsets of adjacency the set of variables are no longer enough for deciding of dependencies between variables due to the existence of hidden variables. Therefore, we may have extra edges in the initial skeleton stage [30]. After, the orientation part, which begins with $v - \textit{structures}$ check, is passed. The algorithm orientates unshielded triples. After this stage, the Possible-D-SEP sets are calculated, which is defined as 2.8.5 FCI Algorithm.

Next, we reorient all edges as $\circ\!\!-\!\!\circ$ and then update skeleton and information in separation sets. Finally, orientation rules are applied for doing directed to many circles in the graph, which are proposed by Zhang in 2008 [100].

As can be seen in the experimental results section, this sometimes reduces the independence test by fifty percent. Thus, it saves us from the unnecessary test repetition that can find thousands for large networks. This allows us to start analysing on a more straightforward graph rather than starting from a complete graph like in the classic FCI. The rest of the algorithm continues the same as in the classic FCI. This simple Figure 3.2 represents a process of FOFCI.

In particular, as it is seen in the Figure 3.2, OCME first updates the estimated covariance matrix in response to incoming data points, CMCD tracks the fitness between the current estimated covariance matrix and the input data. Unlike Kummerfeld and Danks's work and OFCI, CML takes the covariance matrix, and also

Figure 3.2: Basic flowchart of FOFCI algorithm



separation sets of the previous learned casual model as input. Structure learning part starts with checking of causal links in separation set in the prior model. If all or some causal links of the prior model still fits incoming data, we do not need to apply the independence tests which are required to find these causal links.

Then, the structure learning algorithm finds the initial skeleton by starting with the graph obtained after this analysis and updates the separation sets at the same time. After learning the causal model, the separation sets which are updated according to the new model are stored to use in the next change point. The process of FOFCI will be identical to OFCI in cases where the causal structure is completely changed. By comparing to OFCI and DOCL, FOFCI seems to need more memory space to store separation sets of learned models, but it performs significantly better than two algorithms in terms of time and space complexity.

The re-learning should be most frequent after an inferred underlying change, though there should be a non-zero chance of re-learning even when the structure appears to be relatively stable. Kummerfeld's work and OFCI have the limitations such as the over the computational cost of re-learning of stable parts in cases where only some parts of the causal structure are changed.

Therefore, FOFCI fills this gap. Optionally, a probabilistic re-learning scheduler is added to the algorithm, which utilises the pooled p-values calculated in the CMCD module to determine when to re-learn the causal model.

3.7 Really Fast Online Fast Causal Inference (RFOFCI)

The size of the Possible-D-SEP sets, which is defined in [88] plays an essential role in the complexity of the FCI algorithm. As the number of variables in a dataset increases, the number of conditional independence tests performed by the

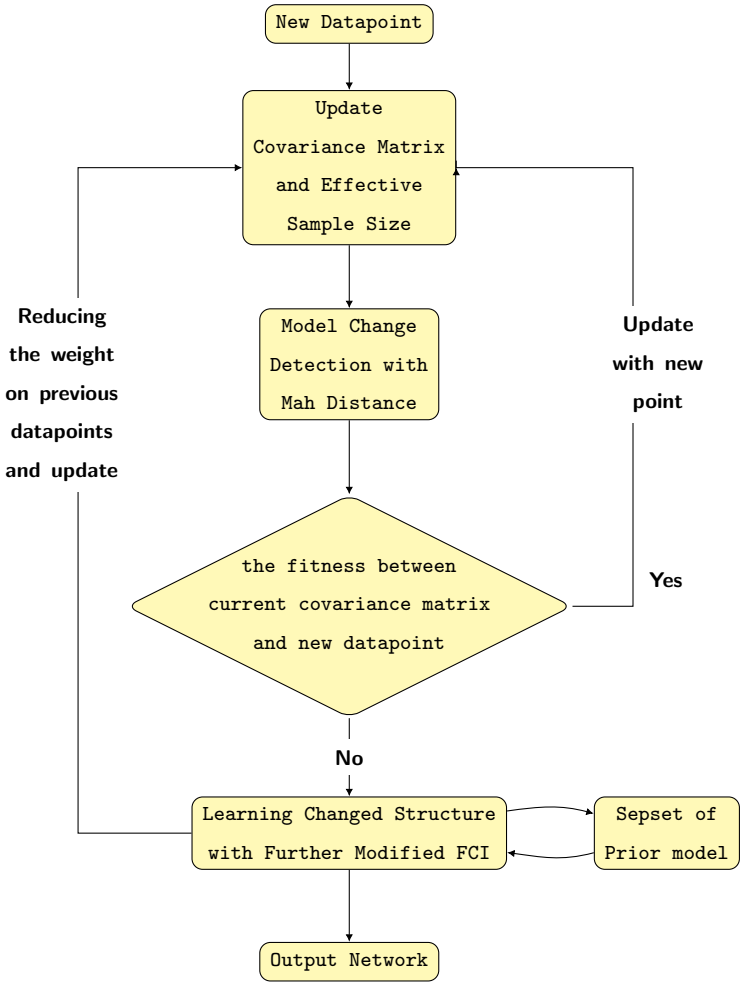
algorithm exponentially grows. Then, the computational complexity of FOFCI dramatically increases because of both computing all Possible-D-SEP sets and testing conditional independence given all subsets of these sets, which can become very large for sparse graphs. Although the FOFCI algorithm is good to learn the changing causal models, it suffers from exponential run-time. Therefore, FOFCI may not be feasible on data sets with large numbers of variables. In this way, we simplified the modified FCI algorithm by removing Possible-D-SEP sets independence test part. We named that as **Further Modified FCI** so that readers don't interfere with the **Modified FCI**. This version does not search the Possible-D-SEP sets but the except of the algorithm the Possible-D-SEP sets is identical to the **Modified FCI**.

In this study, we introduce RFOFCI to fill this gap [36]. RFOFCI is an alternative fast algorithm to the online algorithms previously proposed [37] for one who wants to deal with data sets that are too large or complex to be dealing within the best possible time. The RFOFCI algorithm differs from FOFCI for CML the part by avoiding the conditional independence tests given subsets of Possible-D-SEP sets.

In particular, as it is seen in the Figure 3.3, RFOFCI uses a modified version of the FCI algorithm. In this modified version, the algorithm takes the separation sets of the previous model as input, unlike the classic FCI. In the first part of the algorithm, it is found out whether the causal links of the previous model's separation sets still fit the updated covariance matrix. If some of them still fit, the independence tests that will be applied to determine these relations are eliminated. Unlike the classic FCI and FOFCI, this simplified version does not search Possible-D-SEP sets parts. Therefore, we can start analysing on a more straightforward graph rather than starting from a complete graph like in the classic FCI.

RFOFCI is faster than FOFCI and OFCI. Sometimes, the output of RFOFCI

Figure 3.3: Basic flowchart of RFOFCI algorithm



is slightly less informative than FOFCI and OFCI, but the causal interpretation of its output is still sound. As can be seen in the experimental results section, this sometimes reduces the independence test by ninety per cent.

Chapter 4

EXPERIMENTAL RESULTS

In this chapter, we represent the performance results of the proposed algorithms which are OFCI, FOFCI and RFOFCI and previously presented algorithm whic is FCI are compared under a variety of conditions. These results compare the performance of our proposed algorithms to FCI.

4.1 Synthetic Datasets Application

Our goal is not only to present algorithms that work in real-world scenarios, that is tracking the change of causal structure, but also to propose an algorithm that can compete with and take over the existing batch structure learning. So these methods can also work as a batch structure learning algorithm, even the causal structure has not changed and competes in terms of cost. We used the FCI algorithm [83] as a batch learning algorithm to compare. The desired algorithms optionally include a probabilistic re-learning scheduler that allows us to learn the causal graph again at any time. This feature is required to fairly compare our online algorithms and FCI.

The re-learning scheduler is scheduled for the change points we specified for

both settings. As we mentioned in the proposed algorithm section, algorithms have three distinct parts which OCME, CMCD and CML. While algorithms are working, the CMCD saves changing points determined by Mahalanobis distance and continue to CML part. The algorithms first check the existence of an optional learning schedule as input in CML part. If we add a learning schedule which includes sample size numbers as input, the algorithms learn structure when they reach this sample size. Otherwise, the algorithms learn structure at the changing points determined by Mahalanobis distance. This is important to note that the input schedule works independently from updating sufficient statistics. So, it is optional to add a scheduler and allows to learn anytime we specified. Instead, we also see from the Mahalanobis distance graphs that the CMCD part of algorithms successfully detects the main graph change points, which are 10000, 20000 and 30000 so the algorithms always respond with a high spike at these points.

In this study, we used the data which is assumed to have a multivariate Gaussian distribution. Synthetic datasets are used to verify the accuracy of our online algorithm inference approach when given a known ground truth network. Results are evaluated under the condition where the true partial ancestral graph is changed during the data collection process. We designated one, two and three main change points. Up until the first change point, online and batch algorithms should perform similarly.

We have created each synthetic dataset by following the same procedure by using the **pcalg** package for R [32]. First, we generated four random DAGs, which each DAG has the same number of nodes and is different from each other. Each random DAG is generated to a data with a given number of vertices p', expected neighbourhood size $E(N)$ and sample size 10000 for number of variables $p' \in \{8,10,13,15,18,25,30,35,40,100,125,175,200\}$. Next, we concatenated these datasets which are generated from these four different graphs that have the same

characteristics (vertices, $E(N)$ and sample size) to obtain a dataset with 40000 samples. Our goal here is to obtain a large amount of data that is suitable for the real world, where the structure changes at some point. Therefore the dataset is created by aggregating four different graphs' distributions. Thus, we want to present algorithms that can detect these changes in the environment. That means there are three change points in each data. We do this to see the performance of OFCI, FOFCI and RFOFCI in the case where the causal structure is changed multiple times, which is suitable to a real-world. We restrict each graph to have two latent variables that have no parents and at least two children. (Selection variables are not considered in this study.)

We separated datasets according to the number of variables to three simulation settings to make easy to analyse : small-scale $p' \in \{8,10,13,15,18\}$, average-scale $p' \in \{25, 30, 35, 40\}$ and large-scale $p \in \{100, 125, 175, 200\}$. We only named the scales according to ourselves and we did not get references from anywhere. We also tried to examine all of them together, but due to the size of the data sets, the study looked much more complicated and we decided to examine it separately.

4.1.1 Small Scale

To generate synthetic datasets corresponding to a network, we appeal to the *pcalg* R package, see code 4.1. Firstly, we generated 40 different random directed acyclic graphs (DAGs) with 5 replicates (160 in total) by using the *randDAG* function by the following process mentioned above. There is no specific reason to test 40 different random directed acyclic graphs (DAGs) with 5 replicates (160 in total). We will try to explain step by step how we get the data. First of all, we will start by defining the function we used in the first step.

Definition 27. ***randDAG:*** *It is a function in* pcalg *R package [31], which is generating random directed acyclic graphs (DAGs) with fixed expected number of*

neighbours $E(N)$.

Usage:

- *randDAG(p′, E(N), method ="regular")*

Arguments:

- **p′ :** *integer, at least 2, indicating the number of nodes in the DAG.*
- **E(N) :** *a positive number, corresponding to the expected number of neighbours per node, more precisely the expected sum of the in- and out-degree.*
- ***method:*** *a string, specifying the method used for generating the random graph.*
- ***regular:*** *Graph where every node has exactly N incident edges.*

Listing 4.1: Random True PAG Generating R Code

```
1 g <- randDAG(p', E(N), "regular")
2 cov.mat <- trueCov(g)
3 true.corr <- cov2cor(cov.mat)
4 L <- c(L1,L2)
5 true.pag <- dag2pag(suffStat = list(C=true.corr, n=10^4), indepTest = ...
      gaussCItest, graph=g , L=L, alpha=0.05)
```

After, we defined different nodes to be latent variables. The true covariance matrix of a generated DAG we generated are computed by using the *trueCov* function. To obtain sufficient statistics for generating PAGs, we transformed covariance matrix into a correlation matrix via *cov2cor* function. With the *dag2pag* function in *pcalg* R package, the generated DAGs are converted with randomly selected latent variables into their corresponding (unique) Partial Ancestral Graphs (PAGs) by using the true correlation matrices with a large virtual sample size and alpha.

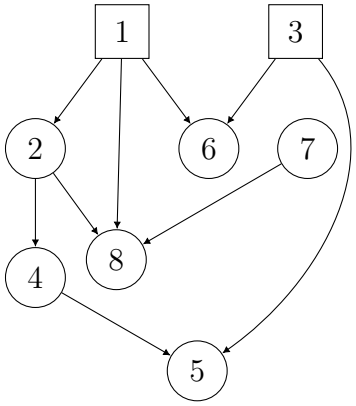


Figure 4.1: Random DAG

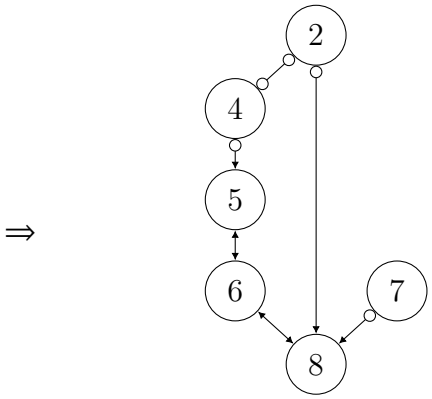


Figure 4.2: Corresponding PAG, L(1,3)

Fig 4.1 shows: an example random DAG produced by *randDAG* function and Fig 4.2 shows: corresponding (unique) Partial Ancestral Graph (PAG) for this DAG and the chosen latent variables *L(1,3)*.

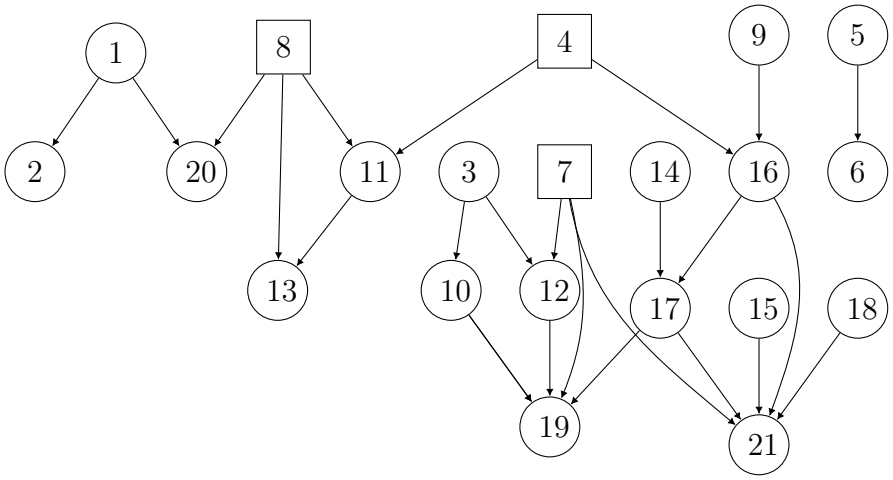


Figure 4.3: Random DAG

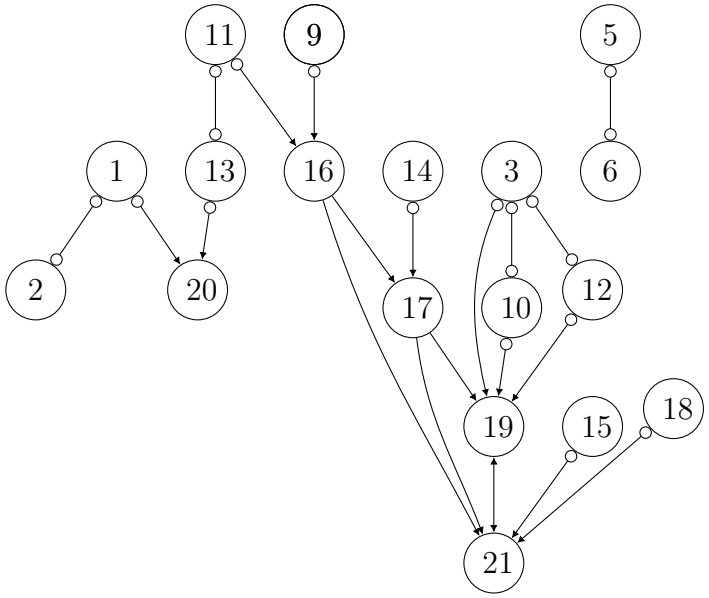


Figure 4.4: Corresponding PAG, L(4,7,8)

Another example, Fig 4.3 shows: an example random DAG produced by *randDAG* function and Fig 4.4 shows: corresponding (unique) Partial Ancestral Graph (PAG) for this DAG and the chosen latent variables *L(4,7,8)*.

After this step, we remove the latent variables from the covariance matrix of the DAG and obtain a covariance matrix *true.cov1*, the basically it is removing two variables from the data. See code 4.2.

Listing 4.2: Latent Variable Remove R Code from Covariance Matrix

```
1 covariance.matrix<-trueCov(g)
2 true.cov1 <- cov.mat[-L,-L]
```

We generate data by using this covariance matrix *true.cov1* and sample size. We use a simple Matlab function, which we wrote below. It generates generate n samples from a d dimensional multivariate Gaussian distribution with a given a specific covariance matrix *true.cov1*.

See code 4.3; we first generate data from a standard Gaussian. After, we

subtract the column mean and its sample covariance matrix from the corresponding column elements of a matrix X, respectively. Last, we are multiplying them by the Cholesky factor of *true.cov1*. Therefore, when we compute covariance and correlation matrices of X, we will obtain the same with the true graph.

Instead of producing data directly from some actual distribution, we have generated data in this way. The main reason for this is full fairness. we want to obtain data which has the same covariance and correlation matrices with those of the true graph. The input of the FCI is the correlation matrix. However, OFCI and FOFCI is a dataset.

First, we generate a random graph, and then we calculate the graph's covariance matrix, next we generate multivariate normal distribution data by this matrix. When we calculate the covariance matrix of this data for a backward control, we sometimes get a different covariance matrix from the originally used covariance matrix to obtain data. We chose this way to eliminate this possibility of differences and to compare on equal terms. Therefore we wrote a generating function, see code 4.3, to guarantee the same covariance matrix retrospectively.

Listing 4.3: Data Generating Code from Covariance Matrix

```
function data_matrix=datagenerateV2(n=10^4, true.cov1)
d=size(true.cov1,2);
Sigma=true.cov1;
X = rand(n,d);
X = bsxfun(@minus, X, mean(X));
X = X * inv(chol(cov(X)));
X = X * chol(Sigma);
data_matrix=X;
end
```

For small scale experimental results, we generated randomly 40 partial ancestral graphs for each number of variables $p' \in \{8,10,13,15,18\}$. Then, datasets are generated of these true PAGs that each has n =40000 sample size and the p-value

for independence tests set to $\alpha = 0.05$ with $E(N)$ =3 (at most). Therefore, we obtained 40 datasets which all have the changing causal structure in 10000, 20000 and 30000 data points.

The resulting evaluation of the desired algorithm is based on the exact true graph means includes direction of edges, which is a PAG used for generating dataset rather than the Markov equivalence class of the true graph. If we base on Markov equivalence classes of the true DAG we used to obtain PAG, we would be ignoring the ability of the desired algorithm to detect the existence of latent variables. Whereas, the online algorithm is advantageous to detect the place of latent variables while learning. Additionally, such as $X \rightarrow Y \rightarrow Z$ and $X \leftarrow Y \leftarrow Z$ are both causal graphs. These two graphs are Markov equivalent, but their causal relationship is entirely different. Therefore, it might be the wrong evaluation in this kind of situation.

Fig 4.5 represents the mean of missing or extra edge number by comparing to true graph (that is the learning ability to true graph) when the causal structure changes three times during the data collection process. In the graph, zero means that there is no missing edge and the algorithm works perfectly. High numbers represent the poor fit to the true causal model.

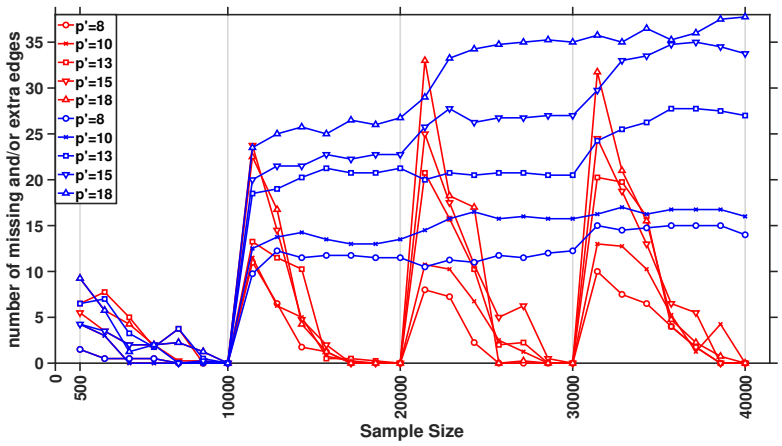


Figure 4.5: Average number of missing and/or extra edges FCI (blue) and OFCI (red) versus to True PAGs including the direction of the edges, p' is indicating the number of nodes

Fig 4.5 represents the mean of missed edge number when the causal structure changes three times during the data collection process. Each line in the Fig 4.5 (b) shows the mean of the number of missed edges of the online (red) and FCI (blue) algorithm for two datasets that have 40000 sample size and changing structure in 10000, 20000 and 30000 data points. Zero means that there is no missed edge with the true underlying causal model. In other words, the algorithm can learn the true model. High numbers indicate poor fit and uninformative network. By representing this graph, our purpose is to emphasise complex outputs of the batch structure learning algorithm when the structure is changed a few times.

As can be seen Fig 4.5, the algorithm works perfectly for situations where the structure changes completely, because if the structure is completely changed, it adjusts its weight accordingly and the previous structure has little effect on the changed structure. However, this will not be the same for causal models that are not completely changed (just partially changed causal model). So adjusting the weights will be more difficult if the structure has been partially changed.

