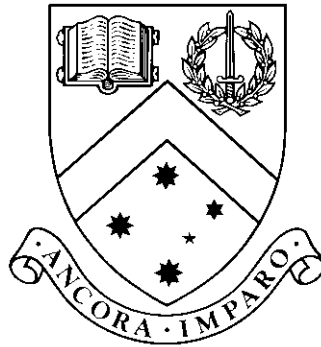


Discovering and Learning Causal Bayesian Models with Latent Variables

by

Xuhui Zhang



Thesis

Submitted by Xuhui Zhang

for fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Supervisor: Dr. Kevin B. Korb

Associate Supervisors: Dr. Steven Mascaro

Prof. Ann E. Nicholson

**Clayton School of Information Technology
Monash University**

February, 2021

© Copyright

by

Xuhui Zhang

2021

To my mother, Yuxiang Chen

Contents

List of Tables	vi
List of Figures	viii
Abstract	xi
Acknowledgments	xiii
1 Introduction	1
1.1 Motivation and Objectives	2
1.2 Contributions and Outline	3
1.3 Thesis Layout	3
2 Bayesian Networks	5
2.1 Bayesian Network Basics	5
2.1.1 Causal Bayesian Networks	7
2.1.2 D-separation	9
2.1.3 Markov Equivalence	10
2.1.4 Causal Discovery vs Regression	11
3 Learning With Latent Variables	13
3.1 The Necessity of Modeling with Latent Variables	13
3.2 Causal Discovery With Latent Variables	14
3.3 Constraint-based Learners	16
3.3.1 IC Algorithm	18
3.3.2 PC Algorithm	19
3.3.3 FCI Algorithm	20
3.4 Metric-based Learners	21
3.4.1 EM Algorithm	22
3.4.2 SEM Algorithm	23
3.4.3 CaMML	24
3.5 Evaluation Metrics	27
3.5.1 Edit Distance	27
3.5.2 KL and CKL Divergence	29
4 Dependency Pattern Discovery	31
4.1 Definition of Dependency Matrix	31
4.2 A Systematic Search for Triggers	32
4.3 Learning Triggers With Causal Discovery Algorithms	38
4.4 Experiment	40
4.4.1 Generating Simulated Datasets of Triggers	40
4.4.2 Experimental Results	42

4.5	Summary	45
5	EM-CaMML for Latent Variable Models	47
5.1	EM-CaMML Algorithm Design	47
5.1.1	A New Score Metric for Latent Nodes	49
5.1.2	EM Algorithm Integration	51
5.1.3	EM-CaMML Workflow	56
5.2	Experiment	57
5.2.1	Initial Structure Selection	57
5.2.2	Generating Complete DAGs for PC and FCI	58
5.2.3	Arc Prior Probability Optimization	60
5.2.4	Alarm	64
5.2.5	Stud Farm	67
5.2.6	Bookbags	69
5.2.7	Mendel Genetics	71
5.3	Software Design and Implementation	74
5.4	Summary	76
6	EM-CaMML Extensions	79
6.1	Learning Partial Triggers	79
6.1.1	Definition of Partial Trigger	79
6.1.2	Experiment Results and Discussion	81
6.2	Discovering Multiple Latent Variables	86
6.2.1	Markov Blanket Discovery	86
6.2.2	Experiment Results and Discussion	89
6.3	Summary	92
7	Conclusion and Future Research	93
7.1	Main Contributions	93
7.2	Future Work	94
	References	97
	Glossary	103
	Notation	105
	Appendix A All Single Latent Triggers for Five Observed Variables	107
	Appendix B FCI, PC and Trigger-PC Performance by Arc Strength	109
	Appendix C Test Network Selection	113
C.1	Test Network Sources	113
C.2	All Test Networks	114

List of Tables

3.1	Four different mutations used in CaMML.	26
4.1	Execution time of Trigger original and Trigger-threads.	34
4.2	Number of DAGs and triggers.	35
4.3	Triggers found (the Big-W and covered Big-W) for four observed variables (node “H” represents the latent).	35
4.4	Edit distance for PC and FCI output.	39
4.5	Number of simulated datasets for trigger.	41
4.6	Number of simulated datasets for DAG structures (no latent variable).	41
4.7	The artificial datasets used for comparing Trigger-PC with PC and FCI.	41
4.8	The optimized alpha of each algorithm.	42
4.9	Alpha optimization of Trigger cases (using Type 2 datasets).	42
4.10	Alpha optimization of DAG cases (using Type 3 datasets).	43
4.11	Results using optimized alpha. See Table 4.13 for more detailed results of different arc strengths (bold and italic represent significantly the best and worst results).	44
4.12	Results using default alpha 0.05. See Table 4.14 for more detailed results of different arc strengths (bold and italic represent significantly the best and worst results).	44
4.13	Results of networks with different arc strengths using optimized alpha (bold and italic represent significantly the best and worst results).	45
4.14	Results of networks with different arc strengths using default alpha 0.05 (bold and italic represent significantly the best and worst results).	45
5.1	Experiment configurations for comparing EM-CaMML with PC, FCI and SEM.	57
5.2	Arc orientation rules for PC and FCI output.	59
5.3	Edit distance rules used in this thesis.	60
5.4	The legend keys used in all experiment result graphs in Section 5.2.4 - 5.2.7.	64
6.1	Example MBs of the network shown in Figure 6.15.	89
6.2	MBMML+CPT result.	89
6.3	MB selection process using MB_pick_greedy function (the value in bracket represents the score).	90
B.1	Results of networks with maximum arc strengths using optimized alpha (bold and italic represent significantly the best and worst results).	109
B.2	Results of networks with medium arc strengths using optimized alpha (bold and italic represent significantly the best and worst results).	110
B.3	Results of networks with minimum arc strengths using optimized alpha (bold and italic represent significantly the best and worst results).	110

B.4	Results of networks with maximum arc strengths using default alpha 0.05 (bold and italic represent significantly the best and worst results).	111
B.5	Results of networks with medium arc strengths using default alpha 0.05 (bold and italic represent significantly the best and worst results).	111
B.6	Results of networks with minimum arc strengths using default alpha 0.05 (bold and italic represent significantly the best and worst results).	112
C.1	All test networks.	113
C.2	Arc prior optimization test networks.	113

List of Figures

1.1	Example of how a latent node can simplify a fully connected model (node “H” represents the latent).	2
2.1	Sprinkler network.	6
2.2	An example NB.	8
2.3	Chain structure in d-separation.	9
2.4	Common cause structure in d-separation.	9
2.5	Common effect structure in d-separation.	9
2.6	Example of Markov Equivalent Models.	10
3.1	The Big-W, a causal structure with four observed variables and one latent variable H.	14
3.2	An undirected graph, G1.	16
3.3	A directed graph, G2.	16
3.4	An example of inducing path graph G3.	17
3.5	An example of partially oriented inducing graph G4.	18
3.6	Example of how two TOMs (with different node orderings) are grouped into a DAG.	26
3.7	Example of how two DAGs are grouped into a SEC.	26
3.8	Example of how to calculate edit distance.	28
3.9	A structure with multiple latent nodes (“H.1” and “H.2”).	28
4.1	A DAG of four nodes and its dependency matrix (0 and 1 represent conditional independence and dependence respectively) conditioned on node Y.	31
4.2	Execution time of Trigger original and Trigger-threads.	34
4.3	Some examples of triggers of five observed variables (node “H” represents the latent).	36
4.4	An example of MAG.	37
4.5	Two MAGs that are not Markov equivalent to any DAG which correspond to Big-W and Covered Big-W.	37
5.1	A triggering submodel with 6 observed nodes containing Big-W (node “H” represents the latent).	48
5.2	Example of Expected Counts.	50
5.3	Test networks for EM integration experiment (node “H” represents the latent).	53
5.4	The execution time of EM inside and outside of Space Mission network (x and y axis represent sample size and execution time respectively).	53
5.5	The execution time of EM inside and outside of Earthquake network (x and y axis represent sample size and execution time respectively).	54
5.6	The execution time ratio of EM inside and outside by sample size (x and y axis represent number of MCMC iterations and the ratio respectively).	54

5.7	The execution time ratio of EM inside and outside (x and y axis represent number of MCMC iterations and the ratio respectively).	54
5.8	EM-CaMML workflow.	56
5.9	Initializations 1, 2 and 3 (node “H” represents the latent).	58
5.10	Example conversion of bi-directional cliques into latent nodes (H_1 and H_2).	58
5.11	Process of arc prior probability optimisation.	60
5.12	Original Big-W Alarm network and its variant (node “H” represents the latent).	61
5.13	Level 1 F-score results of Big-W networks.	62
5.14	Level 2 F-score results (Criteria 1) of Big-W networks.	63
5.15	Level 2 F-score results (Criteria 2) of Big-W networks.	63
5.16	Big-W Alarm network (node “H” represents the latent).	64
5.17	Edit distance results of Big-W Alarm cases.	65
5.18	How FCI output is fixed of Big-W Alarm network (node “H” represents the latent).	65
5.19	Typical example of SEM and EM-CaMML learned model of Big-W Alarm using ini2 at sample size of 2000 (node “H” represents the latent).	65
5.20	KL and CKL results of Big-W Alarm cases.	66
5.21	Results of latent node connection and trigger subnet learning of Big-W Alarm cases.	66
5.22	Covered Big-W Stud Farm network (node “H” represents the latent).	67
5.23	Edit distance results of Covered Big-W Stud farm cases.	67
5.24	Example of SEM output of Covered Big-W Stud Farm (node “H” represents the latent).	68
5.25	KL and CKL results of covered Big-W Stud Farm cases.	68
5.26	Results of latent node connection and trigger subnet learning of covered Big-W Stud Farm cases.	69
5.27	Latent Bookbags network (node “H” represents the latent).	69
5.28	Edit distance results of Latent Bookbags cases.	69
5.29	An EM-CaMML (ini3) output of Latent Bookbags at sample size 5000 (node “H” represents the latent).	70
5.30	KL and CKL results of Latent Bookbags cases.	70
5.31	Probabilities of latent node connected of Latent Bookbags cases.	71
5.32	Probabilities of latent node connected of Latent Animals and Asia cases.	71
5.33	No Latent Mendel Genetics network.	71
5.34	Edit distance results of No Latent Mendel Genetics cases.	72
5.35	The original structures learned by PC and FCI of No Latent Mendel Genetics cases.	72
5.36	The lower and upper bound fixes of PC and FCI outputs of No Latent Mendel Genetics cases.	72
5.37	Two typical false positives of No Latent Mendel Genetics cases (node “H” represents the latent).	73
5.38	KL and CKL results of No Latent Mendel Genetics cases.	73
5.39	Probabilities of latent node connected of No Latent Mendel Genetics cases.	74
5.40	The UI design of EM-CaMML program.	75
6.1	An example partial trigger (node “H” represents the latent).	79
6.2	Partial trigger test networks (node “H” represents the latent, and the number represents the arc strength of each arc).	80
6.3	Edit distance results of learning partial trigger.	81
6.4	EM-CaMML learned model using ini1 with sample size 5000 (node “H” represents the latent).	81

6.5	SEM learned models using ini1 with sample size 5000 (node “H” represents the latent).	82
6.6	PC/FCI learned models with sample size 5000 (node “H” represents the latent).	82
6.7	An example of PC/FCI output with a spurious latent at sample size 5000 (node “H” represents the latent).	82
6.8	Edit distance results of learning partial trigger (only H).	83
6.9	Edit distance results of learning partial trigger (only T).	83
6.10	KL results of learning partial trigger.	84
6.11	CKL results of learning partial trigger.	84
6.12	Probability of finding a latent node in a partial trigger.	85
6.13	Probability of finding a trigger subnet in a partial trigger.	85
6.14	An example of the MB of Target node T.	86
6.15	The true network used for testing the discovery of multiple latent variables (nodes “V2” and “V4” are latent).	88
6.16	The resulting partitions (in dashed lines) using MB_pick_greedy function.	91
6.17	The complete network learned by EM-CaMML using ini3 (nodes “V2” and “V4” are latent, arcs in red represent all the mistaken arcs and green arcs represents true arcs but absent in the learned network).	91
6.18	The subnet learned by EM-CaMML over MB-16 using ini2 (nodes “V2” and “V4” are latent, arcs in red represent all the mistaken arcs).	92
C.1	Big-W test networks (node “H” represents the latent).	114
C.2	Covered Big-W test networks (node “H” represents the latent).	114
C.3	Latent test networks (node “H” represents the latent).	115
C.4	No latent test networks.	115

Discovering and Learning Causal Bayesian Models with Latent Variables

Xuhui Zhang
Monash University, 2021

Supervisors: Dr. Kevin B. Korb, Dr. Steven Mascaró and Prof. Ann E. Nicholson

Abstract

The causal discovery of Bayesian networks is an active and important topic in artificial intelligence, as sources and volumes of data continue to grow along with the popularity of Bayesian modeling methods. Causal Bayesian networks allow people to investigate causal relationships and modeling under uncertainty in an intuitive fashion. However, in many real world cases, some variables cannot be directly measured or people are simply unaware of their existence; these are called latent variables. Relatively little research has been done into how to explicitly incorporate latent variables into causal model discovery. In many cases, the influence of latent variables reveals itself in patterns of measured dependency that cannot be reproduced using the observed variables alone, under the assumptions of the causal Markov property and faithfulness. That is, latent models will do a better job of representing observed dependencies than fully observed models, and hypothesizing the existence of latent variables provides an advantage in the causal discovery process. In this thesis, we develop a unique explicit process for positing latent variables and incorporating them in a metric-based causal discovery program. We develop and test an explicit program for discovering dependency patterns (“triggers”) in sample data that suggest the presence of latent variables. We develop a new MCMC algorithm incorporating Expectation Maximization for parameterizing latent subnets in the general causal discovery process. We look at incorporating Markov Blanket discovery as a means of scaling up the process to multiple latent variables in larger networks, and we conduct a variety of experiments demonstrating the viability of these techniques in comparison with existing latent discovery algorithms.

Discovering and Learning Causal Bayesian Models with Latent Variables

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

This thesis includes the following publications and the co-authors listed reflect the active collaboration between the supervisors and myself, the candidate, working within the faculty of Information Technology:

- Zhang, X., Korb, K. B., Nicholson, A. E. and Mascaro, S. (2017). Applying dependency patterns in causal discovery of latent variable models, *Australasian Conference on Artificial Life and Computational Intelligence*, Springer, pp. 134-143.
Contribution: Xuhui Zhang (70%), Kevin Korb (15%), Ann Nicholson (5%), Steven Mascaro (10%)
- Zhang, X., Korb, K. B., Nicholson, A. E. and Mascaro, S. (2016). Latent variable discovery using dependency patterns, *arXiv preprint arXiv:1607.06617*.
Contribution: Xuhui Zhang (70%), Kevin Korb (15%), Ann Nicholson (5%), Steven Mascaro (10%)

Xuhui Zhang
January 2021

Acknowledgments

I would like to thank my supervisors, colleagues, family and friends for the great deal of support that I have received from them throughout my PhD journey.

First and foremost, I would like to express my deep appreciation to my supervisor Dr. Kevin Korb at the Faculty of Information Technology of Monash University. It is truly an honor to be his PhD student. Without his patience and cheerfulness, this work would not have been possible. I owe deep gratitude for all his guidance and moral support over the years, which have been invaluable for both my research and career. His deep understanding of a diverse range of fields has been extraordinarily helpful in completing this thesis.

I would like to thank Dr. Steven Mascaro for his cooperation and cheerfulness in guiding me through this project. Without his assistance and dedicated involvement throughout the process, this thesis would have never been accomplished. I must also thank Prof. Ann Nicholson, for her insightful comments and constructive recommendations on this project.

Getting through my dissertation required more than academic support, and luckily I have many friends to thank for their great support for life. I express my gratitude and appreciation for their friendship. Yang Li, Wilson Alberto Torres, Ye Zhu, James Collier, Yifei Hu, Kai Siong Yow, Hân Duy Phan, Ming Liu, Ying Yang, Yating Zhang, Yuqing Xiong, Joon Won Kang, Cael Skene, Patrick Semple and many others who have been unwavering in their professional and personal support during my PhD study. Without each and every one of you, this thesis would not have been achievable.

Most importantly, none of this could have happened without my family. My sister, Xuhong Zhang, is my mental support for my whole life. My dear parents and other family members, thank you all for your unceasing encouragement and unconditional love.

I gratefully acknowledge Monash University for providing me with the Monash Graduate Scholarship (MGS), as well as all the professional and personal development experiences I received during my research.

Last but not least, I would like to thank all the helpful and enthusiastic staff, present and past, at the Faculty of Information Technology, especially Danette Deriane, Julie Holden and Allison Mitchell. Thank you.

Xuhui Zhang

Monash University
February 2021

Chapter 1

Introduction

Bayesian networks (BN)(Pearl, 1988), as graphical structures for machine learning, are powerful and popular tools for reasoning about uncertainty, representing potentially complex probability distributions. While sparse networks are tractable, generating them from human expert opinion reintroduced the so-called knowledge bottleneck for constructing expert systems, so it was natural that attention quickly turned to their automated generation from sample data (Spirtes et al., 1993). While there has been a great deal of work on the machine learning of Bayesian networks since then, relatively little has treated the learning of Bayesian networks with latent (or hidden) variables, despite latent variable models being one of the most widely used modeling approaches, for example, in the social sciences (Kao et al., 2012; Yang et al., 2019; Chen et al., 2019). In particular, what enables latent variable discovery is the probabilistic dependencies between variables, which will typically be representable only by a proper subset of the possible causal models over those variables, and therefore can provide evidence in favour of those models and against the remaining models, as will be shown by the Bayes factor.

CaMML (Causal Discovery via MML) is a well-established search-and-score causal discovery program (Korb and Nicholson, 2010; O’Donnell, 2013), and has been successfully applied to many different areas, but not previously to latent variable models. It applies the Minimum Message Length (MML) metric (Wallace et al., 1996) and Markov Chain Monte Carlo sampling (MCMC), more specifically the Metropolis-Hastings method (Hastings, 1970), to learn the best causal model given observational data. The posterior distribution over the causal model space is sampled, using an MML score. However, the current MML metric does not deal with latent variables, and how to integrate latent variables into the sampling process has been an open problem.

How to formalize the process of discovering latent variables and models associated with them using MML is the main goal of this PhD project. As learning BNs is an NP-hard problem (Chickering et al., 1994), we decided to search for specific dependency patterns in the sample data that indicate the likely presence of latent variables (that we call “triggers”), which can then be applied to heuristic latent discovery learning to speed up the process. Our goal was to enhance the existing CaMML program to take a set of such potential triggers as an additional input into an MCMC sampling process over the latent model space, with a new MML metric accommodating latent variables and a new search employing the triggers (Zhang et al., 2017). The result should be, ideally, a CaMML which performs the same, or similarly, to the prior version of CaMML when latents are not present, and better when they are, including good performance in identifying and connecting latent variables to the observed (non-latent) variables.

1.1 Motivation and Objectives

In cases where the conditional dependencies among observed variables indicate the presence of latent variables rather than just a fully observed model, introducing a latent variable can help encode the true dependencies in the data (Spirtes et al., 1995). Of course, if latent variables are already known to exist, we can directly insert latent nodes into the structure, using algorithms such as gradient descent or EM to estimate their associated parameters. If the structure is not known, some causal learners, such as SEM, assume there is a latent variable, and learn the best structure around it. However, automatically detecting the existence of a latent variable in a certain position without prior knowledge or human intervention is a more difficult task, but it is also more useful.

For latent variable discovery, there are several things we want to achieve:

- search for specific dependency patterns that indicate the existence of a latent variable
- apply such patterns in a preprocessing step to for causal learning

A more specific objective is to enhance the metric learner CaMML to discover and learn with latent variables. The current CaMML implementation only produces candidate models using the variables explicitly in the input data, i.e., observed models, and has no ability to discover and learn with latents (Korb and Nicholson, 2010). In this thesis, a new automatic end-to-end program, “EM-CaMML”, is proposed, which explicitly detects the existence of latent variables without using any prior knowledge and returns a ranked list of fully parameterized causal Bayesian networks. This thesis will develop or propose:

- an efficient preprocessing step to discover latent variables and propose causal relations between them and observed variables;
- an extension to the current MCMC sampling process that covers the latent model space, incorporating a new EM parameter learning algorithm;
- an approach to learning multiple latent variables, i.e., scaling up EM-CaMML to deal with higher dimensional problems, using a parallelizable heuristic algorithm.

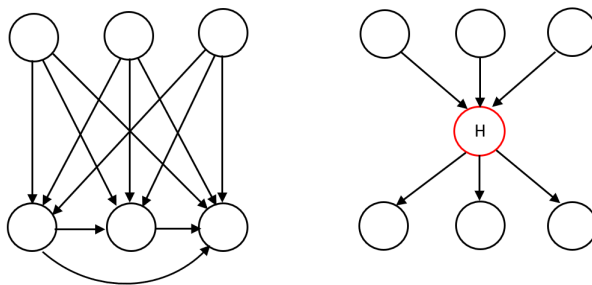


Figure 1.1: Example of how a latent node can simplify a fully connected model (node “H” represents the latent).

It’s worth noting that a latent variable not only can be used to explain the dependencies that otherwise cannot be satisfied with faithful observed networks, but can also be beneficial just for simplifying causal structure. For example, as shown in Figure 1.1, the left model is clearly less useful than the right, being over-complicated (Friedman et al., 1997). Inserting a latent node in the middle greatly simplifies the structure, reducing the number of parameters and so needing less data to achieve robust parameter estimates. So we need to consider this scenario explicitly in EM-CaMML’s development.

1.2 Contributions and Outline

In this thesis, we try to answer a number of questions about discovering latent nodes in causal Bayesian models using sample data and about extending CaMML to learning with latent nodes:

1. How can we systematically find triggers?
2. How can we best use triggers in latent causal discovery?
3. How can we enhance the metric learner CaMML to sample the latent model space?
4. How can we scale up latent discovery?

The corresponding contributions in this thesis are:

- We first formulated the problem of systematically finding dependency patterns which can be faithfully represented using a latent node, but cannot be faithfully reproduced using observed variables alone. This led to an algorithm for generating such patterns (triggers) systematically, so that they may then be explicitly applied in latent variable discovery. Then we developed a latent node detect function, trigger discovery, that would create a latent node on an as-needed basis. We combined this constraint-based technique with a metric-based general causal discovery algorithm to create a hybrid latent variable learner, exploiting the advantages of both approaches.
- We present our work in developing an extension of CaMML, named “EM-CaMML”, that is able to detect and learn with latent variables using a modified MML metric. It is a practical tool which combines constraint-based ideas and MML metric sampling. This chapter also involves different types of test cases, and EM-CaMML proved itself as a well-balanced learner compared to others. EM-CaMML has been published as open source software with a well designed user interface (UI) and it is able to visualize and output a fully parameterized BN.
- We demonstrate the ability of CaMML to be run using variable partitions and its ability to discover multiple latent variables from each partition. Although this function is shown just in a proof of concept study, rather than in a practical implementation, we find that EM-CaMML has a potential for dealing with larger-real world problems by applying recent research on Markov blanket (MB) discovery.

1.3 Thesis Layout

The remainder of this thesis is organized as follows:

- In **Chapter 2** and **3**, we present some basic background knowledge for Bayesian networks, including definitions of key vocabulary. Then several popular causal discovery algorithms that learn with latent variables are reviewed, as well as some basic methods employed by CaMML. We also introduce different evaluation metrics and explain how we adapt them to score causal models with latent variables.
- In **Chapter 4**, we demonstrate our new systematic search algorithm for trigger discovery. Execution time analysis shows that our parallel version of the search algorithm can have a significant benefit in running time.

- In **Chapter 5**, details of EM-CaMML are explained and a comparison of different EM integration methods are examined. An experimental evaluation is conducted, comparing EM-CaMML’s latent discovery with the main alternative algorithms, using a variety of evaluation measures.
- In **Chapter 6**, we cover two extensions to the core EM-CaMML method. The first is learning “partial triggers”, where the latent node in the generating model can explain part of the observed dependencies, while an observed node explains the remainder. The idea is to examine latent discovery algorithms in slightly more challenging environments than where the true explanatory causes are unambiguous. The second extension looks at supporting EM-CaMML in learning higher dimensional real-world networks with latent variables. This examines the use of MB discovery as a preprocessing step, incorporating the results as prior information about partitions of variables, and how CaMML can merge these results into a complete network.
- **Chapter 7** provides discussion about the primary contributions of this thesis and what future work might ensue.

Finally, the Glossary and Notation chapters are included to improve the readability of the thesis, and we show all the supplementary information in the appendices that help to clarify some of the experiment results.

Chapter 2

Bayesian Networks

In this chapter, we present some background about Bayesian networks (BN). First, we define and illustrate Bayesian networks and also introduce parameter estimation. We follow by explaining the difference between non-causal and causal Bayesian networks. In Section 2.1.2 we explain the d-separation rules in detail which can graphically explain the conditional dependencies in a given model. Such rules play an important role in discovering and learning latent variables in this thesis. Then Markov equivalence is described in Section 2.1.3, which explains why some causal discovery algorithms return an equivalence class rather than a fully directed model. Finally, we discuss the difference between causal discovery and regression in Section 2.1.4 and explain why causal discovery has been chosen for this thesis.

2.1 Bayesian Network Basics

Bayesian networks¹ are probabilistic graphical models that implement Bayesian inference for probability updating, taking advantage of a Directed Acyclic Graph (DAG) to factorize (and simplify) the computations. Bayesian networks contain nodes (either discrete or continuous) corresponding to distinct random variables, edges representing direct dependencies among the corresponding variables, and a Conditional Probability Distribution (CPD) for each node. The CPD defines the relationship between a given variable and its direct parents, providing the probability of each variable’s value given every possible combination of parent values (Korb and Nicholson, 2010). If variables are discrete, the CPD can be represented as a Conditional Probability Table (CPT). While conditional distributions may be either discrete or continuous, in this thesis we consider discrete variables only.

Figure 2.1 shows the Sprinkler network, a simple example BN with four discrete nodes. We can see that there are some direct relationships based on the structure: *Sprinkler* depends on *Cloudy*, *Rain* depends on *Cloudy*, *Wet Grass* depends on both *Sprinkler* and *Rain*. Conditional probabilities can be read from the corresponding CPT, so $p(\textit{Wet Grass} = T | \textit{Sprinkler} = T, \textit{Rain} = F) = 0.9$. Another important concept of Bayesian networks is “probabilistic reasoning”, which occurs as a flow of information through the network. In this process, a BN can be used to find a posterior distribution over unobserved variables by conditioning upon newly observed variables (“evidence” variables), and allowing a “flow” of information to occur through the connections in the network. This is sometimes called “propagation”, “conditioning” or “belief updating”.² For example, if we set *Sprinkler* to be true (as evidence), then the chance of *Wet Grass*

¹Sometimes called Bayes networks, belief networks, etc.

²Note that such flow does not always follow the direction of the arcs. Indeed, it’s not always a “flow”, since sampling techniques may be used, for example.

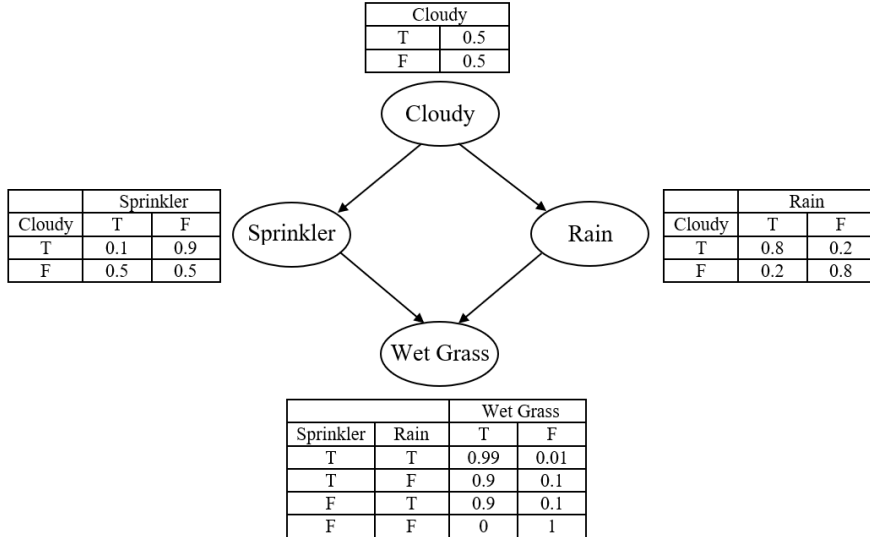


Figure 2.1: Sprinkler network.

being true will increase because of the propagation through $Sprinkler \rightarrow Wet\ Grass$ and $Sprinkler \rightarrow Cloudy \rightarrow Rain \rightarrow Wet\ Grass$. On the other hand, if we first learn the value of $Wet\ Grass$ is true, this evidence will increase the chances of $Rain$ being true because of $Wet\ Grass \rightarrow Rain$, and also increase the chance of $Sprinkler$ being true because of the flow $Wet\ Grass \rightarrow Sprinkler$ as well as $Wet\ Grass \rightarrow Rain \rightarrow Cloudy \rightarrow Sprinkler$.

As we can see, Bayesian networks are graphical models capable of displaying relationships intuitively and clearly. Where $X \rightarrow Y$ represents a directed arc, we say X is a parent of Y and Y is a child of X . The indegree of a node X is the number of parents it has, the outdegree is the number of its children, and the degree of a node is the number of nodes adjacent to it, i.e., the sum of its indegree and outdegree. When these directed relationships are explicitly causal, as in the Sprinkler network, we have causal Bayesian networks. In part because causality is an intuitive concept, in many ways more intuitive than the conditional probabilities to which they give rise, causal Bayesian networks have become a popular modeling tool, with applications ranging from medical diagnosis (Velikova et al., 2014; Zarikas et al., 2015) to cyber security (Mueller et al., 2019; Xie et al., 2010).

If the structure is known, an important additional task for learning a BN is to estimate the CPT parameters for each node from empirical data. One of the simplest methods is Maximum Likelihood Estimation (MLE), which finds model parameters which best fit the training data. In other words, MLE maximizes the probability of the empirical data given the model parameters and structure. For a discrete BN, let $V = (X_1, X_2, \dots, X_T)$ represent the set of nodes in a BN with T nodes and Θ the current parameter assignment. Then, for a node i whose value is k , and given j is the value for its parent set ($\pi(X_i)$), the conditional probability can be denoted as:

$$\theta_{i,j,k} = P(X_i = k | \pi(X_i) = j) \tag{2.1}$$

then the joint log-likelihood $L(X; \Theta)$ will be:

$$\begin{aligned}
L(X; \Theta) &= \log \left(\prod_{i=1}^T P(X_i | \pi(X_i)) \right) \\
&= \log \left(\prod_{i=1}^T \theta_{i, \pi(X_i), X_i} \right) \\
&= \sum_{i=1}^T \log \theta_{i, \pi(X_i), X_i}
\end{aligned} \tag{2.2}$$

For a given dataset D that contains N independent samples d_1, d_2, \dots, d_N , if we assume no missing data, then each sample d_n is actually one instantiation of each node where $d_n = (X_1^n, X_2^n, \dots, X_T^n)$, indexed by $1 \leq n \leq N$. We can then derive the log-likelihood of D as:

$$\begin{aligned}
L(D; \Theta) &= \sum_{n=1}^N L(X^n; \Theta) \\
&= \sum_{n=1}^N \sum_{i=1}^T \log \theta_{i, \pi(X_i^n), X_i^n} \\
&= \sum_{i,j,k} \log(\theta_{i,j,k}) N_{i,j,k}
\end{aligned} \tag{2.3}$$

where $N_{i,j,k} = \sum_{n=1}^N \mathbb{1}_{\pi(X_i^n)=j, X_i^n=k}$ that counts the number of samples in D such that the values of node X_i are equal to k and its parents values $\pi(X_i)$ equal j . Here $\mathbb{1}$ denotes an indicator function so that $\mathbb{1}_{\pi(X_i^n)=j, X_i^n=k}$ equals 1 if $\pi(X_i^n) = j, X_i^n = k$ and otherwise 0. So, in order to maximize the log-likelihood $L(D; \Theta)$, the chosen value $\theta_{i,j,k}$ will be:

$$\hat{\theta}_{i,j,k} = \frac{N_{i,j,k}}{\sum_k N_{i,j,k}} \tag{2.4}$$

which is calculated by using a Lagrange multiplier method (Tembo et al., 2016) with the constraint $\sum_k \theta_{i,j,k} = 1$. $\hat{\theta}_{i,j,k}$ is the empirical frequency of $X_i = k$ when $\pi(X_i) = j$. For example, in the Sprinkler network, if the empirical data contains 1000 rows, and there are 640 samples where $Cloudy = T \& Rain = T$, and 160 samples where $Cloudy = T \& Rain = F$, then the estimated value $P(Rain = T | Cloudy = F)$ in $\hat{\theta}$ will be $160 / (640 + 160) = 0.2$.

The above example is solvable with MLE since there are no missing data or latent variables. In the event that there are missing data or latent variables present, we cannot estimate the parameters directly by aggregating the occurrence of the sample data matching the instantiation of corresponding nodes via MLE. This problem can be solved by using either Gradient Descent (Binder et al., 1997) or the Expectation Maximization (EM) algorithm (Dempster et al., 1977; Lauritzen, 1995).

2.1.1 Causal Bayesian Networks

A causal BN is a DAG where each nonroot node is the direct causal effect of its parents; it provides a flexible tool that can be used to formalize, measure, and support causal reasoning to solve problems concerning an underlying dataset (Pearl and Verma, 1991). In all BNs, an arc represents a probabilistic dependency of the child on the parent, while in causal BNs, each arc represents a direct causal relation that also induces a probabilistic dependency. Modeling with BNs presupposes that the absence of arcs indicates an absence

of direct dependencies in the system being modeled, which is called the “Markov property” (Pearl, 1986), or for causal BNs the “causal Markov property”. This is a precondition for the model being adequate for the domain. The converse condition is called “faithfulness” (Meek, 1995): corresponding to each active pathway (d-connection, Section 2.1.2 below) is a probabilistic dependence in the system being modeled. A lack of faithfulness is not a fatal problem for a model; that is, the model can nevertheless properly represent the probabilistic system in question. However, it suggests that a simpler model may be adequate for representing the probability distribution. In causal discovery, it is generally assumed that learned models will be faithful, but there are no guarantees. Given the Markov property, any variables d-separated (see details in Section 2.1.2) in a BN are independent in the system being modeled, given a set of observations. Given these independencies, the joint distribution for a Bayesian network with variables X_1, \dots, X_k factorizes as:

$$P(x_1, \dots, x_k) = \prod_{i=1}^k P_i(x_i | \pi(x_i)) \quad (2.5)$$

where $\pi(x_i)$ denotes the joint values of the parents of variable X_i in the network.

Searching the space of causal models for those which can explain the probabilistic dependencies shown in the data is what causal discovery is based upon. However, learning a causal model is never an easy job, especially when allowing for latency. And, in principle, it is important to allow for latency. A great many variables we now take for granted in science were originally unknown or unmeasured – i.e., latent. Gravity was a force hypothesized by Newton without his ever directly measuring it. Genes were speculative mechanisms through which phenotypic traits might be inherited before Franklin, Crick and Watson identified their chemical nature. While much can be accomplished without identifying and learning the roles of latent variables, which may be codified in phenomenal laws of a domain, coming to a deeper, unifying account often requires discovering underlying latent variables.

A BN can certainly be a non-causal model, which may represent the same joint distribution as a corresponding causal model (Chickering, 1995), but it is generally more difficult to understand what the non-causal model represents. For example, Naive Bayesian (NB) networks, which are simple but non-causal BNs, have been widely applied for spam filtering (Korb and Nicholson, 2010), but they aren’t helpful for understanding the interdependencies between attributes. The major benefit of using an NB model is its simple structure, leading to easy parameter estimation. For example, Figure 2.2 shows an NB which has a simple parent (class) with four children (attributes).

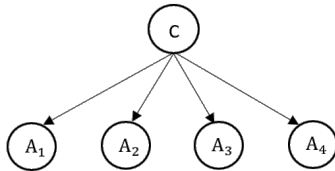


Figure 2.2: An example NB.

In this case, the CPT of each attribute (A_1, A_2, A_3 and A_4) is as simple as possible, containing a single parameter if the variables are binary. With fewer parameters than more complex models, the NB model can be trained to the same precision with far less data and/or noisier data than other models. The factorization for an NB model is:

$$p(A_1, \dots, A_n | C) = \prod_i P(A_i | C) \quad (2.6)$$

where A_i denotes the i th attribute and C represents the class variable.

In contrast, a causal BN is significantly easier to create and reason with than an explicitly non-causal BN, since it is more intuitive and understandable, as the arc “ $X \rightarrow Y$ ” is read as “ X directly causes Y ” rather than an unexplained probabilistic dependency. There may be a tradeoff with machine learning complexity, since causal BNs are typically more complex than, for example, NB models.

2.1.2 D-separation

The relationship between Bayesian network structure and conditional independencies is important for understanding how Bayesian networks work and also for how their automated learning works (Korb and Nicholson, 2010). Although we can perform conditional independence tests to check the dependencies between sets of variables given other observed variables, we can use d-separation rules (Pearl, 1988) to “read” independencies that hold true in the domain given a BN’s DAG structure and the Markov property; likewise, we can read dependencies, given faithfulness.³ In d-separation, “d” stands for “direction-dependent”, which resembles graph connectivity, but with some differences. For example, by considering the direction of traversal of a node along a path, conditioning on that node may “block” or “unblock” the path of dependence between other two nodes at each end. In other words, d-separation works as a graphical criterion for whether or not two sets of variables are independent given a third set. If two sets are “d-separated”, that implies they are conditionally independent on the third set in all probability distributions this graph can represent (Pearl, 2000) under the Markov property. Similarly, d-connection (activation, unblocked path) is a syntactical criterion for representing two sets of variables that are dependent (“d-connected”) when conditioning on a third set, under faithfulness.

D-separation implies that a path p is blocked in the following two scenarios:

1. p contains a chain $X \rightarrow Y \rightarrow Z$ (as Figure 2.3 shows) or a fork $X \leftarrow Y \rightarrow Z$ (as Figure 2.4 shows), and the middle node Y is observed:

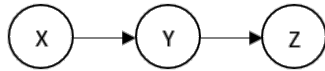


Figure 2.3: Chain structure in d-separation.

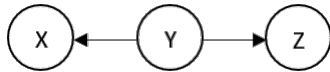


Figure 2.4: Common cause structure in d-separation.

2. p contains a collision $X \rightarrow Y \leftarrow Z$ (as Figure 2.5 shows; this structure, or its middle term, is also called a “collider” or “v-structure”) such that the middle node Y is not observed, nor are any descendants of Y .

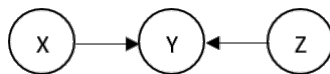


Figure 2.5: Common effect structure in d-separation.

³In general, we will be assuming both the Markov property and faithfulness, rather than repeatedly referring to these two requirements.

In the chain $X \rightarrow Y \rightarrow Z$ and the fork (common cause) structures, $X \leftarrow Y \rightarrow Z$, conditioning on Y will “block” the path: learning about X tells nothing about Z by giving Y as evidence. Contrariwise, in a collision (v-structure) $X \rightarrow Y \leftarrow Z$, if we condition on Y , X and Z will become dependent through this unblocked path. For example, informally speaking, subsequently observing X will “partly explain” the value of Y leaving less explanatory work for Z to do. In Figure 2.1, if we already know “Wet Grass” is true, then finding the node Sprinkler being true will lower the probability that Rain is true, which is an example of “explaining away” (Pearl, 2000).

A formal definition of d-separation (Pearl, 1988) is that for a graph \mathcal{G} , with two different nodes X and Y in \mathcal{G} , and W is a set of nodes in \mathcal{G} not containing either X or Y , then X and Y are d-separated given W in \mathcal{G} if and only if there exists no undirected path U (Section 3.3) between X and Y , such that every collider on U has one or more descendants contained in W (or is itself in W) and there are no other nodes on U in W . X and Y are d-connected given W if and only if they are not d-separated given W . More informally, to determine whether two nodes are d-separated, the method is to find all paths between them of whatever orientation. If any are unidirectional and without observed variables, then the two end-nodes are d-connected. If there are no such paths, then if any path has all of its colliders activated by an observation and otherwise has no observations, then the two end-nodes are d-connected. Otherwise, the end-nodes are d-separated.

2.1.3 Markov Equivalence

As mentioned in former sections, BN models are statistical models representing a set of conditional dependencies among different nodes within the structure. In particular, they can be parameterized to represent some set of probability distributions across its variables. If two such models are Markov equivalent (or statistically equivalent) then one of them can be parameterized to represent some probability distribution if and only if the other can as well (although the parameterizations themselves will likely be distinct) (Verma and Pearl, 1990). If two models are Markov equivalent, they are said to be in the same Markov Equivalence Class (MEC).

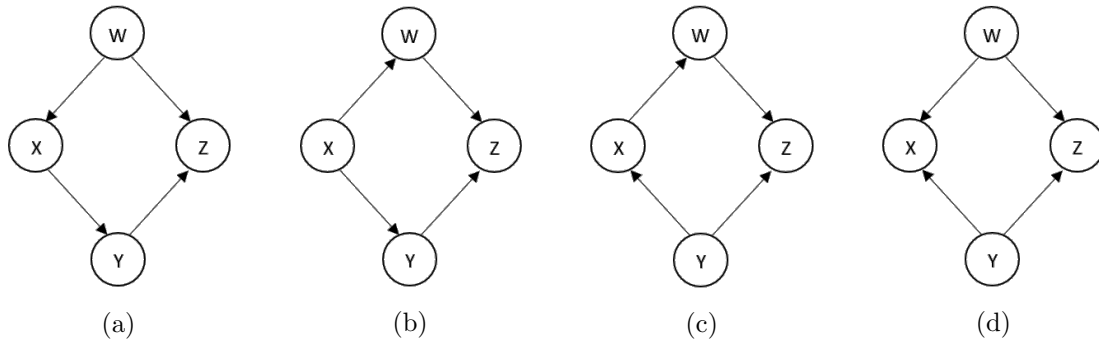


Figure 2.6: Example of Markov Equivalent Models.

For example, in Figure 2.6, model (a) is factorized by $P(W)$, $P(X|W)$, $P(Y|X)$ and $P(Z|WY)$, while model (b) is factorized by $P(X)$, $P(W|X)$, $P(Y|X)$ and $P(Z|WY)$. According to the definition of conditional probability, $P(W)P(X|W) = P(WX) = P(X)P(W|X)$, thus the values obtained from the first set of parameters can determine the values of the second, and vice versa. In such a case, the first two models are Markov equivalent to each other. Similarly, a factorization of model (c) is $P(Y)$, $P(X|Y)$, $P(W|X)$, $P(Z|WY)$ which values are implied by the parameters for either the first or second model. However, the parameters $P(W)$, $P(Y)$, $P(X|WY)$ and $P(Z|WY)$ which are required for model (d) cannot

be determined from any of the other three models. So, model (d) is in a different MEC from the other three.

Markov equivalence can also be determined by simply looking for colliders. In Figure 2.6, the first three models share the same skeleton and an uncovered v-structure “ $W \rightarrow Z \leftarrow Y$ ” (such that W and Y are not directly connected), so they are Markov Equivalent. However, while model (d) has the same skeleton as the first three models, it has different uncovered v-structures (“ $W \rightarrow X \leftarrow Y$ ” and “ $W \rightarrow Z \leftarrow Y$ ”), so it falls into a separate equivalence class.

2.1.4 Causal Discovery vs Regression

Readers may be curious about the difference between regression and causal discovery, as both generate explanatory models from observational data. Regression models aim to reduce the unexplained variation between dependent variables, and ordinary least squares regression parameterizes models by computing regression coefficients that specifically minimize unexplained variance. Statisticians using regression models generally do not believe that every independent variable which helps reduce unexplained variation in the dependent variable is necessarily a relevant causal factor (Korb and Nicholson, 2010). We know that for any variable, whenever there is random variation, its sample correlation with another variable subject to random variation will not be identically zero. Thus, the first variable can be used to reduce the unexplained variation of the second. However, regression methods have no principled way to rule out variables that may have tiny correlations between themselves, despite being evidently unrelated events. Moreover, the probability of a Type I error (getting a significant result when there is no true correlation) may be set, for example, at 5%. It follows that we must expect spurious relations when examining any large number of variables. While there are methods used for variable selection in regression, they are generally aimed at complexity reduction rather than finding causal structure.

