



MONASH UNIVERSITY

DOCTORAL THESIS

---

# Markov chain Monte Carlo methods for Bayesian network inference, with applications in systems biology

---

*Author:*

Salem A. ALYAMI

*Supervisor:*

Jonathan M. KEITH

*A thesis submitted in fulfillment of the requirements*

*for the degree of Doctor of Philosophy*

*in the*

Statistics Group

School of Mathematical Sciences

2017

# Copyright Notice

The author (2017). Except as provided in the Copyright Act 1968, this thesis may not be reproduced in any form without the written permission of the author.

I certify that I have made all reasonable efforts to secure copyright permissions for third-party content included in this thesis and have not knowingly copyright content to my work without the owner's permission.

# Abstract

Learning the structure of Bayesian networks (BNs) from data has become an indispensable tool in a variety of applications to model uncertain knowledge and causality among variables. For example, a common problem in systems biology is to infer the biological molecules (represented by nodes) and causal interactions (represented by directed edges) relevant to a particular biochemical pathway or network. Getting stuck in local maxima is a challenging problem that arises while learning Bayesian network structures. Special attention was recently paid to resolving this problem.

Markov chain Monte Carlo (MCMC) sampling has a wider applicability to infer hard BNs than exact algorithms, despite the fact that MCMC sampling may exhibit slow convergence. This thesis proposes MCMC samplers that have been known to substantially avoid getting stuck in local maxima but have not been modified to fit simulating Bayesian network structures from discrete search spaces. The proposed MCMC samplers are new instances of the Neighbourhood Sampler (NS), Hit-and-Run (HAR) sampler, and Metropolis-Hastings (MH) sampler.

The MH sampler is a baseline and popular MCMC method. One of its limitations is that it can potentially get stuck at a local maximum graph in the search space. A revised version of the MH sampler has been introduced in this thesis.

The NS possesses a reduction step in which rejected elements are excluded from being chosen a second time. Also, each new element is chosen in two steps: starting from an initial element  $X$ , a neighbour  $Y$  is first selected, then a neighbour  $Z$  of  $Y$  is proposed. These two steps help to ameliorate the problem of local modes.

The HAR sampler is a unique MCMC method that proposes movements across continuous search spaces in a large step so that it can move from a current point to another distant point in the space. The features of the HAR sampler are modified in this thesis to learn BNs from discrete spaces.

The performance of the three MCMC samplers have been evaluated by exploring feasible spaces of BNs uniformly, and learning the structures of BNs when the problem of local maxima is present.

Each of the three MCMC samplers have been implemented using two adaptive techniques proposed in this thesis to reduce the time-complexity required to enumerate adjacent graphs of particular BNs, and to calculate the posterior probability distribution of sampled BNs.

As a part of this project, I developed a new graphical user interface (GUI) designed for practitioners from different disciplines to facilitate sampling and inferring BNs using the three MCMC samplers proposed in this thesis.

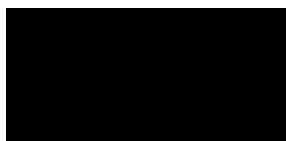
The simulations conducted in this thesis have indicated efficient uniform sampling of Bayesian network structures, and more rapid rates of convergence to the posterior probability distribution using the NS and HAR compared to the MH. The first adaptive algorithm used to enumerate adjacent graphs has achieved a useful speed-up in practice compared to the standard brute-force approach. The second adaptive technique has been shown to be  $O(N)$  times faster than the standard brute-force method in calculating the scoring function of a sampled network.



# Declaration of Authorship

This thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signature:

A solid black rectangular box used to redact the author's signature.

Print Name:

**Salem Ali ALYAMI**

Date:

**June 15, 2017**

# Publications during enrolment

## Papers:

- 1) Alyami, S. A., Azad, A. K. and Keith, J. M. (2016). Uniform sampling of directed and undirected graphs conditional to vertex connectivity. *Electronic Notes in Discrete Mathematics* 53, 43-55, 2016.
- 2) Alyami, S. A., Azad, A. K. and Keith, J. M. (2016). The Neighbourhood MCMC sampler for learning Bayesian networks. In Proceedings of the First International Workshop on Pattern Recognition (eds Jiang X, Chen G, Capi G, Ishii C) Proceedings of SPIE 10011,100111K.

## Poster:

- 3) Alyami, S., Cheung, W., Jin, Y., and Keith, J. (2014). Sampling graph space with the Neighbourhood sampler. Australian Bioinformatics Conference (ABC). Royal Children's Hospital, Melbourne, AUSTRALIA, Poster (61).

## Abstracts and Oral Presentations:

- 4) Alyami, S. A. and Keith, J. M. (2014). Bayesian Inference using Neighbourhood Sampler for learning Bayesian networks. Sixth Annual Conference of the Australasian Bayesian Network Modelling Society (ABNMS). Rotorua, New Zealand, OP(8) page 19.
- 5) Alyami, S. A., Azad, A. K. and Keith, J. M. (2015). Uniform sampling of directed and undirected graphs conditional to vertex connectivity. International Conference on Graph Theory and its Applications (ICGTA). Amrita University, Coimbatore, India, page 36–37.
- 6) Alyami, S. A., Azad, A. K. and Keith, J. M. (2015). A new version of the Hit-and-Run algorithm to sample graph spaces. The 39th Australasian Conference on Combinatorial Mathematics and Combinatorial Computing (39ACCMCC). Queensland University of Technology, Brisbane, Australia, page 7.

# Acknowledgements

I would like to express my sincere gratitude to:

- my supervisor Assoc. Prof. Jonathan M. Keith for the continuous support of my PhD study, for his motivation, patience, and immense knowledge. His high availability and guidance had a significant impact in all the time of research and on the completion of this thesis. With Jon I could develop my programming skills to handle highly complicated data structures, and thus a new graphical user interface has been produced, in collaboration with Jon and fellow PhD student Azad, as a part of this thesis.
- my PhD advisory committee: Assoc. Prof. Tim Garoni (Associate Supervisor) and Assoc. Prof. Catherine Forbes (External Supervisor), for their academic support, suggestions and reflections. Tim and Catherine were very approachable and I am really grateful for their advice to overcome challenges,
- my milestone review panel members for their insightful comments and encouragement, and for the hard questions which boosted me to strengthen my research from various perspectives,
- my sponsor, Al-Imam Muhammed Ibn Saud Islamic University, for their generous sponsorship, funding and unlimited support for me and my family,
- Azad A. K., a member in my research-group, for the useful conversations and programming guidance I have had through the project, resulting in jointly authored papers,
- the administration staff in the faculty of science for all their efforts and diligence over the course, in particular Ms Linda, the administrative officer of

the school of mathematical sciences, for her coordinating role during my milestones,

- Monash university, for all services, study facilities and for their financial funding to participate in three international conferences in India, Japan and Queensland,
- my wife who made my life so meaningful by her moral support and care of me and our kids to overcome all difficulties during my PhD study.

I dedicate this thesis to:

- the soul of my father who passed away when I was ten years old, my mother, my family members, and friends.

# Contents

<b>1</b>	<b>Thesis Objectives</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Motivations . . . . .	2
1.3	General objectives and original contributions . . . . .	3
1.4	Chapters abstracts and objectives . . . . .	4
<b>2</b>	<b>Markov Chain Monte Carlo Sampling</b>	<b>8</b>
2.1	Introduction . . . . .	8
2.2	Markov chains . . . . .	9
2.2.1	Irreducible, aperiodic and recurrent Markov chain . . . . .	10
2.2.2	Limiting distribution . . . . .	12
2.2.3	Detailed balance property . . . . .	13
2.3	Monte Carlo simulation . . . . .	14
2.4	Markov chain Monte Carlo methods . . . . .	18
2.4.1	Metropolis-Hastings method . . . . .	18
2.4.2	Neighbourhood Sampler . . . . .	20
2.4.3	Hit-and-Run algorithm . . . . .	22
2.5	MCMC sampling issues and convergence diagnostic tests . . . . .	25
2.5.1	MCMC sampling issues . . . . .	25
2.5.2	Convergence diagnostics . . . . .	28
<b>3</b>	<b>A Review of Bayesian Networks</b>	<b>32</b>
3.1	Introduction . . . . .	32

3.2	Notations and definitions . . . . .	33
3.2.1	Directed acyclic graphs . . . . .	33
3.2.2	Markov property . . . . .	34
3.2.3	Markov blanket . . . . .	35
3.2.4	Conditional probabilities table . . . . .	35
3.2.5	Joint probability function . . . . .	38
3.2.6	Equivalent graphs . . . . .	38
3.3	Graph space . . . . .	39
3.4	Graph constraints . . . . .	40
3.4.1	Connectivity . . . . .	41
3.4.2	Acyclicity . . . . .	42
3.4.3	Node degree . . . . .	42
3.5	Bayesian inference . . . . .	43
3.5.1	Posterior distribution . . . . .	43
3.5.2	Prior distribution . . . . .	44
3.5.3	Bayesian estimation . . . . .	45
3.6	Learning Bayesian networks . . . . .	45
3.6.1	Learning Bayesian network parameters . . . . .	46
3.6.2	Learning Bayesian network structures . . . . .	49
3.6.3	MCMC methods for learning Bayesian networks . . . . .	49
3.6.4	Non-MCMC methods for learning Bayesian networks . . . . .	51
<b>4</b>	<b>Using the MH, NS and HAR to Sample Bayesian Networks</b>	<b>54</b>
4.1	Introduction . . . . .	54
4.2	Enumerating a set of adjacent graphs . . . . .	55
4.3	Standard brute-force approach . . . . .	56
4.4	Assigning candidate graphs iteratively using the MH, NS and HAR . . . . .	57
4.5	Metroplis-Hastings sampler . . . . .	58

4.6	Neighbourhood Sampler . . . . .	61
4.7	Hit-and-Run sampler . . . . .	62
4.7.1	Constructing a path in a space of graph . . . . .	63
4.7.2	The diameter of a space of graphs . . . . .	64
4.7.3	Algorithm . . . . .	68
4.7.4	Acceptance-rejection ratio in the HAR . . . . .	70
4.8	Generating an initial network at random . . . . .	72
4.9	Conclusion . . . . .	74
<b>5</b>	<b>Sampling Bayesian Networks Uniformly</b>	<b>75</b>
5.1	Introduction . . . . .	75
5.2	Methods and model . . . . .	76
5.3	Experimental results . . . . .	77
5.3.1	Transition options . . . . .	77
5.3.2	Sampling with $ \mathcal{K} $ iterations . . . . .	80
5.3.3	Sampling BNs uniformly . . . . .	82
5.3.4	Sum of squared differences . . . . .	88
5.4	Conclusion . . . . .	91
<b>6</b>	<b>Adaptive Algorithms for Faster Adjacent Graphs Enumeration and Function Scoring</b>	<b>92</b>
6.1	Introduction . . . . .	92
6.2	Adaptive technique for faster enumeration of adjacent graphs . . . . .	93
6.2.1	Notations, definitions and propositions . . . . .	94
6.2.2	Algorithm and illustrative example . . . . .	96
6.2.3	Standard brute-force vs adaptive algorithm . . . . .	102
6.2.3.1	Brute-force algorithm: complexity analysis . . . . .	103
6.2.3.2	Adaptive algorithm: worst-case complexity . . . . .	104

6.2.3.3	Simulation study: speed-up achieved in practice . . .	105
6.3	Adaptive function scoring to compute Bayesian network parameters .	106
6.3.1	Conditional probabilities in a graph . . . . .	108
6.3.2	Exploring dataset . . . . .	109
6.3.3	Big-O expression . . . . .	110
6.3.4	Adaptive scoring function . . . . .	111
6.3.5	Algorithm and illustrative example . . . . .	113
<b>7</b>	<b>Applications of Bayesian networks in Systems Biology Using the</b>	
	<b>MH, NS and HAR</b>	<b>118</b>
7.1	Inferring structures from Microarray data . . . . .	119
7.1.1	Background . . . . .	119
7.1.2	Experimental results . . . . .	120
7.1.3	Conclusion . . . . .	123
7.2	Inferring the Mendel Peas network . . . . .	123
7.2.1	Background . . . . .	123
7.2.2	Experimental results . . . . .	123
7.2.3	Conclusion . . . . .	128
7.3	Inferring the Diagnostic Chest Clinic network . . . . .	128
7.3.1	Background . . . . .	128
7.3.2	Experimental results . . . . .	128
7.3.3	Conclusion . . . . .	133
7.4	Inferring the Raf-Signaling Pathway network . . . . .	134
7.4.1	Background . . . . .	134
7.4.2	Experimental results . . . . .	135
7.4.3	Conclusion . . . . .	146
7.5	Inferring the KEGG Pathways network . . . . .	146
7.5.1	Background . . . . .	146



7.5.2	Learning initial networks . . . . .	147
7.5.3	Experimental results . . . . .	149
7.5.3.1	Comparing the performance of MH, NS and HAR based on the same number of iterations . . . . .	149
7.5.3.2	Comparing the MH, NS and HAR based on time elapsed . . . . .	158
7.5.4	Conclusion . . . . .	160
<b>8</b>	<b><i>BNMCMC</i>: A New Graphical User Interface</b>	<b>162</b>
8.1	Introduction . . . . .	162
8.2	Main functions of <i>BNMCMC</i> . . . . .	164
8.2.1	Variables in <i>BNMCMC</i> . . . . .	164
8.2.2	Sampling Bayesian networks uniformly . . . . .	165
8.3	Sampling methods in <i>BNMCMC</i> . . . . .	165
8.4	Parameters model in <i>BNMCMC</i> . . . . .	165
8.5	Limitations of <i>BNMCMC</i> . . . . .	166
8.6	User guidelines . . . . .	166
8.6.1	Data-file format . . . . .	166
8.6.2	Steps to run <i>BNMCMC</i> . . . . .	167
8.6.3	Modular design . . . . .	169
8.7	Illustrative practical example . . . . .	170
8.8	Conclusion . . . . .	175
<b>9</b>	<b>Conclusion</b>	<b>176</b>
9.1	Scope of the thesis . . . . .	176
9.2	Findings summary . . . . .	177
9.3	Future work . . . . .	178
9.3.1	Possible update to <i>BNMCMC</i> . . . . .	178

9.3.2	Mathematical work . . . . .	180
<b>A</b>	<b>Using MCMC Samplers to Sample Bayesian Networks</b>	<b>182</b>
A.1	Flowchart describing the NS process . . . . .	182
A.2	All possible paths within three transitions of a graph of three nodes .	183
<b>B</b>	<b>Sampling Bayesian Networks Uniformly</b>	<b>184</b>
B.1	Checking normality . . . . .	184
<b>C</b>	<b>Applications</b>	<b>188</b>
C.1	Inferring structures from Microarray network . . . . .	188
C.1.1	The true structure of Microarray network . . . . .	188
C.1.2	The CPTs of Microarray network . . . . .	188
C.2	Inferring the Mendel Peas network . . . . .	189
C.2.1	The CPTs of Mendel network . . . . .	189
C.2.2	Summary statistics . . . . .	190
C.2.3	Mendel Peas network based on adding and deleting edges . . .	190
C.3	Inferring the Diagnostic Chest network . . . . .	191
C.4	Inferring the Raf-Signaling Pathway network . . . . .	192
C.4.1	Twelve random initial networks . . . . .	192
C.4.2	Log posterior for 36 chains and 10,000 iterations each . . . . .	194
C.4.3	All edge posteriors for Raf-Signaling network . . . . .	196
C.4.3.1	Metropolis-Hastings sampler . . . . .	196
C.4.3.2	Hit-and-Run sampler . . . . .	197
C.4.3.3	Neighbourhood Sampler . . . . .	198
C.5	Inferring the KEGG Pathways network . . . . .	199
C.5.1	More simulation run using the MH sampler . . . . .	199
C.5.2	Figures of best scoring networks . . . . .	200

<b>D Conclusion</b>	<b>208</b>
D.1 Sampling CUDGs uniformly . . . . .	208
<b>Bibliography</b>	<b>210</b>

# List of Figures

2.1	Space dimensions . . . . .	23
3.1	Bayesian Network structure . . . . .	34
3.2	DAG representing an extension of the Markov property . . . . .	34
3.3	Markov blanket . . . . .	35
3.4	An example of inferred Bayesian network. . . . .	36
3.5	Equivalent networks . . . . .	39
3.6	Graph space of three nodes . . . . .	40
4.1	All possible adjacent graphs . . . . .	56
4.2	Calculate $q(G H)$ with the MH . . . . .	60
4.3	Calculate $q(H G)$ with the MH . . . . .	60
4.4	Graphical example to define $\mu$ . . . . .	61
4.5	Transition by adding and deleting only . . . . .	65
4.6	Minimum transitions between a pair of graphs . . . . .	66
4.7	Probability of transition with the HAR . . . . .	72
5.1	AD vs ADR with 100,000 iterations . . . . .	79
5.2	AD vs ADR with 500,000 iterations . . . . .	80
5.3	Sampling the graph space of four nodes . . . . .	81
5.4	Sampling the graph space of five nodes . . . . .	82
5.5	MH with four nodes . . . . .	83
5.6	HAR with four nodes . . . . .	84
5.7	NS with four nodes . . . . .	84

5.8	MH with five nodes . . . . .	86
5.9	HAR with five nodes . . . . .	86
5.10	NS with five nodes . . . . .	87
5.11	SSD values vs iterations with a space of four nodes. . . . .	89
5.12	SSD values vs iterations with a space of five nodes. . . . .	89
5.13	SSD values vs MCMC sampler with a space of four nodes. . . . .	90
5.14	SSD values vs MCMC sampler with a space of five nodes. . . . .	90
6.1	An initialised Bayesian network . . . . .	100
6.2	Operations to assign adjacent graphs . . . . .	102
6.3	Compare CPTs . . . . .	107
6.4	An initialized Bayesian network . . . . .	114
6.5	Moving to an adjacent graph after adding an edge . . . . .	115
6.6	Moving to an adjacent graph after deleting an edge . . . . .	116
6.7	Moving to an adjacent graph after reversing an edge . . . . .	116
7.1	Log likelihood functions with four nodes . . . . .	120
7.2	Sampled graphs and their frequencies with the NS . . . . .	121
7.3	Sampled graphs and their frequencies with the HAR . . . . .	121
7.4	Sampled graphs and their frequencies with the MH . . . . .	122
7.5	Sampled edges with highest proportions . . . . .	122
7.6	Log posterior with six nodes . . . . .	124
7.7	Geweke diagnostic for Mendel network using the MH . . . . .	125
7.8	Geweke diagnostic for Mendel network using the NS . . . . .	125
7.9	Geweke diagnostic for Mendel network using the HAR . . . . .	126
7.10	The posterior mean probabilities of edges with six nodes . . . . .	127
7.11	Gelman & Rubin diagnostic with eight nodes using the MH . . . . .	129
7.12	Gelman & Rubin diagnostic with eight nodes using the NS . . . . .	130

7.13 Gelman & Rubin diagnostic with eight nodes using the HAR . . . . .	130
7.14 Chest Clinic network learned by non-MCMC samplers . . . . .	131
7.15 Chest Clinic network learned by MCMC samplers . . . . .	132
7.16 Raf-Signaling Pathway . . . . .	135
7.17 Posterior of Raf-Signaling Pathway with the MH . . . . .	136
7.18 Posterior of Raf-Signaling Pathway with the HAR . . . . .	137
7.19 Posterior of Raf-Signaling Pathway with the NS . . . . .	137
7.20 Posterior mean for the 1st chain of Raf-Signaling Pathway . . . . .	139
7.21 Posterior mean for the 2nd chain of Raf-Signaling Pathway . . . . .	139
7.22 Posterior mean for the 3rd chain of Raf-Signaling Pathway . . . . .	140
7.23 Posterior mean for the 4th chain of Raf-Signaling Pathway . . . . .	140
7.24 Posterior mean for the 5th chain of Raf-Signaling Pathway . . . . .	141
7.25 Posterior mean for the 6th chain of Raf-Signaling Pathway . . . . .	141
7.26 Posterior mean for the 7th chain of Raf-Signaling Pathway . . . . .	142
7.27 Posterior mean for the 8th chain of Raf-Signaling Pathway . . . . .	142
7.28 Posterior mean for the 9th chain of Raf-Signaling Pathway . . . . .	143
7.29 Posterior mean for the 10th chain of Raf-Signaling Pathway . . . . .	143
7.30 Posterior mean for the 11th chain of Raf-Signaling Pathway . . . . .	144
7.31 Posterior mean for the 12th chain of Raf-Signaling Pathway . . . . .	144
7.32 A Bayesian network inferred by GES and HCS . . . . .	148
7.33 Inferring KEGG Pathways network with MCMC samplers . . . . .	150
7.34 Posterior of KEGG Pathway network with the MH . . . . .	151
7.35 KEGG Pathways network inferred by the HAR (GES) . . . . .	153
7.36 KEGG Pathways network inferred by the HAR (HCS) . . . . .	154
7.37 KEGG Pathways network inferred by the NS (GES) . . . . .	155
7.38 KEGG Pathways network inferred by the NS (HCS) . . . . .	156
7.39 All common edges among the best scoring networks . . . . .	158

7.40	MH, NS and HAR performance after a certain amount of time . . . .	160
8.1	<i>BNMCMC</i> window. . . . .	163
8.2	Data file format used in <i>BNMCMC</i> . . . . .	167
8.3	Data file can be converted by <i>BNMCMC</i> . . . . .	167
8.4	Initial network format . . . . .	168
8.5	Modular design for the <i>BNMCMC</i> . . . . .	170
8.6	Input settings in <i>BNMCMC</i> . . . . .	171
8.7	Log posterior vs iterations in <i>BNMCMC</i> . . . . .	171
8.8	SSD vs Lag in <i>BNMCMC</i> . . . . .	172
8.9	TD vs ED in <i>BNMCMC</i> . . . . .	173
8.10	List of edges with posterior probabilities in <i>BNMCMC</i> . . . . .	174
8.11	Edges of the highest frequency in <i>BNMCMC</i> . . . . .	175
A.1	All possible paths . . . . .	183
B.1	MH with four nodes. . . . .	185
B.2	HAR with four nodes. . . . .	185
B.3	NS with four nodes. . . . .	186
B.4	MH with five nodes. . . . .	186
B.5	HAR with five nodes. . . . .	187
B.6	NS with five nodes . . . . .	187
C.1	The true structure of Microarray network . . . . .	188
C.2	The CPTs of Microarray network . . . . .	189
C.3	Conditional probabilities used to simulate datapoints . . . . .	189
C.4	Three initial graphs for learning Mendel network . . . . .	191
C.5	Posterior edge probabilities for Mendel network . . . . .	191
C.6	Twelve initial networks for Raf-Signaling network . . . . .	193
C.7	Posterior of Raf-Signaling Pathway with the MH . . . . .	194

C.8	Posterior of Raf-Signaling Pathway with the HAR . . . . .	195
C.9	Posterior of Raf-Signaling Pathway with the NS . . . . .	195
C.10	Posterior edges probabilities with the MH . . . . .	196
C.11	Posterior edges probabilities with the HAR . . . . .	197
C.12	Posterior edges probabilities with the NS . . . . .	198
C.13	Posterior of KEGG Pathway network with the MH . . . . .	199
C.14	Best scoring KEGG network learned by HAR (GES) . . . . .	200
C.15	Best scoring KEGG network learned by HAR (HCS) . . . . .	201
C.16	Best scoring KEGG network learned by NS (HCS) . . . . .	202
C.17	Best scoring KEGG network learned by NS (GES) . . . . .	203
C.18	NS when the network learned by the GES is the initial network . . .	204
C.19	NS when the network learned by the HCS is the initial network . . .	205
C.20	HAR when the network learned by the GES is the initial network . .	206
C.21	HAR when the network learned by the HCS is the initial network . .	207
D.1	All possible adjacent graphs in a CUDG . . . . .	208
D.2	Neighbourhood Sampler vs uniform distribution for CUDGs . . . . .	209



# List of Tables

1.1	Chapters that include original contributions.	3
3.1	CPTs of three nodes	36
3.2	Data-points observed for three variables.	37
3.3	Graph space sizes of BNs and connected BNs	39
3.4	Some common conjugate prior distributions	44
5.1	Summary statistics of the MH, HAR and NS frequencies	88
6.1	Effects a single transition on lists	98
6.2	Populate adjacent graphs	100
6.3	Update lists after adding an edge	101
6.4	Update lists after deleting an edge	101
6.5	Update lists after reversing an edge	102
6.6	Comparing speed: non-adaptive approach vs adaptive approach	106
6.7	Effects on $Pa(v_i)$ and $Pa(v_j)$	114
6.8	Speed comparison using MD	117
7.1	Quantitative outputs for the inferred Raf-Signaling Pathway	145
7.2	The highest log scoring function produced by MCMC samplers	150
7.3	Quantitative outputs for the inferred KEGG Pathways network	157
7.4	Quantitative outputs after a certain amount of time	159
9.1	Ranking MCMC samplers at different performance criteri	177

C.1 Numerical outputs with six nodes . . . . .	190
C.2 CPTs of Chest Clinic network . . . . .	192

# List of Abbreviations

<b>MCMC</b>	<b>Markov Chain Monte Carlo</b>
<b>BN</b>	<b>Bayesian Netowrk</b>
<b>BNs</b>	<b>Bayesian Netowrks</b>
<b>RGs</b>	<b>Random Graphs</b>
<b>NS</b>	<b>Neighborhood Sampler</b>
<b>HAR</b>	<b>Hit and Run</b> sampler
<b>MH</b>	<b>Metropolis Hastings</b> sampler
<b>SSD</b>	<b>Sum of Squared Differences</b>
<b>MLE</b>	<b>Maximum Likelihood Estimation</b>
<b>DM</b>	<b>Dirichlet Multinomial</b>
<b>DAG</b>	<b>Directed Acyclic Graph</b>
<b>DAGs</b>	<b>Directed Acyclic Graphs</b>
<b>CDAG</b>	<b>Connected Directed Acyclic Graph</b>
<b>CUDG</b>	<b>Connected Un-Directed Graph</b>
<b>GUI</b>	<b>Graphical User Interface</b>
<b>BFS</b>	<b>Breadth First Search</b>
<b>DFS</b>	<b>Depth First Search</b>
<b>GES</b>	<b>Greedy Equivalence Search</b>
<b>HCS</b>	<b>Hill Climbing Search</b>
<b>TS</b>	<b>Tabu Search</b>
<b>GS</b>	<b>Grow Shrink</b> algorithm
<b>TD</b>	<b>True Distribution</b>

<b>ED</b>	<b>E</b> mpirical <b>D</b> istribution
<b>CPTs</b>	<b>C</b> onditional <b>P</b> robability <b>T</b> ables
<b>RS</b>	<b>R</b> ejection <b>S</b> ampler
<b>iid</b>	<b>i</b> ndependent and <b>i</b> dentical <b>d</b> istributed
<b><i>BNMCMC</i></b>	<b>B</b> ayesian <b>N</b> etwork <b>M</b> arkov <b>C</b> hain <b>M</b> onte <b>C</b> arlo package
<b>w.r.t</b>	<b>W</b> ith <b>R</b> egard <b>T</b> o

# List of Symbols

$t$	discrete time (sampling time or iteration time)
$\mathbb{N}$	set of discrete-times (sampling times or iteration times)
$\mathbf{X}$	a sequence of random variables
$X$	random variable
$X_t$	the state of a process at time $t$
$X_0$	initial state
$\mathcal{X}$	the state space (search space or graph space)
$p_{ij}^t$	probability to move from state $i$ to state $j$ in $t$ steps
$f$	target probability distribution (posterior distribution)
$q$	proposal probability distribution
$\alpha$	parameter of Dirichlet distribution - vector or scalar
$G$	a graph
$G_0$	initial graph
$H$	candidate graph
$G'$	adjacent graph
$E$	set of directed edges
$e$	a single directed edge in $E$
$V$	set of variables (nodes)
$v$	a single node in $V$
$n$	number of nodes in a BN
$r$	number of state values
$p$	path in a space of graphs

$\lambda$	maximum number of graphs permitted on a path
$\ell$	an integer sampled uniformly between 1 and $\lambda$
$p_\ell$	a path in graph space containing $\ell$ graphs
$\mathcal{P}$	set of paths
$A_a$	list of addable edges
$D_a$	list of deletable edges
$R_a$	list of reversible edges
$A_n$	list of non-addable edges
$D_n$	list of non-deletable edges
$R_n$	list of non-reversible edges
$\mathcal{G}_a$	set of adjacent graphs constructed based on $A_a$
$\mathcal{G}_d$	set of adjacent graphs constructed based on $D_d$
$\mathcal{G}_r$	set of adjacent graphs constructed based on $R_r$
$\mathcal{N}$	set of adjacent graphs, $\mathcal{N} = (\mathcal{G}_a \cup \mathcal{G}_d \cup \mathcal{G}_r)$
$\mu$	total number of adjacent graphs in $\mathcal{N}$
$U$	uniform value sampled between 0 and 1
$Pa(v)$	a set of parents of node $v$
$\Delta$	the collection of all sets of parents
$D(v)$	a set of descendants of node $v$
$D$	a set of data points
$O$	big O notation
$(i, j)$	a pair of nodes
$\Omega$	the set of all possible edges
$\mathcal{J}$	number of parent configurations in a CPT
$k$	an indicator of a single state value
lag	a vector of lags at which to calculate the SSD
$T_a[i, j]$	transition between two adjacent graphs by adding an edge

$T_d[i, j]$	transition between two adjacent graphs by deleting an edge
$T_r[i, j]$	transition between two adjacent graphs by reversing an edge
$\mathcal{E}(G)$	set of all addable, deletable and reversible edges, $\mathcal{E}(G) = (A_a \cup D_d \cup R_r)$
$\xi(v)$	total number of conditional probabilities in a CPT for a particular node $v$
$\xi(G)$	total number of conditional probabilities in a graph $G$
$m$	number of observations for a particular node
$\omega$	number of parents for a particular node
$T_N$	number of cell explorations in the data matrix for a single cell in a CPT
$T_v$	number of cell explorations in the data matrix for the entire CPT of a node
$T_G$	number of cell explorations in the data matrix for the entire CPTs of a graph

# Chapter 1

## Thesis Objectives

### 1.1 Introduction

The title of this thesis combines three subject areas: Markov chain Monte Carlo (MCMC) approach, Bayesian networks (BNs), and their applications in systems biology. Here, the MCMC approach is used to sample Bayesian network structures from a defined graph space to infer causality directly from observational biological dataset. A commonly challenging problem that arises while inferring BNs is the high possibility for a sampling approach to get stuck in a local maxima solution. Finding an efficient sampling algorithm to substantially resolving this shortcoming remains a significant open problem.

This thesis proposes MCMC samplers that have the potential to effectively inferring BNs when the problem of local maxima is present. The proposed MCMC samplers, in this thesis, are new instances of the Neighbourhood Sampler (NS), Hit-and-Run (HAR) sampler, and Metropolis-Hastings (MH) sampler.

Section 1.2, Section 1.3 and Section 1.4 in this chapter overview the motivation of thesis, original contributions, and chapters abstracts, respectively.

The structure of the thesis is as follows. Chapter 2 presents a literature review of MCMC sampling including the process of Markov chain and the principle of Monte Carlo simulation. Chapter 3 reviews the main concepts, definitions, features, and



search space of BNs, as well as outlines certain MCMC and non-MCMC approaches that have been widely cited in the literature. Chapter 4 explains in detail how one can implement the NS, HAR and MH methods to infer BNs from their discrete spaces. Chapter 5 proposes the NS and HAR methods as new MCMC approaches to simulate BNs uniformly, and compares their performances with the MH sampler. Chapter 6 proposes two adaptive techniques proposed in this thesis to reduce the time complexity required by MCMC samplers to enumerate adjacent graphs and calculate the scoring function for a Bayesian network. Chapter 7 provides five applications involving 4, 6, 8, 11, and 51 nodes to evaluate samplers. Chapter 8 introduces a Graphical User Interface developed in this thesis to enable practitioners and academic researchers from different disciplines to implement the MCMC samplers proposed in this thesis. Chapter 9 ends with conclusions and possible future work.

## 1.2 Motivations

A new version of the NS was recently proposed in [1]. The sampler was introduced with a number of features that distinguish it from other samplers that also simulate over neighboring graphs, e.g. the MH sampler. The features of the NS are discussed in Chapter 4. This thesis provides a new instance of the NS to infer BNs from discrete graph spaces.

The HAR sampler is one of the fastest and most efficient MCMC algorithms for continuous spaces. The core idea of the HAR sampler is to "hit" a particular point in a continuous search space and then "run" in a random direction to another point determined by a distribution. This thesis attempts to follow the same process by hitting a particular graph in a discrete space and then running along a random path defined as a sequence of adjacent graphs.

The MH sampler is a popular MCMC method. To make the MH sampler comparable with the NS and HAR, we implement the samplers using the discrete uniform distribution as their proposal distribution, and a Dirichlet-multinomial distribution as their target function.

To reduce the time complexity while using the samplers to learn BNs, two adaptive techniques were developed. The potential of the three MCMC samplers and the two adaptive techniques was mainly examined to simulate BNs uniformly and to infer BN structures from discrete dataset and search spaces.

### 1.3 General objectives and original contributions

The chapters that include original contributions are Chapter 4, Chapter 5, Chapter 6, Chapter 7, and Chapter 8. Table 1.1 describes the contributions of these chapters.

Chapter	Contributions
4	explains theoretically how to modify the NS, HAR and MH algorithms to infer BNs.
5	implements the NS, HAR and MH to infer BNs and compares samplers' performances.
6	proposes two adaptive techniques to quickly enumerate adjacent graphs and to calculate the scoring function of posterior probability distribution.
7	applies the NS, HAR and MH to a variety of applications in systems biology, and compares samplers' performances. It also proposes a new approach to effectively define initial BNs.
8	deploys a new graphical user interface developed using C# to infer Bayesian network structures.

TABLE 1.1: Chapters that include original contributions.

## 1.4 Chapters abstracts and objectives

The subsequent sections outline the main specific objective for each chapter, and its contribution to the thesis.

### **CHAPTER 2: Markov Chain Monte Carlo Sampling**

This chapter attempts to provide a self-contained overview of Markov chain Monte Carlo (MCMC) sampling. This includes highlighting the main properties of Markov chains and the principles of Monte Carlo simulation. It also considers some MCMC samplers of interest, and discusses their relative features and limitations. The chapter covers the main issues that may arise when using MCMC methods, as well as some reliable diagnostic tests that are widely used to evaluate MCMC outputs.

### **CHAPTER 3: A Review of Bayesian Networks**

This chapter not only reviews the main notations and definitions related to BNs, but also reviews the approach of Bayesian inference to compute posterior distributions over graph spaces. Further, it explains how to fully learn a Bayesian network given some data-points using the Bayesian inference and MCMC method. The chapter explicitly outlines two essential types of learning to infer BNs: structure learning and parameter learning. A review of current MCMC and non-MCMC methods proposed to infer BNs is also provided.

### **CHAPTER 4: New Versions of the HAR, MH and NS to Simulate Bayesian Networks**

Sampling over a space of BNs is a useful stochastic technique to infer causality for a set of variables given their dataset. This chapter explains how to infer BNs from discrete spaces using the NS, HAR and MH algorithms, including proposal

distribution, posterior distribution, constructing a set of adjacent graphs, generating a candidate graph, initialising starting graphs, and how to use a standard brute-force approach to check acyclicity and connectivity for a sampled connected BN.

## **CHAPTER 5: Uniform Sampling of Bayesian Networks Conditional on Vertex Connectivity**

This chapter validates the proposed MCMC samplers by simulating uniformly from small spaces of BNs. It then compares the computational efficiency of the NS and HAR sampler with the MH sampler, by mainly testing whether a MCMC sampler is able to explore the entire space and whether the sampled graphs match the target uniform distribution. Experimentally, the convergence behaviour for each MCMC sampler has been investigated by performing simulations with the number of iterations ranging from 1000 up to 50 000 000. The experimental results in this chapter have demonstrated the ability of the proposed samplers to generate BNs uniformly. That is, a random sequence of BNs with a large number of iterations would ultimately follow a uniform pattern, in which the probability of sampling a specific Bayesian network is the proportion of those BNs in the target space.

## **CHAPTER 6: Adaptive Techniques for Faster Adjacent BNs Assignment and Function Scoring**

This chapter proposes two adaptive techniques to accelerate inference while simulating BNs:

- An adaptive method for faster adjacent graphs enumeration: this section proposes a new adaptive technique to quickly define the next set of all possible adjacent graphs  $\mathcal{N}(G')$  of graph  $G'$  given the current  $\mathcal{N}(G)$  of graph  $G$ , where

$G' \in \mathcal{N}(G)$ . Unlike the conventional brute-force approach, the new adaptive technique does not need to check every single pair of nodes for a given graph. Although, the new adaptive algorithm may have a worst-case execution  $O(V^4)$  as the standard brute-force approach, it still achieves a better speed up in practice.

- Dynamic scoring function for learning Bayesian network parameters: parameter learning in a discrete Bayesian network is another time-consuming problem. This section aims to adaptively populate the conditional probability tables (CPTs) for a given Bayesian network. It updates the CPTs only for the nodes whose parent nodes have been changed after a single transition from  $G$  to  $G' \in \mathcal{N}(G)$  i.e. it updates only the probabilities of affected variables in the new structure. The chapter also discusses the main factors that make a CPT computationally expensive, and describes the Big-O complexity of the adaptive technique compared to the conventional method. The section is supported by a range of illustrative examples, propositions and lemmas.

## **CHAPTER 7: Applications of Bayesian Networks in Systems Biology Using NS, HAR and MH Algorithms**

MCMC methods for BNs inference are ideally suited to predict a Bayesian network structure of cell signaling pathways. This chapter assesses the NS, HAR and MH methods through simulation studies. It applies the proposed MCMC samplers to analyse some biological BNs, and to infer cause-effect relationships among a set of interacting variables using five BNs of size 4 up to 51 nodes. The datasets used to infer BNs are either real-life observations or simulated using known conditional probability tables of true structures.

## **CHAPTER 8: *BNMCMC*: A New Graphical User Interface to Infer Bayesian Networks Using MCMC Samplers**

This software chapter presents a graphical user interface developed in this thesis to infer BNs using the proposed MCMC samplers. It is a user-friendly environment designed to facilitate analysis by practitioners from different disciplines. Currently, this software is only a desktop version, but in future, I would like to distribute it as a web-based application for better reachability among the scientific community.

## **CHAPTER 9: Conclusion**

This chapter describes some possible future work to be done to the current version of *BNMCMC* package, and future theoretical work following on from the thesis.

# Chapter 2

## Markov Chain Monte Carlo Sampling

### 2.1 Introduction

Sampling from a complex model is not just used as an optimization tool when deterministic methods are unavailable (e.g. *junction tree* [2], *cutset conditioning* [3], *variable elimination* [4], and *systematic maximum a posteriori assignment (MAP) search* [5]). It is also used to estimate marginal distributions, quantify uncertainty in a posterior distribution and estimate integrals.

The Markov chain Monte Carlo (MCMC) approach [6–14] is a form of stochastic sampling. In 1953, [15] developed the first MCMC approach called the Metropolis algorithm in the field of statistical physics. The MCMC algorithm provides a framework for sampling from complicated probability distributions. MCMC sampling generates a Markov chain which is constructed specifically to converge to the target probability distribution  $f$ . The idea is to iteratively generate random points from a proposal distribution  $q$ , and accept or reject the proposed point, thus forming a random walk. Formally,  $f$  is the steady-state distribution of a Markov chain. The empirical distribution of the observed states converges weakly to  $f$ .

This chapter is constructed as follows. Section 2.2 highlights the main notations

and properties of Markov chains. Section 2.3 outlines the principles of Monte Carlo simulation and briefly reviews some of its main methods. Section 2.4 considers some MCMC samplers of interest and their special cases. Section 2.5.1 frames the main issues that may arise when using MCMC methods. Section 2.5.2 introduces some reliable diagnostic tests that are widely used to evaluate MCMC outputs.

## 2.2 Markov chains

I follow standard approaches to define a Markov chain [8, 16–22], as a basis for exploiting the Markov property in constructing dependent Monte Carlo simulations (Section 2.3). A stochastic process [18, 23, 24] in discrete time  $t \in \mathbb{N} = \{0, 1, 2, \dots\}$  is a sequence of random variables  $X_0, X_1, X_2, \dots$  denoted by  $\mathbf{X} = \{X_t : t \geq 0\}$ , where the value  $X_t$  refers to the state of the process at time  $t$ , and  $X_0$  denotes the initial state. The state space  $\mathcal{X}$  is defined as the collection of all possible values that the  $X_t$  can take.

A stochastic process in a countable state space is called a Markov chain if Equation 2.1 is satisfied for all times  $t \geq 0$  and all states  $i_0, \dots, i_t, j \in \mathcal{X}$ .

$$P(X_{t+1} = j | X_t = i_t, X_{t-1} = i_{t-1}, \dots, X_0 = i_0) = P(X_{t+1} = j | X_t = i_t). \quad (2.1)$$

**Definition 1** (Markov property). The Markov property asserts that future states, given the present state, only depend on that present state  $X_t$  and are conditionally independent of past states.

I write  $p_{ij}^{(t)}$  to describe the probability of transition from state  $i$  into state  $j$  in  $t$  steps, such that  $p_{ij}^{(t)} = P(X_t = j | X_0 = i)$ . The matrix  $P = (p_{ij})$  is called a *transition kernel* (or Markov kernel). Note that  $p_{ij}$  is a function that takes values between 0 and 1 to denote the probability that the chain, whenever in state  $i$ , moves next (one unit of time later) into state  $j$ , and is referred to as a *one-step transition probability*.



**Definition 2.** A Markov chain  $X_t$  is said to be *time-homogeneous* if Equation 2.2 holds, so that transition probabilities are independent of  $t$ .

$$P(X_{t+1}|X_t) = P(X_1|X_0), \quad \forall t. \quad (2.2)$$

### 2.2.1 Irreducible, aperiodic and recurrent Markov chain

This section reviews three important properties of Markov chains: *irreducibility*, *aperiodicity*, and *positive recurrence*. If a Markov chain satisfies these properties, it is then called *ergodic* (Theorem 2.2.1), which is a sufficient condition to converge to a *stationary distribution* (Definition 12). Definitions 3 and 4 describe when a single state and Markov chain are irreducible, respectively. Definitions 5 and 6 describe when a single state and Markov chain are aperiodic, respectively. Definitions 9 and 10 describe when a single state and Markov chain are positive recurrent, respectively.

**Definition 3.** A state  $j$  is said to be accessible from state  $i$  if there is an integer  $t \geq 0$  such that

$$P(X_t = j | X_0 = i) = p_{ij}^{(t)} > 0. \quad (2.3)$$

When  $p_{ij}^{(t)} = 0$  for all  $t$ , then state  $j$  is not accessible from state  $i$ .

**Definition 4.** A Markov chain is said to be *irreducible* if it is possible to transit from any particular state  $i$  to any other state  $j$ , in a finite number of steps, with positive probability.

**Definition 5.** A state  $i$  is *periodic* if returning to state  $i$  can only occur at regular times. The period of a state is formally defined as

$$s_i = \gcd\{t : P(X_t = i | X_0 = i) > 0\}, \quad (2.4)$$

where "gcd" is the greatest common divisor. Conversely, a state  $i$  is said to be *aperiodic* if  $s_i = 1$ .

**Example 1.** Suppose it is possible to return to a particular state in  $\{6, 9, 12, 15, \dots\}$  time steps; here  $s = 3$ .

**Definition 6.** A Markov chain is *aperiodic* if every state is aperiodic.

**Proposition 1.** An *irreducible* Markov chain only requires one aperiodic state to imply all states are aperiodic [25].

**Definition 7.** A state  $i$  is said to be *transient* if, with positive probability, one can start from  $i$  and never return to it. Otherwise, the state  $i$  is said to be *recurrent*.

**Definition 8.** An irreducible Markov chain is said to be *recurrent* if and only if all states are recurrent, *transient* otherwise.

**Proposition 2.** The expected number of visits to the state  $i$  is finite if state  $i$  is transient [26] i.e.

$$\sum_{t=0}^{\infty} p_{ii}^{(t)} < \infty \quad (2.5)$$

and infinite if state  $i$  is recurrent i.e.

$$\sum_{t=0}^{\infty} p_{ii}^{(t)} = \infty \quad (2.6)$$

**Proposition 3.** Let  $T_i$  be the first return time to state  $i$  [18]:

$$T_i = \min\{t \geq 1 : X_t = i | X_0 = i\}, \quad (2.7)$$

the *mean recurrence time* at state  $i$  is defined as the expected return time  $\mu_i$ :

$$\mu_i = E[T_i]. \quad (2.8)$$

Given the value of  $\mu_i$  in Proposition 3, a recurrent state is either *null recurrent* or *positive recurrent*.

**Definition 9.** A state  $i$  is said to be *positive recurrent* if  $\mu_i$  is finite, that is,  $\mu_i < \infty$ . Otherwise, the state  $i$  is *null recurrent* i.e.  $\mu_i = \infty$ .

**Definition 10.** A Markov chain is said to be *positive recurrent* if all states in an irreducible Markov chain are positive recurrent.

**Theorem 2.2.1.** A state  $i$  is said to be ergodic if it is aperiodic and positive recurrent. If all states in an irreducible Markov chain are ergodic, then the chain is said to be ergodic.

## 2.2.2 Limiting distribution

**Definition 11.** The probability distribution  $f = [f_0, f_1, f_2, \dots]$  is called the limiting distribution of the Markov chain  $X_t$  if

$$f_j = \lim_{t \rightarrow \infty} P(X_t = j | X_0 = i), \quad \forall i, j \in \mathcal{X}, \quad (2.9)$$

provided that this limit exists and  $\sum_{j \in \mathcal{X}} f_j = 1$ .

**Proposition 4.** If the limiting distribution in 2.9 exists, it then does not depend on the initial state ( $X_0 = i$ ) [27] i.e

$$f_j = \lim_{t \rightarrow \infty} P(X_t = j), \quad \forall j \in \mathcal{X}. \quad (2.10)$$

**Definition 12.** A discrete-time stochastic process  $\{X_t : t \geq 0\}$  is *stationary* if for any time points  $i_1, \dots, i_t$  and any  $m \geq 0$ , the joint distribution of  $(X_{i_1}, \dots, X_{i_t})$  is the same as the joint distribution of  $(X_{i_1+m}, \dots, X_{i_t+m})$ .

The term stationary in Definition 12 refers to stationary in time. Note that the distribution of  $X_t$  is the same for all  $t$ .

A special case of a positive recurrent state is an *absorbing* state. An absorbing state is such that when a stochastic process enters that state, it becomes impossible to leave it again, so  $p_{ii} = 1$ . Note that an irreducible Markov chain is positive recurrent if and only if there exists a stationary distribution. Further, a Markov chain is an absorbing chain if and only if the chain contains at least one absorbing state, and it is possible to go from a non-absorbing state to an absorbing state, though not necessarily in one step.

**Proposition 5.** If a Markov chain is irreducible and aperiodic, then Expression 2.11 has a unique solution [20, 21]:

$$f = f \cdot P, \quad (2.11)$$

here, the unique solution is the limiting distribution of the Markov chain as in Equation 2.10, and moreover this distribution is stationary [28].

### 2.2.3 Detailed balance property

A sufficient condition for a probability density function (pdf)  $f$  to be stationary for a Markov process is the *detailed balance* property. The detailed balance can be expressed as

$$f_i \cdot p_{ij} = f_j \cdot p_{ji}, \quad (2.12)$$

where  $f_i$  and  $f_j$  are the equilibrium probabilities of being in states  $i$  and  $j$ , respectively. A Markov chain is said to be *reversible* if there is a pdf  $f$  over its states that satisfies the condition in 2.12 for all times  $t$  and all states  $i$  and  $j$ . Reversibility can also be defined based on the Kolmogorov criterion which states that for any closed loop of states, the product of transition rates must be the same. This satisfies [29]

$$p_{12} \cdot p_{23} \cdot \dots \cdot p_{(j-1)j} \cdot p_{j1} = p_{1j} \cdot p_{j(j-1)} \cdot \dots \cdot p_{32} \cdot p_{21}, \quad (2.13)$$

for all finite sequences of states  $1, 2, \dots, j \in \mathcal{X}$ .

## 2.3 Monte Carlo simulation

The transition rule for a Markov chain is often given and the task is to determine the stationary distribution. In contrast, Markov chain Monte Carlo simulation prescribes a target stationary distribution and the task is to develop a suitable transition rule often involving a proposal distribution. The Markov chain Monte Carlo approach was first developed by physicists. Monte Carlo methods typically produce identical independent distributed (iid) samples and are free from the problem of auto-correlated samples inherent in MCMC methods. A direct application of Monte Carlo simulation is to estimate integrals given a randomly distributed variable  $x$ . Suppose we have a complex single-dimensional integral:

$$G = \int_a^b f(x)dx. \quad (2.14)$$

Now attempt to decompose  $f(x)$  into the product of two functions as follows

$$G = \int_a^b f(x)dx = \int_a^b p(x)h(x)dx, \quad (2.15)$$

where  $p(x)$  is an arbitrary function and  $h(x)$  is a probability density defined over the interval  $(a, b)$ . The resulting integral in expression 2.15 can be expressed as the expectation of  $p(x)$  over the density function  $h(x)$

$$E_{h(x)}[p(x)] \approx G_t = \frac{1}{t} \sum_{i=1}^t p(x_i), \quad (2.16)$$

where the  $x_i$  are drawn from the density function  $h(x)$ , so that as  $t \rightarrow \infty$ ,  $G_t \rightarrow G$  by the ergodic property in Theorem 2.2.1. One main drawback with Monte Carlo integration is that it is difficult to obtain samples from complicated distributions.

In some cases, a cumulative distribution function (cdf)  $F(x)$  may be easier to work with than the associated density function  $f(x)$ , and in fact the latter might not even exist. The inverse transform sampler (ITS) provides a way to generate a one-dimensional random variable when the cdf is available. Let  $X$  be a continuous random variable with cdf  $F(x)$ . Suppose one wants to sample values  $X$  that are distributed according to  $F(x)$ . One can sample from  $f(x)$  if the inverse cumulative distribution function can be formulated as:

$$x = F^{-1}(u). \quad (2.17)$$

Simply generate a uniform random number  $u$  from  $U(0, 1)$  and compute Equation 2.17. One of the limitations of ITS is that it requires a closed form expression of  $F(x)$ , which is not always available for some desirable distributions (e.g the Gaussian distribution). The rejection Monte Carlo sampler was proposed to overcome this limitation [30].

The rejection sampler (RS) [13, 30, 31] is a pseudo-random number sampling technique. The sampler forms the basis for many Monte Carlo algorithms. The RS can be applied in problems where the goal is to approximate integrals with respect to (w.r.t.) the pdf of interest [32]. One may use a proposal (for instance a proxy) distribution  $q(x)$  in order to sample from a target probability distribution  $f(x)$ . The proposal distribution  $q(x)$  must *envelop* (cover) the target probability distribution  $f(x)$ . That is,  $f(x) < Mq(x)$ , where  $M > 1$  is an appropriate upper bound on  $\frac{f(x)}{q(x)}$ . To carry out the RS, sample  $x$  from  $q(x)$ , and  $u$  uniformly from  $(0, 1)$ . If the inequality in Equation 2.18 holds, accept  $x$ . Otherwise, reject the value of  $x$  and repeat until acceptance occurs [13].

$$\frac{f(x)}{Mq(x)} > u. \quad (2.18)$$

The RS does not require evaluating the normalising constant of either  $f(x)$  or  $q(x)$ . However, in high dimensional spaces it is hard to efficiently design a rejection sampling method for a particular distribution. Further, there may be a difficulty in determining a suitable analytic form for the envelope distribution since it requires identifying a suitable value for  $M$ .

Adaptive rejection sampling (ARS) refines RS to provide a method that can overcome these problems. ARS [32, 33] starts with a trial proposal distribution  $q_0(x)$ . The idea is to continue improving the proposal distribution as long as we do Monte Carlo sampling; so when we iterate the procedure, we get a sequence of proposal functions that converge to the target pdf [32] while the proportion of accepted samples grows. The simplest form of ARS is restricted to log-concave densities [32]. Typically, two common approaches can be adopted to apply ARS: derivative-free ARS [33] and derivative-based ARS [32]. Essentially, for both approaches, the log-concavity condition should be checked for  $f(x)$ , i.e.  $V(x)$  in Equation 2.19 is strictly concave  $\forall x \in D$ , where  $D \subseteq \mathbb{R}$  is the domain.

$$V(x) = \log(f(x)). \quad (2.19)$$

Since ARS is restricted to log-concave densities, it is unsuitable for many practical applications [34, 35]. The main advantage of using ARS is to obtain a higher mean acceptance rate by reducing the chance of rejection for the ensuing iterations.

RS can be regarded as a special case of importance sampling (IS) [36], and the relationship between RS and IS [37] was studied in [38], with emphasis on their relative efficiencies. IS is not a method to draw samples from a probability distribution  $f(x)$ , it is rather a discrete method for approximating expectations

directly  $E_f[h(x)]$ .

$$\begin{aligned} I[h] &\equiv E_f[h(x)] = \int h(x)f(x)dx \\ &= \int h(x)\frac{f(x)}{q(x)}q(x)dx \\ &\approx \frac{1}{t} \sum_{i=1}^t h(x_i)\frac{f(x_i)}{q(x_i)}. \end{aligned} \quad (2.20)$$

where the  $x_i$  are drawn from a distribution with density  $q(x)$ . Then, one can write

$$\hat{I}[h] = \frac{1}{t} \sum_{i=1}^t h(x_i)w(x_i); \quad w(x_i) = \frac{f(x_i)}{q(x_i)}, \quad (2.21)$$

where  $w(x_i)$  called the importance weight.

An important feature of IS is the potential to reduce variance of  $\hat{I}[h]$ , by selecting an appropriate  $q(x)$ . It also provides an unbiased approximation, since  $E\{\hat{I}[h]\} = I[h]$ . The importance weights play an important role in the method. However, these weights should be calculated accurately, otherwise, poor estimates will be obtained.

Simulated annealing (SA) [39] is a Monte Carlo method for optimisation. Note that the SA is actually a modified MCMC method, but with a variable target probability distribution. In particular, the varying target probability distribution converges to a point mass located at the optimal solution, and is actually one of the few optimisers with provable convergence to the global maximum, however slow. This method is inspired by the annealing method used in heat treatment of metals, where slow cooling cycles are maintained alternately with processes of re-heating, thus providing access to a configuration with minimal energy.

SA was independently described by [39] to search for feasible solutions and ultimately converge to an optimal solution. For an easy description of SA, the reader is referred to the textbook by [40]. For further information see [41, 42] who have written monographs on the subject of SA. Also, a brief introduction to the actual



mechanics of simulated annealing can be found in [43].

## 2.4 Markov chain Monte Carlo methods

Markov chain Monte Carlo (MCMC) methods are used to sample from arbitrary probability distributions  $f(x)$  [44]. By construction, MCMC sampling exploits dependent samples which enable it to simulate from difficult distributions, but this comes at a cost. It is useful in Bayesian inference, for example, where we need to integrate over possibly high-dimensional probability distributions to make inference about a model or to make predictions. MCMC methods play an indispensable role in pattern recognition. Theoretical and applied treatments of MCMC methods can be found in [6, 8, 11–13, 45–48].