As is clear from the graphs, the performance of the online algorithm and FCI

could not be distinguished from each other until 10000 data points in which the first changing point of the causal structure. The desired algorithm optionally includes a probabilistic re-learning scheduler that allows us to re-learn the causal graph at any time we required. In this way, we scheduled learning process for {500, 1000, 3500, 5000, 7500, 9000, 10000, 11000, 13000, 15000, 17500, 18000, 19000, 20000, 20500, 23000, 25000, 27500, 28000, 29000, 30000, 30500, 33000, 35000, 37500, 38000, 39000, 40000} datapoints for the online algorithm. These numbers are selected randomly to represents the behaviour of the algorithms in detail until change point. Intervals represent just data points. The online algorithm re-learned the causal structure, and FCI was rerun after these data points. After the underlying causal structure is changed in 10000, 20000 and 30000 data points, the online algorithm significantly outperformed FCI. As the datasets are a mix of four different distributions that indicate a large number of synthetic variables, FCI works poorly after the first changing datapoint. The Online algorithm measures major Mahalanobis distance for changing data points as it can be seen from the example in Fig 4.6. Therefore, it leads to higher weights and learns the new underlying causal structure.

The algorithm does not store any of datapoints coming sequentially. Its memory requirements are just for the estimated covariance matrix. Therefore, the algorithm has significant storage advantages for computational devices that cannot store all data. Additionally, the online algorithm has an essential advantage in terms of computational time for complex networks. To emphasise, we compare the time differences of the structure learning processing (in seconds) between the online and FCI algorithm in Table 4.1. The table represents learning processing time (in seconds) for 30 different datasets having sample sizes: 20000, 30000 and 40000. The 20000 sample size dataset has one causal structure change, the 30000 and 40000 sample size datasets have the structure changing two and three times,

respectively.

For datasets with just eight variables, the FCI algorithm outperforms the online algorithm by a small amount. However, for complex networks, the online algorithm outperforms FCI significantly. Especially, as the size of datasets grows, the performance gap grows greater.

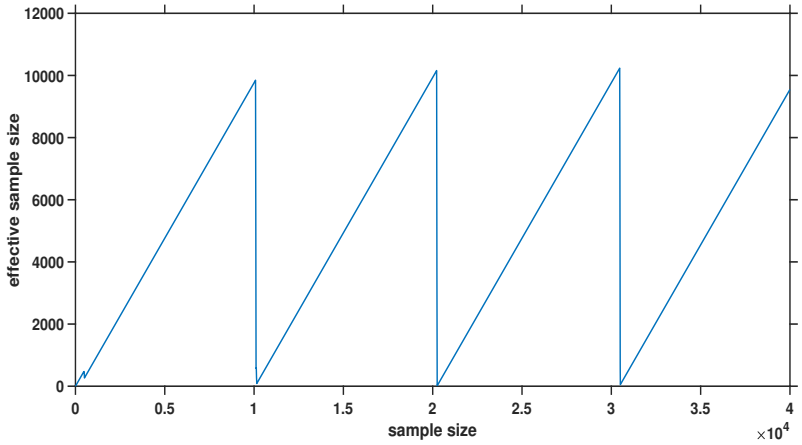


Figure 4.6: Effective sample Size change while learning

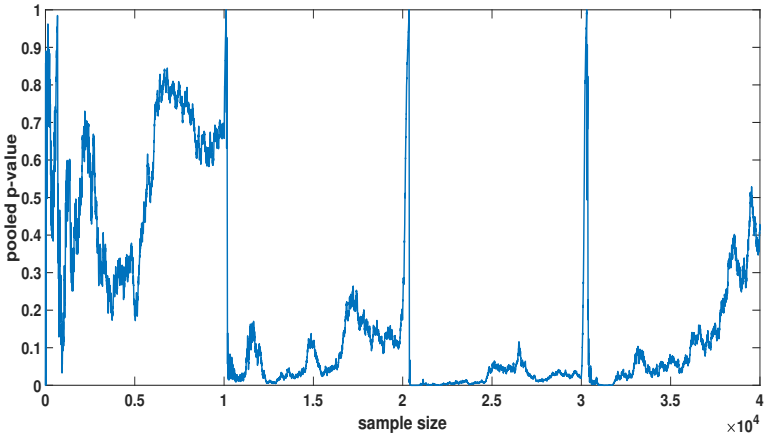


Figure 4.7: Pooled p-values change while learning

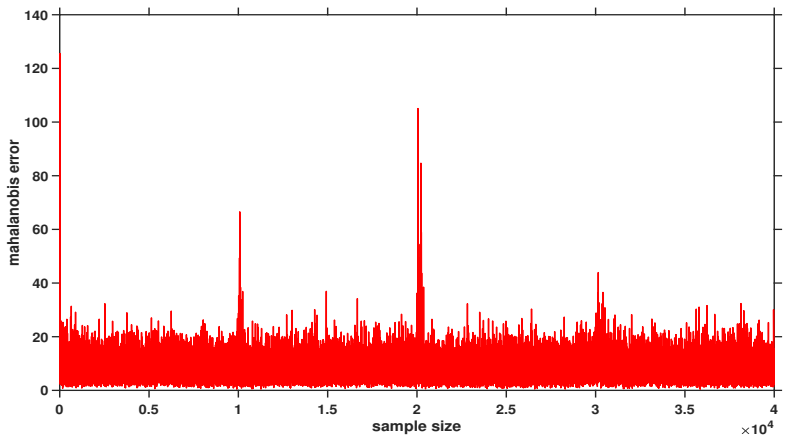


Figure 4.8: Mahalanobis distances change while learning

In Fig 4.6, Fig 4.7 and Fig 4.8, we aim to represent the changing of sample size, pooled p-values and Mahalanobis distances during the learning process. We just used 18 variables, 40000 sample size example.

Table 4.1: Average Running Time in seconds for 20000, 30000 and 40000 points

Dataset ($\|Vars\|$)	20000		
	FCI	OFCI	Better?
Dataset (8)	**1.5**	4.5	×
Dataset (10)	4.5	**4.5**	✓
Dataset (13)	27.5	**6**	✓
Dataset (15)	57	**16**	✓
Dataset (18)	259	**10.5**	✓
	30000		
	FCI	OFCI	Better?
Dataset (8)	**4.5**	6.5	×
Dataset (10)	13.5	**7**	✓
Dataset (13)	74	**14.5**	✓
Dataset (15)	225.5	**36**	✓
Dataset (18)	1093.5	**94.5**	✓
	40000		
	FCI	OFCI	Better?
Dataset (8)	**7**	9.5	×
Dataset (10)	23	**10**	✓
Dataset (13)	171.5	**22.5**	✓
Dataset (15)	542.5	**51.5**	✓
Dataset (18)	2475	**202**	✓

We compare structure learning time differences (in seconds) between OFCI and FCI in Table 4.1. The Table 4.1 represents learning processing time (in seconds) for 40 different datasets having sample sizes: 20000, 30000 and 40000. For datasets with just dataset(8), the FCI outperforms OFCI by a small amount. However, for complex networks, OFCI outperforms FCI significantly because OFCI updates just the estimated covariance matrix. Especially, as the size of datasets grows, the

performance gap becomes greater.

4.1.2 Average Scale

To generate synthetic datasets corresponding to a network, we appeal to the *pcalg* R package. For average scale experimental results, the simulation setting is as follows. For each value of $p' \in \{25, 30, 35, 40\}$, we generated 160 random DAGs with $E(N)$ =2 (at most). We generated a data set that has n =40000 sample size and the p-value for independence tests set to $\alpha = 0.05$ by using the *randDAG* function by following the process mentioned above. In 4.1.1 Small Scale part, we tried to explain step by step how we get the data. We follow the same process for average and large scale datasets.

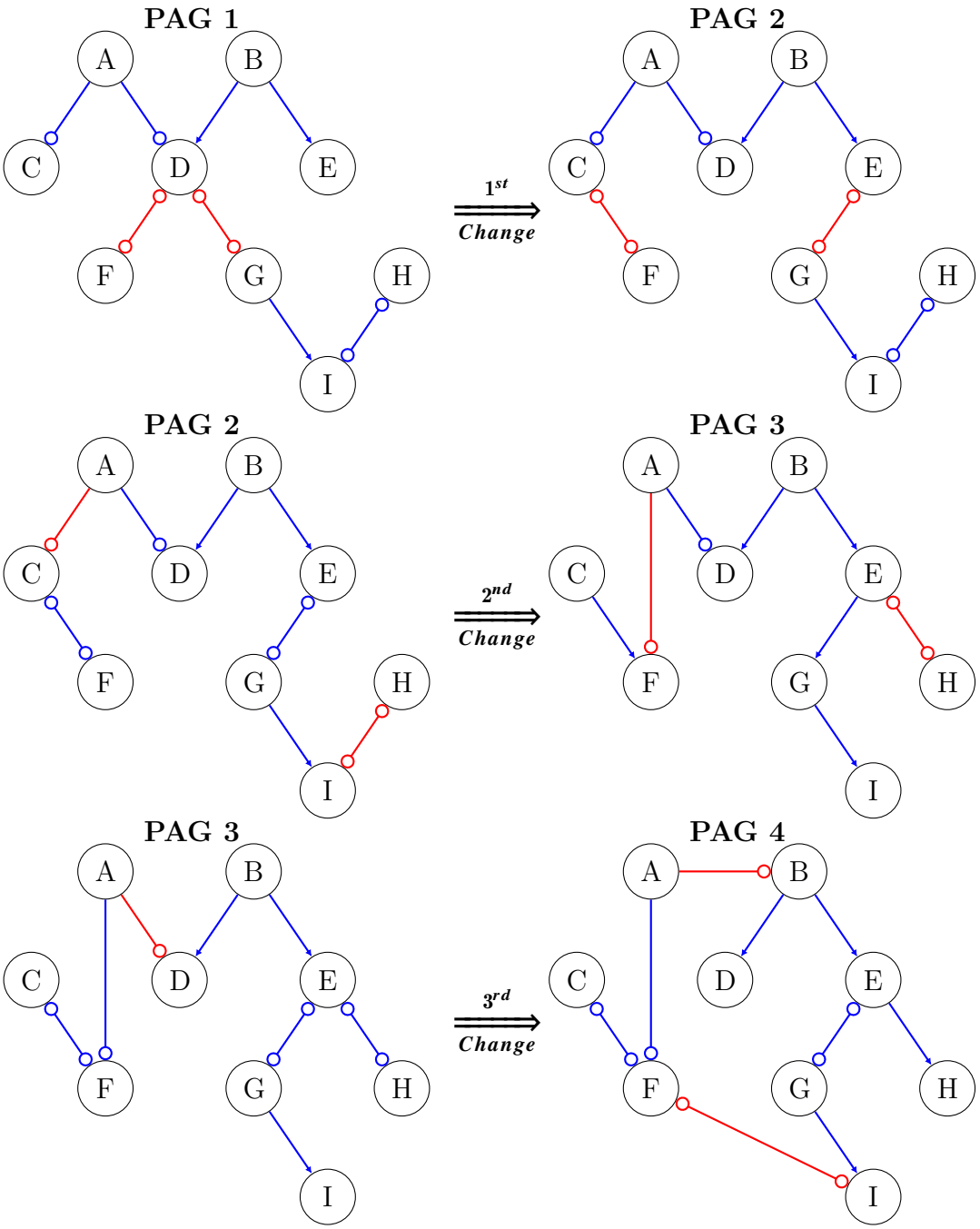


Figure 4.9: Representative example of 4 different PAGs for generating data process.

Figure 4.9 shows a representative example of our data generating process. We generated random partial ancestral graphs (PAGs) for these experimental results.

In this process, we produced a random PAG as the first step, then modified this graph, assuming that the structure changed over time. We changed the graph three times in total, we got 4 PAGs at final. We limited these changes to be between 4-8 edges for the average scale. In other words, there are 4-8 edges differences between the first PAG and the second PAG, and the other edges are identical. There are 4-8 edges differences between the second PAG and the third PAG, and the other edges are identical. There are 4-8 edges differences between the third PAG and the fourth PAG, and the other edges are identical. Thus, we have ensured that only local parts (not entirely) change at each change point and that the change between the first PAG and the last PAG become significant.

As illustrated in the Figure 4.9, the joint edges between the graphs at each change point are shown as blue and the different edges as red. In this way, we obtain four different PAGs and generate random variables by using their covariance.

In real-world datasets, some edges or adjacencies in the model may stay stable during data collection and learning process. As we see from 4.1.1 Small Scale part, OFCI is a structure learning algorithm that works successfully when the causal structure changes entirely over time. However, OFCI has the advantage only when the causal structure is completely changed. If the structure does not change completely, that is to say, that some parts of the structure still fit the incoming data, OFCI must repeat some independence tests to learn the unchanged parts again.

In the real world scenario, it is more likely that some parts of a structure change than all of it. Since the complexity of structure learning in the Bayesian networks increases exponentially depending on the number of variables, we need to save on the high cost of re-learning for the unchanging parts in large structures. Therefore, we proposed the FOFCI and RFOFCI algorithms which allow us to use prior learned model information while learning the new changed structure. When

the structure is changed, the OFCI updates the covariance matrix and learn the new changed structure in the light of this information. the FOFCI and RFOFCI algorithms save the learned models in each change point and use these models as well as the updated covariance matrix to reduce complexity.

First, we investigated the performances of OFCI, FOFCI and RFOFCI, considering the number of differences in the output by comparing to the Markov equivalence class of the true DAG. We made an average scale schedule review. The re-learning scheduler is scheduled for the main change points, which are 10000, 20000, 30000, 40000 for all algorithms. Normally, OFCI, FOFCI and RFOFCI are triggered for re-learning by large p-values but comparing for each change point requires a very long review. From the Mahalanobis graphs, we already see that the algorithms successfully perceive the main change points with higher peaks, which is much higher from other points. Therefore, we scheduled to re-learn to main change points are 10000, 20000, 30000, 40000 to see how successful it has approximated the true graph at the end of each change point.As the FCI is not an online algorithm, we rerun (reset) FCI at these change points.

The re-running FCI after these change points is not a fair approach for online algorithms. So online algorithms here proposed to take and process data individually but we have looked at it collectively for the FCI. In the real world, the data will be collected and analysed when available. Therefore, it is impossible to work with the FCI in cases where the data set is not complete. For example, if the data set is insufficient, all variables may be dependent. It will be impossible to compute exponentially. Nevertheless, we still compared them according to their learning performance at these points.

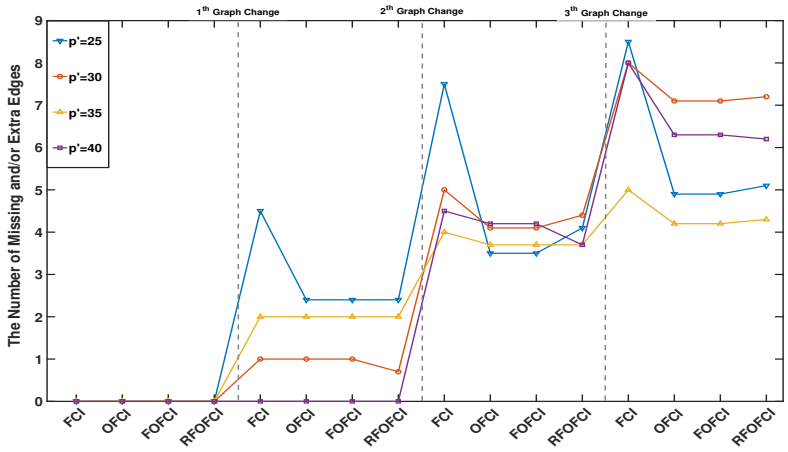


Figure 4.10: The average number or missing and/or extra edges, p' is indicating the number of nodes

Figure 4.10 shows the average number of missing or extra edges over 5 replicates, and we see that this number was almost identical for all algorithms. As expected, OFCI and FOFCI perform the same to learn the true causal model. they outperform RFOFCI to learn the true causal model in some cases. It should be identical for OFCI and FOFCI learning performance. So the only difference between them is that FOFCI should learn the same model with OFCI with far fewer independence tests by using the models learned in previous exchange points. With this experimental result, we have confirmed this. In graph 4.10, zero means that there are no missing or extra edges, and the algorithm work correctly. High numbers represent the poor fit to the true causal model.

As we mentioned in the section where we examined the results for small scale, the algorithm works perfectly for situations where the structure changes completely, because if the structure is completely changed, it adjusts its weight accordingly and the previous structure has little effect on the changed structure. However, this is not same for causal models that are not completely changed as seen Figure 4.10. So now it will be more difficult for the algorithm to determine

the weights than the completely changed structure and there will be more errors in the learned structure.

However, there is a situation that should not be confused here in the small scale part since we compared only two algorithms (because OFCI and FOFCI work identically in completely changing structures) we can show their learning performances over time. The structure of the two graphs, which are Fig 4.5 and Fig 4.10, is different from each other. In Fig 4.10, we gave directly the learning performance results of algorithms at each change point rather than the overtime. The reason for the increase in the fig 4.10 is only related to the dimension of the data. As the dimension of the data increases, the number of incorrect edges increases. if we performed the algorithms change over time (which would be a very complicated figure), we would able to see that learning performances improved as new data arrived.

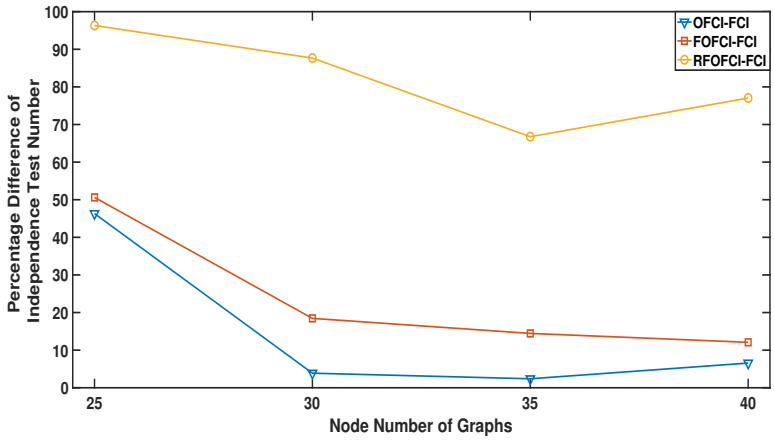


Figure 4.11: The average percentage differences of the number of conditional independence tests required for FCI and other algorithms to learn a model

Figure 4.11 shows The average percentage differences of the number of conditional independence tests required for FCI and other algorithms to learn a model in average scale settings. We first determined the number of necessary independence

tests to learn a model of FCI, OFCI, FOFCI and RFOFCI. Then, we calculated how many percent less independence the other algorithms need to learn a model from FCI. To explain with a very simple mathematical calculation; Let us assume that if the FCI performs 10000 independence tests to learn a graph which has 40 nodes and other algorithms perform 1000 tests for the same graph, other algorithms learn better or the same with ninety percent fewer independence tests.

We see that RFOFCI requires significantly fewer independence tests compared to OFCI and FOFCI to learn the causal model for all the same parameter settings. In Figure 4.11, high numbers represent the success of the algorithm in pruning search space. For example, RFOFCI learns the structure by applying over the ninety percent less independence tests than FCI for 25 variable graphs. RFOFCI outperforms OFCI and FOFCI in a large margin.

Table 4.2: The average percentage differences of the number of conditional independence tests required for FCI and other algorithms to learn a model

Dataset	OFCI	FOFCI	RFOFCI	Better?
Dataset (25)	46.3	50.6	**96.3**	✓
Dataset (30)	3.8	18.4	**87.6**	✓
Dataset (35)	2.3	14.4	**66.7**	✓
Dataset (40)	6.5	12.0	**77.0**	✓

It can be seen in detail in Table 4.4. We continued with a comparison of average percentage reduction of conditional independence test number performed by OFCI, FOFCI and RFOFCI (in seconds) under the same simulation settings. Table 4.5 shows the average running times.

Table 4.3: Average Running Time in seconds for 40000 sample size data

Dataset	OFCI	FOFCI	RFOFCI	Better?
Dataset (25)	16.3	14.8	**13.8**	✓
Dataset (30)	19.5	18.9	**15.4**	✓
Dataset (35)	26.0	21.8	**20.9**	✓
Dataset (40)	33.4	28.0	**22.1**	✓

We see that RFOFCI is faster for all parameter settings. RFOFCI learned the causal models faster than OFCI and FOFCI. As the scale expands, the difference between them also grows.

The algorithm does not store any of datapoints coming sequentially. Its memory requirements are just for the estimated covariance matrix and sample size. Therefore, the algorithm has significant storage advantages for computational devices that cannot store all data.

4.1.3 Large Scale

The simulation setting is as follows. For each value of $p \in \{100, 125, 175, 200\}$, we generated 160 random DAGs with $E(N)$ =2. We generated a data set that has n =40000 sample size and the p-value for independence tests set to $\alpha = 0.05$. As we noted in 4.1.2 Average Scale, we applied the same procedure for generating datasets with an average scale part.

We generated random partial ancestral graphs (PAGs) for these experimental results. In this process, we produced a random PAG as the first step, then modified this graph, assuming that the structure changed over time. We changed the graph three times in total, we got 4 PAGs at final. We limited these changes to be between 8-12 edges for the large scale. In other words, there are 8-12 edges differences between the first PAG and the second PAG, and the other edges are identical. There are 8-12 edges differences between the second PAG and the third PAG, and

the other edges are identical. There are 8-12 edges differences between the third PAG and the fourth PAG, and the other edges are identical. Thus, we have ensured that only local parts (not entirely) change at each change point and that the change between the first PAG and the last PAG become significant. we investigated the performances of OFCI, FOFCI and RFOFCI, considering the number of differences in the output by comparing to the Markov equivalence class of the true DAG. We made a large scale schedule review. The re-learning scheduler is scheduled for the main change points, which are 10000, 20000, 30000, 40000 for all algorithms.

In real-world datasets, some edges or adjacencies in the model may stay stable during data collection and learning process. Therefore, while generating random PAGs, we give attention to generate sample graphs which have local differences.

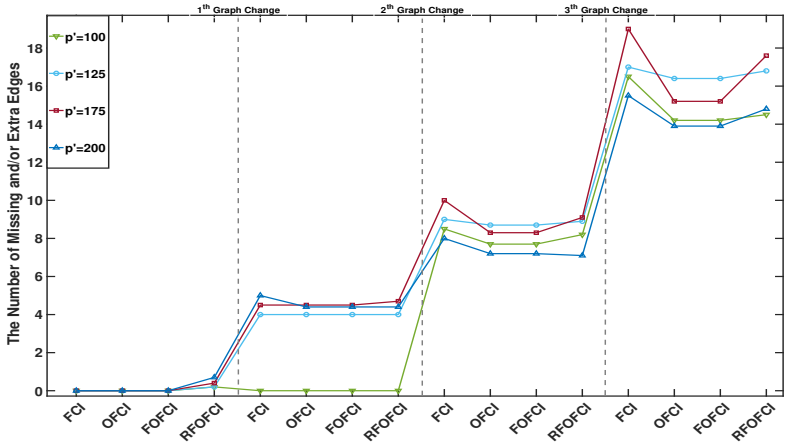


Figure 4.12: The average number of missing or extra edges over 5 replicates, p' is indicating the number of nodes

Figure 4.12 shows the average number of missing or extra edges over five replicates, and we see that this number was almost identical for all algorithms. As expected, OFCI and FOFCI perform the same and outperform RFOFCI to learn the true causal model in some cases. Zero means that there are no missing or extra edges, and the algorithm works correctly. High numbers represent the poor

fit to the true causal model. In Fig 4.12, we gave directly the learning performance results of algorithms at each change point rather than the overtime. The reason for the increase in the fig 4.12 is only related to the dimension of the data. As the dimension of the data increases, the number of incorrect edges increases. if we performed the algorithms change over time (which would be a very complicated figure), we would able to see that learning performances improved as new data arrived.