In contrast, the conditional independence learning applied in causal discovery aims at finding simpler models that specifically encode the dependency patterns found in the data, so the use of vanishing partial correlations has clear justification. We can justify eliminating (isolating) specific variables, as well as specific arcs, in causal discovery by looking at the relation between d-separation in causal graphs and conditional independence in probability distributions. Both constraint-based and metric-based learners support principled arc addition/deletion and variable selection respectively. Details will be explained in later chapters of this thesis.

Chapter 3

Learning With Latent Variables

The value of Bayesian networks for modeling causal systems and decision making under uncertainty is becoming clearer to the wider community, as evidenced by a robust growth in their use in industry, business and government.¹ This growth is limited, however, by the classic “knowledge bottleneck” for expert systems: the time and availability of experts whose knowledge and understanding of a domain might be required to construct Bayesian networks. A consequence is the parallel growth in interest in automating the learning of networks from sample data. At the same time, Big Data has become Big Business, further fuelling interest in data mining Bayesian networks. Despite the huge volumes of data produced by many organizations, especially those doing their business on the internet, not every variable of interest is identified and measured; many practical application problems have sparse or missing data. Improving causal discovery in the presence of latent variables is an important step in bringing Bayesian network modeling to new domains.

In this chapter, we review some background for discovering and learning with latent variables in Bayesian causal modeling. It starts by explaining the necessity and benefits of incorporating latent variables and introducing the formal definition of triggers. Some well-known constraint-based learners and metric-based learners that learn with latent variables are then presented. Such algorithms are not only the benchmarks in our subsequent experiments, but also inspire the application of constraint-based methods to simplify introducing latent variables into the CaMML causal discovery program. We describe this extended CaMML in detail. Finally, we present some metrics for evaluating causal model discovery, and show how to apply them to learning with latent variables.

3.1 The Necessity of Modeling with Latent Variables

Latency is both ubiquitous and important. For example, while Galileo developed accurate mathematical laws of motion, they remained unexplained until Newton came up with the previously unmeasured concept of gravity. Gravitational waves have recently been successfully measured, but for several centuries the variable played a key role in physical theory without any direct measurement. Latent variable modelling has a long history in statistics, starting with Spearman’s (1904) work on intelligence testing. Although factor analysis and related methods are used to posit latent variables and measure their hypothetical effects, they do not provide clear means for deciding when to add latent variables given a dependency pattern amongst observed variables.

One solution is to always use a fully connected structure, since they can be parameterized to represent any observed dependency pattern at all. However, updating fully connected networks is computationally exponential, and the parameterization process for

¹Google Ngram Viewer shows a greater than 800% increase in book titles with “Bayesian Network” since 1995.

them is likewise exponential. Moreover, this fully connected structure abandons the attempt to understand a domain causally and forces a purely phenomenal approach. Assuming we are interested in a causal understanding of a domain, whether on simplicity grounds or in order to predict the consequences of interventions, a willingness to find and model with latent nodes can often simplify the model while retaining representation for the observed probability distribution. For example, we showed in Figure 1.1 a simpler model with a latent node H that can represent the same dependency structure as the much more complex model on the right, as Friedman pointed out in his work (Friedman et al., 1997).

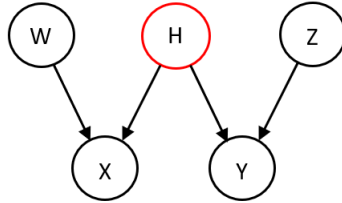


Figure 3.1: The Big-W, a causal structure with four observed variables and one latent variable H .

Another substantial advantage of latent variable models is that they can often better encode the conditional dependencies and independencies in the data. Regardless, in many cases, the influence of latent variables is real and important, and optimal modeling cannot be done without them. As shown in Figure 3.1, if we assume the data show the independencies $W \perp\!\!\!\perp \{Y, Z\}$ and $Z \perp\!\!\!\perp \{X, W\}$, it is impossible to construct a network in the observed variables alone that reflects both of these independencies while also reflecting the dependencies implied by the d-connections in the latent variable model (i.e., with the model being faithful). We call dependency patterns with this property — incapable of being encoded in the observed variables alone assuming both the causal Markov property and faithfulness — *triggers*,² since they can act as triggers to extend a search of the causal model space to incorporate latent variables. Note that we only consider triggers with only one latent node due to the search complexity and time constraint. More details are shown in Chapter 4.

This kind of latent variable discovery began with the CMU causal discovery group associated with Clark Glymour, who were looking at these issues since before Verma and Pearl invented their constraint-based approach, as illustrated in their *Discovering Causal Structure* (Glymour et al., 1987).³ Glymour et al. (1987) limited themselves to linear networks and used partial correlation tests to judge dependence and independence; these tests remain in their program TETRAD V, which also includes PC and other discovery algorithms (Spirtes et al., 2016). Their work included the discovery of latent variable models suggested by patterns of observed positive and vanishing correlations (and partial correlations) that may be due to an unmeasured common cause (Glymour and Spirtes, 1988).

3.2 Causal Discovery With Latent Variables

Causal discovery, as implied by its name, attempts to discover and build models that accurately reflect the underlying causal relationships in the sample data. However, building a causal model has never been an easy task (Korb and Nicholson, 2010), as a huge number of

²Note that we will indifferently refer to the causal models that generate these dependency patterns triggers as well.

³Glymour’s group and Pearl’s group discovered the existence of each other shortly thereafter. The CMU PC algorithm was the first practical implementation of Verma and Pearl’s IC algorithm.

possible causal structures can be derived even given a limited number of variables (Asher and Asher, 1976). Moreover, it is difficult to make a balance between efficient inference and avoiding over-fitting (Korb and Nicholson, 2010). In order to solve such problems, there are a number of algorithms that have been developed to learn Bayesian Networks for causal models which fall into two main categories.

The first category (introduced by Verma and Pearl (1991)), named constraint-based algorithms, which applies certain statistical tests for conditional independence along with some rules, aims to produce structures or patterns (i.e., the IC algorithm). To be more specific, these algorithms interrogate the sample data for sets of variables for finding conditional independencies (e.g., via discovering uncovered v-structure), and the results are used as a guide for the search through a model space. Famous examples are PC and FCI algorithms, and their extensions (Pearl and Verma, 1991; Spirtes et al., 1993).

The second category, named metric-based algorithms, apply scoring metrics in the search process over an exponential model space, attempting to find a model which either maximises or minimises the score. The scoring metric functions could be, for example, the Minimum Message Length (MML) score (Wallace and Boulton, 1968; Wallace et al., 1996), which is used in the program CaMML (see details in Section 3.4.3) or the Minimum Description Length (MDL) score in various encoding versions (e.g., Lam and Bacchus, 1994; Suzuki, 1996; Friedman and Goldszmidt, 1998). Another well-known function is the K2 metric (Cooper and Herskovits, 1991), which was the first significant attempt of BNs using a Bayesian approach in structural learning, as well as the BDe score, proposed by Heckerman et al. (1995), which is based on the K2 score but considers the edit distance in the given expert-elicited network as prior knowledge. For metric-based learners, apart from CaMML, one of the most widely used learners is called “Structural Expectation Maximization (SEM)”, developed by Nir Friedman (Friedman et al., 1997), which optimises BIC (Schwarz et al., 1978) or MDL scores using EM algorithm.

The general process of a metric-based learner is to search through a number of candidate models, and evaluate them according to a selected scoring metric. These models will be altered iteratively and re-evaluated until the best score reaches convergence. Unlike constraint-based algorithms, there is no testing for specific dependency structure in metric-based algorithms explicitly, as the data will be best accommodated by models (e.g., as reflected in the maximum likelihood) that have the most appropriate structure. Additionally, instead of returning a class of Markov equivalent models like constraint-based learners do, most metric-based algorithms converge with a set of ranked explicit models, and this will be an advantage under some circumstances.

All varieties of causal discovery algorithms assume the causal Markov property; that is, where the data show (conditional) independencies, the models discovered will (generally) likewise show independencies, in the form of d-separation. Similarly, they assume faithfulness, meaning the models discovered will (generally) not include arc structures (d-connected relationships) which carry no conditional dependencies. Faithfulness implies that a Causal BN is a proper encoding of patterns of dependencies between measured variables. While there is some dispute in the philosophy of science about the proper boundaries of these assumptions (Arntzenius, 1999), causal discovery algorithms operate non-problematically under them across a wide range of problems, and we assume them in our work here. Under these assumptions, certain dependency patterns (the triggers described above) can reliably indicate the presence of unmeasured causes which will be explained in detail in Chapter 3.

3.3 Constraint-based Learners

Before exploring some of the well-known constraint-based algorithms, the different types of graphs used by them, and their properties, need to be explained. The difference between them is in the kinds of arcs they contain. An **undirected graph** (e.g., G1 shown in Figure 3.2) contains only undirected arcs (e.g., $A-B$). While a **directed graph** (e.g., G2 shown in Figure 3.3) contains only directed arcs (e.g., $A \rightarrow B$), without any bi-directional arcs. Note that a DAG is a special case of a directed graph, containing no cycles. A **directed path** is a sequence of nodes starting with A and ending with B , and for every pair of adjacent nodes X, Y which occur in that order, there exists the arc between them ($X \rightarrow Y$). For example, $\langle C, D, F \rangle$ is a directed path in G2. In contrast, any direction of arcs between adjacent nodes in an **undirected path** is allowed; thus directed paths are special cases of undirected paths. For example, $\langle C, D, E, F \rangle$ is an undirected path in G2, as there is an arc between adjacent nodes along the path (Spirites et al., 1993).

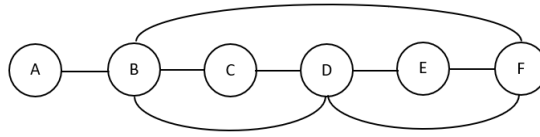


Figure 3.2: An undirected graph, G1.

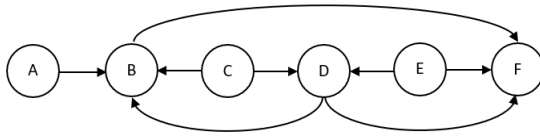


Figure 3.3: A directed graph, G2.

Given a DAG \mathcal{G} over a set of variables \mathcal{V} (e.g., $\mathcal{V} = \{A, B, C, D, E, F\}$ in G2), \mathcal{O} is an observable subset of \mathcal{V} containing two different nodes A and B , and we use $\mathcal{O} \setminus \{A, B\}$ to represent a set of nodes in \mathcal{O} that are not members of $\{A, B\}$. Verma and Pearl (1990) introduced the definition of **inducing path**. That is, an undirected path \mathcal{U} between A and B is an inducing path relative to \mathcal{O} if and only if every node in \mathcal{O} on \mathcal{U} except for the endpoints is a collider on \mathcal{U} , and every collider on \mathcal{U} is a parent of either A or B . This tells us that A and B are not d-separated by any given subset Z of $\mathcal{O} \setminus \{A, B\}$ as evidence if and only if there is an inducing path over the subset \mathcal{O} between A and B . For example, in G2, the undirected path $\mathcal{U} = \langle A, B, C, D, E, F \rangle$ is an inducing path over $\mathcal{O} = \{A, B, D, F\}$ as each collider on \mathcal{U} (B and D) is an ancestor of the endpoint F , and the nodes on \mathcal{U} that are also in \mathcal{O} except the endpoints A and F (which are B and D) are colliders on \mathcal{U} . So given the definition of inducing path, Verma and Pearl (1990) proposed the definition of **inducing path graph** \mathcal{G}' over \mathcal{O} for DAG \mathcal{G} if and only if:

- i \mathcal{O} is an observable subset of nodes in \mathcal{G} ;
- ii there is an arc $A \leftrightarrow B$ in \mathcal{G}' if and only if A and B are in \mathcal{O} , and there is an inducing path between A and B in \mathcal{G} over \mathcal{O} which is into A and into B ;
- iii there is an arc $A \rightarrow B$ in \mathcal{G}' if and only if A and B are in \mathcal{O} with no arc $A \leftarrow B$ in \mathcal{G}' , and there is an inducing path in \mathcal{G} between A and B over \mathcal{O} that is out of A and into B ;
- iv there are no other arcs in \mathcal{G}' .

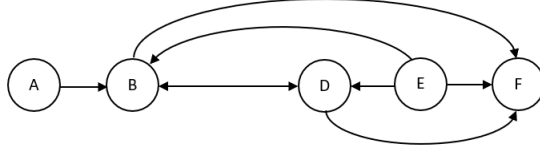


Figure 3.4: An example of inducing path graph G_3 .

For example, the graph shown in G_3 (shown as in Figure 3.4) is the inducing path graph of the DAG shown in G_2 over $\mathcal{O} = \{A, B, D, E, F\}$. As the undirected path $\langle B, C, D, E \rangle$ is an inducing path in G_2 over $\mathcal{O} = \{A, B, D, E, F\}$, thus there is an arc present between B and E in G_3 . Similarly, as there are two inducing paths $\langle B, D \rangle$ and $\langle B, C, D \rangle$ over $\mathcal{O} = \{A, B, D, E, F\}$, there have to be a bi-directional arc between B and D in G_3 . In other words, every arc in G_3 represents an inducing path of G_2 over $\mathcal{O} = \{A, B, D, E, F\}$. Based on the definition of inducing path graph, if node A and B are adjacent in an inducing path graph, then they are d-connected given every subset of $\mathcal{O} \setminus \{A, B\}$. In other words, if A and B are not adjacent in the inducing path graph, then they are d-separated given some subset of $\mathcal{O} \setminus \{A, B\}$.

In an inducing path graph, there could be two different types of arcs: $A \rightarrow B$ (directed arc) entails that every inducing path over \mathcal{O} between A and B is out of A and into B , while $A \leftrightarrow B$ (bi-directional arc) entails that every inducing path over \mathcal{O} between A and B is into A and into B . For the latter type of arc, Spirtes, Glymour and Scheines (1993) clearly stated that the bi-directional arc can only occur when there is a **latent common cause** between A and B , and this statement tells us how constraint-based learners discover latent variables. However, there could be more types of arcs contained in a **partially oriented inducing path graph** (POIPG), which is the output graph format of some constraint-based learners (e.g., FCI algorithm) proposed by Spirtes, Glymour and Scheines (1993). Apart from the directed arc and bi-directional arc, there are another two sorts of arcs: $o \rightarrow$ and $o - o$, where the circles at the ends indicate that the algorithm cannot determine whether the ends should be arrowheads or tails. It should be pointed out that the symbol “*” is used as a metasymbol to represent any of the three types of ends (“>”, “o” or EM (the empty mark)), but “*” is only used to represent EM or “>” that can appear in an inducing path graph and does not occur in a POIPG. The purpose of using a POIPG is to represent the adjacencies in \mathcal{G}' , and some of the directions of the arcs in \mathcal{G}' that are common to all inducing path graphs which share the same d-connection relations as in \mathcal{G}' . We use $\text{Equiv}(\mathcal{G}')$ to represent the set of inducing path graphs over the same nodes who share the same d-connections as in \mathcal{G} . In other words, every inducing path graph in $\text{Equiv}(\mathcal{G}')$ has the same adjacencies. So a graph π is a POIPG of \mathcal{G} with inducing path graph \mathcal{G}' over \mathcal{O} if and only if:

- i if there is any arc between A and B in π , it has to be one of the six types: $A \rightarrow B$, $A \leftarrow B$, $A o \rightarrow B$, $A \leftarrow o B$, $A o - o B$, or $A \leftrightarrow B$;
- ii π and \mathcal{G}' have the same nodes;
- iii π and \mathcal{G}' have the same adjacencies;
- iv if $A o \rightarrow B$ occurs in π , then every inducing path graph in $\text{Equiv}(\mathcal{G}')$ has to contain either $A \rightarrow B$ or $A \leftrightarrow B$;
- v if $A \rightarrow B$ occurs in π , then every inducing path graph in $\text{Equiv}(\mathcal{G}')$ has to contain $A \rightarrow B$;
- vi if $A * - * \underline{B} * - * C$ is in π , then the arcs between A and B , and B and C do not collide at B (indicated by the underline) in any inducing path graph in $\text{Equiv}(\mathcal{G}')$;

- vii if $A \leftrightarrow B$ is in π , then every inducing path graph in $\text{Equiv}(\mathcal{G}')$ has to contain $A \leftrightarrow B$;
- viii if $A \text{ } o\text{--}o \text{ } B$ is in π , then every inducing path graph in $\text{Equiv}(\mathcal{G}')$ has to contain either $A \rightarrow B$, $A \leftarrow B$ or $A \leftrightarrow B$.

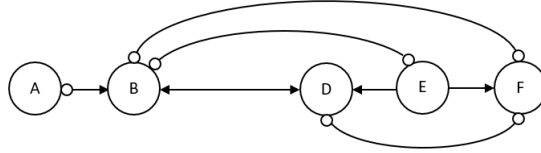


Figure 3.5: An example of partially oriented inducing graph G4.

For example, G4 (shown in Figure 3.5) is a POIPG in which not all arcs are oriented. Again, the bi-directional arc between A and B indicates there is a latent common cause. POIPG is normally the intermediate result in most cases of many constraint-based learners. The first step is to determine the node adjacencies. Note that in a POIPG, A and B are adjacent in π if and only if they are d-connected in any given subset of $\mathcal{O} \setminus \{A, B\}$. Once the adjacencies in a POIPG are determined, to make it more informative, a simple but inefficient method is to construct every possible inducing path graph with the same adjacencies as \mathcal{G}' , discard the ones that do not share the same d-connection relations as in \mathcal{G}' , and keep track of any orientation features that are shared by all graphs in $\text{Equiv}(\mathcal{G}')$. Instead of this impractical method, there are different algorithms (e.g., the FCI algorithm) that aim to make the POIPG to be informative by orienting as many arcs as possible, which is the major task for most constraint-based learners.

3.3.1 IC Algorithm

The IC (Inductive Causation) algorithm was introduced by Verma and Pearl (1990) and aims to learn a Statistical Equivalence Class (SEC) satisfying a set of conditional dependencies present in the given data. However, in order to determine whether two variables are conditionally independent (e.g., $A \perp\!\!\!\perp B | C$), this algorithm relies on an “oracle” reporting all the conditional independencies and dependencies between variables. The details of IC algorithm is shown as follows:

Input: \hat{P} a stable probability distribution on set \mathcal{V} of variables.
Output: an SEC $\hat{\mathcal{G}}$ that is compatible with \hat{P} .

The IC algorithm contains three main steps:

1. Recovers all direct dependencies as undirected arcs in $\hat{\mathcal{G}}$. For each two variables A and B , search for a set of variables S that satisfies $(A \perp\!\!\!\perp B | S)$, and if such S does not exist, place an undirected arc $A-B$.
2. Recovers some of the arc directions induced by common effect variables. For any pair of non-adjacent variables A and C (which $A-C$ does not exist) which have a common neighbour B ($A-B-C$), replace the chain structure by $A \rightarrow B \leftarrow C$ if and only if for every possible S that $A, C \notin S$ and $B \in S$, $(A \not\perp\!\!\!\perp C | S)$. Meanwhile, if there is no such S containing B that d-separates A and C , then the connections between A, B and C will be oriented as $A \rightarrow B \leftarrow C$.
3. When trying to orient all remaining arcs according to the first two steps, there are two constraints that must be satisfied:

- The orientation should not create new V-structures.
- The orientation should not create directed cycles.

Along with the three main steps, there are some additional rules (Verma and Pearl, 1992) that could help to output a maximally oriented pattern:

- **Rule 1** If there is a chain $A \rightarrow B-C$, and B and C are not adjacent, then direct $B-C$ as $B \rightarrow C$.
- **Rule 2** If there is a chain $A \rightarrow C \rightarrow B$, orient $A-B$ as $A \rightarrow B$.
- **Rule 3** If there are two chains $A-C \rightarrow B$ and $A-D \rightarrow B$, then orient $A-B$ as $A \rightarrow B$.
- **Rule 4** If there are two chains $A-B \rightarrow D$ and $A-C-D$, then orient $A-C$ as $A \rightarrow C$, and orient $D-C$ as $D \rightarrow C$.

However, as IC algorithm examines the conditional dependencies between every possible pair of variables, the first step is exponential in the number of variables and not practical for real-world scale networks. Besides the practical problems, the “oracle” used by IC algorithm is useful for proving theoretical learning properties, but does not immediately suggest a practical implementation. Different researchers have developed different algorithms which apply similar rules and attempt to make the concept more practical.

3.3.2 PC Algorithm

As an alternative to IC algorithm, Spirtes, Glymour and Scheines (1993) proposed the PC algorithm, which aims to produce the result model in a practical way. PC algorithm shares the same basic steps with the IC algorithm but starts with a fully connected model and progressively removes any arcs that cannot be justified by the conditional dependencies present in the given data, and stops when a particular level of detailed model is reached. In this way, not all possible subsets of variables need to be examined which can greatly reduce the execution time for sparse graphs, although it still remains exponential in the worst case scenario. The “oracle” used in IC algorithm is replaced by performing conditional significance tests (e.g., chi-square test) between variables. For example, if the correlation between two variables fail to meet a given acceptance level, suggesting a vanishing partial correlation, then these two variables are considered to be independent. PC starts with a fully connected, undirected graph and attempts to remove arcs iteratively by conditioning on different evidence sets with different numbers (denoted by n) of nodes. We use $\text{Adjacencies}(\hat{\mathcal{G}}, A)$ here to represent a set of nodes adjacent to A and $\text{Sepset}(A, B)$ to represent a set of nodes that d-separate A and B in $\hat{\mathcal{G}}$.

The details of PC algorithm is shown as follows:

Input: \hat{P} a stable probability distribution on set \mathcal{V} of variables.
Output: an SEC $\hat{\mathcal{G}}$ that is compatible with \hat{P} .

1. Initializes $\hat{\mathcal{G}}$ with the fully connected undirected model where every node is a neighbour to all other nodes.
2. Initialize $n = 0$.
repeat
repeat

- select an ordered pair of variables A and B which are adjacent in $\hat{\mathcal{G}}$ such that $\text{Adjacencies}(\hat{\mathcal{G}}, A) \setminus \{B\}$ has cardinality $\geq n$, and let S be a subset of $\text{Adjacencies}(\hat{\mathcal{G}}, A) \setminus \{B\}$ of cardinality n , and if A and B are d-separated given S , then delete $A-B$ and record S in $\text{Sepset}(A, B)$ and $\text{Sepset}(B, A)$;
- until all ordered pairs of adjacent nodes A and B that $\text{Adjacencies}(\hat{\mathcal{G}}, A) \setminus \{B\}$ has cardinality $\geq n$ and all subset S of $\text{Adjacencies}(\hat{\mathcal{G}}, A) \setminus \{B\}$ of cardinality n have been tested for d-separation;
- $n = n + 1$;
- until for each ordered pair of adjacent nodes A, B , the $\text{Adjacencies}(\hat{\mathcal{G}}, A) \setminus \{B\}$ is of cardinality $< n$.
3. For each triplet of nodes A, B, C such that the pair A, B and the pair B, C are each adjacent in $\hat{\mathcal{G}}$ but A and C are not, then if B is not in $\text{Sepset}(A, C)$, direct $A-B-C$ as $A \rightarrow B \leftarrow C$ if and only if B is not in $\text{Sepset}(A, C)$.
 4. repeat
 - if $A \rightarrow B$, B and C are adjacent in $\hat{\mathcal{G}}$, A and C are not adjacent in $\hat{\mathcal{G}}$, and there is no arrowhead at B , then orient $B-C$ as $B \rightarrow C$.
 - if there is a directed path from A to B , then orient $A-B$ as $A \rightarrow B$.
 until no more arcs can be oriented.

While the IC algorithm is not particularly useful in discovering latent variables, the bi-directional arcs returned by PC algorithms mean that the two variables have a statistical effect on each other and can be understood as the effect of a latent variable. In practice, the PC algorithm performs well in dealing with small scale networks with a large number of samples, and it is still a good algorithm for medium-scale networks. However, the number of errors of statistical tests increases if the sample is small or the cardinality of the conditioning set is large (Spirtes et al., 1993). Although PC was not designed to discover latent variables explicitly, as its results nevertheless indicate latent common causes, we will use it as an experiment benchmark in Chapter 5.

3.3.3 FCI Algorithm

Another well-known constraint-based algorithm is the Fast Causal Inference (FCI) algorithm (Spirtes et al., 1993), which is able to correctly and efficiently infer causal relations from conditional independence statements in large scale sample data, and detect the presence of some latent variables. It is extended from the CI (Causal Inference) algorithm (Spirtes et al., 1993), as since CI returns a POIPG, but suffers from the way the adjacencies are constructed especially for dealing with large numbers of variables, as there are too many subsets to be conditioned on in order to test the conditional independence between two variables. In addition, for discrete distributions, there are no reliable tests of independence of two variables unless the sample sizes are enormous when conditioning on a large set of other variables.

In order to solve this issue, like the PC algorithm, FCI first removes the arc between variables A and B if they are d-separated given any subsets of the nodes adjacent to them. This reduces the number of conditioning subsets to test. But as the structure is continually being updated, by finding additional arcs to delete in this manner, there will remain some incorrect arcs, since some possible evidence sets will include non-directly adjacent nodes that are missed. FCI, being a heuristic algorithm, pays some price in accuracy. After performing the first three steps of PC, a relatively “thin” and informative graph is produced, eliminating certain variables that are definitely not d-separating A and B ,

and any nodes that are not confirmed to not be in $\text{Sepset}(A, B)$ will be put into $\text{Possible-Sepset}(A, B)$ (the more tests are performed, the smaller size the $\text{Possible-Sepset}(A, B)$ would be), and do the same for $\text{Possible-Sepset}(B, A)$. If A and B are d-separated given every possible subset of $O \setminus \{A, B\}$, then it is obvious that they are also d-separated given some subset of $\text{Possible-Sepset}(A, B)$, or some subset of $\text{Possible-Sepset}(B, A)$. In order to orient arcs, we can see there is a balance between reducing the size of $\text{Possible-Sepset}(A, B)$ (by performing more d-separability tests) and the extra work required to eliminate nodes from $\text{Sepset}(A, B)$. Spirtes, Glymour and Scheines (1993) suggested that in a sparse network, we may not need to determine whether two variables are d-separated given a set C for any C containing a large number of nodes. So here we present the FCI algorithm with details as follow:

Input: \hat{P} a stable probability distribution on set \mathcal{V} of variables.

Output: a POIPG F that is compatible with \hat{P} .

1. Initialize $\hat{\mathcal{G}}$ with the fully connected undirected model where every node is a neighbour to all other nodes.
2. Initialize $n = 0$.
repeat
repeat
select an ordered pair of variables A and B which are adjacent in $\hat{\mathcal{G}}$ such that $\text{Adjacencies}(\hat{\mathcal{G}}, A) \setminus \{B\}$ has cardinality $\geq n$, and let S be a subset of $\text{Adjacencies}(\hat{\mathcal{G}}, A) \setminus \{B\}$ of cardinality n , and if A and B are d-separated given S , then delete $A-B$ and record S in $\text{Sepset}(A, B)$ and $\text{Sepset}(B, A)$;
until all ordered pairs of adjacent nodes A and B that $\text{Adjacencies}(\hat{\mathcal{G}}, A) \setminus \{B\}$ has cardinality $\geq n$ and all subset S of $\text{Adjacencies}(\hat{\mathcal{G}}, A) \setminus \{B\}$ of cardinality n have been tested for d-separation;
 $n = n + 1$;
until for each ordered pair of adjacent nodes A, B , the $\text{Adjacencies}(\hat{\mathcal{G}}, A) \setminus \{B\}$ is of cardinality $< n$.
3. Let F be the undirected graph resulting from step 2. Orient every arc as $o-o$. For each triplet of nodes A, B, C such that the pair A, B and the pair B, C are each adjacent in $\hat{\mathcal{G}}$ but A and C are not, direct $A*-B*-C$ as $A* \rightarrow B \leftarrow *C$ if and only if B is not in $\text{Sepset}(A, C)$.
4. For each pair of variables A and B adjacent in F , if A and B are d-separated given any subset S in $\text{Possible-Sepset}(B, A) \setminus \{A, B\}$, or any subset S of $\text{Possible-Sepset}(B, A) \setminus \{A, B\}$ in F , then remove the arc between A and B , and record S in $\text{Sepset}(A, B)$ and $\text{Sepset}(B, A)$.

As FCI is far more effective than most other constraint-based algorithms (Spirtes et al., 1993) especially in dealing with large number of variables, and it is capable of discovering the presence of latent variables (latent common causes), we use it as one of the benchmarks of the experiments in Chapter 5.

3.4 Metric-based Learners

As an alternative way to constraint-based learners, metric-based learners generate random models and evaluate them using a score function (e.g., BIC (Schwarz et al., 1978), MDL

(Lam and Bacchus, 1994) and MML) that considers accuracy, complexity etc. All the sampled models are scored and refined repeatedly until it has converged to a certain point or reached the maximum number of iterations. However, given the number of nodes, the search space could be exponential; Robinson (1977) proved that the number of both possible labeled and unlabeled DAGs are super exponential. Instead of using DAGs, some algorithms use patterns (e.g., SEC), but it is still not practical to search over the entire space as Gillispie and Perlman (2013) showed that the search space of labeled Markov equivalence class is also super exponential. Thus, many heuristic methods such as greedy search, genetic algorithms and simulated annealing aim to find the best model without searching over the entire space. Another solution is to use MCMC (e.g., Metropolis-Hasting) to sample a number of models and select the one which has the optimal posterior probability as the output. Users can control the sampling space by limiting the maximum number of samples, the maximum number of iterations.

For latent variable discovery specifically, perhaps the best known algorithm is Structure Expectation Maximization (SEM) algorithm, which applies a heuristic search with a score function (Friedman et al., 1997; Friedman, 1998). It utilizes EM and performs a search for the best latent structure inside the EM procedure (Friedman et al., 1997), producing a single structure with latent nodes. Among sampling methods, we introduce CaMML in this section, which applies an MCMC sampling method along with the MML score function for causal models. The original CaMML does not discover and learn with latent nodes, so we will discuss the potential for CaMML to be extended to learning with latent nodes and the details will be presented in Chapter 5.

3.4.1 EM Algorithm

When we try to learn the parameters of a BN, sometimes there are missing values in the dataset or even latent variables. In this case, if we use MLE, then it is not straightforward to learn the parameters that maximizes the log-likelihood of the dataset (by using equation 2.4) for a given structure. One of the popular solutions is the EM algorithm, which was originally proposed by Dempster et al., (1977) , that aims to learn the parameters with the presence of missing values or latent nodes. It assumes missing values are independent from the observed values and produces a point estimate of the model parameters. There are two main steps in EM: the Expectation step (E_step) which estimates the probability distribution of missing data and computes the expected sufficient statistics (i.e., sum of the posterior probability of all data points, known as “soft counts” that match both required instantiations of each node as well as its parents); the Maximization step (M_step) which re-estimates the model parameters using some kind of scoring metrics (e.g., maximum a posterior (MAP) or MLE) based on current expected sufficient statistics. For learning parameters in a BN, it starts with an arbitrary initial parameter assignment and maximizes the data likelihood by running the two steps in a loop until convergence:

Input: \hat{P} a stable probability distribution \mathcal{D} and a DAG \mathcal{G} on set \mathcal{V} of variables.
Output: a parameter set $\hat{\Theta}$ that maximizes the likelihood of \hat{P} given \mathcal{G} .

1. Initialize the parameters $\hat{\Theta}$ randomly, choose a threshold ε and maximum number of iterations n_{max} .
2. Initialize $n = 0$.
repeat
 - E_step: compute the probability distribution over missing data \mathcal{D}^* :

$$P(\mathcal{D}^*|\mathcal{D}, \hat{\Theta}) = \frac{P(\mathcal{D}|\mathcal{D}^*, \hat{\Theta})P(\mathcal{D}^*|\hat{\Theta})}{\sum_{\mathcal{D}^*} P(\mathcal{D}|\mathcal{D}^*, \hat{\Theta})P(\mathcal{D}^*|\hat{\Theta})}$$

M_step: Re-estimate $\hat{\Theta}$ by computing MLE or MAP given $P(\mathcal{D}^*|\mathcal{D}, \hat{\Theta})$:

$$\hat{\Theta} = \hat{\Theta}'; n = n + 1;$$

until $|\hat{\Theta}' - \hat{\Theta}| < \varepsilon$ or $n = n_{max}$.

As a popular algorithm, EM has been widely applied in Bayesian modeling, natural language processing, computer vision etc. However, similar to many parameter estimation methods, it generally converges to a local maxima rather than a global maxima. As the original EM only produces a point estimate of model parameters, researchers have developed different extensions that aim to learn both the model structure and parameters (see details in Section 3.4.2 and Chapter 5).

3.4.2 SEM Algorithm

The SEM was claimed as the first method that is able to learn network structure from incomplete data (Friedman et al., 1997; Friedman, 1998) for factored models (e.g., Bayesian networks, multinets, decision trees, decision graphs and other probabilistic models). It is an extension from EM algorithm and it finds the best structure inside the EM procedure. Starting with early work on defining the MS-EM and AMS-EM algorithms (Dempster et al., 1977), Friedman finalised the SEM algorithm in 1998 (Friedman, 1998) which attempts to optimize the expected Bayesian score rather than produce an approximation, as the exact Bayesian score can give a better assessment of each candidate model given the data. Like the EM algorithm, SEM has been proved to converge to a local maxima and requires a lot of computation of the expected statistics which makes the algorithm not so capable in large scale domains.

Let \mathbf{O} be the set of observed variables and \mathbf{H} denotes the set of latent variables. We use a lower-case letter \mathbf{o} and \mathbf{h} to denote a set of instantiations of \mathbf{O} and \mathbf{H} respectively in model M_n . Then the general outline of Bayesian SEM (Friedman, 1998) is shown as below:

Input: \hat{P} a stable probability distribution D and a DAG \mathcal{G} on set \mathcal{V} of variables.

Output: a parameterized model M_n by computing MAP given \hat{P} .

1. Start with a given model M_0 with random parameters $\hat{\Theta}$.

2. Initialize $n = 0$.

repeat

 Compute the posterior $P(\hat{\Theta}_{M_n}|M_n^h, \mathbf{o})$

 E_step: perform a search over models and evaluate each candidate model M_n by:

$$\begin{aligned} \text{Score}(M : M_n) &= E[\log P(\mathbf{h}, \mathbf{o}, M^h)|\mathbf{o}, \hat{\Theta}_{M_n}, M_n^h] \\ &= \sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{o}, \hat{\Theta}_{M_n}, M_n^h) \log P(\mathbf{h}, \mathbf{o}, M^h) \end{aligned}$$

where M^h denotes the hypothesis that the conditional independence assertions implied by M hold in the true joint probability distribution of \mathcal{V} , and function E denotes the expected score of the current model being examined.

M_step: select the best model M_{n+1} that maximizes $Score(M : M_n)$ among these encountered during the search.

$$M_n = M_{n+1}; n = n + 1;$$

until $Score(M_n : M_n) = Score(M_{n+1} : M_n)$.

The above outline only gives a framework, to specify the SEM completely, the search procedure has to be decided in the E_step. For example, Friedman applied a greedy hill climbing search in the SEM paper (Friedman, 1998) by performing possible arc additions, removals and reversals. At each iteration, the algorithm tries to maximize the expected Bayesian score, and Friedman claims that if a model M_{n+1} that maximizes the expected Bayesian score at each iteration, then M_{n+1} will be a better choice than M_n . In practice, we can choose a M_{n+1} in M_step that is better than M_n , but not necessarily the best, and this weaker selection criteria can make the whole process more efficient as only a subset of all possible models are evaluated.

3.4.3 CaMML

CaMML, as a well-designed metric-based learner, was originally developed by Chris Wallace and Kevin Korb at Monash University (Wallace et al., 1996). There are several implementations of CaMML, such as CaMML on linear models (Wallace and Korb, 1999) and the CaMML Java implementation which learns causal models using the Core Data Mining System (CDMS) (O’Donnell, 2010, 2013). In this section, we will briefly introduce some basic components and terminologies of CaMML and show a detailed plan of what metrics and functions need to be adapted in order to learning with latent variables.

There are three main components in O’Donnell’s CaMML implementation: a pre-sampling, a simulated annealing search and a MCMC sampling function. In all of the three components, **TOM** (Totally Ordered Model) is an important representation which contains a total ordering of nodes and a list of arcs whose direction is consistent with the given node ordering. The total node ordering here means the order of causal process irrespective of whether there is a connection between these nodes (Korb and Nicholson, 2010). For example, if we already know node A comes before B , then the direction is settled and the only remaining issue is to decide whether there is a direct dependency between them. One DAG can represent multiple TOMs, but one TOM cannot represent more than one DAG. For example, in the directed 3-variable chain structure shown in Figure 2.3, there is only one total ordering $\langle X, Y, Z \rangle$ while the common cause structure in Figure 2.4 has two different total orderings $\langle Y, X, Z \rangle$ and $\langle Y, Z, X \rangle$, and both structures are Markov equivalent. The reason why TOM is used in CaMML is somewhat a necessity, as it is a plausible view of causal structures and a correct MML coding practice (Korb and Nicholson, 2010). If use DAG only, we are ignoring the underlying causal process by allowing different but consistent TOMs to be entertained. As the causal process grows, there could be new variables being added and learned, and it is possible that there will be only one possible linear extension which is compatible with the original problem. Thus the more TOMs a DAG can represent, the more possible for it to be recognized and the greater prior probability for it to be true in the end.

To cost a TOM, a MML adaptive code implementation (O’Donnell, 2010) for comparing different structures and encoding multinomial CPT parameter of each node is used, and three different costs are included: structure cost, parameter cost and the data cost (all in “nits”) given the model. The structure cost⁴ is given as:

⁴Note that the MML cost for DAGs requires adjusting with the subtrahend $-\ln M$, where M is the number of linear extensions of the DAG (Wallace et al., 1996). However, since we are sampling the TOM (linear extension) space, rather than the DAG space, this count is already implicit in the sampling process.

$$StructureCost = \ln(k!) - nArcs \times \ln(p_{arc}) - \left(\frac{k \times (k - 1)}{2} - nArcs \right) \times \ln(1 - p_{arc}) \quad (3.1)$$

where

- k denotes the node number.
- p_{arc} is the probability of an arc existing.
- $nArcs$ is the arc number.

and the parameter cost and the data cost given the model is encoded together:

$$CPTCost = \frac{|pa| \times (|X| - 1)}{2} \times \ln \frac{\pi e}{6} + \sum_{pa_i}^{|pa|} \ln \left(\frac{(N(pa_i) + |X| - 1)!}{(|X| - 1)! \times \prod_{x_i}^{|X|} (N(pa_i, x_i)!)} \right) \quad (3.2)$$

where

- $|pa|$ and $|X|$ denotes the number of parent state combinations and current node state number respectively.
- $N(pa_i)$ is the number of samples where parent state combination = pa_i .
- $N(pa_i, x_i)$ is the number of observations where parent state combination = pa_i and child = x_i .

The first part reports the cost of the parameters, while the second denotes the cost of the data given the structure and parameters. Thus, the total MML cost of a TOM is the *StructureCost* plus the *CPTCost* of each node. These scoring functions are used in all the three components of CaMML.

The pre-sampling starts with an empty TOM, with temperature as 1.0 and p_{arc} as 0.5 (default), performs $7 \times k^3$ mutations (shown in Table 3.1, as each different mutation has a different associated probability), stores the resulting best TOM with its cost, and re-estimate p_{arc} using Equation 3.4. Then the algorithm repeats 10 times using a highly connected structure (each node has maximum 7 parent nodes), performs $7 \times k^3$ mutations, stores the best TOM and its cost, re-estimates p_{arc} using Equation 3.4. Finally, an empty TOM is used as the starting point, performs $7 \times k^3$ mutations again, and stores the best TOM found and its cost, re-estimates p_{arc} using Equation 3.4, repeat for 10 times. As the search space could be exponentially large, thus the pre-sampling phase is helpful to find a relatively good starting point for the following CaMML processes by considering the execution time and the quality of the result.

After the pre-sampling process is finished, CaMML applies a simulated annealing search starting at the best TOM found so far, for the purpose of skipping from local minima. It starts with temperature of 2.0 (explained below) and this is reduced to 1.0 uniformly by performing 12 epochs (known as “cooling down”). Each run includes $10 \times k^3$ mutations at each temperature, and the p_{arc} will be re-estimated as per Equation 3.4 if a new best TOM is found. In practice, a higher temperature can make CaMML visit more TOMs, as it is more likely to accept negative mutations. By contrast, a low temperature will make CaMML stay at a limited TOM space and will be at risk to produce a local minimum result.

Mutation	Probability	Description
Temporal	1/3	Randomly select two nodes which are neighbours in the total ordering. Reverse the arc between them if exists.
Skeletal	1/3	Toggle the existence of an arc between two randomly selected nodes.
Double Skeletal	1/6	Select three nodes that follow $X \prec Y \prec Z$ in the total ordering, then toggle the arcs $X \rightarrow Z$ and $Y \rightarrow Z$ if exist.
Parent Swap	1/6	Randomly select three nodes that $X \prec Z$ and $X \rightarrow Z$ but not $Y \rightarrow Z$, then toggle $X \rightarrow C$ and $Y \rightarrow Z$. If no such case exists, performs a different mutation.

Table 3.1: Four different mutations used in CaMML.

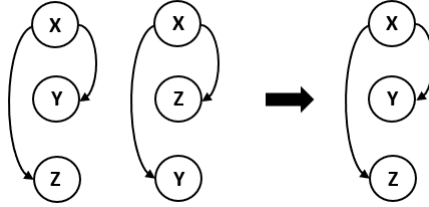


Figure 3.6: Example of how two TOMs (with different node orderings) are grouped into a DAG.

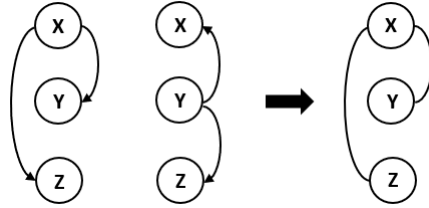


Figure 3.7: Example of how two DAGs are grouped into a SEC.

Then, in CaMML’s MCMC sampling function, starting from the best TOM found by the simulated annealing search and temperature being reset to 1.8, a number of TOMs will be sampled during each epoch. The maximum number of epochs is set as $k^3 \times 200$, or $10^3 \times 200$ if k is less than 10. In the process, a mutation is selected from Table 3.1 based on the current TOM being worked on. If the cost of new generated TOM is less than the current TOM, the new generated TOM will be accepted, or if the following equation returns true, accept:

$$\frac{Cost(M') - Cost(M)}{temperature} < -\ln(rand(0.0, 1.0)) \quad (3.3)$$

where $Cost(M')$ and $Cost(M)$ represent the total cost of the new TOM and the current TOM respectively, and $rand(0,1)$ is a random function that generates a random value between 0.0 and 1.0. Then all accepted TOM will be grouped together to estimate the posterior over DAGs. Figure 3.6 shows how two TOMs on the left which have different node orderings ($X \prec Y \prec Z$ and $X \prec Z \prec Y$) can be grouped into a DAG on the right. Then such DAGs are “cleaned” by removing any spurious arcs before being further joined into SECs, which represent a number of DAGs that are Markov equivalent given the data

provided. So, if we assume the two DAGs are cleaned on the left of Figure 3.7, then they can be joined into a (SEC) on the left. As CaMML applies MML, any SECs which are equivalent in MML score will be grouped together into an MML Equivalent Class (MMLEC). The high level hierarchy of models joined by CaMML follows a many-to-one relationship, and the final goal is to find a DAG that best trades off fit and complexity. Additionally, as the sampling process of CaMML has a uniform prior over TOMs (O’Donnell, 2010) (unless an explicit structure prior is applied), we could learn the best node ordering for each candidate DAG, which could reduce the TOM space. However, this will reduce the efficiency dramatically thus is out of our scope.