The MCMC approach has clearly the disadvantage that it generates correlated samples due to Markov chain process. Further, the initial samples are more highly influenced by the arbitrary starting points than other non-MCMC methods, so they may follow a different distribution. These limitations are addressed in Section 2.5.1 and convenient diagnostic tests for analysing MCMC outputs are provided in Section 2.5.2.

The transition kernel must ensure rapid convergence to  $f(x)$  and good mixing i.e. moves throughly between high and low densities, for the efficiency of the method. However, in multimodal target probability distributions, some simple kernels of MCMC methods may fail to produce fast convergence to the target probability distribution [12].

### 2.4.1 Metropolis-Hastings method

The Metropolis algorithm was first introduced in [44]. It aims to generate a sequence of draws from some desired probability distribution (target distribution)

$f(X) = \frac{p(X)}{z}$ , where  $p(x)$  is proportional to  $f(x)$  and  $z$  is the normalisation factor. This method was first developed by [15] inspired by a computer simulation model of physical simulated annealing. Their method incorporates a temperature of the system, and calculates the Boltzmann average of a property of the system (see Boltzmann distribution law in [15]). The Metropolis algorithm only considers *symmetric* proposals i.e.  $q(Y|X)=q(X|Y)$ , where  $q$  is referred to as the *proposal density*, candidate-generating density or jumping distribution. One possible proposal density is the Gaussian distribution proposed in [49]. A nice introduction and background to the Metropolis algorithm can be found in [50].

One main feature of using the Metropolis algorithm is that since the computation depends only on the ratio  $\frac{f(Y)}{f(X)}$ , the normalisation factor  $z$  cancels. Thus, there is no need to calculate  $z$ , which is often difficult to compute in practice [17]. Instead, one needs only to calculate ratios of the form  $\frac{p(Y)}{p(X)}$ .

The Metropolis-Hastings (MH) sampler is a widely used MCMC simulation technique. The MH sampler was first presented in [17]. The MH sampler uses a Markov process which *asymptotically* reaches a unique stationary distribution  $f(X)$  [13]. Particularizations of the MH sampler include the Metropolis algorithm [15], Metropolised Independence sampler [17] and Gibbs Sampling [51] as pointed out by [52]. The MH algorithm can also be used within other samplers such as Metropolis-within-Gibbs (MWG) [19] and Adaptive Rejection Hastings-Metropolis (ARHM) [34]. Thus the MH algorithm is an important MCMC sampler to be compared with any suggested MCMC sampler. A nice tutorial to the concept of Metropolis-Hastings sampler and its associated methods can be found in [53].

Unlike the Metropolis algorithm, MH sampler can use asymmetric proposal distributions [54]. The MH sampler can also be used for sampling from multi-dimensional distributions, even when the number of dimensions is high. It is also a convenient way to do MCMC sampling when the normalisation factor is hard to

compute. The MH sampler is described in Algorithm 1 [53]. If the Metropolis algorithm is implemented, the fraction  $\frac{q(X_t|Y)}{q(Y|X_t)}$  in Equation 2.22 should be set to one. That is, the proposed state  $Y$  is accepted with probability  $\alpha$  expressed in Algorithm 1, where  $\frac{f(Y)}{f(X_t)}$  is the likelihood ratio, and  $\frac{q(X_t|Y)}{q(Y|X_t)}$  is the ratio of the proposal density.

---

**Algorithm 1** Metropolis-Hastings method

---

Given a starting element  $X_t = X_0 \in \mathcal{X}$  and set  $t = 0$ .

1. Generate a candidate element  $Y$  from some proposal distribution  $q(Y|X_t)$ .

2. Take:

$$X_{t+1} = \begin{cases} Y & \text{with probability } \alpha(X_t, Y), \\ X_t & \text{with probability } 1 - \alpha(X_t, Y), \end{cases}$$

where

$$\alpha(X_t, Y) = \min \left( \frac{f(Y)q(X_t|Y)}{f(X_t)q(Y|X_t)}, 1 \right). \quad (2.22)$$

3. Set  $t = t + 1$ . Go to 1.

---

As with the Metropolis algorithm, the proposal distribution has to be tuned to achieve an efficient sampling algorithm. Although, the limiting distribution of MH sampler is  $f(X)$ , convergence can be slow [8, 55], especially when the number of dimensions is high.

### 2.4.2 Neighbourhood Sampler

The Neighbourhood Sampler (NS) is a powerful MCMC method introduced in [1] to sample from continuous probability distributions. Each iteration involves sampling from a local neighbourhood  $\mathcal{N}_X$  uniquely associated with a particular element  $X$ . The sampler requires that  $X \in \mathcal{N}_X$  for all  $X \in \mathcal{X}$ . These neighbourhoods are

selected in such a way that a reflexive property holds, so that  $X \in \mathcal{N}_Y$  if and only if  $Y \in \mathcal{N}_X$ , and thus it is appropriate to refer to such an  $X$  and  $Y$  as neighbours. One useful feature is that each iteration of the NS involves sampling uniformly from such neighbourhoods, no matter how complicated the target probability distribution. The NS assumes a target probability distribution function  $f(X)$  is defined over a finite space  $\mathcal{X}$ , with a counting measure  $\mu$  also defined on  $\mathcal{X}$ . Formally, let  $\mathbb{S}$  be defined on a measurable space  $\mathcal{X}$ . Let sigma-algebra  $\Sigma$  be a measurable subsets consisting all the subsets of  $\mathbb{S}$ . Then, the counting measure  $\mu$  on this measurable space  $(\mathbb{S}, \Sigma)$  is the positive measure defined by [56]

$$\mu(A) = \begin{cases} |A| & \text{if } A \text{ is finite} \\ +\infty & \text{if } A \text{ is infinite} \end{cases}$$

for all  $A \in \Sigma$ , where  $|A|$  denotes the cardinality of the set  $A$  [56].

One advantage that the NS has relative to the closely related random walk sampler is that it is less likely to become trapped in a local mode. The reason for this is that at each iteration the NS transits through two elements consecutively. From element  $X$  it transits to an element  $Y$ , which is sampled uniformly from the set  $\mathcal{N}_X$ . It then transits to element  $Z$  which is sampled uniformly from  $\mathcal{N}_Y$ . Then, a rejection step is applied to reduce  $\mathcal{N}_Y$  until a particular  $Z$  in the reduced neighbourhood is accepted. That is, each rejected element  $Z$  is excluded from being sampled again from the same set  $\mathcal{N}_Y$ . This increases the chance of moving to a new element  $Z \neq X$  compared to the random walk sampler, although it is still possible to have  $Z = X$ , since  $X \in \mathcal{N}_Y$  [1]. Algorithm 2 describes the general NS for sampling from an arbitrary distribution  $f$  w.r.t.  $\mu$  [1].

---

**Algorithm 2** Neighbourhood Sampler
 

---

Given the current state  $X_t = X_0 \in \mathcal{X}$  and set  $t = 0$ :

1. Generate  $Y \sim \mathbf{U}(\mathcal{N}_{X_t})$  where  $\mathbf{U}(\mathcal{N}_{X_t})$  is the uniform distribution (w.r.t.  $\mu$ ) on  $\mathcal{N}_{X_t}$ . Set  $H = \mathcal{N}_Y$ .
  2. Generate  $U \sim \mathbf{U}(0, f(X_t)/\mu[\mathcal{N}_{X_t}])$ .
  3. Generate  $Z_1 \sim \mathbf{U}(H)$ .
  4. Set  $k = 1$  and iterate the following steps until  $f(Z_k)/\mu[\mathcal{N}(Z_k)] \geq U$ :
    - (a) Reduce  $H$  by excluding  $Z_k$ .
    - (b) Generate  $Z_{k+1} \sim \mathbf{U}(H)$  and set  $k := k + 1$ .
  5. Set  $X_{t+1} = Z_k$ .
  6. Set  $t = t + 1$ . Go to **1**.
- 

It is still possible for the NS to become stuck in a local mode for a long time. However, one possible technique to improve mixing is to expand each neighbourhood  $\mathcal{N}_X$  to include sequences that can be obtained from  $X$  by two successive transitions [1].

### 2.4.3 Hit-and-Run algorithm

The basic version of the Hit-and-Run (HAR) approach was first introduced by [57]. It involves a multivariate proposal and a Metropolis step for rejection. It was introduced to independently sample uniform points over a continuous convex space  $\mathcal{X}$ . The HAR sampler with a Metropolis rejection step was popularised by [58] and involves symmetric trial transitions such that its stationary distribution is uniform

on a bounded open space domain  $\mathcal{X}$ . The term “Hit-and-Run” was also proposed by [58], due to its ability to run across the search space, and hit distant elements. This feature provides good mixing and rapid convergence to the target probability distribution. The HAR sampler is included in the “LaplacesDemon” package [59] which is a freely contributed R package for Bayesian inference. The sampler was shown to be one of the fastest known methods to sample random points in a high dimensional convex set [60, 61]. It often achieves a more rapid rate of convergence than the Gibbs sampler [9], in particular when the parameters are correlated [62].

The core idea of sampling by the HAR algorithm is summarised in the following linear equation:

$$X_{t+1} = \lambda_t \cdot d_t + X_t, \quad (2.23)$$

where  $d_t$  is a random direction in the space at iteration  $t$ , and is often sampled uniformly, and  $\lambda_t$  is a real number used to scale the length of the direction  $d_t$ . Figure 2.1 illustrates the idea: first a direction  $d_t$  is selected as a point on a unit  $k$ -dimensional sphere in a convex space  $\mathcal{X}$ , then a new element is selected at a scale size  $\lambda$  units in the same direction. Equation 2.23 produces a Markov chain in which the probability distribution of the next point  $X_{t+1}$  conditionally depends only on the current point  $X_t$  and not on the sequence of other historical events that preceded  $X_t$ .

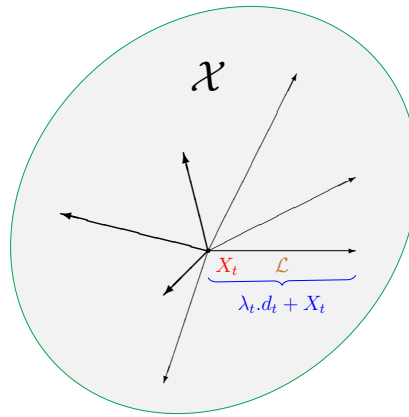


FIGURE 2.1: Some possible directions on the unit  $k$ -dimensional sphere.

Algorithm 3 [58, 63–65] iteratively specifies the main steps to perform the general HAR sampler with a target probability distribution  $f(x)$ . The HAR algorithm can be described as a *line sampler* to sample from  $f(X)$  on a convex space  $\mathcal{X}$ .

---

**Algorithm 3** Hit-and-Run algorithm

---

Choose a starting point  $X_t = X_0 \in \mathcal{X}$  and set  $t = 0$ :

1. Define a real length  $\lambda_t > 0$  at random.
2. Generate a uniformly distributed unit direction  $d_t := (d_t^1, \dots, d_t^k) \subseteq \mathbb{R}^n$ .
3. Find the intersection line  $\mathcal{L} = \{\mathcal{X} \cap Y | Y = X_t + \lambda_t d_t \in \mathcal{X}\}$ , where  $\lambda$  is a signed distance.
4. Select the candidate point  $Y$  according to a full defined distribution along  $\mathcal{L}$ .
5. Apply the MH acceptance ratio by taking:

$$X_{t+1} = \begin{cases} Y & \text{with probability } \alpha(X_t, Y), \\ X_t & \text{with probability } 1 - \alpha(X_t, Y), \end{cases}$$

where  $\alpha(X_t, Y)$  is as in Equation 2.22.

6. Set  $t = t + 1$ . Go to 1.
- 

At each iteration  $t$  in Algorithm 3, the HAR sampler first randomly generates a direction  $d_t$  in a convex  $\mathcal{X} \subseteq \mathbb{R}^k$  as a unit vector of dimension  $k$ . The generated direction is then scaled by a pre-determined length  $\lambda$ . A line segment  $\mathcal{L}$  is therefore defined by the intersection of the corresponding straight line passing through  $X_t$  in direction  $d_t$  of length  $\lambda$  and the gray oval convex space  $\mathcal{X}$  [66] (see Figure 2.1). Next, the candidate point  $X_{t+1}$  is selected according to the full defined distribution

along the intersection between the line  $\mathcal{L}$  and space  $\mathcal{X}$ . Then accept the candidate point if it satisfies the acceptance ratio defined in step 13 in Algorithm 3.

## 2.5 MCMC sampling issues and convergence diagnostic tests

Section 2.5.1 and Section 2.5.2 discuss two main aspects which are of interest when using MCMC sampling: MCMC convergence issues and MCMC convergence diagnostics, respectively.

### 2.5.1 MCMC sampling issues

Unlike some methods of Monte Carlo simulation, MCMC sampling produces dependent samples as a result of running a Markov chain process defined in Section 2.2. The dependency among samples leads to the problem of high autocorrelation which is an undesirable property, because it may slow convergence to the target probability distribution or cause poor mixing (i.e. the chain moves slowly in the state space). Other issues include how to effectively assign the following: a proposal distribution, number of chains, starting values, thinning, burn-in, stopping time, proper scale, and auxiliary variables. A brief summary and suggested solutions to these issues is outlined in the next few paragraphs.

A proposal distribution should ultimately generate samples from the target probability distribution [8]. Convergence to the target probability distribution may not be rapid, and can depend on the choice of proposal distribution. Finding a proposal distribution that produces efficient sampling can be difficult when we have multi-dimensional spaces, or many local minima or maxima. It is important to carefully choose the proposal distribution in order to facilitate sampling and evaluation [67].



It is possible to run a single chain only. However, various recommendations have been made in the literature, such as running many short chains [68], multiple long chains [69], or one very long chain [70]. It might be desirable to run many short chains to obtain independent samples from the target probability distribution. However, it should not be done unless there is a clear need for independent samples [8, 71]. Convergence diagnostics are still an area of active research [8]. However, running a single chain for one very long run can be more precise in finding new modes [8]. Running multiple short chains may result in none of them converging and thus is undesirable for inference. Reducing dependency on the initial values can be achieved by using a parallel chains technique. However, comparison between chains can never prove convergence [72].

Burn-in is, loosely speaking, the number of iterations that the chain takes to converge to the stationary distribution. Burn-in can be influenced by the proposal distribution. One can approximate the length of burn-in based on convergence diagnostics [73]. A *convergence diagnostic* is a tool to determine the length of burn-in. It uses theoretical and approximation methods to analyse Monte Carlo output [8]. One popular technique is the so-called trace plot or time-series plot. Using MCMC output, we plot the iterates of particular parameters and monitor trends [74]. Convergence can be assessed by looking for trends in these plots that suggest non-stationary behaviour. One can also monitor the autocorrelations between successive iterates, since highly correlated samples can lead to slow convergence. Further, model re-parametrisation techniques can play a significant role in speeding up the mixing.

For a Markov chain that is irreducible and aperiodic, the stationary distribution will not be affected by choosing any arbitrary starting value, since the stationary distribution is independent of starting values. A technique called "over-dispersed"

starting values was suggested by [69] to be used in multiple chains to assist in assessing convergence [72]. If running an MCMC from many widely dispersed initial states yield comparable sample densities, which is easily checked, one tend to conclude that the sampler works.

One can alternatively use other techniques to select initial values, such as simulated annealing [75], *ad hoc* methods [76], or maximum likelihood estimates once informative priors are available. However, starting values should be chosen more carefully for slow-mixing chains to avoid a lengthy burn-in. Section 7.5.2 proposes a new approach - based on heuristic search algorithms reviewed in Section 3.6.4 - to effectively define an initial network, in particular, when inferring medium or large BNs.

It is common practice to thin the MCMC output by discarding all samples except every  $k^{\text{th}}$  sampled value. The goal is to overcome the problem of high correlation between consecutive iterations. In the ecological literature, thinning MCMC output has been regarded as inefficient and unnecessary since the early 1990's [77]. However, thinning has been found to be a practical necessary. For example, suppose one can store no more than 10,000 samples. Then, it would be better to run the MCMC twice as long, keeping every second sample.

Practically, the number of samples used determines the precision of the estimator. One possible method to determine when to stop sampling is to run several chains in parallel with different starting values, and compare the estimates. Other proposed methods that aim to estimate the variance of estimators can be seen in [78] and [73].

For MH sampler with a proposal centred on the current sample, finding an appropriate scale for the proposal so that the variance is neither too small nor too large, often enables more efficient sampling. If the variance is too small, then all proposed values are likely to be accepted, but each step is small [79]. It was conjectured by [17] that an effective acceptance rate of about  $1/2$  is optimal. Consequently, the

algorithm may traverse the state space very slowly. On the other hand, if the variance is too large, then all proposed values are likely to be rejected and the algorithm stays trapped at the same place. A detailed description of the relationship between correlation and convergence can be found in [55].

Another implementational issue is use of an auxiliary variable to improve convergence and MCMC performance in the case of highly multi-modal target probability distributions. The idea of auxiliary variables was first introduced in the context of the Ising model by [80]. Basically, the auxiliary variables technique aims to add one or more variables  $u \in U$  to the state variable  $x$  of the Markov chain [10]. An augmented distribution of  $x$  and  $u$  can be defined by taking  $p(x)$  to be the marginal for  $x$ . The conditional  $p(u|x)$  can be specified arbitrarily. Then, a Markov chain on  $\mathcal{X} \times U$  can be constructed, which alternates between two types of transition at each iteration [81].

It is usually necessary to analyse the outputs by summarising them after obtaining samples from simulation. For example, the posterior distribution in Bayesian analysis can be summarised using one or more estimators such as means, variances, correlations, and marginal distributions. These estimators can be easily calculated based on the samples. Typically, we can estimate marginal distributions by kernel density estimation; for further details see [68].

### 2.5.2 Convergence diagnostics

In this thesis I adopted several MCMC diagnostics recommended in the literature [74, 82–84]. Some of these diagnostics are available in the package *coda* [85] and *boa* [86] for R.

I have used the Gelman and Rubin diagnostic [84] as a convergence test. This

test measures the difference between the within-chain variance and the between-chain variance using a value called the “scale reduction factor”. It requires simulating multiple chains ( $m \geq 2$ ) each of length  $2n$ , where  $n$  is the number of draws (samples), with overdispersed starting values. The first  $n$  samples in each chain are then discarded, and the within-chain and between-chain variances are evaluated.

The Gelman and Rubin diagnostic is implemented as follows. Let  $x_{ij}$  be draw number  $i$  in chain  $j$ . Find the within-chain variance  $W$  and between-chain variance  $B$ . Equation 2.24 and Equation 2.25 calculate  $W$  and  $B$ , respectively.

$$W = \frac{1}{m} \sum_{j=1}^m s_j^2 \quad (2.24)$$

where

$$s_j^2 = \frac{1}{n-1} \sum_{i=1}^n (x_{ij} - \bar{x}_j)^2 \quad \text{and} \quad \bar{x}_j = \frac{1}{n} \sum_{i=1}^n x_{ij}$$

$$B = \frac{n}{m-1} \sum_{j=1}^m (\bar{x}_j - \bar{\bar{x}})^2 \quad (2.25)$$

where

$$\bar{\bar{x}} = \frac{1}{m} \sum_{j=1}^m \bar{x}_j$$

Next, find the weighted sum  $\widehat{Var}(x)$  of  $W$  and  $B$  using Equation 2.26.

$$\widehat{Var}(x) = \left(1 - \frac{1}{n}\right)W + \frac{1}{n}B \quad (2.26)$$

Then, use Equation 2.27 to calculate the potential scale reduction factor  $\hat{R}$ .

$$\hat{R} = \frac{\widehat{Var}(x)}{W} \quad (2.27)$$

The output of the Gelman and Rubin test consists of the 50% and 97.5% quantiles of the distributions of scale reduction factors. If these quantiles are both close to 1,

the chains may be considered to be sampling from the same distribution. If both quantiles are high (commonly greater than 1.1 or 1.2), the number of iteration  $t$  needs to be increased.

Another MCMC test is the Geweke diagnostic [82], which takes two non-overlapping parts of the Markov chain (usually the first 10% and last 50%, as suggested by Geweke [82]). If the two means in the two time intervals are not significantly different, one may assume that the two samples come from the same distribution and that the chain has converged to the target distribution somewhere inside the first 10%. The test statistic of the Geweke diagnostic is converted into a Z-score with the standard errors adjusted for autocorrelation. For every sampler output, set two random variables  $X_1$  and  $X_2$  to refer to the first 10% and last 50% of the sampled draws, respectively. Then, similar to the two samples  $T$  test of means  $\bar{x}_1$  and  $\bar{x}_2$  when the two sample variances  $s_1^2$  and  $s_2^2$  are not equal, calculate:

$$T = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n} + \frac{s_2^2}{m}}} \quad (2.28)$$

When  $n, m \rightarrow \infty$ , Equation 2.28 can be approximated using the standard normal  $Z$ , and the sample variances  $s_1^2$  and  $s_2^2$  need to be adjusted for autocorrelation. Note that Geweke diagnostic test uses spectral densities to estimate the  $s_1^2$  and  $s_2^2$  [82]. Values outside 2 standard deviations are taken to indicate that a longer chain is needed.

Another diagnostic test is the Heidelberger-Welch test [87]. It is used to determine the number of iterations to keep and the number to discard. The null hypothesis is that the samples are drawn from the same distribution. The diagnostic is applied to the whole chain. If the null hypothesis is rejected, then the first 10% are discarded and the test is applied again. If the null hypothesis is rejected for the second time, the first 20% of the chain is discarded and the test is repeated. This process is repeated using intervals of 10% of the chain until either the null

hypothesis is accepted, or 50% of the chain is discarded. The latter is taken to indicate failure to converge and that a longer MCMC run is required.

Another useful indicator of MCMC performance for a finite state space is the sum of squared differences (SSDs) between the target probabilities  $f_i$  for each state  $i$  and the observed proportions  $\hat{f}_i$  of the  $n$  samples that are in each state  $i$ . Formally, it is defined as follows:

$$SSDs = \sum_i (\hat{f}_i - f_i)^2. \quad (2.29)$$

One would expect the SSDs to decrease as the number of iterations increases until the chain reaches convergence.

One more graphical diagnostic I consider is the time-series plot of the log-likelihood at each iteration [84–86], which can be used to judge the point at which burn-in has occurred.

The autocorrelations between successive iterations given MCMC outputs are another widely used diagnostic test. The autocorrelation function  $\rho_k$  expressed in Equation 2.30 is the correlation between  $n$  draws  $x_i$  and their  $k^{th}$  lag:

$$\rho_k = \frac{\sum_{i=1}^{n-k} (x_i - \bar{x})(x_{i+k} - \bar{x})}{\sum_{i=1}^n (x_i - \bar{x})^2}, \quad (2.30)$$

where  $\bar{x}$  indicates the average of  $x_1, \dots, x_n$ . The autocorrelation  $\rho_k$  is expected to decrease as  $k$  increases. If autocorrelation is still relatively high for large values of  $k$ , this indicates slow convergence.

# Chapter 3

## A Review of Bayesian Networks

### 3.1 Introduction

Bayesian networks (BNs) (also called Bayesian Graphs or Bayes Nets) are a broad class of graphs providing a compact representation of joint probability distributions and allowing efficient belief updating based on probabilistic reasoning. Some of the references considered in this chapter include [88–93].

One goal of using BNs is to uncover statistical relationships among variables from an observed dataset. The more data-points collected, the more precisely their relationships can be inferred. BNs have been widely applied in different domains. In systems biology, variables may represent gene expression levels, signaling molecules, lipids, or any biologically relevant molecule [94, 95]. In medicine, the ALARM network is a medical diagnostic system to monitor patients in intensive care situations [96]. In weather forecasting, the HailFinder network [97] attempts to model severe weather conditions. For understanding the mathematical description behind BNs, the reader is referred to [3, 98].

This chapter provides a brief overview on Bayesian network models. Section 3.2 describes the main notations and definitions related to BNs. Section 3.3 discusses the size of graph space of BNs. Section 3.4 outlines some possible constraints to reduce the search space of BNs. Section 3.5 reviews the approach of Bayesian

inference to compute posterior distributions over graph spaces. Section 3.6, in general, explains how to fully learn a Bayesian network given some data-points using a Bayesian inference and MCMC sampling. A range of the widely used MCMC and non-MCMC approaches to infer BNs are also reviewed in Section 3.6.3 and Section 3.6.4, respectively.

## 3.2 Notations and definitions

This section is intended to serve as a basis for understanding Bayesian network models, including their definition, applications, conditional probability tables, and other main properties.

### 3.2.1 Directed acyclic graphs

BNs are a multivariate distribution satisfying certain constraints implied by directed acyclic graphs (DAGs). They are specifically used as a probabilistic method to visually represent directed causal relationships between variables, learned from a dataset.

I use  $G$  to refer to a graph or Bayesian network. A Bayesian network  $G$  can be expressed as a pair  $(V, E)$ , where  $V$  is a set of nodes (vertices) representing random variables  $X_1, X_2, \dots, X_n$  and  $E$  is a set of directed edges (arcs) representing relationships between pairs of random variables. A directed edge between any pair of variables  $X_i \rightarrow X_j$  indicates that  $X_j$  depends on  $X_i$ , and the variable  $X_i$  is said to be a parent of variable  $X_j$ , or  $X_j$  is the child node of  $X_i$ . The set of parent variables to  $X_j$  is denoted  $\text{Pa}(X_j)$ . Typically, such dependencies are intended to model cause-effect relationships e.g. given a particular edge  $X_i \rightarrow X_j$ , we can say that  $X_i$  causes  $X_j$  or  $X_j$  is on the effect of  $X_i$ . That is, the set of parents for a particular node  $X_i$  are considered to be causes of  $X_i$  or  $X_i$  is on the effect of the



set of its parents. Variables that may be reached via a directed path from  $X_i$  are called descendants of  $X_i$ . I denote the set of descendants of variable  $X_i$  by  $D(X_i)$ . By definition, an acyclic graph contains no cycles or self-loops. Figure 3.2 shows valid and invalid Bayesian network structures.

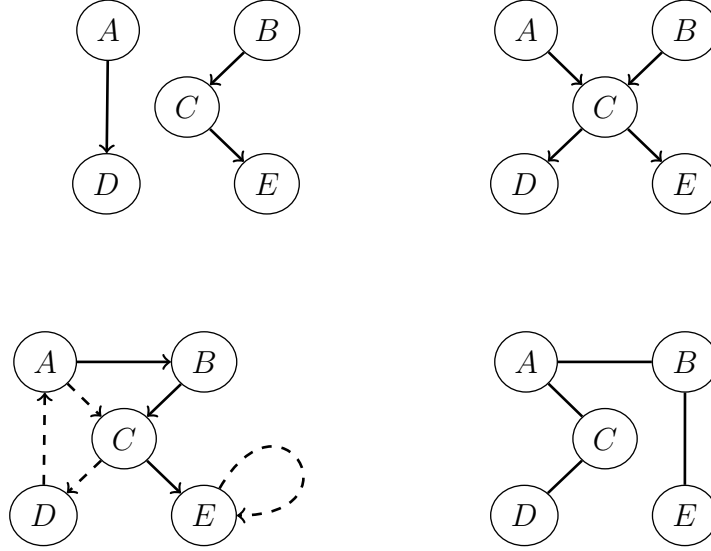


FIGURE 3.1: From left to right, top to bottom: disconnected Bayesian Network, connected Bayesian Network, invalid Bayesian Network because of cyclicity, invalid Bayesian Network because of undirected edges.

### 3.2.2 Markov property

Using Definition 1 of Markov property in Chapter 2, a Markov chain is graphically represented as

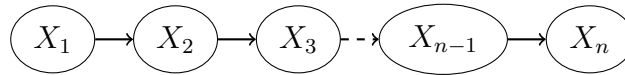


FIGURE 3.2: A directed acyclic graph represents an extension of the Markov property.

Thus an important property in BNs is the *Markov assumption*, which requires that each random variable  $X_i$  is conditionally independent of the values of its non-descendants, given the values of its parents in  $G$ .

### 3.2.3 Markov blanket

The Markov blanket  $\mathcal{B}(v)$  of a node  $v \in V$  is the set of nodes consisting of its parents, its children, and any other parents of its immediate children. Conditional on the values of the other nodes in its Markov blanket, each node is independent of the rest of the network. That is, for any node  $u \in V - \mathcal{B}(v) - \{v\}$ ,  $v \perp u | \mathcal{B}(v)$ . In other words, the joint distribution of the nodes in the Markov blanket  $\mathcal{B}(v)$  of a node  $v$  is sufficient knowledge for calculating the distribution of the node  $v$  [99]. An example of Markov blanket is illustrated in Figure 3.3.

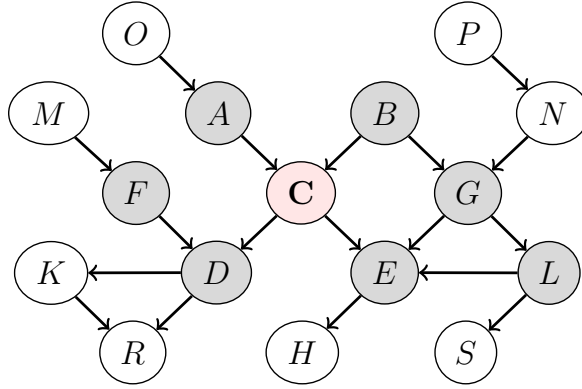


FIGURE 3.3: Example of a Markov blanket of node  $C$ . The members of the blanket are coloured in gray.

Figure 3.3 shows that node  $C$  is conditionally independent of the entire network, given its Markov blanket (the set of nodes colored in gray). That is, every set of nodes in Figure 3.3 is conditionally independent of node  $C$  when conditioned on the set of  $\mathcal{B}(C)$ , and the probability is then calculated using the Markov property.

### 3.2.4 Conditional probabilities table

The distribution of each child node  $X_i$  in a BN is dependent on its parents  $P(X_i | Pa(X_i))$  and encoded in a table in the form of local conditional probabilities. Given the Bayesian network shown in Figure 3.4, I use observational data formatted as in Table 3.2 to learn the conditional probabilities tables (CPTs) for the nodes W, S and R as illustrated in Table 3.1.

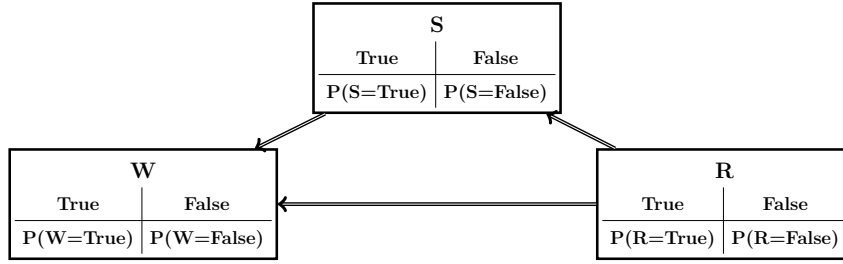


FIGURE 3.4: An example of inferred Bayesian network.

Index	Combinations of parents		Node W i=3	
$\mathcal{J}=4$	Node R	Node S	r=2	
			k=1	k=2
			True	False
j=1	True	True	$N_{311}$	$N_{312}$
j=2	True	False	$N_{321}$	$N_{322}$
j=3	False	True	$N_{331}$	$N_{332}$
j=4	False	False	$N_{341}$	$N_{342}$

Index	Combination of parents	Node S i=2	
$\mathcal{J}=2$	Node R	r=2	
		k=1	k=2
		True	False
j=1	True	$N_{211}$	$N_{212}$
j=2	False	$N_{221}$	$N_{222}$

Index	Combination of parents	Node R i=1	
$\mathcal{J}=1$	NA	r=2	
		k=1	k=2
		True	False
j=1	NA	$N_{111}$	$N_{112}$

TABLE 3.1: From top to bottom: the conditional probability tables for the nodes W, S and R shown in Figure 3.4.

Here  $N_{ijk}$  is the number of observations in a single state value cell  $k$  of node  $i$  corresponding to a parent configuration  $j$ ,  $r$  is the total number of state values (bins) that a particular node can take, and  $\mathcal{J}$  is the total number of combinations of parent state values. Formally,  $X$  is said to be conditionally independent of  $Y$  given  $Z$  if

$$P(X|Y, Z) = P(X|Z)$$

and I denote this statement by  $(X \perp Y|Z)$ , which means that the two nodes  $X$  and  $Y$  are conditionally independent given the third node  $Z$  if and only if they are independent in their conditional probability distribution given  $Z$ . That is,  $X$  and  $Y$  are conditionally independent given  $Z$  if and only if, given any value of  $Z$ , the probability distribution of  $X$  is the same for all values of  $Y$  and the probability distribution of  $Y$  is the same for all values of  $X$ . A key advantage of the Bayesian network is its compact representation of the joint probability distribution. By employing the conditional independence, the joint probability distribution over the variables  $S$ ,  $R$  and  $W$  becomes:

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

Table 3.2 contains three random variables  $S$ ,  $R$  and  $W$ . Each row in the Table represents a single observation of the values of all variables at particular times  $\{1, \dots, m\}$ , and  $m$  is the total number of observed data-points.

Observation	S	R	W
1	$a_1$	$b_1$	$c_1$
2	$a_2$	$b_2$	$c_2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$m$	$a_m$	$b_m$	$c_m$

TABLE 3.2: Data-points observed for three variables.

### 3.2.5 Joint probability function

The conditional dependencies encoded in a Bayesian network decompose a complicated multi-dimensional distribution into a product of lower dimensional distributions, as follows. Suppose that the random variables  $X_1, X_2, \dots, X_n$  have been sorted so that  $X_j$  is not a parent of  $X_i$  for any  $j > i$ . (Note this is always possible for a directed graph that does not contain cycles) Then one may write:

$$\begin{aligned} P(X_1, X_2, \dots, X_n) &= \prod_{i=1}^n P(X_i | X_1, X_2, \dots, X_{i-1}) \\ &= P(X_1)P(X_2|X_1)P(X_3|X_1, X_2) \dots P(X_n|X_1, X_2, \dots, X_{n-1}). \end{aligned} \quad (3.1)$$

Note that  $Pa(X_1)$  in Equation 3.1 is equal to  $\phi$ . Equation 3.1 can then be simplified using the Markov property as in Equation 3.2.

$$\begin{aligned} P(X_1, X_2, \dots, X_n) &= P(X_1|Pa(X_1))P(X_2|Pa(X_2)) \dots P(X_n|Pa(X_n)) \\ &= \prod_{i=1}^n P(X_i|Pa(X_i)). \end{aligned} \quad (3.2)$$

### 3.2.6 Equivalent graphs

A central concept in the analysis of DAGs is the *class of equivalent graphs*. Two DAGs with the same number of nodes are equivalent if they have the same underlying undirected graph structure, and the same v-structures [100]. Figure 3.5 illustrates how two DAGs are equivalent by satisfying these criteria. A *v-structure* is defined as a subgraph consisting of 3 nodes,  $X$ ,  $Y$  and  $Z$ , of the form  $X \rightarrow Y \leftarrow Z$  with no edges connecting  $X$  and  $Z$ .

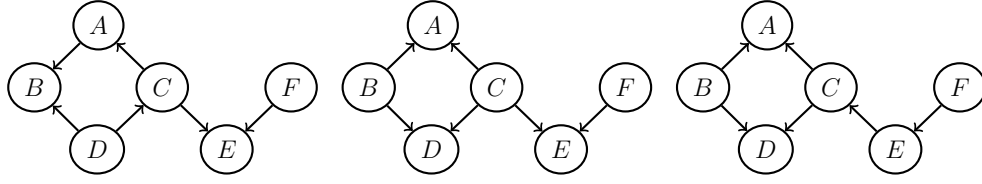


FIGURE 3.5: The leftmost DAG and the middle DAG are equivalent because they satisfy both criteria, but the middle DAG and the rightmost DAG are not because they have different v-structures.

A set of equivalent graphs describe the same dependencies among variables. Therefore, they should be sampled with equal probability.

### 3.3 Graph space

Sampling algorithms in spaces of DAGs are computationally intensive because the number of DAGs increases super-exponentially with the number of nodes of the graph. The size of the space of DAGs is  $2^{O(n^2)}$  [101]. Table 3.3 shows the total number of all possible DAGs and connected DAGs (CDAGs) for certain numbers of nodes, and emphasises the need for MCMC samplers that can efficiently traverse such graph spaces. The reader is referred to Section 3.4.1 for a discussion of the connectivity of a graph.

Nodes N.	DAGs	CDAGs
3	25	18
4	543	446
5	29 281	26 431
$\vdots$	$\vdots$	$\vdots$
10	1 4 175 098 976 430 598 100	NA

TABLE 3.3: Number of all possible DAGs and connected DAGs increases exponentially as the number of nodes increases.

Below is a mathematical expression used to calculate the total number of DAGs given a certain number of nodes [102]:

$$f(n) = \sum_{i=1}^n (-1)^{i+1} \binom{n}{i} 2^{i(n-i)} f(n-i) \quad (3.3)$$

Note that, a thorough search of the relevant literature e.g. [3, 88–93, 98, 99, 101–109] found no expression for the number of CDAGs. The space sizes of CDAGs listed in Table 3.3 were tractable enough to be calculated using the brute-force approach presented in Section 4.3 by first checking the constraints of acyclicity and connectivity for graphs consisting of 3, 4, and 5 nodes, and then enumerating all valid structures. The space of CDAGs with only 3 nodes is small enough that the entire probability distribution can be evaluated and used to assess the performance of a sampler (see Figure 3.6).

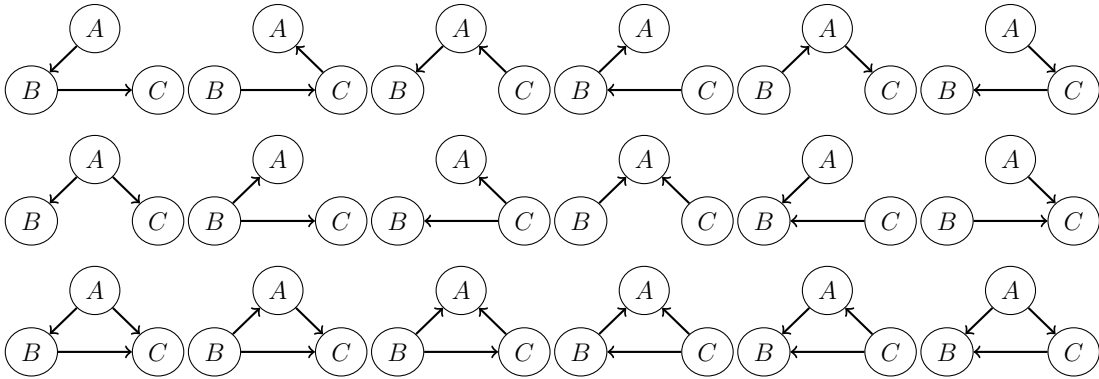


FIGURE 3.6: All possible (18) CDAGs. These comprise all possible connected BN structures containing 3 nodes.

### 3.4 Graph constraints

Adding justifiable additional restrictions on a very large directed acyclic graph (DAG) space is a sound technique that I use here to reduce its cardinality [106, 110, 111]. Reducing the number of acceptable graphs in the space can facilitate rapid convergence of an MCMC algorithm. However, accurate prior knowledge of

plausible restrictions on the graph is required to do this appropriately, since making improper restrictions may remove the true graph from the space. Sections 3.4.1, 3.4.2 and 3.4.3 describe three restrictions that are appropriate in many applications: connectivity, acyclicity, and limiting in-degree and out-degree.

Note that Chapter 6 involves a new adaptive technique which was proposed in this thesis to quickly enumerate a set of adjacent graphs. However, I initially used the two techniques explained in Section 3.4.1 and Section 3.4.2.

### 3.4.1 Connectivity

The connectivity restriction requires all the nodes to be connected to at least one other node in the network i.e. an undirected path exists between any two nodes in the graph. Note that I consider a directed graph to be connected if there is an undirected path between any two nodes in the graph, that is, if the undirected graph obtained by replacing all directed edges with undirected edges (and removing any duplicated edges) is connected. This restriction ensures a network with  $n$  nodes must have at least  $n - 1$  edges. The edge that, if removed, would disconnect the graph into two sub-graphs is called a *bridge*. The maximum number of bridges in a connected directed acyclic graph is  $|V| - 1$  [112]. For clarification, if an edge that is not a bridge is deleted, the graph remains connected and any edge that was a bridge is still a bridge. Thus the number of bridges stays the same or increases. Keep deleting non-bridges until only bridges are left. This graph has  $|V| - 1$  bridges, which must be more than the original graph. Note that before any edge  $(X_i, X_j)$  may be deleted, I observe the connectivity of the resulting graph. I delete that edge and apply the Breadth-First-Search (BFS) algorithm [103] to detect if the graph becomes disconnected.

Early in this project, I initially adopted a simple method to check connectivity. After removing a particular edge  $(X_i, X_j)$ , I checked whether there exists any simple



path from any other node in the graph to the node  $X_i$  or node  $X_j$ . Here, the BFS algorithm was used to find such simple path(s). BFS is a very well-known graph traversal algorithm which starts at any arbitrary node and explores all immediately adjacent nodes before observing others. If there exist any such simple path(s), the resulting graph would be connected even if I remove the edge  $(X_i, X_j)$ . Therefore, the edge  $(X_i, X_j)$  will be considered as a deletable edge. If I decide to reject the deletion, I restore back the edge  $(X_i, X_j)$  that I deleted previously. Note, for the purpose of DAG connectivity, I do not consider the directions of edges.

### 3.4.2 Acyclicity

Acyclicity is a required restriction for a Bayesian network. I initially used the well-known Depth-First-Search (DFS) algorithm [113] to detect cycles. Before adding any edge  $(X_i \rightarrow X_j)$  into a particular graph, I observe whether I can find any cycle in the resulting graph. That means, I first add that edge  $(X_i \rightarrow X_j)$  and run DFS algorithm to determine whether any cycle evolves in the resulting graph. If not, I considered that edge to be an addable edge in the graph, otherwise not. DFS is an algorithm for traversing tree or graph data structures. One starts at an arbitrary root and explores as far as possible along each branch before backtracking. If I decide to reject the insertion, I remove the edge  $(X_i \rightarrow X_j)$  that I added before.

### 3.4.3 Node degree

The *centrality degree* of a particular node in a graph is concerned with the number of edges associated with that node [114]. A node with high degree of centrality is in some sense of high significance to the structure of the graph. I use two measures to assign the degree of centrality: in-degree and out-degree.

In-degree and out-degree are integer numbers that respectively represent the numbers of parents and children that a particular node possesses, or equivalently

the respective number of head and tail endpoints incident on a node. Setting a maximum number of parents or children for each node can dramatically reduce the size of graph space where there is reliable prior knowledge about these parameters.

## 3.5 Bayesian inference

In many applications there may be additional information about the parameter  $\theta$  of a particular probability distribution  $f_\theta(x)$ . Bayesian inference deals with the parameter  $\theta$  as a random variable, and thus a prior distribution  $P(\theta)$  is defined. The distribution  $P(\theta)$  is used to describe the prior knowledge about  $\theta$ . Bayesian inference ultimately aims to determine how prior beliefs change after collecting data.

### 3.5.1 Posterior distribution

Let  $D = \{d_1, d_2, \dots, d_m\}$  represent a set of data-points drawn independently from a probability distribution  $P(D|\theta)$ . The posterior distribution  $P(\theta|D)$  is a conditional probability distribution of the parameter  $\theta$  given a dataset  $D$ . It describes the degree of belief about different values of  $\theta$  after observing  $D$ . The joint probability of  $\theta$  and  $D$  is  $P(D, \theta)$  and can be calculated as [89, 93]

$$P(D, \theta) = P(\theta)P(D|\theta) = P(D)P(\theta|D) \quad (3.4)$$

Based on expression 3.4, and using Bayes Rule, one can write:

$$P(\theta|D) = \frac{P(\theta)P(D, \theta)}{P(D)} \quad (3.5)$$

where  $P(D)$  is the marginal function of  $D$  which can be given by integrating a density function

$$P(D) = \int P(D, \theta) d\theta = \int P(\theta) P(D|\theta) d\theta$$

or by summing a probability mass function

$$P(D) = \sum_{\theta} P(D, \theta) = \sum_{\theta} P(\theta) P(D|\theta)$$

### 3.5.2 Prior distribution

There are two common types of prior distribution: conjugate prior distributions and non-informative priors. These are not mutually exclusive: a conjugate prior can also be non-informative. Typically, a non-informative prior reflects a balance among samples when no information about a particular variable is available. A common non-informative prior is the uniform distribution. In a conjugate prior, the calculated posterior distribution is in the same parametric family as the prior distribution. Examples of likelihood distributions and their conjugate distributions are listed in Table 3.4. Using a conjugate prior distribution, it is possible to compute the closed-form expression of a posterior distribution, hence numerical integration may not be necessary.

Distribution	Parameter	Conjugate prior distribution
Binomial	success probability	Beta
Poisson	mean	Gamma
Exponential	inverse mean	Gamma
Normal	mean (known variance)	Normal
Normal	variance (known mean)	Gamma inverse
Multinomial	$(\alpha_1, \dots, \alpha_k)$ vector of parameters	Dirichlet

TABLE 3.4: Some common conjugate prior distributions

### 3.5.3 Bayesian estimation

The posterior distribution can be summarised as a point estimation using one of the central tendency measures such as the mean, median, or mode. In practice, it is common to calculate the expected value  $E(\theta)$

$$\theta^* = E(\theta) = \begin{cases} \int \theta P(\theta|D) d\theta \\ \sum \theta P(\theta|D) \end{cases} \quad (3.6)$$

It is possible also to specify an interval estimation  $I(\theta|D)$  for the parameter  $\theta$  using the prior distribution before even obtaining the sample observations. For example, if  $\theta$  has a prior distribution  $P(\theta)$  and  $\int_a^b P(\theta) d\theta = 1 - \alpha$ , one can say that the interval  $(a, b)$  contains  $\theta$  with probability  $(1 - \alpha)$ . After obtaining the observations and building the posterior distribution  $P(\theta|D)$ , one can choose two values  $(t_1, t_2)$ , such that

$$\int_{t_1}^{t_2} P(\theta|D) d\theta = 1 - \alpha \Leftrightarrow P(t_1 < \theta < t_2) = 1 - \alpha \quad (3.7)$$

Practically,  $(1 - \alpha)$  is fixed and one searches for the appropriate values  $(t_1, t_2)$ , where  $t_i$  is a function in the sample observations  $D$ .

## 3.6 Learning Bayesian networks

Inferring a Bayesian network typically involves two conceptually different elements: structure learning and parameter learning. Structure learning involves inferring the variables that interact and the causal directions of those interactions; in other words it is inferring the set of edges connecting a set of candidate nodes. For a fixed structure, parameter learning involves quantitatively estimating probabilistic dependencies between variables. In practice, structure and parameter learning may

be performed simultaneously. In this thesis, both types of learning are explicitly considered while sampling BNs. More precisely, I use Bayesian inference to learn the conditional probabilities among variables and MCMC samplers to learn the structures of BNs.

### 3.6.1 Learning Bayesian network parameters

Learning the parameters of a BN corresponds to learning the local conditional probabilities among the variables encoded. Given data  $D$  and fixed graph  $G$ , I first define the probability distribution  $P(X_i|Pa(X_i))$  for each variable  $X_i$  given its parents  $Pa(X_i)$ . I write  $P(X_i = k|Pa(X_i) = j) = \theta_{ijk}$ , where  $\theta_{ijk}$  is the probability of each state value (bin)  $k$  within each variable  $X_i$ , given that its parents are in configuration  $j$ .

A conventional approach to estimate the parameters of a Bayesian network is to use the maximum likelihood estimate (MLE). The MLE maximises the likelihood function  $L(\theta : D)$ , on the data where  $\theta = (\theta_{ijk})$ , and then attempts to find the parameter value that maximises the value of the likelihood function [88, 92]:

$$\hat{\theta} = \arg \max_{\theta} L(\theta : D) \quad (3.8)$$

The posterior distribution used in Bayesian inference is different from maximum likelihood estimation in several ways [93]. First, it is not a point estimate that assigns a particular value to  $\hat{\theta}$ , but instead is a distribution that assigns a probability density to each possible value of  $\theta$ . Second, it always takes the prior distribution into account.

Using Bayesian inference, the Multinomial distribution is a common model used when the nodes are discrete valued to relate the values in a CPT to the observed data [104]. To complete the Bayesian inference, one also need to assign prior probabilities

to parameter values. Here, I use the conjugate prior of the Multinomial model, which is the Dirichlet distribution. The Dirichlet distribution over these parameters is expressed as:

$$\begin{aligned} P(\theta_{ij1}, \theta_{ij2}, \dots, \theta_{ijr_i} | G) &= \text{Dir}(\theta_{ij1}, \theta_{ij2}, \dots, \theta_{ijr_i} | \alpha_{ij1}, \alpha_{ij2}, \dots, \alpha_{ijr_i}) \\ &= \Gamma(\alpha_{ij}) \prod_{k=1}^{r_i} \frac{\theta_{ijk}^{\alpha_{ijk}-1}}{\Gamma(\alpha_{ijk})}, \end{aligned} \quad (3.9)$$

where  $r_i$  is the number of possible state values (bins) for  $X_i$ ,  $\alpha_{ijk}$  are the hyperparameters and  $\alpha_{ij} = \sum_{k=1}^{r_i} \alpha_{ijk}$ . Assuming local and global parameter independence, the distribution over the set of parameters for the entire Bayesian network is:

$$P(\theta | G) = \prod_{i=1}^n \prod_{j=1}^{\mathcal{J}_i} \Gamma(\alpha_{ij}) \prod_{k=1}^{r_i} \frac{\theta_{ijk}^{\alpha_{ijk}-1}}{\Gamma(\alpha_{ijk})} \quad (3.10)$$

where  $\mathcal{J}_i$  is the number of possible state values for the parents of  $X_i$ . Hence, the posterior distribution of the parameters conditional on the data set is also a member of the Dirichlet family:

$$\begin{aligned} P(\theta_{ij1}, \dots, \theta_{ijr_i} | D, G) &= \text{Dir}(\theta_{ij1}, \dots, \theta_{ijr_i} | \alpha_{ij1} + N_{ij1}, \dots, \alpha_{ijr_i} + N_{ijr_i}) \\ &= \prod_{i=1}^n \prod_{j=1}^{\mathcal{J}_i} \Gamma(\alpha_{ij} + N_{ij}) \prod_{k=1}^{r_i} \frac{\theta_{ijk}^{N_{ijk} + \alpha_{ijk} - 1}}{\Gamma(N_{ijk} + \alpha_{ijk})} \end{aligned} \quad (3.11)$$

where  $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$ . From Equation 3.11 it follows that [104]:

$$P(D | G) = \prod_{i=1}^n \prod_{j=1}^{\mathcal{J}_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + N_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(\alpha_{ijk} + N_{ijk})}{\Gamma(\alpha_{ijk})} \quad (3.12)$$

This result is obtained by integrating over  $\theta$ :  $P(D | G) = \int P(D, \theta | G) d\theta$ . Expression 3.12 is useful in that it gives the likelihood of a set of data in terms of the structure only, without reference to the parameters associated with each node.

In this thesis, the Dirichlet priors have been chosen to produce a non-informative prior:

$$\alpha_{ijk} = \frac{\alpha}{\mathcal{J}_i r_i} \quad (3.13)$$

where  $\alpha$  is the total imaginary counts for the Dirichlet prior, which has been set in this thesis to one. Note that when  $\alpha = 1$ , the Dirichlet distribution will be equivalent to a uniform distribution [115]. The posterior probability distribution of the graph  $G$  given data  $D$  can now be constructed as:

$$P(G|D) = \frac{P(D|G)P(G)}{P(D)} \quad (3.14)$$

To sample from 3.14 using MCMC, one needs only consider the numerator, since the denominator does not depend on  $G$  and will cancel out. For simplicity, one can write:

$$P(G|D) \propto P(D|G)P(G) \quad (3.15)$$

In this thesis, I assume a uniform prior  $P(G)$  on the graph. Note also that equivalent graphs have the same prior probabilities, likelihoods and posterior probabilities. Consequently, equivalence classes of graphs have prior and posterior probabilities proportional to the number of equivalent graphs in that class. This is possibly undesirable, as there is no obvious reason why larger equivalence classes should be preferred *a priori*. In principle, this effect could be counteracted by assigning a prior probability to each graph  $G$  inversely proportional to the size of its equivalence class. However, for simplicity I have retained the uniform prior in what follows.

A practical issue that arises when working with Equation 3.12 is the very high values that result from multiplying several gamma functions together. The solution is to work with the *log* of these values wherever possible.

### 3.6.2 Learning Bayesian network structures

One technique used to learn a BN structure is to sample from a posterior distribution over the space of that Bayesian network, using MCMC samplers [105, 108, 116]. This presupposes that a prior distribution and likelihood model have been defined over graph space, and that Bayes Rule has been applied to obtain a posterior distribution.

**Remark 1.** Practitioners can use a MCMC sampler to infer the structure that best represents the relationships among variables, but it may instead be advantageous to infer the probability that certain nodes or edges are present in the true graph, and thus avoid selecting any one structure as optimal.

A properly designed MCMC sampler must, in theory, converge to the required limiting distribution. However, in practice, such methods can become effectively “trapped” in local modes of the posterior distribution, only infrequently moving between modes. Among the main goals of this thesis is to present new and efficient samplers that are capable of traversing graph spaces with reduced frequency of getting trapped in local modes. Chapter 4 describes new instances of certain MCMC samplers chosen as promising techniques for learning BN structures.

### 3.6.3 MCMC methods for learning Bayesian networks

MCMC sampling has a wider applicability to infer hard BNs than exact algorithms [117–120]. MCMC methods also provide a more powerful framework to sample from distributions with local modes than Monte Carlo methods [13, 20, 66], despite the fact that MCMC may exhibit slow convergence.

This section attempts to outline significant MCMC approaches proposed in the literature to improve inferring BNs. The original version of MCMC approach to infer BNs was proposed in [121], where every single transition in a Markov chain is performed by modifying, at most, a single edge in the current graph. When the size



of the search space of Bayesian network structures grows, the sampler may exhibit slow mixing and less efficiency in visiting low probability regions in the true posterior distribution.

This issue was slightly resolved in [122] by providing several graphical monitors proposed to assess approximations to the posterior probability distributions. The original MCMC approach was varied in [123] to reduce sampling from the entire search space of Bayesian network structures to sampling over the search space of topological node orders. This allows for better mixing and converging to the same posterior probability estimates. This approach was extended in [124] to combine the order-space MCMC with other several non-MCMC approaches e.g. importance weighting in order to create a fast structural learning algorithm. The MCMC approach based on node order was also adopted in [123] to infer Bayesian network structures when the amount of observational data is limited. More recently, a new MCMC algorithm based on partial node orders was also proposed in [125]. However, a disadvantage of performing proposal transitions based on node order rather than entire structure space is that it is not always applicable due to the difficulty to explicitly specify priors over BNs [126]. As a consequence, several variants of the order MCMC algorithm have been developed in the literature to address this prior bias [124, 127].

A recently proposed MCMC approach in [128] aims to approximate a Bayesian solution to the problem of learning Bayesian network by constraining the search space of Bayesian network structures to a given relaxed structure with undirected edges. This method tends to fit with larger BNs but not for small sample sizes, and also requires prior determination of a specific structure that involves undirected edges which is not always available.