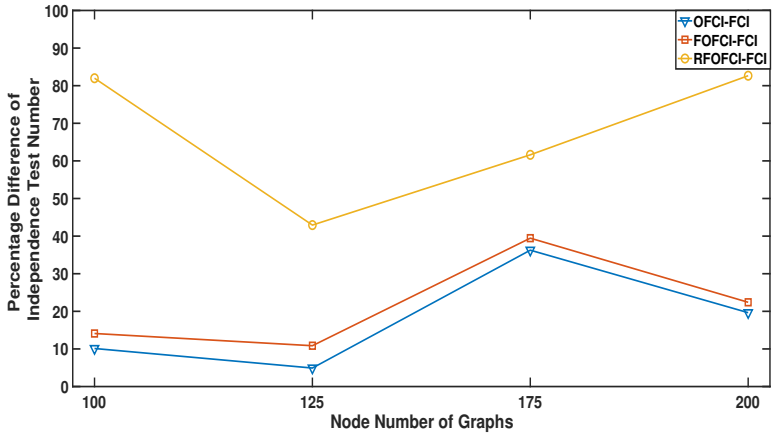


Figure 4.13: The average percentage differences of the number of conditional independence tests required for FCI and other algorithms to learn a models

Figure 4.13 shows the average percentage difference of independence test number in large-scale settings. We first determined the number of necessary independence tests to learn the causal model for FCI, OFCI, FOFCI and RFOFCI. Then, we calculated the percentage difference of the independence test of these algorithms according to FCI. In Figure 4.13, we see that RFOFCI requires significantly fewer independence tests compared to OFCI and FOFCI to learn the causal model for all the same parameter settings.

Table 4.4: The average percentage differences of the number of conditional independence tests required for FCI and other algorithms to learn a model

Dataset	OFCI	FOFCI	RFOFCI	Better?
Dataset (100)	10.1	14.1	**81.9**	✓
Dataset (125)	4.9	10.8	**42.9**	✓
Dataset (175)	36.2	39.4	**61.6**	✓
Dataset (200)	19.6	22.4	**82.6**	✓

It can be seen in detail in Table 4.4. We continued with a comparison of average percentage reduction of conditional independence test number performed by OFCI, FOFCI and RFOFCI (in seconds) under the same simulation settings. Table 4.5 shows the average running times in the small and large-scale setting.

Table 4.5: Average Running Time in seconds for 40000 sample size data

Dataset	OFCI	FOFCI	RFOFCI	Better?
Dataset (100)	113.8	101.6	**98.6**	✓
Dataset (125)	171.1	154.8	**148.0**	✓
Dataset (175)	338.5	307.9	**298.3**	✓
Dataset (200)	501.4	377.7	**373.3**	✓

We see that RFOFCI is faster for all parameter settings. RFOFCI learned the causal models faster than OFCI and FOFCI. As the scale expands, the difference between them also grows.

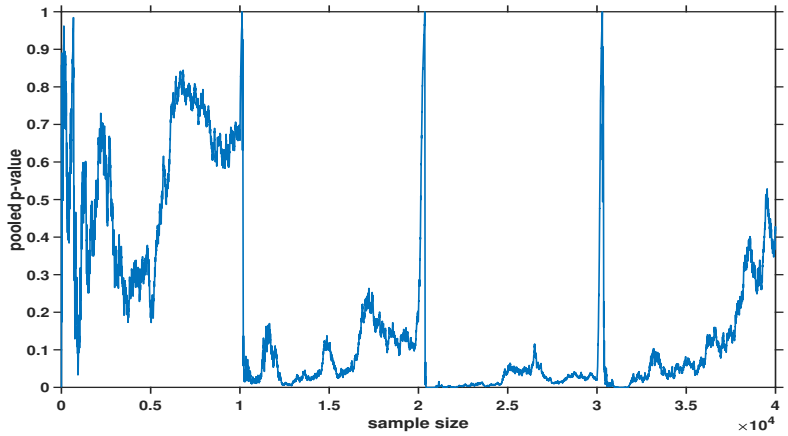


Figure 4.14: Pooled p-values

We also represented pooled p-values Figs 4.14. The datasets are a mix of four different distributions that indicate a large number of synthetic variables.

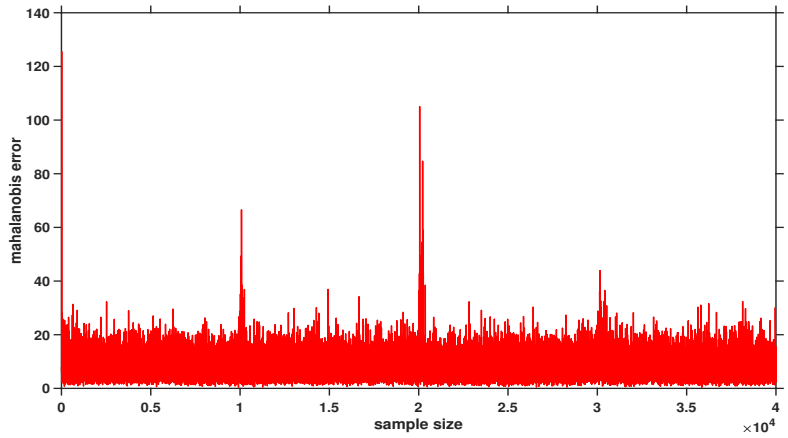


Figure 4.15: Mahalanobis distances

CMCD part of three algorithms measures significant Mahalanobis distance at changing datapoints as can be seen from the example in Figure 4.15. Therefore, it leads to higher weights and learns the new underlying causal structure.

4.2 Real World Data Application

We have applied the FOFCI algorithm to seasonally adjusted price index data available online from the U.S. Bureau of Labor Statistics to confirm the efficiency of the change detection part of the online learning algorithms. Since there is no true graph in real life, we do not have a true graph to compare the learned structure of this data set. Thus, we approached an assumption that the high peaks in the Figure 4.16 may indicate the economic changes during the period.

We have limited the data to commodities extending to at least 1967 and resulting in a data set of 6 variants: Apparel, Food, Housing, Medical, Other, and Transportation. Data were collected monthly from 1967 to 2018 and reached 619 data points. Due to significant trends in the indices over time, we used the month-to-month differences.

As we do not have a true graph, we do not have a mechanism to measure fitness between outputs and the true model. The selection of data and variables was chosen entirely based on the work of Kummerfeld and Danks [45]. Data is taken from U.S. Bureau of Labor Statistics `https://www.bls.gov/rda/` website with its data finder tool. We do not know (maybe no one knows) how many data samples or variables we need to analyze. Therefore, we have only one choice. It is to examine whether economic crises may affect the connections between variables and whether there are changes in time and to analyze the points of change by looking at the Mahalanobis graph. This is examined the CMCD part of our study. This part is identical in three algorithms, which are OFCI, FOFCI and RFOFCI. Therefore, we did not analyze OFCI and RFOFCI algorithms in this section because it had no logic. As this part of these three algorithms that detect changing, structures are identical, the Mahalanobis graphs of all three are identical. Therefore it is sufficient to show the results for only one.

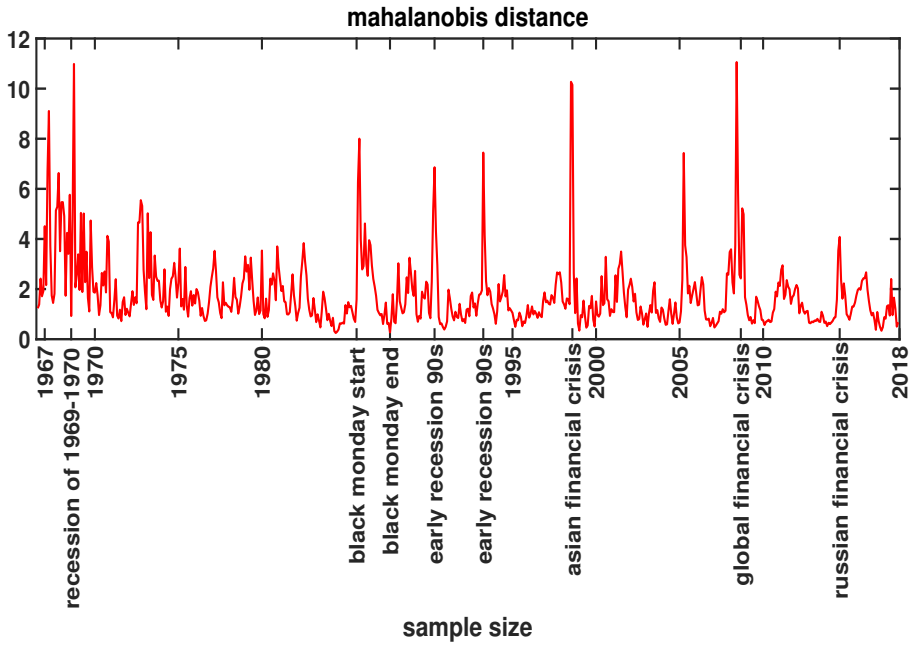


Figure 4.16: Mahalanobis distances

Figure 4.16 shows Mahalanobis distance that has been collected for each month. Notably, we assume the proposed algorithm detects a shift in the volatility of the causal relationships among these price indexes around recession of 1969-1970, the black Monday start in 1986, the black Monday end in 1986, 1990s early recession, Asian financial crisis in 1997, Global financial crisis in 2007-2008 and Russian financial crisis in 2014. The reason the graph looks quasi-periodic is that the selection of data and variables was chosen entirely based on the work of Kummerfeld and Danks [45]. We do not know (maybe no one knows) how many data samples or variables we need to analyze. We do not have enough economic background. We just assume that the changing relationships of these variables may be detected by Mahalanobis distance. Our assumption that the peak points in the Mahalanobis graph coincided with some economic crises strengthened our assumption. Nevertheless, as you will see from figure 4.20, the dependency is changed a lot between variables. That may indicate that the variable or data sample is not enough.

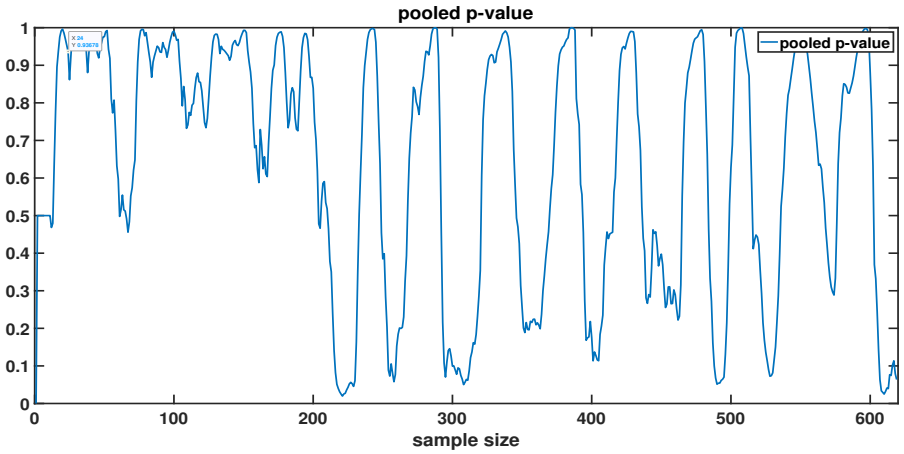


Figure 4.17: Pooled p-values

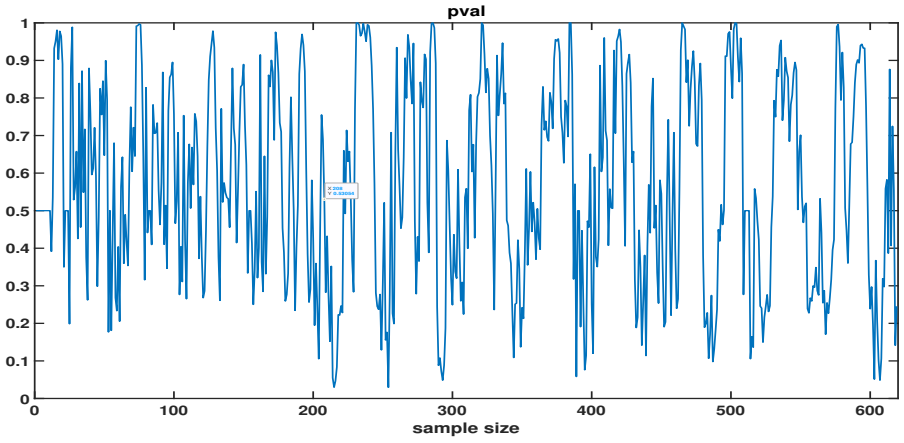


Figure 4.18: P-values

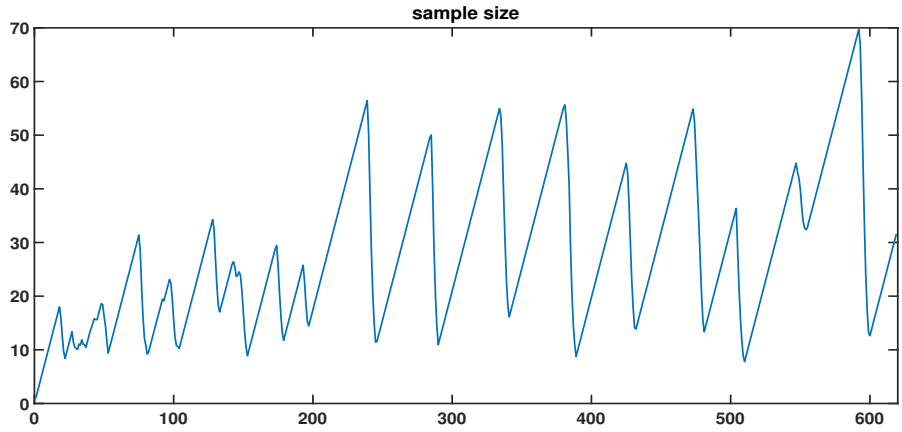


Figure 4.19: Effective sample size

Figure 4.17, Figure 4.18 and Figure 4.19 shows the drivers of these changes: the pooled p-values, the p-values and the effective sample size. This real-world case study also shows the importance of using pooled p-values.

We do not have a true graph in the real world to compare with the output of the algorithm. Therefore, we will just display 8 change point outputs for this here and all change points outputs in Appendix A. As this study aims to propose an efficient algorithm for real-world cases, we will not measure financial performance; analysing and interpreting on this data. In graphs, A is Apparel, F is Food, H is Housing, M is Medical, O is Other Goods and Services, and T is Transportation. We may just assume that the changing relationships of these variables may be detected by Mahalanobis distance. Our assumption that the peak points in the Mahalanobis graph coincided with some economic crises strengthened our assumption. As seen from figure 4.20, the dependency is changed a lot between variables and many edges are unknown head. That may indicate that there are hidden variables or data sample is not enough to make an assumption.

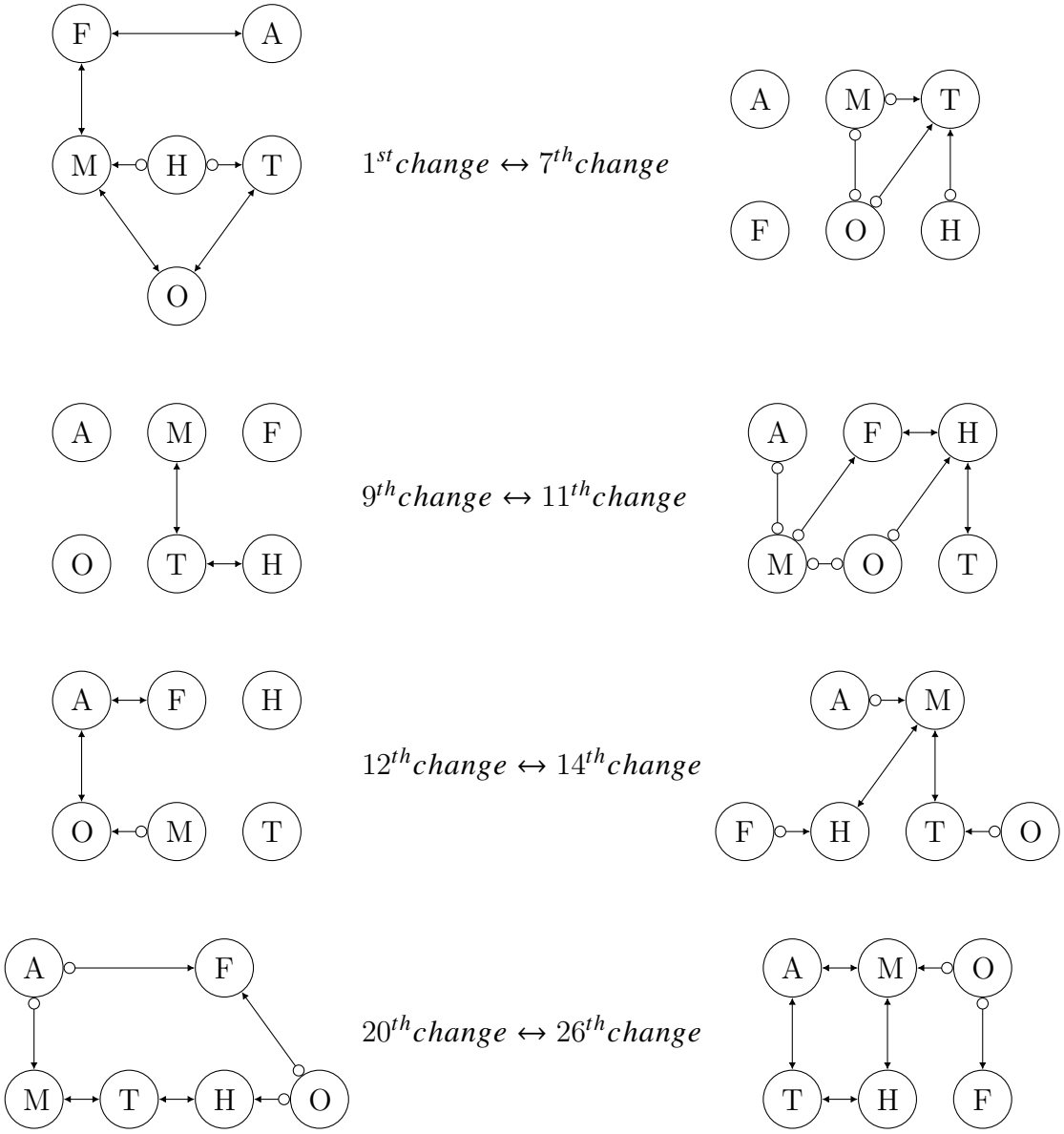


Figure 4.20: FOFCI Real-world data change point outputs

We assume that Figure 4.20 change points outputs might relate some crises such as 7^{th} change point matches Black Monday start year 1986, 9^{th} change point is Black Monday end year 1988, 11^{th} and 12^{th} change points are Early recessions 90s, 14^{th} change point is Asian financial crisis in 1997 and 20^{th} change point is Global financial crisis in 2007-2008.

The simulations were performed on a dual-core Intel Core i5 with 2.6 GHz and 16 GB RAM on macOS using Matlab R2018a.

Chapter 5

GENERAL CONCLUSION

5.1 Discussion and Future Research

In last ten-year, data stream has become an active research area. There are a lot of studies dealing with different methods to analyse rapidly arriving data in real time. the new and old data instances can be stored and processed with batch algorithms for learning model but requires too much computing and memory space. There are very strong constraints and requirements in real world, which can affect the learning process.

In a real-world scenario, when we do not have enough large databases, we need alternative memory. In this kind of situation, It is not possible to examine this data more than once; therefore, the secondary memory may be unreasonable. Sometimes, an intelligent agent must use a domain model to perform a performance task, even if the entire dataset is not available. These changing world conditions make it difficult for intelligent agents to survive. Therefore, online learning brings a natural solution to deal with these changing world situations. It keeps a domain model throughout the entire learning process and uses an only fixed amount of main memory.

In this study, we deal with the problem of Causal Bayesian Network structure learning from data streams. We addressed the constraint-based Bayesian network structure learning technique to learn the structure of Causal Bayesian Network because it can handle a large number of variables. Except for Kummerfeld and Danks's work [46] (their study does not allow hidden variables) and our works [37], to the best of our knowledge, there is not an existing online algorithm which uses constraint-based structure learning technique to learn causal information between random variables when allowing arbitrarily many latent and selection variables. Therefore, we purpose to reveal a novel and significant algorithm in such settings.

In this thesis, we introduced three online structure learning algorithms which are OFCI, FOFCI and RFOFCI. We have presented these three algorithms in the following order: OFCI, FOFCI and RFOFCI.

We first introduced the OFCI algorithm, which is a modified version by using algorithm plug-and-play feature. We replaced the PC with the FCI algorithm, which allows for the existence of hidden and selection variables to make the algorithm suitable for most real-world applications. In a nutshell, OFCI is Kummerfeld and Danks's work-PC+FCI and outputs a PAG. However, Kummerfeld and Danks's work and OFCI algorithms only detect and respond to change points. So they start learning from a complete graph at each time. Although it is very vital to identify these change points, re-learning each time increases the cost because we know some parts will be stable after changes. So if we only search for the changing parts of the model, then one can talk about learning online. Otherwise, it will just remain as an algorithm which can track the structure changes.

In this way, we proposed the FOFCI to reduce the computational cost of re-learning and make the proposed algorithms more online. The FOFCI algorithm differs from both OFCI and Kummerfeld and Danks's work [46] for causal model learning the part. The FOFCI allows us to use prior model skeleton information

and saves us from repeating the independence tests required to find out the connections that existed in the previous model and still fit the new data. The results show the efficacy of the proposed algorithms compared to FCI.