CaMML takes a record of each TOM that has been visited and uses the aggregated counts to estimate the probability distribution of all MMLECs. Such MMLECs are ranked based on MML cost and CaMML is able to export a parameterized DAG for the convenience of user needs. In the actual implementation (i.e., the version described in (O’Donnell, 2010, 2013)), before performing the sampling process, the probability of an arc presenting p_{arc} is set to be 0.5 by default, and it is updated after each epoch by:

$$p_{arc} = \frac{0.5 + nArcs}{1 + m(m - 1)/2} \quad (3.4)$$

where $1 + m(m - 1)/2$ means the maximum number of arcs that a m nodes TOM can hold.

As CaMML has been proved to be a solid causal BN learner (O’Donnell, 2010; Korb and Nicholson, 2010), it also has a great potential to be extended, such as the CaMML extension on learning dynamic Bayesian networks (Pérez-Ariza et al., 2012), learning MB with CaMML (Li, 2020). To learn with latent nodes specifically, first we need to develop a method that can detect the existence of latent nodes and learn how to connect the latent nodes. We also need to compute the CPT of the latent nodes as well as the CPTs of any observed nodes who are their direct descendent in the learned structure. Finally, how to evaluate the model with latent variables and implement the discovering and learning processes into CaMML will be part of this thesis as well.

3.5 Evaluation Metrics

In this section, several popular methods that evaluate causal BN are presented. However, there is no agreed standard for evaluating causal discovery. Here we focus on two major approaches which are: how close is the learned structure to the true structure as well as how accurate the parameters are in the learned models. Due to the nature of the model with latent nodes, the number of nodes in the learned model could be different from the true model. For example, if there is a latent node in the true model, but the latent variable discovery function failed to detect the existence of the latent node, then there will not be any latent node appearing in the learned model (i.e., a false negative learned case). So we need to alter the learned model to put it in a form that can be used to compare with the true model. All details of how we evaluate causal model with latent nodes will be shown in this section.

3.5.1 Edit Distance

Edit distance is the most commonly used evaluation measure which informs how similar one structure is to another. To be more specific, the edit distance between two models equals the minimum number of arc deletions, additions and reversals to turn one model into the other (Korb and Nicholson, 2010). The distance is the same in either direction as it is a symmetrical metric. For example, the edit distance between the two structures in Figure

3.8 is 3, which includes an arc reversal between X and W , an arc deletion between H and Z , and an arc addition between H and Y (if we assume each of these operation contributes 1 to the total edit distance). Edit distance is simple and straightforward, however it is entirely indifferent to dependency strengths between variables, let alone interaction effects between multiple parents. It is the crudest of our measures, but nevertheless does provide some guidance on causal structure learning performance.

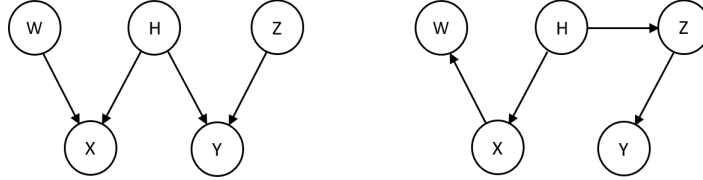


Figure 3.8: Example of how to calculate edit distance.

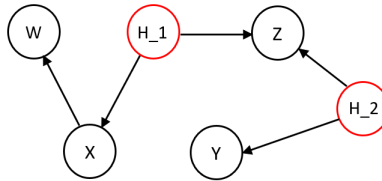


Figure 3.9: A structure with multiple latent nodes (“H_1” and “H_2”).

The latent node itself does not contribute to the edit distance score, however it will have implications in terms of how it is connected to other nodes in a model and we want to measure these implications independent of more distant arcs. For example, still in Figure 3.8, let node H be a latent node. We note the edit distance for arcs connected to the **latent node only** is 2, which includes an arc deletion between H and Z , and an arc addition between H and Y . So the edit distance for a specific node measures the difference between the arcs in the true structure and learned structure that relate to that node only.

Another point that should be mentioned here is that there could be multiple latent nodes discovered by PC and FCI, but the edit distance is calculated between two structures containing the same number of nodes. To solve these discrepancies, assuming there is only one latent node in the true structure, we select the latent node which shares the most arcs with the latent node in the true structure,⁵ and any arcs that are related to other proposed latent nodes will be treated as mistakes. For example, in the model in Figure 3.9, there are two proposed latent nodes H_1 and H_2 , but we assume the left model in Figure 3.8 is the true structure. Then H_1 will be selected as the best latent node as it shares more arcs with H in the true structure than H_2 . So the edit distance for the latent node only will be 2, and the total edit distance between the two models is 6. In contrast, to calculate edit distance when there is no latent node in the learned model, we add in the same number of latent nodes but leave them disconnected in the learned model in order to match variables with the true model. In addition, if there is no latent node in the true model but the learned model has some latent nodes, then any arc that relates to the latent nodes will be treated as a mistake in the learned model. As the edit distance measures structural differences, which can give us a direct assessment of a causal model when considering faithfulness as the assumption, it is a useful and intuitive metric that will be used throughout the thesis.

⁵Using a random tiebreaker, if needed.

3.5.2 KL and CKL Divergence

Kullback-Leibler Divergence (KL) (Kullback and Leibler, 1951; Kullback, 1959) is a measure of how far a learned probability distribution q diverges from the true distribution p :

$$KL(p, q) = \sum_i p_i \log \left(\frac{p_i}{q_i} \right) \quad (3.5)$$

where i ranges over all possible joint instantiations of nodes and by convention $0 \log 0 = 0$ and $x \log \frac{x}{0} = \infty$.

KL is not actually a distance metric, since it is not symmetric and violates the triangle property. Nevertheless, it is in some ways an ideal way to measure how far away a learned distribution is from the truth. In information-theoretic terms, KL can be described as the number of excess bits (using log base 2) that are expected to efficiently encode a new sample from the true distribution using the learned distribution. That is, information theory shows us that the optimal model for encoding samples is the true distribution itself; KL tells us how many additional bits are needed when we fail to use the true distribution. This means KL is an optimal way of expressing how bad our learned model is for predicting and encoding new samples (bad, because the larger KL is, the worse our encoded model is). What KL does not reflect, or measure, is anything specifically structural about the learned model. That is, any Markov equivalent model will yield precisely the same KL. More than that, any model which can represent the original probability distribution, including, for example, a fully connected model, will yield precisely the same KL (assuming maximum likelihood parameterization). In general, KL respects neither the simplicity of a model nor its causal semantics, and thus is suspect as a sole arbiter of causal discovery performance.

Unlike KL Divergence, Causal Kullback-Leibler divergence (CKL) (O’Donnell et al., 2007) takes into account causality when comparing two models, by using hypothetical interventions (Korb and Nyberg, 2006). Where causal orientations are misattributed, interventions reveal the differences and CKL is sensitive to them.

$$CKL(P_1, P_2) = \sum_{\vec{x}'} \sum_{\vec{x}} P'_1(\vec{x}', \vec{x}) \log \frac{P'_1(\vec{x}', \vec{x})}{P'_2(\vec{x}', \vec{x})} \quad (3.6)$$

where P_1 is the true distribution and P_2 is the learned distribution, while \vec{x}' and \vec{x} range over instantiations of intervention and original variables, and P'_1 and P'_2 extend P_1 and P_2 respectively, to the fully augmented space (that is there is an intervention node associated with each node from the original space, and such interventions can be viewed as observations).

There are three different variants of CKL described in (O’Donnell et al., 2007), each applying different types of intervention. We use CKL3 here as it finds an optimal augmentation for assessing causal models, where each node has an equal chance of not being intervened upon, and, when that happens, all other nodes are intervened upon. The aim is to perform an idealized experiment where all possible factors are controlled except for a target variable.

As KL and CKL require the learned and true model to have the same number of nodes, when the number of latent nodes is different in the learned and true model, we will use the same method as above to select the best latent node in calculating edit distance (see details in Section 3.5.1). If there are more latent nodes in the learned model than the true model, then any latent nodes except the best matching one will be marginalized out. So the unmatched latent node does not contribute to the KL or CKL computation in 3.5 and 3.6. Similarly, all latent nodes will be marginalized out in the learned model when there is no latent node in the true model. When the true model contains one or more

latent nodes and there is no latent node in the learned model, then we will add the same number of latent nodes (with the same number of states but uniform distribution) to the learned model. Moreover, if the learned network already has a disconnected node (e.g., as the SEM model might), we again attribute a uniform distribution to it. Another issue is that when measuring KL or CKL, the order of states matters, since target nodes are compared with each other state-by-state. If the states are learned correctly, but the order is not learned at all (which is the case for all latent model learners here), then misaligning states will hugely exaggerate the differences between models. Hence, we added a step for aligning state orders prior to computing the KL and CKL measures, which we applied to all learned latent models. In particular, we found the best state orders for any relevant variable (latent or child of a latent) by optimizing the KL/CKL score to be returned.

Chapter 4

Dependency Pattern Discovery

The same factors which enable latent variable discovery also enable causal discovery: particular probabilistic dependencies between variables, as estimated from available data, will typically be represented only by a proper subset of the possible causal models over those variables. Therefore, they provide evidence in favour of those models against the remaining models, as can be seen in the Bayes factor. As mentioned in Section 3.1, some dependency structures between observed variables will provide evidence favoring latent variable models over fully observed models, because they can explain the dependencies better than any fully observed model. Thus we can describe an algorithm for systematically searching for such dependencies. The result is a clutch of triggers. And, although they have been previously identified in the statistical literature (Richardson et al., 1999; Richardson and Spirtes, 2003), their use is new to the practical machine learning literature. In causal discovery algorithms these triggers can be used, for example, in preprocessing the data to abduce latent variable subnetworks for subsequent use by the main discovery algorithm.

So in this chapter, we first give the definition of a dependency matrix and explain how to use it to find triggers and subsequently use triggers to discover latent variables. We are keen to know how many triggers there are given different numbers of observed variables, so we introduce a systematic search and report the findings in Section 4.2. In Section 4.3, we present our initial attempt to apply the triggers as a data preprocessing step for the main causal discovery algorithms, which is named “Trigger-PC”. In Section 4.4, we test Trigger-PC against PC and FCI for how well they learn trigger structures using an optimized significance level. Finally, we propose an approach to applying triggers in CaMML. This includes a trigger detect function using dependency matrices, sampling space enhancement.

4.1 Definition of Dependency Matrix

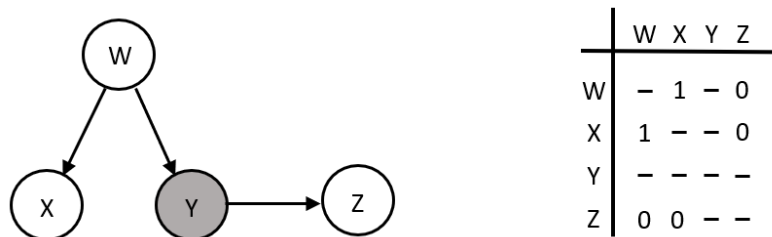


Figure 4.1: A DAG of four nodes and its dependency matrix (0 and 1 represent conditional independence and dependence respectively) conditioned on node Y .

Before we introduce our systematic search for triggers, we need to find a way to describe the conditional dependencies implied by a given structure under the faithfulness assumption. Thus we consider the dependency matrix, which represents the conditional dependencies between pairs of variables in a DAG, using “0” to represent that two corresponding variables are independent from each other given an evidence set, while “1” asserts conditional dependence. For example, in Figure 4.1, the 3×3 matrix on the right gives all dependencies (W and X) and independencies (W and Z , X and Z) in the DAG (on the left) conditioned upon Y . Clearly, dependency matrices are symmetric.

As dependency matrices are useful representations of conditional dependencies in a DAG, we can use them to search for triggers by looking for any unique dependency matrices (aka dependency patterns, Section 4.2). Dependency matrices also play an important role in the latent node detect function under faithfulness. So we can apply a conditional dependency test (i.e., a conditional χ^2 test) to find the conditional dependencies in the data and check whether they match any trigger. More details appear in Section 4.3.

4.2 A Systematic Search for Triggers

Here we describe a systematic algorithm for finding triggers given a certain number of observed variables. Latent variables are typically considered only in scenarios where they are common causes (Spirtes et al., 1993), i.e., having two or more children. As Friedman (Friedman et al., 1997) points out, a latent variable as a leaf or as a root with only one child would marginalise out without affecting the distribution over the remaining variables; hence, they could not participate in a trigger. The same is true of a latent variable that mediates only one parent and one child. Therefore, our trigger algorithm restricts itself to looking for latent common causes.¹ For simplicity, we also restrict trigger discovery to look for single latent variables rather than multiple latent variables. Again, subsequent general causal discovery can link together subnets suggested by multiple such triggers, as we explore in Chapter 6. Also, it will be clear from the trigger algorithm that it could readily be extended to look for two or three latent variables at a time, in case that should improve performance.

Algorithm 1 finds all possible (single-variable) triggers for a given number of observed variables.² It starts by enumerating all possible fully observed DAGs in n variables (where n is necessarily a small number, since this first step is already super exponential; see Robinson, 1977).³ Then it generates all possible d-separating evidence sets. In a network with n variables, there are $1 + \sum_{i=1}^{n-2} C_i^n$ such evidence sets, including the empty set. For example, for a network with four variables W , X , Y and Z , there are eleven evidence sets:

$$\emptyset, \{W\}, \{X\}, \{Y\}, \{Z\}, \{WX\}, \{WY\}, \{WZ\}, \{XY\}, \{XZ\}, \{YZ\}.$$

For each fully observed DAG, it produces the corresponding dependency matrix for each evidence set using the d-separation rules (e.g., for the four variables W , X , Y and Z , it produces eleven dependency matrices), and we call such a matrix set a “complete set of dependency matrices”. Next, it generates all possible single hidden-variable models for each DAG by replacing the arc between every pair of connected observed variables with a hidden common cause. It then generates the complete set of dependency matrices but

¹Note that these restrictions imply, for example, that we would not be finding any such latent variable model as that in Figure 1.1. However, these restrictions apply only to our initial search for useful triggers; subsequent search through the latent variable model space by a general causal learner can find these models, as Friedman’s work demonstrates (Friedman et al., 1997).

²Our Java Trigger program is available at: <https://github.com/zxh298/Trigger>

³We could have tried heuristic search to explore larger DAG spaces instead of a full enumeration, however, time did not permit us to explore that option.

between observed variables for each latent variable model, conditioned upon each evidence set (the latent node is not included in the evidence set). The set of dependencies of a latent variable model is a **trigger** if and only if these dependency sets cannot be matched by any fully observed DAG in terms of d-separation.

Algorithm 1 Find triggers for n observed variables

```

1: Let  $\mathcal{T}$  be an empty set; ( $\mathcal{T}$  will end up being all the triggers for  $n$  observed variables);
2: Generate all possible DAGs  $\overline{\mathcal{G}}$  (ignoring labels) for  $n$  observed variables;
3: Generate all evidence sets  $\overline{\mathcal{E}}$  for  $n$  observed variables;
4: Let  $\overline{\mathcal{M}}$  be an empty set;
5: for each  $\mathcal{G} \in \overline{\mathcal{G}}$  do
6:   Let  $\mathcal{M}$  be an empty set;
7:   for each  $\mathcal{E} \in \overline{\mathcal{E}}$  do
8:     Generate the dependency matrix  $m$  by applying d-separation;
9:     Add  $m$  to  $\mathcal{M}$ ;
10:  end for
11:  Add  $\mathcal{M}$  to  $\overline{\mathcal{M}}$ ;
12: end for
13: for each  $\mathcal{G} \in \overline{\mathcal{G}}$  do
14:   for each arc  $\mathcal{A}$  in  $\mathcal{G}$  do
15:     Replace  $\mathcal{A}$  with a latent variable as a common cause, yielding  $\mathcal{G}'$ ;
16:     Let  $\mathcal{M}'$  be an empty set;
17:     for each  $\mathcal{E} \in \overline{\mathcal{E}}$  do
18:       Generate the dependency matrix  $m'$  by applying d-separation;
19:       Add  $m'$  to  $\mathcal{M}'$ ;
20:     end for
21:     boolean  $isTrigger = true$ ;
22:     for each  $\mathcal{M} \in \overline{\mathcal{M}}$  do
23:       if  $\mathcal{M}'$  and  $\mathcal{M}$  contain exactly the same dependency matrices then
24:          $isTrigger = false$ ;
25:         break;
26:       end if
27:     end for
28:     if  $isTrigger = true$  then
29:       Add  $\mathcal{M}'$  to  $\mathcal{T}$ ;
30:     end if
31:   end for
32: end for
33: Output  $\mathcal{T}$ ;

```

To derive the time complexity of Algorithm 1, we break the whole process into several parts with line numbers (e.g., “ O_2 ” means the time complexity of Line 2). First, given n as the number of observed variables, generating all possible DAGs at Line 2 takes $O_2 = O(2^{\frac{n^2-n}{2}}) \approx O(2^{n^2})$ (Robinson, 1977). Then, the time complexity of generating all evidence sets at Line 3 is roughly $O_3 = O(2^n)$, as creating an evidence set requires one binary choice per $n - 2$ potential evidence nodes. Generating a dependency matrix given each \mathcal{E} at Line 8 requires $O_8 = O(\frac{n^2-n}{2})$ dependency tests. So the time complexity of the **for** loop from Line 5 to Line 12 is $O_5 \times O_7 \times O_8 = O_2 \times O_3 \times O_8 = O(2^{n^2}) \times O(2^n) \times O(\frac{n^2-n}{2}) \approx O(2^{n^2})$. Then at Line 15, replacing an arc with a latent variable takes a constant time complexity ($O_{15} = O(1)$), while the **for** loop from Line 17 to Line 20 has the same complexity as Line 7 to Line 10, $O_{17} = O(2^n) \times O(\frac{n^2-n}{2})$. Next, at Line 22, the size of $\overline{\mathcal{M}}$ equals to the size of $\overline{\mathcal{G}}$, and to make sure the trigger cannot be matched by any observed DAGs, we need to consider all possible label assignments, which takes an extra time complexity of $O(n!)$. At Line 23, the size of \mathcal{M} equals to the size of $\overline{\mathcal{E}}$, and checking whether two dependency matrices are the same takes $O(\frac{n^2-n}{2})$. So the time complexity from Line 22 to Line 27 is again $O_{22} = O(2^{n^2}) \times O(n!) \times O(2^n) \times O(\frac{n^2-n}{2})$. Together with the two **for** loops at Line 13 ($O_{13} = O(2^{n^2})$) and Line 14 ($O_{14} = O(\frac{n^2-n}{2})$), the total time complexity from Line

13 to Line 31 is $O_{13} \times O_{14} \times (O_{15} + O_{17} + O_{22}) = O(2^{n^2}) \times O(\frac{n^2-n}{2}) \times (O(1) + O(2^n) \times O(\frac{n^2-n}{2}) + O(2^{n^2}) \times O(n!) \times O(2^n) \times O(\frac{n^2-n}{2})) \approx O(2^{n^2})$. Overall, the time complexity of the algorithm will be:

$$\begin{aligned} O(\text{Algorithm1}) &= O_2 + O_3 + O_5 \times O_7 \times O_8 + O_{13} \times O_{14} \times (O_{15} + O_{17} + O_{22}) \\ &= O(2^n) + O(2^{n^2}) + O(2^{n^2}) + O(2^{n^2}) \\ &\approx O(2^{n^2}) \end{aligned}$$

So we can see that the time complexity is dominated by the exponential factor 2^{n^2} . In order to search for triggers more efficiently, we also developed a multi-threaded version of the trigger program⁴ named “Trigger-threads”, which runs Algorithm 1 by applying multiple threads to three aspects of the original Trigger algorithm:

- Generate $\bar{\mathcal{G}}$ for n observed variables.
- Generate $\bar{\mathcal{M}}$ for $\bar{\mathcal{G}}$.
- Trigger searching process (from Line 13 to Line 32).

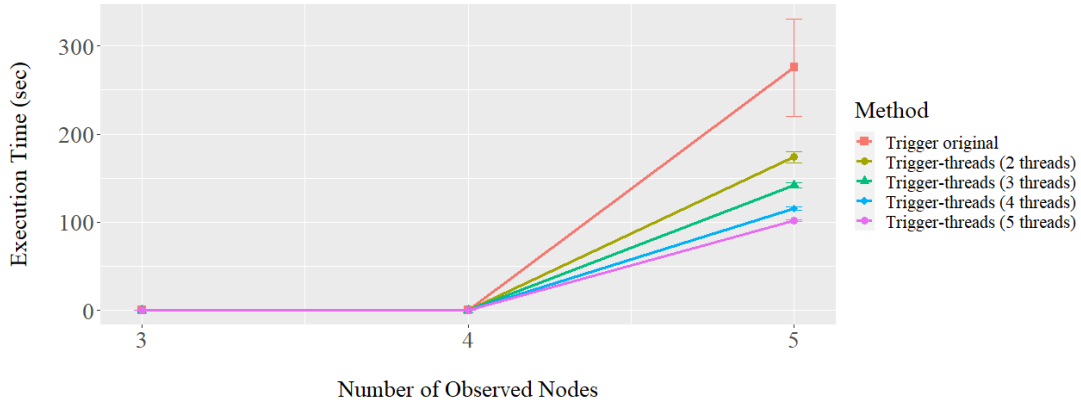


Figure 4.2: Execution time of Trigger original and Trigger-threads.

	3 nodes	4 nodes	5 nodes
Trigger original	0.004 (0.002, 0.007)	0.301 (0.272, 0.330)	275.661 (220.528, 330.793)
Trigger-threads (2 threads)	0.155 (0.150, 0.159)	0.368 (0.335, 0.400)	173.690 (167.082, 180.298)
Trigger-threads (3 threads)	0.155 (0.151, 0.159)	0.336 (0.302, 0.369)	141.942 (139.334, 144.551)
Trigger-threads (4 threads)	0.155 (0.150, 0.159)	0.307 (0.270, 0.344)	115.420 (113.747, 117.092)
Trigger-threads (5 threads)	0.105 (0.100, 0.109)	0.262 (0.225, 0.299)	101.770 (100.580, 102.959)

Table 4.1: Execution time of Trigger original and Trigger-threads.

Although this multi-threaded implementation won’t help with very large problems, it finishes much faster in practice with a small subnetworks. We tried running the original

⁴Available at: https://github.com/zxh298/Trigger_threads

Trigger and the Trigger-threads program with 2, 3, 4, and 5 threads 20 times each. Figure 4.2 and Table 4.1 show the average actual execution times (with 95% confidence intervals using the t-distribution).⁵ Searching for triggers with 3 observed nodes, the original program was a bit faster than the multi-threaded version because the actual trigger search process took less time than the data wrangling process (e.g., creating threads, combining results). However, when the number of observed nodes grows, the Trigger-threads program significantly outperforms the original. We did not try running the original program for 6 observed nodes as it became impractical with the super exponential step of generating all possible DAGs. Even for Trigger-threads, it took 1069559.1 seconds (around 12 days) for a run to finish searching triggers for 6 observed nodes (not shown).

Number of variables	Number of DAGs	Number of connected DAGs	Number of triggers
3	6	4	0
4	31	24	2
5	302	268	57
6	5984	5667	2525

Table 4.2: Number of DAGs and triggers.

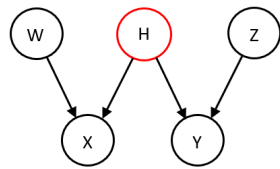
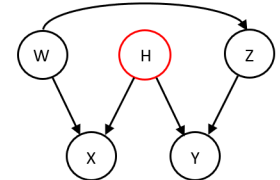
DAG (with one latent node H)	Trigger (Conditional dependencies)
	Given \emptyset : $W \perp\!\!\!\perp X, X \perp\!\!\!\perp Y, Y \perp\!\!\!\perp Z$ Given W: $X \perp\!\!\!\perp Y, Y \perp\!\!\!\perp Z$ Given X: $W \perp\!\!\!\perp Y, Y \perp\!\!\!\perp Z$ Given Y: $W \perp\!\!\!\perp X, X \perp\!\!\!\perp Z$ Given Z: $W \perp\!\!\!\perp X, X \perp\!\!\!\perp Y$ Given W and X: $Y \perp\!\!\!\perp Z$ Given W and Y: $X \perp\!\!\!\perp Z$ Given W and Z: $X \perp\!\!\!\perp Y$ Given X and Y: $W \perp\!\!\!\perp Z$ Given X and Z: $W \perp\!\!\!\perp Y$ Given Y and Z: $W \perp\!\!\!\perp X$
	Given \emptyset : $W \perp\!\!\!\perp X, W \perp\!\!\!\perp Y, W \perp\!\!\!\perp Z$ $X \perp\!\!\!\perp Y, X \perp\!\!\!\perp Z, Y \perp\!\!\!\perp Z$ Given W: $X \perp\!\!\!\perp Y, Y \perp\!\!\!\perp Z$ Given X: $W \perp\!\!\!\perp Z, W \perp\!\!\!\perp Y, Y \perp\!\!\!\perp Z$ Given Y: $W \perp\!\!\!\perp X, W \perp\!\!\!\perp Z, X \perp\!\!\!\perp Z$ Given Z: $W \perp\!\!\!\perp X, X \perp\!\!\!\perp Y$ Given W and X: $Y \perp\!\!\!\perp Z$ Given W and Y: $X \perp\!\!\!\perp Z$ Given W and Z: $X \perp\!\!\!\perp Y$ Given X and Y: $W \perp\!\!\!\perp Z$ Given X and Z: $W \perp\!\!\!\perp Y$ Given Y and Z: $W \perp\!\!\!\perp X$

Table 4.3: Triggers found (the Big-W and covered Big-W) for four observed variables (node “H” represents the latent).

⁵This experiment was done using a single Intel I9-9900KS CPU (with 16 cores) and 32GB 3600 MHz RAM.

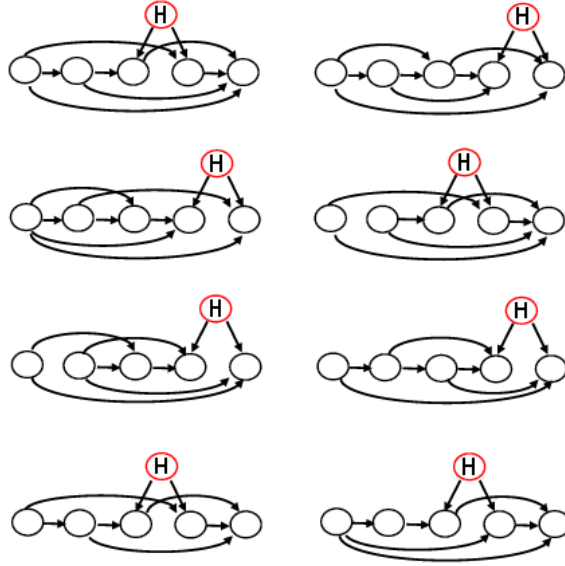


Figure 4.3: Some examples of triggers of five observed variables (node “H” represents the latent).

Finally, the number of distinct triggers, given the number of observed variables and ignoring labels and isolated nodes, is shown in Table 4.2. We use “Number of DAGs” to represent all possible DAGs for a certain number of (observed) variables, while “Number of connected DAGs” means all possible DAGs with no disconnected node. There are two triggers with four observed variables (where “H” represents the hidden variable), as shown in Table 4.3, which we named “Big-W” and “Covered Big-W” respectively. Figure 4.3 shows some example structures of the 57 possible triggers given five observed variables (see Appendix A for all 57 triggers).

Due to the complexity of learning DAGs with latent variables in statistical modeling, Richardson and Spirtes (2003) claimed that ancestral (mixed) graphs are more feasible representations to model conditional independence structures. Ancestral graphs represent latency without necessarily explicitly representing each individual latent variable, and if we assume there is only one latent node per one bi-directional arc, then we can turn the bi-directional arc into a latent node. A mixed graph (which allows both directed arcs and bi-directional arcs) is an ancestral graph if the following conditions are met (Richardson et al., 1999):

1. There is no directed cycle.
2. There is no almost-directed cycle; that is, if there is a bi-directional arc between two variables X and Y , $X \leftrightarrow Y$, then there is neither a directed path from X to Y , nor one from Y to X .
3. For any two variables X and Y which are connected by an undirected arc ($X—Y$) and any third variable Z , there is no arc $Z \rightarrow X$ or $Z \leftrightarrow X$.

A PAG (partial ancestral graph) then can be used as a representation of a class of Markov equivalent ancestral graphs. An ancestral graph is called a maximal ancestral graph (MAG) if there is no inducing path (see the definition in Chapter 3) between any two non-adjacent variables (no arc between them) in the graph (Zhang, 2008), and is maximal in the sense that no additional arc may be added to the graph without changing the independencies. PAG and MAG hold a one to many relationship, as PAG represents an equivalence class of MAG, by displaying all common edge marks shared by all members

and using circles for any marks that are not common. The latent variables in a MAG are not explicitly included, but are indicated by the presence of bi-directional arcs.

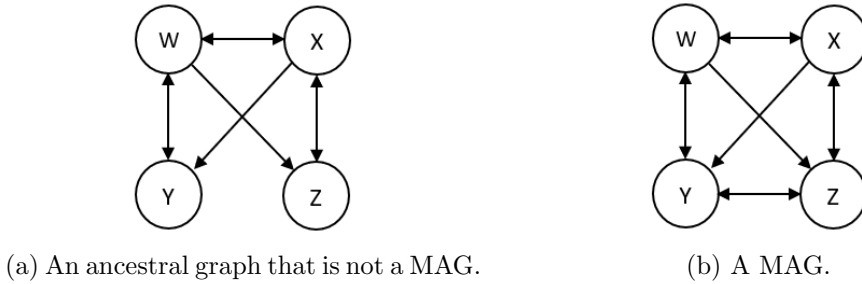


Figure 4.4: An example of MAG.

For example, in Figure 4.4a, the graph is ancestral but not maximal, as there is an inducing path between Y and Z because W is a collider that is a parent of Z , but Y and Z are not adjacent. However, a unique MAG corresponds to any non-MAG obtained by adding bi-directional arcs (Richardson and Spirtes, 2003). We can see that the MAG shown in Figure 4.4b has an additional bi-directional arc between Y and Z and is a unique MAG supergraph for the graph shown in Figure 4.4a.

The different types of graphs below follow a sequence of one-to-many relationships from left to right:

vertex-edge graph \rightarrow mixed graph \rightarrow PAG \rightarrow ancestral graph \rightarrow MAG \rightarrow DAG

Hence, DAGs are special cases of MAGs, lacking bi-directional arcs.

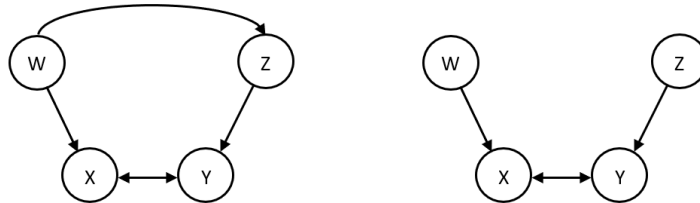


Figure 4.5: Two MAGs that are not Markov equivalent to any DAG which correspond to Big-W and Covered Big-W.

Spirtes et al. (1993) show that it is possible to characterize Markov equivalence classes of latent variable models in graphical terms. However, due to the complexity of learning DAGs with latent variables, searching for MAGs is a more tractable approach. Richardson et al. (1999) focus on how to parameterize, estimate and score MAGs containing no undirected arcs, allowing more than one latent node. Similar to our findings, this research shows that every MAG of three variables is Markov equivalent to some DAG, and there are six MAGs containing four variables that are not Markov equivalent to any DAG.⁶ For learning the best MAG given sample data, a heuristic greedy search for linear models was developed by Spirtes et al. (1997), which learns equivalence classes of MAGs (which will be grouped into a PAG as the output) with latent variables. This greedy search starts from a given MAG (e.g., an MAG with no arcs) and applies arc addition, removal or reversal to the current best MAG, if it improves the BIC score, and this process is repeated until convergence. A simulation study shows that it performs well when the sample size is large, the number of latent variables small, graphs sparse, and the data has no noise.⁷

⁶We were unaware of this work when the trigger program was developed.

⁷They do not offer evidence of a robust, practical learning method that works for large networks and smaller sample sizes, however.

As our Algorithm 1 only considers one latent variable, the problem becomes much simpler and it is the first practical attempt to search for triggers within DAGs to the best of our knowledge.

We can see the two triggers in Table 4.3 correspond to the two MAGs (shown in Figure 4.5) which are not Markov equivalent to any DAG when considering only one bi-directional arc (Richardson et al., 1999). As DAGs are special cases of MAGs, our search program is a reasonable alternative to the MAG search algorithms. Our suggestion for overcoming the fiercely exponential size of the latent DAG search space is to search in an initial phase only for latent-indicating dependency structures of small size (five observed variables and one latent), relying upon subsequent general causal discovery for learning more complex latent subnetworks with multiple latent variables. In other words, our aim is to embed trigger discovery in a practical general latent discovery pipeline.

So to conclude, all the dependency structures in the observed variables revealed as triggers by this algorithm, or a generalization incorporating multiple latent variables, will be better explained with latent variables than without. While it is not necessary to take triggers into account explicitly in latent variable discovery, since random structural mutations combined with standard metrics may well find them, they can be used to advantage in the discovery process, by focusing it, making it more efficient and more likely to find the right structure. In addition, as we noted above, a fully connected model can always be parameterized to fit those dependency patterns, but the price in complexity may well be too high. Smaller fully observed models will not fit the data exactly, and, whereas that may not matter if the sample size is small, given a larger number of observations, the more exactly fitting latent model must eventually become the better explanatory model. Ideally, we should like to modify causal discovery algorithms to be able to identify such cases and return the best explanatory models, rather than only the best observational model. Another important finding is that all the triggers of five and six observed variables have a subnet, which is the same as one of the two triggers with four observed variables (shown in Table 4.3). This finding will greatly simplify the search process and make the whole process more efficient, as we only need to save two triggers in advance and do further discovery using them. Since CaMML is already able to take prior knowledge of what experts believe the learned network should be, we can use such priors to bias CaMML to favor a trigger in its search process, and let CaMML decide how to connect it to the remaining network (see details in Chapter 5).

4.3 Learning Triggers With Causal Discovery Algorithms

Having developed our trigger-finder and applied it to small graphs, we were interested in putting it into practice and comparing it with existing programs. The existing programs which have systematically incorporated latent variable discovery from the beginning, and which are, in fact, the most popular programs for causal discovery in general, come from the Carnegie Mellon Philosophy group and are incorporated into TETRAD, namely PC and FCI (see details in Section 3.3). PC in particular has been re-implemented in numerous Bayesian network platforms. Hence, they are the natural foil against which to compare anything we might produce. Our ultimate goal, mentioned above, is to incorporate latent variable discovery into a metric-based program (see Chapter 5).

However, The PC and FCI algorithms do not generally return a single DAG, but a hybrid graph (see details in Section 3.3). An arc between two nodes in such a hybrid graph may be either undirected '—' or bi-directional '↔', which indicates the presence of a latent common cause. Additionally, the graph produced by FCI may contain 'o—o' or 'o→'. The circle represents an unknown relationship, which means it is not known whether an arrowhead occurs at that end of the arc (Spirtes et al., 1993). So, in order to

True arc	Learned arc	Edit Distance
$X \rightarrow Y$	$X \text{---} Y$	2
	$X \rightarrow Y$	0
	$X \leftarrow Y$	4
	$X \text{ o} \rightarrow Y$	1
	$X \leftarrow \text{o} Y$	3
	$X \leftrightarrow Y$	2
	$X \text{ o-o} Y$	2
	null	6
$X \leftrightarrow Y$	$X \text{---} Y$	4
	$X \rightarrow Y$	2
	$X \leftarrow Y$	2
	$X \text{ o} \rightarrow Y$	1
	$X \leftarrow \text{o} Y$	1
	$X \leftrightarrow Y$	0
	$X \text{ o-o} Y$	2
	null	6
null	$X \text{---} Y$	6
	$X \rightarrow Y$	6
	$X \leftarrow Y$	6
	$X \text{ o} \rightarrow Y$	6
	$X \leftarrow \text{o} Y$	6
	$X \leftrightarrow Y$	6
	$X \text{ o-o} Y$	6
	null	0

Table 4.4: Edit distance for PC and FCI output.

measure how close the models learned by PC and FCI are to the true model, we developed a special version of the edit distance between graphs (shown in Table 4.4), which contains the edit distances for how “close” each type of learned arc is to the true arc. The first two columns show the true arc (e.g., $X \rightarrow Y$) and learned arc (e.g., $X \leftarrow Y$) respectively and the third column shows the edit distance (i.e., 4) between them.

Our trigger discovery algorithm is intended to be used as a pre-processing phase in latent variable learning, however no existing latent variable discovery algorithm has attempted to use triggers in the discovery process. So, for a proper comparator, we decided to see how well a trigger filter as a front-end to PC would work, yielding Trigger-PC (shown as Algorithm 2). If Trigger-PC finds a trigger pattern in the data, then it returns that trigger structure, otherwise it returns whatever structure the PC algorithm returns, while replacing any incorrect bi-directional arcs with undirected arcs. As Trigger-PC is an extension of original PC, it also needs this special version of edit distance (see Table 4.4). It returns a more specific structure than PC or FCI and reduces the chance of returning a false positive latent node. To prove this, we have done some experiments to assess the performance of the Trigger-PC algorithm in comparison with PC and FCI, and report detailed results for all three algorithms below.

Algorithm 2 Trigger-PC Algorithm

```
1: Let  $\mathcal{D}$  be the test dataset;
2: Let  $\mathcal{D}_{labels}$  be the node labels in  $\mathcal{D}$ ;
3:  $|\mathcal{D}|$  be the number of variables in  $\mathcal{D}$ ;
4: Add one extra node label "H" (denotes a latent node) to  $\mathcal{D}_{labels}$ ;
5: Let  $Triggers$  be the unlabeled triggers given  $|\mathcal{D}|$ ;
6: Use conditional  $\chi^2$  tests to get the complete set of dependency matrices  $\mathcal{M}_{\mathcal{D}}$  in  $\mathcal{D}$ ;
7: Let  $matchTrigger = false$ ;
8: Let  $\mathcal{G}$  be an empty DAG;
9: for each trigger  $t$  in  $Triggers$  do
10:   Let  $\mathcal{L}$  be all possible label assignments for  $t$  using  $\mathcal{D}_{labels}$ ;
11:   for each  $l$  in  $\mathcal{L}$  do
12:     Assign  $l$  to  $t$ , yield  $t_{labeled}$ ;
13:     Generate the complete set of dependency matrices of  $t_{labeled}$  using d-separation, yield  $\mathcal{M}_t$ ;
14:     if  $\mathcal{M}_t$  matches  $\mathcal{M}_{\mathcal{D}}$  then
15:        $matchTrigger = true$ ;
16:        $\mathcal{G} = t_{labeled}$ ;
17:       break;
18:     end if
19:   end for
20:   if  $matchTrigger = true$  then
21:     break;
22:   end if
23: end for
24: if  $matchTrigger = true$  then
25:   Output  $\mathcal{G}$ ;
26: end if
27: if  $matchTrigger = false$  then
28:   Run PC Algorithm with input dataset  $\mathcal{D}$ ;
29:   Let  $\mathcal{G}_{pc}$  be the result structure produced by PC Algorithm;
30:   if there exist any bi-directed arcs in  $\mathcal{G}_{pc}$  then
31:     Replace all bi-directed arcs by undirected arcs, yield  $\mathcal{G}_{pc*}$ ;
32:     Output  $\mathcal{G}_{pc*}$ ;
33:   end if
34: end if
```

4.4 Experiment

This section reports our experimental design and results comparing Trigger-PC with PC and FCI. First, we describe how the test datasets were generated, being simulated from different artificial BNs with a variety of overall arc strengths. Then we present the experimental results assessing these algorithms on latent variable discovery. Our procedure, briefly, was:

1. Generate random networks of a given number of variables (both with and without latents), with three categories of dependency: weak, medium and strong.
2. Generate artificial data sets using these networks.
3. Optimize the significance level (alpha) of the PC and FCI programs using the above.
4. Experimentally test and compare Trigger-PC, PC and FCI, report results, and perform a comprehensive analysis.

4.4.1 Generating Simulated Datasets of Triggers

The datasets we used were generated from all fully observed as well as all (single) latent variable models with four or five observed variables, with both the observed and latent

variables having either two or three states.⁸ We wanted to test learning performance given a range of dependency strengths, from weak to medium to strong, so we used a genetic algorithm to find parameters reflecting these ranges, as determined by mutual information (I) which can be used to measure dependency between two nodes (Nicholson and Jitnah, 1998). As a non-negative and symmetric measure, $I(X, Y)$ reports the reduction in uncertainty of node X from learning node Y and is zero if and only if X and Y are mutually independent (Shannon, 1948; Pearl, 1988):

$$I(X, Y) = \sum_{x,y} p(X, Y) \log \frac{p(X, Y)}{p(X)p(Y)} \quad (4.1)$$

Since $p(X, Y) = p(X)p(Y|X)$, the equation can be rewritten as:

$$I(X, Y) = \sum_x \sum_y p(X)p(Y|X) \log \frac{p(Y|X)}{p(Y)} \quad (4.2)$$

The purpose was to find good representative, but random, graphs with the three levels of desired dependency strengths between variables, in order to test the learning algorithms across different degrees of difficulty in recovering arcs.

To make the learning process more efficient, we set the arities for all nodes in a network to be the same, either two or three. We randomly initialized all variables' CPT parameters for each individual graph and used a population of 100 individuals and ran the GA for 100 generations. We used each configuration (number of nodes and arities) three times, the first two to obtain networks with the strongest and weakest (overall) dependencies between parents and their children and the third time to obtain networks closest to the average of those two degrees of strengths.

Number of observed variables	Number of trigger structures	Total number of simulated datasets
4	2	36
5	57	1026

Table 4.5: Number of simulated datasets for trigger.

Number of observed variables	Number of DAG structures	Total number of simulated datasets
4	24	432
5	268	4824

Table 4.6: Number of simulated datasets for DAG structures (no latent variable).

Number of observed nodes	Contains a latent node	Number of datasets	Type
4	True	$2 \times 2 \times 3 \times 3 = 36$	Type 1
5	True	$57 \times 2 \times 3 \times 3 = 1026$	Type 2
4	False	$24 \times 2 \times 3 \times 3 = 432$	Type 3
5	False	$268 \times 2 \times 3 \times 3 = 4824$	Type 4

Table 4.7: The artificial datasets used for comparing Trigger-PC with PC and FCI.

For each of the three varieties of network, we generated artificial datasets of three different sample sizes using Netica (*Netica API*, 2012): 100, 1000 and 10000. Per Table 4.5, we produced datasets (Type 1 and 2) for every trigger structure of four and five observed variables. We did the same for all possible DAG structures (Type 3 and 4) without latent variables (ignoring isolated nodes; see Table 4.6), in order to check for false positives. The result was a set of about 6000 datasets for comparing our algorithms.

⁸The datasets are available at: <https://sourceforge.net/projects/triggers-of-bn-latent-variable/>

This large number of datasets resulted from: the differing number of states (either 2 or 3),⁹ arc strengths (low, medium and high) and sample sizes (100, 1000 and 10000). For example, there are 57 trigger structures, 2 arities (two or three states), 3 arc strengths (high, medium and low), 3 sample sizes (100, 1000, 10000) which give $57 \times 2 \times 3 \times 3 = 1026$ simulated datasets (details are shown in Table 4.7).

4.4.2 Experimental Results

As mentioned in the experimental procedure, we optimized the significance level alpha for PC, FCI and Trigger-PC for the datasets we simulated.¹⁰ They have a default alpha level (0.05), but the authors have in the past recommended optimizing the alpha level for the task at hand, so here we did that. Our idea was to give the performance of the three algorithms the benefit of any possible doubt. Given our artificial datasets, we optimized the performance of each algorithm in terms of our version of edit distance between the learned and generating models (see Table 4.4). The datasets were divided into a 50-50 split, that is half for finding the best alpha and the other half for testing.