Another improvement made to MCMC approach was in the applications where the problem of incomplete data is present [129, 130]. A new transition technique

was proposed in [126] based on edge reversal considering the entire search space of structures, which significantly improved learning true structure. It aims to apply a substantial modification for every sampled network to facilitate traversing the search space effectively. For every MCMC iteration, it selects an edge from the generated Bayesian network, and then reverses its direction. Next, it resamples a new set of parents for each of the two nodes that are linked by the directed edge.

Special attention was recently paid to resolve the problem of local maxima solution arising while learning BNs structures. Since the MCMC sampling produces highly correlated samples, one challenge is how to introduce transitions that can take larger steps in the search space of a Bayesian network [131]. Based on similar idea, this thesis seeks to propose MCMC samplers that have been known to substantially avoid getting stuck in local maxima but have been exploited to explore discrete spaces of Bayesian network. An instance of such MCMC methods is the Hit-and-Run sampler and the Neighbourhood sampler reviewed in Sections 2.4.2 and 2.4.3, and modified in Sections 4.6 and 4.7, respectively.

### 3.6.4 Non-MCMC methods for learning Bayesian networks

For the purpose of comparison with the MCMC samplers proposed in this thesis, I highlight some of the most widely used non-MCMC methods to learn BNs, to be used as a baseline along with the Metropolis-Hastings MCMC sampler to assess the performances of the new methods.

Readers may find this reference book [132] useful to understand the theoretical and pseudocode basis of most of stochastic simulation algorithms drawn from different sub-fields of artificial intelligence. Another reference focused on the algorithms and issues of BNs is [133].

A commonly used category of algorithms to learn BNs is score-based algorithms [134]. This category of algorithms aims to maximise the pre-assigned score of each

Bayesian network using a heuristic search. One of the most widely studied heuristic search methods is Greedy Algorithms (GAs) [134–136]. GAs typically update a given Bayesian network by either adding, deleting or reversing a particular directed edge at each step. Among the most widely used GAs are Hill-Climbing Search (HCS) and Tabu Search (TS) [134, 136–139].

The HCS algorithm starts with an arbitrary Bayesian network, and then iteratively applies a local search to its neighbors in the hope of finding a neighboring network with a better score. It repeats this process until no further improvements can be obtained. The TS algorithm also runs a local search similar to the HCS; however, it intentionally enhances the performance of local search by relaxing its acceptance function, i.e. when the search gets stuck at a local minimum and no improving move is available, worsening moves can then be accepted. The TS algorithm also uses a memory structure that describes all visited solutions. If a particular Bayesian network has been previously visited but did not improve the score, it is then marked as "tabu" and not considered again.

Heuristic search methods can get stuck in a local mode when the immediate neighbours of a network do not provide any better solution. For more details and illustrative visualisation of using the HCS and TS, the reader is referred to [140].

Another major category of algorithms for learning BNs is constraint-based algorithms. These aim to analyse the probabilistic relations entailed by the Markov property of BNs with conditional independence tests and then construct a Bayesian network that satisfies the corresponding d-separation statements (Note, a path between two nodes  $u$  and  $v$  is said to be d-separated by a set of nodes  $Z$  if and only if the path contains (at least): 1) a chain,  $u \leftarrow m \leftarrow v$ , such that  $m \in Z$ , 2), a common cause,  $u \leftarrow m \rightarrow v$ , such that  $m \in Z$ , or 3) a common effect,  $u \rightarrow m \leftarrow v$ , such that  $m \notin Z$ ).

One common algorithm in this category is the Grow-Shrink (GS) algorithm [141].

The GS approach constructs BNs by identifying the Markov blanket for each node. There are two phases involved in the GS algorithm: growing phase and shrinking phase. To find a Markov blanket of a particular node  $v \in V$  in  $G(V, E)$  using the GS algorithm: First, process the growing phase as follows: 1) define an empty set  $\mathcal{S}$ , 2) test the independence for each node  $u \in V - \{v\}$  with node  $v$ , 3) if  $u$  is dependent on  $v$  given the current  $\mathcal{S}$ , then add it to  $\mathcal{S}$ , 4) stop if there are no more such nodes  $u \in V - \{v\}$ . Note that the examining order of nodes  $u \in V - \{v\}$  is arbitrary. Second, apply the shrinking phase as follows:

- Test the dependence for each node  $u \in \mathcal{S}$  with node  $v$ .
- If  $u$  is independent of  $v$  given  $\mathcal{S}$ , then remove  $u$  from  $\mathcal{S}$ .
- Stop when all nodes in  $\mathcal{S}$  are checked.

Note that this algorithm is done in such a manner as to avoid producing dense nets or incorrect causal relationships. An illustrative example of the operation of GS algorithm can be found in [140].

# Chapter 4

## Using the MH, NS and HAR to Sample Bayesian Networks

### 4.1 Introduction

This chapter considers sampling graphs from *discrete* spaces using MCMC sampling. I first assume that a target distribution  $f(x)$  has been defined on a discrete space  $\mathcal{X}$  that contains graphs, and  $|\mathcal{X}|$  is the (finite) total number of graphs in that target space. The chapter specifically discusses the implementations of *three* MCMC samplers: Metropolis-Hastings (MH) sampler, Hit-and-Run (HAR) sampler and Neighbourhood Sampler (NS) to sample Bayesian networks. Each of the three MCMC samplers typically requires defining the following: proposal distribution, target distribution, rejection-acceptance step, and transition step to generate a candidate graph from the same target space using a Markov chain process. These four components are discussed for each MCMC sampler in the following subsections.

Section 4.2 describes how to define a set of local adjacent graphs given a connected Bayesian network. Section 4.3 defines a possible standard brute-force approach to check acyclicity and connectivity for a generated random graph. Section 4.4 discusses various techniques to generate candidate graphs using different MCMC samplers. Section 4.5, Section 4.6 and Section 4.7 explain how to use the MH, NS

and HAR, respectively, to sample connected Bayesian network structures from their discrete graph spaces.

## 4.2 Enumerating a set of adjacent graphs

Two graphs are considered adjacent in their space if they differ in their structures by only one edge. In order to find one possible adjacent graph  $G'$  of a current graph  $G$ , one can modify  $G$  by using one of the following operations: add an edge to  $G$ , delete an edge from  $G$ , or reverse an existing edge in  $G$ , provided that the graph  $G'$  remains a connected Bayesian network. The option of reversing an edge is only available if the edges in  $G$  are directed (as in Bayesian networks).

To construct the set of adjacent graphs  $\mathcal{N}_G$  for a particular Bayesian network  $G$ , one must consider all possible edges that can be added, deleted and reversed while the Bayesian network remains connected and acyclic. All such adjacent graphs plus the original graph itself are defined as the set of adjacent graphs  $\mathcal{N}_G$  of graph  $G$ .

Figure 4.1 shows how all possible addable, deletable and reversible edges are identified to obtain the corresponding adjacent graphs of a particular initial Bayesian network. The total number of the graphs in  $\mathcal{N}_G$  is denoted  $\mu_G$  which may be any positive integer greater than or equal one. For example, in Figure 4.1,  $\mu_G = 10$  since we have three addable edges, three deletable edges, three reversible edges, and the graph  $G$  itself.

One possible standard brute-force approach to detect addable, deletable and reversible edges is provided in the following section. A modified version of brute-force approach to efficiently detect deletable edges is also described in Remark 2.

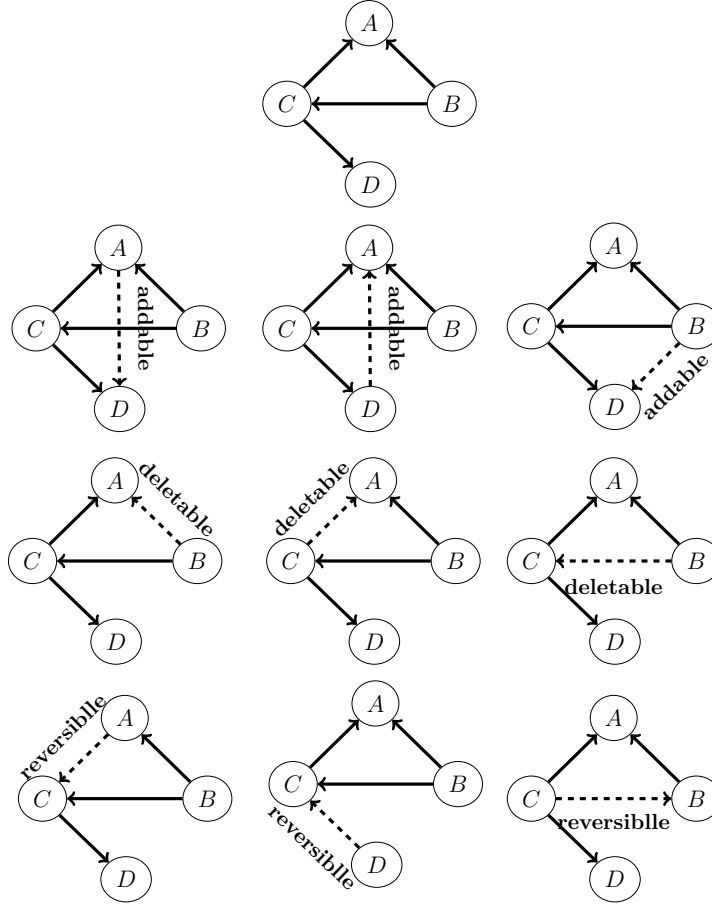


FIGURE 4.1: From top to bottom: Initial connected Bayesian network, all possible addable edges, all possible deletable edges, and all possible reversible edges, provided that all graphs remain connected and acyclic.

### 4.3 Standard brute-force approach

A brute-force approach [142, 143] is a search technique that aims to systematically enumerate all possible solutions for basic algorithms or data structures. Although, it can be intuitively implemented and has the potential to find a solution, it becomes computationally costly when the number of candidate solutions increases. Note that a brute-force approach does not consider efficiency in finding solutions. To enumerate all possible adjacent graphs for a particular graph  $G$ , a brute-force approach typically checks *every* single pair of nodes that belongs to  $G$  to determine whether an edge can be added, deleted, or reversed provided that the graph structure remains a connected Bayesian network.

For each pair of nodes  $(i, j)$  in graph  $G(V, E)$ , if there is an edge in the graph  $G$ , then I checked whether a candidate adjacent graph can be produced by either adding an edge, deleting an edge or reversing an edge between  $i$  and  $j$ . For this purpose, I created a temporary graph  $G'$  by either adding an edge, deleting an edge or reversing an edge between  $i$  and  $j$ , respectively, and checked the constraints e.g. cyclicity after adding or reversing an edge or connectivity after deleting an edge. For cyclicity or connectivity checking, BFS algorithm can be used. If  $G'$  maintains all constraints, I consider that  $(i, j)$  as a valid addable, deletable or reversible edge.

**Remark 2.** The deletable edges can be found more efficiently by detecting bridges first, then all non-bridges are identified as deletable edges. To do so, a modified-DFS algorithm [144] can be used which requires two additional arrays: *low* and *pre*. For each vertex  $i$ , *pre*[ $i$ ] saves the order in which DFS traverses  $i$  based on the pre-order traversal, and *low*[ $i$ ] saves the lowest pre order number of any vertex connected to  $i$ .

## 4.4 Assigning candidate graphs iteratively using the MH, NS and HAR

The MCMC samplers proposed in this thesis use different processing mechanisms to transit from a current graph to another candidate graph in the same discrete space. Below, I briefly clarify the main idea of how to transit among graphs in the same space using the MH, NS and HAR.

The MH Sampler generates a candidate graph  $G'$  from the set of all immediate adjacent graphs using the uniform distribution. An immediately adjacent graph  $G'$  differs from  $G$  by only one edge.

The NS considers a candidate graph after applying two such steps. The sampler first generates  $G' \sim \mathcal{U}(\mathcal{N}_G)$ , where  $\mathcal{U}$  denotes the uniform distribution. It then



generates  $G'' \sim \mathcal{U}(\mathcal{N}_{G'})$ . Taking two steps in this manner helps reduce the problem posed by local modes. Note that the two-steps transition guarantees that for each graph  $G'' \in \mathcal{N}(G')$ , it also satisfies that  $G'' \in \mathcal{N}(G)$ .

The HAR sampler enables transitions from current graphs to distant graphs. The sampler generates a sequence of graphs, in which each graph is immediately adjacent to the graph that precedes it. The maximum number of graphs in the sequence is chosen by the user. The sampler aims to substantially resolve the problem of getting stuck in a local mode.

Regardless of the processes used to generate candidate graphs, all of the MCMC samplers considered in this thesis require iteratively defining sets of adjacent graphs. In Chapter 6, I develop new adaptive techniques to quickly enumerate these adjacent graphs.

## 4.5 Metropolis-Hastings sampler

The MH algorithm was first used in [121] to sample Bayesian graphical structures according to their posterior distributions. The Metropolis-Hastings sampler allows asymmetric transition probabilities to generate a candidate graph.

To sample graphs with the MH, one must first define a proposal distribution. To keep the sampler comparable with the other two MCMC samplers (NS and HAR), it is possible to set the proposal to the uniform distribution over the same adjacent graphs. Given an initial graph  $G = G_t$ , where  $f(G) > 0$ , draw a connected graph,  $H \in \mathcal{N}_G$  in accordance with the uniform proposal  $q(H|G)$  with its implied probability  $1/\mu_G$ . Conversely, it is also required to find  $q(G|H)$  with its implied probability  $1/\mu_H$ . Then, draw a uniform (0,1) random value  $U$ , and then take  $G_{t+1} = G$  if  $\frac{f(H)}{f(G)} \cdot \frac{q(G|H)}{q(H|G)} \geq U$ , or  $G_{t+1} = H$  otherwise. The following pseudocode in Algorithm 4 describes sampling graph spaces using the MH sampler.

---

**Algorithm 4** Sampling graph space with the MH sampler

---

```

1: Initialise graph  $G_0$ , and set  $t := 0$ .
2: for all  $t = 0, 1, \dots, n$  do
3:   Given the current graph  $G_t = G$ , find  $\mathcal{N}_G$  and  $\mu_G$ .
4:   Generate  $U \sim \text{U}(0, 1)$ .
5:   Sample graph  $H \sim \text{U}(\mathcal{N}_G)$ , and find  $\mathcal{N}_H$  and  $\mu_H$ .
6:   if  $\frac{f(H)}{f(G)} \cdot \frac{q(G|H)}{q(H|G)} \geq U$  then
7:     set  $G_{t+1} = H$ 
8:     goto 2
9:   else
10:    set  $G_{t+1} = G$ 
11:    goto 2
12:   end if
13: end for

```

---

Figure 4.2 and Figure 4.3 show how to find the probabilities of transitions between two adjacent graphs  $G$  and  $H$  using uniform proposals. Calculating the probabilities depends on the (finite) total number of adjacent graphs  $\mu_G$  and  $\mu_H$ . That is, each adjacent graph of  $G$  or  $H$  is sampled with a probability of  $\frac{1}{\mu_G}$  or  $\frac{1}{\mu_H}$ , respectively. Given the graph  $G$  in Figure 4.2, there are four adjacent graphs  $\{G, C, H, D\}$  that were assigned after considering all addable, deletable and reversible edges of  $G$  plus the given graph itself. The probability of the transition from graph  $G$  to one of its adjacent graphs (e.g.  $H$ ) is thus  $\frac{1}{4}$ .

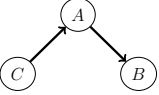
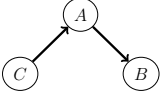
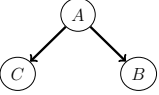
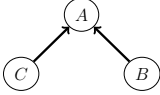
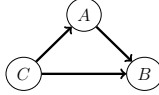
$G$				
$\mathcal{N}(G)$	$G$ 	$C$ 	$H$ 	$D$ 
Probability	$P(G G)$ $\frac{1}{4}$	$P(C G)$ $\frac{1}{4}$	$P(H G)$ $\frac{1}{4}$	$P(D G)$ $\frac{1}{4}$

FIGURE 4.2: The probability to transit from graph  $G$  to one of its adjacent graphs is 25%.

Given the graph  $H$  in Figure 4.3, there are five adjacent graphs  $\{H, G, E, F, B\}$ . If one wants to move from graph  $H$  back to graph  $G$ , the probability is  $\frac{1}{5}$ , because  $\mu_H = 5$ .

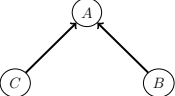
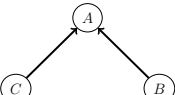
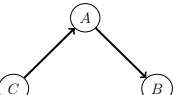
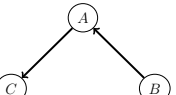
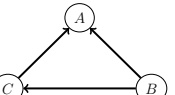
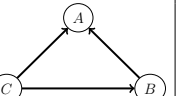
$H$					
$\mathcal{N}(H)$	$H$ 	$G$ 	$E$ 	$F$ 	$B$ 
Probability	$P(H H)$ $\frac{1}{5}$	$P(G H)$ $\frac{1}{5}$	$P(E H)$ $\frac{1}{5}$	$P(F H)$ $\frac{1}{5}$	$P(B H)$ $\frac{1}{5}$

FIGURE 4.3: The probability to go back from graph  $H$  to one of its adjacent graphs is 20%.

## 4.6 Neighbourhood Sampler

The process of sampling Bayesian networks using the NS begins by selecting an arbitrary initial graph  $G_0$ , at  $t = 0$ . The set of local adjacent graphs  $\mathcal{N}_{G_0}$  must then be identified for  $G = G_0$ , and the cardinality of that adjacent graph  $\mu_G$  determined. The proposal distribution of the NS is always the uniform distribution. The sampler then generates a uniform value  $U$  from the interval  $U(0, \frac{f(G)}{\mu_G})$ . Next, it samples another graph  $H_1$  uniformly from  $\mathcal{N}_G$ , that is,  $H_1 \sim U(\mathcal{N}_G)$ . Given  $\mathcal{N}_{H_1}$ , sample a second graph  $H_2$  uniformly, that is,  $H_2 \sim U(\mathcal{N}_{H_1})$ . Having calculated  $\mu_{H_2}$ , then apply the following acceptance ratio. If  $\frac{f(H_2)}{\mu_{H_2}} \geq U$ , accept the graph  $H_2$  and set  $G_{t+1} = H_2$ . Otherwise, exclude the graph  $H_2$  from  $\mathcal{N}_{H_1}$  and select another  $H_2$  until the acceptance ratio is satisfied. Once  $G_{t+1}$  has been selected, the entire process is iterated. I particularise the NS for sampling Bayesian networks, as summarised in Algorithm 5. In Appendix A Section A.1, I provide a flowchart to represent Algorithm 5.

Figure 4.4 illustrates how the mediator graph  $H_1$  in the NS enables a larger number of graphs to be reached within one iteration, providing a better chance to move to a new graph. Thus an advantageous property of the NS is that it is less likely than MH to get stuck in a local maximum for some number of iterations.

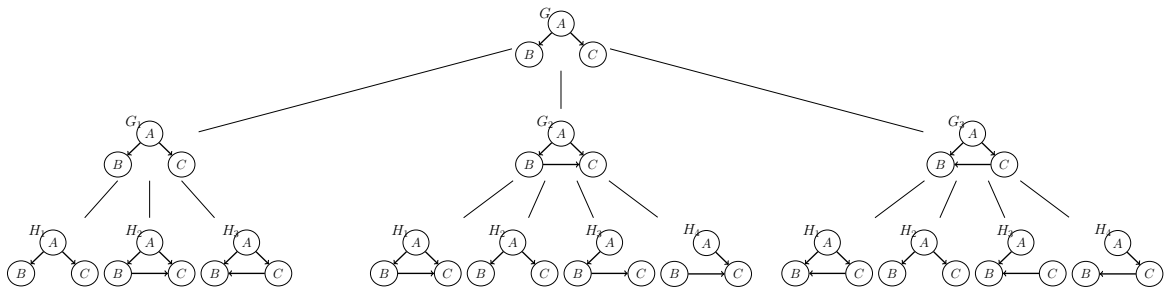


FIGURE 4.4: Note that the thin lines are showing how Bayesian Networks are related. Finding neighborhoods of a Bayesian Network with three nodes: there are 3 possible Bayesian Networks if we apply the MH sampler and 7 possible distinct Bayesian Networks (after excluding similar Bayesian Networks) if we apply the NS.

**Algorithm 5** Sampling graph space with the NS

---

```

1: Initialise a Bayesian network  $G_0$ , and set  $t := 0$ .
2: for all  $t = 0, 1, \dots, n$  do
3:   Given the current graph  $G_t = G$ , find  $\mathcal{N}_G$  and  $\mu_G$ .
4:   Generate  $U \sim \text{U}(0, \frac{f(G)}{\mu_G})$ .
5:   Sample graph  $H_1 \sim \text{U}(\mathcal{N}_G)$  and find its  $\mathcal{N}_{H_1}$ .
6:   for all  $k = 1, 2, \dots, |\mathcal{N}_{H_1}|$  do
7:     Sample graph  $H_{2_k} = H_2 \sim \text{U}(\mathcal{N}_{H_1})$ , and find  $\mathcal{N}_{H_2}$  and  $\mu_{H_2}$ .
8:     if  $\frac{f(H_2)}{\mu_{H_2}} \geq U$  then
9:       set  $G_{t+1} = H_{2_k}$ 
10:      goto 2
11:     else
12:       Exclude  $H_{2_k}$  from  $\mathcal{N}_{H_1}$ .
13:      goto 6
14:     end if
15:   end for
16: end for

```

---

## 4.7 Hit-and-Run sampler

This section proposes a new version of the HAR sampler that was reviewed in Section 2.4.3. Unlike the original HAR sampler, the new sampler is used to generate samples from *discrete* graph spaces. At iteration  $t$ , the sampler assumes that the next iterated graph  $G_{t+1}$  is defined by the current graph  $G_t$ ,  $\ell$  and  $p$ , where  $p$  is a random path that represents a sequence of graphs and  $\ell$  is the length of  $p$ , where  $\ell \geq 1$ . The new sampler preserves many of the attributes of the original HAR sampler reviewed in Chapter 2.

The following subsections are constructed as follows. Section 4.7.1 provides some notation and explains how to define a single path of length  $\ell$  in a graph space of connected Bayesian networks. Section 4.7.2 discusses the diameter of a graph space of BNs and proposes some upper bounds for the length of a single path  $p$  in  $\mathcal{X}$ . Section 4.7.3 explains the algorithm for the new sampler, including the rejection step and the proposal distribution.

### 4.7.1 Constructing a path in a space of graph

A path  $p$  in a space of graphs  $\mathcal{X}$  is defined as a sequence of graphs in which each graph is adjacent to the graph that precedes it. Let  $\lambda$  be the maximum number of graphs permitted on a single path  $p$  in  $\mathcal{X}$ . The value of  $\lambda$  is a positive integer and is fixed for all iterations. Let  $\ell$  be a positive integer number which is strictly less than or equal to  $\lambda$ . Let  $p_\ell$  be a particular path of length  $\ell$ . Algorithm 6 is a pseudocode describing the path construction.

---

#### Algorithm 6 Constructing a path in a space of graphs $\mathcal{X}$

---

- 1: Initialise a graph  $H_1$ .
  - 2: Define an integer length  $\ell \geq 2$ .
  - 3: Define the path  $p_\ell$ , and consider  $H_1 \in p_\ell$ .
  - 4: **for all**  $k = 2, \dots, \ell$  **do**
  - 5:   Sample  $H_k \in \mathcal{N}(H_{k-1})$
  - 6:   Store  $H_k \in p_\ell$ .
  - 7: **end for**
  - 8: Return  $p_\ell$ .
- 

That is, to construct a path  $p_\ell$  of length  $\ell$  in a space of graphs  $\mathcal{X}$ , first set  $H_1$  to be the first graph on the path  $p_\ell$  and set  $k = \{2, \dots, \ell\}$ . Second, move from  $H_1$  to the second graph  $H_2 \in p_\ell$  which is one of its adjacent graphs, so that  $H_2 \in \mathcal{N}(H_1)$ .

Third, keep generating graphs for each  $k$  until reaching  $H_\ell \in \mathcal{N}(H_{\ell-1})$ . Each graph on the path  $p_\ell$  is sampled uniformly from a collection of graphs that are adjacent to the preceding graph on the path  $p_\ell$ .

**Remark 3.** Note that every iteration in the MH sampler contains a short path of length two graphs:  $G$  and  $G' \in \mathcal{N}(G)$ . In the NS, every single iteration contains a path of length three graphs:  $G$ ,  $G' \in \mathcal{N}(G)$  and  $G'' \in \mathcal{N}(G')$ .

In the HAR sampler, the length  $\ell$  of a path is randomly generated at every iteration. Therefore,  $\ell$  is treated as a random variable, and accordingly it is necessary to define a discrete distribution  $f(\ell)$ . One possible distribution is the discrete uniform distribution, with appropriately selected lower and upper bounds. To ensure the sampler has the potential to move, the lower bound is set to one. That is, there is at least a length  $\ell = 2$  for a particular path  $p$ . Note it is still possible for the sampler to retain the current graph if the proposed graph is not accepted by the rejection step.

The upper bound  $\lambda$  is optionally determined by the user. Note that setting  $\lambda$  to a high number may result in significant processing delays. In this thesis, I set a maximum path length of five as default. This allows for greater difference between consecutively sampled graphs than the NS or MH.

### 4.7.2 The diameter of a space of graphs

The diameter of the space in a connected BN is influenced by two main factors. First, graphs are constrained to satisfy certain structural constraints such as connectivity and acyclicity. Second, neighbour relations in graph space are determined by the allowed adjustments to edges such as insertion, deletion or reversal. The adjustment using reversal has the potential to improve efficiency by using shorter path lengths than is possible by only considering adding and deleting.

**Example 2.** Consider the left-most directed graph in Figure 4.5. The graph is sparse and is connected by the minimum number of edges. Consider how many steps are required to reach the right-most graph in Figure 4.5. Let's first consider only the operations of adding and deleting adjustments: one might directly delete  $A \rightarrow B$  in the left-most graph and then add  $B \rightarrow A$  to get the right-most graph. This, however, is not permitted because deleting  $A \rightarrow B$  would disconnect the graph. Alternatively, one might add  $A \rightarrow D$ , for example, in order to make  $A \rightarrow B$  deletable as shown in Figure 4.5. Next, add the edge  $B \rightarrow A$  before deleting  $A \rightarrow D$  again, so that the length of this path is now four. However, if we consider the reversal adjustment, the movement from the left graph to the right graph can be directly carried out by reversing the edge  $A \rightarrow B$ . The length of the path has been reduced to one.

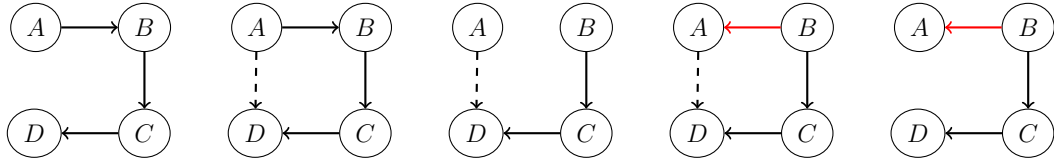


FIGURE 4.5: Moving from the left-most graph to the right-most graph in four steps considering only adding and deleting edges.

■

**Theorem 4.7.1.** A lower bound to the diameter of a graph space of connected BNs is  $\frac{n(n-1)}{2} + 1$ , for  $n \geq 4$ .

**Proof:**

The idea is to start with a graph consisting of a path connecting all nodes, and finish with a graph that contains all the edges pointing back from later to earlier nodes in this path (see Figure 4.6 for the case  $n = 4$ ). The latter graph is connected for  $n \geq 4$  (but not for  $n = 3$ ). The latter graph is also acyclic because any directed path in this graph can only have a decreasing sequence of labels, if the nodes are labelled with their order in the first graph. One needs to delete  $(n - 1)$  edges and



add  $\left\lceil \frac{n(n-1)}{2} - (n-1) \right\rceil$  edges, so at least  $\frac{n(n-1)}{2}$  additions and deletions are required. But the first move cannot be a deletion because that disconnects the graph, and cannot be an addition in the correct direction, since that would create a cycle, so one needs at least one reversal. ■

**Example 3.** Consider the pair of graphs  $G_1$  and  $G_2$  in Figure 4.6. To transit from  $G_1$  to  $G_2$  or from  $G_2$  to  $G_1$ , requires at least three additions and three deletions. However, the first transition cannot delete any edges without disconnecting the graph, and cannot add any of the new edges that are needed without creating a cycle. Thus, at least one reverse and at least 7 transitions in total are required. In fact only 7 transitions are required: if one starts by reversing  $A \rightarrow B$ , each subsequent move can be either an addition or a deletion.

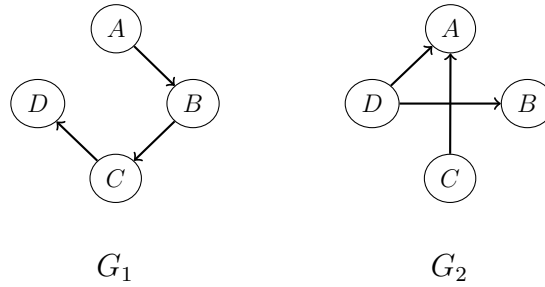


FIGURE 4.6: A pair of graphs that requires at least  $\frac{n(n-1)}{2} + 1$  transitions.

■

Theorem 4.7.2 proves the upper bound to the diameter of a graph space of connected BNs.

**Theorem 4.7.2.** An upper bound to the diameter of a graph space of connected BNs is  $n(n-1)$ .

**Proof:**

Consider any two connected, directed, acyclic graphs on the same nodes,  $G_1 = (V, E_1)$  and  $G_2 = (V, E_2)$ . Consider three sets of edges.  $R$  is the set of edges that are in  $E_1$  and the reverse edge is in  $E_2$  — these are the edges that need to be

reversed.  $D$  is the set of edges in  $E_1$  that are not in  $E_2$  and not in  $R$  — these are the edges that need to be deleted. Finally,  $A$  is the set of edges in  $E_2$  that are not in  $E_1$  and their reverse is not in  $R$  — these are the edges that need to be added. Note  $R$ ,  $D$  and  $A$  do not intersect.

First consider the edges in  $D$ . Order them and work through the list deleting them until you find an edge  $e_1$  that can't be deleted without disconnecting the graph. This edge must be a bridge between two otherwise disconnected components  $H_1$  and  $H_2$  of the current graph. Since  $e_1$  is not in  $E_2$ , but  $G_2$  is connected, there must be an edge  $e_2$  in  $A$  connecting  $H_1$  and  $H_2$ . If  $e_2$  connects  $H_1$  and  $H_2$  in the same direction as  $e_1$ , then  $e_2$  can be added without creating any cycles and now  $e_1$  can be deleted without disconnecting the graph. On the other hand, if  $e_1$  and  $e_2$  connect  $H_1$  and  $H_2$  in opposite directions, first reverse  $e_1$ , then add  $e_2$ , then delete  $e_1$ . Either way,  $e_1$  is deleted and  $e_2$  is added to the graph and removed from  $A$ . Continue in this manner until there are no edges left in  $D$ .

Next consider the edges in  $R$ . Order them and work through the list reversing them until you find an edge  $e$  that can't be reversed without creating a cycle. There must therefore be an alternative path connecting the endpoints of  $e$  in the same direction as  $e$ . Thus  $e$  can be deleted without disconnecting the graph. Do this, and add the reverse of  $e$  to the set  $A$  (to be added later, since adding it now would create a cycle). Continue in this manner until there are no edges left in  $R$ . The resulting graph must be a sub-graph of  $G_2$ , since edges only need to be added to create  $G_2$ . These edges can now all be added without creating cycles.

Note each edge  $e$  in the original set  $D$  has been involved in at most two transitions: a reversal and a deletion. Also note that the reverse of  $e$  is not in  $E_1$  (since that would create a cycle in  $G_1$ ) or in  $E_2$  (since then  $e$  would be in  $R$ , not  $D$ ). Hence the reverse of  $e$  does not need to be added, reversed or deleted. Thus the edge  $e$  and its reverse together contribute at most two transitions.

Similarly, each edge  $e$  in the original set  $R$  has been involved in at most two transitions: a deletion and an addition. Also note that no additional operations have to be performed involving either  $e$  or its reverse. Thus the edge  $e$  and its reverse together contribute at most two transitions.

Since there are at most  $n(n-1)$  possible directed edges, and these can be split into  $\frac{n(n-1)}{2}$  pairs of an edge and its reverse, at most  $\frac{2n(n-1)}{2}$  transitions are required.

■

Figure A.1 in Section A.2 is an illustrative example showing all possible paths of length three for a connected graph consisting of three nodes.

From a practical point of view, it is computationally infeasible to enumerate all paths from a given graph. This would dramatically increase simulation times. For all the applications in this thesis, we set the maximum length to 5 as default. This number facilitates traversing into graph space farther than the MH and NS, and thus generates samples which are less dependent. Again, this technique in principle would facilitate exploring graph spaces better than the MH.

### 4.7.3 Algorithm

In order to sample graphs from a discrete space  $\mathcal{X}$  using the HAR sampler, consider the following six parameters:  $\lambda$ ,  $\ell$ ,  $G_t$ ,  $\mu_{G_t}$ ,  $H_\ell$ , and  $\mu_{H_\ell}$ . First randomly initialise a graph  $G_0$  at  $t = 0$ . Then define an integer upper bound  $\lambda$  which will be fixed for all iterations. Given the current graph  $G_t$ , find  $\mathcal{N}_{G_t}$  and  $\mu_{G_t}$ . Generate a value  $U$  uniformly from  $(0, 1)$ . Sample an integer length  $\ell$  uniformly between  $[1, \lambda]$ . Set  $H_0 = G_t$ , and then sample a sequence of graphs in which  $H_k \sim \mathcal{U}(\mathcal{N}_{H_{k-1}})$  starting from  $H_0$  until the proposed graph  $H_\ell$  is sampled. Find  $\mathcal{N}_{H_\ell}$  and  $\mu_{H_\ell}$ . Then, accept the graph  $H_\ell$  or reject it based on the acceptance ratio illustrated in Step 13 in Algorithm 7. The latter algorithm describes how to sample discrete graph spaces using the HAR sampler.

---

**Algorithm 7** Sampling graph space with the HAR
 

---

```

1: Initialise a Bayesian network  $G_0$ , and set  $t := 0$ .
2: Define an integer length  $\lambda$ 
3: for all  $t = 0, 1, \dots, n$  do
4:   Given the current graph  $G_t$ , find  $\mathcal{N}_{G_t}$  and  $\mu_{G_t}$ 
5:   Generate  $U \sim \mathcal{U}(0, 1)$ .
6:   Sample an integer  $\ell \sim \mathcal{U}(1, \lambda)$ 
7:   Set  $H_0 = G_t$ 
8:   for all  $k = 1, \dots, \ell$  do
9:     Sample graph  $H_k \sim \mathcal{U}(\mathcal{N}_{H_{k-1}})$ 
10:    Find  $\mathcal{N}_{H_k}$ 
11:  end for
12:  Find  $\mu_{H_\ell}$ 
13:  if  $\frac{f(H_\ell)}{f(G_t)} \cdot \frac{\mu_{G_t}}{\mu_{H_\ell}} \geq U$  then
14:    set  $G_{t+1} = H_\ell$ , goto 3
15:  else
16:    set  $G_{t+1} = G_t$ , goto 3
17:  end if
18: end for

```

---

**Theorem 4.7.3.** The Markov chain generated by Algorithm **7** is *irreducible*.

**Proof:**

A Markov chain is irreducible if any two graphs of this chain communicate. That is, for any two connected BNs, there is a positive probability of selecting a sequence of transitions connecting those BNs. Using Theorem **4.7.2**, it is always possible to transform one connected BN into another by a sequence of additions, deletions and reversals. The opposite transformation is also true. Moreover, each transition occurs

with positive probability. This implies that the Markov chain is irreducible. ■

**Theorem 4.7.4.** The Markov chain generated by Algorithm 7 is *aperiodic*.

**Proof:**

For each of the samplers described here, there is a positive probability that the proposed new graph will be the current graph  $G_t$ . Thus there is a positive probability of immediately repeating the same state, implying aperiodicity. ■

#### 4.7.4 Acceptance-rejection ratio in the HAR

The original HAR sampler uses the MH acceptance-rejection ratio step which comprises two probability distributions. First, the target distribution  $f$ , which is often a posterior distribution formed from prior knowledge combined with an observational dataset. Second, the transition proposal  $q$  defined for two neighbouring graphs. The MH acceptance-rejection ratio is compared with a sampled uniform probability value between 0 and 1 as shown in Step 13 in Algorithm 7, and the proposal is accepted if:

$$\frac{f(H_\ell)}{f(G_t)} \cdot \frac{q(H_{\ell-1}, H_{\ell-2}, \dots, H_2, H_1, G_t | H_\ell)}{q(H_1, H_2, \dots, H_{\ell-2}, H_{\ell-1}, H_\ell | G_t)} \geq U \sim \mathcal{U}(0, 1) \quad (4.1)$$

Note that Equation 4.1 computes the proposal probability for a single path instead of summing over all paths, and the two values  $q(H_{\ell-1}, H_{\ell-2}, \dots, H_2, H_1, G_t | H_\ell)$  and  $q(H_1, H_2, \dots, H_{\ell-2}, H_{\ell-1}, H_\ell | G_t)$  are the transition probabilities from graph  $H_\ell$  to graph  $G_t$  and from graph  $G_t$  to graph  $H_\ell$ , respectively.

**Proposition 6.** Given two graphs  $G_t$  and  $H_\ell$ , the ratio of their proposals is:

$$\frac{q(H_{\ell-1}, H_{\ell-2}, \dots, H_2, H_1, G_t | H_\ell)}{q(H_1, H_2, \dots, H_{\ell-2}, H_{\ell-1}, H_\ell | G_t)} = \frac{\mu_{G_t}}{\mu_{H_\ell}}. \quad (4.2)$$

**Proof**

$$q(H_{\ell-1}, H_{\ell-2}, \dots, H_2, H_1, G_t | H_\ell) = \frac{1}{\mu_{H_\ell}} \times \frac{1}{\mu_{H_{\ell-1}}} \times \frac{1}{\mu_{H_{\ell-2}}} \times \dots \times \frac{1}{\mu_{H_2}} \times \frac{1}{\mu_{H_1}} \quad (4.3)$$

$$q(H_1, H_2, \dots, H_{\ell-2}, H_{\ell-1}, H_\ell | G_t) = \frac{1}{\mu_{G_t}} \times \frac{1}{\mu_{H_1}} \times \frac{1}{\mu_{H_2}} \times \dots \times \frac{1}{\mu_{H_{\ell-2}}} \times \frac{1}{\mu_{H_{\ell-1}}} \quad (4.4)$$

We divide Equation 4.3 by Equation 4.4 in Equation 4.5:

$$\therefore \frac{q(H_{\ell-1}, H_{\ell-2}, \dots, H_2, H_1, G_t | H_\ell)}{q(H_1, H_2, \dots, H_{\ell-2}, H_{\ell-1}, H_\ell | G_t)} = \frac{\frac{1}{\mu_{H_\ell}}}{\frac{1}{\mu_{G_t}}} = \frac{\mu_{G_t}}{\mu_{H_\ell}}. \blacksquare \quad (4.5)$$

**Example 4.** Figure 4.7 displays four descendant graphs  $\{G_1, G_2, G_3, G_4\}$  sampled from uniform distributions in two opposite directions. For instance, Figure 4.7 shows that  $G_3 \sim \mathcal{U}(\mathcal{N}_{G_4})$  so  $q(G_3|G_4) = \frac{1}{5}$ , because  $\mu_{G_4} = 5$ , and  $G_4 \sim \mathcal{U}(\mathcal{N}_{G_3})$  so  $q(G_4|G_3) = \frac{1}{4}$ , because  $\mu_{G_3} = 4$ . Figure 4.7 specifically shows how to transit from graph  $G_1$  (red color) to graph  $G_4$  (blue color) and vice versa with a path length  $\ell = 3$ .

Equation 4.6 computes the ratio probability between the two proposals from graph  $G_1$  to graph  $G_4$  and from graph  $G_4$  to graph  $G_1$ . The result in Equation 4.6 verifies Proposition 6. Note that the fractions placed above graphs in Figure 4.7 represent the transition probabilities to those graphs from the graphs that precede them. Note also that these fractions consider the probabilities that the proposed new graphs can be the same as the current graphs at each stage.

$$\frac{q(G_1, G_2, G_3 | G_4)}{q(G_4, G_3, G_2 | G_1)} = \frac{\frac{1}{6} \times \frac{1}{6} \times \frac{1}{4}}{\frac{1}{6} \times \frac{1}{4} \times \frac{1}{5}} = \frac{5}{6}. \quad (4.6)$$

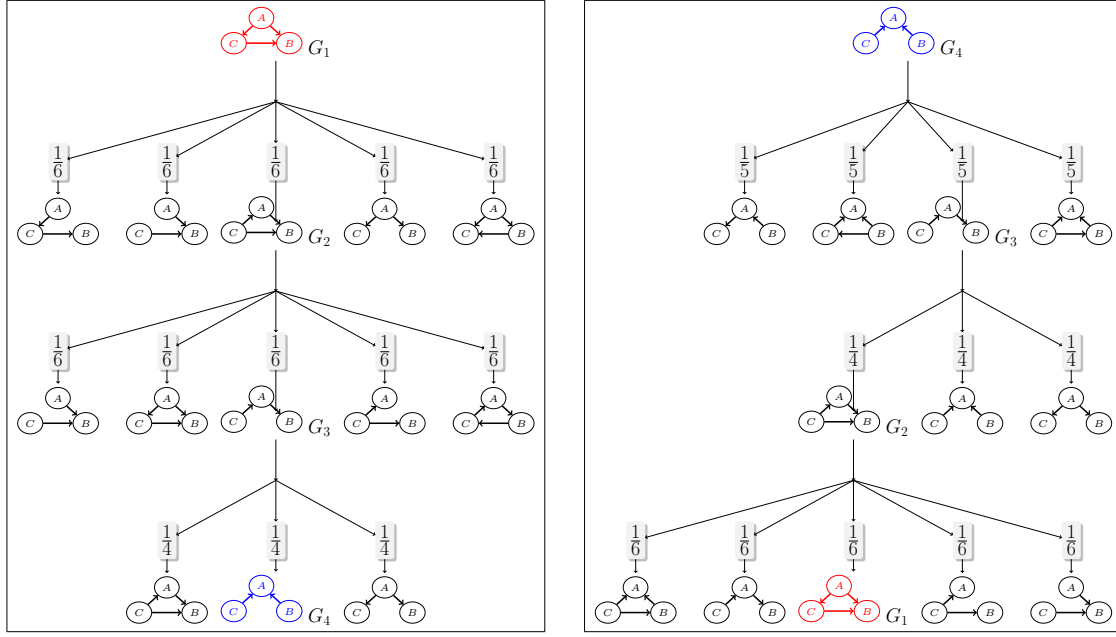


FIGURE 4.7: Calculating probabilities uniformly between two distant graphs using the Hit-and-Run sampler. Beside the number of descendant graphs at each stage, a value of 1 was added to each denominator to allow for choosing the same graph.

■

## 4.8 Generating an initial network at random

For all MCMC samplers in this thesis, I used the following simple approach to generate a random initial network.

1. Declare an empty network,  $G(V, E)$ , where  $V$  is the set of nodes,  $N = |V|$ , and  $E$  is an empty edge set.
2. Choose a pair  $(i, j)$  uniformly i.e. all such edges have equal probabilities, where  $i$  is not equal to  $j$  (to avoid diagonal entries).
3. If  $(i, j)$  does not belong to  $E$  already, then randomly decide whether to add an edge or not, with equal probability. Otherwise, do nothing.

4. If the decision is to add an edge, and adding the edge  $(i, j)$  would not violate constraints e.g. cyclicity, include that edge in the graph and go to Step 6. Otherwise, go to Step 5.
5. If the edge  $(i, j)$  is rejected at Step 4, repeat Step 4 for  $(j, i)$  with the same constraint checking.
6. Repeat steps from 2 to 5 for  $N^2$  times.
7. After repeating Steps 2-5  $N^2$  times, check whether the resultant graph is a connected one or not. For that purpose we applied BFS (Breadth First Search) algorithm to test network connectivity. If the graph is connected, then we have DAG, and go to Step 9. Otherwise, go to Step 8.
8. Repeat the whole process Steps 1-7 until finding a connected, directed, acyclic graph satisfying parents and children constraints.
9. End.

In Step 2, each pair  $(i, j)$  has equal probabilities to be considered for the Step 3 i.e. forming an edge between them or not forming an edge between them. Note that in Step 3, a pair  $(i, j)$  for which an edge does not belong to  $E$ , random deciding (with equal probabilities) whether to add an edge between them or not, would likely facilitate better randomness in terms of the graph structure compared to the choice of merely adding an edge which may always yield denser graph structure. Also note that we repeat Steps 2-5 for  $N^2$  times because there are  $N^2$  node pairs in total for  $N$  nodes, and essentially each node pair  $(i, j)$  has equal chance to be considered for further action (Step 2).



## 4.9 Conclusion

This chapter has explained the theoretical framework of the MH, NS and HAR when they are used to sample and infer Bayesian network structures given a discrete dataset. It has attempted to facilitate better understanding by providing a range of illustrative examples, useful remarks, theorems, pseudocode algorithms and figures. So that the proposed MCMC methods are applicable in practice and possibly can be extended in future work to achieve better inferring of BNs.

# Chapter 5

## Uniform Sampling of Bayesian Networks Conditional on Vertex Connectivity <sup>1</sup>

### 5.1 Introduction

Random graphs (RGs) are the subject of a broad area of research originating in the early works of [145] and [146], drawing insights from both graph theory and probability. One particular area of interest is the generation of random graphs. Many applications in graph analysis require a space of graphs or networks to be sampled uniformly at random [147–150]. Sampling random graphs in an unbiased manner over a space of graphs is still a challenge, and has opened a new area of research. The main problem is that many random processes used to generate random graphs do not effectively allow graphs to have the same chance of being drawn [151, 152]. Two commonly used approaches to generate random graphs are by using a probability distribution, or by using a random process [153]. An example of using probability distributions is the Erdős-Reényi model [154], which generates random

---

<sup>1</sup>This chapter is modified from a published paper entitled "Uniform sampling of directed and undirected graphs conditional to vertex connectivity" by Salem A. Alyami, AKM Azad, and Jonathan M. Keith, *Electronic Notes in Discrete Mathematics* 53, 43-55, 2016.

graphs with a uniform probability distribution over the space of all graphs with a given number of nodes and edges. The Erdős-Reényi model can be implemented by starting with a given number of nodes with no edges and then iteratively adding one new edge at a time, sampled uniformly over the set of missing edges, until the required number of edges is obtained [153]. An example of using random processes is the simple algorithm used by [152] to simulate BNs (directed acyclic graphs) in which the number of nodes and their average degree are taken as an input. The algorithm then computes a threshold value  $t$ , and for each pair of nodes, a random number  $r \in [0, 1]$  is generated. If  $r \leq t$ , then the pair is linked by an edge.

There are two main goals of this chapter. First, the uniform distribution is used to compare samplers' performance and their computational efficiency to sample BNs. Second, it aims to provide a novel implementation of MCMC samplers that are efficiently capable of sampling graphs that are uniformly distributed over certain spaces. Specifically, I consider spaces consisting of directed acyclic graphs in which all vertices are connected. Such graphs arise in a variety of applications, in particular in the study of BNs.

Section 5.2 specifies the MCMC methods and their sampling model used to generate BNs at random. Experimental results are provided in 5.3.

## 5.2 Methods and model

In this chapter, I use three MCMC samplers to sample BNs uniformly at random: the MH, NS and HAR conducted using Algorithm 4, Algorithm 5 and Algorithm 7, presented in Chapter 4, respectively. Sampling graphs from a finite search space  $\mathcal{X}$  using a discrete uniform distribution assumes that all graphs in  $\mathcal{X}$  are equally probable. I assume that the uniform distribution in Equation 5.1 has been defined

over a space of BNs.

$$f(N, E) = \frac{1}{|\mathcal{X}|}. \quad (5.1)$$

In fact, one can use an unnormalised target function  $f(X) = 1$ , since the normalisation constant can be canceled in all of the required comparisons. I validated the MCMC samplers and software by testing whether a MCMC sampler is able to explore the entire space and whether the sampled graphs match the target uniform distribution. I also investigated convergence behaviour by performing simulations with the number of iterations ranging from 1000 up to 50 000 000.

## 5.3 Experimental results

The structure of this section is as follows. Section 5.3.1 discusses the effects on the frequency distribution, if one considers some or all of the transition options available for a MCMC sampler to move from a current to next graph. Section 5.3.2 investigates how many distinct graphs can be sampled by each MCMC sampler in  $|\mathcal{X}|$  iterations. Section 5.3.3 uses the MCMC samplers to explore the entire spaces of feasible BNs, and checks that the sampled graphs match the target uniform distribution. Section 5.3.4 reports the SSDs for the sampled frequencies at different settings.

### 5.3.1 Transition options

Moving from a particular graph to one of its adjacent graphs is typically an essential step when a heuristic sampler is implemented to sample graphs or networks. A commonly used technique is to add a non-existing edge to the current graph or to delete an existing edge from it. I use AD to refer to the set of all possible addable and deletable edges (recall that after adding or deleting an edge, the graph remains a connected DAG). The AD set is applicable for both directed and undirected graphs. Note that in directed graphs,  $(i, j)$  and  $(j, i)$  can both be distinct members of AD.

Reversing an existing edge is another possible option for modifying the current graph to transit to one of its neighbors. This option is only applicable for directed graphs (Bayesian networks). I use ADR to refer to the set of all possible addable, deletable and reversible edges. Note  $|\text{ADR}| \geq |\text{AD}|$ . The effect of using the AD set versus the ADR set while using MCMC methods is investigated below.

To assess the sampling methods, I consider a Bayesian network with four nodes. The graph space contains 446 connected BNs. I evaluate the ability of our MCMC samplers to explore the entire 446 graphs, and evaluate how effectively their frequencies match the uniform distribution. I ran 100,000 iterations twice for each MCMC sampler. The first 100,000 was run with the AD set for all samplers, and the second 100,000 with the ADR set for all samplers.

Each row in Figure 5.1 compares the performance of each MCMC sampler when it considers AD versus ADR. Figure 5.1 shows that the MH method produces graphs with greater variation in sampling frequency using the AD set than using the NS and HAR methods. With the ADR set, all three methods appear to produce an approximately uniform distribution.

That is, the NS is less affected by using the AD set compared to the MH. One possible reason is that the NS produces samples that are less correlated compared to those with the MH. The NS has the chance to, in a single iteration, jump to a graph that differs from the current graph by two edges. The sampler with the ADR set has produced a slightly more uniform distribution compared to its performance with the AD set.

Unlike the NS and MH, the HAR is not significantly affected by using the AD set. It is not possible to visually differentiate between the two frequency distributions produced by the HAR with AD and ADR sets, as shown in Figure 5.1. This is possibly due to its ability to move to a distant graph via a sequence of adjacent graphs within a single iteration, and thus less correlated samples are produced.

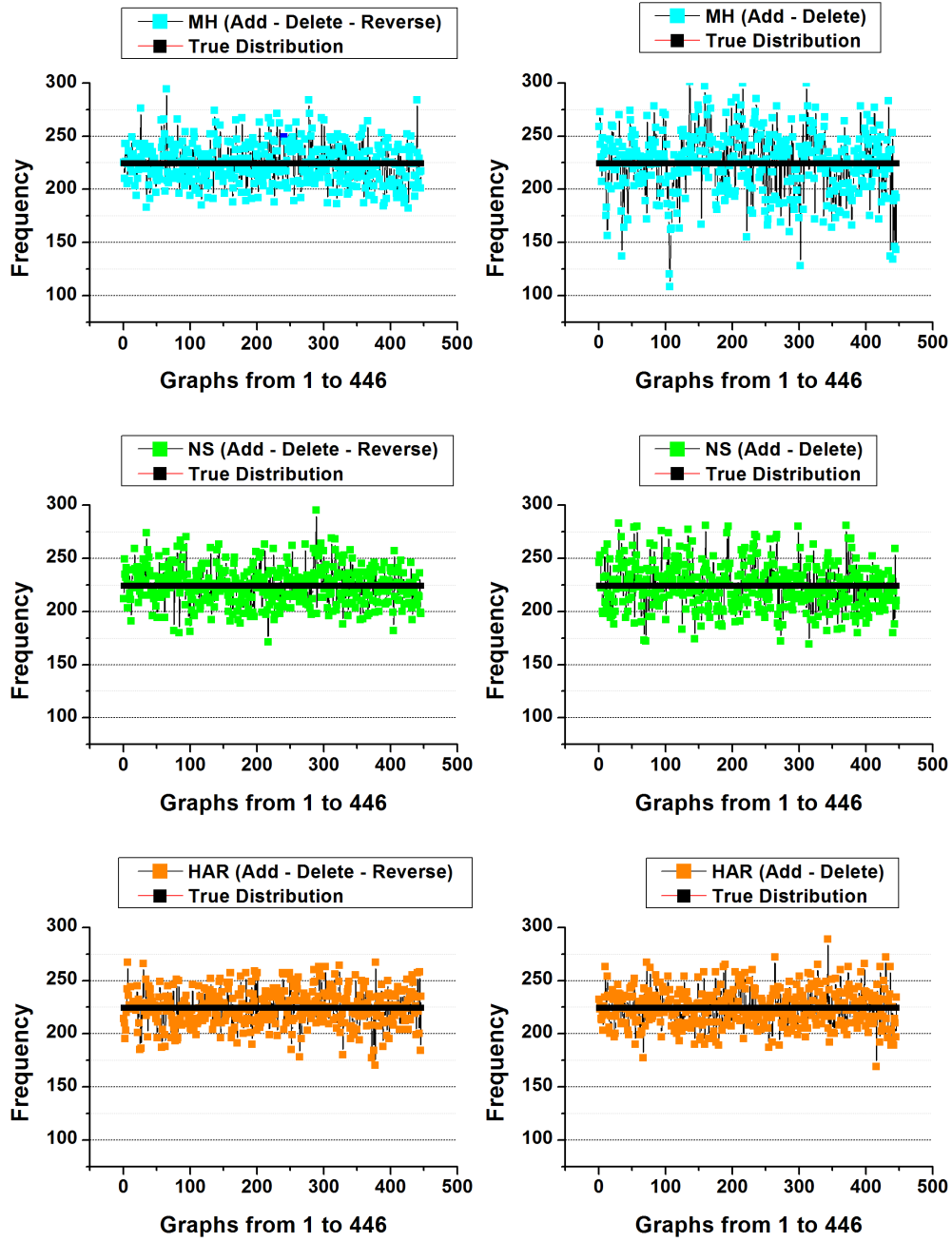


FIGURE 5.1: MH, NS and HAR performance comparisons with AD set versus ADR set with four nodes using 100 000 iterations. The scales of both vertical axis and horizontal axis have been set to the same lower and upper bounds with the same increments for all plots.

Figure 5.2 investigates the performance of the MH sampler. It performs another comparison using the AD set versus the ADR set when the number of iterations has

increased to 500,000.