Last, we introduced an alternative fast algorithm to the online algorithms previously proposed [37] for one who wants to deal with data sets that are too large or complex to be dealing with in best possible time for learning causal models, which is called RFOFCI. We evaluated the performance of this algorithm by testing them on synthetic and real data. The results show the efficacy of the RFOFCI compared to online algorithms OFCI and FOFCI previously proposed.

We evaluated the performance of these algorithms by testing them on synthetic and real data. We separated the synthetic data experimental results into three scales to show the performance differences in different settings. For small scale synthetic datasets, the outputs of OFCI and FOFCI are identical to each other and better than FCI. Also, FOFCI requires fewer conditional independence tests than OFCI and FCI to learn the causal model for both small and large numbers of variables. Additionally, we showed that FOFCI is faster than OFCI due to the smaller search space of the FOFCI algorithm.

For average and large scale synthetic data sets, the outputs of OFCI, FOFCI and RFOFCI are almost identical to each other for most cases. Also, FOFCI requires substantially fewer conditional independence tests than OFCI and FOFCI to learn the causal model for both average and large numbers of variables. Additionally, we showed that RFOFCI is faster than OFCI and FOFCI concerning fewer conditional independence test number.

The online algorithms proposed here are useful for learning changing causal structure. We showed that the algorithms are useful for tracking changes and learning new causal structure in a reasonable amount of time. However, the algorithms have limitations. Sometimes, the new model learning process of algorithms

takes a long time because they require most of the data samples to learn the true model. This means that online algorithms will perform poorly if the causal structure changes rapidly. As it can be seen from Figure 4.10 and Figure 4.12, the output of RFOFCI is slightly less informative in some situations, regarding conditional independence information.

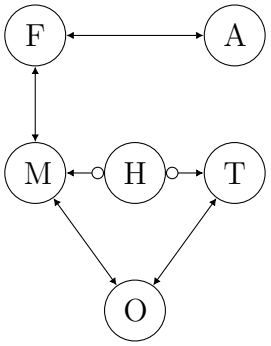
OFCI, FOFCI and RFOFCI have a plug-and-play design. That indicates that it has a structure that allows you to easily replace any structure learning algorithm with the existing one and bring it online. This allows us to adapt to every new and effective learning algorithm in structure learning algorithms to the system. This feature allows us for easy modification to use alternative algorithms. A range of alternative structure learning algorithms could be used for the learning part, constraint-based methods such as RFCI [32] and score-based methods such as greedy search algorithms, depending on the assumptions one can make. Thus, the developments in structure learning algorithms will automatically improve the performance of this online structure learning algorithm. Besides, it is essential to note that the slowest part of the algorithm is the Causal Model Change Detector part. Our development in this study was only for Causal Model Learner part, not for CMCD. CML was already the quickest part of the method. Therefore, an improvement in the CMCD part will make significantly more contribution than those of structure learning part. For example, the fitness between the new coming data point and the current estimated (weighted) covariance matrix is given by the Mahalanobis distance in the desired algorithm. Therefore, an alternative distance measure algorithm may perform differently.

OFCI, FOFCI and RFOFCI can track sufficient statistics for a linear Gaussian system efficiently. This problem is much harder for categorical/discrete variables or non-linear systems, as there will typically not be any compact representation of the sufficient statistics. One potential advantage of this approach is a way to

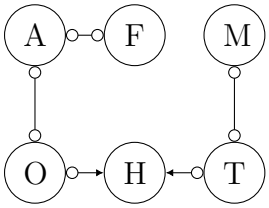
learn conditional independence constraints in an online fashion, and then those constraints can be fed into any structure learning algorithm we want.

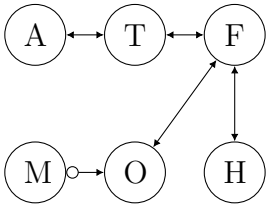
Appendix A

Real-World Data Structure Change Process

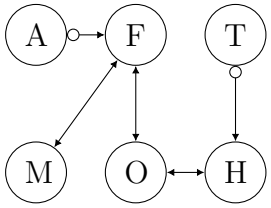


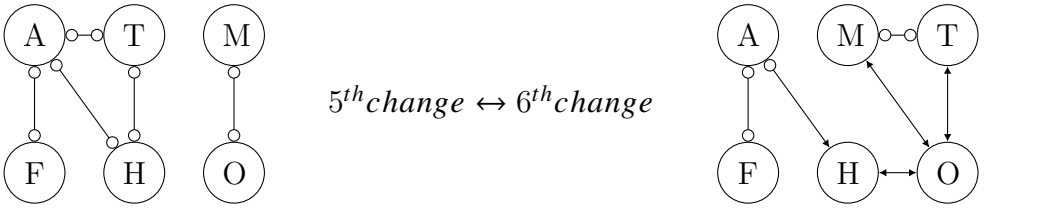
$1^{st} change \leftrightarrow 2^{nd} change$

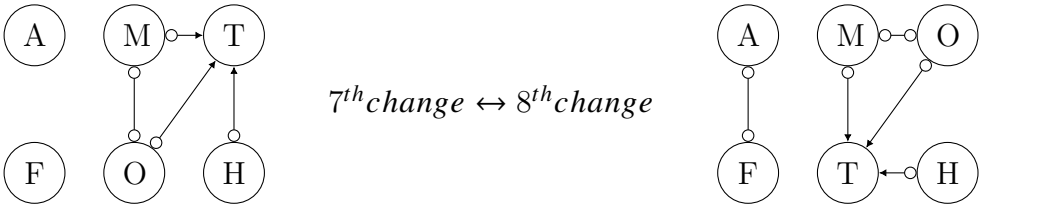


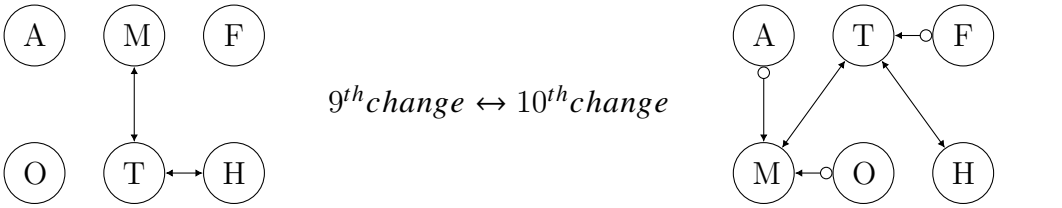


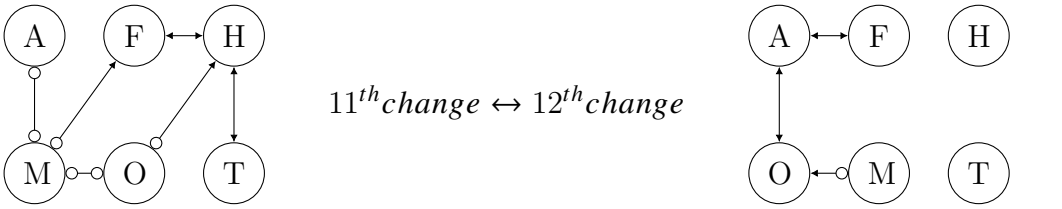
$3^{rd} change \leftrightarrow 4^{th} change$

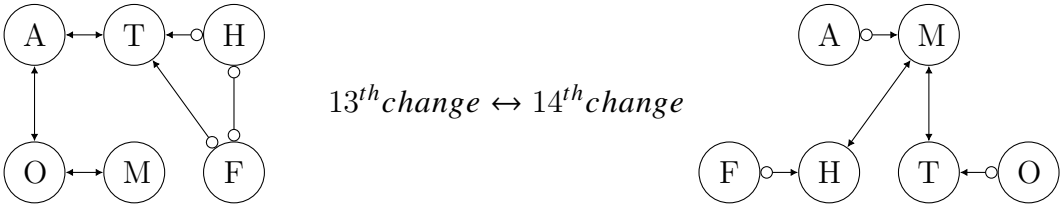


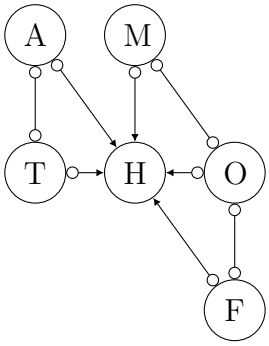

A
T
M
F
H
O
$5^{th}change \leftrightarrow 6^{th}change$
A
M
T
F
H
O


A
M
T
F
O
H
$7^{th}change \leftrightarrow 8^{th}change$
A
M
O
F
T
H

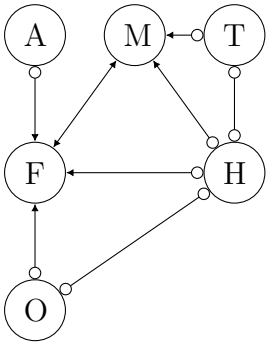

A
M
F
O
T
H
$9^{th}change \leftrightarrow 10^{th}change$
A
T
F
M
O
H

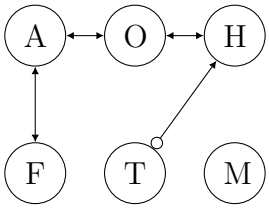

A
F
H
M
O
T
$11^{th}change \leftrightarrow 12^{th}change$
A
F
H
O
M
T


A
T
H
O
M
F
$13^{th}change \leftrightarrow 14^{th}change$
A
M
F
H
T
O

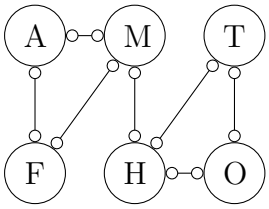


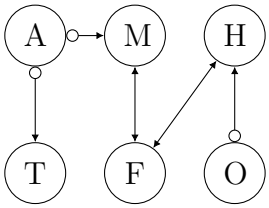
$15^{th} change \leftrightarrow 16^{th} change$



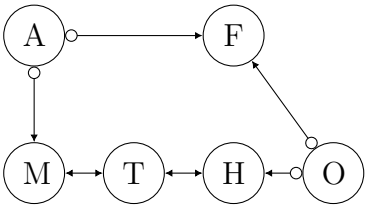


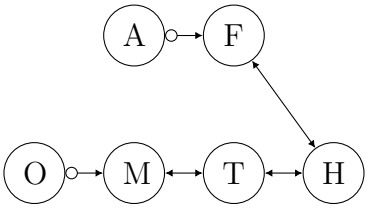
$17^{th} change \leftrightarrow 18^{th} change$



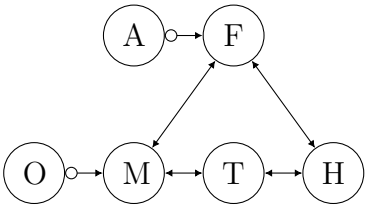


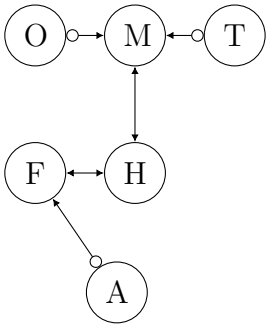
$19^{th} change \leftrightarrow 20^{th} change$



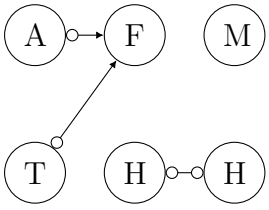


$21^{th} change \leftrightarrow 22^{th} change$

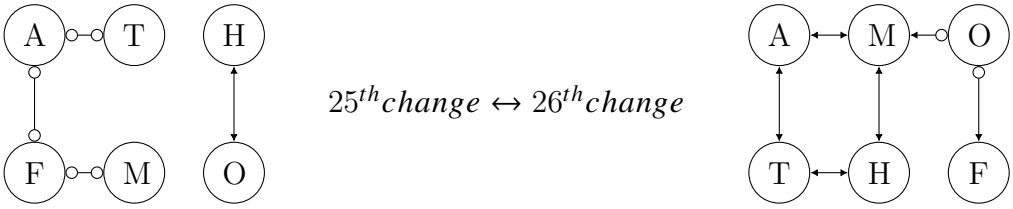




$23^{th} change \leftrightarrow 24^{th} change$



Appendix A. Real-World Data Structure Change Process



Appendix B

Online Algorithms Matlab Code (created it ourself)

B.1 FCI Algorithm Matlab Code (created it ourself)

Listing B.1: FCI Algorithm Matlab Code (created it ourself) [83], which we implemented from R FCI algorithm in pcalg package [31]

```
1  %I have rewrite and fixed possible d separation part
2  function [graph, sepset, skeleton] = Algorithm_FCI(corlength, cormatrix, ...
       samplesize, alpha, verbose)
3  % LEARN_STRUCT_PDAG_PC Learn a partially oriented DAG (pattern) using the PC ...
       algorithm
4  % P = learn_struct_pdag_pc(cond_indep, n, k, ...)
5  % n is the number of nodes.
6  % k is an optional upper bound on the fan-in (default: n)
7  % cond_indep is a boolean function that will be called as follows:
8  %   feval(cond_indep, x, y, S, ...)
9  % where x and y are nodes, and S is a set of nodes (positive integers),
10 % and ... are any optional parameters passed to this function.
11 % The output P is an adjacency matrix, in which
12 % P(i,j) = -1 if there is an i->j edge.
```

```
% P(i,j) = P(j,i) = 1 if there is an undirected edge i <-> j
% The PC algorithm does structure learning assuming all variables are observed.
% See Spirtes, Glymour and Scheines, "Causation, Prediction and Search", 1993, ...
    p117.
% This algorithm may take O(n^k) time if there are n variables and k is the max ...
    fan-in,
% but this is quicker than the Verma-Pearl IC algorithm, which is always O(n^n).
[G, sepset, complexity_timer]=Algorithm_OFCI_Skeleton_Search(cormatrix, ...
    corlength, alpha, samplesize, verbose);
G=G.*1;
[¬, unfTripl, sepset]=pc_cons_internV2(G, cormatrix, sepset, samplesize, alpha, ...
    verbose);
pag = R0_V4(G, sepset, unfTripl, verbose);
[pag, sepset, indtestnumber] = pdsep_V2(pag, sepset, corlength, alpha, verbose, ...
    cormatrix, samplesize);
skeleton=pag;
graph = R0_V4_2(pag, sepset, verbose);
flag=1;
while(flag)
    flag=0;
    [graph, flag] = R1(graph, flag, verbose);
    [graph, flag] = R2(graph, flag, verbose);
    [graph, flag] = R3(graph, flag, verbose);
    [graph, flag] = R4(graph, sepset, flag, verbose);
    [graph, flag] = R8(graph, flag, verbose);
    [graph, flag] = R9(graph, flag, verbose);
    [graph, flag] = R10(graph, flag, verbose);
end
total=complexity_timer+indtestnumber;
fprintf('the total number of independence test = %d \n', total);
end

function [graph, flag] = R1(graph, flag, verbose)
% If x*->yo-*c and x,z not adjacent ==> x*->y->z
[Xs,Ys] = find(graph == 2 & graph' ≠ 0);
ind=[Xs Ys];
for i = 1:length(ind)
    a = ind(i,1);
    b = ind(i,2);
    tmp1 = intersect(find(graph(b,:) ≠ 0), find(graph(:,b)==1));
    tmp2 = intersect(find(graph(a,:) == 0), find(graph(:,a) == 0));
```

```
49      indC = intersect(tmp1,tmp2);
50      indC = setdiff(indC, a);
51  if (¬isempty(indC))
52  for j = indC
53  if verbose
54     fprintf('\nRule 1'); fprintf('\n Orient: %d', a); fprintf(' *-> %d',b); ...
           fprintf('o-* %d', j); fprintf(' as: %d -> ', b); fprintf(' %d ', j); ...
           fprintf(' \n ');
55  end
56     graph(b,j) = 2;
57     graph(j,b) = 3;
58     flag = 1;
59  end
60  end
61  end
62  end
63
64  function [graph, flag] = R2(graph, flag, verbose)
65  % If x*->yo-*c and x,z not adjacent ==> x*->y->z
66  [Xs,Zs] = find(graph == 1 & graph' ≠ 0);
67  ind=[Xs Zs];
68  for i = 1:size(ind,1)
69      a = ind(i,1);
70      c = ind(i,2);
71      tmp1 = intersect(find(graph(a,:) == 2), find(graph(:, a) == 3));
72      tmp2 = intersect(find(graph(c,:) ≠ 0), find(graph(:, c) == 2));
73      tmp12 = intersect(tmp1,tmp2);
74      tmp3 = intersect(find(graph(a,:) == 2), find(graph(:, a) ≠ 0));
75      tmp4 = intersect(find(graph(c,:) == 3), find(graph(:, c) == 2));
76      tmp34 = intersect(tmp3,tmp4);
77  if (¬isempty(tmp12)||¬isempty(tmp34))
78  if verbose
79     fprintf('\nRule 2');
80     fprintf('\n Orient: %d -> anynode', a); fprintf('*-> %d ', c);
81     fprintf(' or ');
82     fprintf('%d *-> anynode',a); fprintf('-> %d ', c);
83     fprintf(' with %d *-o %d ', a, c);
84     fprintf('as: %d *-> %d \n', a, c);
85  end
86     graph(a,c) = 2;
87     flag = 1;
```

```
88  end
89  end
90  end
91
92  function [graph, flag] = R3(graph, flag, verbose)
93  % If x*->y<-*z, x*-o8o-*z, x,z not adjacent, 8*-oy ==> 8*->y
94  [Ths, Ys] = find(graph == 1);
95  nedges = length(Ths);
96  for i = 1:nedges
97      a = find(graph(:,Ths(i)) == 1 & graph(:,Ys(i)) == 2);
98      len = length(a);
99      f = false;
100 for j = 1:len
101 for k = j+1:len
102 if(graph(a(j),a(k)) == 0 && graph(Ths(i),Ys(i)) == 1)
103 if verbose
104    fprintf('\nRule 3'); fprintf(' Orient: %d', Ys(i)); fprintf(' *-> %d\n', Ths(...
           i));
105 end
106    graph(Ths(i),Ys(i)) = 2;
107    flag = 1;
108    f = true;
109 break;
110 end
111 end
112 if(f)
113 break;
114 end
115 end
116 end
117 end
118
119 function [graph, flag] = R4(graph, sepset, flag, verbose)
120 % Start from some node X, for node Y
121 % Visit all possible nodes X*->V & V->Y
122 % For every neighbour that is bi-directed and a parent of Y, continue
123 % For every neighbour that is bi-directed and o-*Y, orient and if
124 % parent continue
125 % Total: n*n*(n+m)
126 % For each node Y, find all orientable neighbours W
127 % For each node X, non-adjacent to Y, see if there is a path to some
```

```
128 % node in W
129 % Create graph as follows:
130 % for X,Y
131 % edges X*->V & V -> Y --> X -> V
132 % edges A <-> B & A -> Y --> A -> B
133 % edges A <-* W & A -> Y --> A->W
134 % discriminating: if path from X to W
135 [rows,cols] = find(graph ≠ 0 & graph' == 1);
136 ind=[rows cols];
137 while (¬isempty(ind))
138       b = ind(1,1);
139       c = ind(1,2);
140       ind(1,:) = [];
141       tmp1 = intersect(find(graph(b,:) == 2), find(graph(:, b) ≠  0));
142       tmp2 = intersect(find(graph(c,:) == 3), find(graph(:, c) == 2));
143       indA = intersect(tmp1,tmp2);
144 while (¬isempty(indA) && graph(c, b) == 1)
145       a = indA(1);
146       indA(1)=[];
147       Done = false;
148 while ((¬Done) && (graph(a, b) ≠ 0) && (graph(a,c) ≠ 0) && (graph(b, c) ≠ 0))
149       md_path = minDiscrPath(graph, a, b, c);
150       N_md = length(md_path);
151 if (N_md == 1)
152       Done = true;
153 else
154 if (ismember(b, sepset{md_path(1), md_path(N_md)}) || ismember(b, sepset{md_path...
        (N_md), md_path(1)}))
155 if verbose
156    fprintf('\nRule 4'); fprintf('\nThere is a discriminating path between %d', ...
          md_path(1)); fprintf('and %d', c); fprintf('for %d', b); fprintf(',and %d...
          ', b); fprintf('is in Sepset of %d', c); fprintf('and %d', md_path(1)); ...
          fprintf('. Orient: %d', b); fprintf('-> %d \n', c);
157 end
158    graph(b, c) = 2;
159    graph(c, b) = 3;
160    flag = 1;
161 else
162 if verbose
163    fprintf('\nRule 4'); fprintf('\nThere is a discriminating path between %d', ...
          md_path(1)); fprintf('and %d', c); fprintf('for %d', b); fprintf(',and %d...
```

```
          ', b); fprintf('is in Sepset of %d', c); fprintf('and %d', md_path(1)); ...
          fprintf('.Orient: %d', a); fprintf('<-> %d', b); fprintf('<-> %d \n', c);
end
    graph(a, b) = 2;
    graph(b, c) = 2;
    graph(c, b) = 2;
    flag = 1;
end
    Done = true;
end
end
end
end
end%function

function [graph,flag] = R8(graph, flag, verbose)
[r,c] = find(graph == 2 & graph' == 1);
nedges = length(r);
for i = 1:nedges
     out = find(graph(:,r(i)) == 3);
if(any(graph(out,c(i)) == 2 & graph(c(i),out)' == 3))
if verbose
    fprintf('\nRule 8'); fprintf('\nOrient: %d', r(i)); fprintf(' -> %d', out); ...
          fprintf(' -> %d', c(i)); fprintf('or %d', r(i)); fprintf('-o %d', out); ...
          fprintf('-> %d', c(i)); fprintf('with %d', r(i)); fprintf('o-> %d', c(i))...
          , fprintf('as %d', r(i)); fprintf(' -> %d \n', c(i));
end
    graph(c(i),r(i)) = 3;
    flag = 1;
end
end
end

function [graph,flag] = R9(graph, flag, verbose)
% unshieldedTriples=[];
% R9: Equivalent to orienting X <-o Y as X <-> Y and checking if Y is an
% ancestor of X (i.e. there is an almost directed cycle)
[row1,col1] = find(graph == 2 & graph' == 1);
ind=[row1 col1];
nedges = length(row1);
for i = 1:nedges
```

```
200      a = row1(i); c = col1(i);
201      ind(1,:)=[];
202      indB=find((graph(a,:) == 2 | graph(a,:) == 1) & (graph(:,a)' == 1 | graph(:,...
             a)' == 3) & (graph(c,:) == 0 & graph(:,c)' == 0));
203      indB=setdiff(indB, c);
204  while ((¬isempty(indB)) && (graph(c, a) == 1))
205      b = indB(1);
206      indB(1) = [];
207      upd = minUncovPdPath(graph, a, b, c);
208  if (length(upd) > 1)
209      graph(c, a) = 3;
210  if verbose
211     fprintf('\nRule 9'); fprintf('\nThere exists an uncovered potentially ...
           directed path between %d and %d', a, c); fprintf('. Orient: %d  -> %d \n'...
           , a, c);
212  end
213     flag = 1;
214  end
215  end
216  end%for i=nedges
217  end%function
218
219  function [graph,flag] = R10(graph, flag, verbose)
220  [rows,cols] = find(graph == 2 & graph' == 1);
221  ind = [rows cols];
222  while (¬isempty(ind))
223        a = ind(1,1);
224        c = ind(1,2);
225        ind(1,:) = [];
226        [¬,indB] = find((graph(c, :) == 3 & graph(:, c) == 2));
227  if (length(indB) ≥ 2)
228        counterB = 0;
229  while ((counterB < length(indB)) && (graph(c,a) == 1))
230        counterB = counterB + 1;
231        b = indB(counterB);
232        indD = mysetdiff(indB, b);
233        counterD = 0;
234  while ((counterD < length(indD)) && (graph(c, a) == 1))
235        counterD = counterD + 1;
236        d = indD(counterD);
```

```
237 if ((graph(a, b) == 1 || graph(a, b) == 2) && (graph(b, a) == 1 || graph(b, a) ...
        == 3) && (graph(a, d) == 1 || graph(a, d) == 2) && (graph(d, a) == 1 || ...
        graph(d, a) == 3) && graph(d, b) == 0 && graph(b, d) == 0)
238 if verbose
239    fprintf('\nRule 10 '); fprintf('\nOrient: %d', a); fprintf('-> %d \n', c);
240 end
241    flag = 1;
242    graph(c, a) = 3;
243 end
244 end
245 end
246 end
247 end
248 end
```