Data type	FCI	PC	Trigger-PC
Type 1	0.413	0.402	0.443
Type 2	0.16	0.131	0.138
Type 3	0.129	0.114	0.154
Type 4	0.129	0.126	0.141

Table 4.8: The optimized alpha of each algorithm.

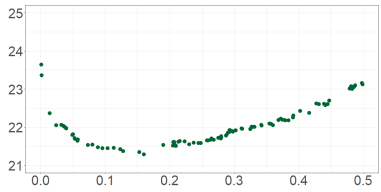
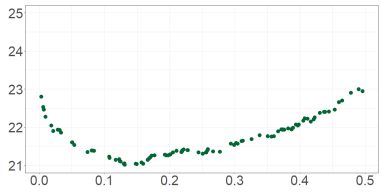
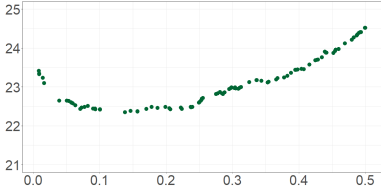
Edit distance	Result details
	Algorithm: FCI Best alpha: 0.16 Minimum edit distance: 21.282 Maximum edit distance: 23.638
	Algorithm: PC Best alpha: 0.131 Minimum edit distance: 21.011 Maximum edit distance: 22.996
	Algorithm: Trigger-PC Best alpha: 0.138 Minimum edit distance: 22.344 Maximum edit distance: 24.517

Table 4.9: Alpha optimization of Trigger cases (using Type 2 datasets).

⁹The arity of every node is the same, either 2 or 3, in each test network.

¹⁰The implementations of PC and FCI used in this experiment are from Tetrad (version 5.3.0) (Spirtes et al., 2016).

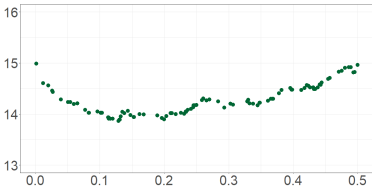
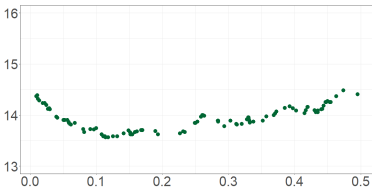
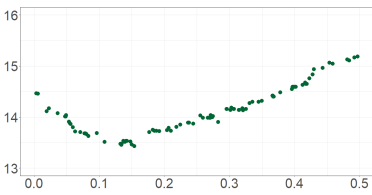
Edit distance	Result details
	Algorithm: FCI Best alpha: 0.114 Minimum edit distance: 13.365 Maximum edit distance: 14.481
	Algorithm: PC Best alpha: 0.129 Minimum edit distance: 13.861 Maximum edit distance: 14.986
	Algorithm: Trigger-PC Best alpha: 0.154 Minimum edit distance: 13.426 Maximum edit distance: 15.185

Table 4.10: Alpha optimization of DAG cases (using Type 3 datasets).

We generated 100 random samples from a uniform distribution ranging from 0.0 to 0.5 to find the best alpha values, which were evaluated against each type of dataset in Table 4.7. We can see that the results (see Table 4.8) for the three algorithms are broadly similar, with most optimal values are in the range of (0.10, 0.16), except the values for Type 1. It may be due to the lack of test datasets compared to other types (there were only $36/2 = 18$ test datasets).

As an example, the edit distance results for each tested alpha of Type 2 datasets are shown as in Table 4.9, where the x axis and y axis of the graphs shown in the table represent values of tested alpha and values of average edit distance respectively, and the green points represent all sampled values. We can see the ED lines of the three algorithms are similar, and the lines are also similar in the results using Type 3 datasets shown in Table 4.10. It is obvious that the default value 0.05 was not the best value, so this optimization step was useful.

Finally, we were ready to test the three algorithms on another half of the datasets. Datasets Type 1 and 2 were used to determine True Positive (TP) and False Negative (FN) results (i.e., finding the real latent and missing the real latent, respectively), while the Type 3 and 4 were used for False Positive (FP) and True Negative (TN) results. Assume the latent variable in every trigger structure is the parent of node X and Y , we used the following definitions:

- TP: The learned model has a bi-directional arc between X and Y .
- FN: The learned model lacks a bi-directional arc between X and Y .
- TN: The learned model has no bi-directional arcs.
- FP: The learned model has one or more bi-directional arcs.

We tested the three algorithms on different datasets against different metrics¹¹ with their corresponding optimized alphas and the default alpha 0.05. For the average ED results, the mean values along with 95% confidence intervals (in brackets) using the t-distribution are shown in the following tables, and we display the results with 95% binomial proportion confidence intervals (in brackets) for other metrics. In order to make the results more readable, bold text is used to indicate significantly different best result while italic text denotes significantly different worst result of each row.¹²

	FCI	PC	Trigger-PC
Average ED	20.708 (20.299, 21.118)	20.005 (19.566, 20.444)	20.121 (19.691, 20.551)
Accuracy	0.731 (0.716, 0.746)	0.754 (0.739, 0.769)	0.838 (0.825, 0.851)
FPR	<i>0.166</i> (0.153, 0.179)	0.137 (0.125, 0.149)	0.008 (0.005, 0.011)
FNR	0.791 (0.777, 0.805)	0.797 (0.783, 0.811)	<i>0.935</i> (0.926, 0.944)
Recall	0.209 (0.195, 0.223)	0.203 (0.189, 0.217)	<i>0.065</i> (0.056, 0.074)
Precision	0.200 (0.186, 0.214)	0.227 (0.212, 0.242)	0.618 (0.601, 0.635)
Specificity	<i>0.834</i> (0.821, 0.847)	0.863 (0.851, 0.875)	0.992 (0.989, 0.995)
F-score	0.204 (0.190, 0.218)	0.214 (0.200, 0.228)	<i>0.118</i> (0.107, 0.129)

Table 4.11: Results using optimized alpha. See Table 4.13 for more detailed results of different arc strengths (bold and italic represent significantly the best and worst results).

	FCI	PC	Trigger-PC
Average ED	21.106 (20.687, 21.525)	20.523 (20.076, 20.971)	20.592 (20.152, 21.032)
Accuracy	0.752 (0.737, 0.767)	0.772 (0.757, 0.787)	0.841 (0.828, 0.854)
FPR	<i>0.137</i> (0.125, 0.149)	0.113 (0.102, 0.124)	0.006 (0.003, 0.009)
FNR	0.808 (0.794, 0.822)	0.808 (0.794, 0.822)	<i>0.927</i> (0.918, 0.936)
Recall	0.192 (0.178, 0.206)	0.192 (0.178, 0.206)	<i>0.073</i> (0.064, 0.082)
Precision	<i>0.217</i> (0.203, 0.231)	0.252 (0.237, 0.267)	0.691 (0.675, 0.707)
Specificity	<i>0.863</i> (0.851, 0.875)	0.887 (0.876, 0.898)	0.994 (0.991, 0.997)
F-score	0.204 (0.190, 0.218)	0.218 (0.204, 0.232)	<i>0.132</i> (0.120, 0.144)

Table 4.12: Results using default alpha 0.05. See Table 4.14 for more detailed results of different arc strengths (bold and italic represent significantly the best and worst results).

With the optimized alpha, the overall performance (shown in Table 4.11) of PC and FCI are quite similar. Neither are finding the majority of latent variables actually there, but both are at least showing moderate FPR. Arguably, false positives are a worse offence than false negatives, since false negatives leave the causal discovery process no worse off than an algorithm that ignores latents, whereas a false positive will positively mislead the causal discovery process. Some indication of this can be seen in Section 5.2 where SEM’s false positive rate leads to an overall worse performance in most cases. To be sure, the relative disvalue of false negatives and false positives is domain dependent, but this

¹¹FPR and FNR represent False Positive Rate and False Negative Rate respectively, and we set β to 1.0 for all the F-score results in this chapter.

¹²We used an approximate (overly conservative) method for determining significant differences between two groups by checking whether their confidence intervals overlap.

difference at least suggests that latent discovery trading off fewer false positives for more false negatives will not be a bad thing in general. A more detailed investigation of this, in either specific contexts or across contexts is beyond the scope of this dissertation, however.

As we expected, Trigger-PC has a much better performance with respect to the FPR because of its pre-processing phase, but also produced more FN results. Compared to PC and FCI, the overall performance of Trigger-PC looks more conservative in terms of returning a positive result. Table 4.12 shows similar results with the default alpha 0.05; the differences across tables are not significant.

	Maximum arc strength			Medium arc strength			Minimum arc strength		
	FCI	PC	Trigger-PC	FCI	PC	Trigger-PC	FCI	PC	Trigger-PC
Average ED	<i>15.490</i>	14.400	14.400	15.854	14.878	15.366	30.781	30.737	30.596
Accuracy	0.768	0.791	0.843	0.703	0.738	0.839	0.722	0.731	0.833
FPR	0.139	0.110	0.006	0.215	0.171	0.017	0.143	0.130	0.001
FNR	0.701	0.707	<i>0.920</i>	0.713	0.718	<i>0.885</i>	0.960	0.966	<i>1.000</i>
Recall	0.299	0.293	<i>0.080</i>	0.287	0.282	<i>0.115</i>	0.040	0.034	<i>0.000</i>
Precision	0.299	0.347	0.737	0.210	0.246	0.571	0.053	0.050	<i>0.000</i>
Specificity	0.861	0.890	0.994	0.785	0.829	0.983	0.857	0.870	0.999
F-score	0.299	0.318	<i>0.144</i>	0.243	0.263	<i>0.191</i>	0.046	0.040	-

Table 4.13: Results of networks with different arc strengths using optimized alpha (bold and italic represent significantly the best and worst results).

	Maximum arc strength			Medium arc strength			Minimum arc strength		
	FCI	PC	Trigger-PC	FCI	PC	Trigger-PC	FCI	PC	Trigger-PC
Average ED	15.643	14.775	14.844	16.044	15.194	15.362	31.632	31.600	31.570
Accuracy	0.765	0.794	0.848	0.739	0.766	0.842	0.751	0.755	0.833
FPR	0.135	0.099	0.002	0.176	0.144	0.016	0.100	0.096	0.001
FNR	0.741	0.741	<i>0.908</i>	0.690	0.690	<i>0.874</i>	0.994	0.994	<i>1.000</i>
Recall	0.259	0.259	<i>0.092</i>	0.310	0.310	<i>0.126</i>	0.006	0.006	<i>0.000</i>
Precision	<i>0.276</i>	0.341	0.889	0.260	0.300	0.611	0.011	0.012	<i>0.000</i>
Specificity	0.865	0.901	0.998	0.824	0.856	0.984	0.900	0.904	0.999
F-score	0.267	0.294	<i>0.167</i>	0.283	0.305	<i>0.209</i>	0.008	0.008	-

Table 4.14: Results of networks with different arc strengths using default alpha 0.05 (bold and italic represent significantly the best and worst results).

In order to better understand how the arc strength impacts these numbers, we show the metrics for all arc strengths in Table 4.13 and 4.14) using the optimized alpha and default alpha respectively (see Appendix B for the confidence intervals). As is to be expected, the performance of all the algorithms degrades in general as the arc strength decreases. We can see that Trigger-PC is finding far fewer latents than either PC or FCI for different arc strengths, but when it asserts their existence, we can have much greater confidence in the claim. Again, the results using the default alpha are only very slightly different. As we indicated above, avoiding false positives, while having at least some true positives, appears to be the more important goal in latent variable discovery. This suggests that a subsequent general causal discovery program using Trigger as a preprocessing step can apply the triggers with enhanced confidence as priors.

4.5 Summary

We have presented the first algorithm to search for and report latent variable triggers from sample data: conditional probability structures that are better explained by latent variable models than by any DAG constructed from the observed variables alone. For simplicity and efficiency, we have limited this to looking for single latent variables at a time, although that restriction can be removed with the consequence that the time complexity will grow even higher. In our trigger discovery experiment, FCI and PC perform quite similarly.

They can both find latent variables when they are there, but with a weak recall rate of around 20%. They also mostly avoid misidentifying ordinary models as latent variable models, but still have FPR of around 11% to 16%. We have also applied the trigger discovery algorithm directly in PC, yielding Trigger-PC, and compared the results to PC and FCI. While on edit distance and F-score, overall accuracy measures, Trigger-PC does a little worse than PC and FCI, one attribute is notable for applying the Trigger algorithm to general causal discovery: it has a significantly lower FPR. This augurs well for the use of triggers in a more robust causal discovery program, since the next stage will not be misled by false positives and can treat any positive latents found as more likely to be actually there than anything nominated by PC or FCI.

The ideal evaluation of the performance across the confusion matrix would be to use a cost/reward matrix to measure the expected value of each different kind of correct and incorrect identification. However, since we are not dealing with a specific application, but a general method, there is no such cost matrix. Still, the considerations above show it is plausible that the worst outcome is a false positive. If latent discovery is meant to somehow supplement or extend existing causal discovery methods (as we intend), then the worst outcome is to positively mislead causal discovery to search in areas of a model space that introduce non-existent structure. By contrast, failing to prompt such a search when it is warranted (false negatives) will not degrade existing discovery methods. In any case, we consider these preliminary results for trigger discovery to be quite promising.

Our next step will be to implement the trigger discovery as a pre-processing phase to CaMML. That is when a trigger is found, a latent variable will be created and added to CaMML's sampling process, whilst making the current MML score compatible with latent variables due to the lack of data. Of course, reparameterization will be needed during the MCMC sampling whenever the current working structure changes, and this is what we will look at next chapter.

Chapter 5

EM-CaMML for Latent Variable Models

Causal Bayesian networks allow people to model systems under uncertainty in a humanly intuitive fashion. Causal discovery supports the automated development of such models from sample data, bypassing expert elicitation and its “knowledge bottleneck”. While sample data is available for many problems, often some variables cannot be directly measured, or perhaps people are unaware of some relevant variables. Somewhat surprisingly, there is not a great deal of research specifically on discovering latent variable causal models, even though latent variables can often significantly reduce overall model complexity. For some specific causal structures with latents, the conditional dependencies they represent can only be satisfied by finding the latent node, or by introducing spurious causal arcs between observed variables. Here we introduce a new end-to-end procedure, called EM-CaMML, which explicitly detects the existence of latent variables without requiring any prior knowledge and which returns a fully parameterized causal Bayesian net. To the best of our knowledge, this is the first practical program that performs metric-based latent causal model discovery explicitly, by directly assessing the value of latent submodels. Finally, we demonstrate experimentally the performance and effectiveness of our method.

The beginning of this chapter shows an overview of how EM-CaMML is designed by explaining the high level workflow, followed by a new extension of the MML metric for scoring causal models with latent variables, presented in Section 5.1.1. Then we demonstrate how the EM algorithm is integrated into EM-CaMML, with a comparison between two approaches, EM inside and outside, in Section 5.1.2. Then a complete EM-CaMML workflow is shown in Section 5.1.3. In Section 5.2, we include the experiments which compare EM-CaMML with several well-known latent variable learners on four types of test networks. Finally, the software development work of EM-CaMML is presented in Section 5.3.

5.1 EM-CaMML Algorithm Design

There has been great interest in learning causal BNs from data, as BNs are uniquely well suited to representing causal structures, given causal relations can be directly captured by the structure of the network (Korb and Nicholson, 2010). Causal discovery aims to construct a BN which has the most probable causal relationships given the joint probability distribution represented by the data. Since this is a difficult task, most methods allow the use of at least some prior knowledge from a domain expert in the form of either hard or soft (i.e., probabilistic) constraints.

The constraint learners PC and FCI are both capable of discovering latent, as well as fully observed, causal structures (see details in Section 3.3). FCI was designed explicitly

for discovering latents; PC was not, however it returns undirected arcs which are naturally interpreted as agnosticism between the two possible directions, along with the possibility of a hidden common cause (Spirtes et al., 1993). Most metric-based learners, on the other hand, simply do not accommodate latent variable discovery. However, when they do, they require the user to specify where possible latent variables might show up. For example, the best known such algorithm, SEM, requires an initial structure be specified (see details in Section 3.4.2), linking in any latents as common causes to a subset of observed variables (Friedman et al., 1997). Then SEM will learn the parameters of the model, using expectation maximization, while finding the structure (adding or dropping arcs) which best suits the data, by minimizing an MDL score (Friedman et al., 1997; Friedman and Goldszmidt, 1998). We can then say SEM has detected a latent node, if the latent volunteered by the user is connected in SEM’s final model. Causal discovery with humans positing in advance what the model space might look like, whether it has latent variables and if so, how many, is not nearly as useful as a fully automated process, which is what we are aiming at.

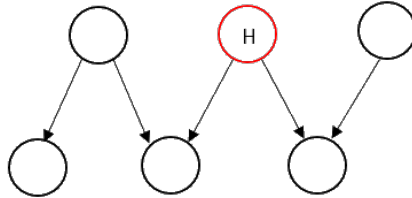


Figure 5.1: A triggering submodel with 6 observed nodes containing Big-W (node “H” represents the latent).

As reported above, we have a trigger program and report all trigger models with $\leq k$ observed variables, finding that all triggers with 5 and 6 observed nodes have either Big-W or Covered Big-W as a subnet (see Section 4.2).¹ For example, we found the 6-node graph of Figure 5.1 which incorporates Big-W. We used this to significantly speed up the search for triggers, considering that there are already 2525 different triggers for 6-node structures, and the number of triggers grows exponentially from there. Another practical decision we made early was to look only for single latent-variable triggers, as multiple hidden-node triggers would be much harder to search for. Since we approach latent detection as a preprocessing step to general causal discovery, there remains the opportunity to stitch multiple latents together in a more complex causal structures later on, as described in Chapter 6.

For the preprocessing step (discovering triggers), we search through partitions of the dataset representing sets of four observed variables, performing χ^2 tests to determine whether the conditional dependencies estimated from the data match the dependencies expected for a trigger, either Big-W or Covered Big-W.² When they are found, latent variables are added to the search space for subsequent causal discovery. The arcs indicated by a latent trigger structure are, in particular, added as soft (probabilistic) constraints, allowing the EM-CaMML to add or remove them, given a strong enough signal from the data.

As mentioned in Section 3.4.3, CaMML is metric-based algorithm which uses MCMC search (Korb and Nicholson, 2010; O’Donnell, 2010) for finding the best causal model given an observational dataset, as measured by MML. It samples from a posterior distribution

¹Due to time restrictions, we have not produced triggers with 7 nodes, but have only found 7-observed node triggers having either Big-W or Covered Big-W as subnets.

²The significance level (alpha) we used for trigger detection is 0.05, which is also the value we used for PC and FCI in our experiments.

over the causal model space, represented by totally ordered models (or TOMs, which are linear extensions of DAGs). EM-CaMML extends CaMML by conducting its sampling over a model space extended by one or more latent variables, assuming the Trigger pre-processing step has added them. EM-CaMML gives a CPT to the latent node, using random parameters that represent a strength of dependency between it and its children, as measured by mutual information. Such strength was optimized for recovering latent structures in terms of F-score, prior to the main experiments reported here (See Section 5.2.3 for details of that optimization study). EM is used to re-estimate parameters for any latent node and its children as an internal loop during the larger MCMC sampling process. The MCMC sampling then uses the MML scores of the current model and its neighbors to find the next sampled model from the neighboring candidates, in the same manner as the original CaMML (Korb and Nicholson, 2010), except that the MML score is adjusted to reflect the presence of the latent variables, as described in Section 5.1.1. The next model found will be used to update the structure for rerunning EM to re-estimate parameters associated with latent variables. These cycles continue until a specified number of iterations have taken place, when the user can see the best MML model or a set of the top MML models, together with their equivalence classes.

There were several difficulties we overcame in terms of EM-CaMML implementation and experiment work. First, how to make a practical latent discovery and learning program which can automatically finish the end to end process without human intervention is a challenging job. This required significant time learning CaMML and its CDMS platform (Comley et al., 2003; Allison, 2005), and adapting the latent node detection and learning modules into the sizeable CaMML project in a development and testing loop. Also, different learners have their own theoretical basis and there is no common definition of a “positive result”, so how to process the results from each benchmark learner to make a fair comparison was certainly another challenge.

5.1.1 A New Score Metric for Latent Nodes

Inspired by how the EM algorithm (Dempster et al., 1977) is used in Friedman’s SEM algorithm (Friedman, 1998; Friedman et al., 1997), we applied EM for computing parameters associated with latent nodes. The EM algorithm starts with a given model structure and initial random model parameters, then runs the Expectation step (E_step) and the Maximization step (M_step) in a loop until convergence. The E_step determines the probability (expected count) of each possible value of the latent variable; the M_step then uses such expected counts to re-estimate the model parameters (see details in Section 3.4.1). However, such expected counts have no place in the existing MML metric. Instead, we use an extension of the current MML approximation (adaptive code) (O’Donnell, 2010; Wallace, 2005) implemented in CaMML. The MML adaptive code is the sum of the structure cost (see Equation 3.1) and the message length of each node’s encoded CPT (see Equation 3.2). For latent nodes, the structure cost equation needs no changes, other than increasing the “k” by the number of such nodes. In dealing with latent nodes we have no data, and so the counts referred to in Equation 3.2 do not exist. However, we can use the soft counts generated by EM for $(N(pa_i) + |X| - 1)$ and $N(pa_i, x_i)$ as real-valued estimates for latent nodes. To be specific, we take each $N(pa_i)$ that refers to a latent parent and split it into j versions of $N(pa_i, latent_j)$ (for every j th state of the latent), and multiply each by EM’s expected proportions for $pa_i + latent_j$ (and do similarly for $N(pa_i, x_i)$). Since EM produces soft counts that can contain fractional numbers rather than discrete counts, so we use the Gamma function to replace the factorial. This yields a “latent adaptive code”:³

³Note that this metric works for any fully observed submodel.

$$CPTCost^* = \frac{|pa| \times (|X| - 1)}{2} \times \ln \frac{\pi e}{6} + \sum_{pa_i}^{|pa|} \ln \left(\frac{\Gamma(N(pa_i) + |X|)}{(|X| - 1)! \times \prod_{x_i}^{|X|} \Gamma(N(pa_i, x_i) + 1)} \right) \quad (5.1)$$

Weights	H	W	X	Y	Z
0.2	T	T	F	T	T
0.8	F				
0.7	T	F	T	T	T
0.3	F				
0.9	T	T	T	T	T
0.1	F				
0.5	T	T	F	T	F
0.5	F				
0.6	T	F	F	T	T
0.4	F				
0.9	T	T	T	T	T
0.1	F				
0.2	T	T	F	T	T
0.8	F				

Figure 5.2: Example of Expected Counts.

We take as an example the Big-W structure shown in Figure 3.1 which contains four observed nodes W, X, Y, Z and a latent node H , and assume each node (including the latent node) is binary with states T and F , and has sample data the same as the discrete data shown in Figure 5.2. There are seven rows of observed data over nodes W, X, Y, Z and the weights (expectations or soft counts) over missing values (per each H instantiation) are shown as in the first column. So in this case, the CPT probabilities (based on the current weights) of node X will be:

$$\begin{aligned} P(X = T|H = T, W = T) &= E(X = T, H = T, W = T)/E(H = T, W = T) \\ &= (0.9 + 0.9)/(0.2 + 0.9 + 0.5 + 0.9 + 0.2) \approx 0.667 \\ P(X = F|H = T, W = T) &= E(X = F, H = T, W = T)/E(H = T, W = T) \\ &= (0.2 + 0.5 + 0.2)/(0.2 + 0.9 + 0.5 + 0.9 + 0.2) \approx 0.333 \end{aligned}$$

Where function “E” represents the expected counts of corresponding node instantiations. Similarly, the other six probability values in the CPT of node X are:

$$\begin{aligned} P(X = T|H = T, W = F) &= 0.7/(0.7 + 0.6) \approx 0.538 \\ P(X = F|H = T, W = F) &= 0.6/(0.7 + 0.6) \approx 0.462 \\ P(X = T|H = F, W = T) &= (0.1 + 0.1)/(0.8 + 0.1 + 0.5 + 0.1 + 0.8) \approx 0.087 \\ P(X = F|H = F, W = T) &= (0.8 + 0.5 + 0.8)/(0.8 + 0.1 + 0.5 + 0.1 + 0.8) \approx 0.913 \\ P(X = T|H = F, W = F) &= 0.3/(0.3 + 0.4) \approx 0.429 \\ P(X = F|H = F, W = F) &= 0.4/(0.3 + 0.4) \approx 0.571 \end{aligned}$$

For node H and W , given they have no parents, then their CPT probabilities will be:

$$\begin{aligned}
P(H = T) &= E(H = T)/(E(H = T) + E(H = F)) \\
&= (0.2 + 0.7 + 0.9 + 0.5 + 0.6 + 0.9 + 0.2)/(0.2 + 0.8 + 0.7 + 0.3 + 0.9 + 0.1 \\
&\quad + 0.5 + 0.5 + 0.6 + 0.4 + 0.9 + 0.1 + 0.2 + 0.8) \approx 0.571 \\
P(H = F) &= 1.0 - P(H = T) \approx 0.429 \\
P(W = T) &= E(W = T)/(E(W = T) + E(W = F)) \\
&= (0.2 + 0.8 + 0.9 + 0.1 + 0.5 + 0.5 + 0.9 + 0.1 + 0.2 + 0.8)/(0.2 + 0.8 + 0.7 \\
&\quad + 0.3 + 0.9 + 0.1 + 0.5 + 0.5 + 0.6 + 0.4 + 0.9 + 0.1 + 0.2 + 0.8) \approx 0.714 \\
P(W = F) &= 1.0 - P(W = T) \approx 0.286
\end{aligned}$$

Thus for node X , as it has four parent state combinations: $\{H = T, W = T\}, \{H = F, W = T\}, \{H = T, W = F\}, \{H = F, W = F\}$, we can calculate its CPT cost as:

$$\begin{aligned}
CPT_cost_X &= \frac{4 \times (2 - 1)}{2} \times \ln\left(\frac{3.142 \times 2.718}{6}\right) \\
&\quad + \ln\left(\frac{\Gamma(N(H = T, W = T) + 2)}{(2 - 1)! \times \Gamma(N((H = T, W = T, X = T)) + 1) \times \Gamma(N((H = T, W = T, X = F)) + 1)}\right) \\
&\quad + \ln\left(\frac{\Gamma(N(H = T, W = F) + 2)}{(2 - 1)! \times \Gamma(N((H = T, W = F, X = T)) + 1) \times \Gamma(N((H = T, W = F, X = F)) + 1)}\right) \\
&\quad + \ln\left(\frac{\Gamma(N(H = F, W = T) + 2)}{(2 - 1)! \times \Gamma(N((H = F, W = T, X = T)) + 1) \times \Gamma(N((H = F, W = T, X = F)) + 1)}\right) \\
&\quad + \ln\left(\frac{\Gamma(N(H = F, W = F) + 2)}{(2 - 1)! \times \Gamma(N((H = F, W = F, X = T)) + 1) \times \Gamma(N((H = F, W = F, X = F)) + 1)}\right) \\
&\approx 0.706 + \ln\left(\frac{\Gamma((0.2 + 0.9 + 0.5 + 0.9 + 0.2) + 2)}{\Gamma((0.9 + 0.9) + 1) \times \Gamma((0.2 + 0.5 + 0.2) + 1)}\right) + \ln\left(\frac{\Gamma((0.7 + 0.6) + 2)}{\Gamma(0.7 + 1) \times \Gamma(0.6 + 1)}\right) \\
&\quad + \ln\left(\frac{\Gamma((0.8 + 0.1 + 0.5 + 0.1 + 0.8) + 2)}{\Gamma((0.1 + 0.1) + 1) \times \Gamma((0.8 + 0.5 + 0.8) + 1)}\right) + \ln\left(\frac{\Gamma((0.3 + 0.4) + 2)}{\Gamma(0.3 + 1) \times \Gamma(0.4 + 1)}\right) \\
&\approx 6.302
\end{aligned}$$

This new version of MML is applied in our EM-CaMML, as the metric to apply to all sampled TOMs. In the actual implementation, the MML cost of one node in the same local subnet will not be calculated twice. In other words, if the dataset does not change then the MML score of a node X will always be the same unless the parents of X have changed. So the MML score of each distinct subnet structure per node will be cached, and when evaluating a new TOM, the MML score of each node will not be calculated again unless the cache does not contain the corresponding information. Note that the cache is maintained within each EM iteration, and whilst we could extend it across EM iterations, the current performance is already acceptable.

5.1.2 EM Algorithm Integration

The Metropolis-Hasting sampling used by EM-CaMML (as in CaMML), produces an estimated posterior distribution of the model space, which can be used to find the highest posterior probability equivalence class, model or region in the model space. The original version of CaMML assumes complete data, hence parameterization is straightforward. After each candidate TOM is generated, maximum likelihood estimation is used to parameterize the network. While EM-CaMML also assumes complete data for observed variables, latent variables of course have no direct data associated with them. To deal with this, we use EM to perform the parameterization instead. We first approached the integration of EM parameterization by embedding it within the MCMC process (“EM inside” below). Thus, where CaMML would generate a candidate structure in each MCMC step and then parameterize via maximizing likelihood, EM-CaMML did the same but

performed an EM parameterization instead. This proved very slow and inefficient, as EM convergence (which can often take many iterations) was required in every MCMC step.⁴

Following Friedman (1998), we re-arranged the algorithm so that the structure search occurs inside the EM process (“EM outside”). In order to compare the time complexity of each, let D be the number of rows in the data, K be the number of nodes including a latent node, L_{em} be the approximate number of iterations for EM to converge, L_{mcmc} be the approximate number of iterations for MCMC to finish, and $O(I)$ be the time complexity of performing inferences in EM,⁵ then a rough estimate of time complexity by breaking the process into “Sampling” and “Scoring” of MCMC is:

$$\begin{aligned} O(MCMC) &= L_{mcmc} \times (O(Sampling) + O(Scoring)) = L_{mcmc} \times (K \times D + 1) \\ &= L_{mcmc} \times K \times D \end{aligned}$$

as the complexity of sampling dominates the whole process of MCMC. Then for EM outside, let “Filling” represent the operation of assigning expected counts to data in the E_step, then the time complexity of EM outside will be:

$$\begin{aligned} O(EMoutside) &= L_{em} \times O(E_step) + L_{em} \times O(M_step) + L_{em} \times O(MCMC) \\ &= L_{em} \times D \times (O(I) + O(Filling)) + L_{em} \times D + L_{em} \times L_{mcmc} \times K \times D \\ &= L_{em} \times D \times O(I) + L_{em} \times D + L_{em} \times L_{mcmc} \times K \times D \\ &= L_{em} \times D \times O(I) \end{aligned}$$

if we assume $O(I) \gg L_{mcmc} \times K$. Then for the EM inside, the time complexity will be:

$$\begin{aligned} O(EMinside) &= L_{mcmc} \times (O(E_step) + O(M_step)) + O(MCMC) \\ &= L_{mcmc} \times (L_{em} \times (D \times O(I)) + L_{em} \times D) + L_{mcmc} \times K \times D \\ &= L_{mcmc} \times L_{em} \times D \times O(I) + L_{mcmc} \times L_{em} \times D + L_{mcmc} \times K \times D \\ &= L_{mcmc} \times L_{em} \times D \times O(I) \end{aligned}$$

if we assume $L_{em} \times O(I) \gg K$. Finally we can get the ratio:

$$O(EMinside)/O(EMoutside) = L_{mcmc} \times L_{em} \times D \times O(I) / L_{em} \times D \times O(I) = L_{mcmc}$$

This result tells us the ratio is a constant L_{mcmc} , which means the ratio is roughly the number of MCMC iterations. The only assumption here is that that time complexity of inferences in EM’s E_step is much greater than the MCMC sampling process. This is fair as the inferences requiring a significant number of conditional probability calculations are obviously more complex than performing a number of MCMC sampling iterations (which consist of TOM mutations and hashing with constant time complexity).

To validate our theoretical result, we performed experiments to show the difference of execution time between EM inside and EM outside. By default, there are 33 Simulated Annealing epochs and 200000 MCMC epochs by default, so EM inside will not finish in a practical time. Thus we did the test using 100, 500, 1000, 2000 MCMC epochs (this will decide how many TOMs are sampled). The test networks we used are Space Mission (Matheson, 1990) and Earthquake (Korb and Nicholson, 2010), shown as in Figure 5.3a and 5.3b respectively. We selected these two networks as the Space Mission network has the same structure as the Big-W trigger and the Earthquake network is the same as

⁴This was true even though our EM takes the convergence point from the EM used in previous MCMC step instead of being reinitialized by random parameters every time, speeding up the process somewhat.

⁵This is an NP-hard problem (Cooper, 1990), and the approximate inference of some special cases can be performed with time polynomial complexity, but the general problem of approximating conditional probabilities with BNs is still NP-hard (Dagum and Luby, 1993).

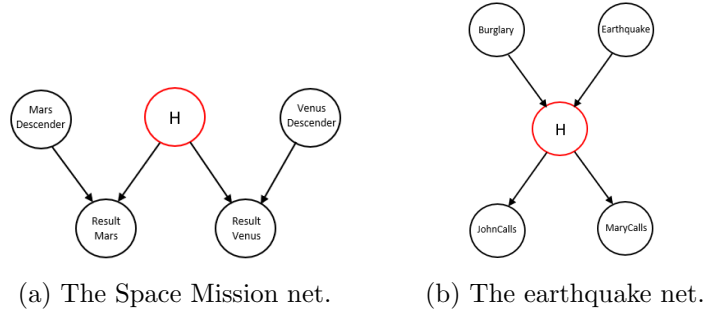


Figure 5.3: Test networks for EM integration experiment (node “H” represents the latent).

Friedman’s example of a model that can be simplified with a latent node (see Figure 1.1). The test data⁶ has five different sample sizes (100, 500, 1000, 2000, 5000), and we simulated 30 datasets per each sample size using forward sampling in Netica (*Netica API*, 2012).

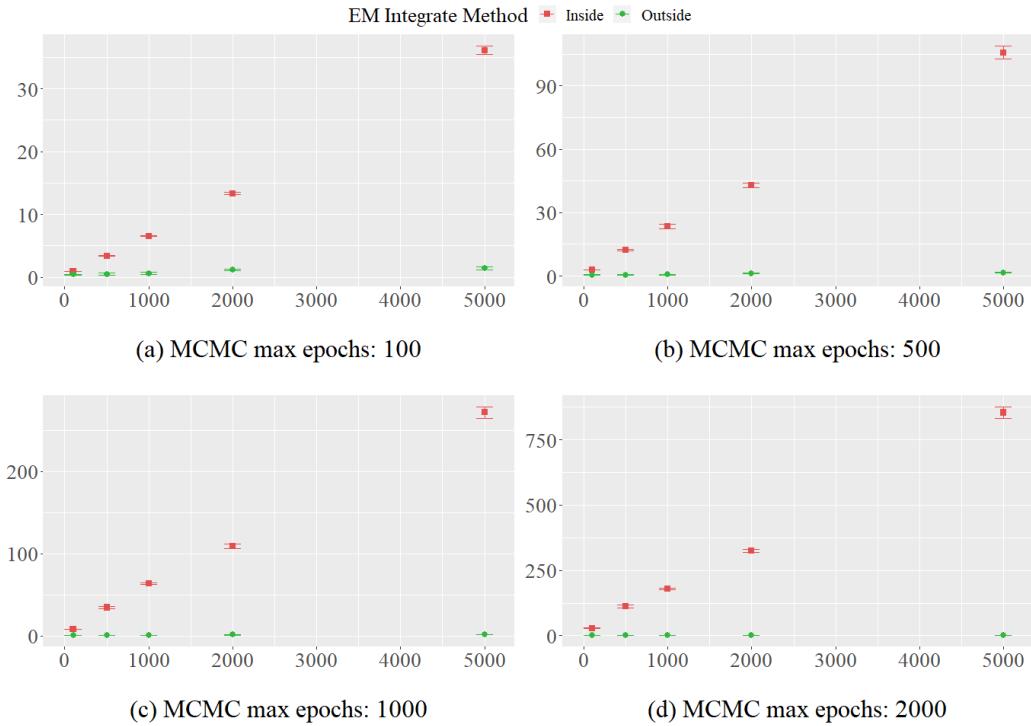


Figure 5.4: The execution time of EM inside and outside of Space Mission network (x and y axis represent sample size and execution time respectively).

The two versions of EM-CaMML were run under the same conditions for this study (i.e., maximum iterations of 30 and a threshold of 0.001). EM outside started by using a random initial structure while EM inside starts by an empty structure (which is the default in the CaMML’s Simulated Annealing). We ran each of the two implementations once per dataset, and all average execution time results (shown in 95% confidence intervals) are shown in Figure 5.4 and 5.5 of the two networks respectively. We use the x axis to represent the sample size and y axis to represent the execution time (seconds).

From the graphs we can see that the general execution time of EM inside is far greater than EM outside and also grows dramatically. In order to verify the theoretical ratio

⁶The test data here is also part of the datasets used in Section 5.2.

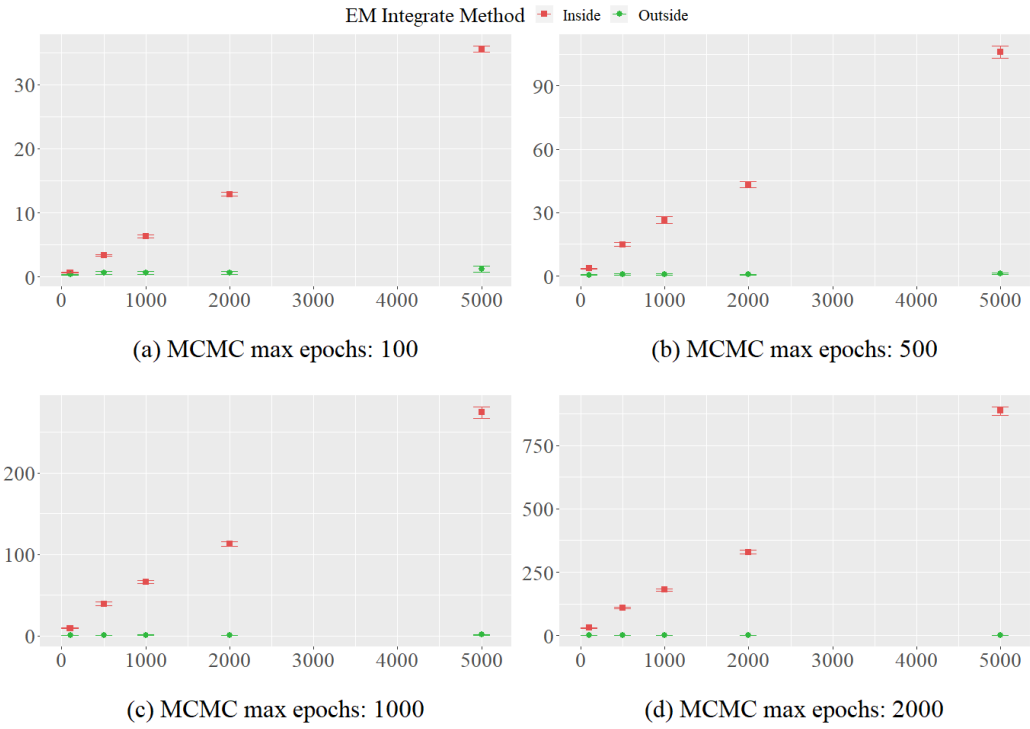


Figure 5.5: The execution time of EM inside and outside of Earthquake network (x and y axis represent sample size and execution time respectively).

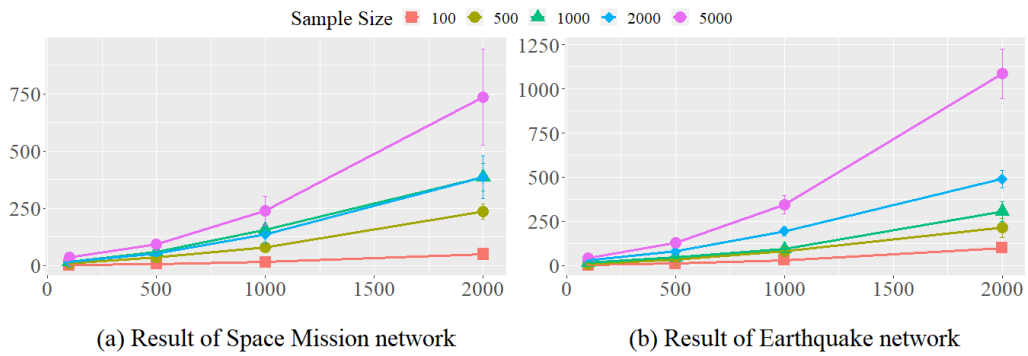


Figure 5.6: The execution time ratio of EM inside and outside by sample size (x and y axis represent number of MCMC iterations and the ratio respectively).

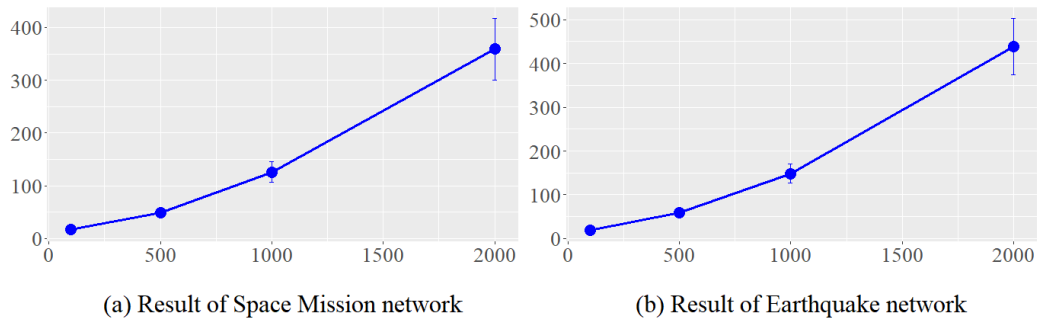


Figure 5.7: The execution time ratio of EM inside and outside (x and y axis represent number of MCMC iterations and the ratio respectively).

L_{mcmc} , we also derived the ratio (in 95% confidence intervals) in practice against the number of iterations. Figure 5.6 and Figure 5.7 show the ratios in terms of different sample sizes and the overall ratio for the two test networks respectively. We use the x axis to represent the number of MCMC iterations and y axis to represent the ratio in practice. These curves demonstrate the ratio is either linear or greater than linear for both graphs, as the results may be affected by data cleaning and correctness, especially when the sample size is small. So in general, all the results convinced us to adopt EM outside for the remainder of our study. Finally, we implemented the EM-CaMML algorithm as follows:

1. Start with a candidate latent structure.
2. Start with randomised parameters for the latent node.
3. LOOP: Use the candidate parameterized model to complete the data for the latent variable.
4. Use the completed data to search over the model space as per normal. Identify the best candidate structure, and parameterize using the completed data.
5. IF latent variable parameter changes below threshold, STOP.
6. ELSE CONTINUE.

After we decided how to integrate EM, then the next question is whether EM-CaMML converges. As the EM used in EM-CaMML is maximising the likelihood, if we assume the model learned is M given dataset \mathcal{D} , then the posterior probability of M given \mathcal{D} will be:

$$P(M|\mathcal{D}) = P(\mathcal{D}|M) \times P(M)/P(\mathcal{D}) \quad (5.2)$$

Based on Bayes' rule, as the evidence $P(\mathcal{D})$ is a constant, maximizing the posterior probability $P(M|\mathcal{D})$ equivalents to maximizing the likelihood $P(\mathcal{D}|M)$. If there is no latent node, there is no need to run EM, as the EM-CaMML will converge due to the properties of the original CaMML. When there is a latent node, EM-CaMML is trying to find the best model at M_step, in regards to the expected counts learned in the E_step. Then E_step of the next iteration re-estimates the expected counts given the model learned from the previous iteration. So in EM-CaMML, M_step tells us what the best model should be from the given the expected counts and E_step tells what the expected counts should be given the current best model. If the estimation of the expected counts is a local maximum, then the model learned in M_step will be the best model in terms of the local maximum and the E_step in the next iteration will not improve the expected counts estimation much, thus EM-CaMML converging to a local maximum. Thus the MML will be improved as well, as we assume the best model has the minimum encoding length. In practice, the result structure will not change much after a few iterations.