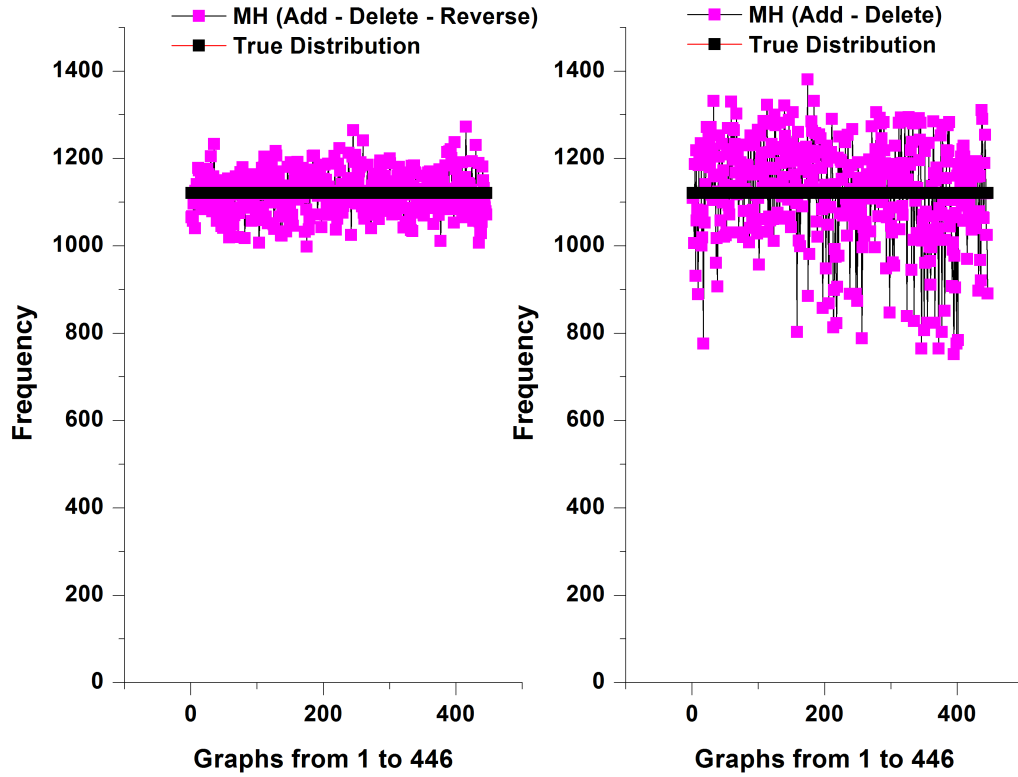


FIGURE 5.2: The MH performance comparison with AD set versus ADR set with four nodes using 500 000 iterations.

Figure 5.2 confirms that the MH sampler using the ADR set produces less variation in sampling proportions for different graphs than using the AD set. There are two possible reasons for this disparity. First, sampling from the AD set instead of the ADR generates more highly correlated graphs. Second, the total number of immediately adjacent graphs is reduced for the AD set, so the sampler is more likely to get stuck in a local mode.

### 5.3.2 Sampling with $|\mathcal{X}|$ iterations

I aim to estimate the total number of graphs (TNGs) generated by each MCMC sampler after running  $|\mathcal{X}|$  iterations. Roughly speaking, the greater the number of

distinct graphs generated by a sampler in  $|\mathcal{X}|$  iterations, the closer the empirical distribution is to uniformity.

The space sizes of four-node and five-node graphs are 446 and 26430, respectively. They are thus small enough to evaluate sampling frequencies for all graphs. The number of iterations in two experiments has been set to these sizes accordingly i.e. 446 and 26430. For each MCMC sampler, I ran 10 chains with 446 iterations in the four-node case and another 10 chains with 26430 iterations in the five-node case. I also fixed random initial networks for all the 20 chains. After running each chain, I recorded the TNGs generated by each sampler separately. Figure 5.3 and Figure 5.4 summarise the obtained TNGs.

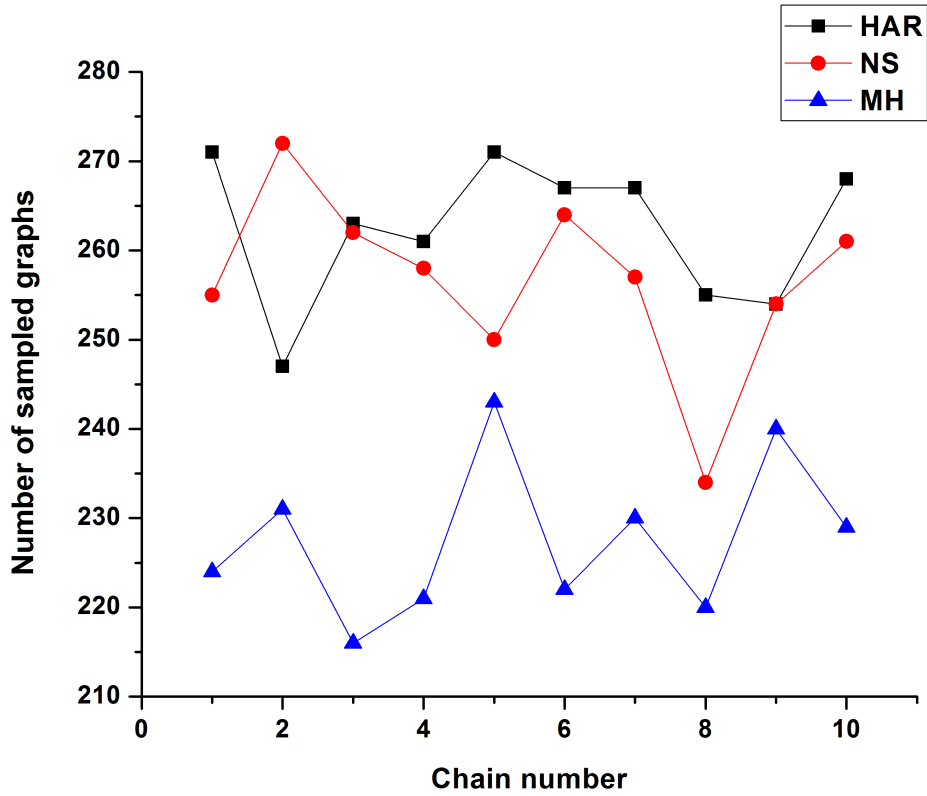


FIGURE 5.3: Sampling the graph space of four nodes with  $|\mathcal{X}| = 446$  iteration.



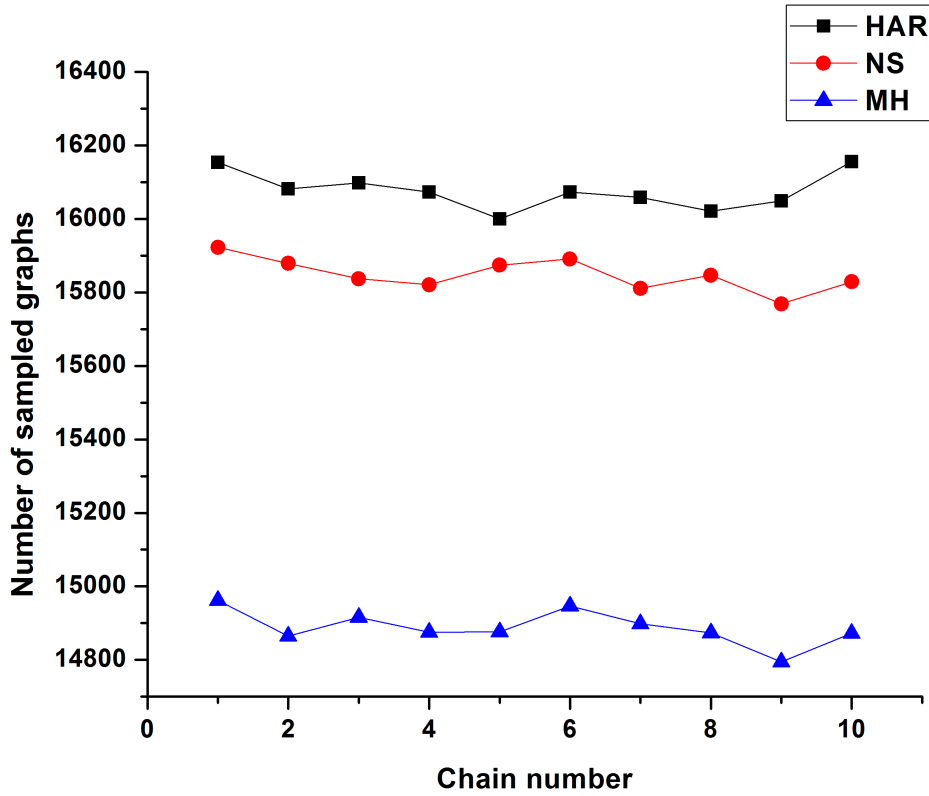


FIGURE 5.4: Sampling the graph space of five nodes with  $|\mathcal{X}| = 26430$  iteration.

The TNGs produced by the NS and HAR are higher than those produced by the MH in both target search spaces. This demonstrates that the NS and HAR have the potential to visit more distinct graphs in the same number of iterations compared to the MH. With the space of four-node graphs, the TNGs produced by the NS and HAR are roughly equal. However, it was rare for the NS to produce TNGs higher than the HAR. With the space of five-node graphs, the HAR is superior in TNGs compared to the NS.

### 5.3.3 Sampling BNs uniformly

In this section, I investigate how well the sampled frequencies fit uniform distributions as the number of iterations increase. The first group of simulations involves BNs with four nodes. There are 446 connected BNs in this space, and I explore

it by using the MH, HAR, and NS algorithms. The three algorithms are run for 1000, 5000, 10 000, 20 000, 50 000, 75 000, 100 000, 250 000, and 500 000 iterations. The expected frequency of each graph given these numbers of iterations should be 2.24, 11.21, 22.42, 44.84, 112.10, 168.16, 224.21, 569.53, and 1121.07, respectively. The expected frequencies for each graph are calculated by dividing every number of iterations by 446.

Figure 5.5, Figure 5.6 and Figure 5.7 show the empirical frequencies with which each graph was sampled by the MH, HAR and NS algorithms, respectively. With all MCMC samplers, the graph space of four nodes has been entirely explored using 5000 iterations as illustrated in the top middle panel of Figure 5.5, Figure 5.6 and Figure 5.7.

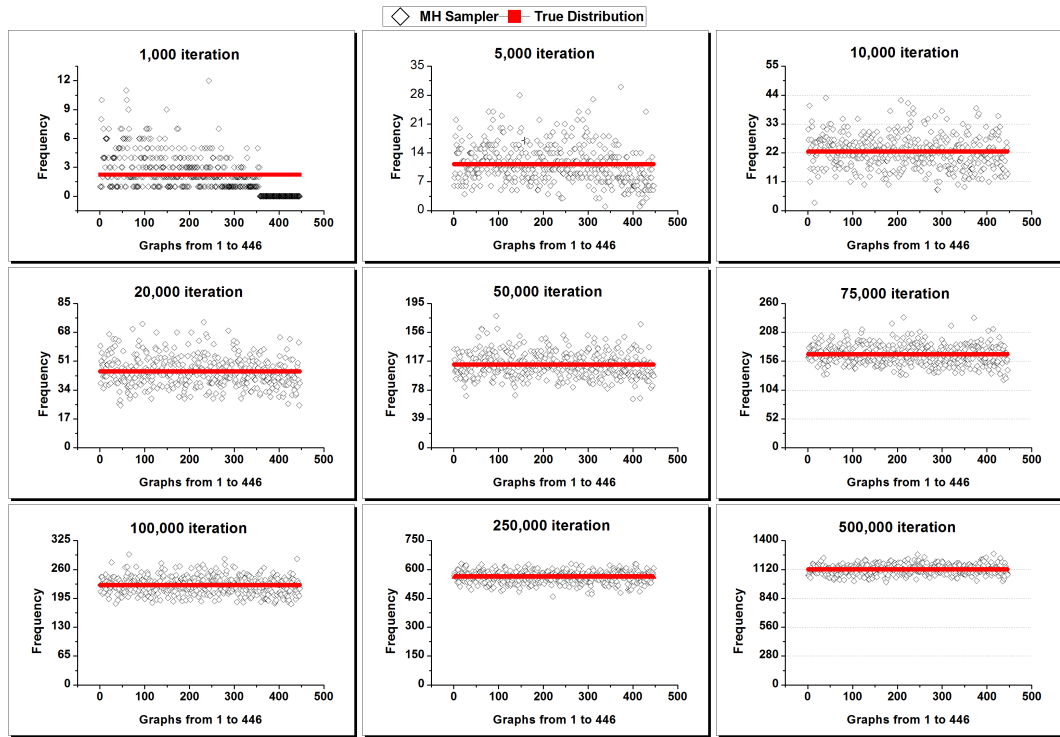


FIGURE 5.5: MH with four nodes for iterations increasing from 1,000 to 500,000.

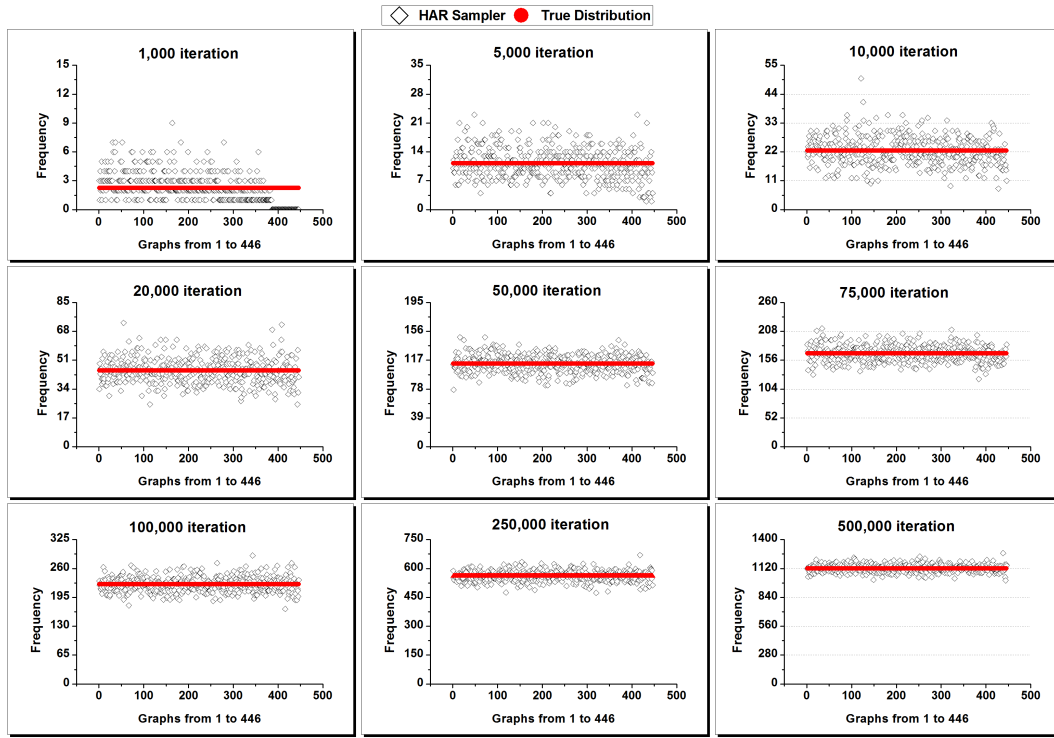


FIGURE 5.6: HAR with four nodes for iterations increasing from 1,000 to 500,000.

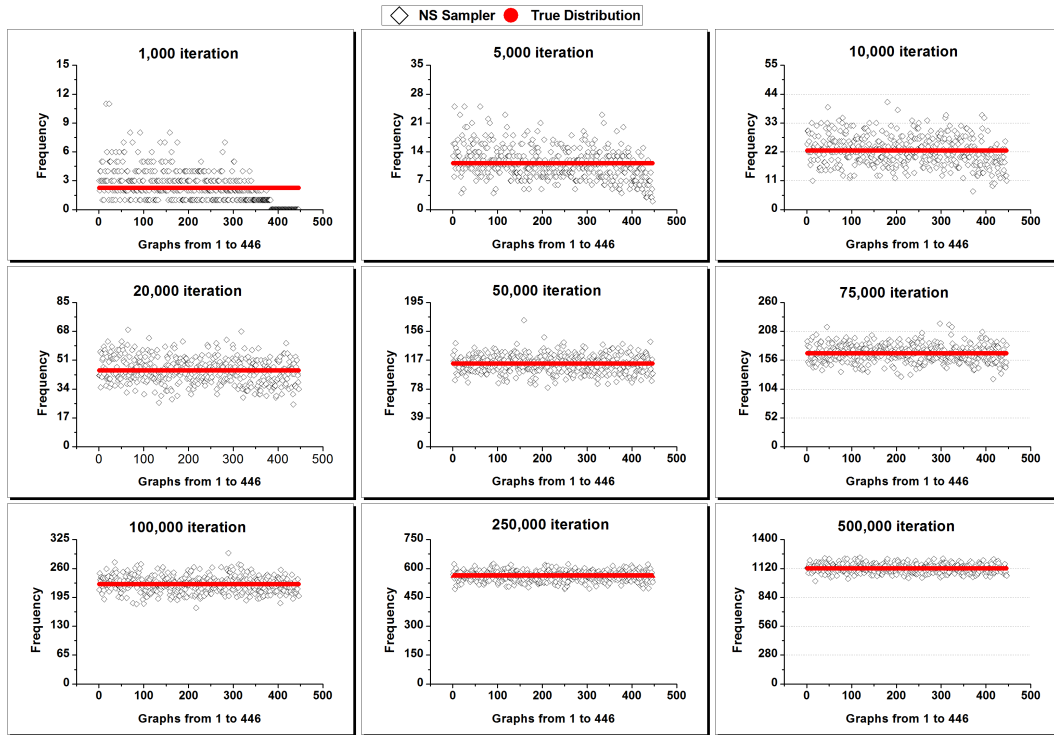


FIGURE 5.7: NS with four nodes for iterations increasing from 1,000 to 500,000.

**Remark 4.** It is possible with the NS and HAR algorithms to explore the whole graph space using less than 5000 iterations (e.g. 2500 iterations) but this will not always sample every graph. The fact that, for all MCMC samplers, the limiting distribution is uniform becomes increasingly apparent as the number of iterations increases from 1000 to 500 000.

The performances of the MH, HAR and NS have been compared also using the Chi-square test to check the goodness of fit for the three MCMC samplers. The Chi square test was applied to the frequencies produced by the 500,000 iterations for the samplers in Figure 5.5, Figure 5.6 and Figure 5.7. The p-values of the MH, HAR and NS are 0.03, 0.01 and 0.007, respectively. The three samplers may therefore be considered to have returned frequencies consistent with the uniform distribution. Consequently, the three samplers may be considered to have converged.

The second group of simulations involves connected BNs with five nodes. There are 26430 CDAGs in this space, and I explore it by using the NS, HAR, and MH algorithms. The MCMC samplers are run for 50 000, 100 000, 250 000, 500 000, 1 000 000, 5 000 000, 10 000 000, 25 000 000, and 50 000 000 iterations. The expected frequency of each graph given these numbers of iterations should be 1.89, 3.78, 9.45, 18.91, 37.83, 189.17, 378.35, 945.89, and 1891.78, respectively. The expected frequencies for each graph are calculated by dividing every number of iterations by 26430.

Figure 5.8, Figure 5.9 and Figure 5.10 show the empirical frequencies with which each graph was sampled by the MH, HAR and NS algorithms, respectively. With all MCMC samplers, the graph space of five nodes has been entirely explored using 250 000 iterations as illustrated in the top right panel of Figure 5.8, Figure 5.9 and Figure 5.10. The simulation results suggest that the MCMC samplers can rapidly explore the entire graph space of 26540 graphs and that these graphs are uniformly distributed, as intended.

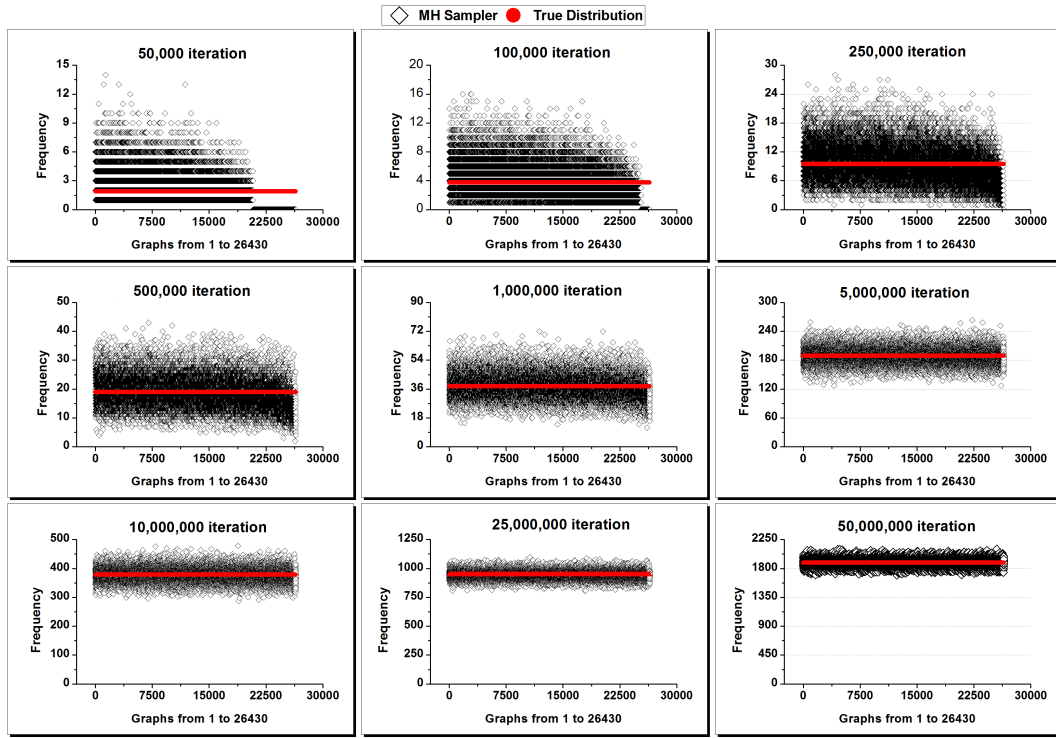


FIGURE 5.8: MH with five nodes for iterations increasing from 50,000, to 50,000,000.

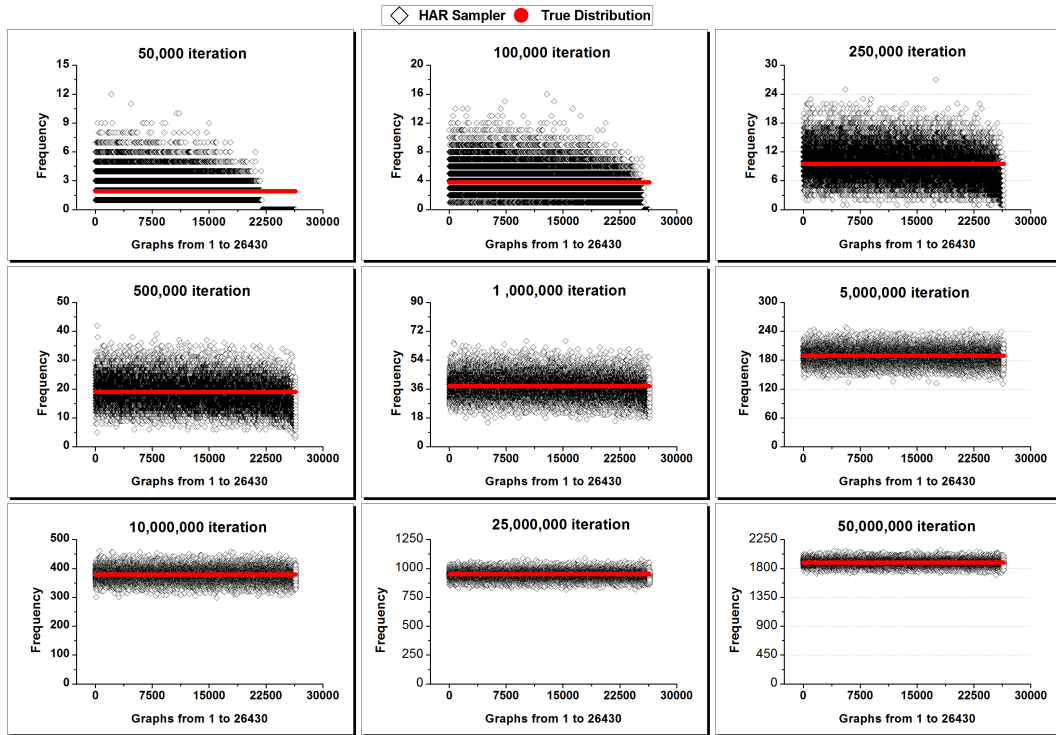


FIGURE 5.9: HAR with five nodes for iterations increasing from 50,000, to 50,000,000.

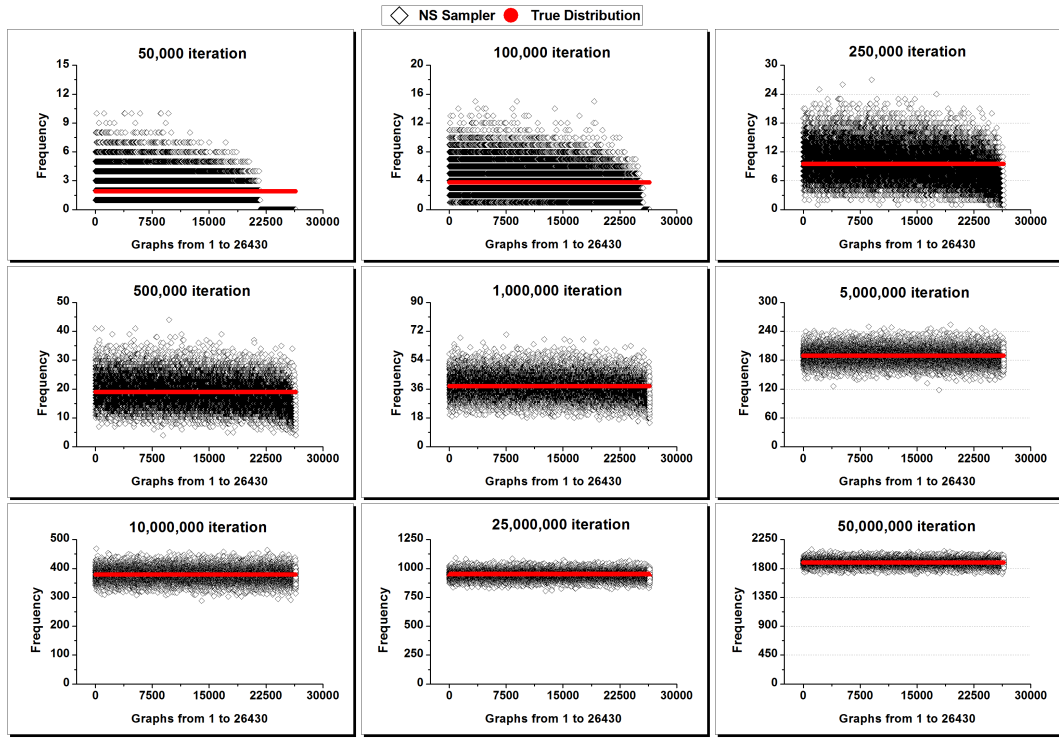


FIGURE 5.10: NS with five nodes for iterations increasing from 50,000, to 50,000,000.

**Remark 5.** Appendix B plots the frequency histograms for the samples obtained by the MH, HAR and NS in Figure 5.5, Figure 5.6, Figure 5.7, Figure 5.8, Figure 5.9 and Figure 5.10. It is noted that the bars in the histograms are symmetrically distributed around the *mean* value in Equation 5.2 when the number of iterations is large.

$$mean = \frac{\#iterations}{|\mathcal{X}|}. \quad (5.2)$$

Some descriptive statistics (mean and standard deviation - SD) were used to summarise the frequencies obtained by the MH, HAR and NS in Figure 5.8, Figure 5.9 and Figure 5.10. The mean and SD were specifically applied to the frequencies produced by the 50,000,000 iterations shown in Figure 5.8, Figure 5.9 and Figure 5.10.

Table 5.1 provides these summary statistics for each MCMC sampler. The means of the MH, HAR and NS are similar and resemble the true value. However, the SDs

of the HAR and NS distributions demonstrate less dispersion compared to the SD of the MH sampler.

Sampler	Mean	SD
<b>MH</b>	1891.79	51.95
<b>HAR</b>	1891.79	46.40
<b>NS</b>	1891.79	47.26

TABLE 5.1: Summary statistics of frequencies after 50,000,000 iterations.

### 5.3.4 Sum of squared differences

I calculate the SSDs between the uniform target distribution and the sampled graph frequencies obtained in Section 5.3.3, at the pre-designated number of iterations. One would expect that the SSD values should ultimately decrease as the number of iterations increases if the chain converges to the desired target distribution.

The SSDs are first calculated for a connected Bayesian network with four nodes at 5000, 10 000, 20 000, 50 000, 75 000, 100 000, 250 000 and 500 000 iterations, and for a connected Bayesian network with five nodes at 50 000, 100 000, 250 000, 500 000, 1 000 000, 5 000 000, 10 000 000, 25 000 000, and 50 000 000 iterations.

Figure 5.11 and Figure 5.12 separately plot each SSD value at each number of iterations. Figure 5.11 and Figure 5.12 show that the NS, HAR and MH rapidly diminish the values of SSDs as the number of iterations increases.

It is also noted that the lowest and second lowest SSDs values always result from the frequencies sampled by the HAR and NS, respectively, as shown in Figure 5.13 and Figure 5.14.

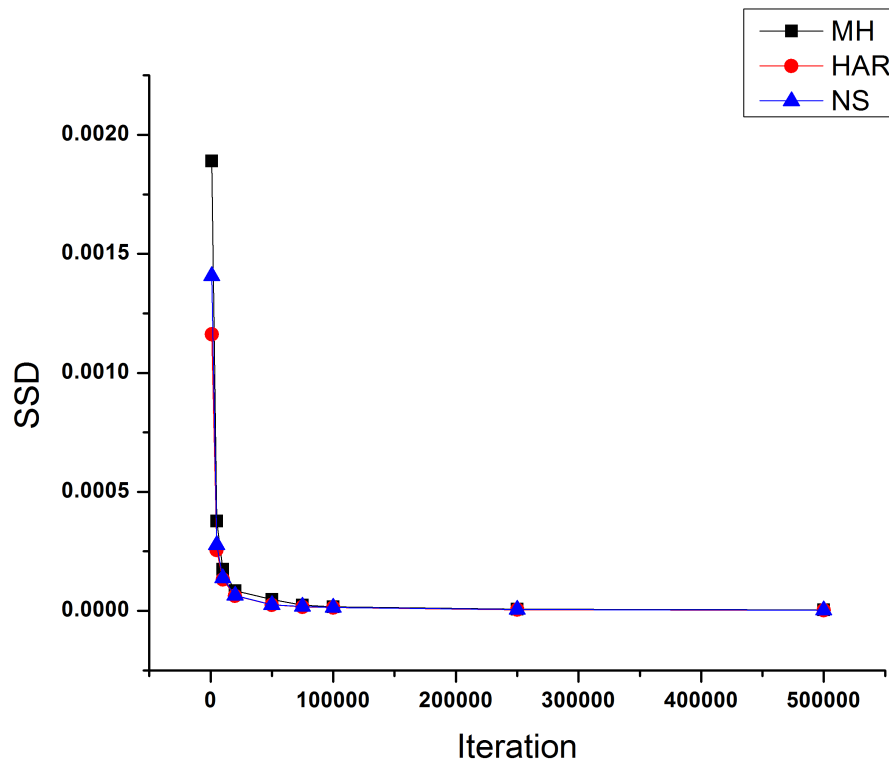


FIGURE 5.11: SSD values vs iterations with a space of four nodes.

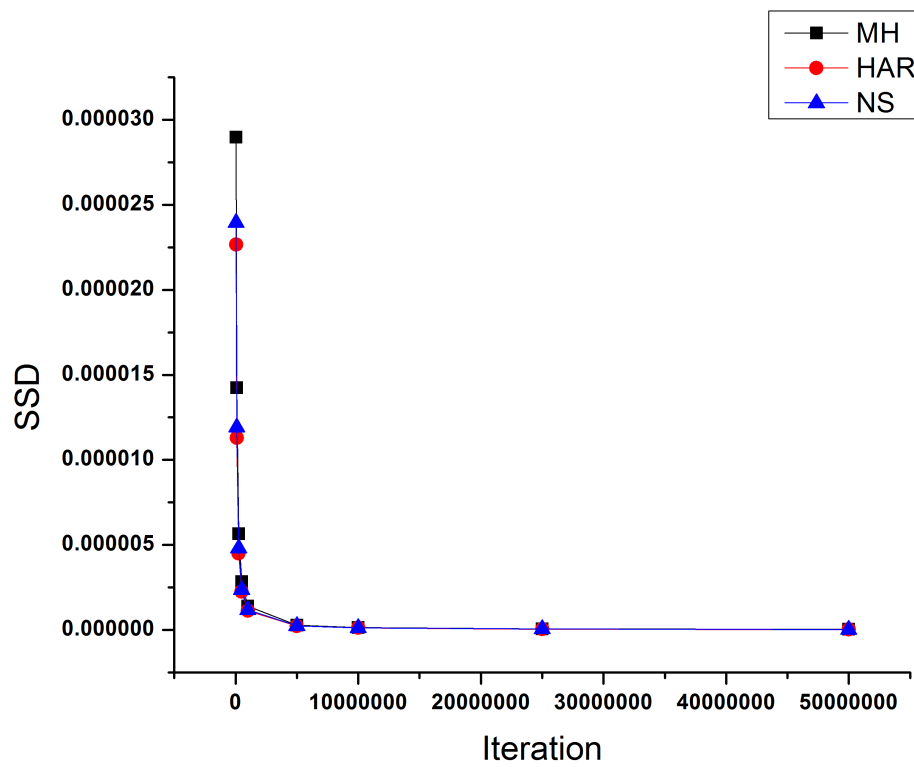


FIGURE 5.12: SSD values vs iterations with a space of five nodes.



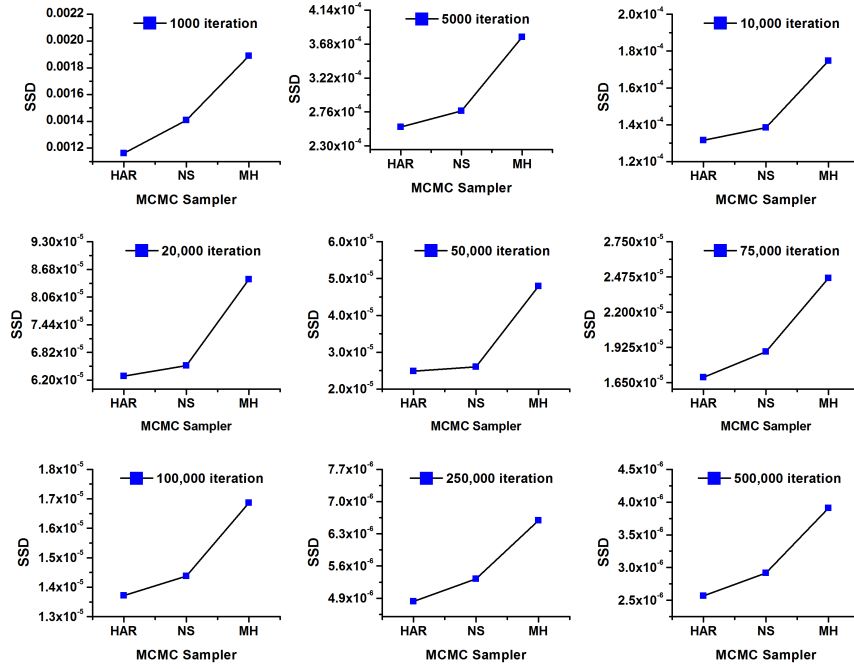


FIGURE 5.13: SSD values vs MCMC sampler with a space of four nodes.

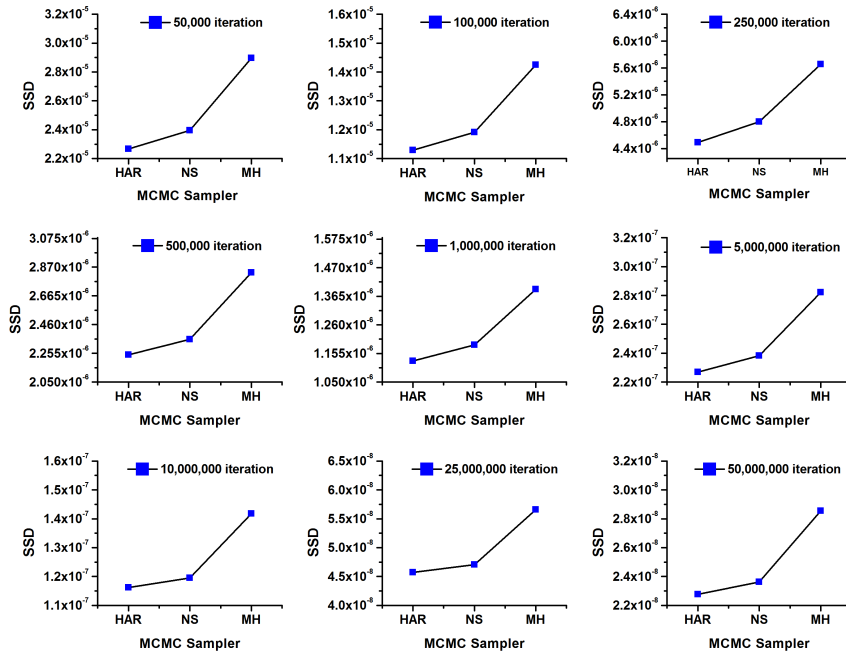


FIGURE 5.14: SSD values vs MCMC sampler with a space of five nodes.

## 5.4 Conclusion

The experimental results in this chapter have demonstrated the ability of the MH, NS and HAR to generate BNs uniformly at random. That is, a random sequence of BNs with a large number of iterations would ultimately follow a uniform pattern, in which the probability of sampling a specific Bayesian network is the proportion of those BNs in the target search space. Note that sampling randomly using the MH, HAR and NS over graph spaces does not aim to entirely explore these spaces, but rather to guarantee a uniform distribution.

## Chapter 6

# Adaptive Algorithms for Faster Adjacent Graphs Enumeration and Function Scoring

### 6.1 Introduction

This chapter proposes two adaptive techniques to solve two time-consuming problems that arise while learning the structure of a Bayesian network. The first adaptive technique aims to quickly enumerate the set of adjacent graphs for a particular graph. Unlike the commonly used brute-force approach reviewed in Section 4.3, the new adaptive technique does not need to check every single pair of nodes in the graph  $G$ . It rather determines the set of adjacent graphs of the next generated graph  $G'$  by updating the set of adjacent graphs of the current graph  $G$ . That is, it finds  $\mathcal{N}(G')$  by updating  $\mathcal{N}(G)$ , where  $G' \sim \mathcal{N}(G)$ . This adaptive technique is explained in Section 6.2. The second adaptive technique aims to quickly populate the CPTs for a given DAG. It is  $O(N)$  times faster than the other widely used approach investigated in Section 6.3.3. It updates the CPTs only for the nodes whose parent nodes have been changed after a single transition from a current graph  $G$  to one of its adjacent graphs is applied. This adaptive technique is explained in Section 6.3.

## 6.2 Adaptive technique for faster enumeration of adjacent graphs

Enumerating a set of adjacent graphs becomes a time-consuming problem when the number of nodes increases. If the number of nodes is high, the size of  $\mathcal{N}(G)$  tends to be high regardless of whether a given graph is dense or sparse. For instance, if the graph is dense, the number of deletable edges is likely to be much larger than the number of addable edges, and *vice versa* if the graph is sparse.

Sampling over sets of the form  $\mathcal{N}(G)$  is a technique used by many structural learning samplers such as Hill-Climbing Search [134, 139], Tabu Search [136–138], and Metropolis-Hastings sampler [121] to incrementally traverse search spaces of graphs. Suppose a graph  $G(V, E)$  has  $n$  vertices. I use an asymmetric  $n \times n$  adjacency matrix to represent  $G$ , as shown in Equation 6.1, with entries that are Boolean values defined in Equation 6.2.

$$G_{ij} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & \cdots & p_{1n} \\ p_{21} & p_{22} & p_{23} & \cdots & p_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & p_{n3} & \cdots & p_{nn} \end{bmatrix} \quad (6.1)$$

$$d_{ij} = \begin{cases} 1 & \text{if there is a directed edge from } i \text{ to } j, \\ 0 & \text{Otherwise.} \end{cases} \quad (6.2)$$

In order to sample a candidate DAG  $G' \sim \mathcal{N}(G)$ , one first needs to identify all the adjacent graphs to  $G$ . To do so, one can check for each pair  $(i, j) \in G$  whether it is addable, deletable, or reversible. Identifying  $\mathcal{N}(G)$  is a time-consuming problem for three reasons. First, as the number of nodes increases, the number of pairs of nodes that need to be checked also increases. Second, the computational cost is

multiplied when a sampler updates  $\mathcal{N}$  multiple times in each sampling iteration. The NS, for example, considers sampling a mediator graph  $G' \sim \mathcal{N}(G)$  between the current graph  $G$  and candidate graph  $G'' \sim \mathcal{N}(G')$ . This requires checking all pairs of nodes in  $\mathcal{N}(G)$ ,  $\mathcal{N}(G')$  and  $\mathcal{N}(G'')$  in every single sampling iteration. Third, some algorithms perform a local search within each sampling iteration. For example, the reduction step involved in the NS applies a local search to satisfy the acceptance ratio.

To define  $\mathcal{N}(G)$  in a single iteration, a graph algorithm (e.g. Depth First Search) is used to check the potential for every single pair of nodes in the graph to provide an addable, deletable, or reversible edge such that the graph remains connected and acyclic. There are  $n \times (n - 1)$  pairs in each graph to be checked. Then, one can sample  $G' \sim \mathcal{N}(G)$ , and again check all pairs in  $G'$  to define the new  $\mathcal{N}(G')$ .

I propose a new adaptive technique to quickly define the next  $\mathcal{N}(G')$ . It identifies the set  $\mathcal{N}(G')$  using computations already performed for  $\mathcal{N}(G)$ , with no need to check *all*  $n \times (n - 1)$  pairs of nodes in  $G'$ . This technique primarily focuses on skipping some redundant or unnecessary checking of pairs during the execution of addable, deletable, and reversible graph searching algorithms.

### 6.2.1 Notations, definitions and propositions

Note that all notations in this Section are new to the thesis and not easily found in the literature. I use  $(i, j, a)$ ,  $(i, j, d)$  and  $(i, j, r)$  to indicate an addable edge, deletable edge, and reversible edge, respectively, where  $i$  and  $j$  are nodes. I also define  $(i, j, x)$ , where  $x$  is either  $a$ ,  $d$  or  $r$ . Let  $A_a$ ,  $D_d$  and  $R_r$  be the sets of all addable edges, deletable edges and reversible edges, respectively. Let  $\mathcal{E}(G)$  be the set of all addable, deletable, and reversible edges in graph  $G$ . That is,  $\mathcal{E}(G)$  contains all  $(i, j, x)$ , where  $(i, j, a)$  is included iff  $(i, j) \in A_a$ ,  $(i, j, d)$  is included iff  $(i, j) \in D_d$ , and  $(i, j, r)$  is included iff  $(i, j) \in R_r$ .

Let  $\mathcal{G}_a$ ,  $\mathcal{G}_d$  and  $\mathcal{G}_r$  be the sets of all adjacent graphs that are constructed based on  $A_a$ ,  $D_d$  and  $R_r$ , respectively. I also define  $\mathcal{N}(G) = \mathcal{G}_a \cup \mathcal{G}_d \cup \mathcal{G}_r \cup \{G\}$  to refer to the set of all possible adjacent graphs that differ from  $G$  by only one edge plus the graph  $G$  itself, while the members of  $\mathcal{N}(G)$  remain connected and acyclic. Let also  $A_n$ ,  $D_n$  and  $R_n$  be the sets of all non-addable, non-deletable and non-reversible edges, respectively, that is edges for which performing the specified action would violate the conditions of connectivity or acyclicity. Let  $E(G)$  be the set of all edges in  $G$ .

Typically, to find  $\mathcal{N}(G)$  based on  $\mathcal{E}(G)$ , all pairs of nodes in  $G$  must be checked. For each  $(i, j) \notin E(G)$ , if a pair of nodes  $(i, j)$  is addable, then add it to  $A_a$ , if not add it to  $A_n$ . Again, deletable and reversible edges are identified by checking every  $(i, j) \in E(G)$ . If an edge is deletable or reversible, add it to  $D_d$  or  $R_r$ , respectively, if not add it to  $D_n$  or  $R_n$ , respectively. I also define  $A = A_a \cup A_n$ ,  $D = D_d \cup D_n$  and  $R = R_r \cup R_n$ . The set  $A$  contains the ordered pairs of nodes that do not belong to  $E(G)$ . The sets  $D$  and  $R$  are both equal to  $E(G)$ . For a sparse DAG, the size of  $A$  will be large, but the sizes of  $D$  and  $R$  will be small. Similarly, for a dense DAG,  $|A|$  will be small, but  $|D|$  and  $|R|$  will be large. The comprehensive set of all pairs of nodes in a DAG  $G$  is defined as  $\Omega(G) = A \cup D \cup R$ . The goal now is to adaptively identify  $\mathcal{N}(G')$  given  $\mathcal{N}(G)$ .

The adaptive technique uses the standard checking only once to populate the initial lists  $(A, D, R)$ . When a neighboring graph is selected from  $\mathcal{N}(G)$ , determine from which list the updated edge was selected:  $A_a$ ,  $D_d$  or  $R_r$ . Then, update  $A_a$ ,  $D_d$  and  $R_r$  given the last change made on  $G$ . That is, when any list is updated (e.g.  $D_d$ ), check whether any edge which was previously deletable has become non-deletable, in which case that edge will be removed from  $D_d$ , and added to  $D_n$ . This process is performed for each list. I use  $T_a[i, j]$ ,  $T_d[i, j]$  and  $T_r[i, j]$  to refer to permitted transformations between two adjacent graphs, either via adding an edge, deleting

an edge or reversing an edge, respectively. For example,  $T_a[i, j] : G \mapsto G'$  represents the graph transformation from  $G$  to  $G'$  where  $G' \sim \mathcal{N}(G)$  using an addable edge  $(i, j)$ .

**Proposition 7.**  $G' \sim \mathcal{N}(G)$  and  $T_a[i, j] : G \mapsto G' \iff G \sim \mathcal{N}(G')$  and  $T_d[i, j] : G' \mapsto G$

**Proof:**

Since both  $G$  and  $G'$  are connected and acyclic,  $(i, j)$  can be added to  $G$  or deleted from  $G'$  without creating cycles or disconnecting the graph. ■

**Proposition 8.** If  $G' \sim \mathcal{N}(G)$  and  $T_a[i, j] : G \mapsto G'$ , then,  $(i, j) \notin A(G')$ .

**Proof:**

Since  $(i, j) \in E(G') \implies (i, j) \notin A_a(G')$  and  $(i, j) \notin A_n(G')$  ■

**Proposition 9.** If  $G' \sim \mathcal{N}(G)$  and  $T_d[i, j] : G \mapsto G'$ , then,  $(i, j) \notin D(G')$ .

**Proof:**

Since  $(i, j) \notin E(G') \implies (i, j) \notin D_d(G')$  and  $(i, j) \notin D_n(G')$ . ■

**Proposition 10.** If  $G' \sim \mathcal{N}(G)$  and  $T_r[i, j] : G \mapsto G'$ . Then:

- |                            |                          |
|----------------------------|--------------------------|
| i) $(i, j) \notin R_r(G')$ | ii) $(j, i) \in R_r(G')$ |
| iii) $(i, j) \notin D(G')$ | iv) $(j, i) \in D(G')$   |

**Proof:**

Since  $T_r[i, j] : G \mapsto G'$ ,

we have  $(i, j) \in E(G) \implies (j, i) \in E(G') \implies (i, j) \notin E(G')$

$\therefore (i, j) \notin R_r(G') \wedge D(G')$  and  $T_r[j, i] : G' \mapsto G$ . ■

### 6.2.2 Algorithm and illustrative example

The new adaptive approach starts with Algorithm 8. It initialises six lists:  $A_a$ ,  $D_d$ ,  $R_r$ ,  $A_n$ ,  $D_n$ , and  $R_n$ . In this initial step, I use the standard algorithms in Section 4.3 to check connectivity and cyclicity for every single pair of nodes in the graph.

---

**Algorithm 8** Populate all lists in  $\Omega(G)$ 

---

Input:  $G(V, E)$ Output:  $A$ ,  $D$  and  $R$  listsPhase 1: Populate  $A$  list.

```

1: for all  $(i,j) \notin E$  do
2:   if  $(i,j)$  is Addable then
3:      $A_a.add((i,j), \text{"addable"})$ 
4:   else
5:      $A_n.add((i,j), \text{"non-addable"})$ 
6:   end if
7: end for

```

Phase 2: Populate  $D$  list.

```

8: for all  $(i,j) \in E$  do
9:   if  $(i,j)$  is Deletable then
10:     $D_d.add((i,j), \text{"deletable"})$ 
11:   else
12:     $D_n.add((i,j), \text{"non-deletable"})$ 
13:   end if
14: end for

```

Phase 3: Populate  $R$  list.

```

15: for all  $(i,j) \in E$  do
16:   if  $(i,j)$  is reversible then
17:     $R_r.add((i,j), \text{"reversible"})$ 
18:   else
19:     $R_n.add((i,j), \text{"non-reversible"})$ 
20:   end if
21: end for

```

---



Note, each of the  $A_a$ ,  $D_d$ ,  $R_r$ ,  $A_n$ ,  $D_n$ , and  $R_n$  were managed with dictionary data structures, where each item is a (key, value) pair, for example, key: the pair  $(i, j)$  and value: addable or non-addable. The following lines explain the core idea of the proposed adaptive technique for adjacent graphs enumeration. At the very first iteration of the whole sampling algorithm, I apply Algorithm 8 in order to populate all the lists  $A$ ,  $D$  and  $R$ . Now, for every DAG sampled during one iteration, I update all the lists according to Algorithm 9. These upgrades of  $A$ ,  $D$  and  $R$  lists in  $\Omega(G)$  only observe the element pairs in the previous versions of these lists, instead of checking all  $n(n - 1)$  pairs. Thus, this adaptive technique provides faster neighbourhood construction. Table 6.1 briefly describes the effect on  $A$ ,  $D$  and  $R$  lists after a single transition between two adjacent graphs is conducted.

Operation	Effect on $A$ list	Effect on $D$ list	Effect on $R$ list
<b>Add an edge</b>	addable may	non-deletable may	reversible may
	become non-addable	become deletable	become non-reversible
<b>Delete an edge</b>	non-addable may	deletable may	non-reversible may
	become addable	become non-deletable	become reversible
<b>Reverse an edge</b>	addable may	No effect	reversible may
	become non-addable and <i>vice versa</i>		become non-reversible and <i>vice versa</i>

TABLE 6.1: Effects of adding, deleting and reversing an edge on  $A$ ,  $D$  and  $R$  lists

Given Algorithm 8 and Table 6.1, Algorithm 9 is used to adaptively update  $\mathcal{N}$ .

**Algorithm 9** Adaptive process for updating  $\Omega(G)$  during  $\mathcal{N}$  construction

---

- 1: Initialise  $\mathcal{E}(G)$ .
  - 2: Randomly sample  $(i, j, x) \in \mathcal{E}(G)$ .
  - 3: **if**  $x = a$  **then**
    - 4: Update A list: (i) Remove  $(i, j)$  from  $A_a$  (ii) Remove  $(j, i)$  from  $A$  (iii) Check edges in  $A_a$  to see if they become non-addable.
    - 5: Update D list: (i) Add  $(i, j)$  to  $D_d$  (ii) Check edges in  $D_n$  to see if they become deletable.
    - 6: Update R list: (i) If  $(i, j)$  and  $(j, i)$  are both addable then the added edge is in  $R_r$ , otherwise it's in  $R_n$  (ii) Check edges in  $R_r$  to see if they become non-reversible.
  - 7: **else if**  $x = d$  **then**
    - 8: Update A list: (i) Add  $(i, j)$  to  $A_a$  (ii) If  $(i, j)$  was reversible, then both  $(i, j)$  and  $(j, i)$  become addable. But if  $(i, j)$  was not reversible,  $(j, i)$  must be placed in  $R_n$  (iii) Check edges in  $A_n$  to see if they become addable.
    - 9: Update D list: (i) Remove  $(i, j)$  from  $D_d$  (ii) Check edges in  $D_d$  to see if they become non-deletable.
    - 10: Update R list: (i) Remove  $(i, j)$  from  $R$  (ii) Check edges in  $R_n$  to see if they become reversible.
  - 11: **else if**  $x = r$  **then**
    - 12: Update A list: (i) Check all edges in  $A = A_a \cup A_n$  to see whether they are addable or not.
    - 13: Update D list: (i) Remove  $(i, j)$  from  $D$  (ii) Add  $(j, i)$  to  $D_d$  if  $(i, j)$  was in  $D_d$ ; else add it to  $D_n$ .
    - 14: Update R list: (i) Remove  $(i, j)$  from  $R_r$  (ii) Add  $(j, i)$  to  $R_r$  (iii) Check all edges in  $R = R_r \cup R_n$  to see whether they are reversible or not.
  - 15: **end if**
  - 16: Update  $\mathcal{E}(G)$ .
  - 17: Find  $\mathcal{N}(G) = (\mathcal{G}_a, \mathcal{G}_d, \mathcal{G}_r)$ .
  - 18: **goto** 2
-

**Example 5.** A Bayesian network with four nodes provides a simple test case to illustrate Algorithm 8 and Algorithm 9. Consider the initial DAG  $G$  in Figure 6.1. The graph is connected and acyclic. I first apply Algorithm 8 in order to populate  $A$ ,  $D$  and  $R$ . The result is shown in Table 6.2. Cells in black colour or gray colour refer to addable, deletable and reversible pairs, or non-addable, non-deletable and non-reversible pairs, respectively. To find  $A = A_a \cup A_n$ , I only check every pair of nodes  $(i, j) \in \overline{E(G)} = \{(A, B), (B, A), (B, C), (C, B)\}$ , where  $\overline{E(G)}$  is the complementary list of  $E(G)$ . Note that the edges  $\{(C, A), (D, A), (B, D), (C, D)\}$  are not considered because they cannot be added, deleted, or reversed. In contrast, to find  $D$  or  $R$ , I only check every edge  $(i, j) \in E(G) = \{(A, C), (A, D), (D, B), (D, C)\}$ .

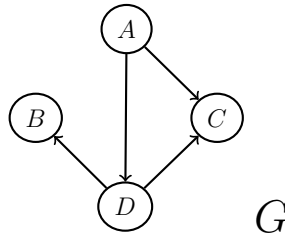


FIGURE 6.1: Initial graph

List	Elements			
$A$	(A, B)	(B, A)	(B, C)	(C, B)
$D$	(A, C)	(A, D)	(D, C)	(D, B)
$R$	(A, C)	(A, D)	(D, C)	(D, B)

TABLE 6.2: Populate  $A$ ,  $D$  and  $R$  lists. Cells in black colour or gray colour refer to addable, deletable and reversible pairs, or non-addable, non-deletable and non-reversible pairs, respectively

In order to transit from  $G$  in Figure 6.1 to one of its adjacent DAGs, one needs first to find  $\mathcal{E}(G)$  and then  $\mathcal{N}(G)$ . Table 6.2 shows permissible transitions and populates  $A$ ,  $D$  and  $R$  lists. Note that some edges may appear in more than one list e.g.  $DC$  is in  $D$  list and  $R$  list. Therefore, I use  $(D, C, d)$  and  $(D, C, r)$

to indicate that  $(D, C) \in D_d$  and  $(D, C) \in R_r$ , respectively. Hence  $\mathcal{E}(G) = \{(A, B, a), (B, C, a), (C, B, a), (A, C, d), (A, D, d), (D, C, d), (A, D, r), (D, C, r), (D, B, r)\}$ .