B.2 FCI Algorithm initial skeleton search algorithm Matlab Code (created it ourself)

Listing B.2: FCI Algorithm initial skeleton search algorithm Matlab Code (created it ourself) [83], which we implemented from R pcalg package [31]

```
1  % > skeleton
2  % function (suffStat, indepTest, alpha, labels, p, method = c("stable",
3  %"original", "stable.fast"), m.max = Inf, fixedGaps = NULL,
4  %fixedEdges = NULL, NAdelete = TRUE, numCores = 1, verbose = FALSE) {
5  function [G, sepset, complexity_timer]=Algorithm_OFCI_Skeleton_Search(cormatrix,...
       p, alpha, n, verbose)
6  complexity_timer=0;
7  %seq_p <- seq_len(p)
8  seq_p=1:p;
9  %G <- matrix(TRUE, nrow = p, ncol = p)
10 G=true(p,p);
11 %diag(G) <- FALSE
12 G=setdiag(G,false);
13 %fixedEdges <- matrix(rep(FALSE, p * p), nrow = p, ncol = p)
14 fixedEdges=zeros(p,p);
15 %sepset <- lapply(seq_p, function(.) vector("list", p))
16 sepset = cell(p,p);
```

```
%done <- FALSE
done=false;
%ord <- 0L
ord=0;
%while (!done && any(G) && ord ≤ m.max) {
while (¬done && ¬isempty(nonzeros(G)))
%n.edgetests[ord1 <- ord + 1L] <- 0
ord1 = ord+1;
done=true;
%ind <- which(G, arr.ind = TRUE)
[X,Y]=find(G);
%ind <- ind[order(ind[, 1]), ]
ind=sortrows([X Y],1);
%remEdges <- nrow(ind)
remEdges=length(ind);
%if ord==0
%fprintf('Order= %d ', ord); fprintf(' remaining edges: %d \n', remEdges);
G_1 = G;
%for (i in 1:remEdges) {
for i=1:remEdges
%for i= 4:6
%x <- ind[i, 1]
x = ind(i, 1);
%y <- ind[i, 2]
y = ind(i, 2);
%if (G[y, x] && !fixedEdges[y, x]) {
if (G(y,x) && ¬fixedEdges(y, x))
%nbrsBool <- G[, x]
nbrsBool = G_1(:,x);
%nbrsBool[y] <- FALSE
nbrsBool(y)=false;
%nbrs <- seq_p[nbrsBool]
nbrs = seq_p(nbrsBool);
%length_nbrs <- length(nbrs)
length_nbrs = length(nbrs);
%if (length_nbrs ≥ ord) {
if (length_nbrs ≥ ord)
%if (length_nbrs > ord)
%done <- FALSE
done = false;
%S <- seq_len(ord)
```

```
58 S = 1:ord;
59 %repeat {
60 %while
61 deneme=false;
62 while ¬deneme
63        deneme=true;
64 %n.edgetests[ord1] <- n.edgetests[ord1] + 1
65        pval = gaussCItest(x, y, nbrs(S), cormatrix, n);
66        complexity_timer=complexity_timer+1;
67 if isempty(nbrs(S))
68 if verbose
69        fprintf('x= %d indep of y= %d given S= ', x, y); fprintf(' p= %d \n', ...
              pval);
70 end
71 else
72 if verbose
73    fprintf('x= %d indep of y= %d given ', x, y); fprintf(' S= %d ', nbrs(S)); ...
          fprintf(' p= %d \n', pval);
74 end
75 end
76 %fprintf('\n');
77 %x= 1  y= 2  S=  : pval = 0
78 %if (pval ≥ alpha) {
79 if (pval ≥ alpha)
80 %G[x, y] <- G[y, x] <- FALSE
81 G(x, y) = false;
82 G(y, x) = false;
83 %sepset[[x]][[y]] <- nbrs[S]
84 sepset{x,y} = myunion(sepset{x,y}, nbrs(S));
85 %fprintf('x= %d and y= %d separation set is: ', x, y); fprintf(' S= %d \n', nbrs...
       (S));
86 %break}
87 break;
88 %else {
89 else
90 %nextSet <- getNextSet(length_nbrs, ord, S)
91 [nextSet, wasLast] = getNextSet(length_nbrs, ord, S);
92 %if (nextSet$wasLast)0
93 if (wasLast)
94 %break
95 break;
```

```
96  else
97  %S <- nextSet$nextSet
98  S =  nextSet;
99  deneme=false;
100 end
101 end
102 end%while ¬deneme
103 end
104 end
105 end
106 %ord <- ord + 1L
107 ord = ord +1;
108 end
109 end
```

B.3 OFCI Algorithm Matlab Code (created it ourself)

Listing B.3: OFCI algorithm Matlab Code (created it ourself)

```
1  %%FCI rule 9 changed with using source r fci code
2  %%fci changed
3  function output = Algorithm_OFCI(data, schedule, alpha, batch, plotgraphs, ...
       verbose)
4  %initialize
5  skelstore={};
6  sepsetstore={};
7  datasize=size(data);
8  mu=mean(data(1:10,:));  %zeros(1,datasize(2));
9  %mu=mean(data);
10 %d=zeros(1,datasize(2));
11 tcov=round(cov(data(1:10,:)),8);
12 %tcov=cov(data);
13 output.graphtimes=[];
14 output.graphs{1}=[];
15 a=ones(1,datasize(1));
16 b=ones(1,datasize(1));
17 b(1)=2;
```

```
18 sample_size=ones(1,datasize(1));
19 %sensitivity for graph search
20 %alpha=.05;
21 tcovstore={};
22 tcorstore={};
23 pdag_count=0;
24 M_error=ones(1,datasize(1))*datasize(2);
25 pverr=ones(1,datasize(1));
26 experrvalpr=.1;
27 experrval=ones(1,datasize(1))*experrvalpr; %expected error value (unbiased ...
       estimated based on observations)
28 %current sample size is also used, but is initialized separately
29 trigger=0;
30 %for CMCD-Causal Model CHANGE DETECTOR
31 initstbias=0;  %this is something like an initial stability bias
32 pval=ones(1,datasize(1))*.5;
33 ntrack=zeros(1,datasize(1));
34 Q=ones(1,datasize(1))*initstbias;
35 sumsqrw=ones(1,datasize(1));
36 poolp=zeros(1,datasize(1));
37 burnin=10;%burnin=datasize(2)*1.05;  %this determines the length of the burn-in ...
       period %!_!_!_using parfindFOUR
38 burnin_MD=chi2inv(.5,datasize(2));  %this is the Mahalanobis Distance to use ...
       during the burn-in period
39 plearn=zeros(1,datasize(1));
40 make_graph=0;
41 fol=.005; %frequency of learning parameter, for probabilistic scheduler.
42 %scale and lower bound parameter for transforming poolp values to weights
43 scpara=.95;   %normal parameter: .95
44 ratpar=3;  %!_!_!_using parfindFOUR  %normal parameter: 1.5
45 %parameter for ratio-type downweighting.  as ratpar ->1, curve steapens/...
       downweights more heavily.
46 %also determines maximum downweight ratio, equal to 1/ratpar (i.e.,
47 %ratpar=1 downweights to an effective sample size of 0 at poolp(j)=1,
48 %ratpar=2 cuts effective sample size in half at poolp(j)=1
49 for j=1:datasize(1)
50 %calc accumulating error rate of correlation
51 %use Mahalanobis error to calc error of new point from old tcov and mu
52 %prob want regular M_error here, not normed error.  Take account for
53 %datasize(2) in the distributional part.
54 %M_error(j) = (data(j,:)-mu)*inv(tcov)*(data(j,:)-mu)';
```

```
%__trying it a new way___
M_error(j) = (data(j,:)-mu)/tcov*(data(j,:)-mu)';
norm_M_error(j)=M_error(j)/datasize(2);
%Update tcovariance Matrix
%use learning rate to update tcov
%replaces commented out update of b(j+1) below
if j>1
   b(j)=b(j-1)+a(j);
end
%regular OCME-Online Covariation Matrix Estimation
   d=(a(j)/b(j))*(data(j,:)-mu);
   mu=mu+d;
for i=1:datasize(2)
  for k=1:datasize(2)
      tcov(i,k)=(1/b(j))*((b(j)-a(j))*tcov(i,k)+(b(j)-a(j))*d(i)*d(k)+a(j)*(data...
          (j,i)-mu(i))*(data(j,k)-mu(k)));
  end
end
tcovstore{j}=tcov;
%update learning rate
%need to track the weighted sum of M_error values, and compare this
%against a distribution which depends on: sample size, datasize(2)
if j>1
   sample_size(j)=(a(j-1)/a(j))*sample_size(j-1)+1;
else
   sample_size(j)=1;
end
%not sample size, actually.  need to track sum of _squared_ weights directly.
%P = normcdf(X,mu,sigma)
%P = chi2cdf(X,V)
%X = norminv(P,mu,sigma)
%calc...  norminv(chi2cdf(M_error(j),datasize(2)),0,1)
%if j>1
%the min is to prevent ntrack(j)=Inf, which causes HUGE PROBLEMS
%ntrack(j)=norminv(min(chi2cdf(M_error(j),datasize(2)),.999),0,1);
%Q(j)=Q(j-1)+a(j)*ntrack(j);
%sumsqrw(j)=sumsqrw(j-1)+a(j)^2;
%poolp(j)=normcdf(Q(j),0,sqrt(sumsqrw(j)));
%during the burn-in period:
if j>1&&sample_size(j-1)≤burnin
%gotta make sure the right things get burned in
```

```
95     pval(j)=.5;
96  %the min is to prevent ntrack(j)=Inf, which causes HUGE PROBLEMS
97     ntrack(j)=norminv(min(pval(j),.999),0,1);
98     Q(j)=Q(j-1)+a(j)*ntrack(j);
99     sumsqrw(j)=sumsqrw(j-1)+a(j)^2; %sum of squared weights
100    poolp(j)=normcdf(Q(j),0,sqrt(sumsqrw(j)));
101    a(j+1)=a(j);
102 if trigger==1
103    experrval(j)=(experrval(j-1)*(sample_size(j)-1)+chi2cdf(M_error(j),datasize...
           (2))-.5)/(sample_size(j));
104 else
105    experrval(j)=experrvalpr;
106 end
107    pcheck=1;
108    trigger=1;
109 end
110 %after the burn-in period is over:
111 if j>1&&isnan(M_error(j))==0&&M_error(j)≥0&&sample_size(j-1)>burnin  %j>burnin ...
       for the burn-in period
112    trigger=0;
113 %Calculating pooled p values and turning them into weights
114 if poolp(j-1)≤0
115    pcheck=0;
116 end
117 %experrval(j)=(experrval(j-1)*(sample_size(j)-1)+chi2cdf(M_error(j),datasize(2))...
       -(.5+pverr(j-1)))/(sample_size(j));
118 %pverr(j)=max(min((priorss*priorerrval+sample_size(j)*experrval(j))/(priorss + ...
       sample_size(j)),pverr(j-1)),0);
119 pval(j)=fcdf((sample_size(j)-datasize(2))/(datasize(2)*(sample_size(j)-1))*...
       M_error(j),datasize(2),sample_size(j)-datasize(2));
120 %pval(j)=max(chi2cdf(M_error(j),datasize(2))-min(pverr(j),pcheck),0);   %***...
       Rewrite this in terms of F-distribution/hotelling's t-square***
121 %pval(j)=chi2cdf(M_error(j)-.03*12*1/sqrt(sample_size(j-1))*20,datasize(2));
122 %the min/maxes are to prevent ntrack(j)=Inf, which causes HUGE PROBLEMS
123 ntrack(j)=norminv(max(min(pval(j),.9999),.0001),0,1); %inverse normal cdf of the...
        pvalue
124 Q(j)=Q(j-1)+a(j)*ntrack(j); %weighted sum of inverse normal cdf of p-values
125 sumsqrw(j)=sumsqrw(j-1)+a(j)^2; %sum of squared weights
126 poolp(j)=normcdf(Q(j),0,sqrt(sumsqrw(j))); %pooled pvalue is the appropriate ...
       normal cdf of Q(j)
127 %plearn(j)=plearn(j-1)+fol*poolp(j);%fol is frequency param, square for scaling
```

```
128 plearn(j)=plearn(j-1)+fol*(poolp(j)-poolp(j)*plearn(j-1));%fol is frequency ...
        param
129 %this needs to be squashed so that a regular-ish p-value (.5)
130 %doesn't cause massive downweighting.
131 if poolp(j)<scpara
132    a(j+1)=a(j);
133 else
134    a(j+1)=1/(1-1/ratpar*((1/(1-scpara))*poolp(j)-(1/(1-scpara)-1)))*a(j);
135 %previous versions:
136 %the min is to prevent sample sizes from getting below a
137 %certain value, since bad stuff happens if it does
138 %a(j+1)=max((expinv((1/(1-scpara))*poolp(j)-(1/(1-scpara)-1))+1)*a(j),a(j));
139 %a(j+1)=min(b(j)*.02,max((expinv((1/(1-scpara))*poolp(j)-(1/(1-scpara)-1))^(.5)...
        +1)*a(j),a(j)));
140 end
141 %a(j+1)=max(expinv(poolp(j)^2)*b(j),a(j));  %expinv is for rescaling the poolp.
142 %____threshold method____
143 %if poolp(j)<.99
144 %a(j+1)=a(j);
145 %a(j+1)=b(j)*.1;
146 end
147 %learn PDAG matrix from the correlation matrices
148 if isempty(schedule)
149 %probabilistic scheduler
150 if rand(1)<plearn(j) && j>24
151    make_graph=1;
152 %Δ_alpha_MTDL=0;
153    plearn(j)=0;
154 end
155 else
156 %Do full run for particular time steps
157 if ismember(j,schedule)==1
158    make_graph=1;
159 else
160    make_graph=0;
161 end
162 end
163 %PC search for graph, then plot it
164 if make_graph==1
165 %calc correlations
166    [¬,ExpCorrC] = cov2corr(tcov);
```

```
167     pdag_count=pdag_count+1; %index the pdags amongst themselves
168  %pdag_index(pdag_count)=j; %index the pdags amongst the timesteps
169  %use the bayes net toolbox to calculate the pdag matrix
170  %pdag{pdag_count} = learn_struct_pdag_pc('cond_indep_fisher_z', length(cor), ...
         length(cor), cor, floor(sample_size(j)), alpha);
171  %plot pdag
172     output.graphtimes(pdag_count)=j;
173  %yes=floor(sample_size(j))
174     tcorstore{pdag_count}=ExpCorrC;
175  %___uncomment the below eventually!!!!!!!!!!!!!!!!!!!!____
176     [output.graphs{pdag_count}, sepset, skeleton]=Algorithm_FCI(length(ExpCorrC),...
             ExpCorrC, floor(j), alpha, verbose);
177  %output.graphs{pdag_count}=ones(10);
178     sepsetstore{pdag_count}=sepset;
179     skelstore{pdag_count}=skeleton;
180  if plotgraphs
181     figure('NextPlot','new')
182     draw_graph_pag(abs(output.graphs{pdag_count}))
183     title(j)
184  end
185     make_graph=0;
186  end
187  end
188  %comparing poolp w/ batch equivalent weighting to the known analytic
189  %solution: chi-square distribution with DOF=#data*variables
190  x=0;
191  chisquaretest=zeros(1,datasize(1));
192  for j=1:length(M_error)
193      if j<burnin
194          x=x+burnin_MD;
195      else
196          x=x+M_error(j);
197      end
198      chisquaretest(j)=chi2cdf(x,j*datasize(2));
199  end
200  if ¬batch
201      plot(sample_size)
202      title('sample size')
203      figure('NextPlot','new')
204      plot(norm_M_error,'red')
205      hold on
```

```
206      plot(MTDL)
207      hold off
208      figure('NextPlot','new')
209      plot(poolp)
210      hold on
211      plot(chisquaretest,'red')
212      title('Comparing pooled p-value to batch analytic solution')
213      legend('pooled p-value','analytic p-value')
214      hold off
215      figure('NextPlot','new')
216      plot(pval)
217      title pval
218      output.skelstore=skelstore;
219      output.tcorstore=tcorstore;
220      output.sepsetstore=sepsetstore;
221      output.sumsqrw=sumsqrw;
222      output.poolp=poolp;
223      output.chisquaretest=chisquaretest;
224      output.Q=Q;
225      output.a=a;
226      output.M_error=M_error;
227      output.ntrack=ntrack;
228      output.tcovstore=tcovstore;
229      output.b=b;
230      output.sample_size=sample_size;
231      output.pverr=pverr;
232      output.experrval=experrval;
233      output.plearn=plearn;
234      output.pval=pval;
235      if isempty(schedule)
236          output.plearn=plearn;
237      end
238  end
```

B.4 Modified FCI Algorithm Matlab Code (created it ourself)

Listing B.4: Modified FCI Algorithm Matlab Code (created it ourself)