However, maximising the posterior in each EM iteration does not necessarily maximize the likelihood. The M_step updates the structure and the E_step updates the expected counts based on the structure by maximising the likelihood in our implementation. This means the structure learned in every iteration will impact both the likelihood maximization in the current iteration and the expected counts used in the next iteration. In other words, EM-CaMML is trying to maximize the posterior and likelihood at the same time, as if the EM converges, then the EM-CaMML will converge.

5.1.3 EM-CaMML Workflow

We now present the complete high level workflow (shown as in Figure 5.8) for EM-CaMML in this section. It starts with a data pre-processing module to validate the input data and compute required statistics (e.g., conditional dependencies between every two nodes). Then the latent node detection module looks for triggers. If found, the trigger structure is passed to EM-CaMML, using soft arc constraints (O’Donnell, 2010) with the optimized prior probability. Such optimized values can be given manually or we can apply some methods to get the best priors (e.g., see Section 5.2.3). Then EM-CaMML will perform the EM-MCMC paired iteration described above to estimate a posterior distribution over the latent model space.

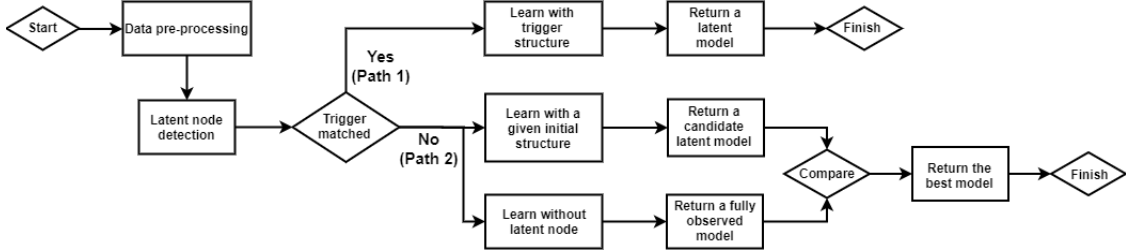


Figure 5.8: EM-CaMML workflow.

If no trigger is matched, CaMML can be run without EM, or, alternatively, latent nodes can nevertheless be posited and passed to EM-CaMML under a variety of scenarios. The latter would be justified if we have any reason to suspect the presence of latents other than trigger patterns in the data. Since EM-CaMML is fully capable of disconnecting the latent variables if the data do not support them, there is in principle nothing lost by doing so other than time. For our studies, we chose the latter path, using a few different initializations for connecting the posited latent node (initialization details are in Section 5.2.1). These different runs of EM-CaMML were compared with the best fully observed models (by running the original CaMML), and the best models and model equivalence classes, as judged by MML. In order for the MML comparison to be fair, the MML cost of the latent model ignored the “data cost” associated with the latent nodes themselves (these are hypothetical observations of latent nodes based upon EM estimated parameters and used in the MCMC search). The MML cost associated with latent nodes included the structure cost and parameters, of course.

Since we choose random parameters for the generating networks, these created weak dependencies between latent nodes and observed nodes, as in Friedman’s study (Friedman et al., 1997). So, again following Friedman, we fixed the structure and ran standard EM for a number of iterations (we used 10 iterations) to obtain stronger dependencies before running the main experiment (Friedman et al., 1997), otherwise the latent nodes were always disconnected.

Another thing we should mention is the EM-CaMML’s sampling space. As in the original CaMML, EM-CaMML performs a simulated annealing search to find a good model as a starting point and then Metropolis-Hastings sampling over the TOM space to estimate a posterior distribution over DAGs, SECs and MMLECs (the latter two are statistical equivalence classes of models). For N nodes, EM-CaMML performs MCMC with, by default, $200 \times N^3$ samples of TOMs during each run. During sampling, EM-CaMML generates a new candidate TOM by performing a mutation on the current TOM, by adding, deleting or reversing arcs; it then accepts or rejects the candidate for the next sample per the Metropolis-Hastings algorithm; finally, it updates its DAG and equivalence class counts. In these steps, described at this level, there is no difference between EM-CaMML and the original CaMML (as we described in Section 3.4.3), although, to be sure,

the set of mutations have effectively been “extended” to include connections to latent nodes.

5.2 Experiment

We evaluated EM-CaMML in comparison with the PC, FCI and SEM algorithms, which are the only other widely available causal discovery algorithms incorporating latent variables. Because we are only learning small latent structures (for the purpose of testing how the trigger detect function might contribute to general causal discovery), some of our test networks were proper subnets of original networks. In addition, Covered Big-W cases were difficult to find in the real-life networks we examined, so we reused Big-W networks by simply adding an additional covering arc.⁷ Of course, we are interested in testing the EM-CaMML process with both positive (latent) and negative (non-latent) cases, hence, we used an equal number of models with latent nodes and with no latents.

Parameter	Value
Network Type	Big-W, Cov Big-W, Latent, No Latent
Number of Obs Nodes	4, 5, 6, 7
Sample Size	100, 500, 1000, 2000, 5000
Number of Runs	30 per sample size

Table 5.1: Experiment configurations for comparing EM-CaMML with PC, FCI and SEM.

We used four generic types of real world networks for testing: those containing a Big-W, Covered Big-W, other latent structures and nets without latents (i.e., fully observed nets). These ranged from four to seven observed variables, plus a possible latent node. Because most of the original networks have more than 7 nodes, which is more than the size of the discovered triggers, we select a proper subnet from each network whose size is less than or equal to 7 (see Appendix C for the subnet structures). Then, each subnet was used to generate thirty samples in sizes 100, 500, 1000, 2000 and 5000 using forward sampling in Netica (*Netica API*, 2012). In total, for each network, there were $5 \times 30 = 150$ datasets generated,⁸ shown in Table 5.1.

5.2.1 Initial Structure Selection

The constraint-based algorithms PC and FCI require no initial structure to find latent models, as their discovery uses only local conditional dependency tests. Both EM-CaMML and SEM start with an initial structure indicating how any latent variables might be related to observed variables. The EM process is very sensitive to this initial structure, so we tested with three distinct ways of initializing structures for EM-CaMML and SEM.

The first initialization (ini1) we used was to set a latent node as the parent of every observed node, which mimics Friedman’s approach with SEM and is like factor analysis. The second initialization (ini2) used conditional dependency tests to allocate observed variables into two groups: parents of the latent variable, which were marginally independent of each other and children of the latent, which were marginally interdependent. These initializations play a role similar to the Big-W triggers, but are given much weaker priors — so such initialization methods can be called “soft trigger” methods. Big-W requires explicit conditional dependency tests; if the data complies with such dependencies, then a hard trigger is learned, in addition to the connections to and between observed nodes. In

⁷We randomly generated parameters for each affected node, while maintaining the original marginal distributions.

⁸The test datasets are available at: <https://sourceforge.net/projects/em-camml-test-data/>

contrast, a soft trigger is used just for getting a relatively good starting point for a latent discovery algorithm, and the arcs (having a weak prior) can be altered easily during the learning process of the algorithm. The last method (ini3) used random structures as a starting point, but with some constraints: the latent should not be a leaf (no children); it should not be connected as an intermediate node in a chain (with no other connections); and it should not be disconnected. The reason for these constraints is that the latent node should play a significant role in the causal structure, and in these three cases its contribution can be marginalized out. So we used these three quite different initial structures in the experiment to get a good view of how much the starting point can affect the results. Of course for EM-CaMML, if a trigger was detected in preprocessing, then none of these initializations were used, but instead the trigger itself was used as a starting point. For SEM, while ini1 corresponds to Friedman’s initialization, we applied all three initializations to give a proper range of comparisons with EM-CaMML. The latent node was given the same arity (number of states) as in the true network (i.e., favoring SEM with the truth about arity).

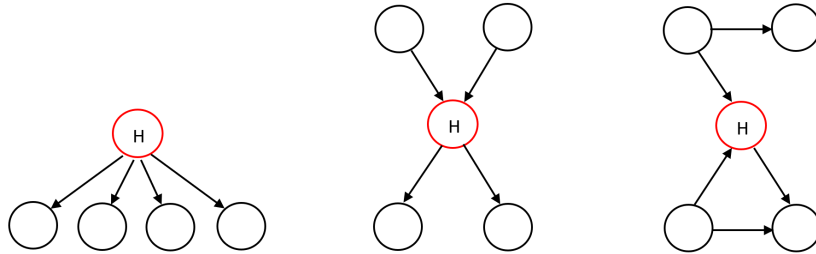


Figure 5.9: Initializations 1, 2 and 3 (node “H” represents the latent).

For example, in Figure 5.9 there are four observed variables and a latent node H. So ini1 makes them all children of the common cause latent node H (as the first structure in Figure 5.9). Based on ini1, if any nodes are marginally independent, then they become parents of H in ini2. The rightmost ini3 provides EM-CaMML with a random initial structure.

5.2.2 Generating Complete DAGs for PC and FCI

As constraint-based algorithms, PC and FCI learn causal models by using local conditional dependency tests. The resultant structure can contain double headed arrows, e.g., $X \leftrightarrow Y$, which indicate the detection of unmeasured common causes. Apart from these, PC could return some undirected arcs, e.g., $X—Y$. FCI may report open arrows, e.g., $X \circ \rightarrow Y$, $X \circ - \circ Y$, which mean the algorithm could not determine the arc direction (Spirtes et al., 1993). Such incompleteness prevents us from learning the model’s parameters, so we need to complete them in order to compare learned models across algorithms.

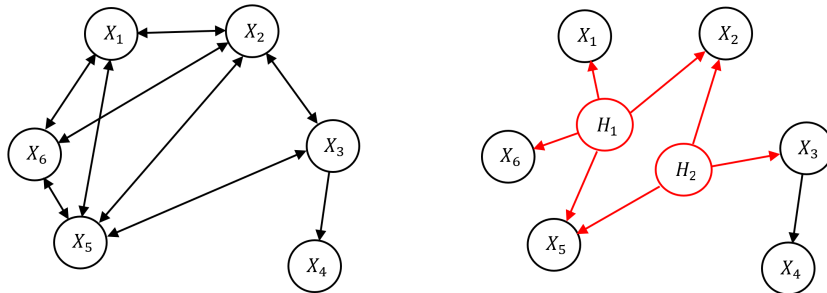


Figure 5.10: Example conversion of bi-directional cliques into latent nodes (H_1 and H_2).

PC and FCI may also return more than one double-headed arc, indicating multiple latent nodes. Since any clique connected by double-headed arcs can have their dependencies accounted for by a single latent node just as well, but more simply, than the suggested number of hidden common causes. For experimental comparison that is what we decided to do, i.e., introduce one latent node as the hidden common cause for each such clique (including two-node cliques, of course). For example, in Figure 5.10, the left graph contains two bi-directional cliques: $\{X_1, X_2, X_5, X_6\}$ and $\{X_2, X_3, X_5\}$, and the right graph is the result of replacing the two bi-directional cliques by two latent nodes H_1 and H_2 along with new arcs in red.

Of course, a structure could contain only one bi-directional arc, which is obviously a bi-directional clique that can be turned into a latent node. We can easily demonstrate that this is a genuine simplification for any case where multiple latent nodes are an issue. So all the bi-directional cliques of PC and FCI outputs will be transformed into latent nodes before evaluation.

Lemma 5.2.1. *Given a bi-directional clique Q over $n > 1$ observed nodes $\{X_1, X_2, \dots, X_n\}$ (which contains $n(n-1)/2$ arcs), a DAG Q' with a latent node H as the root parent of all observed variables in $\{X_1, X_2, \dots, X_n\}$ is the simplest DAG in $\{X_1, X_2, \dots, X_n, H\}$ having the same observable conditional dependencies among $\{X_1, X_2, \dots, X_n\}$ as Q , where simpler means here having fewer arcs.*

Proof. The observable conditional dependencies in Q' are the same as in Q , since conditioning on any subset of observed variables $\{X_1, X_2, \dots, X_n\}$. The remaining nodes are dependent in both Q' and Q , and there are no independencies (assuming faithfulness). The only fully observed network having this property is the clique Q itself, which has $n(n-1)/2 \geq n$ arcs (for $n = 2$ the inequality doesn't hold, however in that case there is no issue of introducing more than one latent node). Any alternative latent variable model with exactly $n(n-1)/2$ arcs is not simpler than Q' in the relevant sense. As Q' has n arcs, any structure simpler than Q' will have something missing (i.e., arcs, dependencies). If X_k is disconnected, then it is independent of other observed nodes and the set of conditional dependencies has not been preserved. If X_k is not disconnected, then it must be connected to some other observed node. Since we're talking about a latent model (the fully observed case was handled above), and the number of arcs is fewer than $n(n-1)/2$, this implies that some *other* observed variable is disconnected, when again the set of conditional dependencies is not preserved. \square

True Arc	Learned Arc	Lower Bound	Upper bound
Null	—	\rightarrow or \leftarrow	\rightarrow or \leftarrow
Null	$o \rightarrow$	\rightarrow	\rightarrow
Null	$o-o$	\rightarrow or \leftarrow	\rightarrow or \leftarrow
\rightarrow	—	\rightarrow	\leftarrow
\rightarrow	$o \rightarrow$	\rightarrow	\rightarrow
\rightarrow	$o-o$	\rightarrow	\leftarrow

Table 5.2: Arc orientation rules for PC and FCI output.

After replacing bi-directional arc cliques by latent nodes we need to resolve any remaining incompleteness in the returned graphs. Since those graphs are ambiguous regarding the generated DAG, we produced lower and upper bound solutions, that is, permissible graph completions which were the closest or furthest from the true graph in terms of edit distance. This bounds the measurement of performance of the algorithms from best to worst respectively in edit distance (see Section 3.5.1). If the two nodes were not connected

True arc	Learned arc	Distance
$X \rightarrow Y$	$X \rightarrow Y$	0
	$X \leftarrow Y$	1
	null	1
null	$X \rightarrow Y$	1
	$X \leftarrow Y$	1
	null	0

Table 5.3: Edit distance rules used in this thesis.

in the true structure, then we uniformly randomly oriented the partial arcs “—” and “o—”. Of course, all the above rules were subject to the constraint of avoiding introducing any cycle or creating or destroying an uncovered v-structure. In particular, our orientation rules are shown in Table 5.2. Finally, after we fixed the PC and FCI output structures, we defined the rules of edit distance (shown as Table 5.3) we use for all experiments that can be directly applied to each algorithm.

5.2.3 Arc Prior Probability Optimization

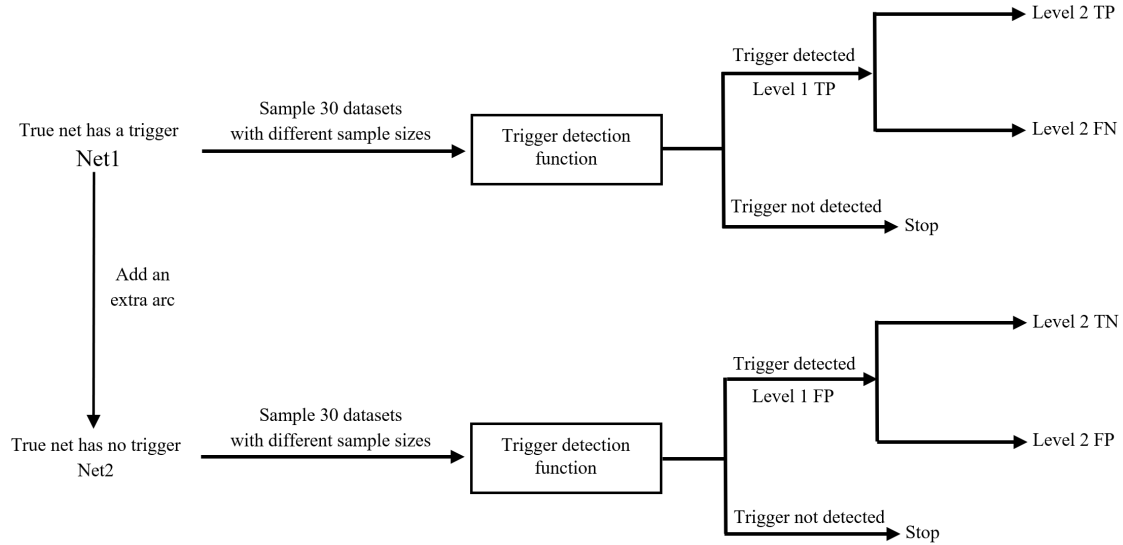


Figure 5.11: Process of arc prior probability optimisation.

To optimise the threshold choice for the trigger detection program as well as the arc prior between the nodes that match a trigger, we did a simple test by trying different thresholds and arc priors. Note that we are only optimising the arc prior if and only if a trigger is detected. If no trigger is detected (shown as “Stop” in Figure 5.11), then in the actual implementation, EM-CaMML will continue with the remaining processes (shown as the path after no trigger matched in Figure 5.8) which needs no arc priors. To be specific, there are two different tests that occur at two different stages (or levels) of the process:

1. Find the best threshold for the trigger detect function (Level 1).
2. If a trigger is detected, find the best arc prior (probability) for the subnet that matches the trigger (Level 2).

So the Level 1 result decides the number of cases that will go to Level 2 for testing, and in Level 2, if the arc prior is small, then it will be more likely that EM-CaMML will change the matched trigger subnet. Otherwise the larger the arc prior, the harder it is for EM-CaMML to change the structure. However, there are very few Level 1 false positive results in our initial experiment, so in order to get enough false positive results, we manually add an extra arc to each of the Big-W test nets (which we refer to as Net1) but keep the marginal distribution of each node in the original network the same, yielding Net2. For example, in Figure 5.12, an extra arc is added between node “ERRCAUTER” and “HRBP”, but the marginal distribution of “ERRCAUTER” and “HRBP” remains the same. We expect these variant networks are able to make EM-CaMML produce a significant number of false positive results at Level 1. Then for each test network, we sample 30 test datasets with five different sample sizes of 100, 500, 1000, 2000 and 5000. So there are $5 \times 30 = 150$ datasets in total.

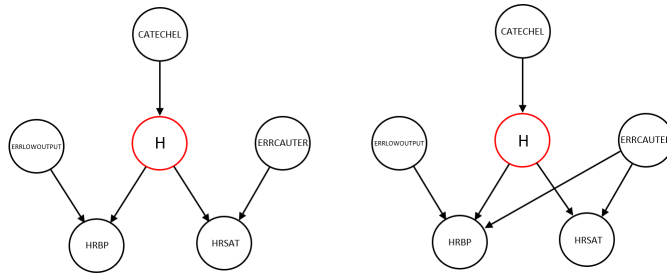


Figure 5.12: Original Big-W Alarm network and its variant (node “H” represents the latent).

For the trigger detection threshold, we first calculate how many cells of the dependency matrix (see Section 4.1) will be changed by adding an extra arc in terms of d-separation rules. For our test networks, the number of cells changed by adding an extra arc depends on the number of nodes in the structure as well as the location of the extra arc. For example, for the Big-W Alarm net, the percentage of cells changed is 0.026 (2.6%), which means if the threshold is equal or greater than 0.026, any test on the sample data using Net2 will be detected as a trigger if the size of the data is big enough and assuming there is no mistake in the χ^2 tests. This value is the upper bound of the test threshold, and we also select two numbers in between the default threshold (0.005) and the upper bound. For example, if the upper and lower bound are 0.026 and 0.005 respectively, then the testing threshold values will contain four values: 0.005, 0.01, 0.02 and 0.026. Table C.2 shows the details of each test network (the structures are shown as in Figure C.1), how we add the extra arc and all the tested thresholds.

There are still some details which need to be clarified in Level 2. Given one Level 2 test prior, any arc present in the matched subnet will be assigned the test prior. In contrast, if there is no arc present, then the arc prior between the two corresponding nodes will be $1 - \text{test prior}$. In other words, this is used to directly inform the prior probability of an arc being present in the final output models. For example, if the test prior is 0.9, and the node set H, W, X, Y, Z matches a trigger, any arc between them that is also present in the trigger will be assigned with arc prior of 0.9, and any arc not in the trigger will be assigned $1 - 0.9 = 0.1$. The tested arc prior range in this study is from 0.6 to 1.0, because 0.5 means an uninformative prior in practice (i.e., tossing an unbiased coin) and we are also interested in some values that are close to 1.0, thus we include 0.999999, 0.9999999 and 0.99999999.

The definitions of TP and FP at Level 1 are straight forward, meaning just whether a trigger subnet is matched.⁹ However, there could be various definitions of TP, FN, TN and FP for Level 2 results. Here we tried two different criteria for how to define them. In the first criteria (Criteria 1), TP means the learned network is exactly the same as Net1 in Figure 5.11, and FP means the latent node in the learned model is important (i.e., not appearing as a leaf node with no child, a parent node with only one child or the middle node in a chain). While in the second criteria (Criteria 2), TP means the latent is merely important, while the FP is same as the first level (Level 1).

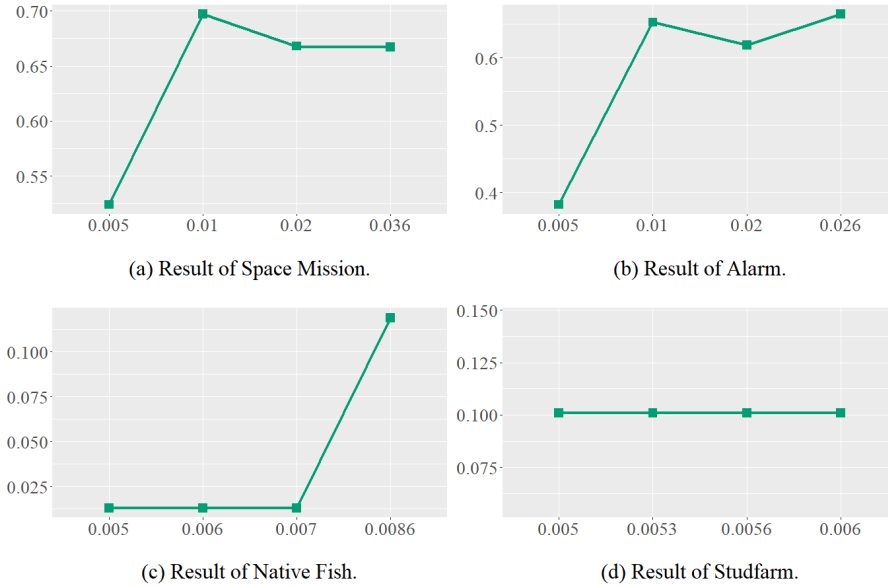


Figure 5.13: Level 1 F-score results of Big-W networks.

The Figure 5.13 shows the Level 1 F-score results (with $\beta = 1.0$) of the test networks, where the x axis represents different thresholds and the y axis represents the F-score. We can see that the F-score increases as the threshold increases in general. There is a trade-off between the threshold and accuracy, as the TN decreases when the threshold increases. For example, for the Space Mission net, the number of TN decreases from 80 to 0 while the number of FP increases from 47 to 150 as the threshold increases from 0.005 to 0.036 (not shown). The best threshold actually depends on how users evaluate the results. In this thesis, we generally stay in a conservative position, attempting to avoid FP results and permitting more TN results. Interestingly, the result line of Stud Farm is consistently flat (0.1), this is because different thresholds made no change in terms of TP and the extra arc helps to produce more FP. We expect the number of positive cases will increase when the sample size is big enough because some of the dependencies of the trigger subnet in Stud Farm are very weak. For example, the marginal dependency (measured by mutual information) between node Fred and Brian is approximately 0.0 bits.

The F-score results of Level 2 are shown as in Figure 5.14 and 5.15 of Criteria 1 and 2 respectively, where the x axis represents the different arc priors and the y axis represents corresponding F-score. In general as Criteria 2 is less strict than Criteria 1 in terms of identifying TP results, we can see that the F-score of Level 2 grow more significantly than Level 1 since more TP are identified especially for Native Fish and Stud Farm. In summary, the Level 1 threshold of 0.005 with Level 2 arc prior of 1.0 (hard constraint) is the best combination in general for both criterion, and we decided to use this combination

⁹Note that for the Native Fish network, there are two triggers that share the same latent node in the model. So we treat it as a TP result if either of them is matched.

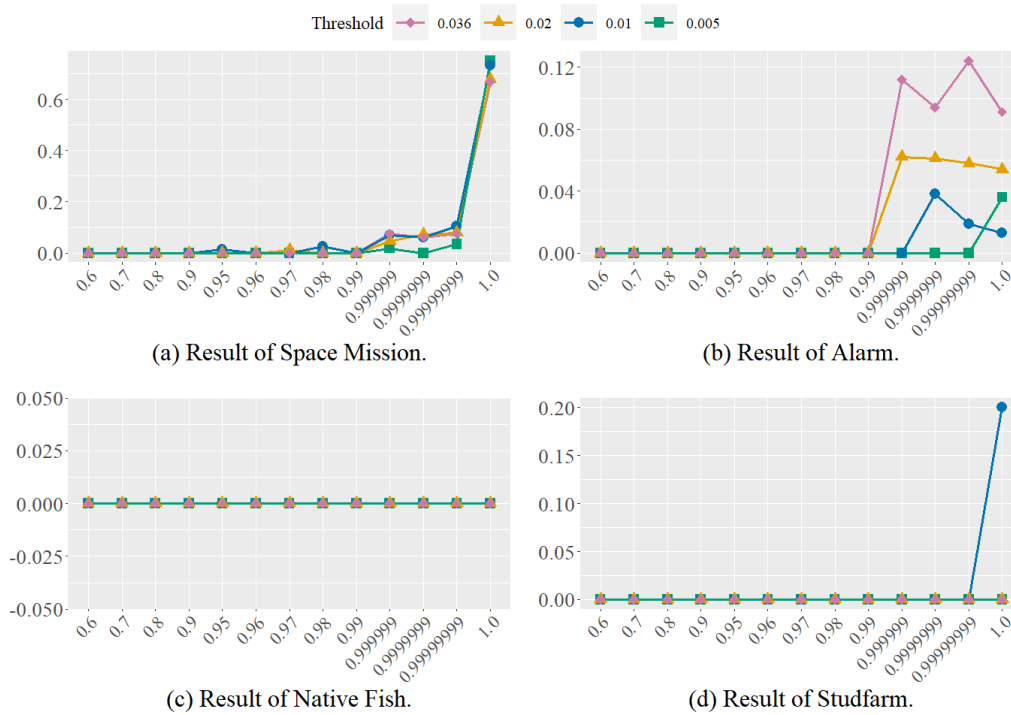


Figure 5.14: Level 2 F-score results (Criteria 1) of Big-W networks.

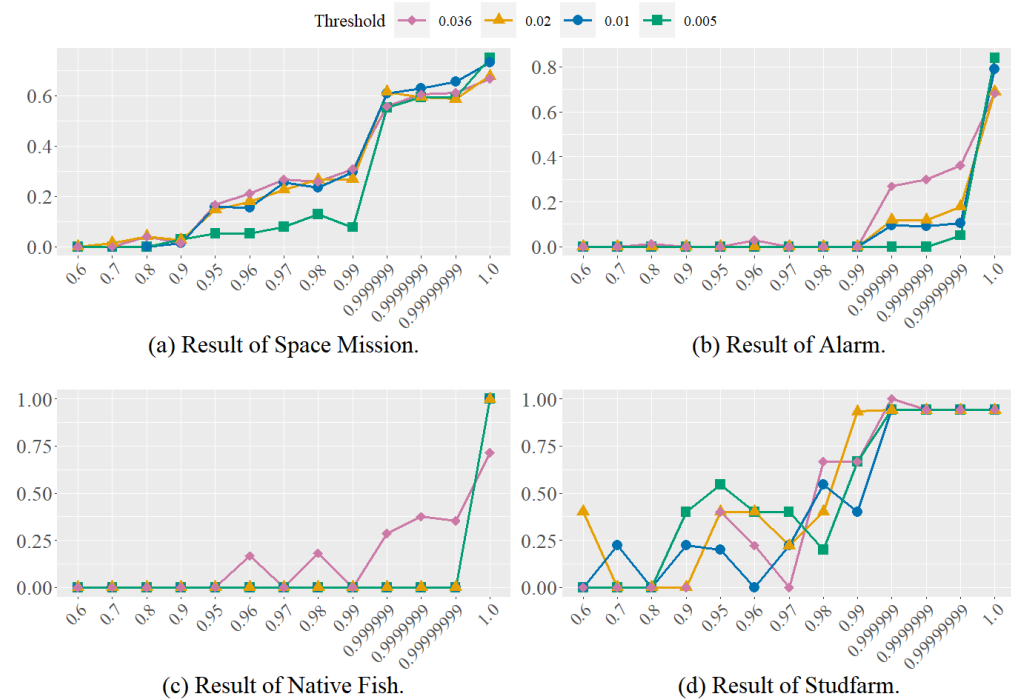


Figure 5.15: Level 2 F-score results (Criteria 2) of Big-W networks.

for EM-CaMML in the following sections of this chapter and the experiments in Chapter 6. This trigger detection threshold and arc prior optimisation study shows a general guide of how to do the parameter optimisation of using EM-CaMML. Of course, users can choose their own parameter combination which best suits their specific requirements. One thing that should be mentioned here is that SEM does not have similar parameters

available, so we will use SEM with its default values for any parameters shared with EM-CaMML.

Our goal is to compare the different algorithms experimentally to gauge their success in detecting latent nodes when present and avoiding false positives when they are not. In the following sections, we will show the actual performance of each algorithm in dealing with different types of networks. The PC and FCI implementations used here are the same as in Section 4.4, and the SEM script (applying the BIC score) is from the Structure Learning Package v1.5 (Leray and Francois, 2004; François and Leray, 2008) in Bayes Net Toolbox v5 (Murphy, 2004). Our EM-CaMML implementation shares the same initial parameters with SEM in each type of test case (i.e., a maximum iteration of 30 with EM threshold of 0.001), and we use the optimized parameters in this section to run EM-CaMML in all the following experiments. For PC and FCI, we report lower and upper bounds for any evaluation measures, bracketing their performance (see Section 5.2.2). All the results using different metrics are shown as 95% confidence intervals. We now review the experimental results in the remainder of this chapter, first separately for each generating network and then collectively.

Algorithm	Key	Algorithm	Key	Algorithm	Key	Algorithm	Key
EM-CaMML (ini1)		SEM (ini1)		FCI (lower)		PC (lower)	
EM-CaMML (ini2)		SEM (ini2)		FCI (upper)		PC (upper)	
EM-CaMML (ini3)		SEM (ini3)					

Table 5.4: The legend keys used in all experiment result graphs in Section 5.2.4 - 5.2.7.

Due to the limited space, we show the legend keys of each algorithm in Table 5.4, which will be used in all result graphs. Different algorithms are shown in different colors, and the specific initializations used by SEM and EM-CaMML are shown in brackets following the algorithm name (e.g., EM-CaMML (ini1) denotes EM-CaMML result using ini1). Similarly, we use the text in the brackets to show whether a PC or FCI result is fixed as lower or upper bound (e.g., PC (lower) means the lower bound of a PC result). For the four test network types (Big-W, Covered Big-W, Latent and No Latent), we will show the results for each in the following sections.

5.2.4 Alarm

As a well-known BN, the aim of the Alarm network is to advise users of potential problems by providing specific messages in a real world domain. It implements an alarm system for patient monitoring by calculating probabilities of different diagnoses given available evidence (Beinlich et al., 1989). Here we selected a subnet which contains six nodes of Alarm as one of our Big-W test cases (shown in Figure 5.16), and the node “HR” is converted to a latent node for our testing purpose.

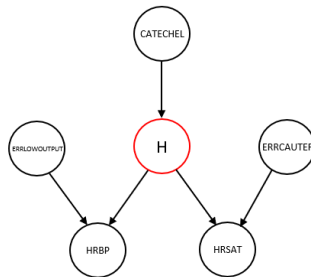


Figure 5.16: Big-W Alarm network (node “H” represents the latent).

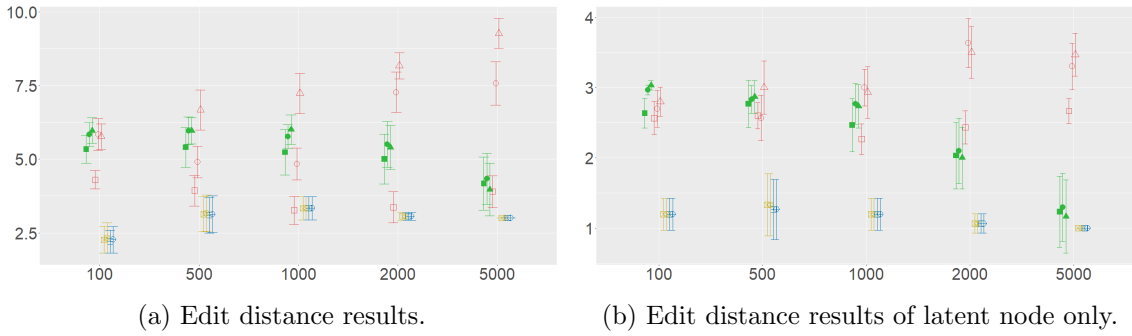


Figure 5.17: Edit distance results of Big-W Alarm cases.

In Figure 5.17 we can see PC and FCI outperforming other methods in general, at least until the largest sample size, when EM-CaMML catches up. We can also see that the lower and upper bounds for PC and FCI are the same, or nearly the same, throughout. As the upper bound and lower bound fixes for PC and FCI outputs are almost the same because of the orientation rules and constraints (mentioned in Section 5.2.2). For example, in Figure 5.18, with the left model being the original output by FCI of the Big-W Alarm network, we can see there are several circles and a bi-directional arc, and the lower and upper bound fixes will be the same as the structure on the right after replacing the bi-directional arc by a latent node H. EM-CaMML steadily improves with sample size, while SEM seems to diverge from the truth, except when provided ini1 (its native initialization).

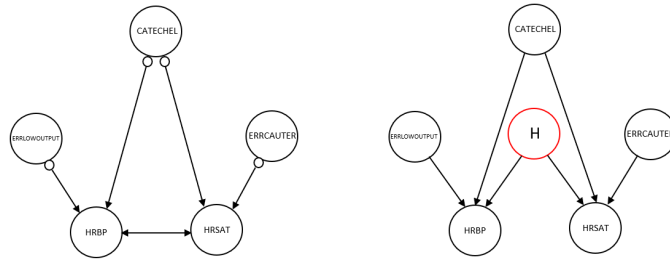


Figure 5.18: How FCI output is fixed of Big-W Alarm network (node “H” represents the latent).

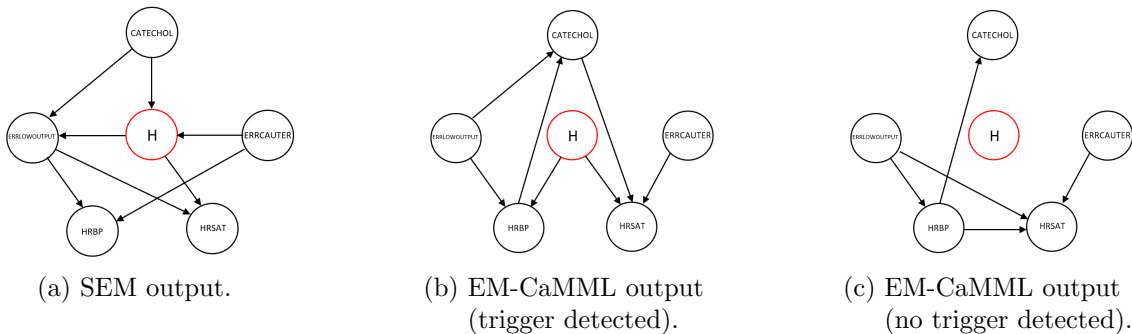


Figure 5.19: Typical example of SEM and EM-CaMML learned model of Big-W Alarm using ini2 at sample size of 2000 (node “H” represents the latent).

In the actual SEM learned structures, the latent node becomes more connected as sample size increases, but SEM fails to find the right connections, a typical learned structure is shown as Figure 5.19a. We can see SEM (ini2) is not diverging from the truth, but it also doesn’t appear to be converging on it with larger samples, whereas EM-CaMML does, regardless of initialization. The Figure 5.19b shows an example output of EM-CaMML (ini2)

when a trigger is detected between nodes ERRLOWOUTPUT, ERRCAUTER, HRBP and HRST. The trigger detect function helps EM-CaMML get a good structure except for the connections with node CATECHOL. However, for EM-CaMML, around 50% of its outputs (see Figure 5.21a) have a disconnected latent node (e.g., shown as Figure 5.19c). The model fails to detect the latent node H but the number of incorrect arcs are still less than for SEM and it at least captures some dependencies.

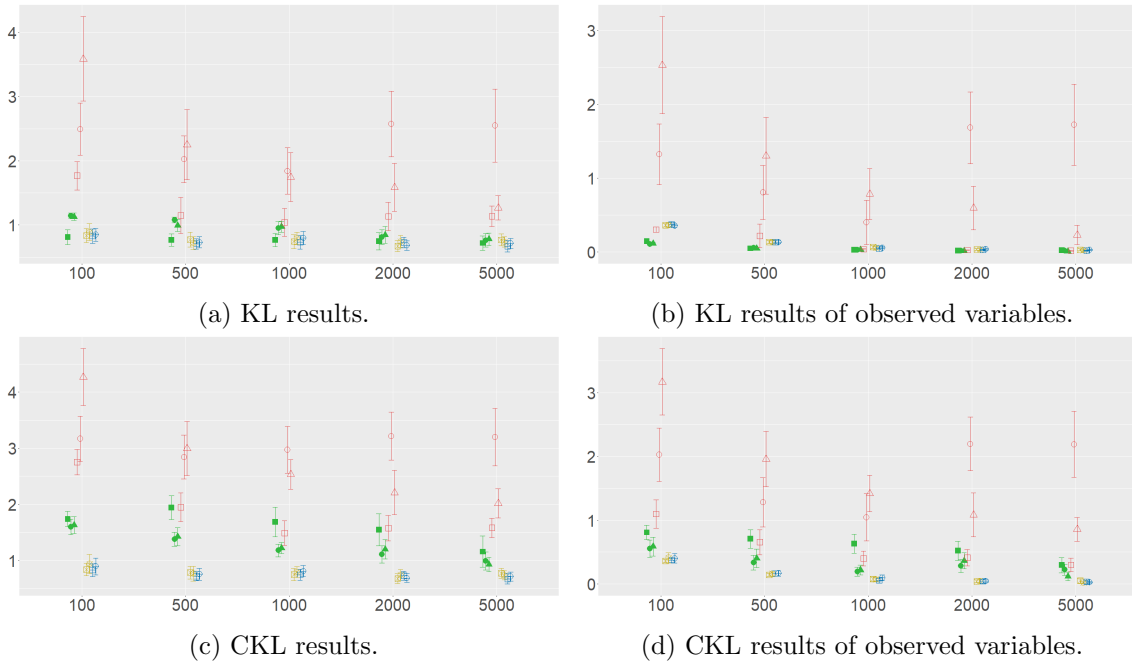


Figure 5.20: KL and CKL results of Big-W Alarm cases.

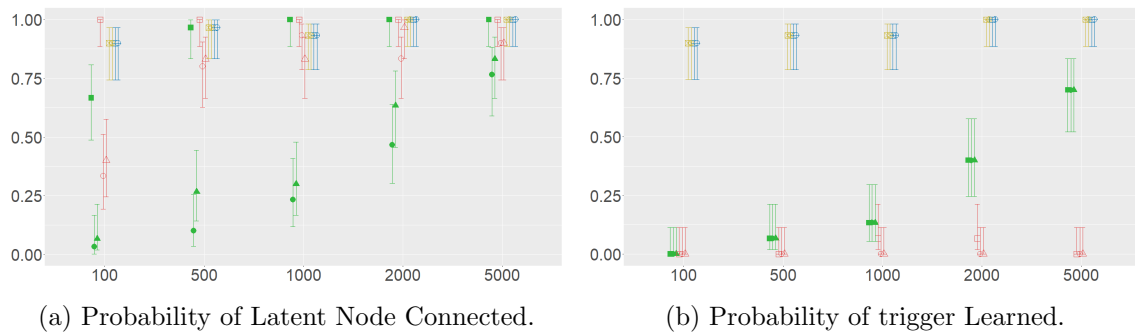


Figure 5.21: Results of latent node connection and trigger subnet learning of Big-W Alarm cases.

In general, the KL and CKL results in Figure 5.20 are quite consistent with the edit distance measures. On KL, EM-CaMML, SEM (ini1), PC and FCI are consistently good. On CKL, PC and FCI are consistently good, EM-CaMML converges to similar performance as samples increase, while SEM shows little or no improvement. We also computed the chance of each algorithm correctly learning that a latent node exists (Figure 5.21a), as well as learning an existing trigger structure (Figure 5.21b). We can see that ini1 produces better true positive rates for connected latent nodes for both EM-CaMML and SEM. Presumably, having the latent as the root of observed nodes better allows it to explain interdependencies. Overall, the latent node is getting more connected as sample

size increases, as expected. For trigger subnet learning,¹⁰ PC and FCI consistently perform the best, however EM-CaMML converges on that optimal performance as sample size increases, regardless of initialization. SEM, on the other hand, is never able to find the triggers with any initialization.

5.2.5 Stud Farm

The Stud Farm network (Nielsen and Jensen, 2007) was built to model the genealogy of a number of horses. A life-threatening genetic disease is carried via a recessive gene, and a horse is classified as either pure, carrier or sick. The probability of being a carrier horse increases if it is a descendent of one who has been diagnosed with the disease. As there is a size limit of EM-CaMML, we select eight nodes from the original network choosing node Ann as the latent. In order to form a Covered Big-W case, we manually add an extra arc connecting node L (an unknown father) and Brian (but maintaining the original marginal distributions, as mentioned at the beginning of Section 5.2), shown in Figure 5.22.

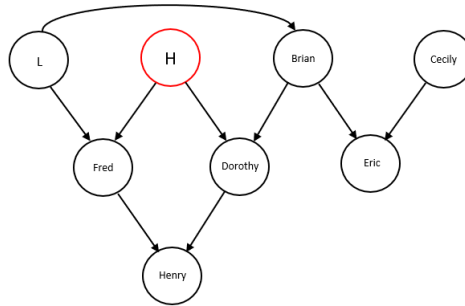
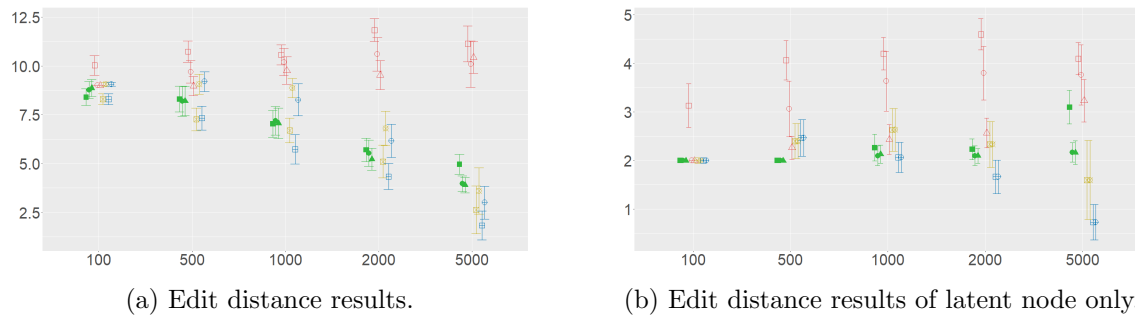


Figure 5.22: Covered Big-W Stud Farm network (node “H” represents the latent).



(a) Edit distance results. (b) Edit distance results of latent node only.

Figure 5.23: Edit distance results of Covered Big-W Stud farm cases.