Suppose one randomly sampled an element from  $\mathcal{E}(G)$  e.g.  $(A, B, a)$ . One would then conduct the steps from 3 to 6 in Algorithm 9, since  $(A, B) \in A_a$ . Table 6.2 is now updated to Table 6.3. Note that the list  $A$  was updated by removing  $(A, B)$  from the addable list  $A_a$ , removing  $(B, A)$  from the  $A$  list, and then only the edges in  $A_a$  are checked to determine whether they become non-addable, as cyclicity may occur after adding  $(A, B)$ . I update  $D$  by adding  $(A, B)$  to the  $D_d$  list and again checking all non-deletable edges in  $D_n$ . Lastly, the list  $R$  was updated by adding  $(A, B)$  to  $R_r$  and then checking all edges in  $R_r$ .

List	Elements				
$A$	(B, C) (C, B)				
$D$	(A, C)	(A, D)	(D, C)	(D, B)	(A, B)
$R$	(A, C)	(A, D)	(D, C)	(D, B)	

TABLE 6.3: Update  $A$ ,  $D$  and  $R$  lists after adding  $(A, B, a) \in \mathcal{E}(G)$ .

I also illustrate steps 7 to 10 in Algorithm 9 by sampling a deletable edge from the list  $\mathcal{E}(G)$  e.g.  $(A, C, d)$ . Table 6.2 is now updated to Table 6.4.

List	Elements					
$A$	(A, B)	(B, A)	(B, C)	(C, B)	(A, C)	(C, A)
$D$	(A, D) (D, C) (D, B)					
$R$	(A, D) (D, C) (D, B)					

TABLE 6.4: Update  $A$ ,  $D$  and  $R$  lists after deleting  $(A, C, d) \in \mathcal{E}(G)$ .

Finally, I illustrate steps 11 to 14 in Algorithm 9 by sampling a reversible pair from the list  $\mathcal{E}_G$  e.g.  $(A, D, r)$ . Table 6.2 is now updated to Table 6.5.

List	Elements			
$A$	(A, B)	(B, A)	(B, C)	(C, B)
$D$	(A, C)		(D, C)	(D, B) (D, A)
$R$	(A, C)		(D, C)	(D, B) (D, A)

TABLE 6.5: Update  $A$ ,  $D$  and  $R$  lists after reversing  $(A, D, r) \in \mathcal{E}(G)$ .

The difference in time complexity between the new adaptive technique and standard brute-force to enumerate adjacent graphs is compared in the following section.

### 6.2.3 Standard brute-force vs adaptive algorithm

In order to enumerate graphs in the neighbourhood of the current graph, there are three main sequential operations required to construct the lists of all addable edges  $A$ , deletable edges  $D$  and reversible edges  $R$ . Figure 6.2 shows these three sequential operations used both in the standard brute-force and adaptive algorithms. The "Find" functions in the standard algorithm are used to calculate by brute force the lists of all addable, deletable and reversible edges, whereas the "Update" functions in the adaptive algorithm are used to dynamically modify all the lists that are already defined in the previous iteration.

Standard algorithm	Adaptive algorithm
Find Neighbors (X) { Sample $Y \in \mathcal{N}(X)$ ; <b>Find</b> A list (Y); <b>Find</b> D list (Y); <b>Find</b> R list (Y); }	Find Neighbors (X) { Sample $Y \in \mathcal{N}(X)$ ; <b>Update</b> A list (Y); <b>Update</b> D list (Y); <b>Update</b> R list (Y); }

FIGURE 6.2: The three sequential operations used in the two algorithms.

### 6.2.3.1 Brute-force algorithm: complexity analysis

Theorem 6.2.1 proves the running-time complexity for the brute-force algorithm.

**Theorem 6.2.1.** The running-time complexity for the standard brute-force algorithm is  $O(V^4)$ .

**Proof:**

In the brute-force algorithm to evaluate the  $A$  list, one checks the cyclicity of the graph for each of the  $(i, j)$  pairs of nodes using the BFS algorithm which has worst-case complexity  $O(V^2)$  ([155]). Since there are  $\Theta(V^2)$  pairs in the graph ( $V$  = the number of nodes in the graph) and for each of them cyclicity needs to be checked for the graph created by their possible inclusion, there are  $O(V^4)$  operations. Also it is necessary to check the number of parents and number of children for each node. These two checks require a linear running-time complexity  $O(V)$  each. Thus the time complexity for the  $A$  list is  $O(V^4)$ .

Evaluating the  $R$  list also consumes the same running-time ( $O(V^4)$ ) as the  $A$  list, because it checks for cycles for each pair of nodes  $(i, j)$ , and the numbers of parents and children for each node.

To evaluate the  $D$  list, a modified version of DFS algorithm (Section 4.3) can be used as a bridge detection algorithm which has worst case time-complexity  $O(V^2)$ . Therefore, the set of bridges  $B$  can be found in  $O(V^2)$ . Next, the set of deletable edges (non-bridges) can be found by evaluating the set-difference between  $E$  and  $B$ , which can be computed in  $O(|E| + |B|) \approx O(V^2)$ .

For the standard brute-force algorithm, the running-time complexity is therefore  $O(V^4)$ . ■

Note that, all node pairs  $(i, j)$  have to be checked in the brute-force approach. However, in the adaptive approach, it is not necessary to check all node pairs. Each of the neighbours of the current graph are generated by changing a single  $(i, j)$  pair,

either by adding, deleting, or reversing, leaving the remaining  $(i, j)$  pairs the same as the original (current) graph.

### 6.2.3.2 Adaptive algorithm: worst-case complexity

The asymptotic behaviour of the three update functions in Figure 6.2 depends on the sizes of the  $A$ ,  $D$  and  $R$  lists, respectively. Note that the number of edges in  $A$  list to be checked by a sampler using the adaptive method is at most  $\frac{n(n-1)}{2} - (n-1)$ . The maximum numbers of edges in the  $D$  and  $R$  lists to be checked by a sampler using the adaptive method are at most  $\frac{n(n-1)}{2}$ . These maximum numbers are less than  $n(n-1)$  which is the maximum number of edges in  $A$  list,  $D$  list, and  $R$  list to be checked by a sampler using the brute-force method. However, Corollary 1 shows that the number of edges that are either addable or deletable scales quadratically. This indicates that the worst-case complexity of the adaptive approach may have similar asymptotic behavior:  $O(V^4)$ , as the standard brute-force approach.

Theorem 6.2.2 and Corollary 1 prove some thresholds on the number of edges for a DAG.

**Theorem 6.2.2.** For any DAG  $G$ , there are at most  $\frac{n(n-1)}{2}$  non-addable edges.

**Proof:**

If one adds an edge to  $G$ , the number of non-addable edges does not decrease, because any edge that would have created a cycle still would. Keep adding edges until obtaining a graph  $H$  with no addable edges. Then  $H$  will have exactly  $\frac{n(n-1)}{2}$  edges. To see this, suppose there is some pair of nodes  $i$  and  $j$  such that adding  $(i, j)$  would create a cycle, and adding  $(j, i)$  would also create a cycle. Then there must be an alternative path from  $i$  to  $j$ , and also an alternative path from  $j$  to  $i$ . But concatenating these two paths would create a cycle in  $H$ , a contradiction. Hence, for every pair of nodes  $i$  and  $j$ ,  $H$  must contain either  $(i, j)$  or  $(j, i)$  but not both, as that would create a cycle. There are exactly  $\frac{n(n-1)}{2}$  node pairs, hence  $\frac{n(n-1)}{2}$  edges

in  $H$ , and  $\frac{n(n-1)}{2}$  non-addable edges for  $H$ , which is an upper bound for the number of non-addable edges for  $G$ . ■

**Corollary 1.** Any DAG  $G$  has at least  $\frac{n(n-1)}{2} - (n-1)$  edges that are either addable or deletable.

**Proof:**

Of the  $n(n-1)$  possible directed edges, at most  $\frac{n(n-1)}{2}$  are non-addable, and at most  $n-1$  are non-deletable. The rest are either addable or deletable. ■

**Remark 6.** The average-case performance of adaptive approach still achieves a useful speed up in practice compared to the BFS and modified DFS discussed in Section 4.3.

### 6.2.3.3 Simulation study: speed-up achieved in practice

I considered the BFS algorithm and modified DFS algorithm presented in Section 4.3 as a non-adaptive approach to enumerate adjacent graphs. Table 6.6 compares the non-adaptive approach versus the new adaptive approach in terms of time required to enumerate adjacent graphs. The simulation compared the two methods for different numbers of nodes  $n$  including 4, 11, 37, 51, and 100 nodes. The total number of iterations  $t$  for each  $n$  is shown in the first column in Table 6.6. Note that the number of iterations was decreased as the number of nodes increased to avoid high time consumption. The two methods started with the same initial network.

For each simulation setting of  $n$  and  $t$ , each method was run for ten trials. At the end of each trial, the time of simulation was recorded in *seconds*. Then, the average and standard deviation of the observed times were calculated in last column in Table 6.6.



Simulation	Trial Method	1	2	3	4	5	6	7	8	9	10	<i>Mean ± SD</i>
$n = 4$ $t = 500,000$	Brute-force	77	79	78	78	77	78	85	77	77	84	$79 \pm 2.828$
	Adaptive	48	48	47	48	48	48	49	48	47	48	$47.9 \pm 0.538$
$n = 11$ $t = 50,000$	Brute-force	232	226	227	228	229	230	230	227	229	227	$228.5 \pm 1.746$
	Adaptive	197	199	199	197	196	197	197	198	197	198	$197.5 \pm 0.921$
$n = 37$ $t = 10,000$	Brute-force	902	914	903	903	909	907	913	908	910	908	$907.7 \pm 3.9$
	Adaptive	278	266	273	272	271	277	275	273	279	275	$273.9 \pm 3.618$
$n = 51$ $t = 5,000$	Brute-force	1490	1519	1503	1487	1483	1498	1512	1523	1490	1495	$1500 \pm 13.152$
	Adaptive	508	515	507	518	523	509	507	502	509	518	$511.6 \pm 6.2$
$n = 100$ $t = 1,000$	Brute-force	1389	1386	1408	1427	1405	1416	1403	1419	1392	1407	$1405.2 \pm 12.663$
	Adaptive	899	897	899	898	909	907	906	899	904	896	$901.4 \pm 4.409$

TABLE 6.6: Comparing speed in seconds: non-adaptive approach vs adaptive approach to enumerate adjacent graphs for iterated connected BNs.

The empirical results in Table 6.6 demonstrate the performance capabilities of the adaptive approach to enumerate adjacent graphs faster than the non-adaptive approach.

### 6.3 Adaptive function scoring to compute Bayesian network parameters

Parameter learning in a discrete Bayesian network is another time-consuming problem. There are three main factors which determine the number of conditional probability distributions required to populate CPTs. First, the CPT of a particular node grows in proportion to the number of state values associated with each parent of node. Second, the more variables and directed edges a Bayesian network includes, the more conditional probabilities there are to calculate. Third, every parameter in a CPT becomes computationally expensive to estimate when the number of observations in a dataset is large. The following example illustrates how the time

complexity depends on these three factors. Figure 6.3 illustrates how the size of a CPT for a particular node grows as the number of parents and state values increase.

Node		
Parent 1	T	F
T	P(TT)	P(TF)
F	P(FT)	P(FF)

Node			
Parent 1	Parent 2	T	F
T	T	P(TTT)	P(TTF)
T	F	P(TFT)	P(TFF)
F	T	P(FTT)	P(FTF)
F	F	P(FFT)	P(FFF)

Node				
Parent 1	Parent 2	H	M	L
H	H	P(HHH)	P(HHM)	P(HHL)
H	M	P(HMH)	P(HMM)	P(HML)
H	L	P(HLH)	P(HLM)	P(HLL)
M	H	P(MHH)	P(MHM)	P(MHL)
M	M	P(MMH)	P(MMM)	P(MML)
M	L	P(MLH)	P(MLM)	P(MLL)
L	H	P(LHH)	P(LHM)	P(LHL)
L	M	P(LMH)	P(LMM)	P(LML)
L	L	P(LLH)	P(LLM)	P(LLL)

Node					
Parent 1	Parent 2	Parent 3	H	M	L
H	H	H	P(HHHH)	P(HHHM)	P(HHHL)
H	H	M	P(HHMH)	P(HHMM)	P(HHML)
H	H	L	P(HHLH)	P(HHLM)	P(HHLL)
H	M	H	P(HMHH)	P(HMHM)	P(HMHL)
H	M	M	P(HMMH)	P(HMMM)	P(HMML)
H	M	L	P(HMLH)	P(HMLM)	P(HMLL)
H	L	H	P(HLHH)	P(HLHM)	P(HLHL)
H	L	M	P(HLMH)	P(HLMM)	P(HLML)
H	L	L	P(HLLH)	P(HLLM)	P(HLLL)
M	H	H	P(MHHH)	P(MHHM)	P(MHHL)
M	H	M	P(MHMH)	P(MHMM)	P(MHML)
M	H	L	P(MHLH)	P(MHLM)	P(MHLL)
M	M	H	P(MMHH)	P(MMHM)	P(MMHL)
M	M	M	P(MMMH)	P(MMMM)	P(MMML)
M	M	L	P(MMLH)	P(MMLM)	P(MMLL)
M	L	H	P(MLHH)	P(MLHM)	P(MLHL)
M	L	M	P(MLMH)	P(MLMM)	P(MLML)
M	L	L	P(MLLH)	P(MLLM)	P(MLLL)
L	H	H	P(LHHH)	P(LHHM)	P(LHHL)
L	H	M	P(LHMH)	P(LHMM)	P(LHML)
L	H	L	P(LHLH)	P(LHLM)	P(LHLL)
L	M	H	P(LMHH)	P(LMHM)	P(LMHL)
L	M	M	P(LMMH)	P(LMMM)	P(LMML)
L	M	L	P(LMLH)	P(LMLM)	P(LMLL)
L	L	H	P(LLHH)	P(LLHM)	P(LLHL)
L	L	M	P(LLMH)	P(LLMM)	P(LLML)
L	L	L	P(LLLH)	P(LLLM)	P(LLLL)

FIGURE 6.3: From left to right, top to bottom: a node with one parent and two state values each, a node with two parents and two state values each, a node with two parents and three state values each, and a node with three parents and three state values each.

### 6.3.1 Conditional probabilities in a graph

Let  $\xi(v) \equiv |\text{CPT}(v)|$  be the number of conditional probabilities in a CPT of a particular node  $v \in V(G)$  in a Bayesian network  $G$ . Let  $\xi(G) \equiv |\text{CPT}(G)|$  be the total number of conditional probabilities in  $G$ , so that:

$$\xi(G) = \sum_{i=1}^n \xi(v_i). \quad (6.3)$$

**Lemma 6.3.1.** Let  $r(v)$  be the number of possible state values of node  $v \in V(G)$ . Let  $\delta_j(v)$  be the number of possible state values of the  $j^{\text{th}}$  parent of node  $v$ . The number of conditional probabilities  $\xi(v)$  for a single node  $v$  that has  $\omega$  parents is expressed in Equation 6.4.

$$\xi(v) = r(v) \prod_{j=1}^{\omega} \delta_j(v). \quad (6.4)$$

■

**Example 6.** Suppose  $v \in G$  is a node with two state values ( $r = 2$ ). Suppose also the node  $v$  has two parents ( $\omega = 2$ ). The first and second parents of node  $v$  have three and four state values, respectively ( $\delta_1(v) = 3$  and  $\delta_2(v) = 4$ ). Using Lemma 6.3.1,  $\xi(v) = 2 \times 3 \times 4 = 24$ . ■

**Lemma 6.3.2.** Consider  $\{v_1, v_2, \dots, v_n\} = V(G)$ . Using Lemma 6.3.1, the number of conditional probabilities  $\xi(G)$  for an entire graph  $G$  is expressed in Equation 6.5.

$$\xi(G) = \sum_{i=1}^n \left[ r(v_i) \prod_{j=1}^{\omega} \delta_j(v_i) \right]. \quad (6.5)$$

■

Note that Lemma 6.3.1 and Lemma 6.3.2 are used in Section 6.3.3 to provide a Big O expression for the new adaptive function scoring algorithm.

### 6.3.2 Exploring dataset

This section calculates the total number of cell explorations in the data matrix for the CPT of a single node and single graph per iteration. Consider a data matrix  $n \times m$ , where  $n$  represents the number of nodes (rows) and  $m$  represents the number of observations (columns). Recall that  $N_{ijk}$  is the number of observations in state value (bin)  $k$  of node number  $i$  corresponding to a parent configuration  $j$ . Let  $T_N(v)$  be the total number of cell explorations in the data matrix for a single cell of a CPT of node  $v$ . Equation 6.6 calculates  $T_N(v)$  by only exploring the observations at nodes that are involved in estimating a CPT.

$$T_N(v) = (\omega(v) + 1)m, \quad (6.6)$$

where  $\omega(v)$  is the number of parents of node  $v$  in a CPT.

**Example 7.** Suppose a graph  $G(V, E)$  consisting of five nodes  $\{v_1, v_2, v_3, v_4, v_5\} \in V$ . Suppose for each of the five nodes there are 1000 observations taking two binary state values: True and False. To calculate a conditional probability e.g.  $P(v_1 = \text{True} | v_3 = \text{True}, v_5 = \text{True})$ , the sampler will only visit the rows that belong to the node  $v_1$  and its two parents  $v_3$  and  $v_5$  in the data matrix. Using Equation 6.6, the  $T_N(v_1)$  of  $P(v_1 = \text{True} | v_3 = \text{True}, v_5 = \text{True})$  is equal to 3000 operations. ■

To calculate the number of data matrix explorations  $T_v$  for the entire CPT of a single node  $v$ , the sampler needs to process all cells in a CPT. Equation 6.7 considers  $m$ ,  $(\omega(v) + 1)$  and  $\xi(v)$  to calculate  $T_v$ :

$$T_v = \xi(v)T_N(v), \quad (6.7)$$

where  $T_N(v)$  has the same value for all cells in the CPT of node  $v$ . In worst-case, for  $n$  nodes in a graph, the sampler needs to evaluate all CPTs in each iteration.

Consider the number of explorations  $T_G$  for an entire  $G$ . Equation 6.8 considers  $m$ ,  $(\omega(v) + 1)$ ,  $\xi(v)$ , and  $n$  to calculate  $T_G$ .

$$T_G = \sum_{i=1}^n T_{v_i} \quad (6.8)$$

**Example 8.** Suppose  $G(V, E)$  is a Bayesian network consisting of eleven nodes ( $n = 11$ ). Suppose also that an MCMC sampler is intended to learn from  $m = 1000$  observations. Note that an acyclic graph must contain at least one node with no parents. I therefore assume that a DAG containing 11 nodes, three of which have no parents, with the other 8 nodes having 3 parents each. Using Lemma 6.3.1, the size of the CPT for each of the 8 nodes is then 81, and the CPT for each of the 3 nodes is 3. Next, the sampler is required to access the observations of  $\omega(v) + 1 = 4$  nodes to estimate the CPT for each of the 8 nodes. That is,  $81 \times 4 \times 1000 = 324000$  operations to calculate the CPT for each of the 8 nodes, and  $3 \times 1 \times 1000 = 3000$  operations to calculate the CPT for each of the 3 nodes. For the entire  $G$ , the observation matrix must be accessed  $T_G = [8 \times 81 \times 4 \times 1000] + [3 \times 3 \times 1 \times 1000] = 2601000$  times to calculate the CPTs for all 11 nodes in that single network  $G$ . ■

### 6.3.3 Big-O expression

In the adaptive function scoring method, for the MH, HAR and NS, there are respectively a maximum of two,  $2\ell$  and four nodes that may change in number of parents per iteration. Therefore, the corresponding CPTs need to be updated only for these two,  $2\ell$  and four nodes. For the rest of the nodes in the graph, their CPTs will be unchanged. Note that with the HAR sampler, in order to take advantage of the new adaptive function scoring method, the value of  $2\ell$  must be less than  $n$ .

The maximum numbers of cells to access in the data matrix for a single CPT in graph  $G$  per iteration using the MH, HAR and NS with the adaptive function

scoring method are  $\sum_{i=1}^2 T_{v_i}$ ,  $\sum_{i=1}^{2\ell} T_{v_i}$  and  $\sum_{i=1}^4 T_{v_i}$ , respectively.

**Remark 7.** The indices  $i$  in  $\sum_{i=1}^2 T_{v_i}$ ,  $\sum_{i=1}^{2\ell} T_{v_i}$  or  $\sum_{i=1}^4 T_{v_i}$  cannot always be  $(i = 1, 2)$ ,  $(i = 1, \dots, 2\ell)$ , or  $(i = 1, 2, 3, 4)$  when using the MH, HAR or NS, respectively. Here, I just assume the worst-case scenario.

The ratios of times per iteration between the brute-force function scoring method and adaptive function scoring method are  $\frac{\sum_{i=1}^n T_{v_i}}{\sum_{i=1}^2 T_{v_i}}$ ,  $\frac{\sum_{i=1}^n T_{v_i}}{\sum_{i=1}^{2\ell} T_{v_i}}$  and  $\frac{\sum_{i=1}^n T_{v_i}}{\sum_{i=1}^4 T_{v_i}}$ . Therefore, the adaptive function scoring method will be approximately  $\frac{n}{2}$ ,  $\frac{n}{2\ell}$  and  $\frac{n}{4}$  times faster than the standard brute-force function scoring method. In other words, it is  $O(N)$  times faster with Big- $O(N)$ . Note that this is the improvement ratio for computing CPTs only.

The following three Lemmas follow from Remark 7.

**Lemma 6.3.3.** Using the MH sampler and adaptive function scoring method, there are at least  $\sum_{i=1}^{n-2} T_{v_i}$  conditional probabilities not calculated, where  $n > 2$ .

**Lemma 6.3.4.** Using the HAR sampler and adaptive function scoring method, there are at least  $\sum_{i=1}^{n-2\ell} T_{v_i}$  conditional probabilities not calculated, where  $n > 2\ell$ .

**Lemma 6.3.5.** Using the NS and adaptive function scoring method, there are at least  $\sum_{i=1}^{n-4} T_{v_i}$  conditional probabilities not calculated, where  $n > 4$ .

### 6.3.4 Adaptive scoring function

Let  $\Delta(G) = \{Pa(v_1), Pa(v_2), \dots, Pa(v_n)\}$  be the collection of all sets of parents for all nodes in a particular  $V(G)$ . Let  $\mathcal{F}(G) = \{P(v_1|Pa(v_1)), P(v_2|Pa(v_2)), \dots, P(v_n|Pa(v_n))\}$  be the set of all conditional probabilities for all nodes in  $V(G)$ . Recall that the probability of a given set of state variables for the nodes of  $G$  can be calculated using Equation 6.9:

$$P(v_1, v_2, \dots, v_n) = \prod_{i=1}^n P(v_i|Pa(v_i)). \quad (6.9)$$

I compare the conditional probabilities in  $\mathcal{F}(G)$  and  $\mathcal{F}(G')$  after applying a single transition from  $G$  to  $G'$  where  $G' \sim \mathcal{N}(G)$ . Theorems 6.3.6, 6.3.7 and 6.3.8 describe respectively the effect of adding, deleting, and reversing an edge on the conditional probabilities in  $\mathcal{F}(G)$  and  $\mathcal{F}(G')$ .

**Theorem 6.3.6.** Suppose  $G$  and  $G'$  are two DAGs sampled from the same search space  $\mathcal{X}$ . If  $G' \sim \mathcal{N}(G)$  and  $T_a[i, j] : G \mapsto G'$ , then,  $P(v_j | Pa(v_j)) \in \mathcal{F}(G)$  is the only conditional probability required to be updated, and there are  $n - 1$  sets of parents in  $\Delta(G)$  and  $\Delta(G')$  that are still identical.

**Proof:**

Adding a single pair  $v_i \rightarrow v_j$  to  $G$  can only change the set of parents of  $v_j$  and the set of children of  $v_i$ . The parents and children of other nodes in  $G$  remain unchanged. That is,  $Pa_G(v_i) = Pa_{G'}(v_i)$  and  $Pa_G(v_j) \neq Pa_{G'}(v_j)$ . ■

**Theorem 6.3.7.** Suppose  $G$  and  $G'$  are two DAGs sampled from the same search space  $\mathcal{X}$ . If  $G' \sim \mathcal{N}(G)$  and  $T_d[i, j] : G \mapsto G'$ , then,  $P(v_j | Pa(v_j)) \in \mathcal{F}(G)$  is the only conditional probability required to be updated, and  $n - 1$  sets of parents in  $\Delta(G)$  and  $\Delta(G')$  are identical.

**Proof:**

Deleting a single pair  $v_i \rightarrow v_j \in G$  can only change the set of parents of  $v_j$  and the set of children of  $v_i$ . The parents and children of other nodes in  $G$  remain unchanged. That is,  $Pa_G(v_i) = Pa_{G'}(v_i)$  and  $Pa_G(v_j) \neq Pa_{G'}(v_j)$ . ■

**Theorem 6.3.8.** Suppose  $G$  and  $G'$  are two DAGs sampled from the same search space  $\mathcal{X}$ . If  $G' \sim \mathcal{N}(G)$  and  $T_r[i, j] : G \mapsto G'$ , then, there are  $n - 2$  sets of parents in  $\Delta(G)$  and  $\Delta(G')$  are identical.

**Proof:**

Reversing a single pair  $v_i \rightarrow v_j \in G$  can only change the sets of parents and children of  $v_j$  and  $v_i$ . The parents and children of other nodes in  $G$  remain unchanged. That is,  $Pa_G(v_i) \neq Pa_{G'}(v_i)$  and  $Pa_G(v_j) \neq Pa_{G'}(v_j)$ . Thus,  $P(v_i | Pa(v_i)) \in \mathcal{F}(G)$  and

$P(v_j|Pa(v_j)) \in \mathcal{F}(G)$  are the only conditional probabilities that are required to be updated. ■

### 6.3.5 Algorithm and illustrative example

Algorithm 10 illustrates the adaptive scoring technique after sampling a pair of nodes, and shows how to adaptively update the conditional probabilities in  $\mathcal{F}(G)$ .

---

#### Algorithm 10 Adaptive scoring function

---

input:  $G(V, E)$

- 1: Find  $\mathcal{F}(G)$ .
  - 2: Randomly sample  $(i, j, x) \in \mathcal{E}(G)$ .
  - 3: **if**  $x = a \vee x = d$  **then**
  - 4:   Update  $P(v_j|Pa(v_j)) \in \mathcal{F}(G)$ .
  - 5: **else if**  $x = r$  **then**
  - 6:   Update  $P(v_i|Pa(v_i)) \in \mathcal{F}(G)$  and  $P(v_j|Pa(v_j)) \in \mathcal{F}(G)$ .
  - 7: **end if**
  - 8: Update  $\mathcal{E}(G)$
  - 9: **goto** 2
- 

Table 6.7 summarises the effect of a single transition between two adjacent graphs on the sets of parents for the relative nodes. If one moves from  $G$  to  $G' \sim \mathcal{N}(G)$  by deleting an edge  $v_i \rightarrow v_j$ , the CPT of node  $v_j$  must be updated, because node  $Pa(v_j)$  has lost one of its parents. If one moves to  $G' \sim \mathcal{N}(G)$  by adding an edge  $v_i \rightarrow v_j$ , the CPT of node  $v_j$  must be updated, because  $Pa(v_j)$  has added one new parent to its set. If one moves to  $G' \sim \mathcal{N}(G)$  by reversing an edge  $v_i \rightarrow v_j$ , the CPTs of node  $v_j$  and  $v_i$  must be updated, because  $Pa(v_j)$  has lost one of its parents and  $Pa(v_i)$  has added one new parent to its set.



Operation	Effect on $Pa(v_i)$	Effect on $Pa(v_j)$	CPT update
<b>Add an edge</b> $v_i \rightarrow v_j$	No effect	one new parent has been added	$P(v_j Pa(v_j))$
<b>Delete an edge</b> $v_i \rightarrow v_j$	No effect	one parent has been removed	$P(v_j Pa(v_j))$
<b>Reverse an edge</b> $v_i \rightarrow v_j$	one new parent has been added	one parent has been removed	$P(v_i Pa(v_i))$ & $P(v_j Pa(v_j))$

TABLE 6.7: Effects of adding, deleting and reversing an edge on  $Pa(v_i)$  and  $Pa(v_j)$ .

**Example 9.** This example implements Algorithm 10 for the initial Bayesian network  $G$  shown in Figure 6.4. The network  $G$  consists of 10 nodes and 13 directed edges. I aim to study the effect on the CPTs after running a single transition from  $G$  to one of its adjacent graphs.

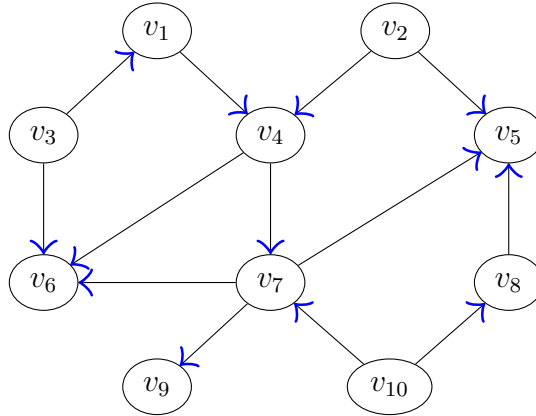


FIGURE 6.4: Initial connected DAG.

The score probability of graph  $G$  in Figure 6.4 is expressed in Equation 6.10.

$$\begin{aligned}
 P(v_1, v_2, \dots, v_{10}) &= \prod_{i=1}^{10} P(v_i | Pa(v_i)) \\
 &= P(v_1 | v_3) \cdot P(v_2) \cdot P(v_3) \cdot P(v_4 | v_1, v_2) \cdot P(v_5 | v_2, v_7, v_8) \cdot \\
 &\quad P(v_6 | v_3, v_4, v_7) \cdot P(v_7 | v_4, v_{10}) \cdot P(v_8 | v_{10}) \cdot P(v_9 | v_7) \cdot P(v_{10})
 \end{aligned} \tag{6.10}$$

**Effect on CPTs after adding an edge:** Let's first modify  $G$  in Figure 6.4 by randomly adding an edge. Suppose I add the directed edge  $v_6 \rightarrow v_9$  to the graph  $G$ . The new graph is shown in Figure 6.5. It is noted that  $Pa(v_9)$  has added one

new parent to its set. The new added parent would affect the CPT value of  $v_9$  and thus  $P(v_9|Pa(v_9))$  is updated and replaced by  $P(v_9|v_6, v_7)$ . The set  $Pa(v_6)$  has not changed, and thus its CPT is not affected by the new added edge.

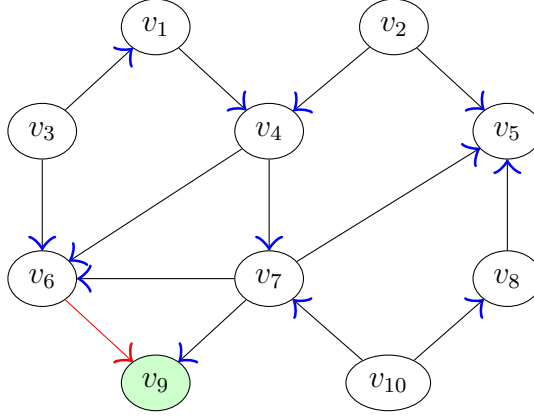


FIGURE 6.5: Moving from  $G$  to an adjacent graph after adding  $v_6 \rightarrow v_9$ .

Equation 6.10 is now updated to Equation 6.11. The updated conditional probability is underlined in Equation 6.11, where other conditional probabilities remain unchanged.

$$\begin{aligned}
 P(v_1, v_2, \dots, v_{10}) &= \prod_{i=1}^{10} P(v_i | Pa(v_i)) \\
 &= P(v_1 | v_3) \cdot P(v_2) \cdot P(v_3) \cdot P(v_4 | v_1, v_2) \cdot P(v_5 | v_2, v_7, v_8) \cdot \\
 &\quad P(v_6 | v_3, v_4, v_7) \cdot P(v_7 | v_4, v_{10}) \cdot P(v_8 | v_{10}) \cdot \underline{P(v_9 | v_6, v_7)} \cdot P(v_{10})
 \end{aligned} \tag{6.11}$$

**Effect on CPTs after deleting an edge:** I again modify  $G$  in Figure 6.4 by deleting an existing edge. Suppose I delete the edge  $v_8 \rightarrow v_5$  from  $G$  in Figure 6.4. The new graph is shown in Figure 6.6. It is noted that only  $Pa(v_5)$  has lost one of its parents. The deleted parent would affect the CPT of  $v_5$  and thus  $P(v_5|Pa(v_5))$  is updated and replaced by  $P(v_5|v_2, v_7)$ . The node  $v_8$  will not be affected by the deleted edge because its set of parents  $Pa(v_8)$  will not change.

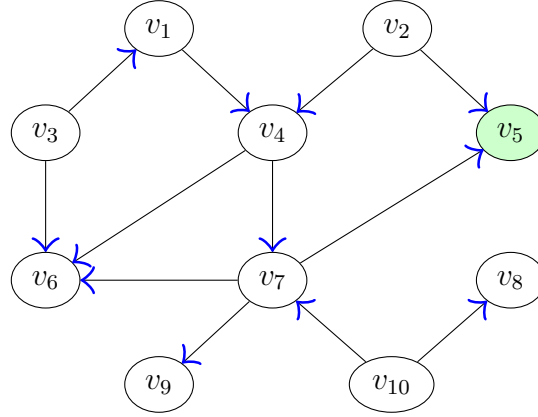


FIGURE 6.6: Moving from  $G$  to an adjacent graph after deleting  $v_8 \rightarrow v_5$ .

Equation 6.10 is now updated to Equation 6.12, and the updated conditional probability is also underlined.

$$\begin{aligned}
 P(v_1, v_2, \dots, v_{10}) &= \prod_{i=1}^{10} P(v_i | Pa(v_i)) \\
 &= P(v_1 | v_3) \cdot P(v_2) \cdot P(v_3) \cdot P(v_4 | v_1, v_2) \cdot \underline{P(v_5 | v_2, v_7)} \cdot \\
 &\quad P(v_6 | v_3, v_4, v_7) \cdot P(v_7 | v_4, v_{10}) \cdot P(v_8 | v_{10}) \cdot P(v_9 | v_7) \cdot P(v_{10})
 \end{aligned} \tag{6.12}$$

**Effect on CPTs after reversing an edge:** I lastly modify  $G$  in Figure 6.4 by reversing an existing edge. Suppose I reverse the edge  $v_5 \rightarrow v_7$  from  $G$  in Figure 6.4. The new graph is shown in Figure 6.7. It is noted that both  $Pa(v_5)$  and  $Pa(v_7)$  have been affected by the reversed edge, and thus their CPTs are updated to  $P(v_5 | v_2, v_8)$  and  $P(v_7 | v_4, v_5, v_{10})$ .

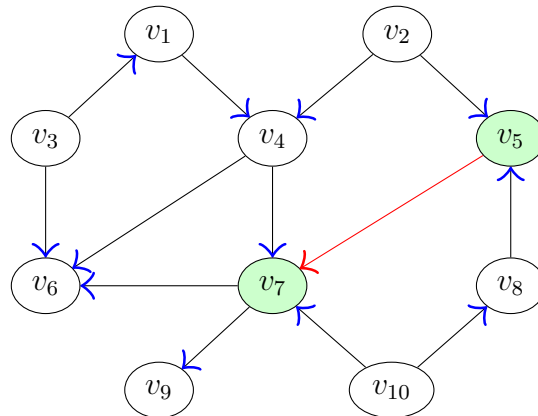


FIGURE 6.7: Moving from  $G$  to an adjacent graph after reversing  $v_5 \rightarrow v_7$ .

Equation 6.13 formulates the scoring probability of the Bayesian network in Figure 6.7, and underlines the two updated conditional probabilities.

$$\begin{aligned}
 P(v_1, v_2, \dots, v_{10}) &= \prod_{i=1}^{10} P(v_i | Pa(v_i)) \\
 &= P(v_1 | v_3) \cdot P(v_2) \cdot P(v_3) \cdot P(v_4 | v_1, v_2) \cdot \underline{P(v_5 | v_2, v_8)} \cdot \\
 &\quad P(v_6 | v_3, v_4, v_7) \cdot \underline{P(v_7 | v_4, v_5, v_{10})} \cdot P(v_8 | v_{10}) \cdot P(v_9 | v_7) \cdot P(v_{10})
 \end{aligned} \tag{6.13}$$

■

**Example 10.** Table 6.8 compares experimental run times for the standard method against experimental run times for the adaptive method, to calculate the scoring function using the Dirichlet-Multinomial distribution. The run times of the two methods were evaluated at different numbers of iterations, including 100, 1,000 and 10,000. The number of nodes is 51, and the maximum number of parents and children for each node are two and five, respectively. The number of observations for each node is 163, and each node takes three discrete state values. Note that this application is presented in detail in Chapter 7 Section 7.5.

Iteration \ Method	Method	
	Brute-force	Adaptive
100	30 - 40 minutes	1 - 2 minutes
1,000	5 - 7 hours	15- 20 minutes
10,000	2 - 3 days	2- 3 hours

TABLE 6.8: Function scoring algorithm: Brute-force method vs Adaptive method, with 51 nodes using Dirichlet-Multinomial distribution.

## Chapter 7

# Applications of Bayesian networks in Systems Biology Using the MH, NS and HAR

In this chapter, I use the NS, HAR and MH to infer various biological BNs. I will use the Dirichlet-multinomial (DM) model to compute the conditional probabilities among variables within the sampled networks. The chapter includes five applications. It starts with a Bayesian network of four nodes learned from microarray data. The total number of graphs sampled from a graph space of four nodes is small enough to plot and identify all sampled graphs. The second application applies the Geweke diagnostic test to a biological Bayesian network with six nodes to assess convergence behavior. The third application applies the Gelman and Robin diagnostic test to a medical Bayesian network of eight nodes. The test essentially requires running multiple chains with disparate initial networks. The fourth application attempts to infer a biological network using MCMC samplers. Note that there is no gold standard network in this application, and thus the inferred structures are compared with the currently accepted structure. The fifth application discusses a recently inferred signaling pathway structure consisting of 51 nodes, and examines the possibility of enhancing its scoring function using MCMC sampling.

**Remark 8.** Using a *graphical user interface* developed for this project, it is possible in the simulation settings to define a maximum number of node-parents, maximum number of node-children, number of iterations, burn-in interval, thinning interval, and initial networks. The datasets of applications may be either simulated so that the conditional probabilities are known, or observed datasets collected from real experiments as in applications four and five.

**Remark 9.** All simulated data in this chapter were generated using the Netica software which has an intuitive user interface. Netica works with BNs and influence diagrams, and has many convenient features for drawing BNs, and defining relationships among variables either by using individual probabilities, equations or learning from a data file. For more detail of the features and specifications of Netica software, the reader is referred to (<https://www.norsys.com/>).

## 7.1 Inferring structures from Microarray data

### 7.1.1 Background

I consider a small Bayesian network to explore and plot all sampled graphs, and then calculate their posterior means. The network is learned from Microarray data in [156]. This network models the causal relationships between the expression levels of four genes: Gene A, Gene B, Gene C, and Gene D. The genes are represented by nodes and their causal influences are represented by directed edges. Section C.1 provides the conditional probabilities of the Microarray network. There are three directed edges connecting the four genes. Each gene is represented by a binary random variable that takes two state values: ‘off’ or ‘on’. These states are denoted ‘0’ and ‘1’ respectively in the conditional probability tables.

### 7.1.2 Experimental results

One useful way to summarise an MCMC sample is to report the estimated marginal posterior probability for each individual edge by determining the proportion of sampled graphs in which that edge is present. I simulated 5000 data points for each variable. The true network and CPTs used to simulate the data are shown in Section C.2. I then attempted to infer the network using the NS, HAR and MH algorithms. At each iteration, a graph is sampled and its probability function is estimated using the DM model given the dataset. Also note that there are no other networks that are equivalent to the true network. Recall that the definition of graph equivalence requires that two graphs have the same v-structures. The true Bayesian network learned in [156] takes the form of single v-structures, hence the absence of equivalent graphs. The log likelihood function was plotted for 1000 iterations for the NS, MH and HAR as shown in Figure 7.1. The samplers converge rapidly to the stationary distribution. I therefore apply a short burn-in period of about 25 iterations for all samplers by discarding the first 25 iterations.

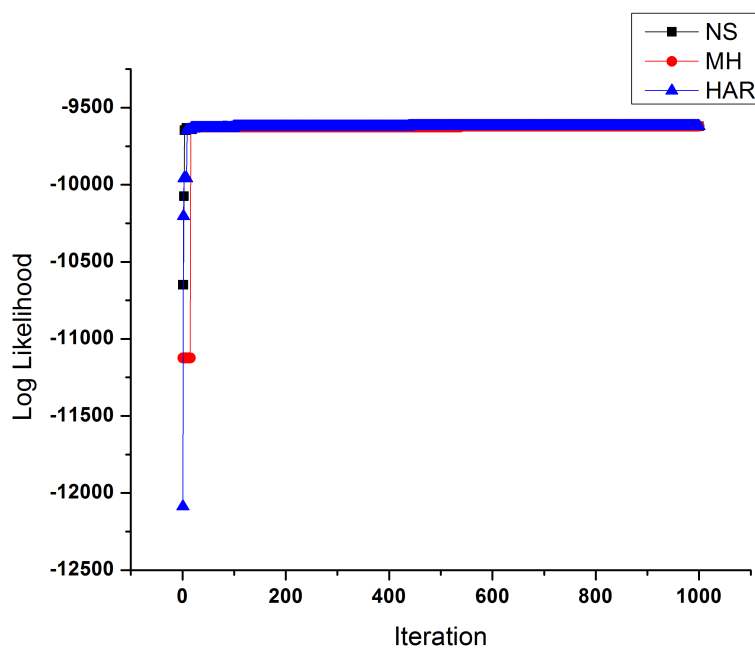


FIGURE 7.1: Log likelihood functions for 1000 iterations.

The total number of graphs sampled by the MH sampler, HAR sampler and NS after burning-in 25 iterations were 12, 10 and 7 respectively. The sampled graphs and their frequencies are shown in Figure 7.2, Figure 7.3 and Figure 7.4.

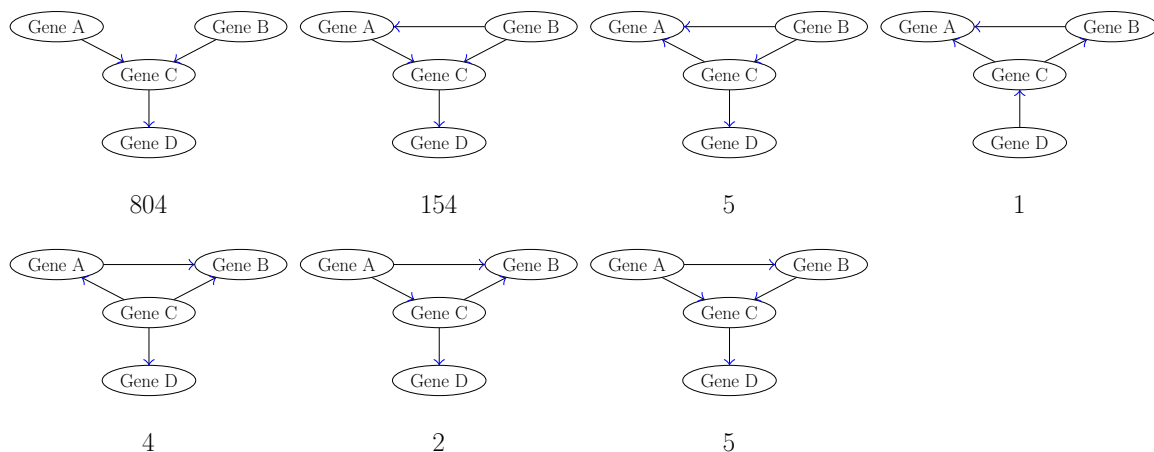


FIGURE 7.2: Seven sampled graphs and their frequencies in 975 iterations of the NS, after a burn-in period of 25 iterations.

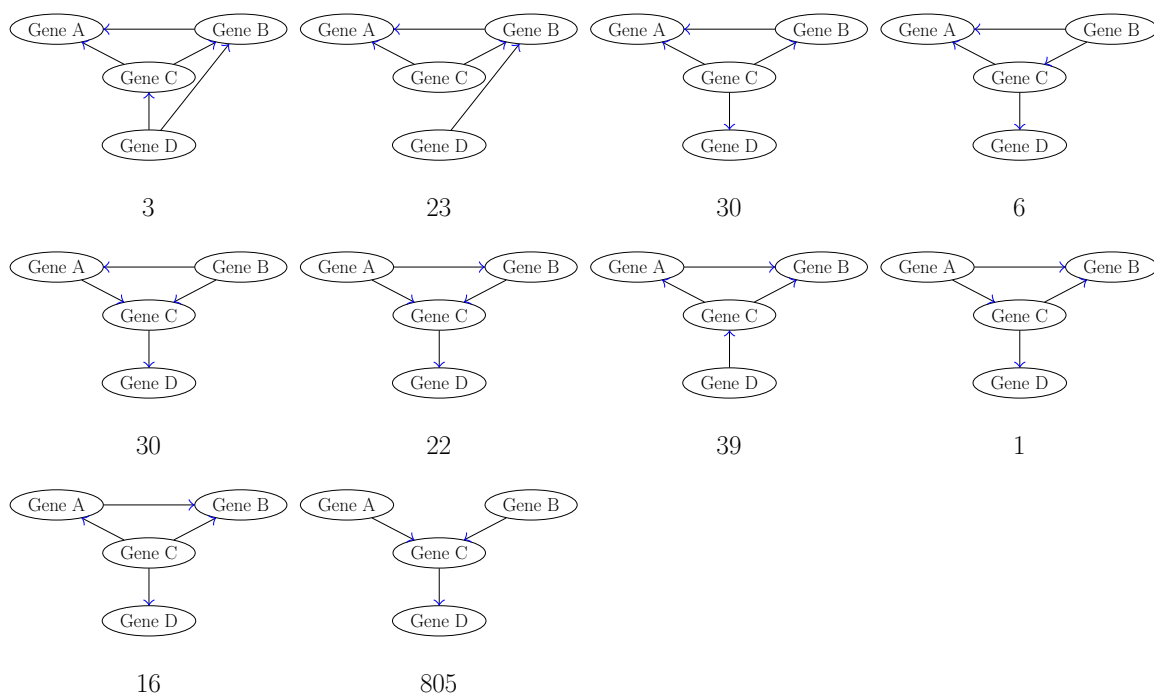


FIGURE 7.3: Ten sampled graphs and their frequencies in 975 iterations of the HAR, after a burn-in period of 25 iterations.



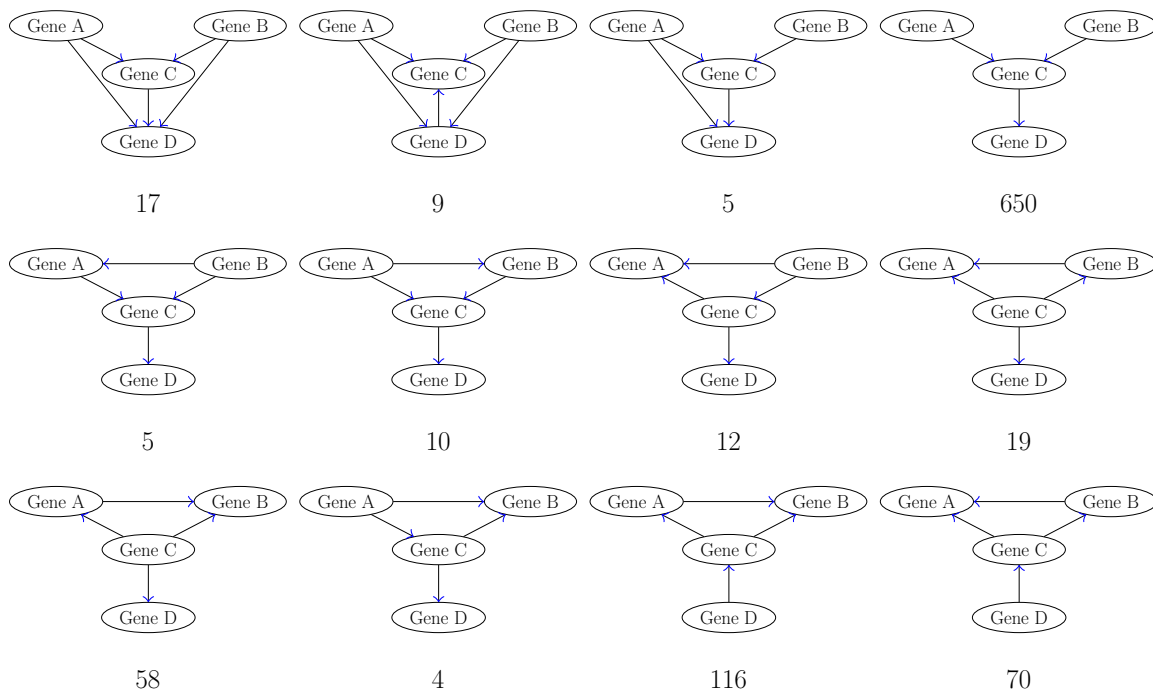


FIGURE 7.4: Twelve sampled graphs and their frequencies in 975 iterations of the MH, after a burn-in period of 25 iterations.

If it is necessary to specify a single network as the best reconstruction, one approach is to include only those edges whose posterior probabilities exceed a given threshold. A threshold value of 0.5 has an intuitive appeal, as it identifies edges that are more likely to be present than absent. Edges exceeding the threshold are shown in Figure 7.5. These posterior networks correspond to the true network.

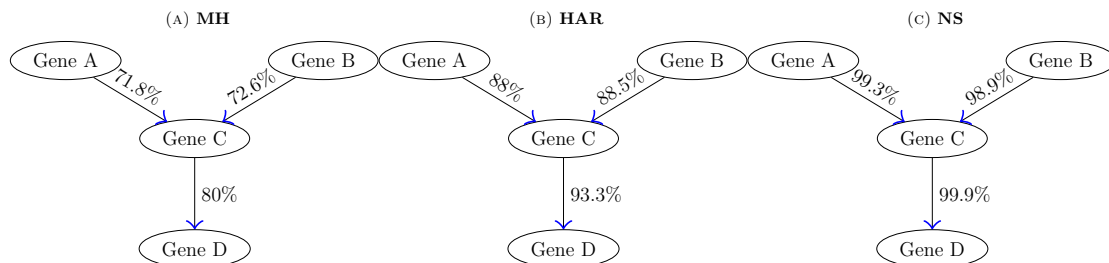


FIGURE 7.5: The sampled edges whose posterior probabilities  $> 50\%$ .

### 7.1.3 Conclusion

The number of graphs explored by the three samplers was much less than the number of graphs in the space. This suggests that the posterior probabilities of the remaining graphs are very low. Although the chains produced by the MH, HAR and NS appear to have converged to the same distribution, the posterior means of the inferred edges after applying a burn-in interval are slightly different. This may suggest true convergence has not yet occurred.

## 7.2 Inferring the Mendel Peas network

### 7.2.1 Background

In 1866, Gregor Mendel proposed some foundational principles to understand how inherited traits are passed between generations. Mendel's principles are applicable to trait inheritance in both plants and animals. The network considered in this section is a representation of the genetics underlying Mendel's famous peas experiment. This network was designed by Norsys Software Corp in 1998 and includes six variables. The two variables P1 and P2 are associated to produce another variable C. Each of these variables represents a plant genotype and has three possible state values RR, Rr and rr, where R is the allele for red and r is the allele for white. These three variables probabilistically determine an additional three variables: the observed colours of P1, P2 and C. Each of these colour variables has two possible state values: red and white.

### 7.2.2 Experimental results

I used the conditional probabilities presented in Section C.2.1 to simulate 5000 data-points for each variable in the Mendel network. I first investigated the convergence behavior of the six genes in Mendel network by reporting the log posterior of graphs. Figure 7.6 plots the log posterior of graphs against 5000 iterations. Figure 7.6

suggests a burn-in interval of approximately 250 iterations for all samplers. Thus, I discard the first 500 iterations and assessed the convergence behavior using the remainder.

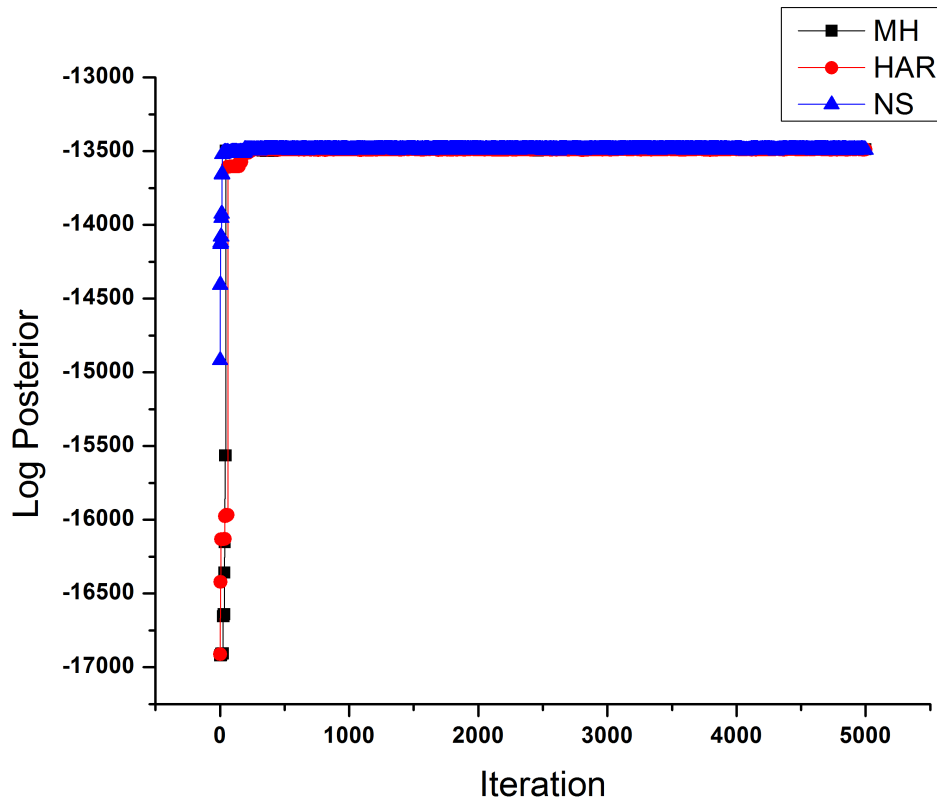


FIGURE 7.6: Log posterior values for 5000 iterations.