Appendix B. Online Algorithms Matlab Code (created it ourself)

```
%I have rewrite and fixed possible d separation part
function [graph, sepset, skeleton] = Algorithm_Modified_FCI(sepset, corlength, ...
    cormatrix, samplesize, alpha, verbose)
% LEARN_STRUCT_PDAG_PC Learn a partially oriented DAG (pattern) using the PC ...
    algorithm
% P = learn_struct_pdag_pc(cond_indep, n, k, ...)
% n is the number of nodes.
% k is an optional upper bound on the fan-in (default: n)
% cond_indep is a boolean function that will be called as follows:
% feval(cond_indep, x, y, S, ...)
% where x and y are nodes, and S is a set of nodes (positive integers),
% and ... are any optional parameters passed to this function.
% The output P is an adjacency matrix, in which
% P(i,j) = -1 if there is an i->j edge.
% P(i,j) = P(j,i) = 1 if there is an undirected edge i <-> j
% The PC algorithm does structure learning assuming all variables are observed.
% See Spirtes, Glymour and Scheines, "Causation, Prediction and Search", 1993, ...
    p117.
% This algorithm may take O(n^k) time if there are n variables and k is the max ...
    fan-in,
% but this is quicker than the Verma-Pearl IC algorithm, which is always O(n^n).
[G, sepset, complexity_timer]=Algorithm_FOFCI_Online_Skeleton_Search(sepset, ...
    cormatrix, corlength, alpha, samplesize, verbose);
G=G.*1;
[¬, unfTripl, sepset]=pc_cons_internV2(G, cormatrix, sepset, samplesize, alpha, ...
    verbose);
 pag = R0_V4(G, sepset, unfTripl, verbose);
[pag, sepset, indtestnumber] = pdsep_V2(pag, sepset, corlength, alpha, verbose, ...
    cormatrix, samplesize);
skeleton=pag;
 graph = R0_V4_2(pag, sepset, verbose);
flag=1;
while(flag)
    flag=0;
    [graph, flag] = R1(graph, flag, verbose);
    [graph, flag] = R2(graph, flag, verbose);
    [graph, flag] = R3(graph, flag, verbose);
    [graph, flag] = R4(graph, sepset, flag, verbose);
    [graph, flag] = R8(graph, flag, verbose);
    [graph, flag] = R9(graph, flag, verbose);
```

```
34      [graph, flag] = R10(graph, flag, verbose);
35  end
36  total=complexity_timer+indtestnumber;
37  fprintf('the total number of independence test = %d \n', total);
38  end
39
40  function [graph, flag] = R1(graph, flag, verbose)
41  % If x*->yo-*c and x,z not adjacent ==> x*->y->z
42  [Xs,Ys] = find(graph == 2 & graph' ~= 0);
43  ind=[Xs Ys];
44  for i = 1:length(ind)
45      a = ind(i,1);
46      b = ind(i,2);
47      tmp1 = intersect(find(graph(b,:) ~= 0), find(graph(:,b)==1));
48      tmp2 = intersect(find(graph(a,:) == 0), find(graph(:,a) == 0));
49      indC = intersect(tmp1,tmp2);
50      indC = setdiff(indC, a);
51  if (~isempty(indC))
52  for j = indC
53  if verbose
54     fprintf('\nRule 1'); fprintf('\n Orient: %d', a); fprintf(' *-> %d',b); ...
           fprintf('o-* %d', j); fprintf(' as: %d -> ', b); fprintf(' %d ', j); ...
           fprintf(' \n ');
55  end
56     graph(b,j) = 2;
57     graph(j,b) = 3;
58     flag = 1;
59  end
60  end
61  end
62  end
63
64
65  function [graph, flag] = R2(graph, flag, verbose)
66  % If x*->yo-*c and x,z not adjacent ==> x*->y->z
67  [Xs,Zs] = find(graph == 1 & graph' ~= 0);
68  ind=[Xs Zs];
69  for i = 1:size(ind,1)
70      a = ind(i,1);
71      c = ind(i,2);
72      tmp1 = intersect(find(graph(a,:) == 2), find(graph(:, a) == 3));
```

```
73      tmp2 = intersect(find(graph(c,:) ≠ 0), find(graph(:, c) == 2));
74      tmp12 = intersect(tmp1,tmp2);
75      tmp3 = intersect(find(graph(a,:) == 2), find(graph(:, a) ≠ 0));
76      tmp4 = intersect(find(graph(c,:) == 3), find(graph(:, c) == 2));
77      tmp34 = intersect(tmp3,tmp4);
78  if (¬isempty(tmp12)||¬isempty(tmp34))
79  if verbose
80     fprintf('\nRule 2');
81     fprintf('\n Orient: %d -> anynode', a); fprintf('*-> %d ', c);
82     fprintf(' or ');
83     fprintf('%d *-> anynode',a); fprintf('-> %d ', c);
84     fprintf(' with %d *-o %d ', a, c);
85     fprintf('as: %d *-> %d \n', a, c);
86  end
87     graph(a,c) = 2;
88     flag = 1;
89  end
90  end
91  end
92
93  function [graph, flag] = R3(graph, flag, verbose)
94  % If x*->y<-*z, x*-o8o-*z, x,z not adjacent, 8*-oy ==> 8*->y
95  [Ths, Ys] = find(graph == 1);
96  nedges = length(Ths);
97  for i = 1:nedges
98      a = find(graph(:,Ths(i)) == 1 & graph(:,Ys(i)) == 2);
99      len = length(a);
100     f = false;
101 for j = 1:len
102 for k = j+1:len
103 if(graph(a(j),a(k)) == 0 && graph(Ths(i),Ys(i)) == 1)
104 if verbose
105    fprintf('\nRule 3'); fprintf(' Orient: %d', Ys(i)); fprintf(' *-> %d\n', Ths(...
           i));
106 end
107    graph(Ths(i),Ys(i)) = 2;
108    flag = 1;
109    f = true;
110 break;
111 end
112 end
```

```
113 if(f)
114 break;
115 end
116 end
117 end
118 end
119
120 function [graph, flag] = R4(graph, sepset, flag, verbose)
121 % Start from some node X, for node Y
122 % Visit all possible nodes X*->V & V->Y
123 % For every neighbour that is bi-directed and a parent of Y, continue
124 % For every neighbour that is bi-directed and o-*Y, orient and if
125 % parent continue
126 % Total: n*n*(n+m)
127 %--------------------------------------------------------------------------
128 % For each node Y, find all orientable neighbours W
129 % For each node X, non-adjacent to Y, see if there is a path to some
130 % node in W
131 % Create graph as follows:
132 % for X,Y
133 % edges X*->V & V -> Y --> X -> V
134 % edges A <-> B & A -> Y --> A -> B
135 % edges A <-* W & A -> Y --> A->W
136 % discriminating: if path from X to W
137 [rows,cols] = find(graph ≠ 0 & graph' == 1);
138 ind=[rows cols];
139 while (¬isempty(ind))
140       b = ind(1,1);
141       c = ind(1,2);
142       ind(1,:) = [];
143       tmp1 = intersect(find(graph(b,:) == 2), find(graph(:, b) ≠  0));
144       tmp2 = intersect(find(graph(c,:) == 3), find(graph(:, c) == 2));
145       indA = intersect(tmp1,tmp2);
146 while (¬isempty(indA) && graph(c, b) == 1)
147       a = indA(1);
148       indA(1)=[];
149       Done = false;
150 while ((¬Done) && (graph(a, b) ≠ 0) && (graph(a,c) ≠ 0) && (graph(b, c) ≠ 0))
151       md_path = minDiscrPath(graph, a, b, c);
152       N_md = length(md_path);
153 if (N_md == 1)
```

```
154         Done = true;
155   else
156   if (ismember(b, sepset{md_path(1), md_path(N_md)}) || ismember(b, sepset{md_path...
          (N_md), md_path(1)}))
157   if verbose
158      fprintf('\nRule 4'); fprintf('\nThere is a discriminating path between %d', ...
             md_path(1)); fprintf('and %d', c); fprintf('for %d', b); fprintf(',and %d...
             ', b); fprintf('is in Sepset of %d', c); fprintf('and %d', md_path(1)); ...
             fprintf('. Orient: %d', b); fprintf('-> %d \n', c);
159   end
160      graph(b, c) = 2;
161      graph(c, b) = 3;
162      flag = 1;
163   else
164   if verbose
165      fprintf('\nRule 4'); fprintf('\nThere is a discriminating path between %d', ...
             md_path(1)); fprintf('and %d', c); fprintf('for %d', b); fprintf(',and %d...
             ', b); fprintf('is in Sepset of %d', c); fprintf('and %d', md_path(1)); ...
             fprintf('.Orient: %d', a); fprintf('<-> %d', b); fprintf('<-> %d \n', c);
166   end
167      graph(a, b) = 2;
168      graph(b, c) = 2;
169      graph(c, b) = 2;
170      flag = 1;
171   end
172      Done = true;
173   end
174   end
175   end
176   end
177   end%function
178
179   function [graph,flag] = R8(graph, flag, verbose)
180   [r,c] = find(graph == 2 & graph' == 1);
181   nedges = length(r);
182   for i = 1:nedges
183       out = find(graph(:,r(i)) == 3);
184   if(any(graph(out,c(i)) == 2 & graph(c(i),out)' == 3))
185     if verbose
186        fprintf('\nRule 8'); fprintf('\nOrient: %d', r(i)); fprintf(' -> %d', out);...
               fprintf(' -> %d', c(i)); fprintf('or %d', r(i)); fprintf('-o %d', out)...
```

```
            ; fprintf('-> %d', c(i)); fprintf('with %d', r(i)); fprintf('o-> %d', c...
            (i)), fprintf('as %d', r(i)); fprintf(' -> %d \n', c(i));
187   end
188     graph(c(i),r(i)) = 3;
189     flag = 1;
190 end
191 end
192 end
193
194 function [graph,flag] = R9(graph, flag, verbose)
195 % unshieldedTriples=[];
196 % R9: Equivalent to orienting X <-o Y as X <-> Y and checking if Y is an
197 % ancestor of X (i.e. there is an almost directed cycle)
198 [row1,col1] = find(graph == 2 & graph' == 1);
199 ind=[row1 col1];
200 nedges = length(row1);
201 for i = 1:nedges
202     a = row1(i); c = col1(i);
203     ind(1,:)=[];
204     indB=find((graph(a,:) == 2 | graph(a,:) == 1) & (graph(:,a)' == 1 | graph(:,...
            a)' == 3) & (graph(c,:) == 0 & graph(:,c)' == 0));
205     indB=setdiff(indB, c);
206 while ((¬isempty(indB)) && (graph(c, a) == 1))
207    b = indB(1);
208    indB(1) = [];
209    upd = minUncovPdPath(graph, a, b, c);
210 if (length(upd) > 1)
211    graph(c, a) = 3;
212 if verbose
213    fprintf('\nRule 9'); fprintf('\nThere exists an uncovered potentially ...
           directed path between %d and %d', a, c); fprintf('. Orient: %d  -> %d \n'...
           , a, c);
214 end
215     flag = 1;
216 end
217 end
218 end%for i=nedges
219 end%function
220
221 function [graph,flag] = R10(graph, flag, verbose)
222 [rows,cols] = find(graph == 2 & graph' == 1);
```

```
223  ind = [rows cols];
224  while (¬isempty(ind))
225        a = ind(1,1);
226        c = ind(1,2);
227        ind(1,:) = [];
228        [¬,indB] = find((graph(c, :) == 3 & graph(:, c) == 2));
229  if (length(indB) ≥ 2)
230        counterB = 0;
231  while ((counterB < length(indB)) && (graph(c,a) == 1))
232        counterB = counterB + 1;
233        b = indB(counterB);
234        indD = mysetdiff(indB, b);
235        counterD = 0;
236  while ((counterD < length(indD)) && (graph(c, a) == 1))
237        counterD = counterD + 1;
238        d = indD(counterD);
239  if ((graph(a, b) == 1 || graph(a, b) == 2) && (graph(b, a) == 1 || graph(b, a) ...
         == 3) && (graph(a, d) == 1 || graph(a, d) == 2) && (graph(d, a) == 1 || ...
         graph(d, a) == 3) && graph(d, b) == 0 && graph(b, d) == 0)
240  if verbose
241     fprintf('\nRule 10 '); fprintf('\nOrient: %d', a); fprintf('-> %d \n', c);
242  end
243     flag = 1;
244     graph(c, a) = 3;
245  end
246  end
247  end
248  end
249  end
250  end
```

B.5 FCI Algorithm online initial skeleton search algorithm Matlab Code (created it ourself)

Listing B.5: FCI Algorithm online initial skeleton search algorithm Matlab Code (created it ourself)

```
1  % > skeleton
```

```
% function (suffStat, indepTest, alpha, labels, p, method = c("stable",
% "original", "stable.fast"), m.max = Inf, fixedGaps = NULL,
% fixedEdges = NULL, NAdelete = TRUE, numCores = 1, verbose = FALSE) {
function [B, sepset, complexity_timer]=Algorithm_FOFCI_Online_Skeleton_Search(...
    sepset, cormatrix, p, alpha, n, verbose)
%seq_p <- seq_len(p)
complexity_timer=0;
seq_p=1:p;
%G <- matrix(TRUE, nrow = p, ncol = p)
B=true(p,p);
%diag(G) <- FALSE
B=setdiag(B,false);
%fixedEdges <- matrix(rep(FALSE, p * p), nrow = p, ncol = p)
fixedEdges=zeros(p,p);
done=false;
%ord <- 0L
ord=0;
dontest=cell(size(cormatrix,1));
%n.edgetests <- numeric(1)
for i=1:length(sepset)
for j=i+1:length(sepset)
    sepset{i,j}=myunion(sepset{i,j},sepset{j,i});
    sepset{j,i}=[];
end
end
[S1,S2]=find(¬cellfun('isempty', sepset));
ind_seps=sortrows([S1 S2],1);
remEdges_seps=size(ind_seps,1);
% fprintf('sepset length= %d \n', remEdges_seps);
for i=1:remEdges_seps
    s1 = ind_seps(i, 1);
    s2 = ind_seps(i, 2);
    pval = gaussCItest(s1, s2, [], cormatrix, n);
    complexity_timer=complexity_timer+1;
if (pval ≥ alpha)
if verbose
   fprintf('x= %d indep of y= %d given S= ', s1, s2); fprintf(' p= %d \n', pval)...
       ;
end
   B(s1,s2)=false;
   B(s2,s1)=false;
```

```
41      sepset{s1,s2}=[];
42  end
43  if B(s1,s2)==true
44       lngth_sp=length(sepset{s1,s2});
45       flag=0;
46  for k=1:lngth_sp
47       condSets = nchoosek(sepset{s1,s2}, k);
48       nofCondSets = size(condSets, 1);
49  for iCondSet = 1:nofCondSets
50       condSet = condSets(iCondSet, 1:k);
51       pval_2 = gaussCItest(s1, s2, condSet, cormatrix, n);
52       complexity_timer=complexity_timer+1;
53  if (pval_2 ≥ alpha)
54  if verbose
55       fprintf('s1= %d still indep of s2= %d given ', s1, s2); fprintf(' S= %d ', ...
             condSet); fprintf(' p= %d \n', pval_2);
56  end
57       B(s1,s2)=false;
58       B(s2,s1)=false;
59       sepset{s1,s2}=condSet;
60       flag=1;
61  break;
62  else
63  if verbose
64      fprintf('do not test dependency between s1= %d and s2= %d given ', s1, s2); ...
            fprintf(' S= %d', condSet); fprintf(' p= %d \n', pval_2);
65  end
66  if length(condSet)==lngth_sp
67      dontest{s1,s2}=condSet;
68  %fprintf('dont test s1= %d and s2= %d given ', s1, s2); fprintf(' S= %d \n', ...
        condSet);
69      sepset{s1,s2}=[];
70  end
71  end
72  end
73  if(flag==1)
74  break
75  end
76  end
77  end
78  end
```

```
while (¬done && ¬isempty(nonzeros(B)))
%done <- TRUE
done=true;
%ind <- which(G, arr.ind = TRUE)
[X,Y]=find(B);
%ind <- ind[order(ind[, 1]), ]
ind=sortrows([X Y],1);
%remEdges <- nrow(ind)
remEdges=length(ind);
if ord==0
   fprintf('Order= %d ', ord); fprintf(' remaining edges: %d \n', remEdges);
end
   G_l = B;
%for (i in 1:remEdges) {
for i=1:remEdges
%for i= 4:6
%x <- ind[i, 1]
x = ind(i, 1);
%y <- ind[i, 2]
y = ind(i, 2);
%if (G[y, x] && !fixedEdges[y, x]) {
if (B(y,x) && ¬fixedEdges(y, x))
%nbrsBool <- G[, x]
    nbrsBool = G_l(:,x);
%nbrsBool[y] <- FALSE
    nbrsBool(y)=false;
%nbrs <- seq_p[nbrsBool]
    nbrs= seq_p(logical(nbrsBool));
%if verbose
%fprintf('the neighbours of x= %d and y= %d given ', x, y); fprintf(' N= %d \n',...
     nbrs);
%length_nbrs <- length(nbrs)
    length_nbrs = length(nbrs);
%if (length_nbrs ≥ ord) {
if (length_nbrs ≥ ord)
%if (length_nbrs > ord)
%done <- FALSE
done = false;
%S <- seq_len(ord)
S = 1:ord;
%repeat {
```

```
119 %while
120 deneme=false;
121 while ¬deneme
122         deneme=true;
123 if ¬isempty(dontest{x,y})
124 if all(ismember(nbrs(S), dontest{x,y}))
125    [nextSet, wasLast] = getNextSet(length_nbrs, ord, S);
126 if (wasLast)
127 break;
128 else
129    S =  nextSet;
130    deneme=false;
131 end
132 end
133 end
134    pval_3 = gaussCItest(x, y, nbrs(S), cormatrix, n);
135    complexity_timer=complexity_timer+1;
136 if isempty(nbrs(S))
137 if verbose
138    fprintf('x= %d indep of y= %d given S= ', x, y); fprintf(' p= %d \n', pval_3)...
           ;
139 end
140 else
141 if verbose
142    fprintf('x= %d indep of y= %d given ', x, y); fprintf(' S= %d ', nbrs(S)); ...
           fprintf(' p= %d \n', pval_3);
143 end
144 end
145 %fprintf('\n');
146 %x= 1  y= 2  S=  : pval = 0
147 %if (pval ≥ alpha) {
148 if (pval_3 ≥ alpha)
149 %G[x, y] <- G[y, x] <- FALSE
150     B(x, y) = false;
151     B(y, x) = false;
152 %sepset[[x]][[y]] <- nbrs[S]
153     sepset{x,y} = myunion(sepset{x,y}, nbrs(S));
154 %fprintf('x= %d and y= %d separation set is: ', x, y); fprintf(' S= %d \n', nbrs...
       (S));
155 %break}
156 break;
```

```
157 %else {
158 else
159 %nextSet <- getNextSet(length_nbrs, ord, S)
160     [nextSet, wasLast] = getNextSet(length_nbrs, ord, S);
161 %if (nextSet$wasLast)0
162 if (wasLast)
163 %break
164 break;
165 else
166 %S <- nextSet$nextSet
167 S =  nextSet;
168 deneme=false;
169 end
170 end
171 end%while ¬deneme
172 end
173 end
174 end%for i=1:remEdges
175 %}}}}}%for(remedges)
176 %ord <- ord + 1L
177 ord = ord +1;
178 %}%while}}
179 end
180 end
```

B.6 FOFCI algorithm Matlab Code (created it ourself)

Listing B.6: FOFCI algorithm Matlab Code (created it ourself)

```
1 %%FCI rule 9 changed with using source r fci code
2 %%fci updated rules seperated
3 function output = Algorithm_FOFCI(data, schedule, alpha, batch, plotgraphs, ...
      verbose)
4 %initialize
5 skelstore={};
6 sepsetstore={};
7 datasize=size(data);
```

```
mu=mean(data(1:10,:));  %mu=mean(data);
tcov=round(cov(data(1:10,:)),8);%tcov=cov(data);
output.graphtimes=[];
output.graphs{1}=[];
a=ones(1,datasize(1));
b=ones(1,datasize(1));
b(1)=2;
sample_size=ones(1,datasize(1));
%sensitivity for graph search
%alpha=.05;
tcovstore={};
tcorstore={};
pdag_count=0;
M_error=ones(1,datasize(1))*datasize(2);
pverr=ones(1,datasize(1));
experrvalpr=.1;
experrval=ones(1,datasize(1))*experrvalpr; %expected error value (unbiased ...
    estimated based on observations)
%current sample size is also used, but is initialized separately
trigger=0;
%for CMCD-Causal Model CHANGE DETECTOR
initstbias=0;  %this is something like an initial stability bias
pval=ones(1,datasize(1))*.5;
ntrack=zeros(1,datasize(1));
Q=ones(1,datasize(1))*initstbias;
sumsqrw=ones(1,datasize(1));
poolp=zeros(1,datasize(1));
burnin=10;%burnin=datasize(2)*1.05;  %this determines the length of the burn-in ...
    period %!_!_!_using parfindFOUR
burnin_MD=chi2inv(.5,datasize(2));  %this is the Mahalanobis Distance to use ...
    during the burn-in period
plearn=zeros(1,datasize(1));
make_graph=0;
fol=.005; %frequency of learning parameter, for probabilistic scheduler.
%scale and lower bound parameter for transforming poolp values to weights
scpara=.95;   %normal parameter: .95
ratpar=3;  %!_!_!_using parfindFOUR  %normal parameter: 1.5
%parameter for ratio-type downweighting.  as ratpar ->1, curve steapens/...
    downweights more heavily.
%also determines maximum downweight ratio, equal to 1/ratpar (i.e.,
%ratpar=1 downweights to an effective sample size of 0 at poolp(j)=1,
```

```
%ratpar=2 cuts effective sample size in half at poolp(j)=1
for j=1:datasize(1)
%calc accumulating error rate of correlation
%use Mahalanobis error to calc error of new point from old tcov and mu
%prob want regular M_error here, not normed error.  Take account for
%datasize(2) in the distributional part.
M_error(j) = (data(j,:)-mu)*pinv(tcov)*(data(j,:)-mu)';
%__trying it a new way___
% M_error(j) = (data(j,:)-mu)/tcov*(data(j,:)-mu)';
norm_M_error(j)=M_error(j)/datasize(2);
%Update tcovariance Matrix
%use learning rate to update tcov
%replaces commented out update of b(j+1) below
if j>1
   b(j)=b(j-1)+a(j);
end
%regular OCME-Online Covariation Matrix Estimation
  d=(a(j)/b(j))*(data(j,:)-mu);
  mu=mu+d;
for i=1:datasize(2)
for k=1:datasize(2)
    tcov(i,k)=(1/b(j))*((b(j)-a(j))*tcov(i,k)+(b(j)-a(j))*d(i)*d(k)+a(j)*(data(j...
        ,i)-mu(i))*(data(j,k)-mu(k)));
end
end
tcovstore{j}=tcov;
%update learning rate
%need to track the weighted sum of M_error values, and compare this
%against a distribution which depends on: sample size, datasize(2)
if j>1
   sample_size(j)=(a(j-1)/a(j))*sample_size(j-1)+1;
else
   sample_size(j)=1;
end
%not sample size, actually.  need to track sum of _squared_ weights directly.
%P = normcdf(X,mu,sigma)
%P = chi2cdf(X,V)
%X = norminv(P,mu,sigma)
%calc...  norminv(chi2cdf(M_error(j),datasize(2)),0,1)
%if j>1
%the min is to prevent ntrack(j)=Inf, which causes HUGE PROBLEMS
```

```
%ntrack(j)=norminv(min(chi2cdf(M_error(j),datasize(2)),.999),0,1);
%Q(j)=Q(j-1)+a(j)*ntrack(j);
%sumsqrw(j)=sumsqrw(j-1)+a(j)^2;
%poolp(j)=normcdf(Q(j),0,sqrt(sumsqrw(j)));
%during the burn-in period:
if j>1&&sample_size(j-1)≤burnin
%gotta make sure the right things get burned in
   pval(j)=.5;
%the min is to prevent ntrack(j)=Inf, which causes HUGE PROBLEMS
   ntrack(j)=norminv(min(pval(j),.999),0,1);
   Q(j)=Q(j-1)+a(j)*ntrack(j);
   sumsqrw(j)=sumsqrw(j-1)+a(j)^2; %sum of squared weights
   poolp(j)=normcdf(Q(j),0,sqrt(sumsqrw(j)));
   a(j+1)=a(j);
if trigger==1
   experrval(j)=(experrval(j-1)*(sample_size(j)-1)+chi2cdf(M_error(j),datasize...
       (2))-.5)/(sample_size(j));
else
   experrval(j)=experrvalpr;
end
   pcheck=1;
   trigger=1;
end
%after the burn-in period is over:
if j>1&&isnan(M_error(j))==0&&M_error(j)≥0&&sample_size(j-1)>burnin  %j>burnin ...
    for the burn-in period
   trigger=0;
%Calculating pooled p values and turning them into weights
if poolp(j-1)≤0
   pcheck=0;
end
%experrval(j)=(experrval(j-1)*(sample_size(j)-1)+chi2cdf(M_error(j),datasize(2))...
    -(.5+pverr(j-1)))/(sample_size(j));
%pverr(j)=max(min((priorss*priorerrval+sample_size(j)*experrval(j))/(priorss + ...
    sample_size(j)),pverr(j-1)),0);
pval(j)=fcdf((sample_size(j)-datasize(2))/(datasize(2)*(sample_size(j)-1))*...
    M_error(j),datasize(2),sample_size(j)-datasize(2));
%pval(j)=max(chi2cdf(M_error(j),datasize(2))-min(pverr(j),pcheck),0);   %***...
    Rewrite this in terms of F-distribution/hotelling's t-square***
%pval(j)=chi2cdf(M_error(j)-.03*12*1/sqrt(sample_size(j-1))*20,datasize(2));
%the min/maxes are to prevent ntrack(j)=Inf, which causes HUGE PROBLEMS
```

```
120 ntrack(j)=norminv(max(min(pval(j),.9999),.0001),0,1); %inverse normal cdf of the...
        pvalue
121 Q(j)=Q(j-1)+a(j)*ntrack(j); %weighted sum of inverse normal cdf of p-values
122 sumsqrw(j)=sumsqrw(j-1)+a(j)^2; %sum of squared weights
123 poolp(j)=normcdf(Q(j),0,sqrt(sumsqrw(j))); %pooled pvalue is the appropriate ...
        normal cdf of Q(j)
124 %plearn(j)=plearn(j-1)+fol*poolp(j);%fol is frequency param, square for scaling
125 plearn(j)=plearn(j-1)+fol*(poolp(j)-poolp(j)*plearn(j-1));%fol is frequency ...
        param
126 %this needs to be squashed so that a regular-ish p-value (.5)
127 %doesn't cause massive downweighting.
128 if poolp(j)<scpara
129    a(j+1)=a(j);
130 else
131    a(j+1)=1/(1-1/ratpar*((1/(1-scpara))*poolp(j)-(1/(1-scpara)-1)))*a(j);
132 %previous versions:
133 %the min is to prevent sample sizes from getting below a
134 %certain value, since bad stuff happens if it does
135 %a(j+1)=max((expinv((1/(1-scpara))*poolp(j)-(1/(1-scpara)-1))+1)*a(j),a(j));
136 %a(j+1)=min(b(j)*.02,max((expinv((1/(1-scpara))*poolp(j)-(1/(1-scpara)-1))^(.5)...
        +1)*a(j),a(j)));
137 end
138 %a(j+1)=max(expinv(poolp(j)^2)*b(j),a(j));  %expinv is for rescaling the poolp.
139 %____threshold method____
140 %if poolp(j)<.99
141 %a(j+1)=a(j);
142 %a(j+1)=b(j)*.1;
143 end
144 %learn PDAG matrix from the correlation matrices
145 if isempty(schedule)
146 %probabilistic scheduler
147 if rand(1)<plearn(j) && j>24
148    make_graph=1;
149 %Δ_alpha_MTDL=0;
150    plearn(j)=0;
151 end
152 else
153 %Do full run for particular time steps
154 if ismember(j,schedule)==1
155    make_graph=1;
156 else
```

```
157     make_graph=0;
158  end
159  end
160  %PC search for graph, then plot it
161  if make_graph==1
162  %calc correlations
163     [¬,ExpCorrC] = cov2corr(tcov);
164     pdag_count=pdag_count+1; %index the pdags amongst themselves
165  %pdag_index(pdag_count)=j; %index the pdags amongst the timesteps
166  %use the bayes net toolbox to calculate the pdag matrix
167  %pdag{pdag_count} = learn_struct_pdag_pc('cond_indep_fisher_z', length(cor), ...
         length(cor), cor, floor(sample_size(j)), alpha);
168  %plot pdag
169     output.graphtimes(pdag_count)=j;
170     tcorstore{pdag_count}=ExpCorrC;
171  if pdag_count==1
172     [output.graphs{pdag_count}, sepset, skeleton]=Algorithm_FCI(length(ExpCorrC),...
              ExpCorrC, floor(j), alpha, verbose);
173     sepsetstore{pdag_count}=sepset;
174     skelstore{pdag_count}=skeleton;
175  else
176     [output.graphs{pdag_count}, sepset, skeleton]=Algorithm_Modified_FCI(...
             sepsetstore{pdag_count-1}, length(ExpCorrC), tcorstore{pdag_count}, floor...
             (j), alpha, verbose);
177     sepsetstore{pdag_count}=sepset;
178     skelstore{pdag_count}=skeleton;
179  end
180  if plotgraphs
181     figure('NextPlot','new')
182     draw_graph_pag(abs(output.graphs{pdag_count}))
183     title(j)
184  end
185     make_graph=0;
186  end
187  end
188  %comparing poolp w/ batch equivalent weighting to the known analytic
189  %solution: chi-square distribution with DOF=#data*variables
190  x=0;
191  chisquaretest=zeros(1,datasize(1));
192  for j=1:length(M_error)
193   if j<burnin
```

```
194      x=x+burnin_MD;
195   else
196  x=x+M_error(j);
197  end
198  chisquaretest(j)=chi2cdf(x,j*datasize(2));
199  end
200  if ¬batch
201      plot(sample_size)
202      title('sample size')
203      figure('NextPlot','new')
204      plot(norm_M_error,'red')
205      hold on
206      plot(MTDL)
207      hold off
208      figure('NextPlot','new')
209      plot(poolp)
210      hold on
211      title('pooled p-value')
212      legend('pooled p-value')
213      hold off
214      figure('NextPlot','new')
215      plot(pval)
216      title pval
217      output.skelstore=skelstore;
218      output.tcorstore=tcorstore;
219      output.sepsetstore=sepsetstore;
220      output.norm_M_error=norm_M_error;
221      output.sumsqrw=sumsqrw;
222      output.poolp=poolp;
223      output.chisquaretest=chisquaretest;
224      output.Q=Q;
225      output.a=a;
226      output.M_error=M_error;
227      output.ntrack=ntrack;
228      output.tcovstore=tcovstore;
229      output.b=b;
230      output.sample_size=sample_size;
231      output.pverr=pverr;
232      output.experrval=experrval;
233      output.plearn=plearn;
234      output.pval=pval;
```

```
235     if isempty(schedule)
236         output.plearn=plearn;
237     end
238 end
```