From Figure 5.23 we can see that EM-CaMML has a similar performance as PC/FCI in terms of edit distance results in general, and their results are getting significantly better as the sample size increases. However, the results for SEM do not improve, and this is because the latent node is getting more connected which causes more incorrect arcs that connect to the latent. As we can see from Figure 5.23b, the edit distance of the latent node only shows SEM bringing in more incorrect arcs that connect to the latent node. EM-CaMML (ini1) does the same only at the sample size of 5000, but its results are still much better than the SEM results. Because the arc strengths of the true structure are not strong (as every node has a very biased CPT value around 1.0-99.0 and only low mutual information¹¹ between every two nodes of around 0.03), the latent node and some observed

¹⁰The “trigger subnet” is just the arcs present in the trigger structure. For example, in the Big-W Alarm network, nodes H, ERRLOWOUTPUT, ERRCAUTER, HRBP and HRST form a Big-W trigger.

¹¹Details of mutual information are shown in Section 6.1.

nodes are disconnected in many SEM outputs of sample size 500 (an example is shown as Figure 5.24a). Such disconnected structure is produced by other algorithms as well, but the edit distance results improve as sample size increases except for SEM. This is because the latent node is more connected at sample sizes of 5000 (see the example output shown as Figure 5.24b) but also more incorrect arcs are produced as well.

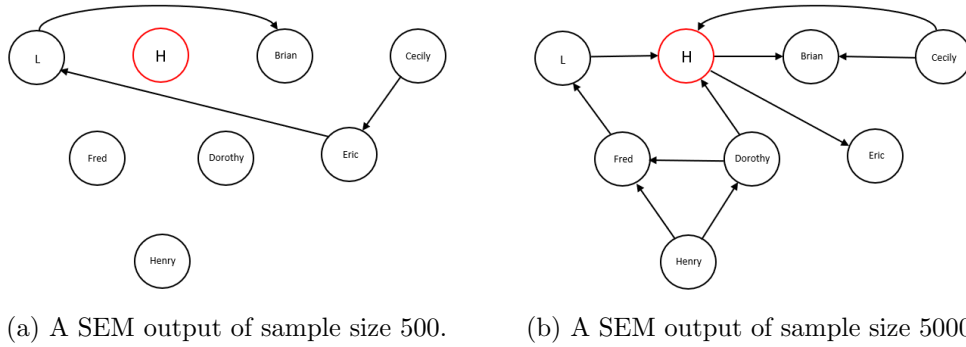


Figure 5.24: Example of SEM output of Covered Big-W Stud Farm (node “H” represents the latent).

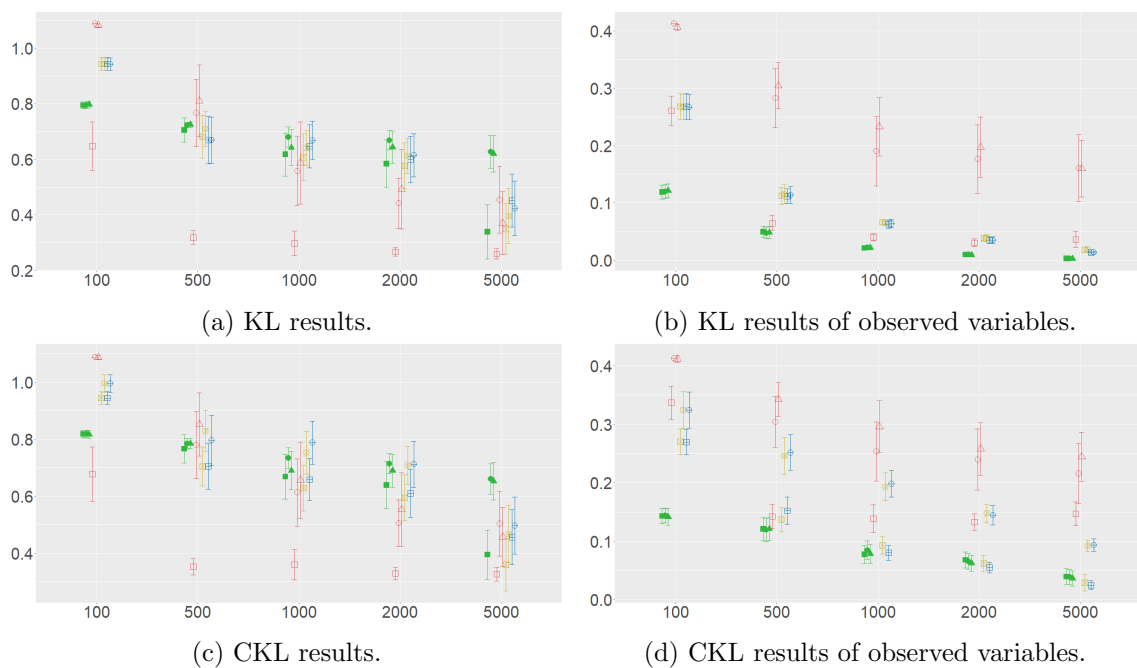


Figure 5.25: KL and CKL results of covered Big-W Stud Farm cases.

The KL and CKL results of Covered Big-W Stud farm (shown as in Figure 5.25) are not so consistent with the edit distance results. SEM performs even better in general. This is because the latent node is disconnected in most EM-CaMML, PC and FCI outputs, and this is indicated by Figure 5.23b as the edit distance of latent node only is around 2 across different sizes. As the latent node is connected more in SEM results, its CPT and children’s CPTs have more opportunities to be learned correctly, thus helping improve the KL/CKL results of the whole structure.

The connection status of the latent node are confirmed in Figure 5.26a too as we can see the probability of a connected latent node is much higher in SEM outputs than other algorithms except at the largest sample size where others catches up. However, in terms of the trigger subnet (shown as in Figure 5.26b), PC and FCI have a better performance in

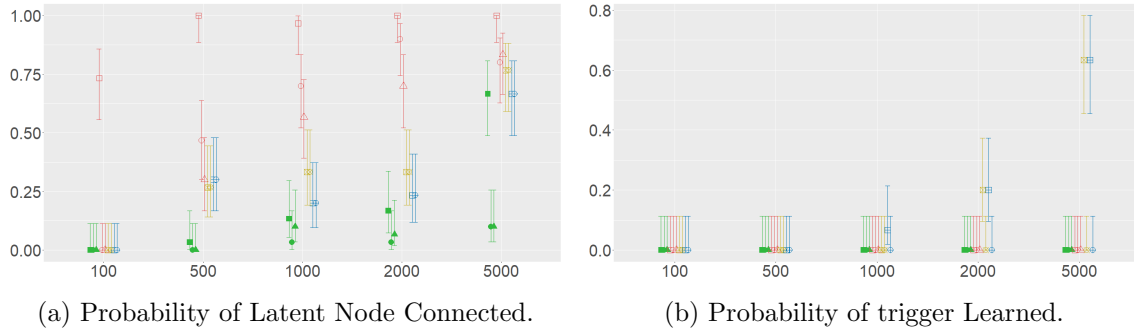


Figure 5.26: Results of latent node connection and trigger subnet learning of covered Big-W Stud Farm cases.

learning it correctly. Both EM-CaMML and SEM failed to learn the trigger subnet with any initialization.

5.2.6 Bookbags

The Bookbags network (Phillips and Edwards, 1966) has the Naive Bayes structure of Figure 5.27, which simulates two book bags (represented by two states in the hidden node, called Book Bag in the original network) each with a different mixture of 10 red and blue poker chips. The posterior on the root node represents the probability of which bag is the most likely source after each of 5 draws. In terms of this structure, we expected Initializations 1 and 2 to produce the best results, since they correspond to the true structure.

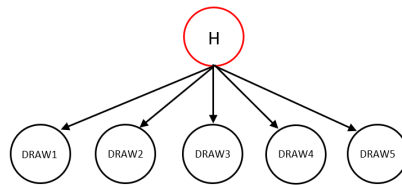


Figure 5.27: Latent Bookbags network (node “H” represents the latent).

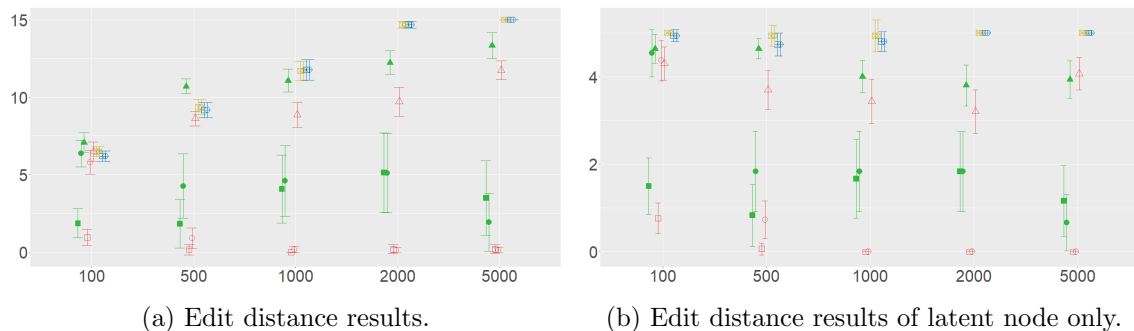


Figure 5.28: Edit distance results of Latent Bookbags cases.

As expected, Initializations 1 and 2 gave both EM-CaMML and SEM a good starting point, while ini3 didn’t even allow these algorithms to converge with larger sample sizes (shown as in Figure 5.28). We also found ini3 led EM-CaMML to produce heavily connected structures when sample size is 5000, and a typical example is shown in Figure 5.29, where all the observed nodes are heavily connected. The latent node H is only connected

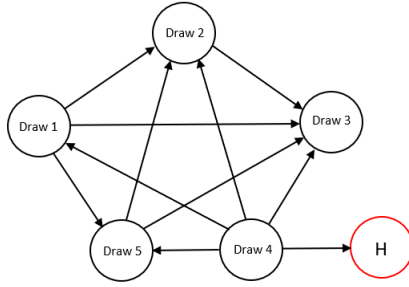


Figure 5.29: An EM-CaMML (ini3) output of Latent Bookbags at sample size 5000 (node “H” represents the latent).

as a child to an observed node, rendering it useless in accounting for interdependencies. That burden has to be carried by a large number of spurious arcs between observed nodes instead, leading to a very poor ED measure. SEM performed similarly using ini3, but with lower densities therefore produced slightly better edit distance results. It appears that the initialization has a greater effect on SEM than EM-CaMML. PC and FCI, in contrast to EM-CaMML and SEM, failed to find any latent node in most cases, resulting in very poor edit distances.

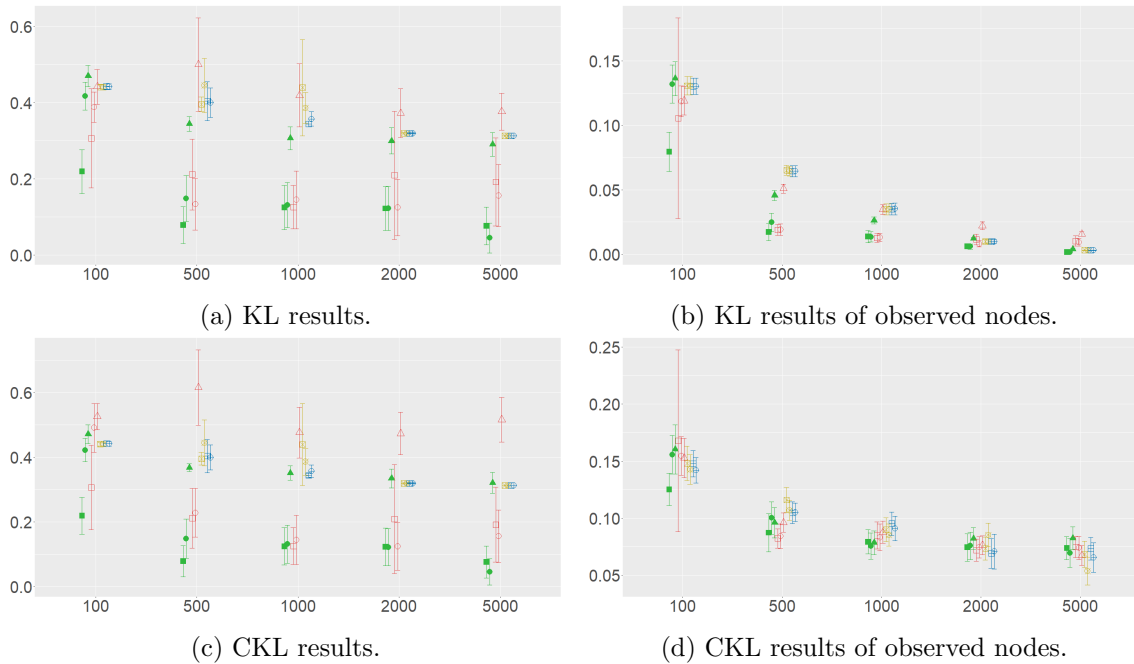


Figure 5.30: KL and CKL results of Latent Bookbags cases.

KL and CKL results (shown in Figure 5.30) are again consistent with ED results, with Initializations 1 and 2 allowing EM-CaMML and SEM to slowly converge on the truth, and ini3 leading to noticeably poorer results. The ini3 results, however, do not simply diverge as the ED results do, suggesting that the large number of spurious arcs are at least being parameterized to fit the real distribution with some effect. PC and FCI results are even worse than EM-CaMML results using ini3. The KL and CKL results for the observed subnetwork show convergence with every method, implying that the KL/CKL difficulties for PC, FCI and the metric learners using ini3 simply have to do with including their measure over the latent node.

The probability of finding a connected latent node (shown in Figure 5.31) with Bookbags (i.e., sensitivity) again illustrates the very poor performance of PC and FCI. SEM

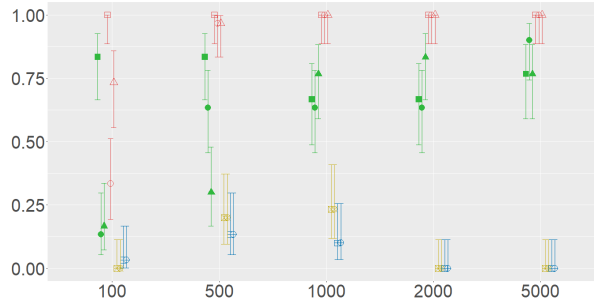


Figure 5.31: Probabilities of latent node connected of Latent Bookbags cases.

tends to slightly outperform EM-CaMML, with the difference diminishing with sample size.

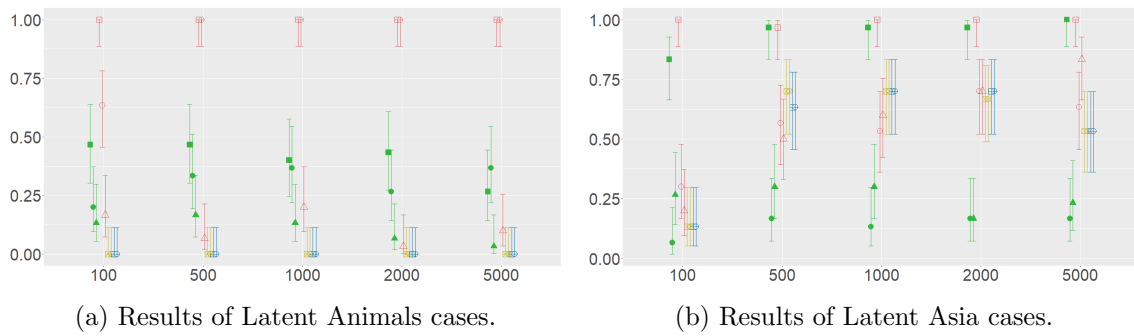


Figure 5.32: Probabilities of latent node connected of Latent Animals and Asia cases.

Overall, for a non-trigger case like Bookbags, we can see that different initializations are important for both EM-CaMML and SEM. PC and FCI appear to have special difficulties with latents as the roots of Naive Bayes structures. For other non-trigger cases, the probability of a connected latent node appear in PC or FCI output is also low (except for the Latent Asia network, but still lower than EM-CaMML and SEM using ini1, see Figure 5.32a and 5.32b in Appendix).

5.2.7 Mendel Genetics

The Mendel Genetics network is used for the experiments done by Mendel who developed the foundations of hereditary genetics, which involved breeding white and red flowered pea plants (Boerlage, 1988). We use this network to assess the tendency of latent discovery algorithms to return false positives; it has no latent node (see Figure 5.33). This test network has five connected nodes and one uncovered v-structure.

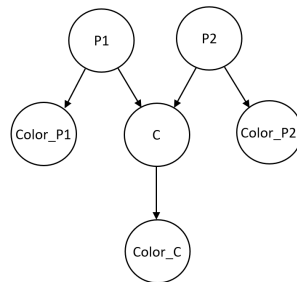


Figure 5.33: No Latent Mendel Genetics network.

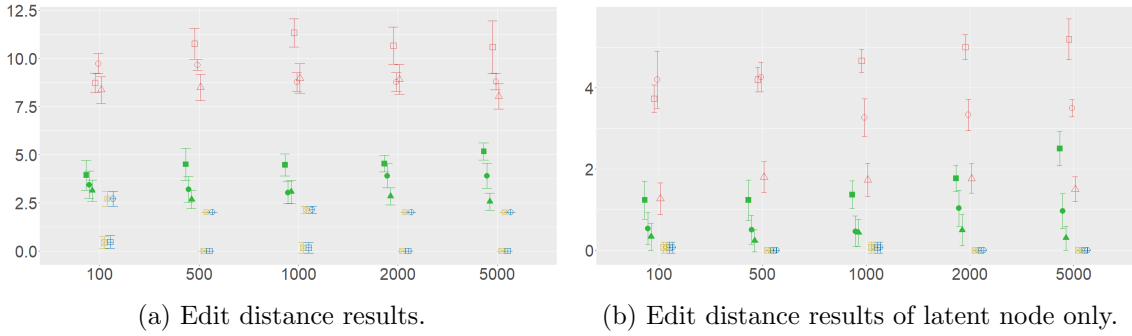


Figure 5.34: Edit distance results of No Latent Mendel Genetics cases.

For the Mendel Genetics net, any latent nodes “discovered” are false positives, which can lead to very poor edit distance scores, since any arc connections to it will add to ED. The edit distance results in Figure 5.34a show that SEM consistently performs poorly, which is explained by its propensity to report false positives, shown in Figure 5.39. This is such a strong tendency, we should probably conclude that SEM is in fact usable only when it is *already known* that there is an active latent variable behind the data! (To be sure, Friedman said as much, when discussing initialization (Friedman et al., 1997), but this requirement significantly diminishes the value of SEM for latent model discovery.)



Figure 5.35: The original structures learned by PC and FCI of No Latent Mendel Genetics cases.



Figure 5.36: The lower and upper bound fixes of PC and FCI outputs of No Latent Mendel Genetics cases.

PC and FCI, on the other hand, do a consistently good job in ED terms. It is interesting that in this case the upper vs lower bound computations are significantly different; the explanation lies in the uncovered v-structure, since occasionally unoriented arcs interact with it, producing distinct ED counts. In Figure 5.35, the left is the original output by PC and the right one is original output by FCI, and the only difference is that the FCI output

has some circles. In this case, based on the rules and constraints mentioned in Section 5.2.2 and given the true structure as in Figure 5.33, the lower and upper bound fixes are same as shown in Figure 5.36. We can see that the difference between the two fixes are the direction of the arcs between $P1$ and $Color_P1$ and between $P2$ and $Color_P2$. New uncovered v-structures at $P1$ and $P2$ are avoided during creation. EM-CaMML performance lies between that of SEM and PC/FCI. Looking at Figure 5.34b suggests that when EM-CaMML finds a false positive it gives it low connectivity.

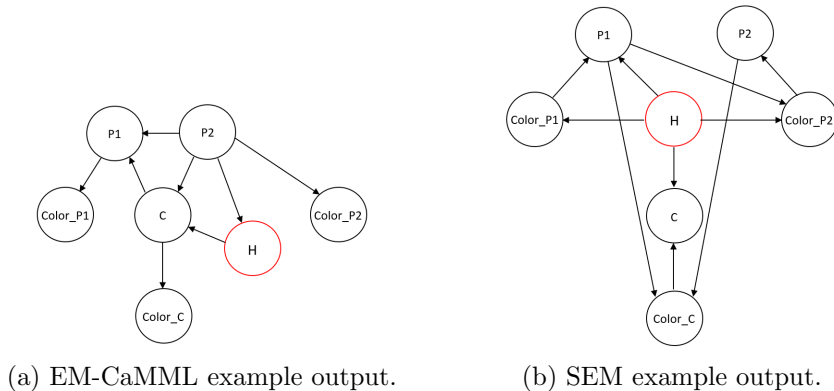


Figure 5.37: Two typical false positives of No Latent Mendel Genetics cases (node “H” represents the latent).

For example, a typical false positive model learned by EM-CaMML using ini1 at sample size of 1000 is Figure 5.37a, where the connected latent node is connected in a chain. Such a chain structure can be marginalized out, meaning the latent has no significant impact, including doing little harm, other than to edit distance. This suggests we should have automatically marginalised out non-functional hidden nodes. By contrast, a typical false positive from SEM with ini1, sample size 1000, is more heavily connected (see Figure 5.37b). In this structure, the latent node has to carry many of the interdependencies between observed variables.

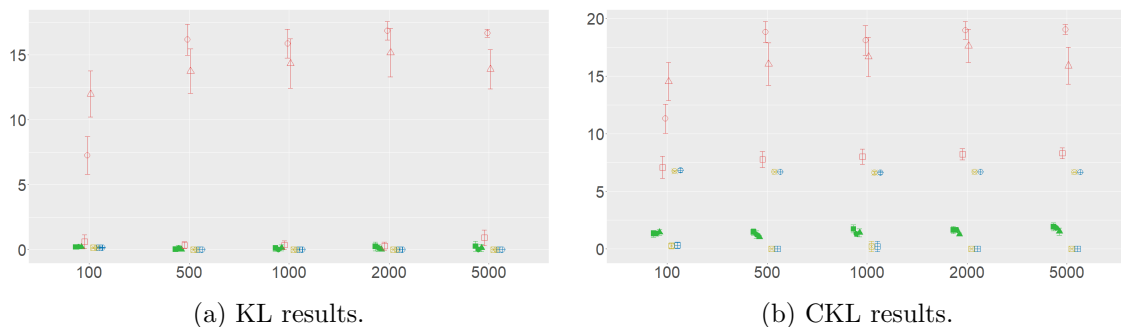


Figure 5.38: KL and CKL results of No Latent Mendel Genetics cases.

The KL and CKL results (shown in Figure 5.38) also show a clear gap between SEM results and others, except for ini1, where the (spurious) latent variable occurs as a common cause to all observed variables, allowing interdependencies, as measured by KL/CKL, to be well represented. Of course, the penalty to SEM ini1 is seen in the ED results. EM-CaMML has a performance similar to PC/FCI in KL. For CKL, EM-CaMML’s performance is about the same as the best-case PC/FCI and clearly better than their worst case.

As we can see from Figure 5.39, the FPR for SEM is very high; it simply assumes the presence of a latent, rather than testing for one. PC/FCI and EM-CaMML (with ini3)

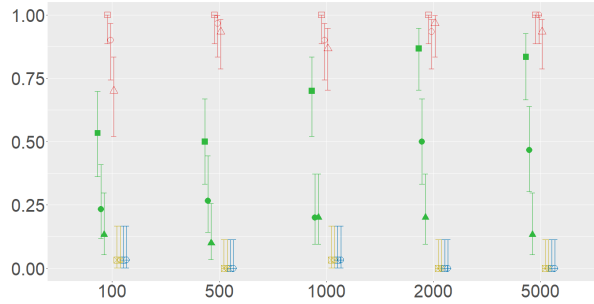


Figure 5.39: Probabilities of latent node connected of No Latent Mendel Genetics cases.

have very few false positives. `ini3` allows EM-CaMML to often start its search with a latent of low, or even no, connectivity; in any case, EM-CaMML can also “unlearn” a connection when the evidence doesn’t support it, although the other initializations show that that ability doesn’t necessarily prevail. EM-CaMML seems to be having real difficulties when given the Factor Analysis type initial structure.

In summary, for fully observed (non-latent) network Mendel Genetics, EM-CaMML has a performance close to best-case PC/FCI and better than PC/FCI’s worst case. SEM proves itself incapable of avoiding false positives.

5.3 Software Design and Implementation

EM-CaMML, which is an extension of original CaMML, has been examined and discussed throughout this thesis. It is not only an extension, but also a new way of enabling CaMML to deal with unobserved causal dependencies which includes a new TOM coster, an enhanced TOM sampling space, a new EM framework and a new version of its UI. In this section, we focus on explaining CaMML and EM-CaMML from a software development perspective (for theoretical details, see Section 3.4.3).

The initial development work on CaMML started with Wallace et al. in 1996. This version was able to learn causal models with continuous data by applying a greedy search. Later, Neil and Korb developed a new version that applied a genetic search for the purpose of learning larger networks. The first version of CaMML to use MCMC sampling dealt with continuous variables and was developed by Wallace and Korb in 1999. It showed similar performance to the genetic version in terms of KL divergence and edit distance performance. After that, a new version of CaMML was developed by Rodney O’Donnell in 2010 during his PhD study which applies MCMC sampling to deal with large scale networks using discrete variables. The earliest versions of CaMML were implemented in C, but switched to Java with O’Donnell’s version. This was built on top of the CDMS platform (Comley et al., 2003; Allison, 2005), which has many benefits such as versatile representation of local structure and prior information, features that were absent in previous versions.

However, no version until now had considered latent factors and were unable to perform MCMC with latent variables. So we developed EM-CaMML, as a milestone update to O’Donnell’s version. All major extensions and enhancements introduced to CaMML are summarised as follows:

- An implementation of EM algorithm in the `camml.core`, which learns with discrete variables. It includes a framework that allows running MCMC sampling between the `E_step` and `M_step`. Instead of using the `Value.Vector` as the default data container, the `WeightedVector` (a function of `cdms.core.VectorFN`) is used as it is able to contain

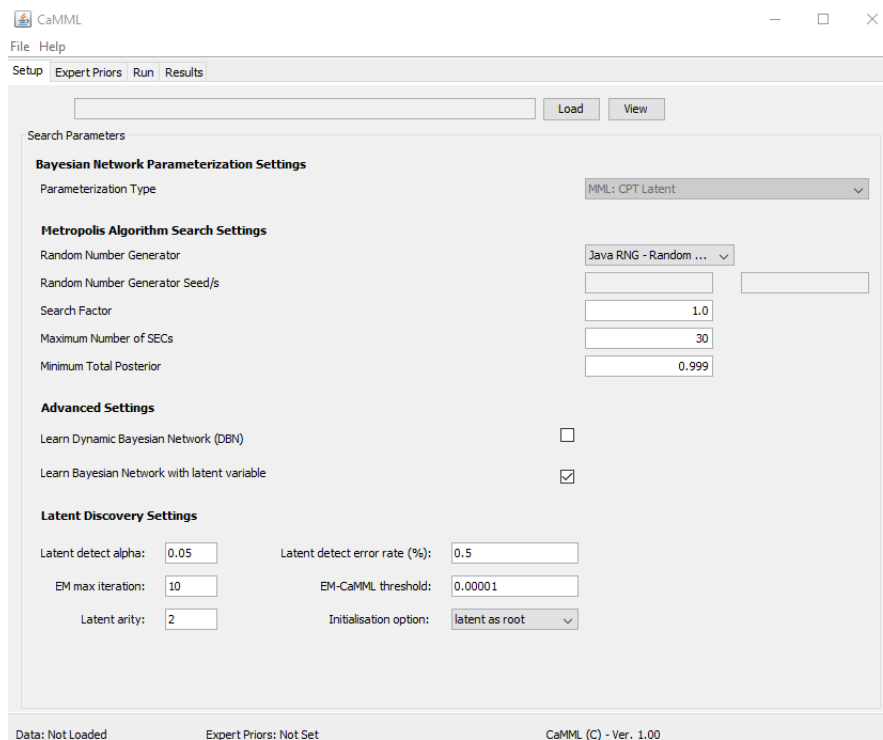


Figure 5.40: The UI design of EM-CaMML program.

the soft counts. The EM implementation allows for a different initial structure, either a DAG or TOM, and updates the structure using MCMC.

- A new package “camml.core.latentDetect” which contains most of our work on finding triggers, a latent node detect function called by EM-CaMML, and also some useful functions used in searching triggers (e.g., enumerating all possible DAGs, generating dependency matrices for a given DAG). It is a relatively independent package and provides a wrapper function called “LatentDetect” which is a wrapper of the EM-CaMML’s preprocessing step.
- An implementation of a new adaptive coder learner “LatentAdaptiveCPTLearner” (located in function “AdaptiveCodeLearner”) which contains our new MML metric for calculating the data and parameter cost of a TOM with latent nodes (see Section 5.1.1).
- A new wrapper function “runSearchLatent” which includes all the main modules of EM-CaMML. It is the implementation of the process shown as in Figure 5.8. It calls the trigger detection function and runs the MCMC with EM integration.
- A new version of the UI (shown as in Figure 5.40) that gives users more options for running the program compared to the original version. When the checkbox “Learn Bayesian Network with latent variable” is ticked, the “MML:CPT Latent” (which calls the “LatentAdaptiveCPTLearner” function) will be automatically selected and users are allowed to specify the parameters of running EM-CaMML (e.g., Latent arity, EM-CaMML threshold).

Of course, due to time constraints, there are still things that are in need of improvement. For example, for simplicity reasons we reset the cache of sufficient count records and MML costs of local structures after MCMC finishes in each EM iteration, which could be reused in the next iteration. In terms of the EM-CaMML workflow, it can be made more

flexible by providing the option to turn off the latent detect function and directly add in a latent node in a given initial structure (i.e., go to Path 2 without checking whether a trigger is matched as shown in Figure 5.8). This is because people may be interested in learning a latent node model with different techniques. Moreover, it would be better if users could specify the trigger detect threshold and arc prior from the UI. Currently such values are hard coded using the results of Section 5.2.3; users have to make the change in code. Nevertheless, we are still making occasional improvements, and the EM-CaMML program is publicly available on GitHub.¹² We hope EM-CaMML could be beneficial to the public and any contributions are welcome.

5.4 Summary

EM-CaMML is a metric based algorithm which incorporates a trigger detection function and produces fully parameterized models. There has not been much attention paid to detecting latent nodes in causal discovery; EM-CaMML fills this gap by explicitly discovering latent variables and learning their direct connections to the rest of the network, utilizing dependency patterns. In terms of how best to integrate EM with MCMC sampling (EM inside and outside), we hope to use more complex test networks in the future to compare them. However, EM outside will significantly increase execution time, so some additional enhancements (e.g., caching the EM statistics for unique TOM structures) will be needed to make the comparison practical. Of course in the future, we can also try other learning metrics (e.g., MDL (Lam and Bacchus, 1994), Bayesian learning score (Cooper and Herskovits, 1992; Heckerman, 2008)) as alternative metric functions in EM-CaMML. Based on our chosen parameter settings, EM-CaMML is able to produce promising results in terms of the different metrics used in the experiments. In order to provide comprehensive testing, different initial structures have been examined and we have done some work on fixing and interpreting results produced by PC and FCI algorithms when rendered in the language of DAGs.

There are many experimental results which have not been shown in this thesis due to the limited available space, but we have made as much as possible available online for readers' interests.¹³ In general, if no trigger is detected, then the choice of initialization structure has a significant impact on the resulting structure, and this is also true for SEM. Especially in the Latent Bookbags cases, the ini1 and ini2 have significant advantages in learning the true structure. Additionally, the results of Big-W cases and Covered Big-W cases are very similar, thus we could potentially just search for Big-W and let the algorithms learn the additional arc for Covered Big-W. PC and FCI perform very well when the latent node is shown as a common cause and has no parent. However, due to the nature of the PC and FCI, they are not so useful in revealing how to connect the latent node when in non-trigger-like cases. This happened across all four Latent (non-trigger) network test cases. Moreover, PC and FCI produce partially oriented graphs, occurring frequently in our experiments, so those using these methods have to deal with this issue using prior knowledge and following our methods for mending the structure. Of course, users still need to apply parameter learning algorithms to parameterize the models generated by PC and FCI. While SEM suffers from different problems, it generally produces more connected structures compared to others, especially for the latent node which will cause more false positive results. We could use optimized parameters for each test network for SEM to get better results, but this is a non-trivial optimisation problem. These disadvantages are either absent or much less severe in the case EM-CaMML, making it a notable and well-rounded solution for latent node discovery and modeling.

¹²The source code is available at: <https://github.com/zxh298/EM-CaMML>

¹³All figures are available at: <https://sourceforge.net/projects/trigger-test-full-results/files/>

We have made EM-CaMML a open source software that is explicitly designed to discover and learn with latent variables. It calls each learning module automatically and smoothly without requiring any prior knowledge about the latent node. In terms of the actual execution time for the above test cases, our Java program EM-CaMML runs much faster than the SEM implementation we used in this chapter. Of course, PC and FCI produce results faster still due to their approach, however, they do not learn fully parameterized models and struggle with latent variables that are poorly matched by the typical dependency patterns that they seek.

Another point is that most of the test networks used in this chapter are only the subnets of larger networks. This is due to the size limitations of our trigger program as well as the high complexity in learning latent variables in larger networks. One possible (albeit exponentially complex) solution is to loop through possible node subsets and check if any of them match a trigger, and use CaMML to learn the structure of the rest. Of course, we can learn a fully observed model and compare it with the model learned with latent nodes in terms of the predictive accuracy or posterior given observations. Alternatively, we can use a heuristic function to split nodes into different subsets (e.g., the MBs) with minimal overlap, and then use EM-CaMML to learn first the regional structures centered on each subset (using triggers) and then their interrelations. A preliminary study of this method is shown in next chapter.

Chapter 6

EM-CaMML Extensions

In this chapter, we discuss two extensions to EM-CaMML in detail. It begins with the introduction of the definition of a partial trigger and examines how different algorithms discover the latent node in partial triggers. Section 6.1 includes details of the experimental work regarding the partial trigger structures. Networks with different arc strengths are used as the test networks, and we expect this will tell us how the latent node is being discovered based on how much contribution it makes to explaining the conditional dependencies in data. Next, a preliminary study on discovering multiple latent variables within different partitions of a large network is presented in Section 6.2. We use a recent Markov Blanket (MB) discovery algorithm to generate partitions and apply EM-CaMML on each of them. The purpose of this study is to show the feasibility of this approach to scaling up EM-CaMML and to indicate how different MB local structures can be merged into a complete network by CaMML.

6.1 Learning Partial Triggers

The following sections will begin with the definition of a partial trigger, followed by the methodology behind the simulation of test datasets and details of the experiments. Then we will show some analysis on the experiment results. Finally, a brief summary of the findings will be shown at the end of this section.

6.1.1 Definition of Partial Trigger

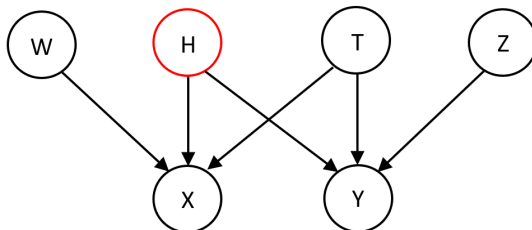


Figure 6.1: An example partial trigger (node “H” represents the latent).

One interesting question is how well causal learners may perform when confronted with partial triggers, where the trigger dependencies may be explained by either a latent node or an observed node. For simplicity, here we only consider the partial trigger in Big-W structure with 5 observed nodes. In Figure 6.1 the network contains one latent node H and five observed nodes W, X, Y, Z and T , and the dependencies between the observed nodes may be fully explained using H without T , T without H , or using both. Using

Figure 6.1 as the true structure, if the arcs $T \rightarrow X$ and $T \rightarrow Y$ are weak, then there needs to be another node playing a role that supports the dependencies between W , X , Y and Z . This is what we call a “partial trigger”. In order to understand how well such partial triggers are discovered by different algorithms, we constructed four artificial test networks with varying arc strengths using mutual information function (I) between variables (see details in Section 4.4.1).

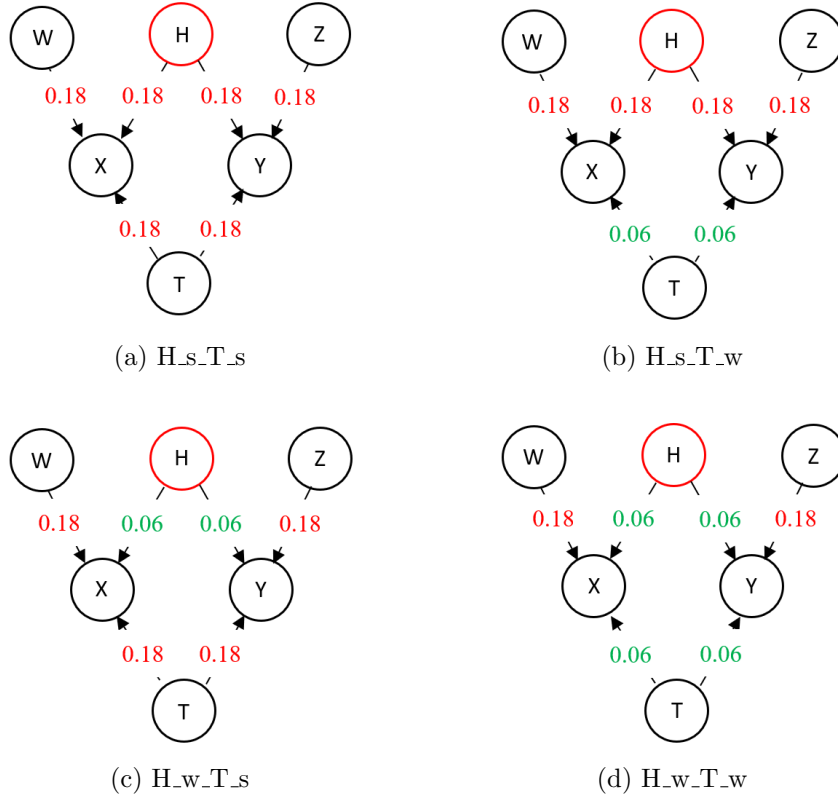


Figure 6.2: Partial trigger test networks (node “H” represents the latent, and the number represents the arc strength of each arc).

Since we want to analyse how T affects trigger discovery, we want to have test cases where T and H have different connection strengths to X and Y in Figure 6.1. We use an I of 0.18 (measured in bits) to represent a strong connection and 0.06 for a weak connection (We also explored 0.12 as medium, but no interesting results were found, and anything smaller than 0.06 was too low), designating these “s” (strong) and “w” (weak) respectively. To simplify the experiment, we used the four different test networks of Figure 6.2, making all variables binary. For example, “H_sT_w” means H is strongly connected and T weakly connected.

Here we follow the same experimental process presented in Chapter 5 using the same optimized parameters for EM-CaMML from Section 5.2.3. We compare EM-CaMML with other benchmark algorithms using 30 synthetic datasets which are sampled from a set of sample sizes $\{100, 500, 1000, 2000, 5000\}$ using forward sampling in Netica, whilst deleting the data for H as it was the target latent node.¹ Thus we have $5 \times 30 = 150$ datasets in total for each network. In order to demonstrate each graph more clearly, we removed the legends from the result graphs, but we refer the reader to the keys in Table 5.4 to identify each learner.

¹The test data is available at: <https://sourceforge.net/projects/partial-trigger-data>

6.1.2 Experiment Results and Discussion

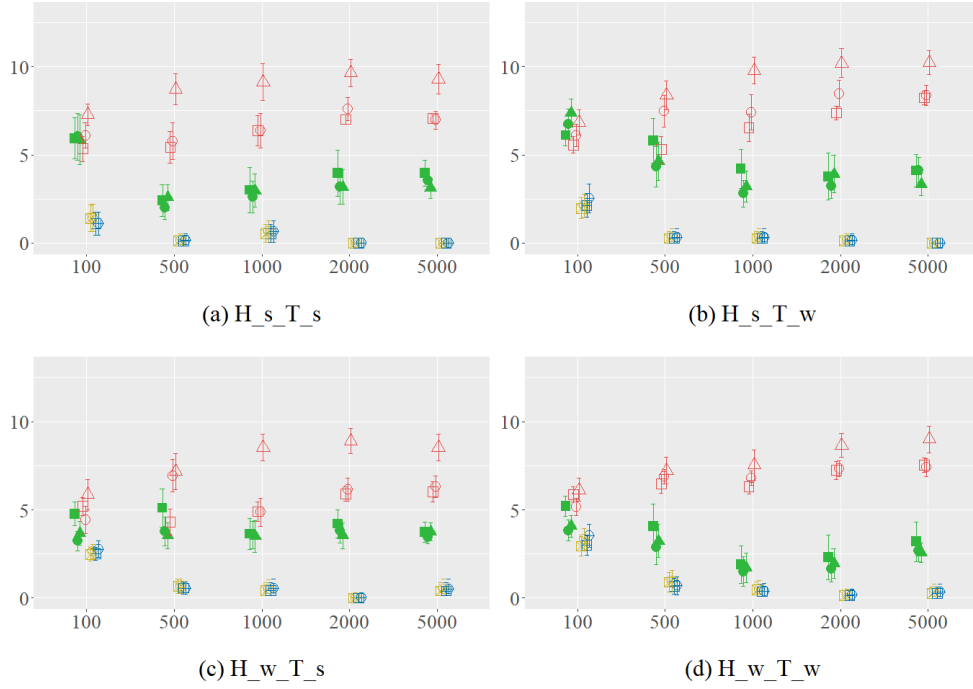


Figure 6.3: Edit distance results of learning partial trigger.

Figure 6.3 shows the edit distance results for each test network respectively. Similar to the Alarm network, PC/FCI perform the best in edit distance terms, followed by EM-CaMML and SEM showing the worst performance. Unfortunately, EM-CaMML also shows little ability to improve with sample size here. Also, ini1 does not appear to help SEM. Similarly to the Trigger test cases, SEM produced many heavily connected networks that didn't deal with the latent node properly. We can see for PC and FCI, when node H is strongly connected (i.e., $H_s.T_s$ and $H_s.T_w$), the edit distance results are generally better than when node H is weakly connected. This is understandable as the latent node appears as a common cause and a strong connection will make PC and FCI more likely to produce a bi-directional arc at the right place.

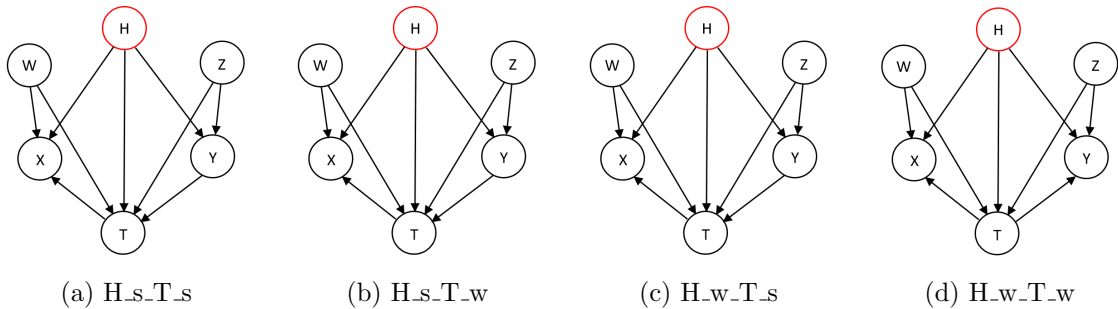


Figure 6.4: EM-CaMML learned model using ini1 with sample size 5000 (node “H” represents the latent).