I applied the Geweke diagnostic test to assess convergence for the three samplers. I assumed that the second half of the Markov chain has converged to the target distribution, and then tested whether an early portion of the Markov chain passes the Geweke diagnostic test. Figure 7.7, Figure 7.8 and Figure 7.9 show that the number of values occurring outside two standard deviations is small, and therefore it is not necessary to run a longer chain, especially for the NS and HAR in Figure 7.8 and Figure 7.9, respectively. The MH sampler has returned two Z score values occurring outside two standard deviations and one of them is less than  $-3$  as shown in Figure 7.7.

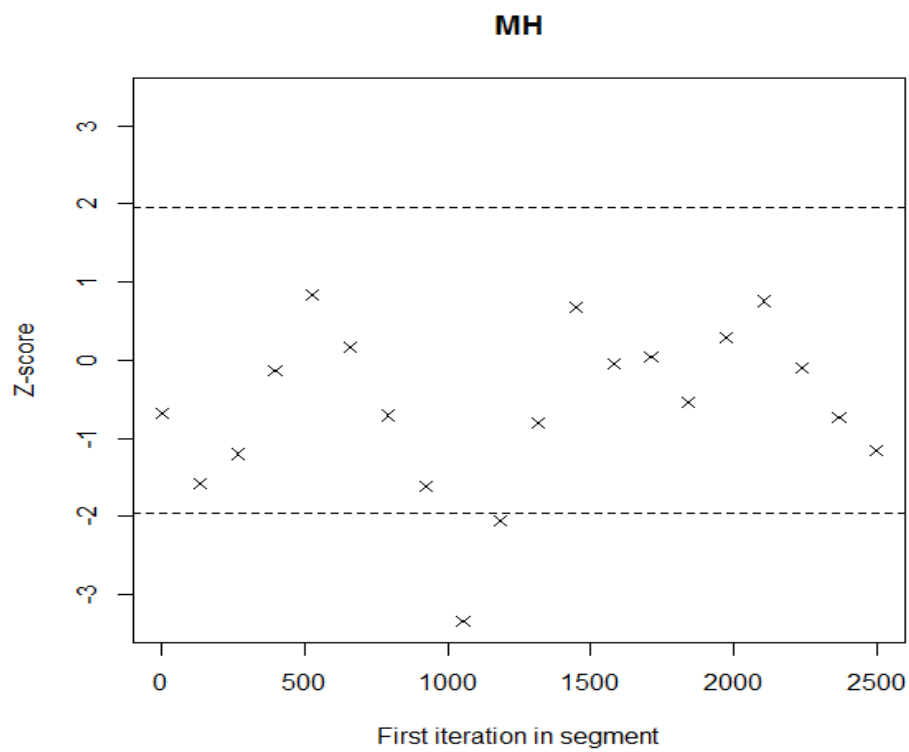


FIGURE 7.7: Geweke diagnostic test with the MH using 5000 iterations.

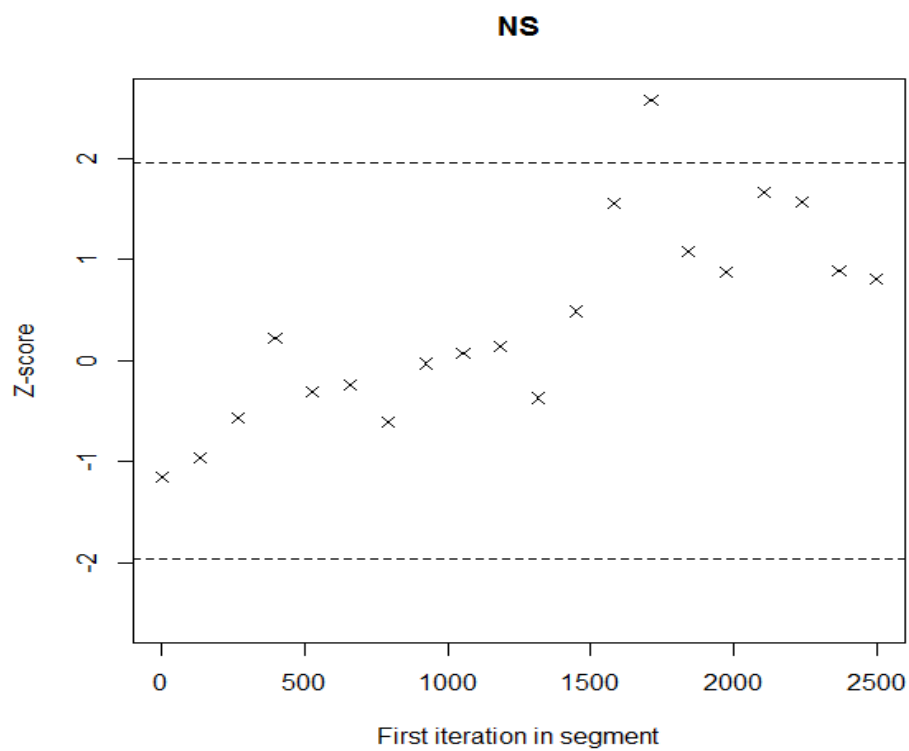


FIGURE 7.8: Geweke diagnostic test with the NS using 5000 iterations.

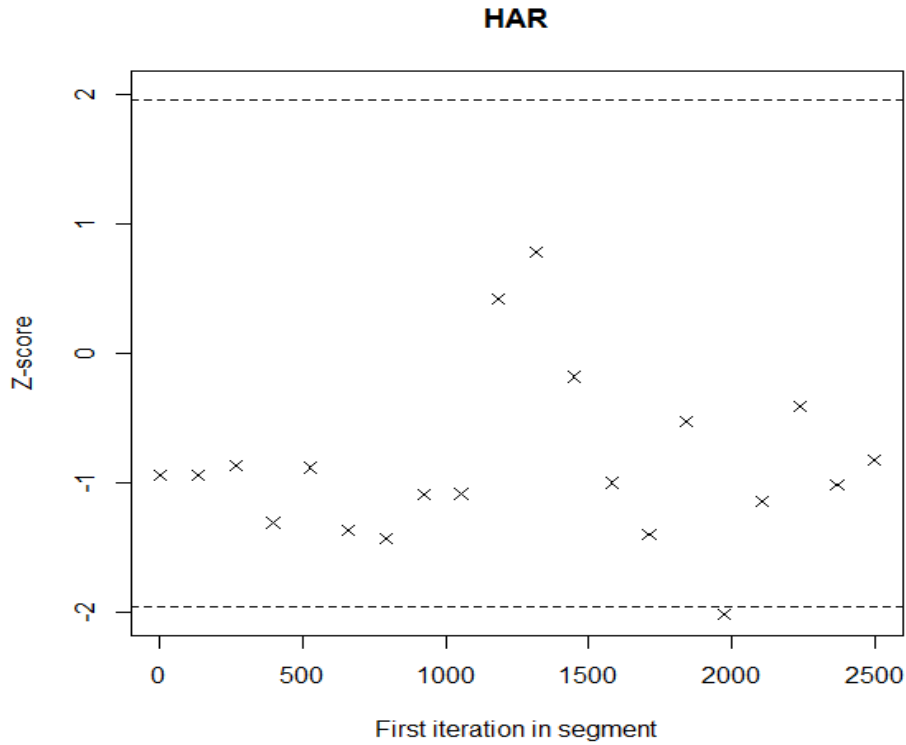


FIGURE 7.9: Geweke diagnostic test with the HAR using 5000 iterations.

The outputs of the three chains produced by the MH, HAR and NS using 5000 iterations are summarised in Table C.1 in Section C.2.2. Note that the summary statistics reported in Table C.1 considered all graphs sampled from the 5000 iterations before applying a burning-in interval.

The posterior mean probabilities of edges for the three chains plotted in Figure 7.6 were also calculated after applying a burn-in interval of 500 iterations each. Figure 7.10 shows the structures learned by the MH, HAR and NS at a threshold  $\geq 50\%$ . The three samplers have precisely inferred the true Mendel network.

**Remark 10.** Section C.5 investigates the potential of the MH, HAR and NS to infer the Mendel network when the transition between adjacent graphs by reversing an edge is not allowed. The performances of the three samplers are evaluated by running three chains of 1000 iterations by each sampler. The posterior means of edges for the nine chains are plotted in Section C.5 at a threshold  $\geq 50\%$ . One

result is that, unlike the HAR and NS, not one of the three chains run by the MH sampler could infer the true structure of the Mendel network.

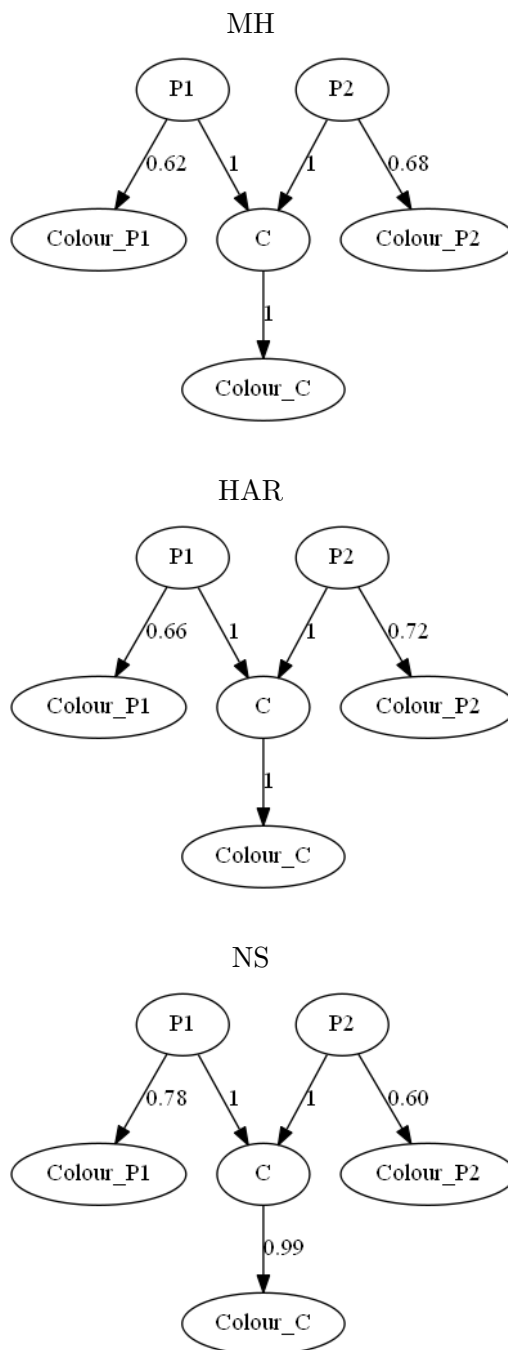


FIGURE 7.10: The posterior means of edges at a threshold  $\geq 50\%$  using the MH, HAR and NS with 5000 iteration and burning-in of 500 iterations.

### 7.2.3 Conclusion

The experimental results in this example have shown the following:

1. Sampling from the target distribution using the MH, HAR and NS can be achieved with at most 5000 iterations.
2. The log posterior values produced by the three samplers suggest all samplers are exploring a common mode after a few hundred iterations.
3. It is possible for Markov chains in a space of BNs with six nodes produced by the MH, HAR and NS to pass the Geweke diagnostic test with 5000 iterations, indicating rapid convergence to the target distribution.

## 7.3 Inferring the Diagnostic Chest Clinic network

### 7.3.1 Background

The Diagnostic Chest Clinic (DCC) network shown in Figure 7.14a is a popular medical Bayes net example, also known as the “Asia” dataset, from [2]. The DCC network aims to represent the risks of a patient having tuberculosis, lung cancer or bronchitis based on several factors, including whether or not a patient is a smoker or has traveled to Asia recently. The network assumes causal relationships among eight variables. Each variable takes a binary value, either 0 or 1 respectively indicating the absence or presence of a particular risk.

### 7.3.2 Experimental results

The maximum numbers of parents and children of each node in the true network are two each. I therefore assumed that the network is connected and that the number of parents and children of each node can never be greater than four. These restrictions reduce the number of graphs in the graph space and the computational time required

to sample from the posterior distribution. I also used the conditional probabilities defined in Section C.3 to simulate 5000 data-points for each variable.

I have used the Gelman & Rubin diagnostic as a convergence test. The Gelman & Rubin test is applicable when the number of chains is greater than or equal to two. Also, the test requires running  $2n$  iterations, and then applying it to the last  $n$  iterations. I therefore ran three chains of  $2n = 10000$  iterations for each MCMC sampler. I also fixed three random initial graphs that are disparate in their structures for each Markov sampler. The Gelman & Rubin test applied to the last  $n = 5000$  iterations of chains suggests that convergence has occurred after 5000 iterations. Figure 7.11, Figure 7.12 and Figure 7.13 show that the scale reduction factors of 50% and 97.5% quantiles calculated for the MH, NS, and HAR are 1.01 and 1.03, 1.01 and 1.03, and 1.03 and 1.08, respectively. All these numbers are less than 1.2 and close to 1, which indicates that the number of iterations does not need to be increased, and graphs have been effectively sampled from the DM distribution.

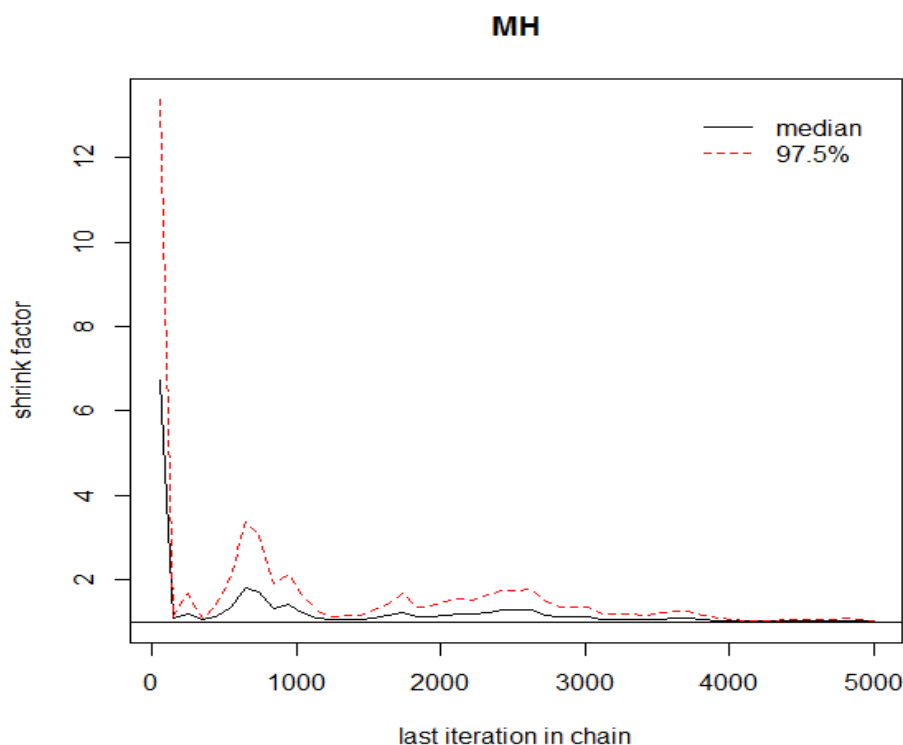


FIGURE 7.11: Gelman & Rubin diagnostic test with the MH.



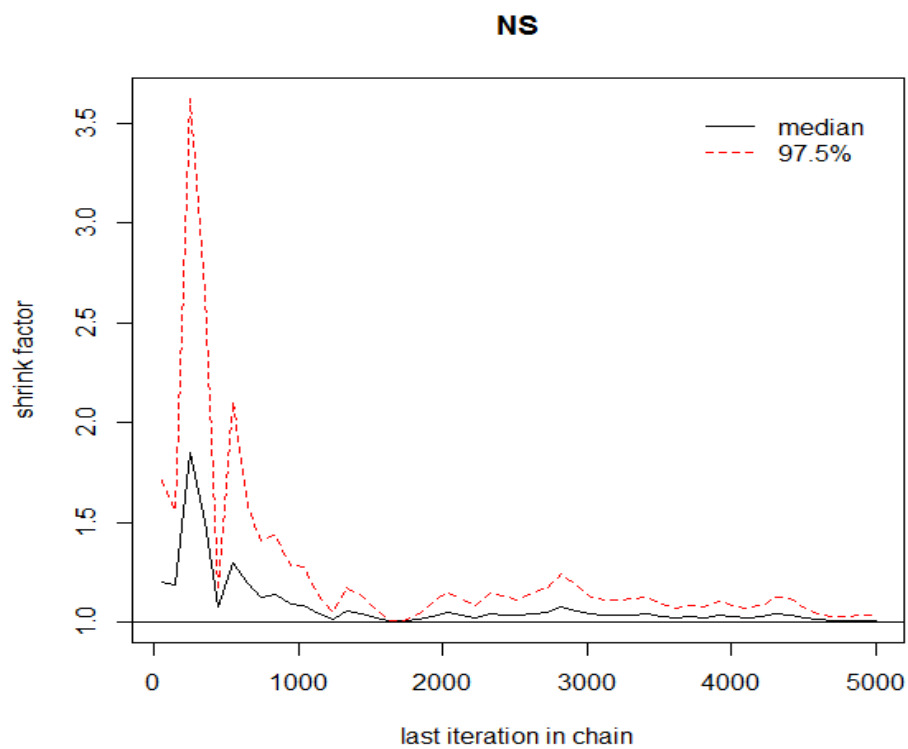


FIGURE 7.12: Gelman & Rubin diagnostic test with the NS.

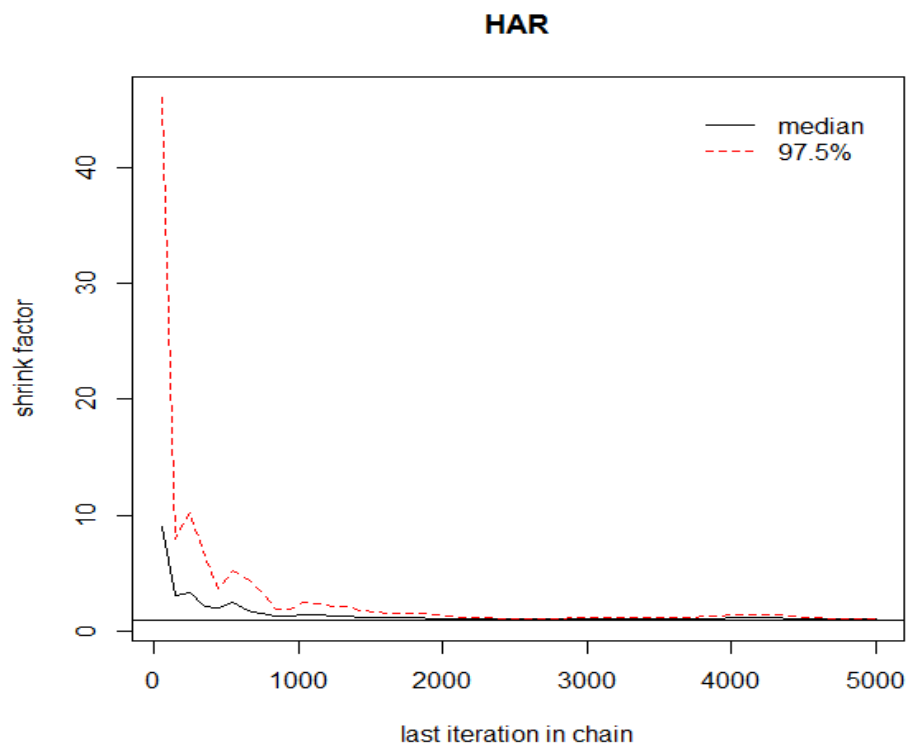


FIGURE 7.13: Gelman & Rubin diagnostic test with the HAR.

The 5000 samples produced by the last 5000 iterations of the chains are used to calculate the posterior means using the MH, HAR and NS. The three MCMC samplers are compared with three non-MCMC samplers which have been widely used in practice to learn Bayesian network structures: Grow-Shrink (GS) algorithm, Hill-Climbing Search (HCS) and Tabu Search (TS). The GS, HCS and TS have been briefly reviewed in Chapter 3.

The GS, HCS and TS were also applied to the same 5000 simulated data-points. The *bnlearn* R package [157] was used to apply the GS, HCS and TS. The three non-MCMC algorithms were initially run at the default settings in the *bnlearn* package. The structures learned by the GS, HCS and TS are shown in Figure 7.14b, Figure 7.14c and Figure 7.14d, respectively.

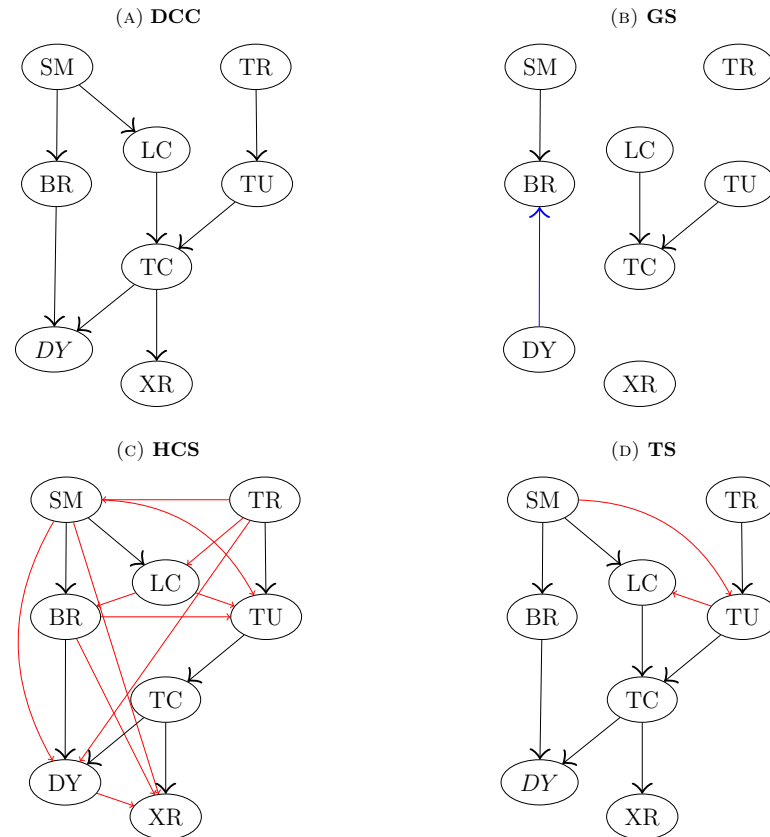


FIGURE 7.14: (A) The true DCC network. (B), (C), and (D) The DCC network learned by the GS, HCS and TS, respectively, using 5000 simulated data-points. Each inferred edge is colored either by black, blue or red to indicate the inference of a correct edge, reversed edge or incorrect edges, respectively.

The structures learned by MCMC samplers are shown in Figure 7.15.

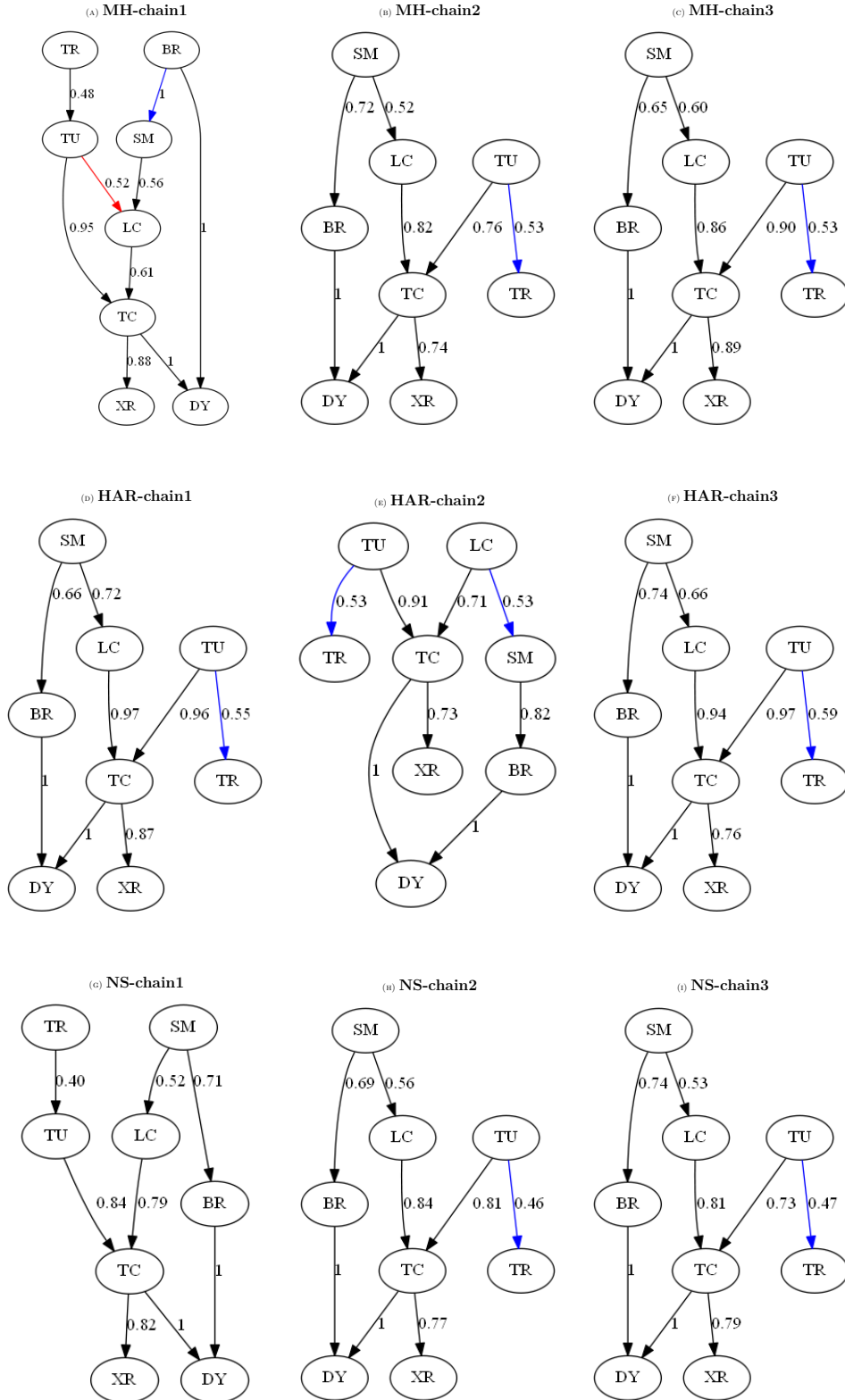


FIGURE 7.15: The DCC network learned by MCMC samplers using 5000 simulated data-points.

The inferred edges are colored using three different colors: black, blue and red to indicate the following: correct edges, reversed edges and incorrect edges, respectively. The dashed directed edges refer to the missing edges that are in the true network but not in the learned network. Even though I increased the number of iterations for the HCS and TS algorithms up to 100,000 using different values of tabu at 10, 50 and 100, no solutions better than those shown in Figures 7.14c and 7.14d were obtained.

The MH, HAR and NS in Figure 7.15 have approximated the true structure effectively with fewer improper directed edges compared to the non-MCMC samplers. It is noted that the direction between the pair of nodes “T” and “R” was the only edge that was not sampled in the correct direction with the MCMC samplers. There are two possible reasons for this slight prediction error that might explain this incorrect direction. First, the expected number of individuals who have both traveled and have Tuberculosis is only 5 out of 10,000, and thus the actual number of such individuals simulated has high proportional variation. Second, this sampled network is equivalent to the true network, and should in principle have the same posterior probability.

### 7.3.3 Conclusion

This example has demonstrated the following:

1. The HAR and NS can infer the true network of eight nodes after estimating the posterior means of edges at a threshold  $\geq 50\%$  with less improper directed edges compared to the MH and non-MCMC samplers e.g. GS.
2. Sampling from the target probability distribution of a Bayesian network of eight nodes is often achieved with the NS and HAR after 5000 iterations, and this is likely to be true for small networks.

3. If one considers the *underlying structure* (where all the edges are undirected) of the true network with eight nodes, the MH, HAR and NS have the potential to infer the same underlying structure of the true network (experimentally, 10 networks out of the 12 networks in Figure 7.15 have the same underlying structure of the true network).
4. 10,000 iterations appears experimentally to be sufficient number to learn a Bayesian network structure consisting of eight nodes and to sample graphs from the target distribution, however, this number might need to be increased if the number of state values for each node is large.

## 7.4 Inferring the Raf-Signaling Pathway network

### 7.4.1 Background

One medium-scale dataset in biological systems is the Raf-Signaling Pathway. The dataset was collected from real experiments and first studied in [158]. I aim to model the causal interactions - as a causal Bayesian network - of Raf-Signaling proteins. The network in Figure 7.16a connects a number of key phosphorylated proteins in human T cell signaling. The network was mapped using classical genetic and biochemistry analysis over the past two decades. Also, the network was constructed with no *a priori* knowledge of pathway connectivity. There are 11 nodes in the network, where the nodes represent proteins (phosphoproteins and phospholipids): RAF (praf), MEK (pmek), PCLg (plc-gamma), PIP2, PIP3, ERK (p44.42), AKT (pakts473), PKA, PKC, P38, and JNK (pjnk). Edges among proteins represent interactions, and arrows indicate the direction of the transmission of protein signals.

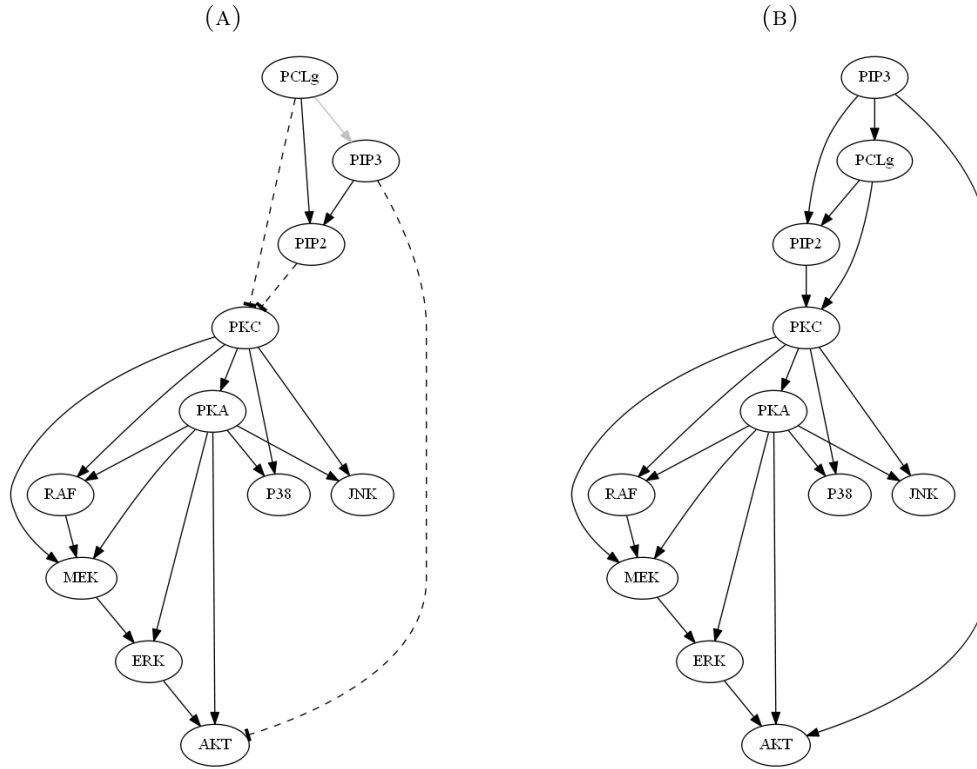


FIGURE 7.16: Raf Signaling Pathway. (A) Original network derived in [158] using classical genetic and biochemistry analysis over the past two decades. In comparison to the network in B, the dashed edges are missed and the one gray edge is reversed. (B) The currently recognised signaling network learned experimentally in the literature.

### 7.4.2 Experimental results

For each node-protein, there are 5400 continuous samples of signaling pathways collected with no missing data-points. The sample values were discretized into three categories: "High", "Medium" and "Low" as presented in [158]. I use MCMC samplers to map the Raf-Signaling Pathways given the discretized dataset. I use the DM distribution to compute the conditional probabilities among proteins. The space of BNs with 11 nodes contains strictly greater than  $4.2 \times 10^{18}$  structures. I reduce this size by setting the maximum numbers of parents and children for each node-protein to three and six, respectively. The latter two numbers were determined according to the maximum numbers of parents and children inferred in [158]. It is also practically desirable to define random and disparate initial graphs. Note

that there is no gold standard structure for the Raf-Signaling network, and thus I compare our inferred structures with the Raf-Signaling Pathway learned in [158], which is plotted in Figure 7.16a. In this application, I used the predefined twelve initial graphs shown in Section C.4.1. The twelve initial graphs were determined randomly using the technique explained in Section 4.8 in Chapter 4.

I used the MH, HAR and NS approaches to run twelve Markov chains with 10,000 iterations each, given the predesignated twelve initial networks. Then, I determined the point at which burn-in has occurred considering the time-series plot of the log posterior at each iteration. Figure 7.17, Figure 7.18 and Figure 7.19 show the log posterior distributions for the twelve Markov chains and 10,000 iterations, produced by the MH, HAR and NS, respectively. The 36 chains are also plotted separately in Section C.4.2.

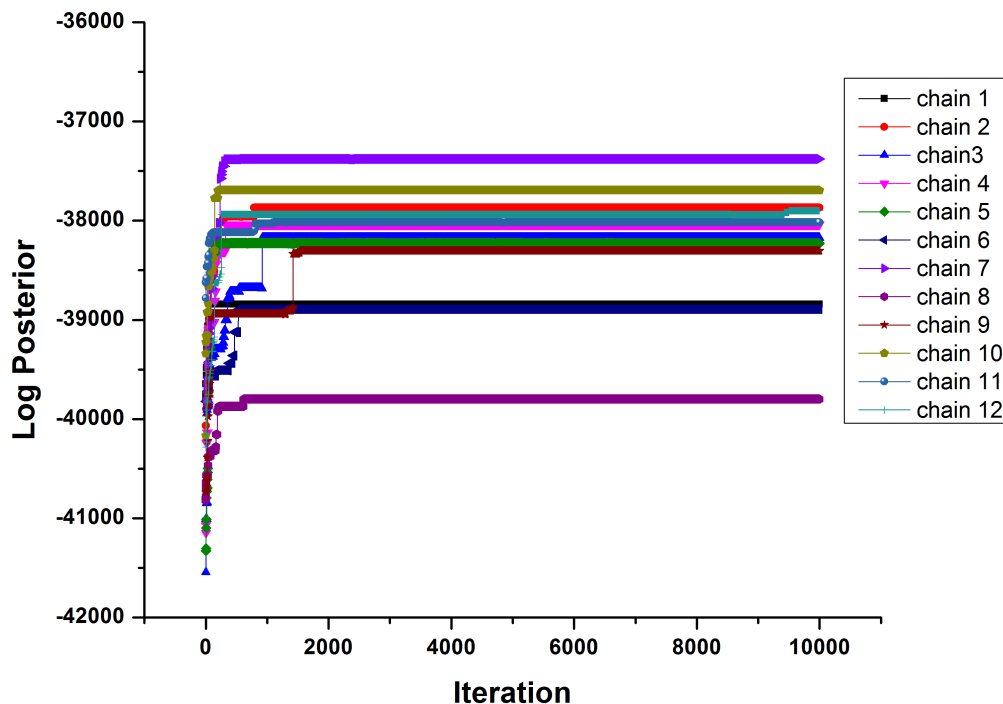


FIGURE 7.17: MH: Log posterior for 12 chains and 10,000 iterations each.

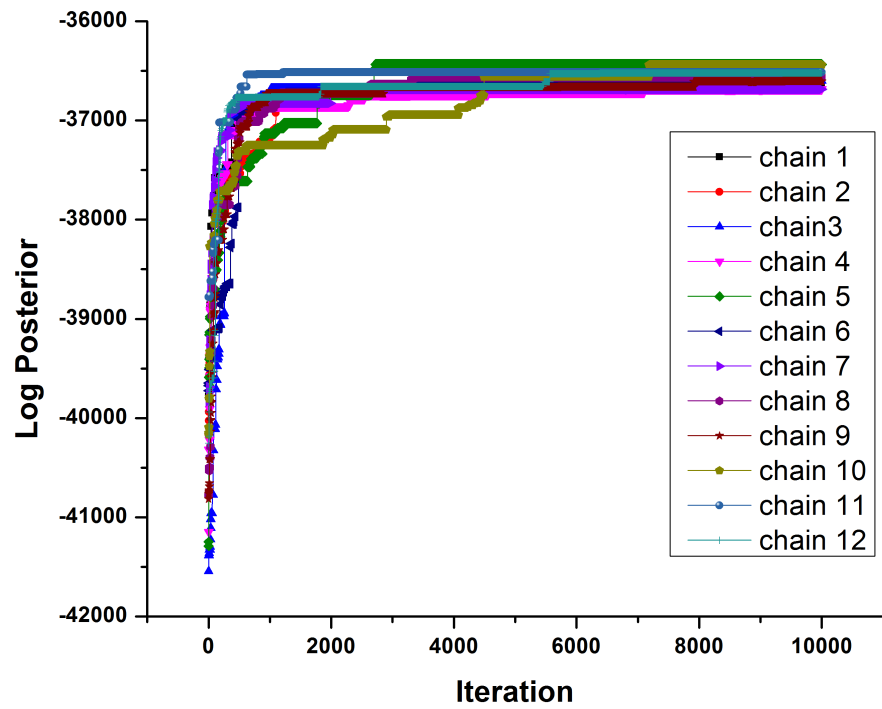


FIGURE 7.18: HAR: Log posterior for 12 chains and 10,000 iterations each.

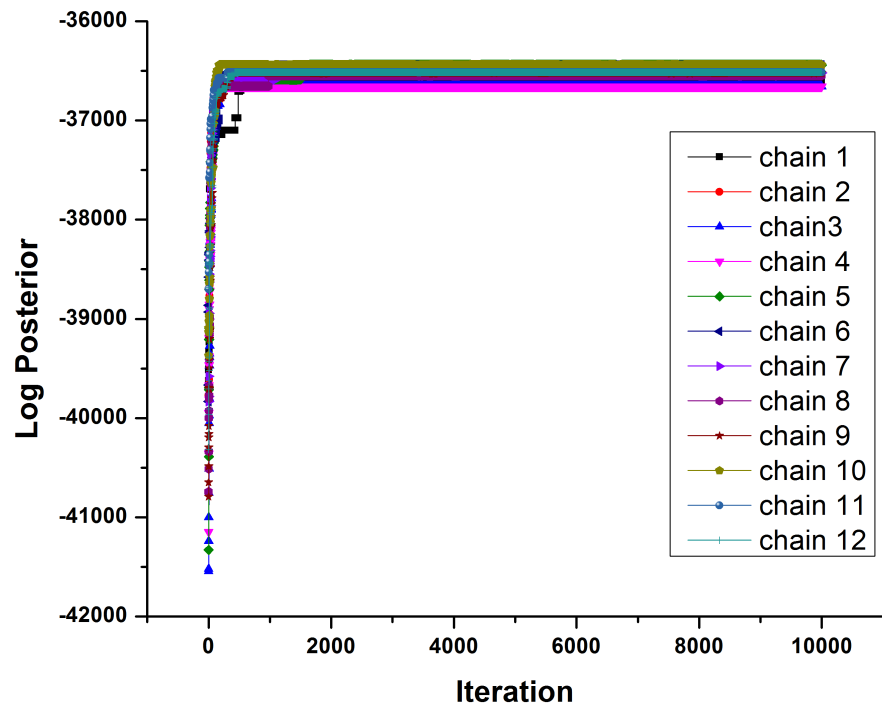


FIGURE 7.19: NS: Log posterior for 12 chains and 10,000 iterations each.



Figure 7.17, Figure 7.18 and Figure 7.19 are used to evaluate and compare the performances of the MH, HAR and NS. Notably, chains in Figure 7.17 are stuck in different local modes, so they clearly have not converged. It is also the fact that increasing the number of iterations for the chains in Figure 7.17 did not help. This highlights a problem with the MH sampler. Strictly speaking, burn-in is not complete, but nevertheless I have identified a "local burn-in" time at which behaviour appears to stabilise in the local mode.

The chains generated by the HAR sampler in Figure 7.18 have ultimately converged to a common local mode even though some chains took a long time to converge. The log likelihoods in Figure 7.18 are higher than the corresponding log likelihoods of the local modes in Figure 7.17.

The log likelihoods generated by the NS in Figure 7.19 have stabilised early to the same common mode reached by the HAR sampler. This suggests shorter burn-in intervals for the chains produced by the NS.

After discarding the burn-in intervals, I report the posterior edge probabilities for the rest of each chain. Figures 7.20 - 7.31 plot the directed edges that have posterior probabilities  $> 50\%$  sampled by the MH, HAR and NS for all the generated chains from 1 to 12, respectively. Figures 7.20 - 7.31 also facilitate comparing the structures learned by the MH, HAR and NS for all chains separately. Again, I compare every inferred structure with the original network in Figure 7.16a. For simplicity, I highlight the true inferred directed edges in *black color*, the reversed true directed edges in *blue color* and the new inferred directed edges are *dashed*, as shown in all Figures 7.20 - 7.31. The probability strength of each inferred directed edge is outlined near the edge on all plots.

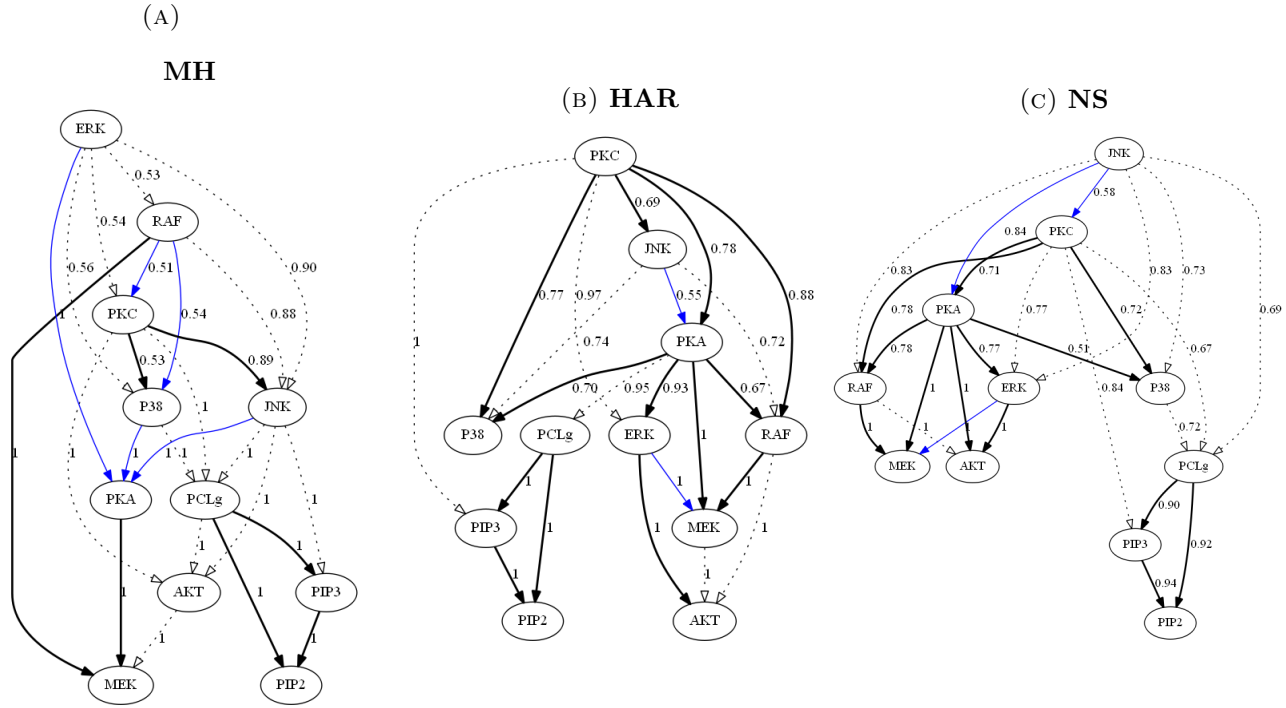


FIGURE 7.20: Posterior means > 50% of the 1st chain. Black, blue and dashed edges highlight the true, reversed and new edges, respectively.

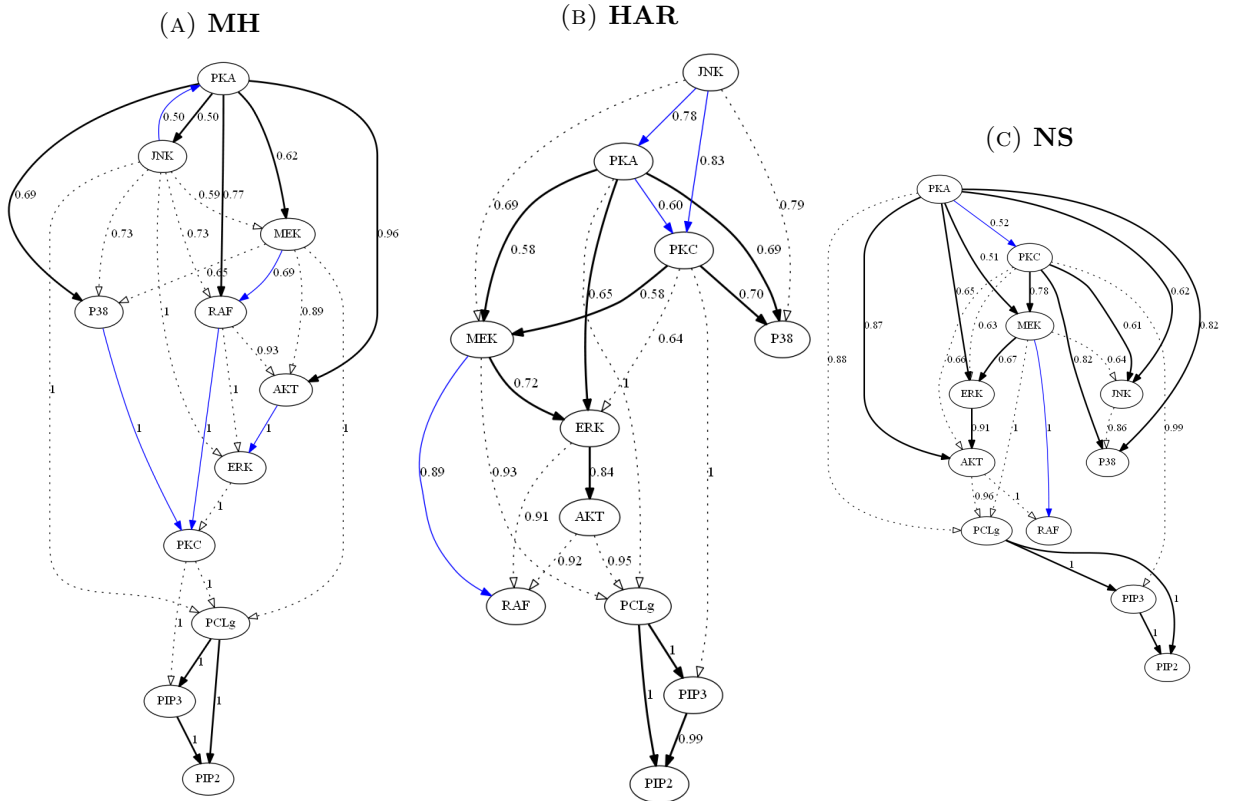


FIGURE 7.21: Posterior means > 50% of the 2nd chain. Black, blue and dashed edges highlight the true, reversed and new edges, respectively.

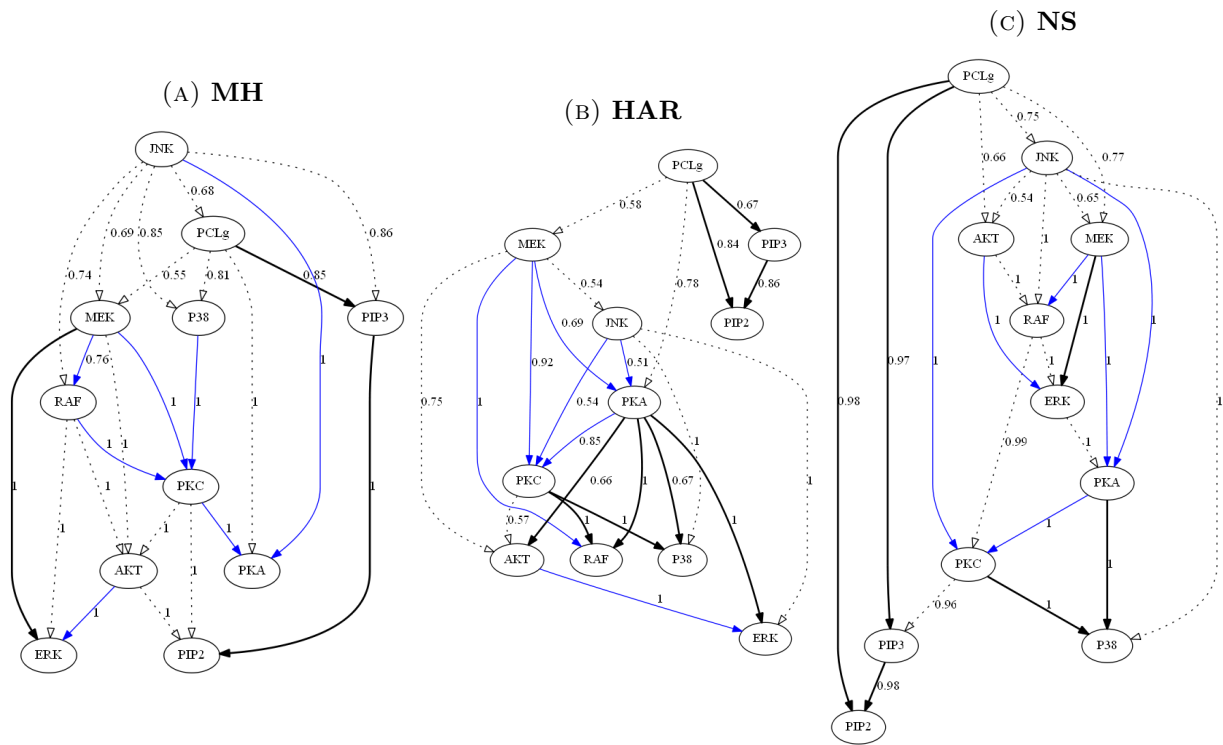


FIGURE 7.22: Posterior means  $> 50\%$  of the 3rd chain. Black, blue and dashed edges highlight the true, reversed and new edges, respectively.

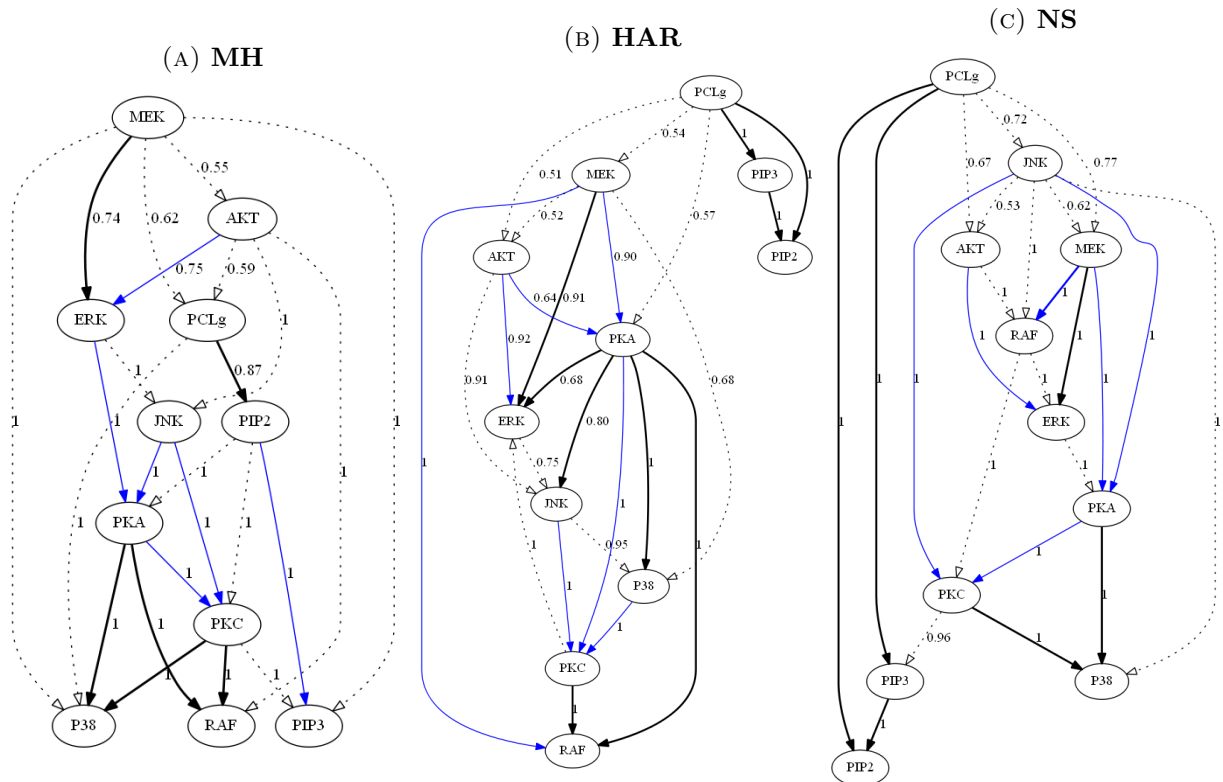


FIGURE 7.23: Posterior means  $> 50\%$  of the 4th chain. Black, blue and dashed edges highlight the true, reversed and new edges, respectively.