B.7 Modified FCI* Algorithm Matlab Code (created it ourself)

Listing B.7: Modified FCI* Algorithm Matlab Code (created it ourself)

```
1  %I have rewrite and removed possible d separation part
2  function [graph, sepset, skeleton] = Algorithm_Modified_FCI_star(sepset, ...
       corlength, cormatrix, samplesize, alpha, verbose)
3  % LEARN_STRUCT_PDAG_PC Learn a partially oriented DAG (pattern) using the PC ...
       algorithm
4  % P = learn_struct_pdag_pc(cond_indep, n, k, ...)
5  % n is the number of nodes.
6  % k is an optional upper bound on the fan-in (default: n)
7  % cond_indep is a boolean function that will be called as follows:
8  % feval(cond_indep, x, y, S, ...)
9  % where x and y are nodes, and S is a set of nodes (positive integers),
10 % and ... are any optional parameters passed to this function.
11 % The output P is an adjacency matrix, in which
12 % P(i,j) = -1 if there is an i->j edge.
13 % P(i,j) = P(j,i) = 1 if there is an undirected edge i <-> j
14 % The PC algorithm does structure learning assuming all variables are observed.
15 % See Spirtes, Glymour and Scheines, "Causation, Prediction and Search", 1993, ...
       p117.
16 % This algorithm may take O(n^k) time if there are n variables and k is the max ...
       fan-in,
17 % but this is quicker than the Verma-Pearl IC algorithm, which is always O(n^n).
18 [G, sepset, complexity_timer]=Algorithm_FOFCI_Online_Skeleton_Search(sepset, ...
       cormatrix, corlength, alpha, samplesize, verbose);
19 skeleton=G;
20 G=G.*1;
21 graph = R0_V4_2(G, sepset, verbose);
22 flag=1;
23 while(flag)
```

```
24      flag=0;
25      [graph, flag] = R1(graph, flag, verbose);
26      [graph, flag] = R2(graph, flag, verbose);
27      [graph, flag] = R3(graph, flag, verbose);
28      [graph, flag] = R4(graph, sepset, flag, verbose);
29      [graph, flag] = R8(graph, flag, verbose);
30      [graph, flag] = R9(graph, flag, verbose);
31      [graph, flag] = R10(graph, flag, verbose);
32  end
33  fprintf('the total number of independence test = %d \n', complexity_timer);
34  end
35
36  function [graph, flag] = R1(graph, flag, verbose)
37  % If x*->yo-*c and x,z not adjacent ==> x*->y->z
38  [Xs,Ys] = find(graph == 2 & graph' ≠ 0);
39  ind=[Xs Ys];
40      for i = 1:length(ind)
41          a = ind(i,1);
42          b = ind(i,2);
43       tmp1 = intersect(find(graph(b,:) ≠ 0), find(graph(:,b)==1));
44       tmp2 = intersect(find(graph(a,:) == 0), find(graph(:,a) == 0));
45       indC = intersect(tmp1,tmp2);
46       indC = setdiff(indC, a);
47  if (¬isempty(indC))
48  for j = indC
49  if verbose
50     fprintf('\nRule 1'); fprintf('\n Orient: %d', a); fprintf(' *-> %d',b); ...
           fprintf('o-* %d', j); fprintf(' as: %d -> ', b); fprintf(' %d ', j); ...
           fprintf(' \n ');
51  end
52     graph(b,j) = 2;
53     graph(j,b) = 3;
54     flag = 1;
55  end
56  end
57  end
58  end
59
60  function [graph, flag] = R2(graph, flag, verbose)
61  % If x*->yo-*c and x,z not adjacent ==> x*->y->z
62  [Xs,Zs] = find(graph == 1 & graph' ≠ 0);
```

```
63  ind=[Xs Zs];
64  for i = 1:size(ind,1)
65      a = ind(i,1);
66      c = ind(i,2);
67      tmp1 = intersect(find(graph(a,:) == 2), find(graph(:, a) == 3));
68      tmp2 = intersect(find(graph(c,:) ≠ 0), find(graph(:, c) == 2));
69      tmp12 = intersect(tmp1,tmp2);
70      tmp3 = intersect(find(graph(a,:) == 2), find(graph(:, a) ≠ 0));
71      tmp4 = intersect(find(graph(c,:) == 3), find(graph(:, c) == 2));
72      tmp34 = intersect(tmp3,tmp4);
73  if (¬isempty(tmp12)||¬isempty(tmp34))
74  if verbose
75     fprintf('\nRule 2');
76     fprintf('\n Orient: %d -> anynode', a); fprintf('*-> %d ', c);
77     fprintf(' or ');
78     fprintf('%d *-> anynode',a); fprintf('-> %d ', c);
79     fprintf(' with %d *-o %d ', a, c);
80     fprintf('as: %d *-> %d \n', a, c);
81  end
82     graph(a,c) = 2;
83     flag = 1;
84  end
85  end
86  end
87
88  function [graph, flag] = R3(graph, flag, verbose)
89  % If x*->y<-*z, x*-o8o-*z, x,z not adjacent, 8*-oy ==> 8*->y
90  [Ths, Ys] = find(graph == 1);
91  nedges = length(Ths);
92  for i = 1:nedges
93      a = find(graph(:,Ths(i)) == 1 & graph(:,Ys(i)) == 2);
94      len = length(a);
95      f = false;
96  for j = 1:len
97  for k = j+1:len
98  if(graph(a(j),a(k)) == 0 && graph(Ths(i),Ys(i)) == 1)
99  if verbose
100    fprintf('\nRule 3'); fprintf(' Orient: %d', Ys(i)); fprintf(' *-> %d\n', Ths(...
           i));
101 end
102    graph(Ths(i),Ys(i)) = 2;
```

```
103     flag = 1;
104     f = true;
105  break;
106  end
107  end
108  if(f)
109  break;
110  end
111  end
112  end
113  end
114
115  function [graph, flag] = R4(graph, sepset, flag, verbose)
116  % Start from some node X, for node Y
117  % Visit all possible nodes X*->V & V->Y
118  % For every neighbour that is bi-directed and a parent of Y, continue
119  % For every neighbour that is bi-directed and o-*Y, orient and if
120  % parent continue
121  % Total: n*n*(n+m)
122  %--------------------------------------------------------------------------
123  % For each node Y, find all orientable neighbours W
124  % For each node X, non-adjacent to Y, see if there is a path to some
125  % node in W
126  % Create graph as follows:
127  % for X,Y
128  % edges X*->V & V -> Y --> X -> V
129  % edges A <-> B & A -> Y --> A -> B
130  % edges A <-* W & A -> Y --> A->W
131  % discriminating: if path from X to W
132  [rows,cols] = find(graph ≠ 0 & graph' == 1);
133  ind=[rows cols];
134  while (¬isempty(ind))
135        b = ind(1,1);
136        c = ind(1,2);
137        ind(1,:) = [];
138        tmp1 = intersect(find(graph(b,:) == 2), find(graph(:, b) ≠  0));
139        tmp2 = intersect(find(graph(c,:) == 3), find(graph(:, c) == 2));
140        indA = intersect(tmp1,tmp2);
141  while (¬isempty(indA) && graph(c, b) == 1)
142        a = indA(1);
143        indA(1)=[];
```

```
144         Done = false;
145 while ((¬Done) && (graph(a, b) ≠ 0) && (graph(a,c) ≠ 0) && (graph(b, c) ≠ 0))
146         md_path = minDiscrPath(graph, a, b, c);
147         N_md = length(md_path);
148 if (N_md == 1)
149     Done = true;
150 else
151 if (ismember(b, sepset{md_path(1), md_path(N_md)}) || ismember(b, sepset{md_path...
          (N_md), md_path(1)}))
152 if verbose
153     fprintf('\nRule 4'); fprintf('\nThere is a discriminating path between %d', ...
            md_path(1)); fprintf('and %d', c); fprintf('for %d', b); fprintf(',and %d...
            ', b); fprintf('is in Sepset of %d', c); fprintf('and %d', md_path(1)); ...
            fprintf('. Orient: %d', b); fprintf('-> %d \n', c);
154 end
155     graph(b, c) = 2;
156     graph(c, b) = 3;
157     flag = 1;
158 else
159 if verbose
160     fprintf('\nRule 4'); fprintf('\nThere is a discriminating path between %d', ...
            md_path(1)); fprintf('and %d', c); fprintf('for %d', b); fprintf(',and %d...
            ', b); fprintf('is in Sepset of %d', c); fprintf('and %d', md_path(1)); ...
            fprintf('.Orient: %d', a); fprintf('<-> %d', b); fprintf('<-> %d \n', c);
161 end
162     graph(a, b) = 2;
163     graph(b, c) = 2;
164     graph(c, b) = 2;
165     flag = 1;
166 end
167     Done = true;
168 end
169 end
170 end
171 end
172 end%function
173
174 function [graph,flag] = R8(graph, flag, verbose)
175 [r,c] = find(graph == 2 & graph' == 1);
176 nedges = length(r);
177 for i = 1:nedges
```

```
178      out = find(graph(:,r(i)) == 3);
179  if(any(graph(out,c(i)) == 2 & graph(c(i),out)' == 3))
180  if verbose
181     fprintf('\nRule 8'); fprintf('\nOrient: %d', r(i)); fprintf(' -> %d', out); ...
            fprintf(' -> %d', c(i)); fprintf('or %d', r(i)); fprintf('-o %d', out); ...
            fprintf('-> %d', c(i)); fprintf('with %d', r(i)); fprintf('o-> %d', c(i))...
            , fprintf('as %d', r(i)); fprintf(' -> %d \n', c(i));
182  end
183     graph(c(i),r(i)) = 3;
184     flag = 1;
185  end
186  end
187  end
188
189  function [graph,flag] = R9(graph, flag, verbose)
190  % unshieldedTriples=[];
191  % R9: Equivalent to orienting X <-o Y as X <-> Y and checking if Y is an
192  % ancestor of X (i.e. there is an almost directed cycle)
193  [row1,col1] = find(graph == 2 & graph' == 1);
194  ind=[row1 col1];
195  nedges = length(row1);
196  for i = 1:nedges
197      a = row1(i); c = col1(i);
198      ind(1,:)=[];
199      indB=find((graph(a,:) == 2 | graph(a,:) == 1) & (graph(:,a)' == 1 | graph(:,...
             a)' == 3) & (graph(c,:) == 0 & graph(:,c)' == 0));
200      indB=setdiff(indB, c);
201  while ((¬isempty(indB)) && (graph(c, a) == 1))
202      b = indB(1);
203      indB(1) = [];
204      upd = minUncovPdPath(graph, a, b, c);
205  if (length(upd) > 1)
206     graph(c, a) = 3;
207  if verbose
208     fprintf('\nRule 9'); fprintf('\nThere exists an uncovered potentially ...
            directed path between %d and %d', a, c); fprintf('. Orient: %d  -> %d \n'...
            , a, c);
209  end
210     flag = 1;
211  end
212  end
```

```
213 end%for i=nedges
214 end%function
215
216 function [graph,flag] = R10(graph, flag, verbose)
217 [rows,cols] = find(graph == 2 & graph' == 1);
218 ind = [rows cols];
219 while (¬isempty(ind))
220         a = ind(1,1);
221         c = ind(1,2);
222         ind(1,:) = [];
223         [¬,indB] = find((graph(c, :) == 3 & graph(:, c) == 2));
224 if (length(indB) ≥ 2)
225     counterB = 0;
226 while ((counterB < length(indB)) && (graph(c,a) == 1))
227     counterB = counterB + 1;
228     b = indB(counterB);
229     indD = mysetdiff(indB, b);
230     counterD = 0;
231 while ((counterD < length(indD)) && (graph(c, a) == 1))
232     counterD = counterD + 1;
233     d = indD(counterD);
234 if ((graph(a, b) == 1 || graph(a, b) == 2) && (graph(b, a) == 1 || graph(b, a) ...
        == 3) && (graph(a, d) == 1 || graph(a, d) == 2) && (graph(d, a) == 1 || ...
        graph(d, a) == 3) && graph(d, b) == 0 && graph(b, d) == 0)
235 if verbose
236    fprintf('\nRule 10 '); fprintf('\nOrient: %d', a); fprintf('-> %d \n', c);
237 end
238    flag = 1;
239    graph(c, a) = 3;
240 end
241 end
242 end
243 end
244 end
245 end
```

B.8 RFOFCI Algorithm Matlab Code (created it ourself)