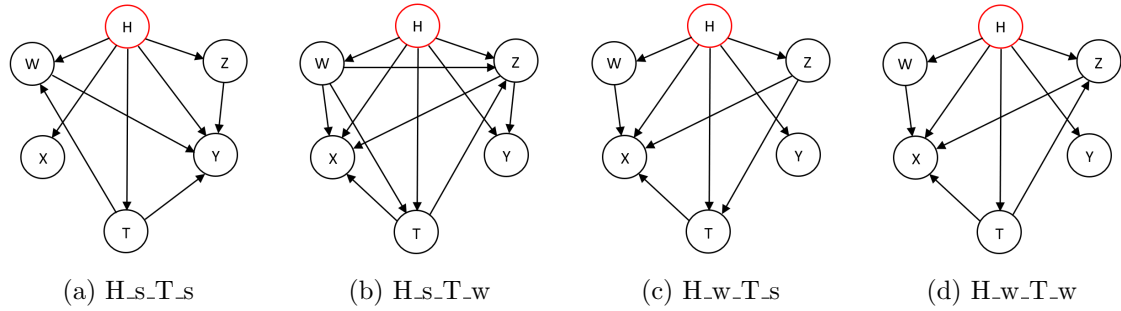


Figure 6.5: SEM learned models using ini1 with sample size 5000 (node “H” represents the latent).

Figure 6.4 shows a typical set of learned models from EM-CaMML using ini1 and sample size 5000. We can see that the node X and Y are correctly identified as the children of latent node H . However, the dependencies between node T and others are not well captured, with T being directly connected to all other nodes. The typical examples seen in Figure 6.5 shows that SEM produces worse mistakes. Thus, W and X are marginally independent, however SEM insists on keeping them as children of H regardless. We also see similar results in Section 5.2 that the latent node in SEM is always highly connected.

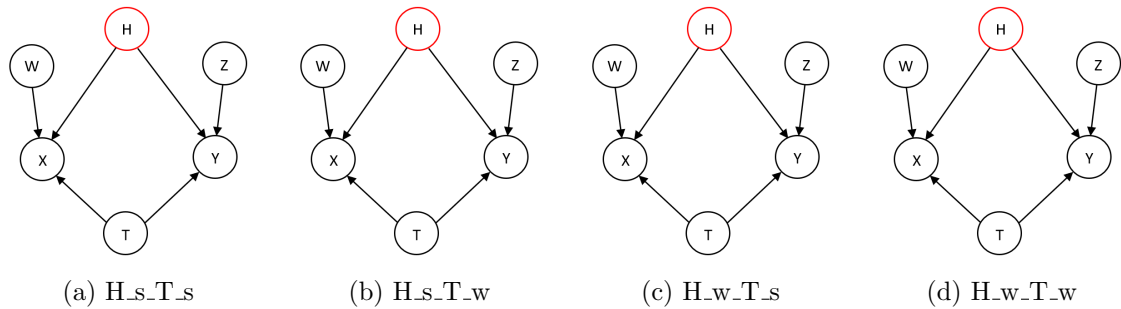


Figure 6.6: PC/FCI learned models with sample size 5000 (node “H” represents the latent).

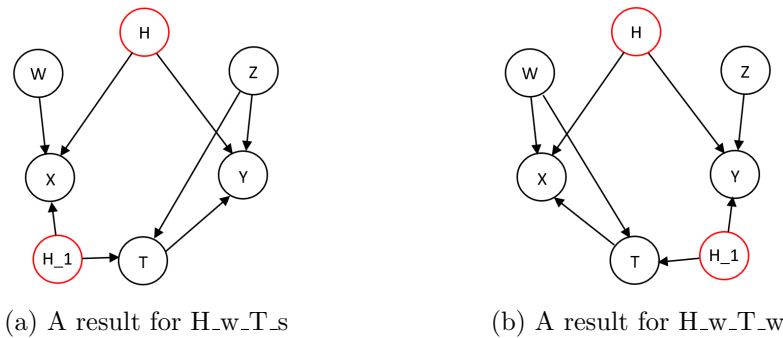


Figure 6.7: An example of PC/FCI output with a spurious latent at sample size 5000 (node “H” represents the latent).

PC/FCI produced very good results with sample size of 5000, with the modal result simply being the correct generating model (Figure 6.6). As there was not much variance among PC and FCI original output structures, the lower and upper bound fix are generally overlapping for most cases. There were only a few cases where PC/FCI made mistakes. For example, for test networks H_w-T_s and H_w-T_w there were a few learned models having more than one discovered latent node (shown as “H” and “H₁” in Figure 6.7). The redundant node H_1 is shown as a common cause of two observed nodes and could

in both cases be replaced by a single directed arc between them. Such structure indicates the contribution of node T is not correctly recognized as node H_1 explains part of the dependencies which should be satisfied by node T .

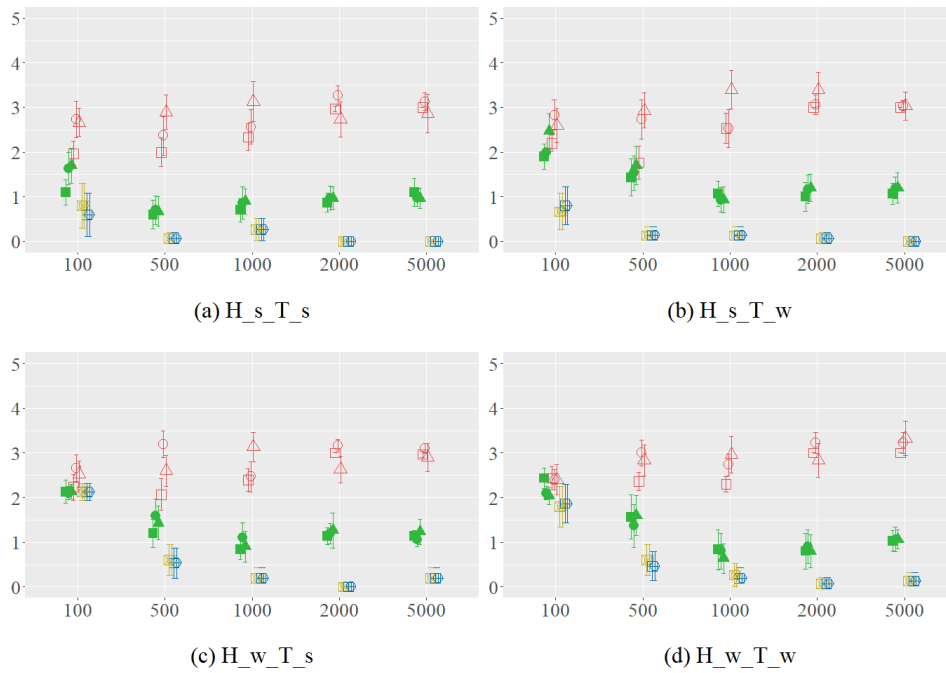


Figure 6.8: Edit distance results of learning partial trigger (only H).

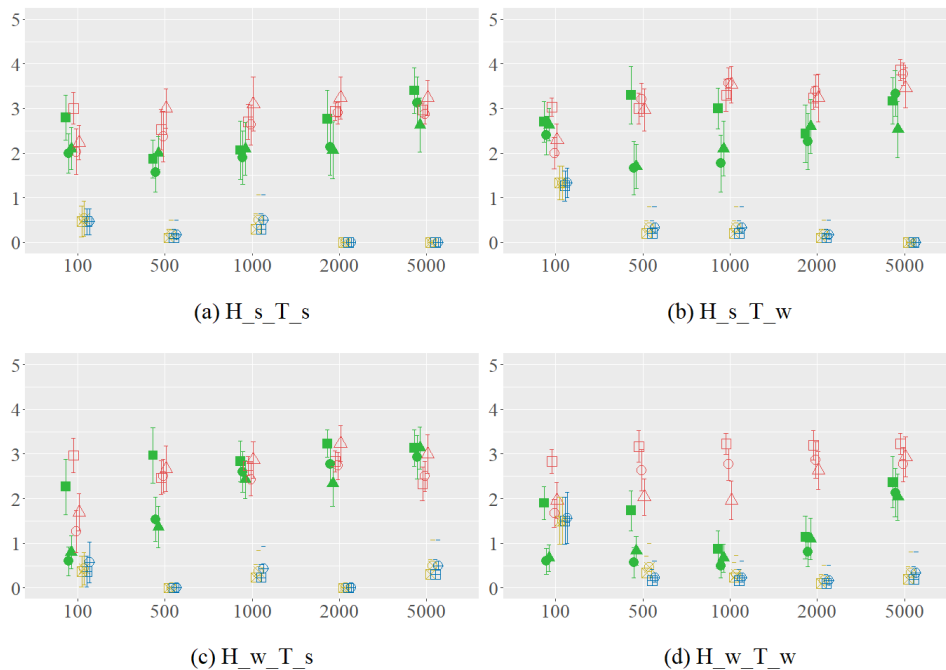


Figure 6.9: Edit distance results of learning partial trigger (only T).

In the edit distance results restricted to nodes H and T in Figures 6.8 and 6.9, we see the same general pattern as with the network as a whole, with PC/FCI clearly outperforming the other two algorithms. EM-CaMML often joins SEM in connecting T incorrectly

to the rest of the network, but clearly does better with H . EM-CaMML cannot match PC/FCI with H , however, since there is a connection between node H and T in most learned models.

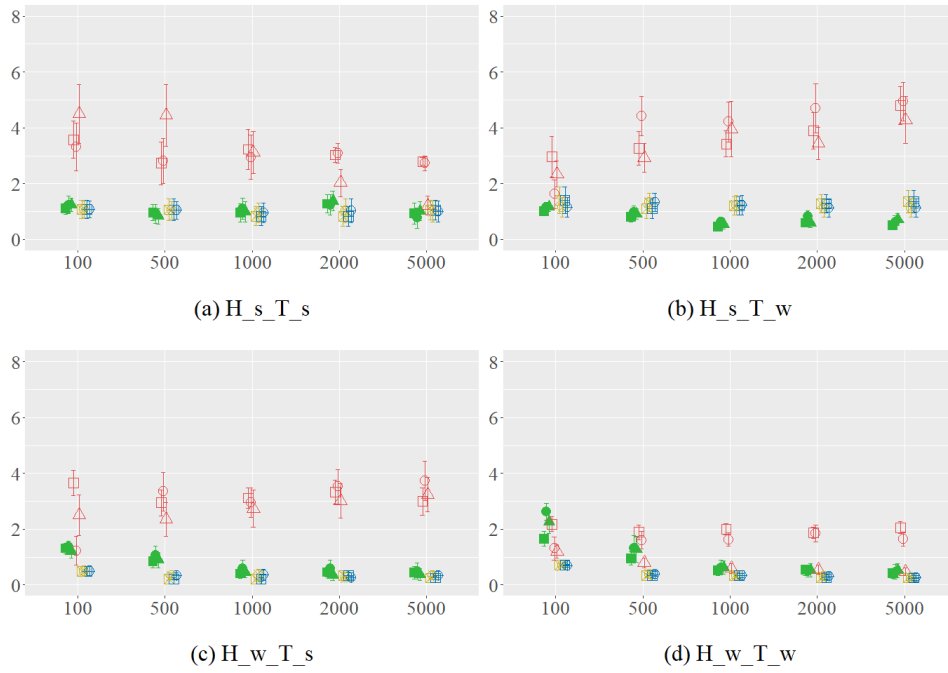


Figure 6.10: KL results of learning partial trigger.

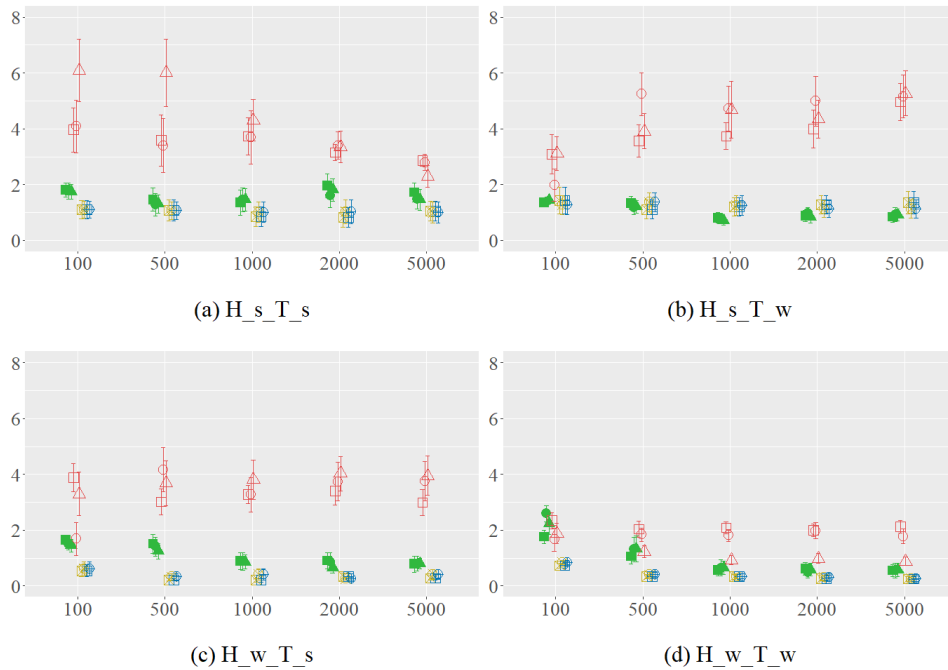


Figure 6.11: CKL results of learning partial trigger.

With KL/CKL measurements (Figures 6.10 and 6.11), there is not much to choose between EM-CaMML and PC/FCI, except perhaps with H_s_T_w, when larger samples put EM-CaMML statistically significantly ahead in KL terms. SEM performs uniformly

poorly, with the exception of $H_w T_w$, when the third Initialization allows it to perform better than otherwise.

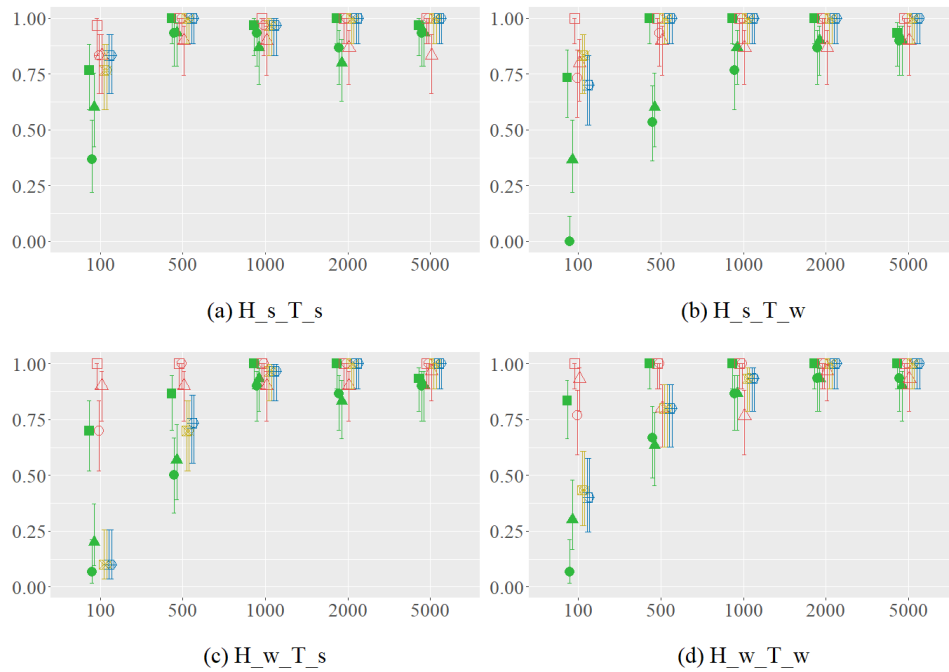


Figure 6.12: Probability of finding a latent node in a partial trigger.

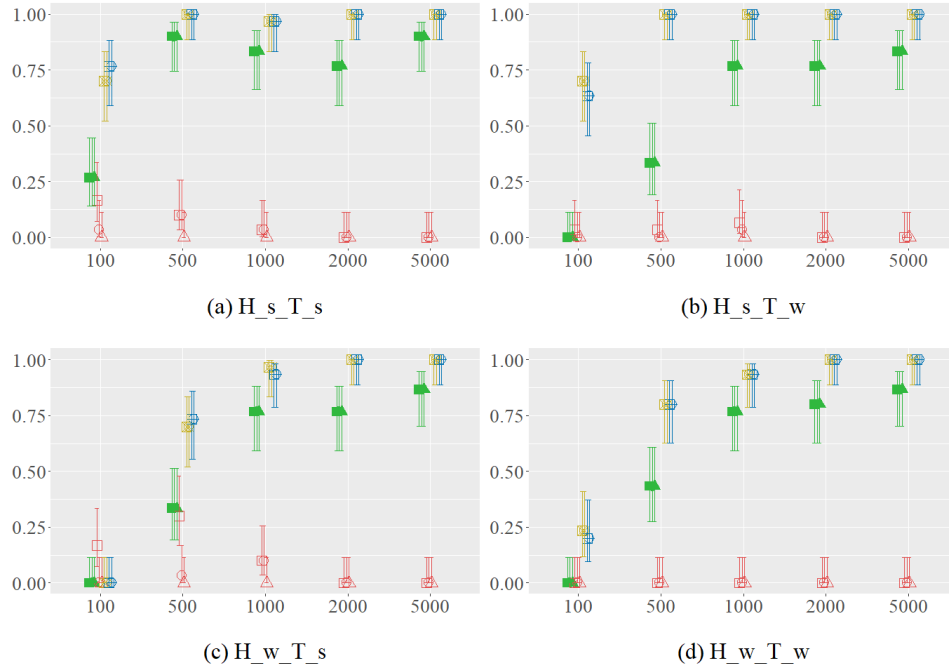


Figure 6.13: Probability of finding a trigger subnet in a partial trigger.

The TPR (true positive rate) for discovering latent nodes (Figure 6.12) shows performance converging on 1 for all methods. Looking at the probability of learning the trigger subnet (which omits T) in Figure 6.13, SEM does uniformly very poorly, while PC/FCI performs the best followed closely by EM-CaMML.

In summary, PC and FCI produce the best structures in edit distance results, and EM-CaMML has a similar performance to PC and FCI for KL/CKL results. For learning the node H , the effect of different arc strengths is not obvious. We can see from Figure 6.8 that if both node H and T are strongly connected, the edit distance for node H is lowest but this is only true using the smallest sample size. When the sample increases, this effect vanishes completely. The main problem for EM-CaMML here is the poor performance in connecting node T , as the edit distance for only T is not getting better even as the sample size increases. Similar to the findings of the Big-W test in Section 5.2.4, when the latent node appears as a common cause, SEM failed to detect the triggers and the latent node was heavily connected compared to other algorithms. In the future, we can investigate different arc density priors for EM-CaMML, particularly to see if less dense networks could help to reduce some wrong arcs, which could produce better results especially in the case shown in Figure 6.4. Of course, such less dense networks could also drop some correct arcs as well. Additionally, we hope to assess each learner using real world datasets with more variable arc strengths, and learn more about how to discover and utilize latent nodes to complement observed nodes in explaining the dependencies.

6.2 Discovering Multiple Latent Variables

In this section, we consider extending our initial study to discovering multiple latent variables in a network with 20 nodes. Our aim is to scale up EM-CaMML with Markov Blanket (MB) partitions and demonstrate how the resulting structure for the partitions can be merged using EM-CaMML. Note that this is a very preliminary study. Some steps are not fully implemented and had to be carried out manually in this research. However, we show EM-CaMML has good potential for discovering multiple latent nodes in large scale networks, and we have developed some ideas on how to properly implement a multi-latent extension of EM-CaMML. We start with some background knowledge about MBs and explain the problem of scaling up to larger networks. Then an experiment is presented showing our preliminary work in extending EM-CaMML to discover multiple latent variables. Finally, we present the results and give a brief summary of findings.

6.2.1 Markov Blanket Discovery

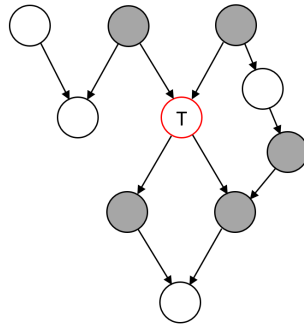


Figure 6.14: An example of the MB of Target node T .

The MB (Pearl, 1988) of a target variable is the minimal set of variables conditioned on, for which all remaining variables are independent of the target node. This property implies that knowledge of variables in the MB is sufficient to determine the probability distribution of the target, with all other information for any variables beyond the MB being superfluous. Under the faithfulness assumption for a BN, the MB of a target variable T is identical to its direct causes and effects, together with the direct causes of direct effects

(i.e., parents of children) of T . Thus MBs can be used to reduce the number of variables required to discover the true direct causes of T (Tsamardinos et al., 2003). To put it simply, an MB of target node T contains parents and children of T and the children’s other parents. For example, in Figure 6.14, the grey shaded nodes form an MB for the target node T .

However, knowing the MB of a target node does not tell us how the variables are connected in the MB, and the definition of MB just tells us the optimal subset of nodes for predictions regarding the target. It is natural to think about using causal learners (e.g., EM-CaMML) to learn a causal model of each MB and even discovering latent variables within each MB. As we have seen, EM-CaMML can only discover latent variables in small networks due to the super exponential time complexity required for searching triggers beyond 6 observable variables (see details in Section 4.2).

There are many studies on modeling with MB. For using MB for feature selection, notable work includes (Koller and Sahami, 1996; Cooper et al., 1997; Strobl and Visweswaran, 2015). Examples of research into structure learning for each MB are (Nägele et al., 2007; Niinimäki and Parviainen, 2012; Ramsey et al., 2017; Gao and Wei, 2018). For this EM-CaMML extension, we use the MBMML algorithm (Li, 2020) proposed by Yang Li in his PhD thesis which applies MML to discovering MB. This algorithm uses MML, which makes it very compatible with EM-CaMML, and it displays competitive performance in many scenarios. This makes it a promising candidate for scaling up EM-CaMML in our future work.

Algorithm 3 MB discovery using MBMML+CPT

```

1: Let  $D$  be a data set of a set of variable  $\mathcal{V}$ ;
2: Let  $\bar{Z}$  be an empty set;
3: Let  $Z$  be an empty set;
4: for each  $v$  in  $\mathcal{V}$  do
5:   Let  $v$  be the target node  $T$ ;
6:    $\mathcal{V}' = \mathcal{V} \setminus T$ ;
7:    $\phi_T(S)$  be the CPT model of  $T$  with a parent set  $S$ ;
8:   Let  $L = I(\phi_T(\emptyset), D_T)$  be the message length of  $\phi_T(\emptyset)$  and  $D_T$ , where  $D_T$  is the data over  $T$ ;
9:   while  $\mathcal{V}' \neq \emptyset$  do
10:     $X_k = \arg \min_{X_i} I(\phi_T(Z \cup X_i), D_T), \forall X_i \in \mathcal{V}'$ ;
11:     $L' = I(\phi_T(Z \cup X_i), D_T)$ 
12:    if  $L' < L$  then
13:       $Z = Z \cup X_k$ ;
14:       $\mathcal{V}' = \mathcal{V}' \setminus X_k$ ;
15:       $L = L'$ 
16:    end if
17:  end while
18:  Add  $Z$  to  $\bar{Z}$ 
19: end for
20: Output  $\bar{Z}$ ;
```

The current MBMML algorithm has three different implementations: MBMML for CPT (MBMML+CPT), MBMML for Naive Bayes Models (MBMML+NB), and MBMML for Markov Blanket Polytree Models (MBMML+MBP). The first implementation assumes all variables in the MB are parents of the target, which has maximal representational power but requires the most data to parameterize accurately. The MBMML+NB assumes all MB variables are independent given the target which minimizes the number of required parameters but loses any dependencies between them. MBMML+MBP provides a compromise between these two extremes by applying an ensembling method that samples as many local polytrees as possible. It encodes the MB variables by learning a variety of structures and outputs a weighted average MML cost across all samples (Li, 2020). For simplicity and to save time, MBMML+CPT is the only MB partition method

for EM-CaMML that we tried. This method (shown as Algorithm 3) applies a greedy search that starts with an empty MB and iteratively adds the best candidate node for improving the total message length of the model, and the result is a set of variable sets that represent the best MB for every node as the target. After getting the result, we need to select the best MBs that cover all the nodes with minimum overlap. To achieve this, we apply a heuristic function (we name it “MB_pick_greedy”) which picks the MB that has the largest number of non-overlapping nodes with the least number of overlapping nodes (including the target), against the MBs that have been picked. To be more specific, this function is represented as:

$$MB_pick_greedy = n_{overlap} - n_{non_overlap} \quad (6.1)$$

where $n_{overlap}$ and $n_{non_overlap}$ represent the number of overlapping and non-overlapping nodes respectively, and the order (from the maximum to minimum) of each MB being selected is based on its score for this function. If there are multiple available choices which are equal in terms of this heuristic function, we will pick one of them randomly. So the complete multiple latent variables learning method proceeds as follows:

1. Apply MBMML+CPT to a given dataset and save the resulting MBs.
2. Apply MB_pick_greedy function to select the best MBs that covers all nodes.
3. Run EM-CaMML on each MB and save the resulting models and necessary sufficient statistics (i.e., soft counts for each discovered latent node).
4. Create structural priors (using hard constraints, see details in Section 5.2.3) for each model from Step 3 and combine all such priors into a complete prior.
5. Run EM-CaMML with the sufficient statistics from Step 3 and the complete prior from Step 4 and save the output model.

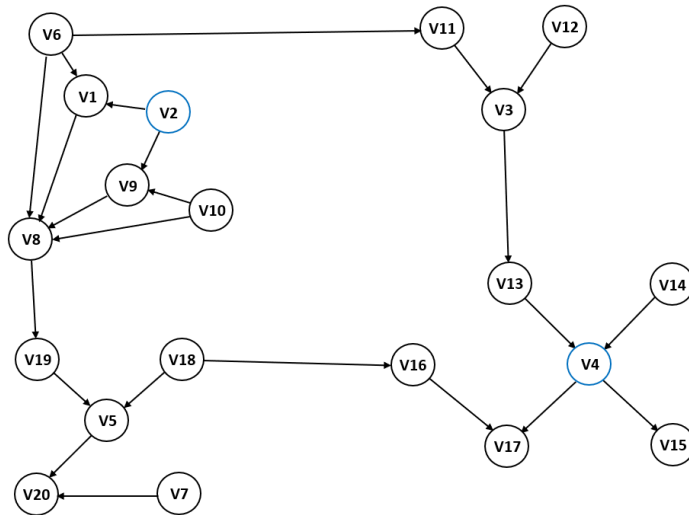


Figure 6.15: The true network used for testing the discovery of multiple latent variables (nodes “V2” and “V4” are latent).

For example, Figure 6.15 shows a model of twenty discrete nodes V_1, V_2, \dots, V_{20} . This network was generated by hand and serves as our true network for testing the method. In this network, all nodes are binary and the CPT parameters for each node are randomly sampled from a symmetric Dirichlet distribution (Gelman et al., 1995) with concentration

parameter α equal to 0.5, which gives relatively more biased CPT values than sampling from a uniform distribution (i.e., where $\alpha = 1.0$). Such CPT values will increase the strength of the network’s arcs and help to reduce the chance of mistakes. Again, this setting is only for testing purposes. From the network, we can see there are four obvious partitions (shown as in Table 6.1) and there are four arcs that link these four partitions: $V6 \rightarrow V11$, $V3 \rightarrow V13$, $V18 \rightarrow V16$ and $V8 \rightarrow V19$. Each partition corresponds to an MB with a target node as follows:

MB	MB variables	Target variable
MB1-true	V2, V6, V8, V9, V10	V1
MB12-true	V3, V11	V12
MB4-true	V13, V14, V15, V16, V17	V4
MB5-true	V7, V18, V19, V20	V5

Table 6.1: Example MBs of the network shown in Figure 6.15.

For our test, we converted node $V2$ and $V4$ into latent nodes (by deleting the corresponding data). As a consequence, the MB variables for target $V1$ will reduce to $\{V6, V8, V9, V10\}$, and we expect the nodes in MB4-true will remain the same, with either $V15$ or $V17$ as the target (this is not true if $V4$ is known). The purpose of including MB4-true in the true network is to see if learning the latent node can help simplify the local structure — otherwise, $V13, V14, V15$ and $V17$ will be connected in a network with high arc density without $V4$. So here we expect the MBMML+CPT algorithm will still put $V13, V14, V15, V17$ into a partition using either $V15$ or $V17$ as the target, while providing EM-CaMML with the opportunity of discovering a simpler representation.

6.2.2 Experiment Results and Discussion

MB	MB variables	Target variable
MB-1	V6, V8, V9, V10	V1
MB-3	V11, V12, V13	V3
MB-5	V7, V18, V19, V20	V5
MB-6	V1, V8, V9, V10, V11	V6
MB-7	V5, V20	V7
MB-8	V1, V6, V9, V10, V19	V8
MB-9	V1, V6, V8, V10	V9
MB-10	V1, V6, V8, V9	V10
MB-11	V3, V6, V12	V11
MB-12	V3, V11	V12
MB-13	V3, V15	V13
MB-14	V15	V14
MB-15	V13, V14	V15
MB-16	V13, V14, V15, V17, V18	V16
MB-17	V13, V14, V15, V16	V17
MB-18	V5, V16, V19	V18
MB-19	V5, V8, V18	V19
MB-20	V5, V7	V20

Table 6.2: MBMML+CPT result.

Here we simulated a dataset of 5000 rows using the test network mentioned in the previous section and deleted the data of V_2 and V_4 , so as to make them latent. Given this dataset, MBMML+CPT produced one MB per node, shown as in Table 6.2.

Iteration	Nodes remained	Nodes covered	Selected MBs	Candidate MBs
0	$V_1, V_3, V_5, V_6, V_7, V_8, V_9, V_{10}, V_{11}, V_{12}, V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, V_{18}, V_{19}, V_{20}$			MB-6 (5), MB-8 (5), MB-16 (5), MB-1 (4), MB-5 (4), MB-9 (4), MB-10 (4), MB-17 (4), MB-3 (3), MB-11 (3), MB-18 (3), MB-19 (3), MB-7 (2), MB-12 (2), MB-13 (2), MB-15 (2), MB-20 (2), MB-14 (1)
1	$V_3, V_5, V_7, V_8, V_{11}, V_{12}, V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, V_{18}, V_{20}$	$V_1, V_6, V_8, V_9, V_{10}, V_{19}$	MB-8	MB-16 (5), MB-17 (4), MB-3 (3), MB-5 (2), MB-7 (2), MB-12 (2), MB-13 (2), MB-15 (2), MB-20 (2), MB-11 (1), MB-14 (1), MB-18 (1), MB-19 (1), MB-6 (-3), MB-1 (-4), MB-9 (-4), MB-10 (-4)
2	$V_3, V_5, V_7, V_{11}, V_{12}, V_{20}$	$V_1, V_6, V_8, V_9, V_{10}, V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, V_{18}, V_{19}$	MB-8, MB-16	MB-7 (2), MB-12 (2), MB-20 (2), MB-3 (1), MB-11 (1), MB-5 (0), MB-13 (0), MB-14 (-1), MB-18 (-1), MB-19 (-1), MB-15 (-2), MB-6 (-3), MB-1 (-4), MB-9 (-4), MB-10 (-4), MB-17 (-4)
3	V_3, V_{11}, V_{12}	$V_1, V_5, V_6, V_7, V_8, V_9, V_{10}, V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, V_{18}, V_{19}, V_{20}$	MB-7, MB-8, MB-16	MB-12 (2), MB-3 (1), MB-11 (1), MB-13 (0), MB-14 (-1), MB-15 (-2), MB-20 (-2), MB-6 (-3), MB-18 (-3), MB-19 (-3), MB-1 (-4), MB-5 (-4), MB-9 (-4), MB-10 (-4), MB-17 (-4)
4		$V_1, V_3, V_5, V_6, V_7, V_8, V_9, V_{10}, V_{11}, V_{12}, V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, V_{18}, V_{19}, V_{20}$	MB-7, MB-8, MB-12, MB-16	MB-14 (-1), MB-13 (-2), MB-15 (-2), MB-20 (-2), MB-3 (-3), MB-11 (-3), MB-18 (-3), MB-19 (-3), MB-1 (-4), MB-5 (-4), MB-9 (-4), MB-10 (-4), MB-17 (-4), MB-6 (-5)

Table 6.3: MB selection process using MB_pick_greedy function (the value in bracket represents the score).

Then we apply the MB_pick_greedy function. As shown in Table 6.3, in iteration 0 (the initialization), there are three candidate MBs which have the same score of 5, so we randomly pick MB-8 (with score of $5 - 0 = 5$) at iteration 1. Since the nodes from MB-8 including the target have been covered, we first need to update the scores of the remaining MBs. For example, the score of MB-5 will be $3 - 1 = 2$ as there are three nodes (V_7, V_{18} and V_{20}) that are not overlapped and one overlapping node V_{19} . Next, in iteration 2, we have MB-16 emerging as the best candidate (with a score of $5 - 0 = 5$) to pick. We repeat this process for 4 iterations until all the nodes are covered. The output partitions are shown in Figure 6.16, and we run EM-CaMML on each of them and save the output models and sufficient statistics. Here we use ini3 (i.e., a random structure, as it is the most general initialisation. See Section 5.2.1 for details) when no trigger is detected and all the other parameter settings are the same as in Section 5.2.

Next, we run EM-CaMML on each of the partitions shown in Figure 6.16. As the size of MB-7 and MB-12 are less than 4 (not enough to run the trigger detect function, see Details in Chapter 4), we run the original CaMML with no prior on them. For MB-8 and MB-16, we run EM-CaMML by first checking whether a trigger is detected, otherwise insert a latent node with a random initialization structure as prior (see the detailed process in Section 5.1.3). After getting the resulting subnet of each MB partition, we then re-run EM-CaMML on all the nodes in the data using all the subnets as prior (hard constraints) as well as the corresponding sufficient statistics.

Finally, Figure 6.17 shows a final EM-CaMML output example by taking the result from the previous step. Arcs in red represent mistakes produced by EM-CaMML, either false positive arcs or arcs facing the wrong direction. Arcs in green are the false negative arcs which are present in the true network but are missing from the learned network. Arcs in black denote all other arcs learned by EM-CaMML which match the true structure. In terms of latent variable detection, the latent node V_2 has been correctly discovered

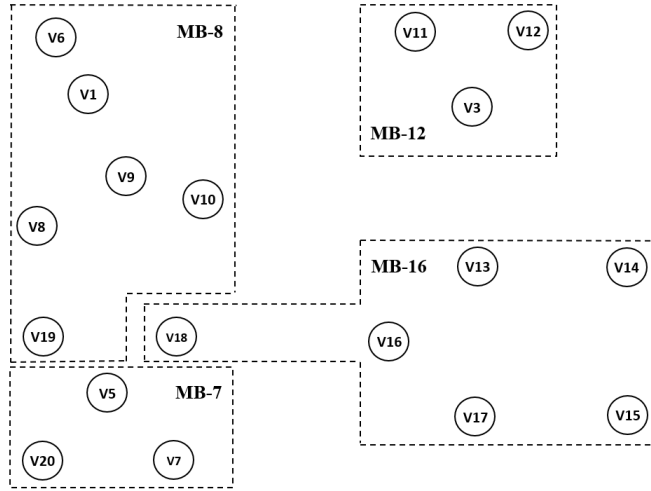


Figure 6.16: The resulting partitions (in dashed lines) using MB_pick_greedy function.

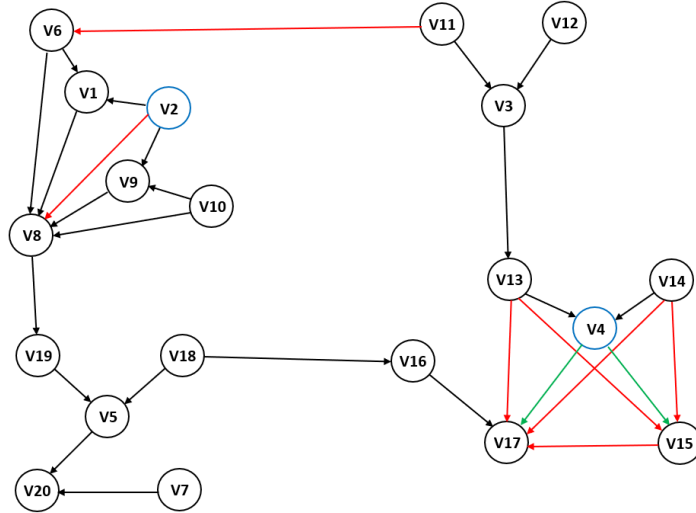


Figure 6.17: The complete network learned by EM-CaMML using ini3 (nodes “V2” and “V4” are latent, arcs in red represent all the mistaken arcs and green arcs represents true arcs but absent in the learned network).

as a Big-W, while latent node $V4$ correctly identified the marginal independence between $V13$ and $V14$, but $V15$ and $V17$ are not children of $V4$ in the learned network. All the connections between different partitions (shown as $V3 \rightarrow V13$, $V8 \rightarrow V19$ and $V18 \rightarrow V5$) are correctly identified by EM-CaMML with only one mistake $V11 \rightarrow V6$, which is in the wrong direction. In general, this learned network appears to be an acceptable structure, and the edit distance between this structure and the true structure is 9, based on the rules shown in Table 5.3. We also repeated this process multiple times, and the final output results are quite similar to Figure 6.17, only with some minor differences in the subnets learned on MB-8 and MB-16. We also tried running EM-CaMML with ini2 (see Section 5.2.1), the result is quite similar and the major difference is a better subnet learned over MB-16. This subnet does not include the red arcs $V13 \rightarrow V17$, $V14 \rightarrow V17$ and $V15 \rightarrow V17$, and more arcs $V4 \rightarrow V17$, $V15 \rightarrow V4$ and $V16 \rightarrow V4$ are introduced (see Figure 6.18). This is expected given the fact that ini2 is more similar to the true structure.

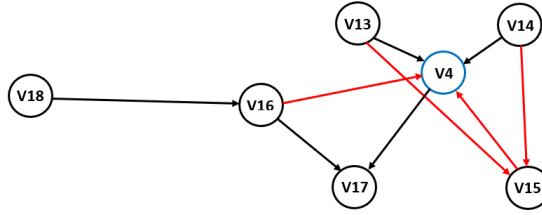


Figure 6.18: The subnet learned by EM-CaMML over MB-16 using ini2 (nodes “V2” and “V4” are latent, arcs in red represent all the mistaken arcs).

To conclude, this proof of concept study shows EM-CaMML has great potential to be extended to discovering multiple latent variables in large networks. However, EM-CaMML can still be improved in different aspects. This includes making the latent node more useful in the case of learning a subnet over MB-16 and experimenting with the other two MBMML implementations (i.e., MBMML+NB and MBMML+MBP), as well as making the whole process more automatic.

6.3 Summary

By studying the two cases presented in this chapter, we examined how EM-CaMML can be used in different scenarios, including when the latent node only partly explains the dependencies, and the case in which multiple latent nodes may exist in a large network. For the first case, as expected, PC and FCI are the best structure learners for dealing with partial Big-W cases, but EM-CaMML nonetheless has similar performance in terms of KL and CKL. For the second case, we have proposed a learning process that extends EM-CaMML to multiple latent nodes in large networks. The MBMML algorithm produces highly accurate partitions and makes for an excellent fit with EM-CaMML as they both apply MML as the learning metric. However, there are still things that need to be resolved in the future, for example, the MB_pick_greedy function can choose an MB that is larger than what EM-CaMML can handle. Given the promising results, delivering an enhanced EM-CaMML which efficiently discovers multiple latent nodes will be in scope for our future plans.

Chapter 7

Conclusion and Future Research

Discovering latent variables is a task with great challenges in causal discovery. This is due to the great uncertainty and complexity in searching the space with latent variables and the difficulties in connecting them with observed nodes. To the best of our knowledge, EM-CaMML is the first machine learner that can discover explicit latent variables automatically and without expert input. It produces a fully parameterized model using a new version of MML, which is explicitly adapted for scoring models with latent variables. It should not be categorized simply as a metric learner, since it has some properties of constraint based learners such as the conditional dependency test used in the latent variable detection function. The experimental results show us the performance of EM-CaMML is consistent and it is competitive in every case when large samples are available. For smaller samples, the performance of EM-CaMML is heavily dependent upon having a useful initial structure. To be sure, as the “no free lunch” theorem shows, there is no universal learner that can perfectly deal with every case. Another message we learned from the experiments we have conducted is that each evaluation measure — Edit Distance, KL and CKL — has something to tell us that is interesting and useful about causal discovery, and the combination of them can give us a better understanding of learning performance than any single measure.

7.1 Main Contributions

Regarding the research questions proposed at the beginning of this thesis (how to systematically find triggers, how to use triggers in causal discovery, how to extend CaMML to learn in latent model space as well as how to scale up the latent discovery), contributions have been made throughout. In Chapter 4, we proposed a systematic search for finding all triggers. Such triggers are properly defined and explained by emphasizing their importance in causal modeling. Due to the super exponential time complexity, we implemented a multi-threaded version of the search program, which runs significantly faster than the original version. We also demonstrated how to use triggers in an integrated latent variable causal discovery system, with a trigger detection function as a pre-processing step for EM-CaMML. This function has a certain level of flexibility that allows users to choose the sensitivity (as a threshold) for matching triggers. The experimental work in Chapter 4 shows the value of integrating the trigger detection function, successfully reducing the false positive rate for latents in comparison with alternative methods. While we think there is more work to do in taking full advantage of trigger discovery, we believe the first two questions have been answered here.

In terms of the research question of enhancing CaMML, the practical program named EM-CaMML was proposed and delivered (see details in Section 5). It implements a comprehensive algorithm that considers three different initial structures and applies a new

MML metric which is able to score a model with latent variables explicitly. The initial structures provide support not only for triggers but also for EM-CaMML in simplifying overly dense subnetworks. For the purpose of validating any learned latent model, EM-CaMML also compares it to the best fully observed model at the very end of the process. If the former model is no better than the latter in terms of MML cost, EM-CaMML will return the fully observed model instead. In addition to this work, we demonstrated how to turn a PC/FCI partial output structure into a fully directed DAG, as well as dealing with minor issues during the evaluation process (e.g., models with different numbers of nodes). The experimental work of Chapter 5 which compares EM-CaMML with the alternative latent discovery algorithms using different real-world test cases also gave us promising results. We can see that EM-CaMML did not always produce the best results, but it does have good overall performance. For the benefit of the research community, EM-CaMML is published online as open source software; this version runs the latent detection and learning process automatically with a well designed user interface and returns a fully parameterized output model.

Finally, two different EM-CaMML extensions are presented in Chapter 6, in a preliminary way. The first one learns partial triggers and is presented together with an experiment comparing EM-CaMML with alternative learners applied to partial triggers. The experimental results there showed PC/FCI produced the best results in most cases, but the performance of PC/FCI and EM-CaMML are close, especially as sample size increases. The KL results are almost entirely indistinguishable and the CKL results are a bit less so. The aim of the second extension, (in principle) parallel learning from Markov Blankets, was to answer the last research question by showing our preliminary work on discovering multiple latent variables. An MML MB discovery algorithm was used and applied to give accurate partitions of the network, allowing EM-CaMML to efficiently deal with latent variables in large networks. EM-CaMML successfully merged each local structure into a completed model with most links between each sub-network being learned correctly in the test case. A detailed process was proposed in the chapter for how to scale up EM-CaMML by encoding the learned MB sub-results. After this proof of concept study, we now aim to fully implement the process into EM-CaMML and do more experiments to justify the results.

7.2 Future Work

The following list describes potential future work that falls outside the scope of this thesis:

1. For the algorithm that searches for triggers (see Algorithm 1), a smarter method is desirable to generate the dependency matrix more efficiently (e.g., by looking for v-structures), as many DAGs share the same conditional dependencies in terms of d-separation.
2. Learning the dimensionality of latent variables could be incorporated, as well as discovering multiple latent nodes. One possibility would be to try different latent node dimensionalities within a range and select the best in an extended sample space. For example, one potential solution proposed by Elidan and Friedman in 2001 starts with exploring all possible states of the latent variables and then iteratively merging selected states help to improve the score, trading off accuracy for simplicity. Although this will obviously increase the complexity and execution time, it is still worth trying in the future.
3. We are considering adding a post processing phase to EM-CaMML because the latent variable could potentially be useless in some way. For example, in the EM-CaMML

results shown in Figure 5.37a the latent node H appears as the middle node in a chain structure $P2 \rightarrow H \rightarrow C$. In this case, the results will improve if there is a post processing phase that marginalizes out the latent node.