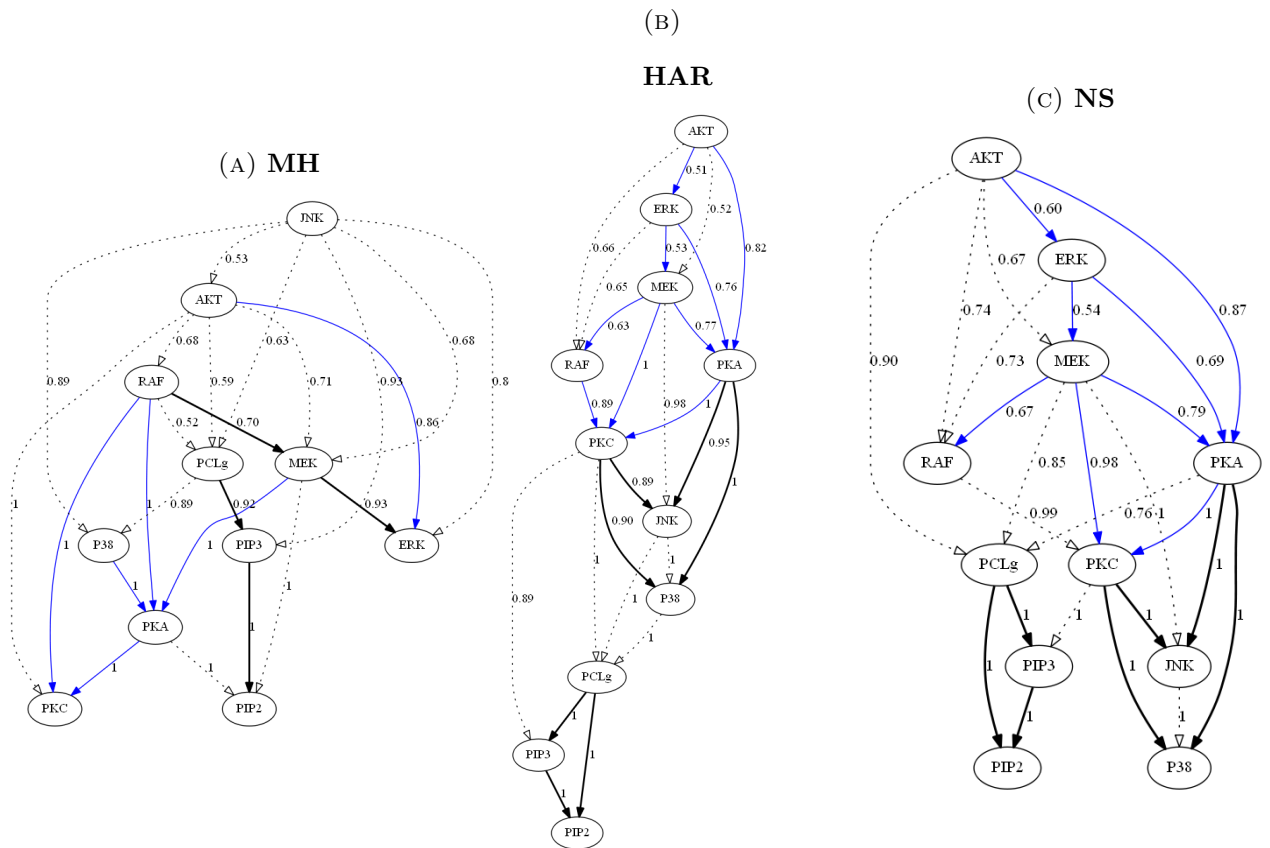


FIGURE 7.24: Posterior means  $> 50\%$  of the 5th chain. Black, blue and dashed edges highlight the true, reversed and new edges, respectively.

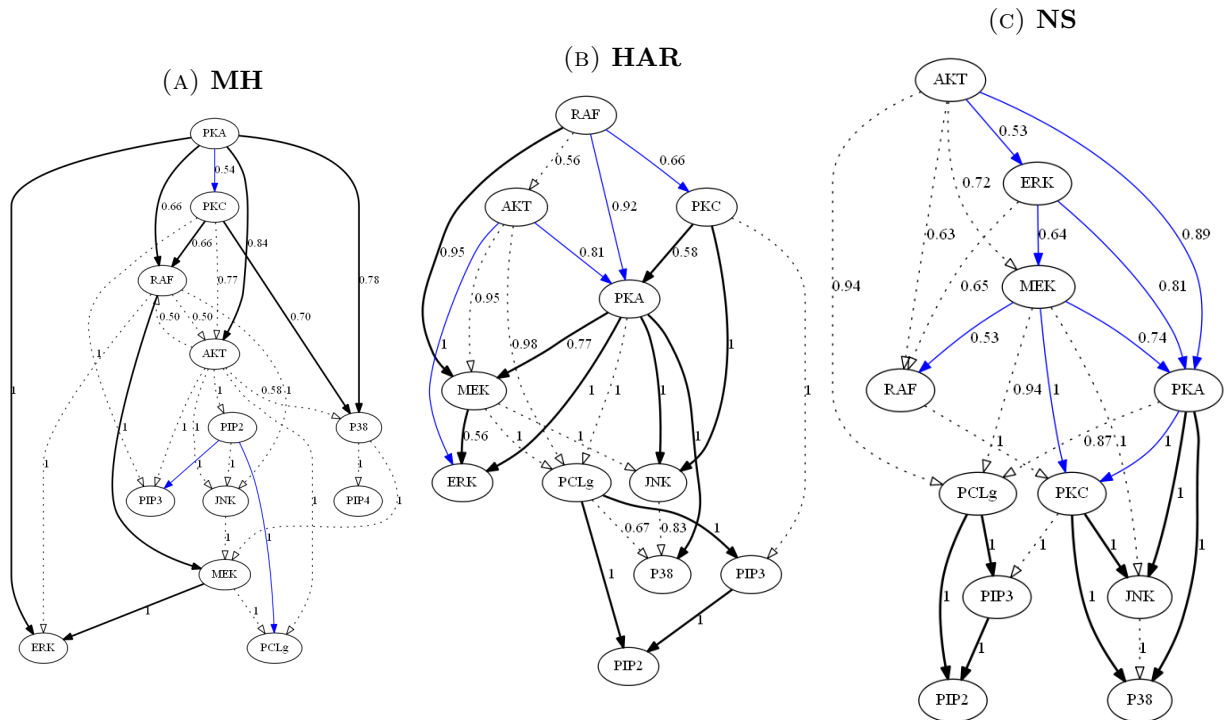


FIGURE 7.25: Posterior means  $> 50\%$  of the 6th chain. Black, blue and dashed edges highlight the true, reversed and new edges, respectively.

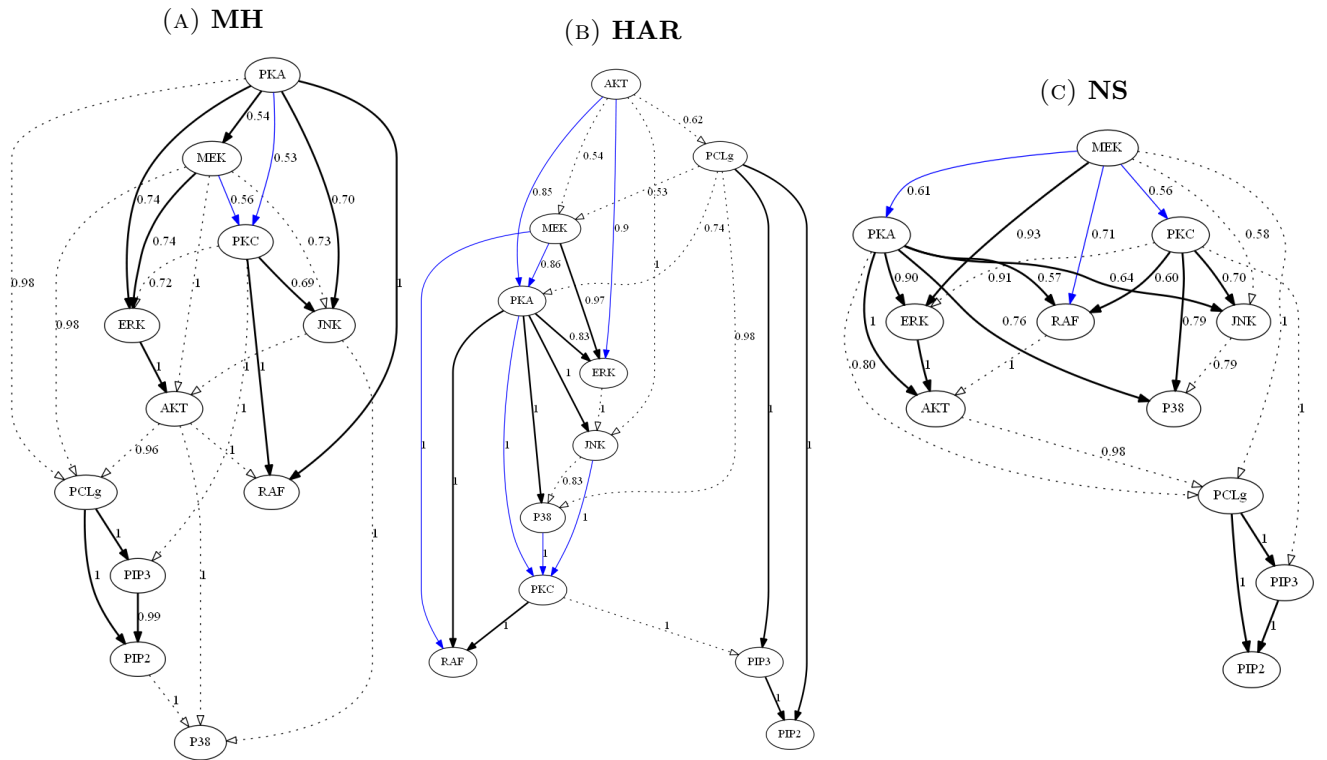


FIGURE 7.26: Posterior means > 50% of the 7th chain. Black, blue and dashed edges highlight the true, reversed and new edges, respectively.

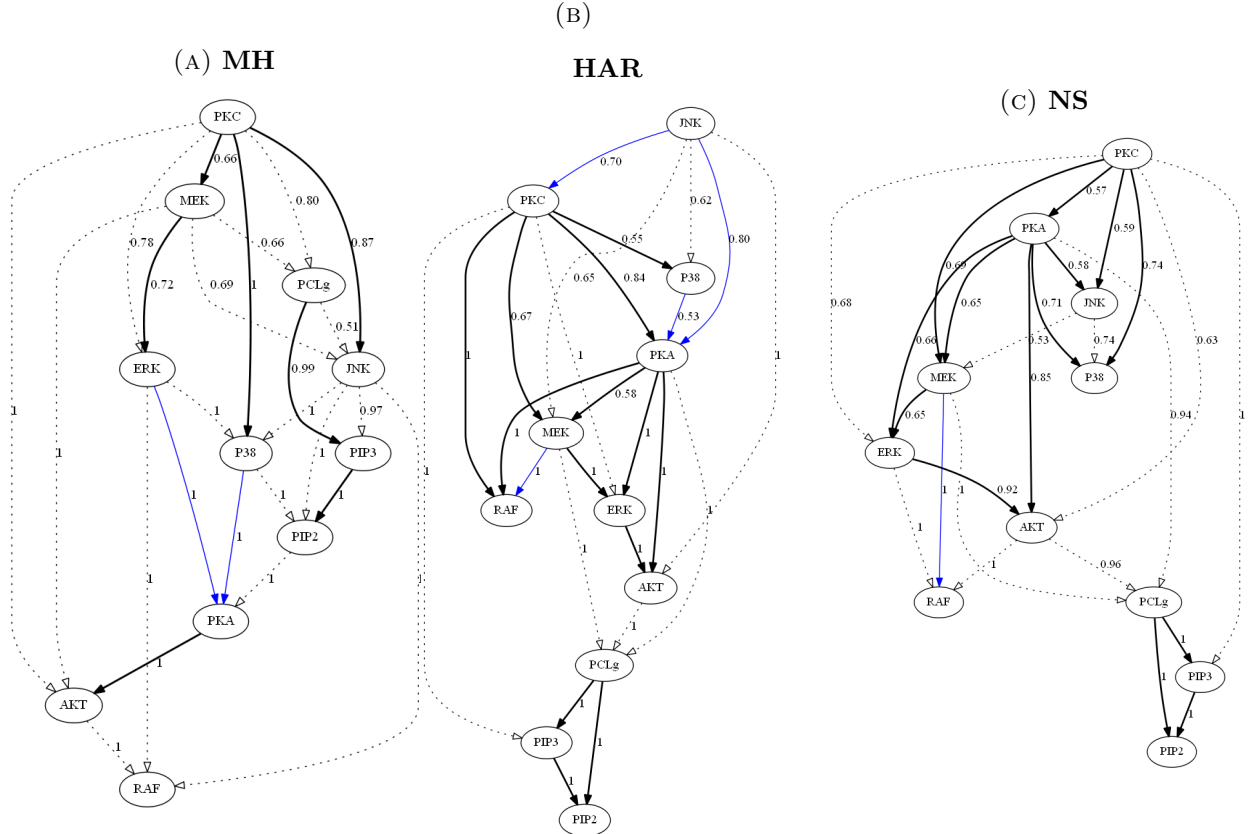


FIGURE 7.27: Posterior means > 50% of the 8th chain. Black, blue and dashed edges highlight the true, reversed and new edges, respectively.

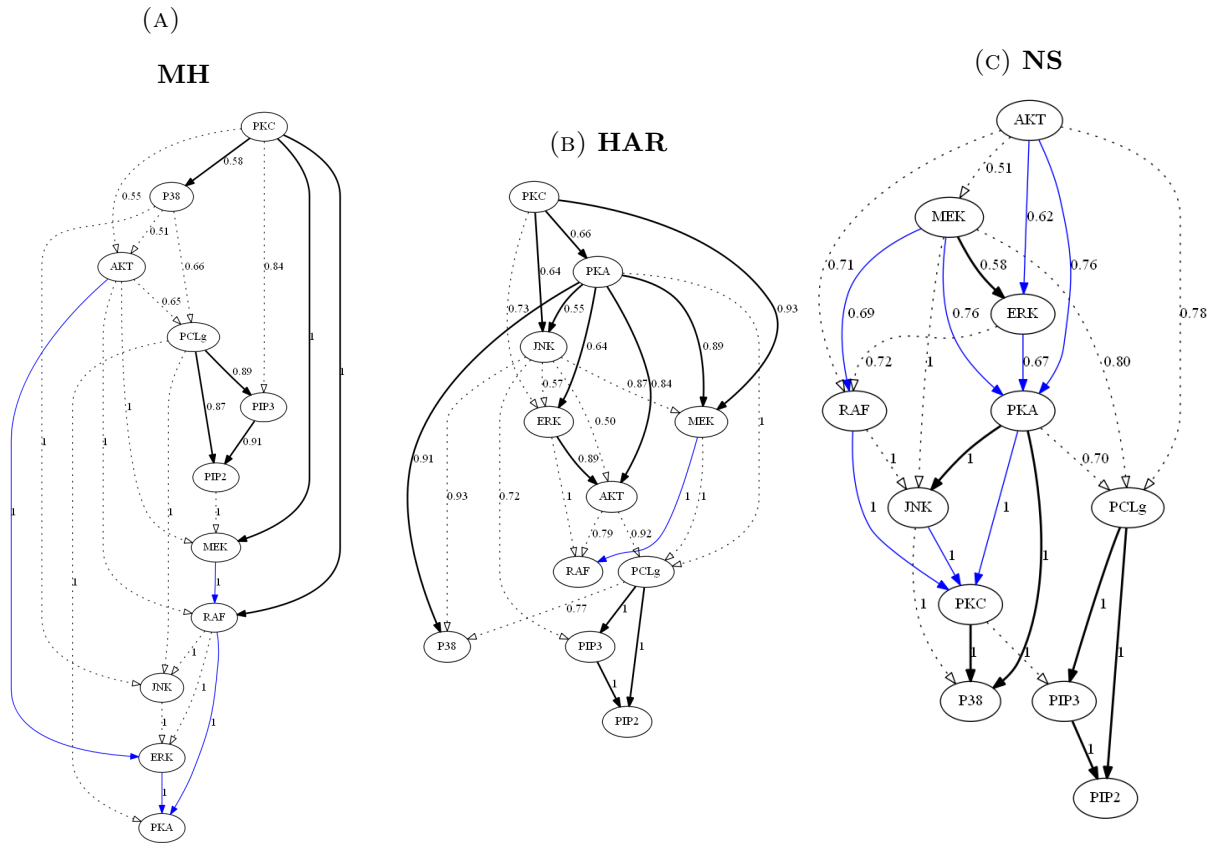


FIGURE 7.28: Posterior means > 50% of the 9th chain. Black, blue and dashed edges highlight the true, reversed and new edges, respectively.

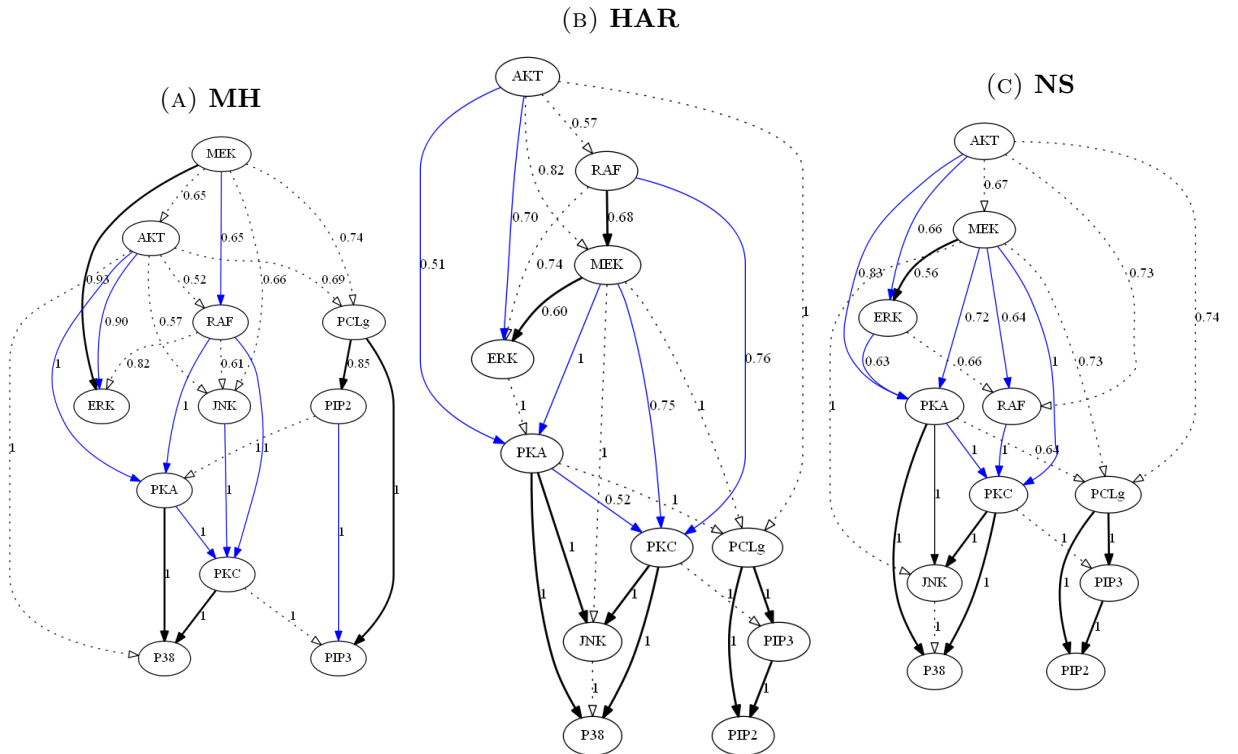


FIGURE 7.29: Posterior means > 50% of the 10th chain. Black, blue and dashed edges highlight the true, reversed and new edges, respectively.

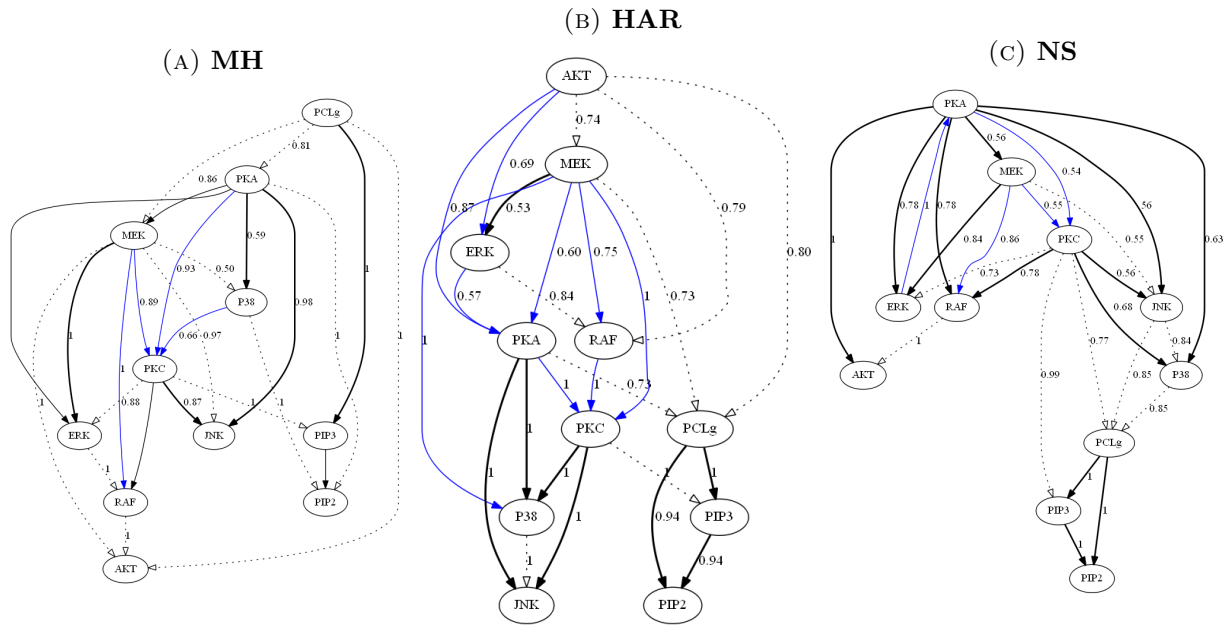


FIGURE 7.30: Posterior means  $> 50\%$  of the 11th chain. Black, blue and dashed edges highlight the true, reversed and new edges, respectively.

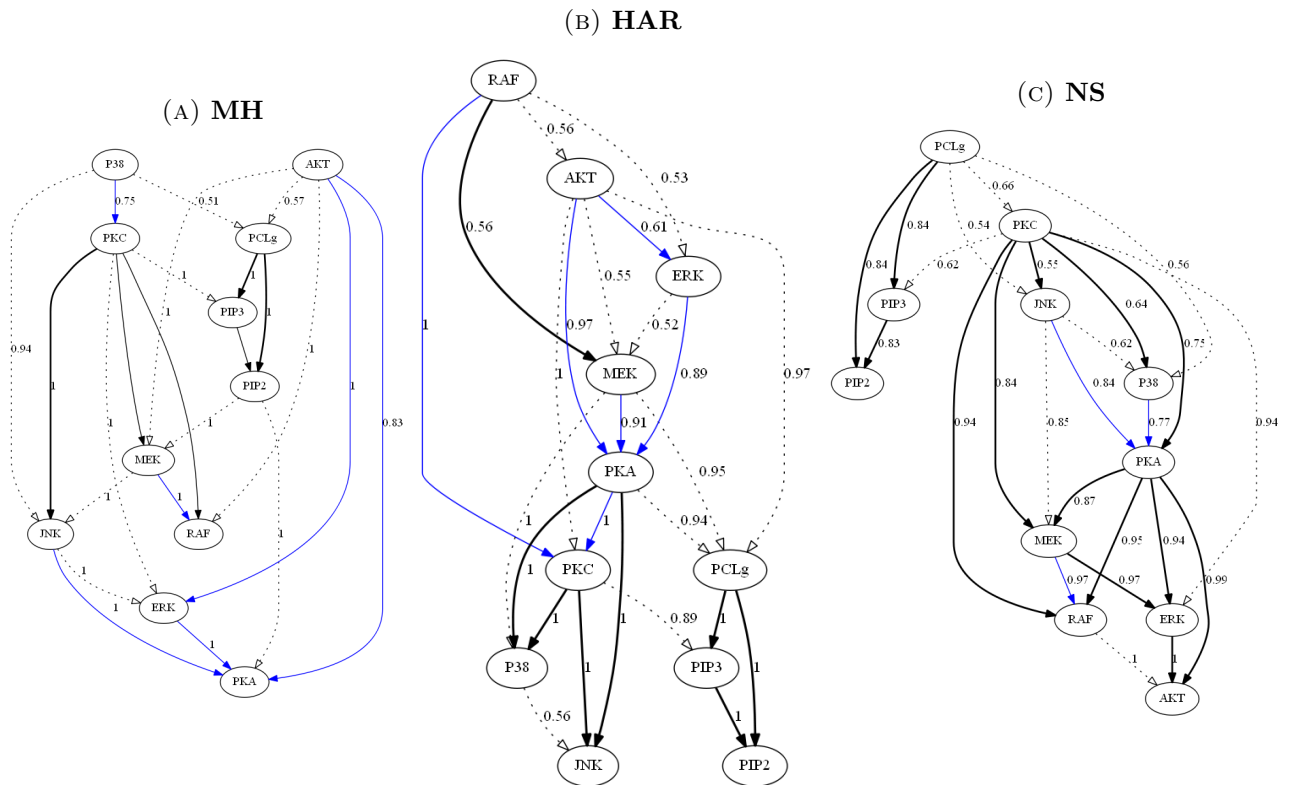


FIGURE 7.31: Posterior means  $> 50\%$  of the 12th chain. Black, blue and dashed edges highlight the true, reversed and new edges, respectively.

Note that each chain in Figures 7.20 - 7.31 apparently has a slightly different distribution. This is more likely due to: 1) the true convergence has not occurred, or

2) more samples should be generated, or 3) the data are not sufficiently informative. Table 7.1 counts, for each inferred structure in Figures 7.20 - 7.31, the number of true inferred edges, reversed true edges, true underlying edges where the directions are ignored i.e. the sum of true edges and reversed true edges, and new inferred edges at a threshold of  $> 50\%$ . I note that, out of the 17 true directed edges in Figure 7.16b, the NS and HAR have properly, on average, predicted 10 directed edges and shown better exploration of true directed edges compared to the MH that, on average, has returned 6 true directed edges. In regards to the number of reversed true edges, the three MCMC samplers have returned roughly the same number which is, on average, equal to 5. I also investigate the number of true predicted edges in the underlying structures by summing the number of true directed edges and the number of reversed true edges. The NS and HAR, on average, predicted 15 out of 17 true underlying edges and have precisely inferred the true underlying structure in two chains. The MH, on average, returned 11 out of 17 true underlying edges, and always has the lowest number of true edges compared to the NS and HAR.

	True directed edges			Reversed true edges			Underlying true edges			New directed edges			Total sampled edges		
	MH	NS	HAR	MH	NS	HAR	MH	NS	HAR	MH	NS	HAR	MH	NS	HAR
chain 1	7	13	12	5	3	2	12	16	14	13	9	7	25	25	21
chain 2	8	13	10	5	2	4	13	15	14	13	9	9	26	24	23
chain 3	3	6	9	7	6	7	10	12	16	13	12	7	23	24	23
chain 4	6	6	9	6	6	7	12	12	16	12	12	9	24	24	25
chain 5	4	7	7	6	8	9	10	15	16	14	10	9	24	25	25
chain 6	8	7	11	3	8	4	11	15	15	14	10	9	25	25	24
chain 7	10	13	9	2	3	6	12	16	15	12	8	9	24	24	24
chain 8	7	14	13	2	1	4	9	15	17	15	10	8	24	25	25
chain 9	6	7	12	4	8	1	10	15	13	14	10	12	24	25	25
chain 10	5	8	9	8	8	6	13	16	15	11	9	10	24	25	25
chain 11	9	13	8	4	4	9	13	17	17	12	7	8	25	24	25
chain 12	6	14	8	6	3	6	12	17	14	11	8	11	23	25	25
Average	6.6	10.1	9.8	4.8	5	5.4	11.4	15.1	15.2	12.8	9.5	9	24.25	24.58	24.16

TABLE 7.1: Quantitative summary of Raf-Signaling Pathway structures inferred by MCMC samplers.



### 7.4.3 Conclusion

The experimental simulations in this application have demonstrated the following:

1. The chains obtained by the MH sampler are often stuck at local modes.
2. Increasing the number of iterations with the MH sampler does not help avoid the problem of local modes in this example.
3. All chains obtained by the NS have early converged to a single common mode which has returned a greater posterior probability value than by the MH.
4. All chains obtained by the HAR sampler have returned the same common mode as the NS, however, the convergence to the common mode has occurred a bit late with some chains.
5. The NS and HAR have learned the structure of the Raf-Signaling Pathway better than the MH by inferring more proper directed and undirected edges that link pairs of nodes properly regardless of the direction of edges.
6. For all samplers, increasing the number of iterations does not typically guarantee inferring more directed true edges, it may rather infer the true underlying structure (where some nodes are linked properly but in the opposite direction), and that is due to the fact that the number of equivalent graphs in the space of 11 nodes is very large.

## 7.5 Inferring the KEGG Pathways network

### 7.5.1 Background

A recently inferred Bayesian network based on Kyoto Encyclopedia of Genes and Genomes (KEGG) pathways was studied in [159], which used the Greedy Equivalence Search (GES) algorithm [160] to learn interactions in cellular pathways. The

GES identifies a network with a higher scoring function compared to several other algorithms. The details of the computational steps of GES can be found in [161]. The KEGG Pathways network inferred in [159] used the True Link Strength Percentage (TLSP) [107] to measure the connection strengths of links in the Bayesian network. The TLSP divides the reduction in entropy of the child node given the parent node by the original entropy of the child node [107].

A total of 163 DNA microarray data sets were obtained from different sources outlined in [159]. The one-sample Kolmogorov-Smirnov test (KS-test) was used to determine whether the observed values are significantly different from a distribution with a zero mean. If the mean value was  $> 0$  or  $< 0$ , representing an up or down regulation, the gene is then assigned  $+1$  or  $-1$ , respectively. Otherwise, the gene is set to 0 to indicate that it does not respond to signalling effects - a means of gene regulation (or it is not differentially expressed).

The numbers of differentially expressed genes in individual pathways was considered to select 51 pathways for further study. The 51 KEGG pathways selected in [159] show significant up or down regulation across different experimental conditions.

### 7.5.2 Learning initial networks

To efficiently learn a Bayesian network for a large number of nodes, it is highly recommended to start with a good initial network rather than starting from a random initial network that might be extremely distant from the optimal network. Practically, it is possible to learn the initial network by applying a heuristic algorithm that quickly returns a Bayesian network with a high scoring function. I implemented the Hill-Climbing search (HCS) - a well-known heuristic search - to learn the Bayesian network of KEGG pathways. Also, I use the Bayesian network inferred by the GES in [159] as an initial network. The two BNs learned by the GES and HCS are plotted in Figure 7.32a and Figure 7.32b, respectively.

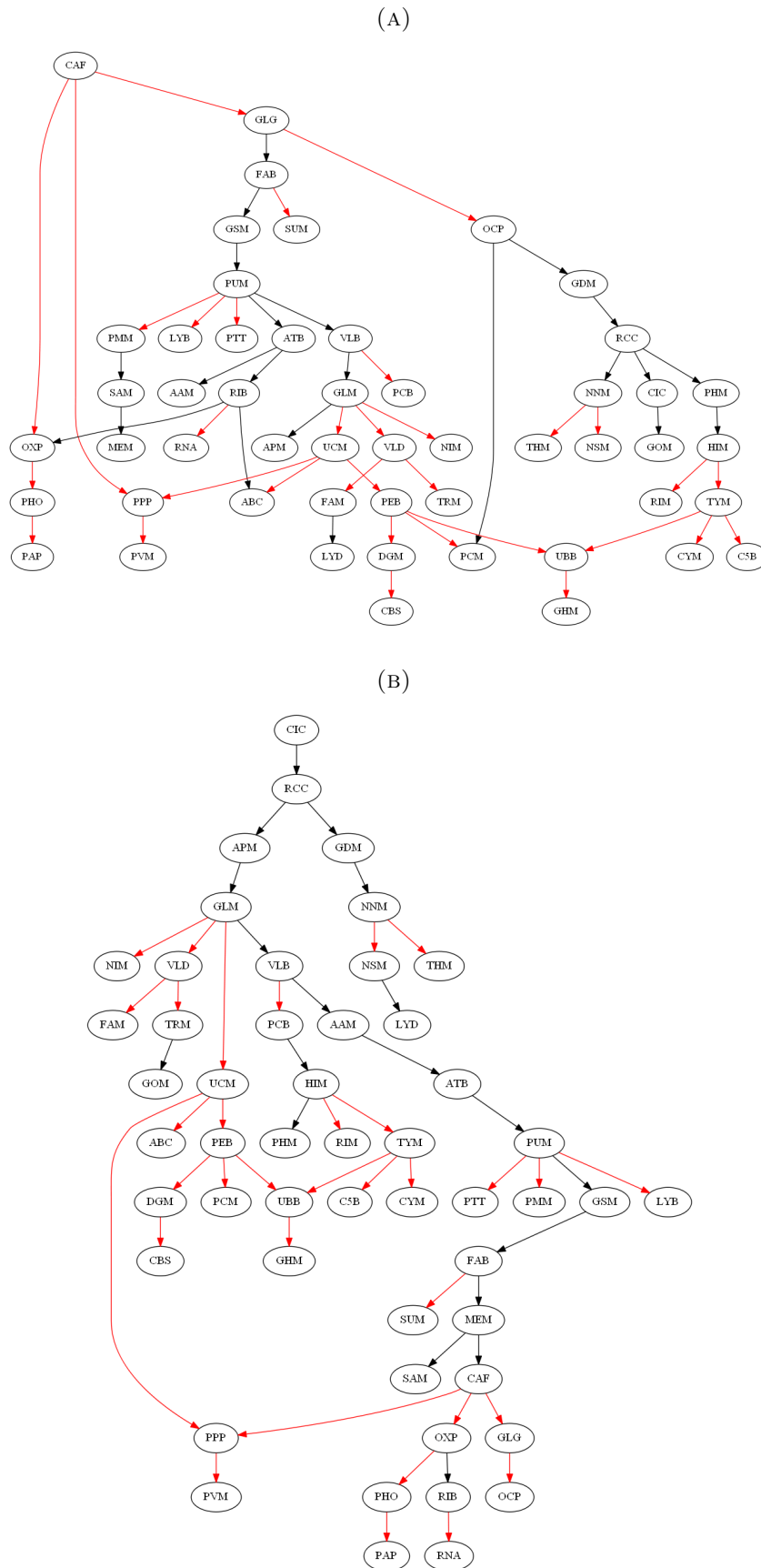


FIGURE 7.32: (A) BN inferred by the GES. (B) BN inferred by the HCS. Red edges highlight the common edges between the two inferred networks.

The common edges between the two inferred networks are colored in red. The log scoring functions for the two networks predicted by the GES and HCS are -2517 and -2481, respectively. I examine whether MCMC samplers are able to improve their scoring functions.

### 7.5.3 Experimental results

I examine the potential of the MH, HAR and NS to find a better scoring network over the GES and HCS. The performance of samplers in this application was investigated experimentally based on: 1) the same number of iterations in Section 7.5.3.1, and 2) the same amount of computation time in Section 7.5.3.2. The maximum number of in-degree and out-degree for each node in the networks learned by the GES and HCS are two and five nodes, respectively. The latter two numbers are used accordingly in this application to restrict the maximum number of in-degree and out-degree for each node. I ran the MH, HAR and NS using the same dataset published in [159] for the 51 pathways of genes.

#### 7.5.3.1 Comparing the performance of MH, NS and HAR based on the same number of iterations

This section runs two Markov chains of 20,000 iterations for each of the three samplers. The initial networks of these two chains were fixed to the networks inferred by the GES and HCS shown in Figure 7.32a and Figure 7.32b. Figure 7.33 plots all the log posterior values produced by the MH, HAR and NS. Figure 7.33 demonstrates that the three MCMC samplers have the potential to improve the scoring functions of the networks learned by the GES and HCS. However, the NS, in particular, has demonstrated an earlier convergence behavior, compared to the MH sampler and HAR sampler. The latter has performed better than the MH sampler, but it is still not converged. Table 7.2 compares the highest log scoring functions produced by the MCMC samplers.

	MH	HAR	NS
<b>GES (-2517)</b>	-1913	-1517	-1287
<b>HCS (-2481)</b>	-1936	-1532	-1256

TABLE 7.2: The highest log scoring function produced by MCMC samplers when the networks of KEGG learned by the GES and HCS are used as initial networks.

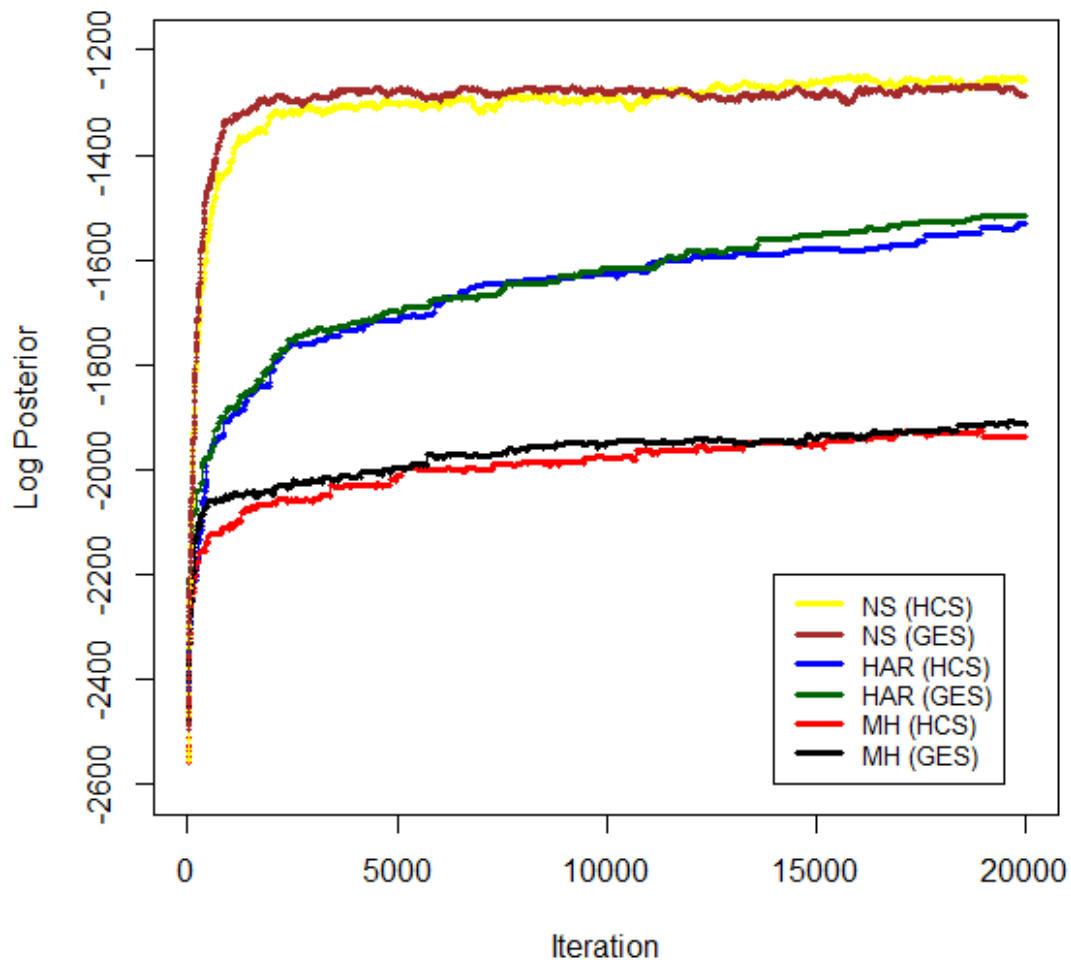


FIGURE 7.33: MCMC samplers' performances to learn KEGG Pathways network: The log posterior values versus 20,000 iterations produced by the MH sampler, HAR sampler, and NS, when the initial networks are produced by the GES and HCS.

To attempt to achieve convergence of the MH sampler and HAR sampler, I ran the two samplers for additional iterations. I therefore ran two Markov chains with 500,000 iterations each using the HAR and MH samplers. Figure 7.34 shows the log posterior values produced with the MH and HAR samplers. It is noted that the log likelihoods for the two chains produced by the HAR sampler have been improved to become roughly similar to that of the NS in Figure 7.33. The two Markov chains produced by the MH sampler did not achieve log likelihoods similar to the NS or HAR, and thus running an even longer chain is still required.

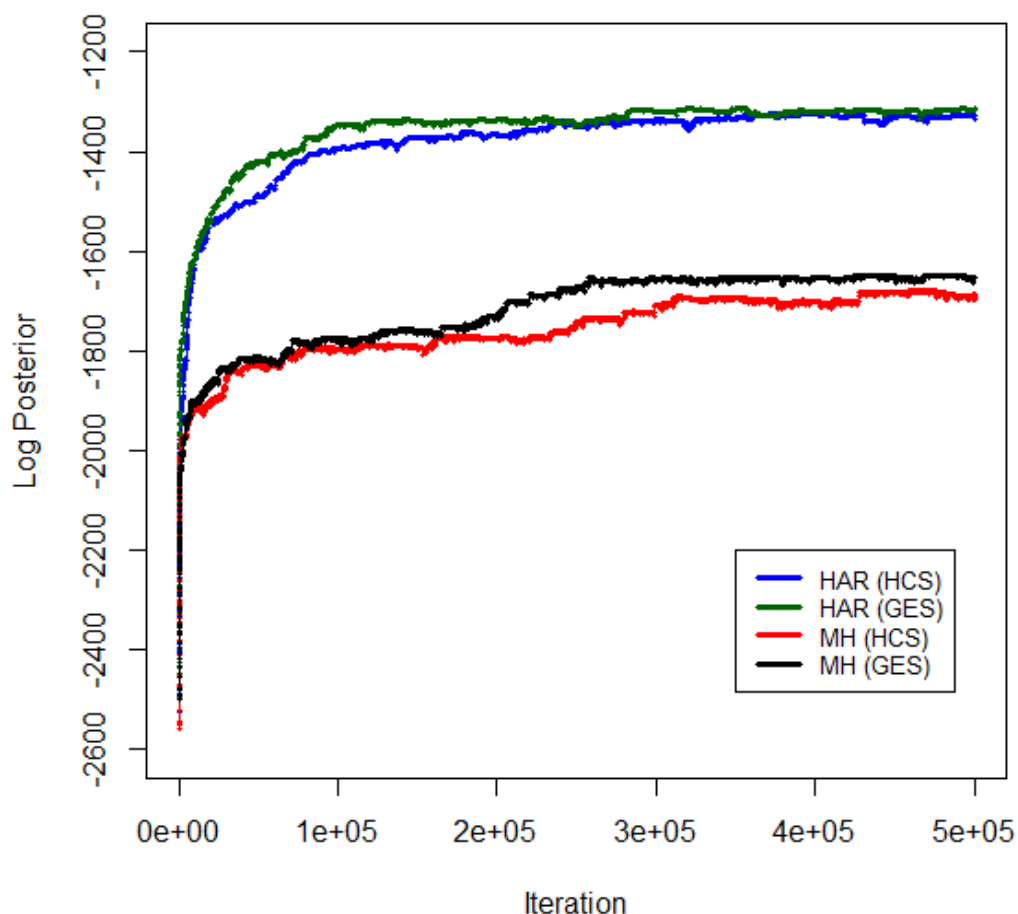


FIGURE 7.34: The log posterior values versus 500,000 iterations produced by the MH and HAR samplers, when the initial networks are produced by the GES and HCS.

**Remark 11.** If one run the three MCMC samplers for a very long time, they are more likely to meet at a common distribution, and the posterior edge probabilities could then be calculated. However, here I aim only to prove the ability of MCMC samplers to enhance the scoring function of the networks learned by heuristic samplers such as the GES and HCS.

In Section C.5.1, I ran another two Markov chains using the MH sampler with one million iterations each. I set the initial networks of the first and second chains to the networks learned by the GES and HCS, respectively. Figure C.13 in Section C.5.1 plots the log posterior values produced with the MH sampler against one million iteration. This result suggests that one million iteration is still not enough for the MH sampler to efficiently reach high probability region, as the NS and HAR.

**7.5.3.1.1 Posterior edge probabilities** To estimate the posterior edge probabilities, I consider only the NS and HAR sampler, since MH is far from convergence. I applied burn-in intervals of 2500 iterations and 100,000 iteration based on log posterior chains produced by the NS and HAR, respectively. Then, I estimate the posterior edge probabilities for each Markov chain separately. I only considered the sampled edges with posterior probabilities greater than or equal to 50%. Figure 7.35 and Figure 7.36 plot the structures inferred by the HAR. Figure 7.37 and Figure 7.38 plot the structures inferred by the NS. I used three different coloured edges: black, red and blue to identify the new inferred edges, unchanged edges, and reversed edges, respectively, in comparison to their initial networks. The probability of each directed edge is indicated near each corresponding directed edge in all Figures.





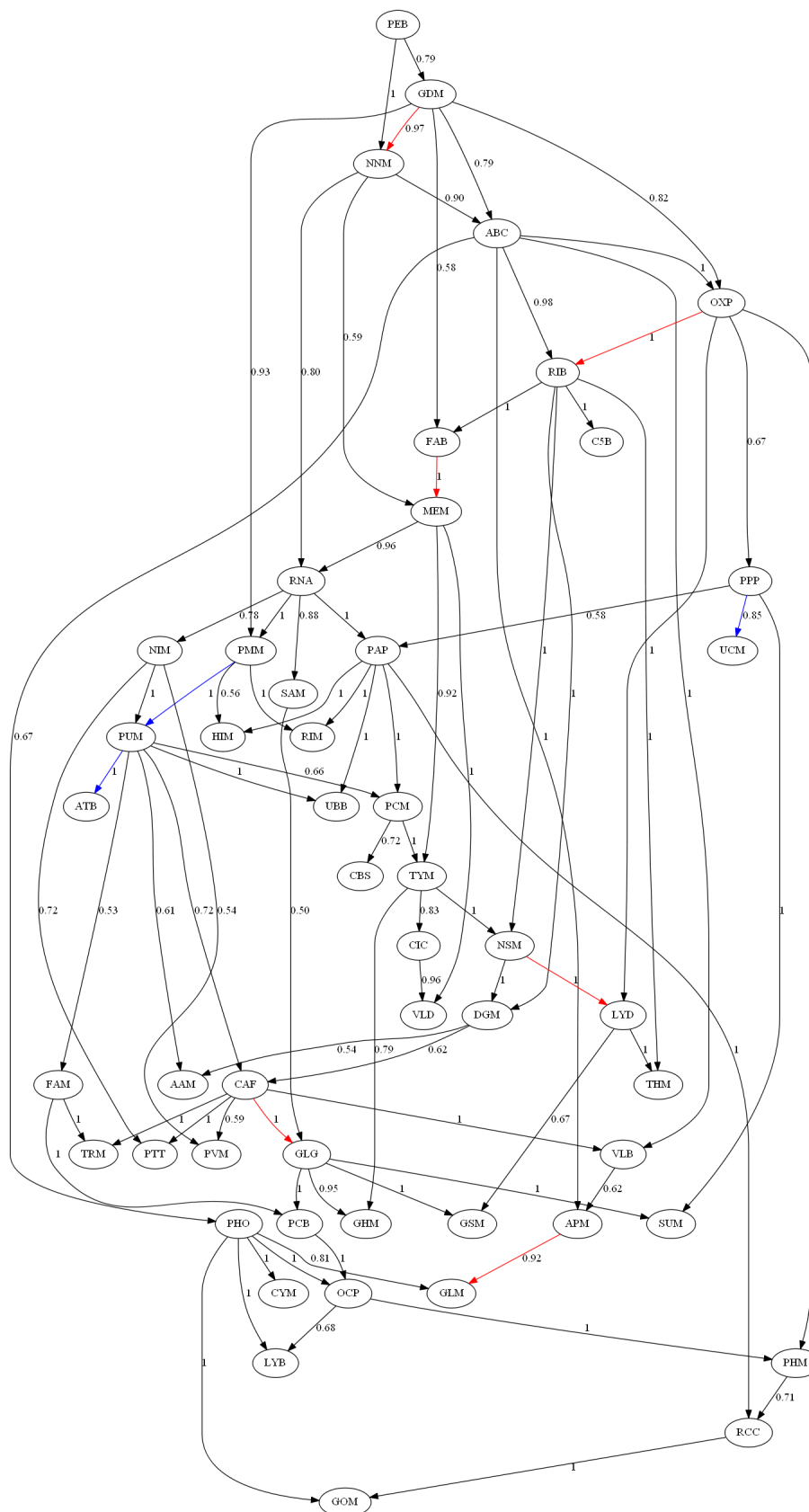


FIGURE 7.36: Bayesian network inferred by the HAR sampler when the initial network is inferred by the HCS. The inferred network represents the posterior edge probabilities  $\geq 50\%$  after running 500,000 iterations and applying a burn-in interval of 100,000 iterations. The black, red and blue edges are used to identify the new inferred edges, unchanged edges, and reversed edges, respectively, in comparison to their initial networks.

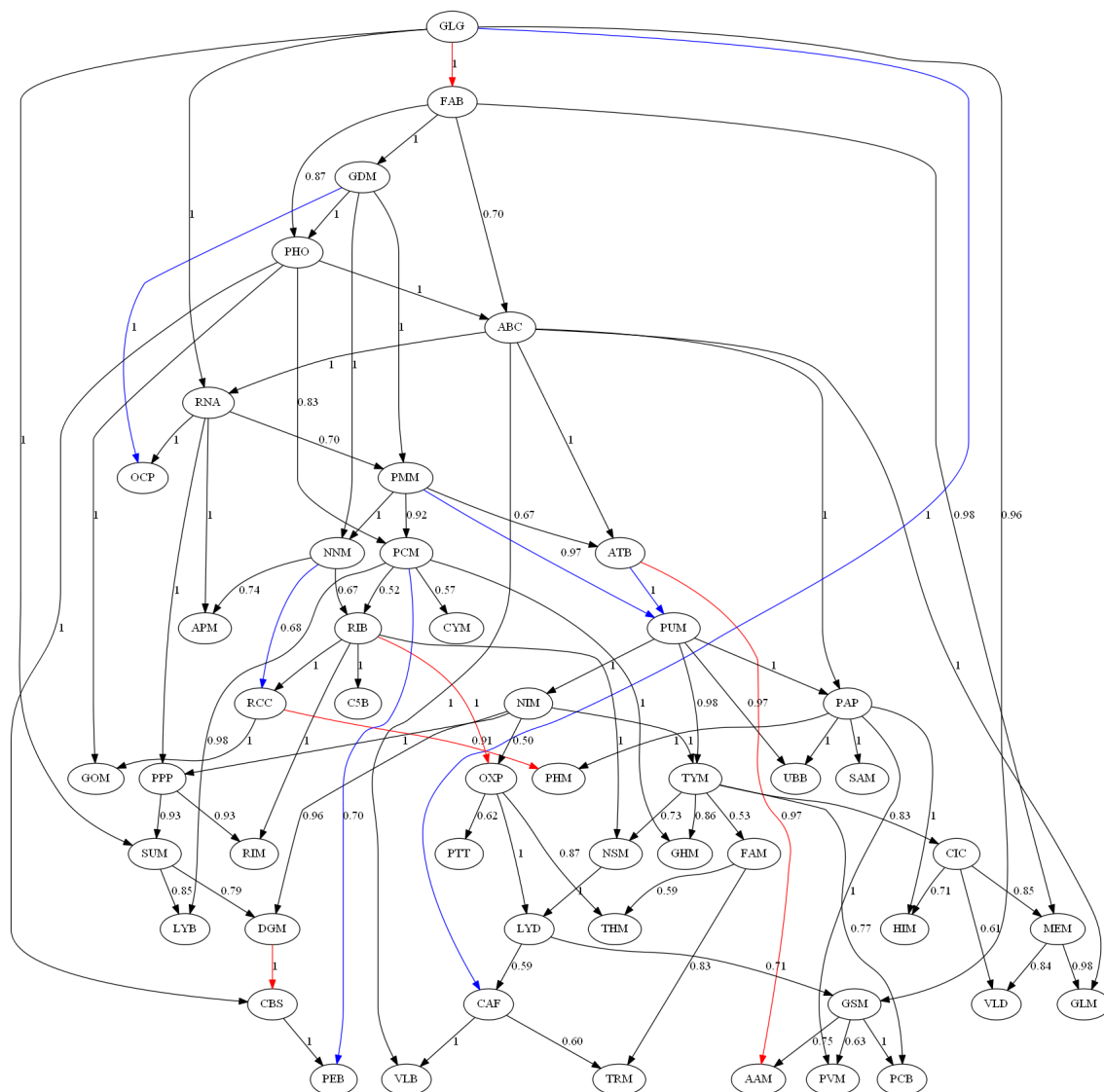


FIGURE 7.37: Bayesian network inferred by the NS when the initial network is inferred by the GES. The inferred network represents the posterior edge probabilities  $\geq 50\%$  after running 20,000 iterations and applying a burn-in interval of 2500 iterations. The black, red and blue edges are used to identify the new inferred edges, unchanged edges, and reversed edges, respectively, in comparison to their initial networks.

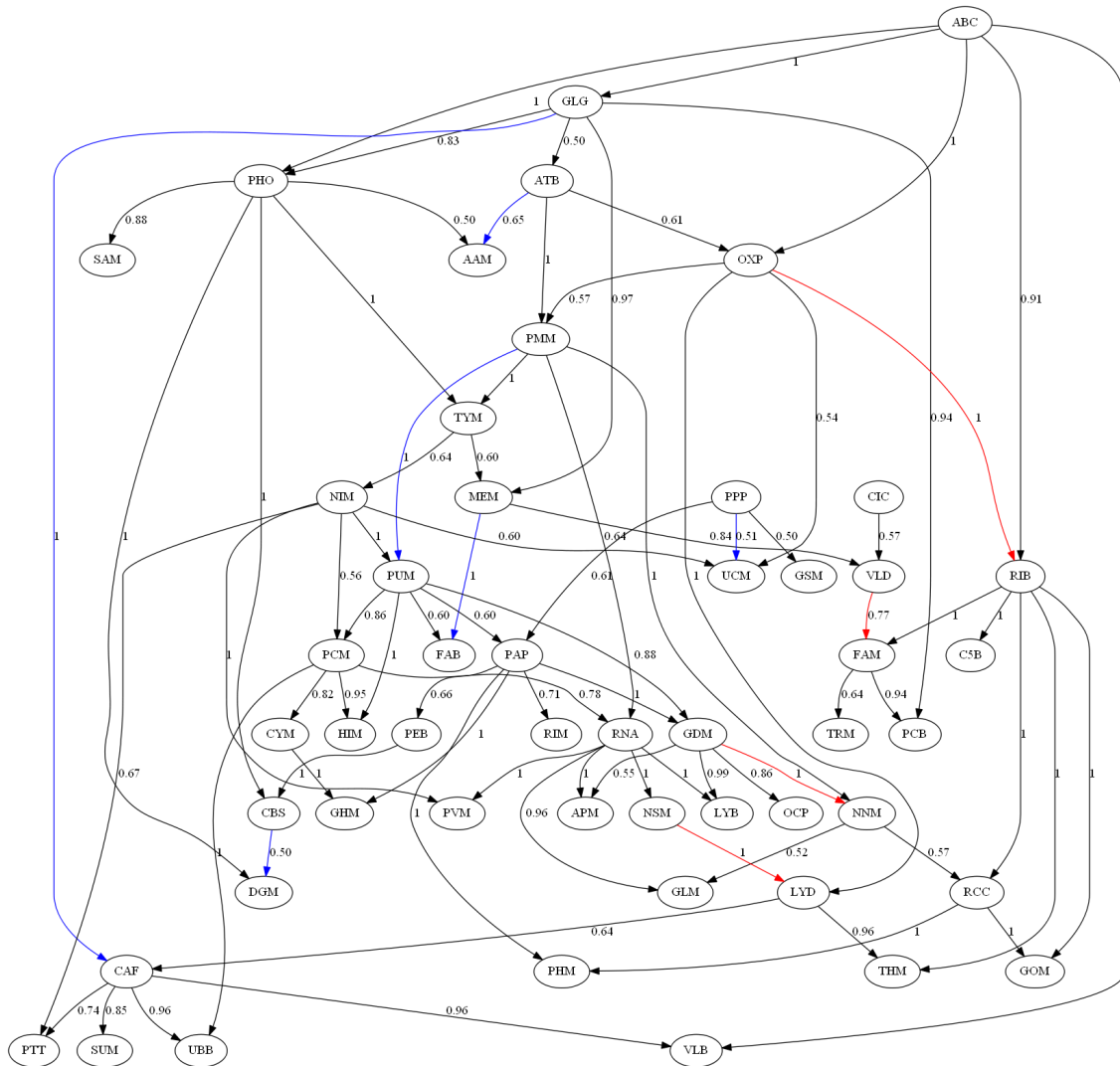


FIGURE 7.38: Bayesian network inferred by the NS when the initial network is inferred by the HCS. The inferred network represents the posterior edge probabilities  $\geq 50\%$  after running 20,000 iterations and applying a burn-in interval of 2500 iterations. The black, red and blue edges are used to identify the new inferred edges, unchanged edges, and reversed edges, respectively, in comparison to their initial networks.