Listing B.8: RFOFCI algorithm Matlab Code (created it ourself)

```
%%FCI rule 9 changed with using source r fci code
%%fci updated rules seperated
function output = Algorithm_RFOFCI(data, schedule, alpha, batch, plotgraphs, ...
    verbose)
%initialize
skelstore={};
sepsetstore={};
datasize=size(data);
mu=mean(data(1:10,:));  %mu=mean(data);
tcov=round(cov(data(1:10,:)),8);%tcov=cov(data);
output.graphtimes=[];
output.graphs{1}=[];
a=ones(1,datasize(1));
b=ones(1,datasize(1));
b(1)=2;
sample_size=ones(1,datasize(1));
%sensitivity for graph search
%alpha=.05;
tcovstore={};
tcorstore={};
pdag_count=0;
M_error=ones(1,datasize(1))*datasize(2);
pverr=ones(1,datasize(1));
experrvalpr=.1;
experrval=ones(1,datasize(1))*experrvalpr; %expected error value (unbiased ...
    estimated based on observations)
%current sample size is also used, but is initialized separately
trigger=0;
%for CMCD-Causal Model CHANGE DETECTOR
initstbias=0;  %this is something like an initial stability bias
pval=ones(1,datasize(1))*.5;
ntrack=zeros(1,datasize(1));
Q=ones(1,datasize(1))*initstbias;
sumsqrw=ones(1,datasize(1));
poolp=zeros(1,datasize(1));
burnin=10;%burnin=datasize(2)*1.05;  %this determines the length of the burn-in ...
    period %!_!_!_using parfindFOUR
burnin_MD=chi2inv(.5,datasize(2));  %this is the Mahalanobis Distance to use ...
    during the burn-in period
plearn=zeros(1,datasize(1));
```

```
37 make_graph=0;
38 fol=.005; %frequency of learning parameter, for probabilistic scheduler.
39 %scale and lower bound parameter for transforming poolp values to weights
40 scpara=.95;   %normal parameter: .95
41 ratpar=3;  %!_!_!_using parfindFOUR  %normal parameter: 1.5
42 %parameter for ratio-type downweighting.  as ratpar ->1, curve steapens/...
       downweights more heavily.
43 %also determines maximum downweight ratio, equal to 1/ratpar (i.e.,
44 %ratpar=1 downweights to an effective sample size of 0 at poolp(j)=1,
45 %ratpar=2 cuts effective sample size in half at poolp(j)=1
46 for j=1:datasize(1)
47 %calc accumulating error rate of correlation
48 %use Mahalanobis error to calc error of new point from old tcov and mu
49 %prob want regular M_error here, not normed error.  Take account for
50 %datasize(2) in the distributional part.
51 %M_error(j) = (data(j,:)-mu)*inv(tcov)*(data(j,:)-mu)';
52 M_error(j) = (data(j,:)-mu)*pinv(tcov)*(data(j,:)-mu)';
53 %__trying it a new way___
54 %M_error(j) = (data(j,:)-mu)/tcov*(data(j,:)-mu)';
55 norm_M_error(j)=M_error(j)/datasize(2);
56 %Update tcovariance Matrix
57 %use learning rate to update tcov
58 %replaces commented out update of b(j+1) below
59 if j>1
60    b(j)=b(j-1)+a(j);
61 end
62 %regular OCME-Online Covariation Matrix Estimation
63 d=(a(j)/b(j))*(data(j,:)-mu);
64 mu=mu+d;
65 for i=1:datasize(2)
66   for k=1:datasize(2)
67       tcov(i,k)=(1/b(j))*((b(j)-a(j))*tcov(i,k)+(b(j)-a(j))*d(i)*d(k)+a(j)*(data...
           (j,i)-mu(i))*(data(j,k)-mu(k)));
68   end
69 end
70 tcovstore{j}=tcov;
71 %update learning rate
72 %need to track the weighted sum of M_error values, and compare this
73 %against a distribution which depends on: sample size, datasize(2)
74 if j>1
75    sample_size(j)=(a(j-1)/a(j))*sample_size(j-1)+1;
```

```
76  else
77     sample_size(j)=1;
78  end
79  %not sample size, actually.  need to track sum of _squared_ weights directly.
80  %P = normcdf(X,mu,sigma)
81  %P = chi2cdf(X,V)
82  %X = norminv(P,mu,sigma)
83  %calc...  norminv(chi2cdf(M_error(j),datasize(2)),0,1)
84  %if j>1
85  %the min is to prevent ntrack(j)=Inf, which causes HUGE PROBLEMS
86  %ntrack(j)=norminv(min(chi2cdf(M_error(j),datasize(2)),.999),0,1);
87  %Q(j)=Q(j-1)+a(j)*ntrack(j);
88  %sumsqrw(j)=sumsqrw(j-1)+a(j)^2;
89  %poolp(j)=normcdf(Q(j),0,sqrt(sumsqrw(j)));
90  %during the burn-in period:
91  if j>1&&sample_size(j-1)≤burnin
92  %gotta make sure the right things get burned in
93     pval(j)=.5;
94  %the min is to prevent ntrack(j)=Inf, which causes HUGE PROBLEMS
95     ntrack(j)=norminv(min(pval(j),.999),0,1);
96     Q(j)=Q(j-1)+a(j)*ntrack(j);
97     sumsqrw(j)=sumsqrw(j-1)+a(j)^2; %sum of squared weights
98     poolp(j)=normcdf(Q(j),0,sqrt(sumsqrw(j)));
99     a(j+1)=a(j);
100 if trigger==1
101    experrval(j)=(experrval(j-1)*(sample_size(j)-1)+chi2cdf(M_error(j),datasize...
           (2))-.5)/(sample_size(j));
102 else
103    experrval(j)=experrvalpr;
104 end
105    pcheck=1;
106    trigger=1;
107 end
108 %after the burn-in period is over:
109 if j>1&&isnan(M_error(j))==0&&M_error(j)≥0&&sample_size(j-1)>burnin  %j>burnin ...
       for the burn-in period
110    trigger=0;
111 %Calculating pooled p values and turning them into weights
112 if poolp(j-1)≤0
113    pcheck=0;
114 end
```

```
%experrval(j)=(experrval(j-1)*(sample_size(j)-1)+chi2cdf(M_error(j),datasize(2))...
    -(.5+pverr(j-1)))/(sample_size(j));
%pverr(j)=max(min((priorss*priorerrval+sample_size(j)*experrval(j))/(priorss + ...
    sample_size(j)),pverr(j-1)),0);
   pval(j)=fcdf((sample_size(j)-datasize(2))/(datasize(2)*(sample_size(j)-1))*...
       M_error(j),datasize(2),sample_size(j)-datasize(2));
%pval(j)=max(chi2cdf(M_error(j),datasize(2))-min(pverr(j),pcheck),0);   %***...
    Rewrite this in terms of F-distribution/hotelling's t-square***
%pval(j)=chi2cdf(M_error(j)-.03*12*1/sqrt(sample_size(j-1))*20,datasize(2));
%the min/maxes are to prevent ntrack(j)=Inf, which causes HUGE PROBLEMS
   ntrack(j)=norminv(max(min(pval(j),.9999),.0001),0,1); %inverse normal cdf of ...
       the pvalue
   Q(j)=Q(j-1)+a(j)*ntrack(j); %weighted sum of inverse normal cdf of p-values
   sumsqrw(j)=sumsqrw(j-1)+a(j)^2; %sum of squared weights
   poolp(j)=normcdf(Q(j),0,sqrt(sumsqrw(j))); %pooled pvalue is the appropriate ...
       normal cdf of Q(j)
%plearn(j)=plearn(j-1)+fol*poolp(j);%fol is frequency param, square for scaling
   plearn(j)=plearn(j-1)+fol*(poolp(j)-poolp(j)*plearn(j-1));%fol is frequency ...
       param
%this needs to be squashed so that a regular-ish p-value (.5)
%doesn't cause massive downweighting.
if poolp(j)<scpara
   a(j+1)=a(j);
else
   a(j+1)=1/(1-1/ratpar*((1/(1-scpara))*poolp(j)-(1/(1-scpara)-1)))*a(j);
%previous versions:
%the min is to prevent sample sizes from getting below a
%certain value, since bad stuff happens if it does
%a(j+1)=max((expinv((1/(1-scpara))*poolp(j)-(1/(1-scpara)-1))+1)*a(j),a(j));
%a(j+1)=min(b(j)*.02,max((expinv((1/(1-scpara))*poolp(j)-(1/(1-scpara)-1))^(.5)...
    +1)*a(j),a(j)));
end
%a(j+1)=max(expinv(poolp(j)^2)*b(j),a(j));  %expinv is for rescaling the poolp.
%____threshold method____
%if poolp(j)<.99
%a(j+1)=a(j);
%else
%a(j+1)=b(j)*.1;
%end
end
%learn PDAG matrix from the correlation matrices
```

```
148 if isempty(schedule)
149 %probabilistic scheduler
150 if rand(1)<plearn(j) && j>24
151    make_graph=1;
152 %Δ_alpha_MTDL=0;
153    plearn(j)=0;
154 end
155 else
156 %Do full run for particular time steps
157 if ismember(j,schedule)==1
158    make_graph=1;
159 else
160    make_graph=0;
161 end
162 end
163 %PC search for graph, then plot it
164 if make_graph==1
165 %calc correlations
166    [¬,ExpCorrC] = cov2corr(tcov);
167    pdag_count=pdag_count+1; %index the pdags amongst themselves
168 %pdag_index(pdag_count)=j; %index the pdags amongst the timesteps
169 %use the bayes net toolbox to calculate the pdag matrix
170 %pdag{pdag_count} = learn_struct_pdag_pc('cond_indep_fisher_z', length(cor), ...
        length(cor), cor, floor(sample_size(j)), alpha);
171 %plot pdag
172    output.graphtimes(pdag_count)=j;
173    tcorstore{pdag_count}=ExpCorrC;
174 if pdag_count==1
175    [output.graphs{pdag_count}, sepset, skeleton]=Algorithm_FCI(length(ExpCorrC),...
          ExpCorrC, floor(j), alpha, verbose);
176    sepsetstore{pdag_count}=sepset;
177    skelstore{pdag_count}=skeleton;
178 else
179    [output.graphs{pdag_count}, sepset, skeleton]=Algorithm_Modified_FCI_star(...
        sepsetstore{pdag_count-1}, length(ExpCorrC), tcorstore{pdag_count}, floor...
        (j), alpha, verbose);
180    sepsetstore{pdag_count}=sepset;
181    skelstore{pdag_count}=skeleton;
182 end
183 if plotgraphs
184    figure('NextPlot','new')
```

```
185     draw_graph_pag(abs(output.graphs{pdag_count}))
186     title(j)
187 end
188     make_graph=0;
189 end
190 end
191 %comparing poolp w/ batch equivalent weighting to the known analytic
192 %solution: chi-square distribution with DOF=#data*variables
193 x=0;
194 chisquaretest=zeros(1,datasize(1));
195 for j=1:length(M_error)
196 if  j<burnin
197     x=x+burnin_MD;
198 else
199     x=x+M_error(j);
200 end
201     chisquaretest(j)=chi2cdf(x,j*datasize(2));
202 end
203 if ¬batch
204     plot(sample_size)
205     title('sample size')
206     figure('NextPlot','new')
207     plot(norm_M_error,'red')
208     hold on
209     plot(MTDL)
210     hold off
211     figure('NextPlot','new')
212     plot(poolp)
213     hold on
214     title('pooled p-value')
215     legend('pooled p-value')
216     hold off
217     figure('NextPlot','new')
218     plot(pval)
219     title pval
220     output.skelstore=skelstore;
221     output.tcorstore=tcorstore;
222     output.sepsetstore=sepsetstore;
223     output.norm_M_error=norm_M_error;
224     output.sumsqrw=sumsqrw;
225     output.poolp=poolp;
```

```
226      output.chisquaretest=chisquaretest;
227      output.Q=Q;
228      output.a=a;
229      output.M_error=M_error;
230      output.ntrack=ntrack;
231      output.tcovstore=tcovstore;
232      output.b=b;
233      output.sample_size=sample_size;
234      output.pverr=pverr;
235      output.experrval=experrval;
236      output.plearn=plearn;
237      output.pval=pval;
238      if isempty(schedule)
239          output.plearn=plearn;
240      end
241  end
```

Appendix B. Online Algorithms Matlab Code (created it ourself)

Abbreviations

AGs Ancestral Graphs

AI Artificial Intelligence

AIC Akaike Information Criterion

BDeu Bayesian Dirichlet equivalent uniform

BE Bayesian estimation

BIC Bayesian Information Criterion

BIC Bayesian information criterion

BNs Bayesian Networks

CB Constraint Based

CBNs Causal Bayesian Networks

CMCD Causal Model Change Detector

CML Causal Model Learner

CPD Bayesian change-point detection

DAG Directed Acyclic Graph

DOCL Dynamic Online Causal Learning

EM Expectation Maximization

FCI Fast Causal Inference

FOFCI Fast Online Fast Causal Inference

GAs Genetic Algorithms

HMM Hidden Markov Models

KL Kullback-Leibler divergence

LL log-likelihood

LoSST Locally Stationary Structure Tracker

MAG Maximal Ancestral Graph

MDL Minimum Description Length ()

MLE Maximum likelihood estimation

MMHC Max-Min Hill Climbing

MNs Markov networks

OCME Online Covariance Matrix Estimator

OFCI Online Fast Causal Inference

PAG Partial Ancestral Graph

PDAGs Partial Directed Acyclic Graph

PGM probabilistic graphical models

RFCI Really Fast Causal Inference

RFOFCI Really Fast Online Fast Causal Inference

SB Score Based

SEMs Causal Structural Equation Models

SHD Structural Hamming Distance

SVI Stochastic variational inference

TDL temporal difference learning

Bibliography

[1] Ryan Prescott Adams and David J. C. MacKay. Bayesian Online Changepoint Detection. oct 2007.

[2] Hirotogu Akaike. *Information Theory and an Extension of the Maximum Likelihood Principle*, pages 199–213. Springer New York, New York, NY, 1998.

[3] Marcus Bendtsen. Regime Aware Learning. *Pgm*, 52:1–12, 2016.

[4] Jeff A. Bilmes. A Gentle Tutorial of the EM Algorithm. 1198(510):126, 1998.

[5] Wray Buntine. Theory refinement on Bayesian networks. pages 52–60, jul 1991.

[6] K.P. Burnham and D.R. Anderson. *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Springer New York, 2003.

[7] by Josep Roure Alcobé, Ramon Sangüesa, and Ulises Cortés Program. Incremental Methods for Bayesian Network Structure Learning. 2004.

[8] Zhiyuan Chen and Bing Liu. Lifelong Machine Learning Test. *AAAI-2015 Workshop on Beyond the Turing Test*, 2016.

[9] Jie Cheng, David Bell, and Weiru Liu. Learning Bayesian Networks from Data: An Efficient Approach Based on Information Theory. 1998.

[10] D M Chickering and C Meek. Finding optimal bayesian networks. *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 94–102, 2002.

[11] Hogun Chong, Mary Zey, David A Bessler, Hogun Chonga, Mary Zeyb, and David A Besslerc '. On Corporate Structure, Strategy, and Performance: A Study with Directed Acyclic Graphs and PC Algorithm On Corporate Structure, Strategy, Performance: A Study with Direct Acyclic Graphs and PC Algorithm. *Source: Managerial and Decision Economics*, 31(1):47–62, 2010.

[12] Diego Colombo. Causal Inference in High-Dimensional Systems. (21445), 2013.

[13] Diego Colombo, Marloes H Maathuis, Markus Kalisch, and Thomas S Richardson. Learning high-dimensional DAGs with latent and selection variables. 2011.

[14] Diego Colombo, Marloes H. Maathuis, Markus Kalisch, and Thomas S. Richardson. Learning high-dimensional directed acyclic graphs with latent and selection variables. *Annals of Statistics*, 40(1):294–321, 2012.

[15] Gregory E Cooper, Gfc@med Pitt Edu, Edward Herskovits Ehh@sumex-Ai, M Stan, and Ford Edu. A Bayesian Method for the Induction of Probabilistic Networks from Data. *Machine Learning*, 9:309–347, 1992.

[16] Gregory F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990.

[17] A V Crawford. *Posterior predictive model checking in bayesian networks*. PhD thesis, Arizona State University, 2014.

[18] James Cussens, Matti Järvisalo, Janne H Korhonen, and Mark Bartlett. Bayesian Network Structure Learning with Integer Programming: Polytopes, Facets and Complexity (Extended Abstract) *. Technical report, 2017.

[19] A.P. Dempster, N.M. Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society Series B Methodological*, 39(1):1–38, 1977.

[20] Pedro Domingos and Geoff Hulten. Catching Up with the Data: Research Issues in Mining Data Streams. *Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2001.

[21] Frederick Eberhardt. Causation and Intervention. 2009.

[22] N Foti, J Xu, D Laird, and E Fox. Stochastic variational inference for hidden Markov models. *Advances in Neural Information ...*, (1):1–9, 2014.

[23] Nir Friedman and Moises Goldszmidt. Sequential update of Bayesian network structure. *UAI'97 Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, (1):165–174, 1997.

[24] Nir Friedman, Lftach Nachman, and Dana Peer. Learning Bayesian Network Structure from Massive Datasets: The "Sparse Candidate" Algorithm. 1999.

[25] Charles Miller Grinstead and James Laurie Snell. Introduction to Probability. *Swarthmore College*, pages 1–520, 2007.

[26] Daniel M Hausman and James Woodward. Independence, Invariance and the Causal Markov Condition. 1999.

[27] David Heckerman and Ross Shachter. Decision-Theoretic Foundations for Causal Reasoning. *Journal of Artiicial Intelligence Research Submitted*, 3(6695):405–430, 1995.

[28] Antti Hyttinen. *Discovering Causal Relations in the Presence of Latent Confounders*. 2013.

[29] M Kalisch and P Bühlmann. Estimating High-Dimensional Directed Acyclic Graphs with the PC-Algorithm. *Journal of Machine Learning Research*, 8:613–636, 2007.

[30] Markus Kalisch, Martin Mächler, Diego Colombo, Alain Hauser, Marloes H Maathuis, and Peter Bühlmann. More Causal Inference with Graphical Models in R Package pcalg. 2012.

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

[32] Markus Kalisch, Eth Zurich, Martin Mächler, Diego Colombo, Marloes H Maathuis, and Peter Bühlmann. Causal Inference using Graphical Models with the Package pcalg. *JSS Journal of Statistical Software MMMMMM YYYY*, VV.

[33] Ron. Kenett and Yossi. Raanan. *Operational risk management : a practical approach to intelligent data analysis*. John Wiley & Sons, 2010.

[34] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[35] Durdane Kocacoban and James Cussens. Online causal structure learning in the presence of latent variables. In *IEEE ICMLA 2019*.

[36] Durdane Kocacoban and James Cussens. Fast online learning in the presence of latent variables. In *ISAAI'19*. Springer Publishing Company, 2019.

[37] Durdane Kocacoban and James Cussens. Online Causal Structure Learning in the Presence of Latent Variables. *arXiv e-prints*, page arXiv:1904.13247, Apr 2019.

[38] Kaname Kojima and Eric Perrier. Optimal Search on Clustered Structural Constraint for Learning Bayesian Network Structure Seiya Imoto Satoru Miyano. Technical report, 2010.

[39] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models*. 2009.

[40] Petri Kontkanen, Petri Myllymm aki, Tomi Silander, Henry Tirri, and Peter Grr unwald CWI. Comparing Predictive Inference Methods for Discrete Domains. Technical report, 1997.

[41] TIMO J. T. Koski and John M Noble. A Review of Bayesian Networks and Structure Learning. 40(1):53–103, 2012.

[42] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[43] Erich Kummerfeld. Theoretical Entities : Their Discovery and Justification. 2015.

[44] Erich Kummerfeld and David Danks. Online Causal Structure Learning. 2010.

[45] Erich Kummerfeld and David Danks. Tracking Time-varying Graphical Structure. *Advances in Nueral Information Processing Systems 26 (Proceedings of NIPS)*, pages 1–9, 2013.

[46] Erich Kummerfeld, David Danks, and Machine Cognition. Online Learning of Time-varying Causal Structures. 2012.

[47] Wai Lam and Fahiem Bacchus. Using new data to refine a Bayesian network. pages 383–390, jul 1994.

[48] Philippe Leray and Olivier François. BNT Structure Learning Package : Documentation and Experiments. *Structure*, 1(November):1–31, 2004.

[49] Guoliang Li and Tze Yun Leong. Active learning for causal bayesian network structure with non-symmetrical entropy. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5476 LNAI:290–301, 2009.

[50] Faming Liang and Jian Zhang. Learning Bayesian networks for discrete data. *Computational Statistics & Data Analysis*, 53(4):865–876, 2009.

[51] Liptak T. On the combination of independent tests. *Magyar Tud. Akad. Mat. Kutato Int. Kozl*, 3:171–197, 1958.

[52] Vadim Lozin. Graph Theory Notes *.

[53] P C Mahalanobis. On the generalized distance in Statistics. 1936.

[54] Marc Maier, Katerina Marazopoulou, and David Jensen. Reasoning about Independence in Probabilistic Models of Relational Data. *Approaches to Causal Structure Learning Workshop, UAI-13*, 2013.

[55] Makram Talih and Nicolas Hengartner. Structural Learning with Time-Varying Components : Tracking the Cross-Section of Financial Time Series. *Journal of the Royal Statistical Society Series B Methodological*, 67(3):321–341, 2005.

[56] Daniel Malinsky and Peter Spirtes. Estimating Causal Effects with Ancestral Graph Markov Models. 52:299–309, 2016.

[57] Dimitris Margaritis. *Learning Bayesian Network Model Structure from Data*. PhD thesis, Carnegie Mellon University, 2003.

[58] Christopher Meek. Causal inference and causal explanation with background knowledge. pages 403–410, aug 1995.

[59] Christopher Meek. Strong completeness and faithfulness in Bayesian networks. 2013.

[60] Gianluigi Mongillo and Sophie Deneve. Online learning with hidden markov models. *Neural computation*, 20(7):1706–16, jul 2008.

[61] Petri Myllym¨aki, Myllym¨ Myllym¨aki, Tomi Silander, Henry Tirri, and Pekka Uronen. B-COURSE: A WEB-BASED TOOL FOR BAYESIAN AND CAUSAL DATA ANALYSIS. Technical Report 3, 2002.

[62] Richard E Neapolitan. *Probabilistic Reasoning In Expert Systems: Theory and Algorithms*. 1989.

[63] Hoai-Tuong Nguyen. *Réseaux bayésiens et apprentissage ensembliste pour l'étude différentielle de réseaux de régulation génétique*. PhD thesis, UNIVERSITÉ DE NANTES, 2019.

[64] Farheen Omar. Online Bayesian Learning in Probabilistic Graphical Models using Moment Matching with Applications by. 2016.

[65] Judea. Pearl. *Probabilistic reasoning in intelligent systems : networks of plausible inference*. Morgan Kaufmann Publishers, 1988.

[66] Judea Pearl. Causal diagrams for empirical research. *Biometrika*, 82(4):669–688, 1995.

[67] Judea. Pearl. *Causality : models, reasoning, and inference*. Cambridge University Press, 2000.

[68] Judea Pearl. Causal inference in statistics: An overview. *Statistics Surveys*, 3(0):96–146, 2009.

[69] Judea Pearl. The Do-Calculus Revisited. 2012.

[70] Judea Pearl and Stuart Rusell. Bayesian Networks. pages 149–153, 2011.

[71] Judea Pearl and Ts Verma. A Theory of Inferred Causation. pages 441–452, 1991.

[72] Eric Perrier. Finding Optimal Bayesian Network Given a Super-Structure Seiya Imoto Satoru Miyano. Technical report, 2008.

[73] Josep Roure. Incremental Methods for Bayesian Network Structure Learning. page 157, 2004.

[74] Saeed Samet, Ali Miri, and Eric Granger. Incremental learning of privacy-preserving Bayesian networks. *Applied Soft Computing Journal*, 13(8):3657–3667, 2013.

[75] Marco Scutari. Learning Bayesian Networks with the bnlearn R Package. *Journal of Statistical Software*, 35(3):1–22, 2010.

[76] Da Shi and Shaohua Tan. Incremental learning Bayesian network structures efficiently. *Proc. 11th Int Control Automation Robotics & Vision (ICARCV) Conf*, (December):1719–1724, 2010.

[77] Tomi Silander. *The Most Probable Bayesian Network and Beyond*. 2009.

[78] Ricardo Silva, Richard Scheines, Cmu Clark Glymour, Cmu Tom Mitchell, and Cmu Greg Cooper. Automatic Discovery of Latent Variable Models. 2005.

[79] Michael R Siracusa. Tractable Bayesian Inference of Time-Series Dependence Structure. *Proceedings of the 23th International Conference on Artificial Intelligence and Statistics*, 5:528–535, 2009.

[80] Christopher; Richardson Thomas Spirtes, Peter; Meek. An Algorithm for Causal Inference in the Presence of Latent Variables and Selection Bias. In *Computation, Causation, and Discovery*. AAAI Press, 05 1999.

[81] Pater Spirtes, Clark Glymour, Richard Scheines, Stuart Kauffman, Valerio Aimale, and Frank Wimberly. Constructing Bayesian Network Models of Gene Expression Networks from Microarray Data. 2000.

[82] Peter Spirtes. Directed Cyclic Graphical Representations of Feedback Models. 1995.

[83] Peter Spirtes. Causal Inference in the Presence of Latent Variables and Selection Bias, 1997.

[84] Peter Spirtes. An anytime algorithm for causal inference. *Proceedings of AISTATS*, pages 213–231, 2001.

[85] Peter Spirtes. Introduction to Causal Inference. *Journal of Machine Learning Research*, 11:1643–1662, 2010.

[86] Peter Spirtes, Clark Glymour, and Richard Scheines. Causation, Prediction, and Search. *Technometrics*, 45(3):272–273, 1993.

[87] Peter Spirtes, Clark Glymour, and Richard Scheines. Causation, Prediction, and Search. *Technometrics*, 45(3):272–273, 2003.

[88] Peter Spirtes, Christopher Meek, and Thomas Richardson. Causal Inference in the Presence of Latent Variables and Selection Bias1. pages 1–46.

[89] Luis Enrique Sucar. *Probabilistic Graphical Models: Principles and Applications*. Springer Publishing Company, Incorporated, 2015.

[90] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, aug 1988.

[91] Ioannis Tsamardinos, Constantin F Aliferis, and Alexander Statnikov. Algorithms for Large Scale Markov Blanket Discovery. 2003.

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

[93] Ioannis Tsamardinos and Sofia Triantafillou. Learning Causal Networks In the presence of Latent Variables.

[94] David M Williamson, Russell G Almond, and Robert J Mislevy. Model Criticism of Bayesian Networks with Latent Variables. *Uncertainty in Artificial Intelligence Proceedings*, pages 634–643, 2000.

[95] Sewall Wright. Correlation and Causation.pdf. 20:557–585, 1921.

[96] X. Yang. Metaheuristic Optimization. *Scholarpedia*, 6(8):11472, 2011. revision #91488.

[97] Sandeep Yaramakala and Dimitris Margaritis. Speculative Markov Blanket Discovery for Optimal Feature Selection. 2005.

[98] Amanullah Yasin. Incremental Bayesian network structure learning from data streams. 2016.

[99] Jiji Zhang. Causal Inference and Reasoning in Causally Insufficient Systems. 2006.

[100] Jiji Zhang. On the completeness of orientation rules for causal discovery in the presence of latent confounders and selection bias. *Artificial Intelligence*, 172(16-17):1873–1896, nov 2008.