4. The current EM-CaMML can only run with a relatively small number of nodes. One of the potential solutions is to run the algorithm on different subsets of the data in parallel and merge different sub-results together. We are encouraged by the promising results of EM-CaMML shown in the proof of concept study (see Section 6.2) that applies an MB discovery technique to this problem, and particularly its success in merging different subnets into a complete model. Meanwhile, different heuristic functions can be tested in Step 2 of the learning method resulting in some non-disjoint partitions. Such partitions could be beneficial as some of the connections between different MBs can be learned by CaMML before the final merge. However, this operation could also introduce more conflicts. In the future, We will therefore implement the process fully and produce an enhanced version of EM-CaMML.
5. In terms of the EM-CaMML workflow (shown in Figure 5.8), improvements can be made by running all methods in parallel and using the resulting best latent model to compare with the best observed model. To be more specific, no matter whether a trigger is detected, we still run EM-CaMML with the three different initializations (see Section 5.2.1), enabling us to be more confident about the EM-CaMML output.

References

- Allison, L. (2005). Models for machine learning and data mining in functional programming, *Journal of Functional Programming* **15**(1): 15.
- Arntzenius, F. (1999). Reichenbach’s common cause principle, Entry in Stanford encyclopedia of philosophy, <http://plato.stanford.edu/entries/physics-Rpcc/>.
- Asher, H. B. and Asher, H. R. (1976). *Causal modeling*, Sage.
- Beinlich, I. A., Suermondt, H. J., Chavez, R. M. and Cooper, G. F. (1989). The alarm monitoring system: A case study with two probabilistic inference techniques for belief networks, *AIME 89*, Springer, pp. 247–256.
- Binder, J., Koller, D., Russell, S. and Kanazawa, K. (1997). Adaptive probabilistic networks with hidden variables, *Machine Learning* **29**(2-3): 213–244.
- Boerlage, B. (1988). Mendel genetics, <https://www.norsys.com/netlibrary/index.htm>. [Online; accessed 20-Oct-2019].
- Chen, X., Yuan, Y. and Orgun, M. A. (2019). Using Bayesian networks with hidden variables for identifying trustworthy users in social networks, *Journal of Information Science* p. 0165551519857590.
- Chickering, D. M. (1995). A transformational characterization of equivalent Bayesian network structures, *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, pp. 87–98.
- Chickering, D. M., Geiger, D., Heckerman, D. et al. (1994). Learning Bayesian networks is NP-hard, *Technical Report MSR-TR-94-17*, Microsoft Research.
- Comley, J. W., Allison, L. and Fitzgibbon, L. J. (2003). Flexible decision trees in a general data-mining environment, *International Conference on Intelligent Data Engineering and Automated Learning*, Springer, pp. 761–767.
- Cooper, G. F. (1990). The computational complexity of probabilistic inference using Bayesian belief networks, *Artificial Intelligence* **42**(2-3): 393–405.
- Cooper, G. F., Aliferis, C. F., Ambrosino, R., Aronis, J., Buchanan, B. G., Caruana, R., Fine, M. J., Glymour, C., Gordon, G., Hanusa, B. H. et al. (1997). An evaluation of machine-learning methods for predicting pneumonia mortality, *Artificial intelligence in medicine* **9**(2): 107–138.
- Cooper, G. F. and Herskovits, E. (1991). A Bayesian method for constructing Bayesian belief networks from databases, *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, pp. 86–94.
- Cooper, G. F. and Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data, *Machine learning* **9**(4): 309–347.

- Corp, N. S. (1998). Animals characteristics, <https://www.norsys.com/netlibrary/index.htm>. [Online; accessed 20-Oct-2019].
- Dagum, P. and Luby, M. (1993). Approximating probabilistic inference in Bayesian belief networks is np-hard, *Artificial intelligence* **60**(1): 141–153.
- Dempster, A. P., Laird, N. M. and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm, *Journal of the Royal Statistical Society. Series B (methodological)* pp. 1–38.
- Elidan, G. and Friedman, N. (2001). Learning the dimensionality of hidden variables, *Proceedings of the Seventeenth conference on Uncertainty in Artificial Intelligence*, pp. 144–151.
- Farrokh Alemi, J. V. (2004). Wrong side surgery, <http://openonlinecourses.com/decisionanalysis/RootCauseAnalysis.asp>. [Online; accessed 20-Oct-2019].
- François, O. and Leray, P. (2008). Bnt structure learning package: installing the package, http://ofrancois.tuxfamily.org/carb/node5_mn.html. [Online; accessed 31-Oct-2019].
- Friedman, N. (1998). The Bayesian structural em algorithm, *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, pp. 129–138.
- Friedman, N. and Goldszmidt, M. (1998). Learning Bayesian networks with local structure, *Learning in graphical models*, Springer, pp. 421–459.
- Friedman, N. et al. (1997). Learning belief networks in the presence of missing values and hidden variables, *Proceedings of the Fourteenth International Conference on Machine Learning*, Vol. 97, Morgan Kaufmann, pp. 125–133.
- Gao, T. and Wei, D. (2018). Parallel Bayesian network structure learning, *International Conference on Machine Learning*, pp. 1685–1694.
- Gelman, A., Carlin, J., Stern, H., and Rubin, D. (1995). Bayesian Data Analysis, *Chapman & Hall, London*.
- Gillispie, S. B. and Perlman, M. D. (2001). Enumerating Markov equivalence classes of acyclic digraph models, *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pp. 171–177.
- Glymour, C., Scheines, R., Spirtes, P. and Kelly, K. (1987). Discovering causal structure: Artificial intelligence, philosophy of science, and statistical modeling.
- Glymour, C. and Spirtes, P. (1988). Latent variables, causal models and overidentifying constraints, *Journal of Econometrics* **39**(1): 175–198.
- Hastings, W. K. (1970). Monte carlo sampling methods using Markov chains and their applications.
- Heckerman, D. (2008). A tutorial on learning with Bayesian networks, *Innovations in Bayesian networks*, Springer, pp. 33–82.
- Heckerman, D., Geiger, D. and Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge and statistical data, *Machine learning* **20**(3): 197–243.

- Kao, L.-J., Chiu, C.-C. and Chiu, F.-Y. (2012). A Bayesian latent variable model with classification and regression tree approach for behavior and credit scoring, *Knowledge-Based Systems* **36**: 245–252.
- Koller, D. and Sahami, M. (1996). Toward optimal feature selection, *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 284–292.
- Korb, K. B. and Nicholson, A. E. (2010). *Bayesian artificial intelligence*, CRC Press.
- Korb, K. B. and Nyberg, E. (2006). The power of intervention, *Minds and Machines* **16**(3): 289–302.
- Kullback, S. (1959). *Information theory and statistics*, John Wiley.
- Kullback, S. and Leibler, R. A. (1951). On information and sufficiency, *The Annals of Mathematical Statistics* **22**(1): 79–86.
- Lam, W. and Bacchus, F. (1994). Learning Bayesian belief networks: An approach based on the mdl principle, *Computational Intelligence* **10**(3): 269–293.
- Lauritzen, S. L. (1995). The em algorithm for graphical association models with missing data, *Computational Statistics & Data Analysis* **19**(2): 191–201.
- Lauritzen, S. L. and Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems, *Journal of the Royal Statistical Society: Series B (Methodological)* **50**(2): 157–194.
- Leray, P. and Francois, O. (2004). Bnt structure learning package: Documentation and experiments, *Technical report*, FRE CNR 2645, Laboratoire PSI, Université et INSA de Rouen.
- Li, Y. (2020). *Moral Markov Blankets: An Investigation of Some Properties and Value for Machine Learning*, PhD thesis, Faculty of Information Technology, Monash University.
- Matheson, J. E. (1990). Using influence diagrams to value information and control, *Influence Diagrams, Belief Nets, and Decision Analysis* pp. 25–48.
- Meek, C. (1995). Strong completeness and faithfulness in Bayesian networks, *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pp. 411–418.
- Mueller, W. G., Memory, A. and Bartrem, K. (2019). Causal discovery of cyber attack phases, *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, IEEE, pp. 1348–1352.
- Murphy, K. (2004). Bayes net toolbox v5 for matlab.
- Nägele, A., Dejori, M. and Stetter, M. (2007). Bayesian substructure learning—approximate learning of very large network structures, *European Conference on Machine Learning*, Springer, pp. 238–249.
- Neapolitan, R. E. (1990). *Probabilistic reasoning in expert systems: theory and algorithms*, John Wiley & Sons.
- Neil, J. R. and Korb, K. B. (1998). The MML evolution of causal models, *Technical Report 98/17*, Department of Computer Science, Monash University, Clayton, Victoria 3168, Australia.

- Netica API* (2012). https://www.norsys.com/netica_api.html. [Online; accessed 14-July-2020].
- Nicholson, A. E. and Jitnah, N. (1998). Using mutual information to determine relevance in Bayesian networks, *Pacific Rim International Conference on Artificial Intelligence*, Springer, pp. 399–410.
- Nicholson, A. E., Woodberry, O. and Twardy, C. (2010). The “Native Fish” Bayesian networks, *Technical report*, Bayesian Intelligence Technical Report 2010/3.
- Nielsen, T. D. and Jensen, F. V. (2007). *Bayesian Networks and Decision Graphs*, Springer.
- Niinimäki, T. and Parviainen, P. (2012). Local structure discovery in Bayesian networks, *Conference on Uncertainty in Artificial Intelligence, Workshop on Causal Structure Learning*, pp. 634–643.
- O’Donnell, R. (2010). *Flexible Causal Discovery with MML*, PhD thesis, Clayton School of Information Technology, Monash University.
- O’Donnell, R. (2013). Camml, <https://github.com/rodneymodonnell/CaMML>. [Online; accessed 29-Mar-2020].
- O’Donnell, R., Korb, K. and Allison, L. (2007). Causal KL: Evaluating causal discovery, *Technical report*, Clayton School of IT, Monash University.
- Pearl, J. (1986). A constraint propagation approach to probabilistic reasoning, *Machine Intelligence and Pattern Recognition*, Vol. 4, Elsevier, pp. 357–369.
- Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*, Morgan Kaufmann.
- Pearl, J. (2000). *Causality: models, reasoning and inference*, Cambridge University Press.
- Pearl, J. and Verma, T. (1991). A theory of inferred causation, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, pp. 441–452.
- Pérez-Ariza, C. B., Nicholson, A. E., Korb, K. B., Mascaro, S. and Hu, C. H. (2012). Causal discovery of dynamic Bayesian networks, *Australasian Joint Conference on Artificial Intelligence*, Springer, pp. 902–913.
- Phillips, L. D. and Edwards, W. (1966). Conservatism in a simple probability inference task, *Journal of Experimental Psychology* **72**(3): 346.
- Ramsey, J., Glymour, M., Sanchez-Romero, R. and Glymour, C. (2017). A million variables and more: the fast greedy equivalence search algorithm for learning high-dimensional graphical causal models, with an application to functional magnetic resonance images, *International Journal of Data Science and Analytics* **3**(2): 121–129.
- Richardson, T., Bailer, H. and Banerjee, M. (1999). Tractable structure search in the presence of latent variables, *Preliminary papers of the Seventh International Workshop on AI and Statistics*, Morgan Kaufmann, pp. 142–151.
- Richardson, T. S. and Spirtes, P. (2003). Causal inference via ancestral graph models, *Oxford Statistical Science Series* **1**(27): 83–105.
- Robinson, R. W. (1977). Counting unlabeled acyclic digraphs, *Combinatorial mathematics V*, Springer, pp. 28–43.

- Schwarz, G. et al. (1978). Estimating the dimension of a model, *The Annals of Statistics* **6**(2): 461–464.
- Shannon, C. E. (1948). A mathematical theory of communication, *Bell System Technical Journal* **27**(3): 379–423.
- Spearman, C. (1904). “General intelligence” objectively determined and measured, *The American Journal of Psychology* **15**(2): 201–292.
- Spirtes, P., Glymour, C. and Scheines, R. (1993). Causation, prediction, and search.
- Spirtes, P., Meek, C. and Richardson, T. (1995). Causal inference in the presence of latent variables and selection bias, *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, pp. 499–506.
- Spirtes, P., Richardson, T. and Meek, C. (1997). Heuristic greedy search algorithms for latent variable models, *Proceedings of AI & STAT’97*, pp. 481–488.
- Spirtes, P., Scheines, R. and Glymour, C. (2016). Tetrad v, http://www.phil.cmu.edu/~tetrad/tetad_old2/current.html. [Online; accessed 14-Jul-2020].
- Strobl, E. V. and Visweswaran, S. (2015). Markov boundary discovery with ridge regularized linear models, *Journal of Causal Inference* **4**(1): 31–48.
- Tembo, S. R., Vaton, S., Courant, J.-L. and Gosselin, S. (2016). A tutorial on the em algorithm for Bayesian networks: application to self-diagnosis of gpon-ftth networks, *International Wireless Communications and Mobile Computing Conference*, IEEE, pp. 369–376.
- Thiruvady, D. (2015). Revolver, <https://www.abnms.org/bn/125>. [Online; accessed 20-Oct-2019].
- Tsamardinos, I., Aliferis, C. F., Statnikov, A. R. and Statnikov, E. (2003). Algorithms for large scale Markov blanket discovery, *FLAIRS Conference*, Vol. 2, pp. 376–380.
- Velikova, M., van Scheltinga, J. T., Lucas, P. J. and Spaanderman, M. (2014). Exploiting causal functional relationships in Bayesian network modelling for personalised healthcare, *International Journal of Approximate Reasoning* **55**(1): 59–73.
- Verma, T. and Pearl, J. (1990). Equivalence and synthesis of causal models, *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, Elsevier, pp. 255–270.
- Verma, T. and Pearl, J. (1992). An algorithm for deciding if a set of observed independencies has a causal explanation, *Proceedings of the Eighth Annual Conference on Uncertainty in Artificial Intelligence*, Elsevier, pp. 323–330.
- Wallace, C. S. (2005). *Statistical and inductive inference by minimum message length*, Springer.
- Wallace, C. S. and Boulton, D. M. (1968). An information measure for classification, *The Computer Journal* **11**(2): 185–194.
- Wallace, C. S. and Korb, K. B. (1999). Learning linear causal models by MML sampling, *Causal models and intelligent data management*, Springer, pp. 89–111.

- Wallace, C. S., Korb, K. B. and Dai, H. (1996). Causal discovery via MML, *Proceedings of the Thirteenth International Conference on Machine Learning*, Vol. 96, Morgan Kaufmann, pp. 516–524.
- Xie, P., Li, J. H., Ou, X., Liu, P. and Levy, R. (2010). Using Bayesian networks for cyber security analysis, *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, IEEE, pp. 211–220.
- Yang, S., Shu, K., Wang, S., Gu, R., Wu, F. and Liu, H. (2019). Unsupervised fake news detection on social media: A generative approach, *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, pp. 5644–5651.
- Zarikas, V., Papageorgiou, E. and Regner, P. (2015). Bayesian network construction using a fuzzy rule based approach for medical decision support, *Expert Systems* **32**(3): 344–369.
- Zhang, J. (2008). Causal reasoning with ancestral graphs, *Journal of Machine Learning Research* **9**(Jul): 1437–1474.
- Zhang, X., Korb, K. B., Nicholson, A. E. and Mascaro, S. (2017). Applying dependency patterns in causal discovery of latent variable models, *Australasian Conference on Artificial Life and Computational Intelligence*, Springer, pp. 134–143.

Glossary

- BN** Bayesian Network. 1, 5–10, 15, 22, 27, 40, 47, 52, 64, 86
- CaMML** Causal Discovery via MML. 1, 13, 15, 24, 27, 46, 74, 77, 93, 95
- CDMS** Core Data Mining System. 24, 49, 74
- CPD** Conditional Probability Distribution. 5
- CPT** Conditional Probability Table. 5, 27, 41, 50, 51, 68, 89
- DAG** Directed Acyclic Graph. vi, viii, 5, 7, 16, 22, 24, 26, 27, 32, 33, 35–38, 43, 49, 56, 59, 75, 76, 94
- ED** Edit Distance. 44, 45, 70, 72, 73, 109–112
- EM** Expectation Maximisation. 7, 15, 22, 49, 51, 52, 55
- FN** False Negative. 43, 62
- FNR** False Negative Rate. 44
- FP** False Positive. 43, 62
- FPR** False Positive Rate. 44–46, 73
- MAG** Maximal Ancestral Graph. viii, 36–38
- MAP** Maximum A Posteriori. 22, 23
- MB** Markov Blanket. vi, 3, 4, 27, 77, 79, 86–90, 92, 94, 95
- MCMC** Markov Chain Monte Carlo. 1, 22, 49, 52, 55, 74, 75
- MDL** Minimum Description Length. 15, 21
- MEC** Markov Equivalence Class. 10
- MLE** Maximum Likelihood Estimation. 6, 7, 22, 23
- MML** Minimum Message Length. 1, 22, 46, 48, 49, 51, 87, 94
- MMLEC** MML Equivalent Class. 27
- NB** Naive Bayesian. 8
- PAG** Partial Ancestral Graph. 36, 37

POIPG Partially Oriented Inducing Path Graph. 17, 21

SEC Statistical Equivalence Class. 18, 22, 27

SEM Structure Expectation Maximization. 22, 44, 63, 64

TN True Negative. 43, 62

TOM Totally Ordered Model. 24, 25, 27, 49, 51, 52, 76

TP True Positive. 43, 62

TPR True Positive Rate. 85

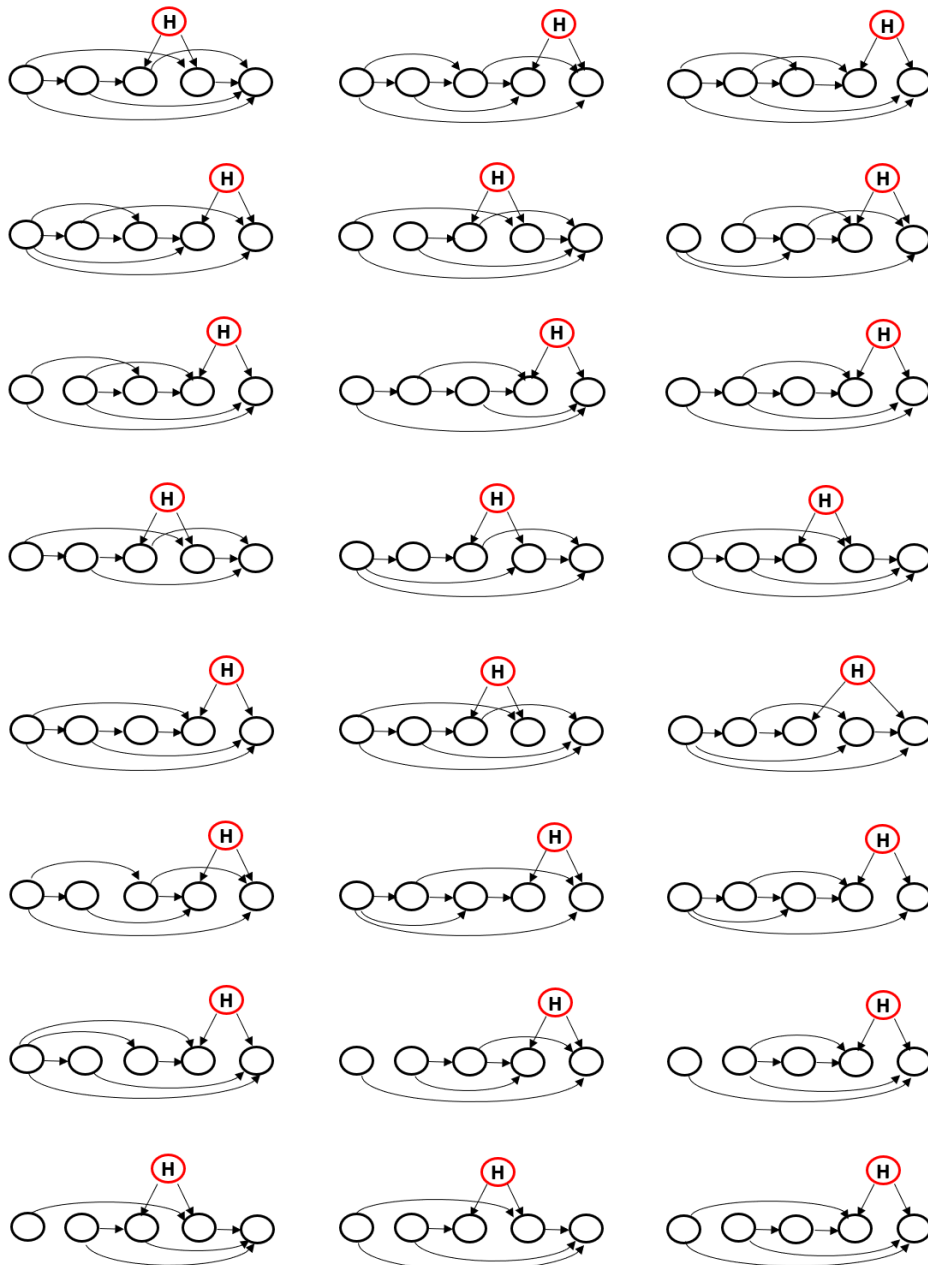
UI User Interface. 3, 74

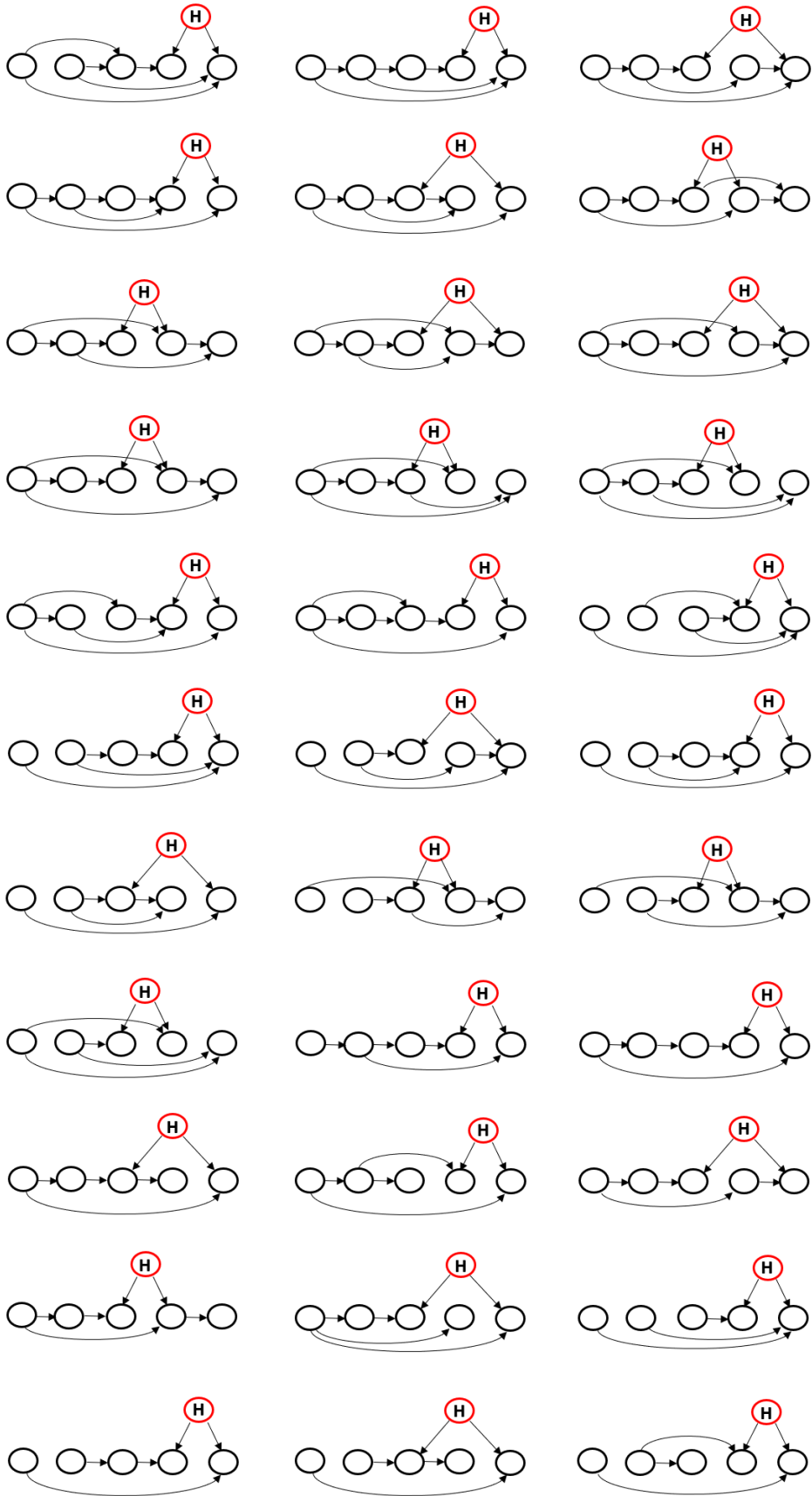
Notation

Symbol	Definition
\emptyset	An empty variable set
χ^2	Chi-square test
X, Y, Z, \dots	Different variable in a model
x, y, z, \dots	Instantiations of variable X, Y, Z, \dots
M_1, \dots, M_n	Model structures
$\pi(X_i)$	Parents of variable X_i
Θ	Complete parameter assignment of a parameterized model
\mathcal{G}	Directed acyclic graph (DAG)
\mathcal{V}	Set of variables $\{X_1, \dots, X_n\}$ in the domain
\mathcal{U}	An undirected path between two variables
\mathcal{D}	Input dataset
$ \mathcal{D} $	Number of variables in \mathcal{D}
\mathcal{D}_X	Data for variable X
\mathcal{D}^*	Missing data in \mathcal{D}
\mathcal{D}_{labels}	Variable labels in \mathcal{D}
\mathcal{L}	all possible label assignments of \mathcal{D}_{labels}
$N_{i,j,k}$	Number of instances in \mathcal{D} that $X_i = k$ and $\pi(X_i) = j$
\mathcal{O}	Set of observable variables
\mathcal{A}	An Oriented arc between two variables
\vec{x}	Instantiation of \mathcal{V}
\mathcal{E}	Set of evidence for applying d-separation
\mathcal{M}	Set of dependency matrices
$P(X_i = x)$	Probability of $X_i = x$ in a parameterized model
$E(X_i = x)$	Expected counts of $X_i = x$ in data with missing values
$X \setminus Y$	Set X with all elements of Y removed
$X \perp\!\!\!\perp Y$	X is independent of Y
$X \perp\!\!\!\perp Y Z$	X is independent of Y given Z
$X \not\perp\!\!\!\perp Y$	X is dependent of Y
$X \not\perp\!\!\!\perp Y Z$	X is dependent of Y given Z
$X \prec Y$	Variable X precedes Y in a total ordering.
$X \succ Y$	Variable X succeeds Y in a total ordering.
$I(M, \mathcal{D})$	Message length for model M and data \mathcal{D}
$\phi_T(S)$	a CPT model contains parents set S of target variable T

Appendix A

All Single Latent Triggers for Five Observed Variables





Appendix B

FCI, PC and Trigger-PC Performance by Arc Strength

	FCI	PC	Trigger-PC
Average ED	<i>15.490</i> (15.011, 15.968)	14.400 (13.864, 14.936)	14.400 (13.879, 14.921)
Accuracy	0.768 (0.742, 0.794)	0.791 (0.766, 0.816)	0.843 (0.821, 0.865)
FPR	0.139 (0.118, 0.160)	0.110 (0.091, 0.129)	0.006 (0.001, 0.011)
FNR	0.701 (0.673, 0.729)	0.707 (0.679, 0.735)	<i>0.920</i> (0.904, 0.936)
Recall	0.299 (0.271, 0.327)	0.293 (0.265, 0.321)	<i>0.080</i> (0.064, 0.096)
Precision	0.299 (0.271, 0.327)	0.347 (0.318, 0.376)	0.737 (0.710, 0.764)
Specificity	0.861 (0.840, 0.882)	0.890 (0.871, 0.909)	0.994 (0.989, 0.999)
F-score	0.299 (0.271, 0.327)	0.318 (0.290, 0.346)	<i>0.144</i> (0.123, 0.165)

Table B.1: Results of networks with maximum arc strengths using optimized alpha (bold and italic represent significantly the best and worst results).

	FCI	PC	Trigger-PC
Average ED	15.854 (15.353, 16.355)	14.878 (14.310, 15.446)	15.366 (14.818, 15.914)
Accuracy	0.703 (0.675, 0.731)	0.738 (0.711, 0.765)	0.839 (0.817, 0.861)
FPR	0.215 (0.190, 0.240)	0.171 (0.148, 0.194)	0.017 (0.009, 0.025)
FNR	0.713 (0.686, 0.740)	0.718 (0.691, 0.745)	<i>0.885</i> (0.866, 0.904)
Recall	0.287 (0.260, 0.314)	0.282 (0.255, 0.309)	<i>0.115</i> (0.096, 0.134)
Precision	0.210 (0.185, 0.235)	0.246 (0.220, 0.272)	0.571 (0.541, 0.601)
Specificity	0.785 (0.760, 0.810)	0.829 (0.806, 0.852)	0.983 (0.975, 0.991)
F-score	0.243 (0.217, 0.269)	0.263 (0.236, 0.290)	<i>0.191</i> (0.167, 0.215)

Table B.2: Results of networks with medium arc strengths using optimized alpha (bold and italic represent significantly the best and worst results).

	FCI	PC	Trigger-PC
Average ED	30.781 (30.092, 31.470)	30.737 (30.032, 31.442)	30.596 (29.896, 31.296)
Accuracy	0.722 (0.695, 0.749)	0.731 (0.704, 0.758)	0.833 (0.810, 0.856)
FPR	0.143 (0.122, 0.164)	0.130 (0.110, 0.150)	0.001 (-0.001, 0.003)
FNR	0.960 (0.948, 0.972)	0.966 (0.955, 0.977)	<i>1.000</i> (1.000, 1.000)
Recall	0.040 (0.028, 0.052)	0.034 (0.023, 0.045)	<i>0.000</i> (0.000, 0.000)
Precision	0.053 (0.039, 0.067)	0.050 (0.037, 0.063)	<i>0.000</i> (0.000, 0.000)
Specificity	0.857 (0.836, 0.878)	0.870 (0.850, 0.890)	0.999 (0.997, 1.001)
F-score	0.046 (0.033, 0.059)	0.040 (0.028, 0.052)	- -

Table B.3: Results of networks with minimum arc strengths using optimized alpha (bold and italic represent significantly the best and worst results).

	FCI	PC	Trigger-PC
Average ED	15.643 (15.146, 16.139)	14.775 (14.214, 15.337)	14.844 (14.303, 15.385)
Accuracy	0.765 (0.739, 0.791)	0.794 (0.770, 0.818)	0.848 (0.826, 0.870)
FPR	0.135 (0.114, 0.156)	0.099 (0.081, 0.117)	0.002 (-0.001, 0.005)
FNR	0.741 (0.715, 0.767)	0.741 (0.715, 0.767)	<i>0.908</i> (0.891, 0.925)
Recall	0.259 (0.233, 0.285)	0.259 (0.233, 0.285)	<i>0.092</i> (0.075, 0.109)
Precision	<i>0.276</i> (0.249, 0.303)	0.341 (0.312, 0.370)	0.889 (0.870, 0.908)
Specificity	0.865 (0.844, 0.886)	0.901 (0.883, 0.919)	0.998 (0.995, 1.001)
F-score	0.267 (0.240, 0.294)	0.294 (0.266, 0.322)	<i>0.167</i> (0.144, 0.190)

Table B.4: Results of networks with maximum arc strengths using default alpha 0.05 (bold and italic represent significantly the best and worst results).

	FCI	PC	Trigger-PC
Average ED	16.044 (15.513, 16.574)	15.194 (14.599, 15.790)	15.362 (14.794, 15.929)
Accuracy	0.739 (0.712, 0.766)	0.766 (0.740, 0.792)	0.842 (0.820, 0.864)
FPR	0.176 (0.153, 0.199)	0.144 (0.123, 0.165)	0.016 (0.008, 0.024)
FNR	0.690 (0.662, 0.718)	0.690 (0.662, 0.718)	<i>0.874</i> (0.854, 0.894)
Recall	0.310 (0.282, 0.338)	0.310 (0.282, 0.338)	<i>0.126</i> (0.106, 0.146)
Precision	0.260 (0.233, 0.287)	0.300 (0.272, 0.328)	0.611 (0.582, 0.640)
Specificity	0.824 (0.801, 0.847)	0.856 (0.835, 0.877)	0.984 (0.976, 0.992)
F-score	0.283 (0.256, 0.310)	0.305 (0.277, 0.333)	<i>0.209</i> (0.184, 0.234)

Table B.5: Results of networks with medium arc strengths using default alpha 0.05 (bold and italic represent significantly the best and worst results).

	FCI	PC	Trigger-PC
Average ED	31.632 (30.964, 32.300)	31.600 (30.922, 32.278)	31.570 (30.886, 32.253)
Accuracy	0.751 (0.725, 0.777)	0.755 (0.729, 0.781)	0.833 (0.810, 0.856)
FPR	0.100 (0.082, 0.118)	0.096 (0.078, 0.114)	0.001 (-0.001, 0.003)
FNR	0.994 (0.989, 0.999)	0.994 (0.989, 0.999)	<i>1.000</i> (1.000, 1.000)
Recall	0.006 (0.001, 0.011)	0.006 (0.001, 0.011)	<i>0.000</i> (0.000, 0.000)
Precision	0.011 (0.005, 0.017)	0.012 (0.005, 0.019)	<i>0.000</i> (0.000, 0.000)
Specificity	0.900 (0.882, 0.918)	0.904 (0.886, 0.922)	0.999 (0.997, 1.001)
F-score	0.008 (0.003, 0.013)	0.008 (0.003, 0.013)	- -

Table B.6: Results of networks with minimum arc strengths using default alpha 0.05 (bold and italic represent significantly the best and worst results).

Appendix C

Test Network Selection

C.1 Test Network Sources

Type	Network Name	Latent Node	Additional Arc
Big-W	Space Mission(Matheson, 1990)	Launch	
Big-W	Alarm(Beinlich et al., 1989)	HR	
Big-W	Naive Fish V1(Nicholson et al., 2010)	Annual Rainfall	
Big-W	Stud farm(Nielsen and Jensen, 2007)	Ann	
covered Big-W	Space Mission(Matheson, 1990)	Launch	Descender → Venus Descender
covered Big-W	Alarm(Beinlich et al., 1989)	HR	ERRLOWOUTPUT → ERRCAUTER
covered Big-W	Naive Fish V1(Nicholson et al., 2010)	Annual Rainfall	Drought Conditions → Pesticide Use
covered Big-W	Stud farm(Nielsen and Jensen, 2007)	Ann	L → Brain
Latent	Earthquake(Korb and Nicholson, 2010)	Alarm	
Latent	Bookbag(Phillips and Edwards, 1966)	Book Bag	
Latent	Animal(Corp, 1998)	Class	
Latent	Asia(Lauritzen and Spiegelhalter, 1988)	either	
No Latent	Revolver(Thiruvady, 2015)		
No Latent	Cancer Neapolitan(Neapolitan, 1990)		
No Latent	Mendel Genetics(Boerlage, 1988)		
No Latent	Wrong Side Surgery(Farrokh Alemi, 2004)		

Table C.1: All test networks.

Network Name	Extra Arc	Tested Threshold
Space Mission(Matheson, 1990)	Venus Descender → Result Mars	0.005, 0.01, 0.02, 0.036
Alarm(Beinlich et al., 1989)	ERRCAUTER → HRBP	0.005, 0.01, 0.02, 0.026
Naive Fish V1(Nicholson et al., 2010)	Pesticide Use → River Flow	0.005, 0.006, 0.007, 0.0086
Stud farm(Nielsen and Jensen, 2007)	L → Dorothy	0.005, 0.0053, 0.0056, 0.056

Table C.2: Arc prior optimization test networks.

C.2 All Test Networks

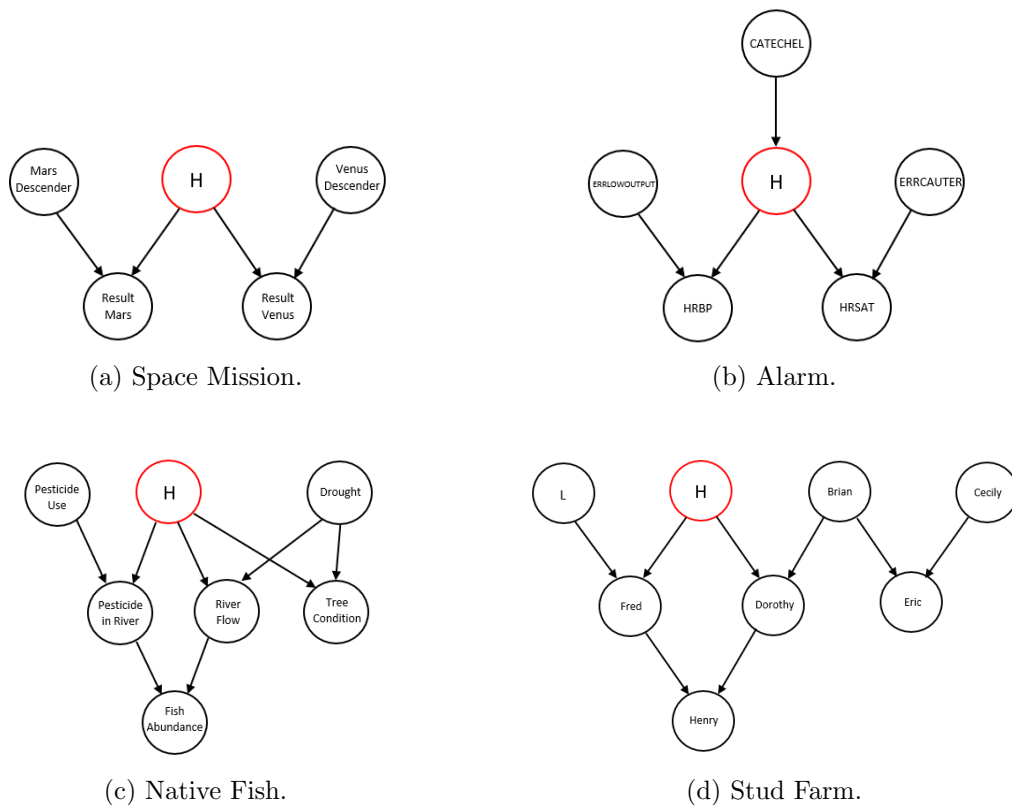


Figure C.1: Big-W test networks (node “H” represents the latent).

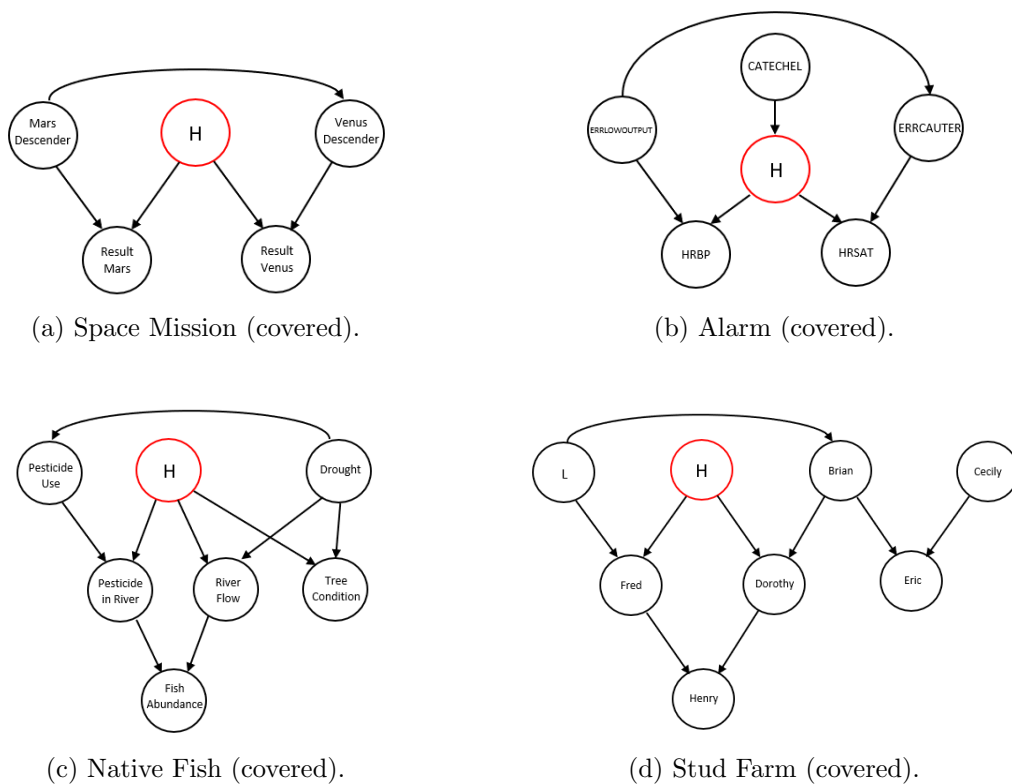
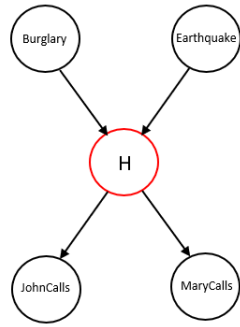
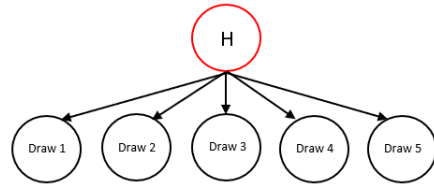


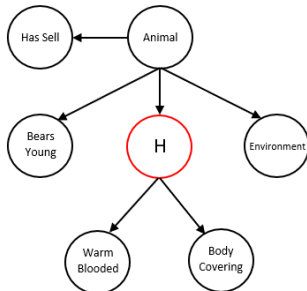
Figure C.2: Covered Big-W test networks (node “H” represents the latent).



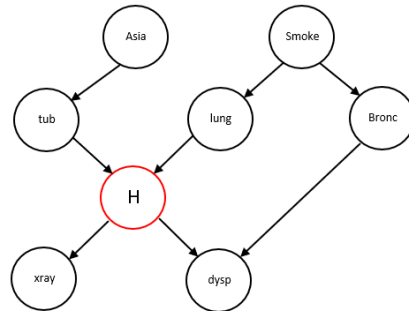
(a) Earthquake.



(b) Bookbags.

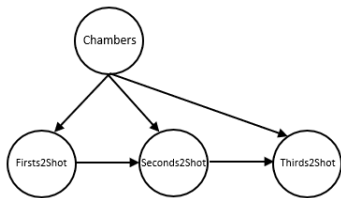


(c) Animal.

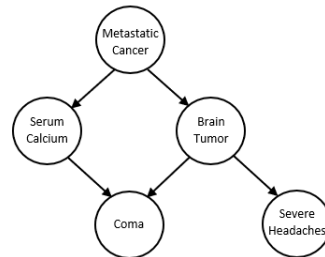


(d) Asia.

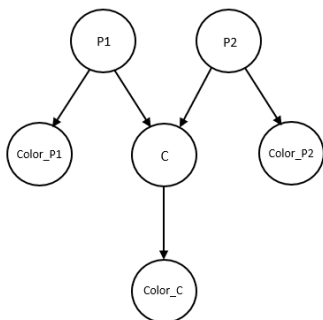
Figure C.3: Latent test networks (node “H” represents the latent).



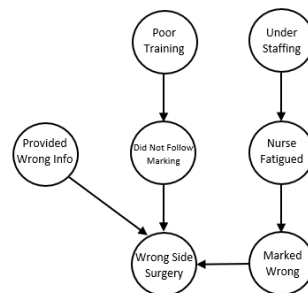
(a) Revolver.



(b) Cancer Neapolitan.



(c) Mendel Genetics.



(d) Wrong Side Surgery.

Figure C.4: No latent test networks.