All posterior edge probabilities of Figure 7.35, Figure 7.36, Figure 7.37, and Figure 7.38, including probabilities less than 50%, are provided in Section C.5.2.

Table 7.3 describes the changes that occurred to the structures of the initial networks learned by the GES and HCS after applying the NS and HAR using 20,000 iterations and 500,000 iterations, respectively. Note that the structures of the two initial networks have been significantly changed by the NS and HAR. This is consistent with the fact that the log scoring functions of the two initial networks have

been dramatically improved by the NS and HAR.

	HAR				NS			
	SE	UE	RE	NE	SE	UE	SE	NE
<b>GES</b>	91	7	4	80	89	5	6	78
<b>HCS</b>	88	6	3	79	84	4	6	74

TABLE 7.3: For each of the NS and HAR sampler, the SE, UE, RE, and NE stand for: the number of sampled edges that show posterior probabilities  $\geq 50\%$ , the number of unchanged edges, the number of reversed edges, and the number of new inferred edges, compared to the initial networks.

This application of 51 nodes demonstrates that the NS and HAR can substantial improve upon the scoring function produced by other heuristic algorithms e.g. the GES and HCS.

**7.5.3.1.2 Best scoring network** Note that the NS and HAR have produced very different networks for different starting points. One main key difference among networks is that most of their inferred edges do not link the same nodes, even if their convergence appear to have occurred. The goal here is to improve on other algorithms, which has been achieved. As a consequence, the best scoring network found may be more useful than the posterior edge probabilities for this example. Figure C.14, Figure C.15, Figure C.17, and Figure C.16 in Section C.5.2 plot the networks with best scoring values produced by the HAR (GES), HAR (HCS), NS (GES), and NS (HCS), respectively. To summarise the common edges among the best networks, I consider every edge that appears in at least two networks. If an edge appears in the four networks, I give it a strength of 4, if it appears in three networks, I give it a strength of 3, if it appears in two networks, I give it a strength of 2, and if it appears in one network, I ignore it. I end up with 86 edges. Three edges out of the 86 edges have a strength of 4. Further, 23 edges out of the 86 edges

have a strength of 3. And, 60 edges out of the 86 edges have a strength of 2. Figure 7.39 plots all the 86 edges, and the strength of each edge is determined near the corresponding edge.

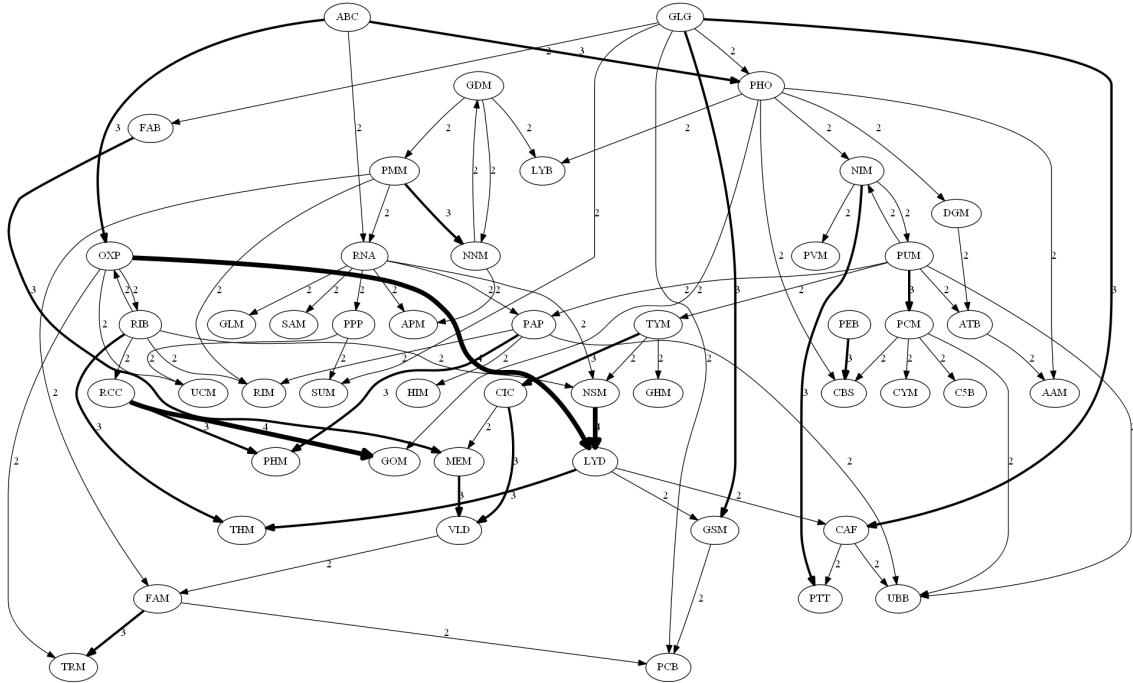


FIGURE 7.39: All edges that appear at least twice among the four best scoring networks. There are three styles of directed edges: double bold, bold, and normal to indicate the edges with strength of four, three, and two, respectively.

### 7.5.3.2 Comparing the MH, NS and HAR based on time elapsed

This section compares the performance of the MH, NS and HAR to infer the KEGG Pathways network based on the same amount of computation time, rather than the number of MCMC iterations that equally fixed. Two Markov chains have been run for each of the three MCMC samplers. The two initial networks for the first and second Markov chains have been set to the two networks learned by the GES and HCS, respectively. The amount of computation time for each of the six chains was set to one hour.

Table 7.4 reports the number of iterations and the highest log posterior value achieved by each chain during one hour of simulation. The highest log posterior

values achieved by the MH, NS and HAR given the initial networks learned by the GES and HCS are  $-1895.90$  and  $-1918.16$ ,  $-1382.29$  and  $-1372.40$ ,  $-1534.43$  and  $-1452.95$ , respectively. The number of iteration run by the MH, NS and HAR given the initial networks learned by the GES and HCS are 36452 and 39075, 998 and 1002, and 19455 and 21005, respectively. These experimental results have shown that when the MH, NS and HAR are run at the same amount of computation time, the NS and HAR would have the potential to sample BNs with higher posterior values than the MH.

	MH		NS		HAR	
	N. iteration	Log posterior	N. iteration	Log posterior	N. iteration	Log posterior
chain (GES)	36452	-1895.90	998	-1382.29	19455	-1534.43
chain (HCS)	39075	-1918.16	1002	-1372.40	21005	-1452.95

TABLE 7.4: For each of the MH, NS and HAR, the total number of iterations achieved and the highest posterior score recorded, after a one hour of simulation is completed.

Figure 7.40 plots the log posterior values of sampled BNs against the number of iterations obtained after one hour of simulation using the MH, NS and HAR. Note that simulations in this experiment do not intend to achieve convergence, they rather monitor the improvement in the scoring function values achieved during certain time of simulation by each sampler.

The NS has significantly achieved more rapid convergence rate, higher log posterior values and lower number of iterations than the HAR and MH. The latter has shown to quickly get stuck in a local mode solution, where the HAR sampler could smoothly improve its log posterior values consistently with the NS even though it runs a larger number of iterations.

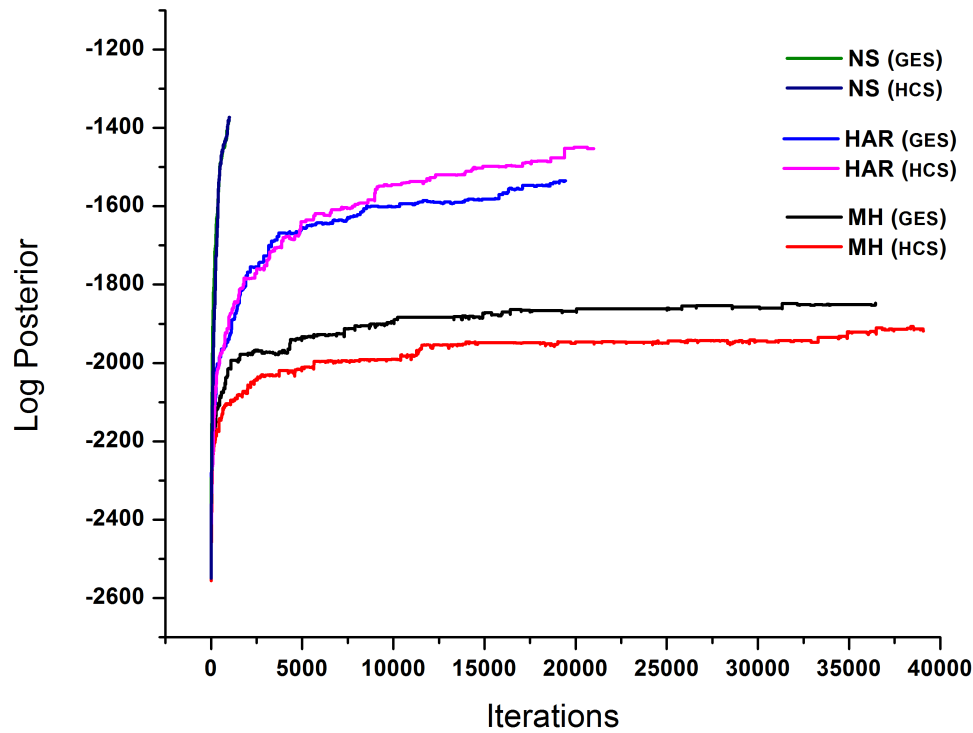


FIGURE 7.40: MCMC samplers' performances to learn KEGG Pathways network for a time elapsed of one hour: The log posterior values versus different number of iterations produced by the MH sampler, HAR sampler, and NS, when the initial networks are produced by the GES and HCS.

### 7.5.4 Conclusion

This example of KEGG pathways inference has demonstrated the following:

1. The HCS and GES are inadequate sampling techniques to provide a robust inference of large BNs.
2. The HCS and GES can provide a solution to generate effective initial networks for medium BNs which would practically accelerate structure inference .
3. The MH, NS and HAR are ideal MCMC techniques to enhance upon the scoring function produced by the GES and HCS.

- 
4. A Bayesian network with highest posterior value has been found to be a useful solution to the problem of structure learning given a certain number of nodes and observational dataset.
  5. It is practically sensible to plot the common edges among several best networks produced by different MCMC samplers in a single network, given that every single common edge is weighted by the number of times it appears among all best networks.
  6. Whether the simulations of MH, NS and HAR are run at the same amount of computation time or at the same number of iterations, the NS was the best in learning the KEGG Pathways network, then the HAR, and lastly the MH sampler.
  7. The inference outputs produced by the MH, NS and HAR never sure that convergence has occurred, but one can see that some obtained models have apparently converged to better solutions, according to the posterior deviance.



## Chapter 8

# *BNMCMC*: A New Graphical User Interface

### 8.1 Introduction

Despite the importance of MCMC methods to infer BNs in various applications from social networks to systems biology, a wide range of relative graphical user interfaces (GUI) do not include MCMC methods. For example, *BNFinder* [162] uses Exact-algorithm, *bnlearn* [157] uses constraint-based (GS, IAMB, Inter-IAMB, Fast-IAMB, MMPC, Hiton-PC), pairwise (ARACNE and Chow-Liu), score-based (Hill-Climbing and Tabu Search) and hybrid (MMHC and RSMAX2), *Bayes Server* [163] uses PC-algorithm, and *GeNie* [164] uses Bayesian Search, PC, Essential Graph Search, Greedy Thick Thinning, Tree Augmented Naive Bayes, Augmented Naive Bayes, Naive Bayes. Also, the MCMC methods that are available in other exist packages of BNs e.g. *Hydra* [165], *Blaise* [166] are mostly limited to a standard MCMC method. Moreover, these packages are used via object-oriented programming language libraries e.g. Java and C++ which require basic programming skills [167].

*BNMCMC* is the acronym for Bayesian network Markov Chain Monte Carlo. It is a new GUI developed in this thesis to facilitate inferring BNs using three MCMC

sampling methods: *Neighbourhood Sampler*, *Hit-and-Run sampler*, and *Metropolis-Hastings sampler*. This interface provides a user-friendly environment with intuitive software design. For each of the samplers, numerical outputs are saved in local files, and graphical outputs are depicted in the result panel. All the input parameters including method and output settings are entered into a separate panel.

Figure 8.1 shows the main desktop window of the *BNMCMC* software.

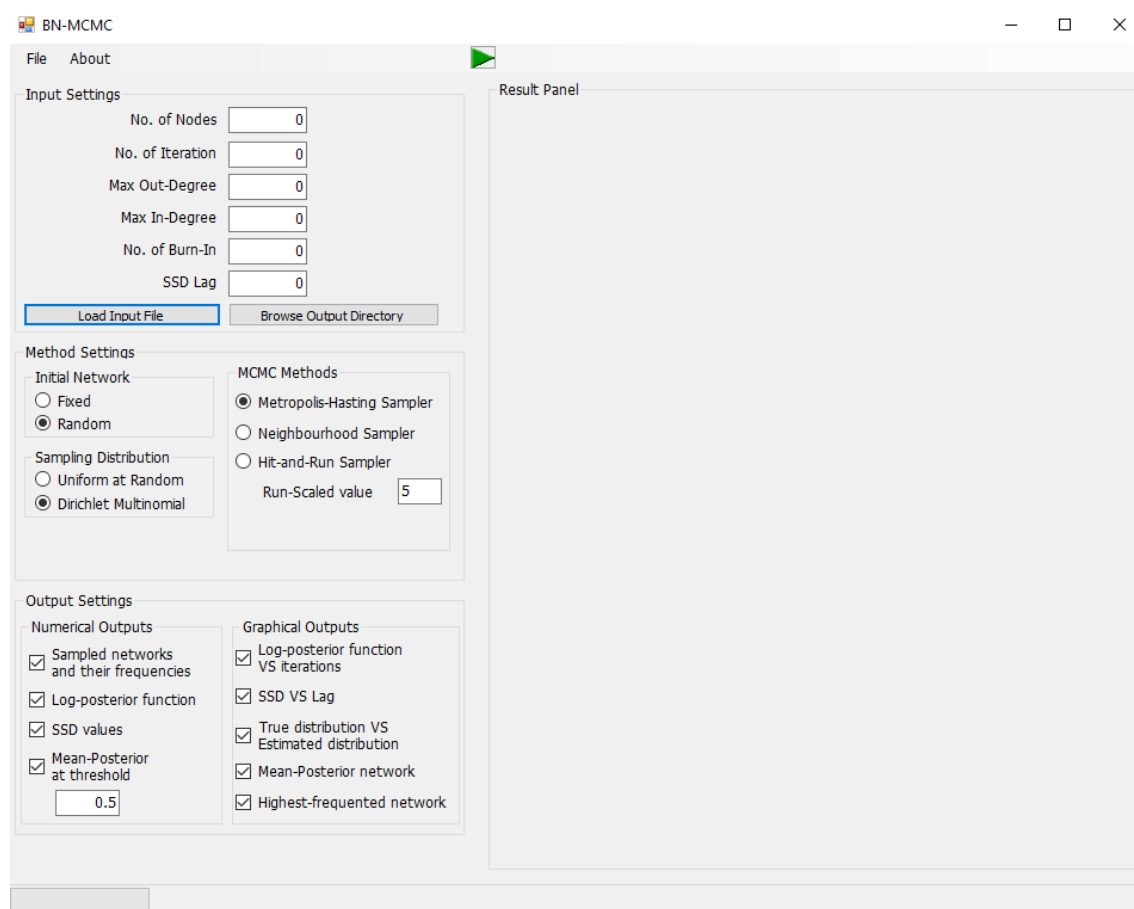


FIGURE 8.1: *BNMCMC* window.

Two practical issues to implement large case studies e.g. Raf signaling include enumerating all possible candidate adjacent graphs and computing the likelihood of a candidate structure during sampling. To address these two challenges, I incorporated two adaptive techniques (adaptive adjacent graphs enumeration and adaptive function scoring) which have been proposed in Chapter 6.

The computing language used to implement the *BNMCMC* package was the C#.NET using Microsoft Visual Studio. As a widely used object-oriented programming language, C#.NET offers an array of advanced data structures and sophisticated coding schemes which are suitable for large-scale software implementations.

The GUI was constructed as a desktop-based Windows application. Although it is primarily targeted for Windows machines, using some wrapper package e.g. Wineskin Winery, it can be run in Mac (Unix-based) machines as well. Note that I did not use any third party library for this software.

This chapter is constructed as follows. Section 8.2 reports the main functions of *BNMCMC* package. Section 8.3 outlines the MCMC samplers available in the *BNMCMC* package. Section 8.4 specifies the Bayesian model used in the *BNMCMC* package to compute the CPTs of variables. Section 8.5 discusses some limitations of the *BNMCMC* package. Section 8.6 contains the user-guidelines to run *BNMCMC*. Section 8.7 is an example guide to show users how to practically use the *BNMCMC*.

## 8.2 Main functions of *BNMCMC*

The *BNMCMC* package undertakes two main functions: it learns BNs structures, and generates BNs uniformly at random. The two functions are discussed below.

### 8.2.1 Variables in *BNMCMC*

The current version of *BNMCMC* package is designed to learn BNs structures from *observed data*. The observed data must satisfy the following criteria:

- All variables must have the same number of observations.
- The variables must have no missing values.
- All observations should be discrete values. Note, continuous values are also applicable once they are discretized.

- Variables in a predefined initial network must be connected by directed edges.

The data-file format required by the *BNMCMC* package is explained in Section 8.6.1.

### 8.2.2 Sampling Bayesian networks uniformly

Sampling BNs uniformly at random does not require observed data. Instead, a Uniform distribution is the target posterior distribution, that is, all graphs in the space are equally likely. Using the *BNMCMC* package, users only need to define the number of nodes and the number of iterations to randomly generate BNs.

## 8.3 Sampling methods in *BNMCMC*

The current version of *BNMCMC* package offers three MCMC samplers: MH, NS and HAR, explained in Chapter 4. The latter sampler requires defining the running-length  $\lambda$  which is initially set by the user. The default setting of  $\lambda$  in the *BNMCMC* package is five.

Each of the three MCMC samplers used in the *BNMCMC* package is run using two adaptive techniques designed in Chapter 6 to reduce the time complexity required to enumerate adjacent graphs, and to compute scoring functions, respectively.

## 8.4 Parameters model in *BNMCMC*

The current version of the *BNMCMC* uses the Dirichlet-Multinomial distribution to learn conditional probabilities among variables within generated BNs. The Dirichlet-Multinomial distribution is a Bayesian model that involves two distributions: the Dirichlet distribution to describe a *priori* knowledge and the Multinomial distribution to describe the observed data likelihood.

## 8.5 Limitations of *BNMCMC*

The *BNMCMC* package may produce slow performance in the following scenarios: large-scale networks, large number of parents for each node, large numbers of observations, or large number of state-values for each node. However, it is highly recommended, where possible, to take advantage of the available restriction settings provided in the *BNMCMC* package. With large networks, it is recommended to start simulation from a high scoring network, perhaps based on prior experience or using heuristics search algorithms as discussed in Chapter 7, Section 7.5.2, rather than an arbitrary random network.

## 8.6 User guidelines

This section explains how to formulate the data files required by *BNMCMC* in Section 8.6.1, and the necessary steps required to run simulations with *BNMCMC* in Section 8.6.2. A modular design for the *BNMCMC* is also diagrammed in Section 8.6.3.

### 8.6.1 Data-file format

The data file is only required if the target distribution is set to the Dirichlet-Multinomial distribution. The *BNMCMC* package reads data from (.txt) format files. The data file must contain the variable names, number of state values for each variable, and observations for each variable. A variable name may contain a combination of letters, numbers, underscores, and dots. Each number of state values in the data file is placed between two underscore lines as follows: "\_2\_". Nominal observations can be coded using integers. For example, we code False and True as "0" and "1" values, respectively. Figure 8.2 shows an example of the accepted file

format. The *BNMCMC* package also provides code to convert the format in Figure 8.3 to the format 8.2.

```

LVFailure    _2_  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  0,  1,
StrokeVolume _3_  1,  0,  1,  2,  0,  0,  1,  1,  2,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
InsuffAnesth _2_  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  0,  0,  1,  0,  0,  0,
SaO2         _3_  0,  0,  1,  1,  0,  1,  0,  1,  2,  1,  1,  1,  2,  1,  1,  1,  1,  1,  1,
ErrCauter    _2_  0,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  0,  0,
VentAlv      _4_  1,  1,  2,  2,  0,  2,  0,  2,  2,  2,  2,  2,  2,  3,  2,  2,  2,  2,  2,  2,
VentLung     _4_  2,  1,  2,  2,  0,  2,  0,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
KinkedTube   _2_  0,  1,  0,  0,  1,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
MinVol       _4_  2,  1,  2,  2,  0,  2,  0,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
PVSat        _3_  0,  0,  1,  1,  0,  1,  0,  1,  2,  1,  1,  1,  2,  1,  1,  1,  1,  1,  1,  1,

```

FIGURE 8.2: Data file format used in *BNMCMC*: From left to right the figure shows: ten variable names, numbers of state values placed between two underscores, and 18 columns of discrete observations.

LVFailure	StrokeVolume	InsuffAnesth	SaO2	ErrCauter	VentAlv	VentLung	KinkedTube	MinVol	PVSat
<u>2</u>	<u>3</u>	<u>2</u>	<u>3</u>	<u>2</u>	<u>4</u>	<u>4</u>	<u>2</u>	<u>4</u>	<u>3</u>
0	1	0	0	0	1	2	0	2	0
0	0	0	0	0	1	1	1	1	0
0	1	0	1	1	2	2	0	2	1
0	2	1	1	0	2	2	0	2	1
0	0	0	0	0	0	0	1	0	0
0	0	0	1	0	2	2	0	2	1
0	1	0	0	0	0	0	1	0	0
0	1	0	1	0	2	2	0	2	1
0	2	0	2	0	2	2	0	2	2
0	1	0	1	0	2	2	0	2	1
1	1	0	1	0	2	2	0	2	1
0	1	0	1	0	2	2	0	2	1
0	1	1	2	0	3	2	0	2	2
0	1	0	1	0	2	2	0	2	1
0	1	0	1	0	2	2	0	2	1
0	1	1	1	0	2	2	0	2	1
0	1	0	1	1	2	2	0	2	1
0	1	0	1	0	2	2	0	2	1
1	1	0	1	0	2	2	0	2	1

FIGURE 8.3: Data file can be converted by *BNMCMC* to the format in Figure 8.2: From top to bottom: ten variable names, numbers of state values placed between two underscores, and 18 rows of discrete observations.

### 8.6.2 Steps to run *BNMCMC*

**Step 1** Load the data file using a (.txt) format by simply pressing the "Load Input File" button shown in Figure 8.1, and then select the file from its directory.

**Step 2** Press the "Browse Output Directory" to choose a directory folder where the output files are saved after they are produced.

**Step 3** Set the simulation inputs and restrictions including the number of nodes, number of iterations, the maximum number of node parents (Max In-Degree), the maximum number of node children (Max Out-Degree), length of burn-in interval, and the lag for sum of squared differences (SSD Lag).

**Step 4** Set the initial network either at random or fixed. Note, when the "Fixed" option is selected, a new window will pop up to enable loading the initial network file. Figure 8.4 illustrates an example of an initial network format consisting of 11 nodes. It is an asymmetric matrix of dimensions  $11 \times 11$ , where ones indicate directed edges. The initial network must be a connected directed acyclic graph. The software checks this condition and produces an error message if not satisfied.

0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0
0,	0,	0,	0,	0,	1,	0,	1,	0,	0,	0
0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0
0,	1,	0,	0,	0,	0,	0,	1,	0,	0,	0
0,	1,	1,	1,	0,	0,	1,	0,	0,	0,	0
1,	0,	1,	0,	0,	0,	0,	0,	0,	0,	0
0,	0,	1,	0,	0,	1,	0,	0,	0,	0,	0
1,	0,	0,	0,	0,	0,	1,	0,	0,	0,	0
1,	1,	0,	1,	1,	0,	0,	0,	0,	0,	1
0,	0,	0,	0,	1,	0,	0,	1,	1,	0,	1
0,	0,	0,	1,	0,	1,	1,	0,	0,	0,	0

FIGURE 8.4: Initial network format describes connectivity among 11 nodes.

**Step 5** Select only one MCMC method. Note, the default "Run-Scaled value" of the HAR sampler is five.

**Step 6** Tick the desired numerical outputs, which may include the log posterior distribution for all sampled graphs, the empirical frequency for each sampled graph, the sum of squared differences versus the lag, and the posterior edge

probabilities of sampled graphs at a particular threshold determined by the user. Note, all numerical outputs are produced in (.csv) format files, and separately saved in the folder selected by the user.

**Step 7** Tick the desired graphical outputs, which may include the log-posterior for each sampled graph against the number of iterations, the sum of squared differences against the lag, the estimated distribution versus the target probabilities for only the sampled graphs, the network structure of the inferred network, and the highest frequency network. Note, the graphical outputs will be shown in the "Result Panel".

**Step 8** Press the green triangle button to run the simulation, then the user will see a short message at the bottom to confirm the simulation has started. When the simulation is completed, another short message will appear to confirm the completion of simulation, and report the total execution time in seconds.

### 8.6.3 Modular design

The diagram in Figure 8.5 illustrates the modular design for the *BNMCMC* software. It shows two groups of components of the software: user interface and *BNMCMC* algorithm. The first group of components is controlled by the user as described by Steps 1-8 in Section 8.6.2, and summarised by the left diagram in Figure 8.5. When all input settings are setup and submitted, then the software performs the process of structural learning by going through the second group of components illustrated by the right diagram in Figure 8.5.



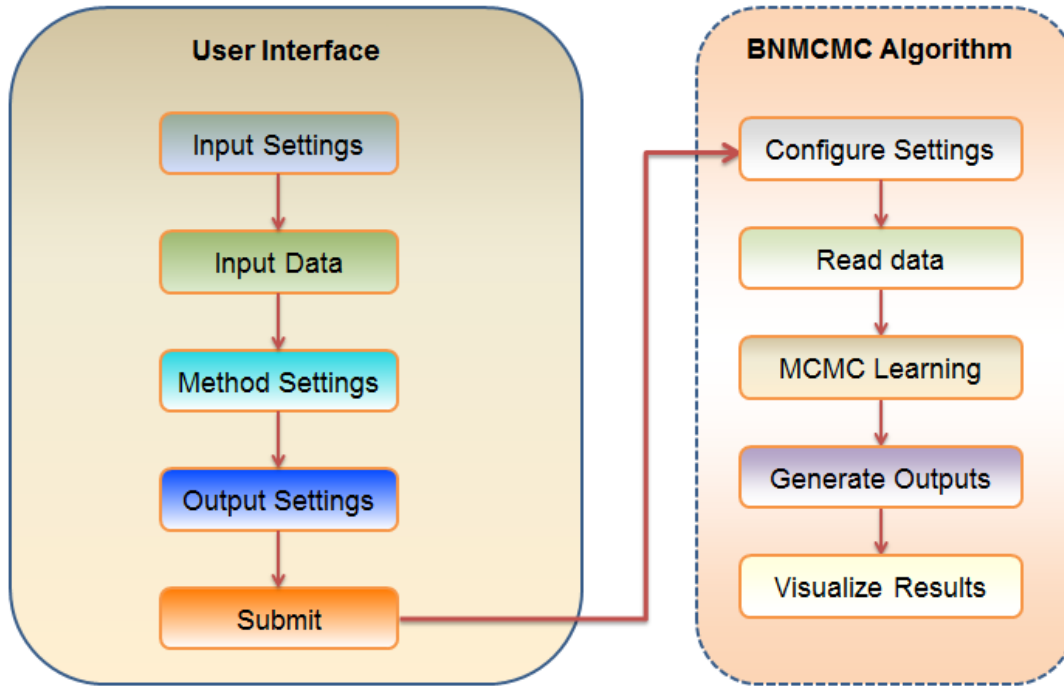


FIGURE 8.5: Diagram illustrates the modular design for the *BNMCMC*.

## 8.7 Illustrative practical example

To learn the structure of Mendel network of six nodes discussed in Chapter 7, first load the data file and select a directory in which to save the output files. All input settings are shown in Figure 8.6. Then, press the green-play button to run the simulation. Note, a short message has appeared at the bottom of Figure 8.6 verifying that the simulation has started. It can be seen in Figure 8.7 that the simulation has now completed and the execution time is 65 seconds. The figure also shows the output plot of log-posterior against number of iterations. It is noted that the log-posterior values demonstrate convergence behavior as the number of iterations increases.

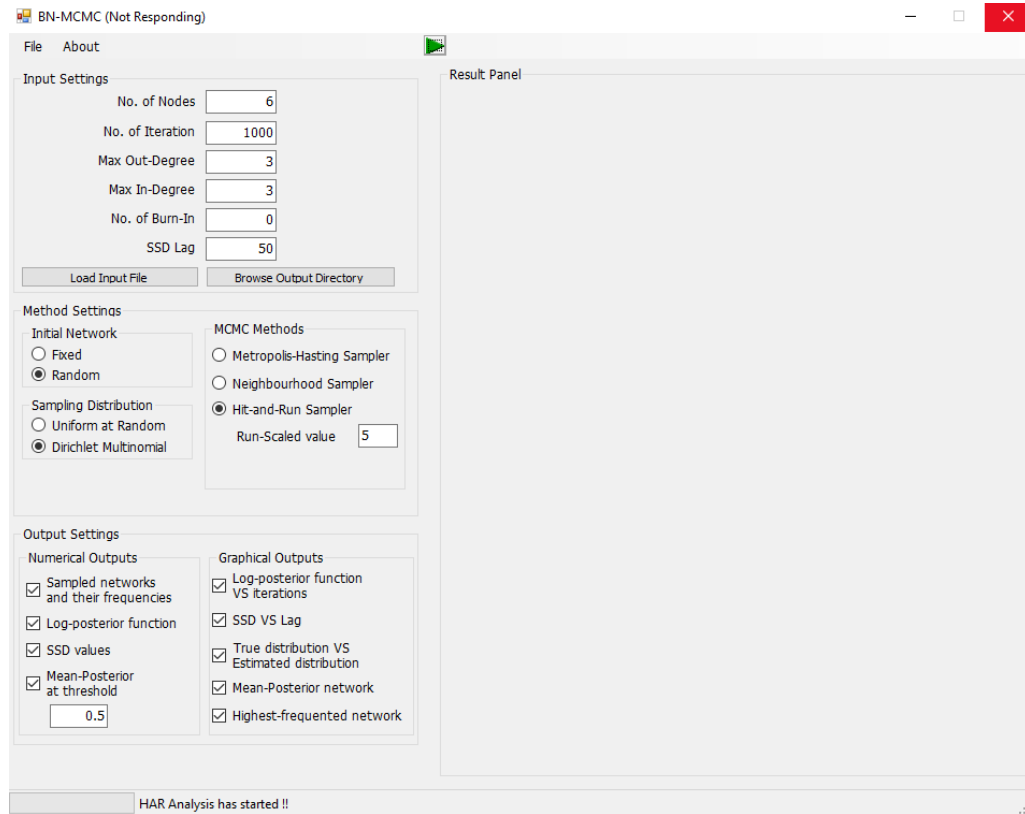


FIGURE 8.6: Input settings for learning Mendel network.

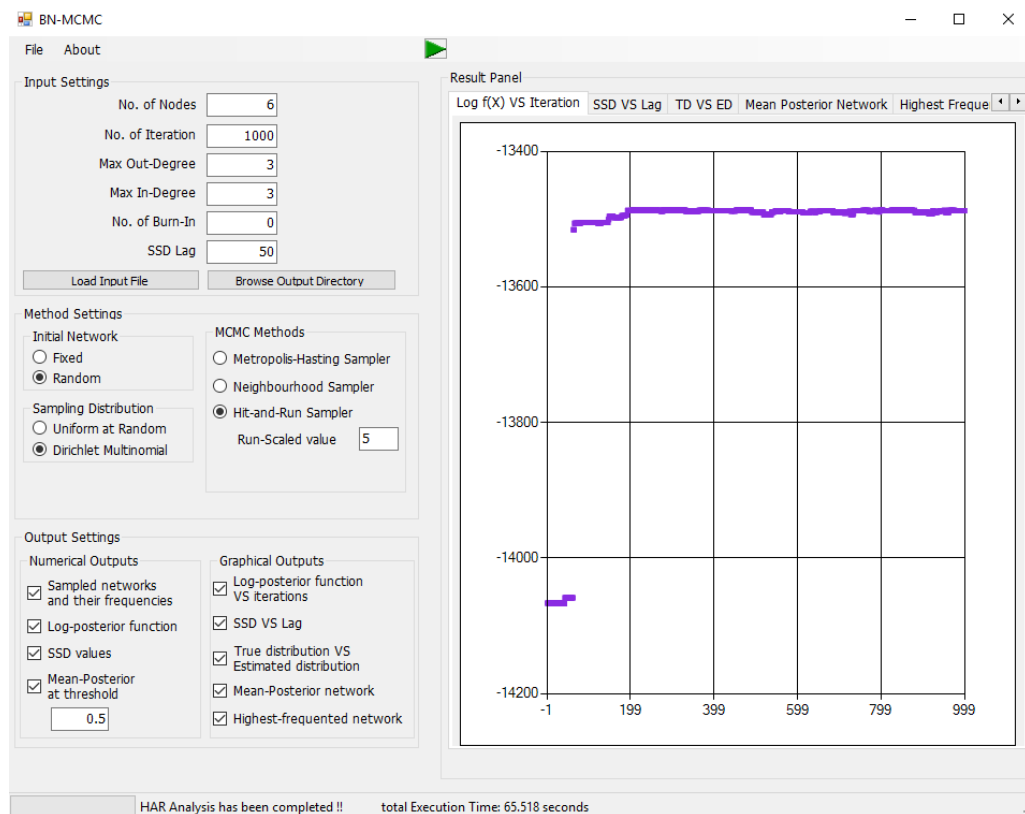


FIGURE 8.7: Log posterior vs iterations for learning Mendel network.

Figure 8.8 shows the second graphical result: the sum of squared differences against the lag. As desired, the SSD values decrease as the lag increases.

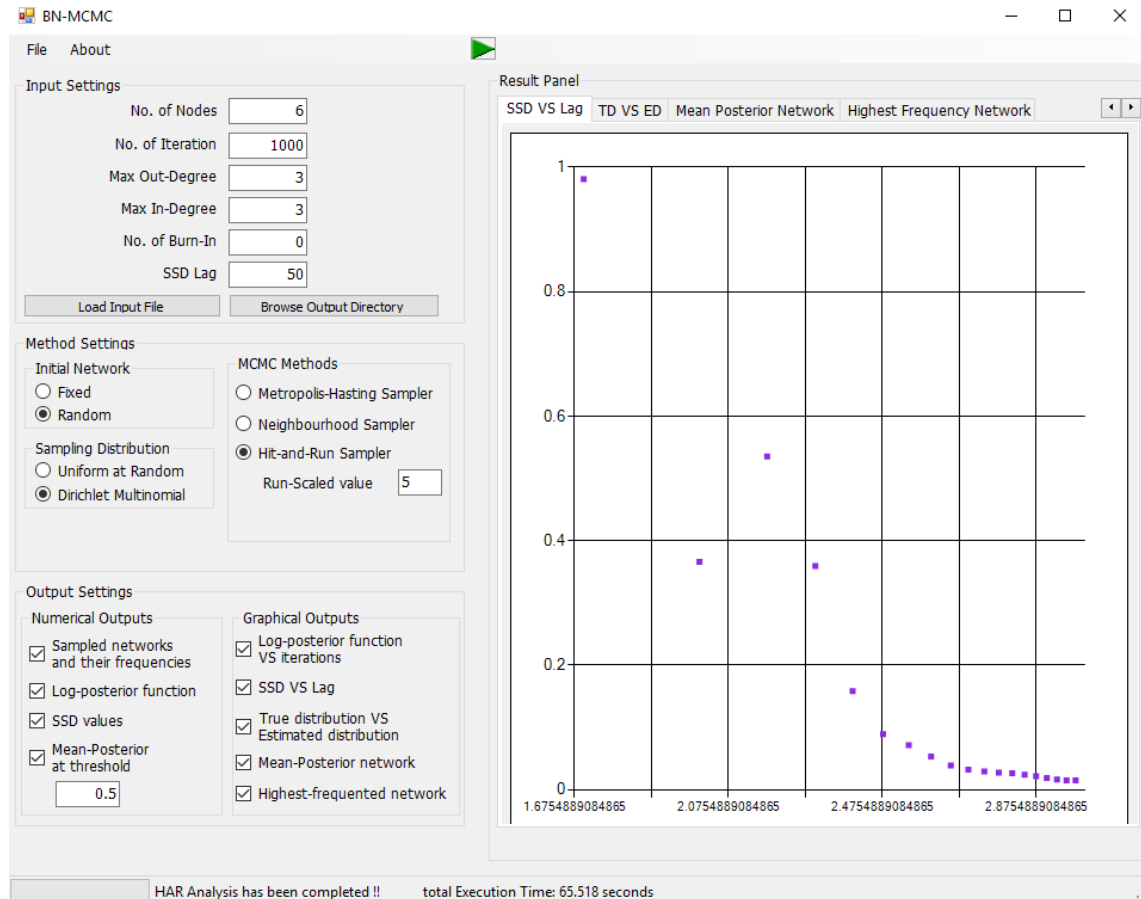


FIGURE 8.8: SSD vs Lag for learning Mendel network.

Figure 8.9 shows the third graphical result: the true target probabilities against the estimated distribution for only the sampled graphs. The difference between the true and estimated proportions can be diminished further if one increases the number of iterations.

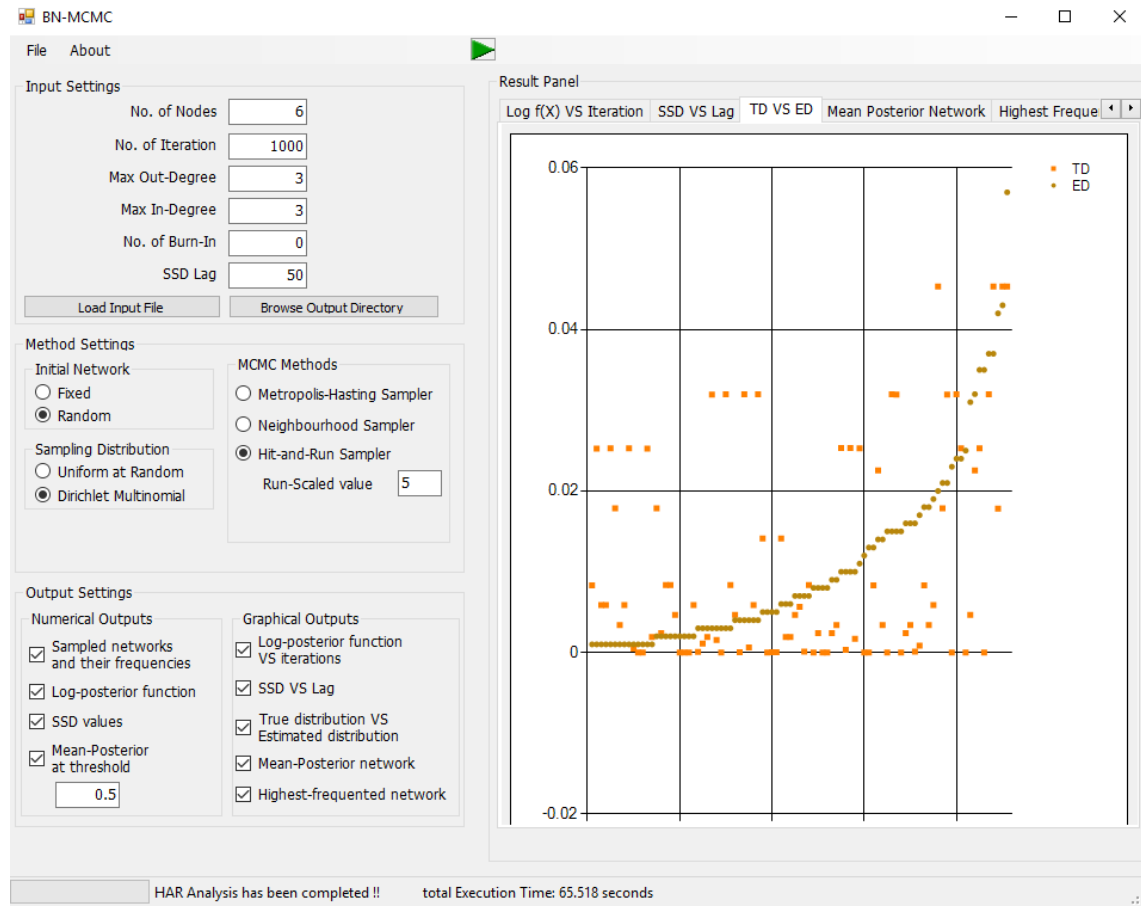


FIGURE 8.9: True target probabilities (TD) against the estimated distribution (ED) for learning Mendel network.

Figure 8.10 shows the fourth graphical result: the network comprised of edges with posterior edge probabilities greater than or equal 0.50. The network includes five directed edges with posterior probabilities  $\geq 50\%$ . The node names in the source column refer to the node parents, and those in the target column refer to the node children. Note that the five predicted directed edges are exactly the same directed edges that appear in the true network.

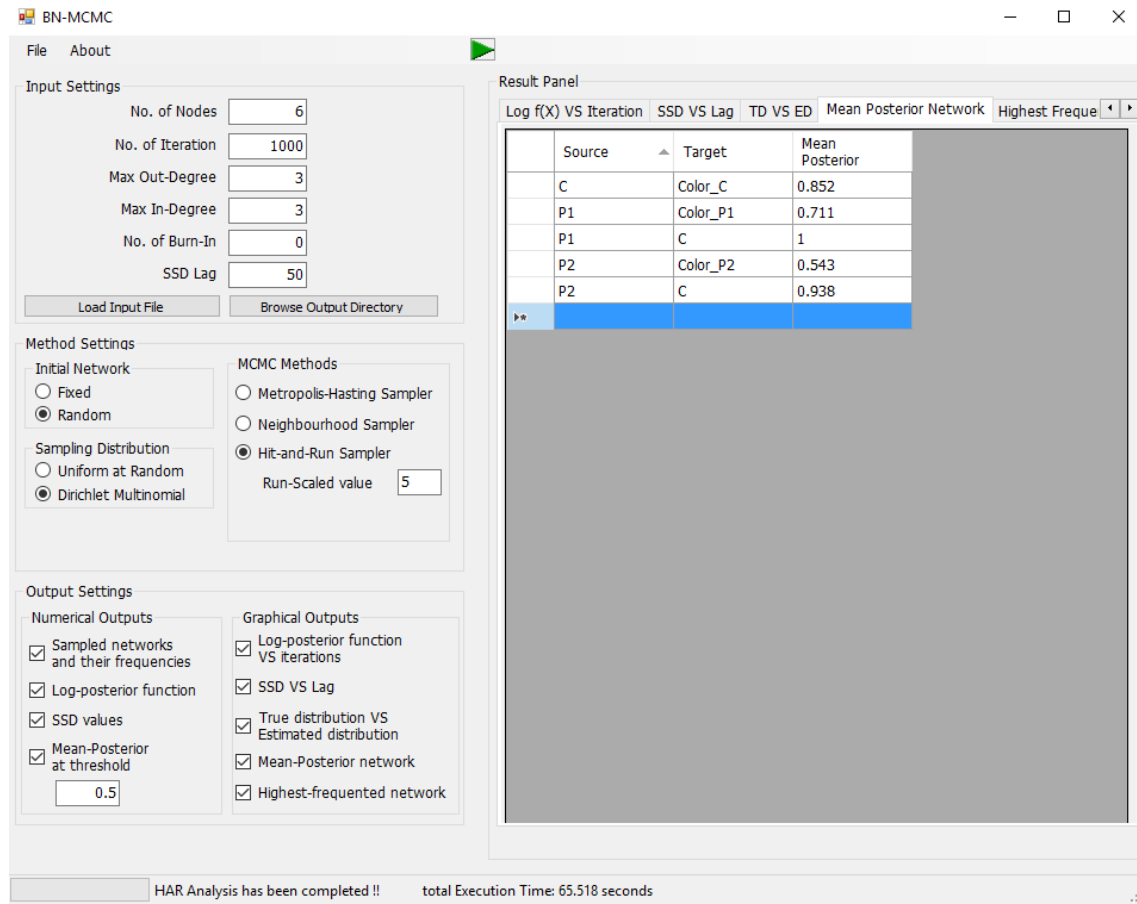


FIGURE 8.10: List of edges with posterior probabilities greater than the specified threshold.

Figure 8.11 shows the fifth graphical result: the network sampled with the highest frequency. Four directed edges out of the five true edges are inferred, and one directed edge was predicted in the opposite direction.

**Remark 12.** It is recommended to consider the network inferred using posterior edge probabilities as the best representative network rather than the highest frequency network.

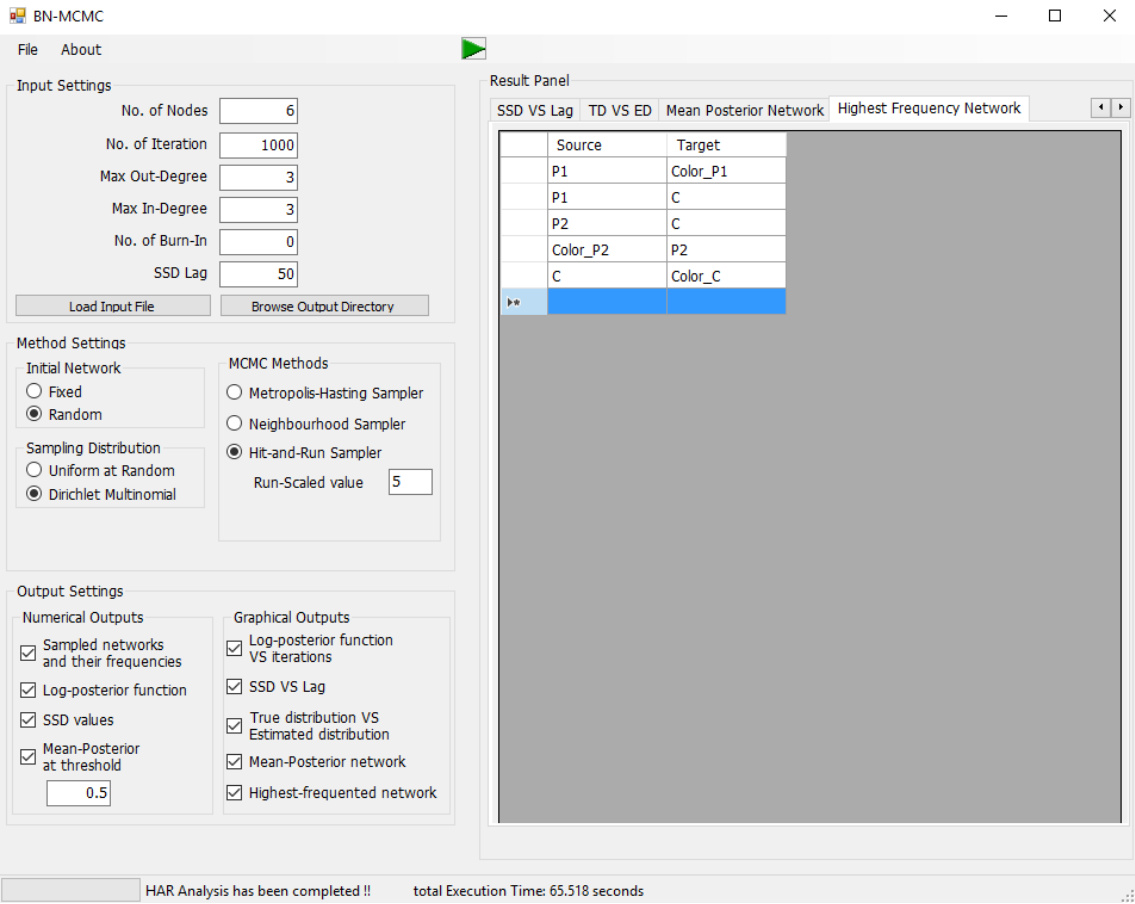


FIGURE 8.11: Edges of the highest frequency network in MCMC output.

## 8.8 Conclusion

Currently, this software is only a desktop version, but in future, I would like to distribute it as a web-based application for better reachability among the scientific community.

This software currently does not provide the option of simulating from or running an MCMC sampler based on a given model structure, nor for informative versions of the prior distribution. In future, I would like to incorporate this feature into this software.

# Chapter 9

## Conclusion

This chapter discusses the following. Section 9.1 briefly outlines the scope of the thesis. Section 9.2 summarises the main findings including the relative features and limitations of MCMC samplers when they are used to infer BNs. Section 9.3 proposes some possible extensions and future work related to the thesis.

### 9.1 Scope of the thesis

This thesis has proposed two MCMC samplers to generate BNs from discrete spaces. The two samplers are new instances of the NS and HAR. The performances of these two samplers have been evaluated in comparison with the MH sampler, which is a general framework for a number of MCMC samplers.

The thesis has also proposed two adaptive techniques to first reduce the time complexity of enumerating a set of adjacent graphs, and second to avoid recalculating conditional probabilities of all nodes within a sampled network.

All the MCMC samplers used in this thesis have been used to first generate BNs uniformly, and second to learn causal relationship structures among variables from observed datasets. The conditional probabilities among variables have been calculated using the Dirichlet-Multinomial distribution, which is a statistical model combining two distributions: the Multinomial distribution to describe the observed dataset and the Dirichlet distribution to describe prior beliefs.

## 9.2 Findings summary

The features and shortcomings of the MH, HAR and NS when used to infer BNs are summarised and ranked in Table 9.1. I used the numbers 1, 2, and 3 as a grading scale to respectively identify best, middle and worst performance of the three samplers, as shown in Table 9.1.

Feature	MH	HAR	NS
Speed per iteration	1	2	3
Generating BNs uniformly	3	1	2
Inferring BN structures	3	2	1
Ease of implementation	1	2	2
Exploring posterior distribution	3	2	1
Less influenced by local maxima	3	1	2
Less correlated samples	3	1	2

TABLE 9.1: Ranking MCMC samplers at different performance criteria, where 1 = best, 2 = middle and 3 = worst.

In terms of the speed per iteration, the MH sampler is the fastest, and experimentally is roughly 50% faster than the HAR sampler when the paths-length is set to five (the default), and 5% faster than the NS. The random BNs generated by the HAR sampler are slightly more uniformly distributed than those by the NS and considerably than those by the MH. The NS rapidly converges to the target distribution in fewer iterations than the HAR and MH. The HAR sampler in turn converges to the same target distribution faster than the MH sampler. Implementation of the MH sampler is generally easier than the NS and HAR because the MH sampler does not involve a redaction step as in the NS or sampling over paths as in the HAR sampler. The problem of local modes in graph spaces is ameliorated by the NS and to a lesser extent by the HAR sampler, whereas the MH sampler is



vulnerable to this problem. The HAR sampler has been shown to produce samples that are less dependent compared to the NS and MH, due to its potential to reach distant graphs in a single transition.

## 9.3 Future work

Below, I describe some more possible work to be done in future to the *BNMCMC*, MCMC samplers and adaptive algorithms.

### 9.3.1 Possible update to *BNMCMC*

The next version of *BNMCMC* is intended to include more options and new learning techniques. Below are some additional features that are under development:

- Define  $f(X)$ : the current version uses the Dirichlet-Multinomial (DM) distribution as a Bayesian inference model to learn the conditional probabilities of BNs from a discrete dataset. The option "Define  $f(x)$ " would enable users to define models other than the DM distribution to learn from discrete and continuous data.
- No. of random BNs: in the current version, I use the uniform distribution to generate random BNs given a number of iterations  $t$ , which typically does not produce  $t$  random networks. I add the option "No. of random BNs" to exactly produce a specific number of random networks from the same space.
- Highest S. BN: this option is used with the posterior distribution when a MCMC sampler is run to learn Bayesian network structures. The "Highest S. BN" aims to produce the network that is learned with the highest score function over the simulation. The result of "Highest S. BN" will be added as a graphical output in the "Result Panel" in *BNMCMC*.

- Metropolis MCMC sampler: this sampler is a special case of the MH sampler, but uses a symmetric proposal distribution. I am currently investigating how to use the uniform distribution to propose graphs symmetrically.
- A list of diagnostic tests: this option would perform some of the diagnostic tests (e.g. Gelman and Rubin test, Geweke test, trace-plot, autocorrelation plot) available in some R packages (e.g. *coda* with outputs produced in .csv files).
- No. of CUAGs: this option would produce connected un-directed graphs (CUDGs). This type of structure is not a Bayesian network; however, I have built new code to effectively generate such structures at random, with many potential applications. For example, some real-world networks are quite large e.g. the *movie actor network* (nodes are used to represent movies and undirected edges are used when two actors have played in the same movie. Such a large example is virtually impossible to describe in detail or model effectively. This problem has been circumvented by considering random undirected connected graphs as network models. Appendix [D.1](#) explains how to construct a set of adjacent graphs for a particular CUDG, and provide some experimental results obtained after generating CUDGs from a space of four and five nodes consisting of 38 and 728 CUDGs, respectively, using the NS.
- Simulation progress: it aims to show a live window to constantly monitor simulation progress. The window may display the progress in the number of iterations, execution time, the number of adjacent graphs instantly for each iteration, the number of rejected networks in the reduction step with the NS, and the number of nodes that have updated their CPTs in a single iteration.
- Multi-mode running: this feature allows users to optionally run the samplers individually or altogether in the same time under the same settings.

### 9.3.2 Mathematical work

- Accelerating convergence with the HAR sampler: one possible technique to enhance the convergence behaviour of the HAR sampler is to add a reduction step to the acceptance ratio step. To do so, one primary procedure is to generate a candidate graph  $G'$  from a sampled path, then a reduction step can do a local search over the adjacent graphs of  $G'$  until the acceptance ratio is satisfied. If the acceptance ratio is not satisfied by any of the adjacent graphs of  $G'$ , I then break the iteration and stay in the current graph. However, this work is still incomplete and requires more theoretical investigation.
- Approximate acceptance ratio: this technique aims to reduce the time complexity required by the reduction step involved in the NS. It intends to estimate the number of adjacent graphs  $|\mathcal{N}|$  instead of searching the adjacent graphs and then counting them. One reason to estimate  $|\mathcal{N}|$  is that the difference in the number of adjacent graphs between two graphs  $G$  and  $G' \in \mathcal{N}(G)$  is very small. This tiny error may not significantly affect the convergence. However, this idea has not been proved theoretically and requires more investigations.
- Improve the time complexity of the adaptive adjacent graphs enumeration: as a continuation of my previous work in this thesis to enhance the time complexity required to enumerate adjacent graphs, I am currently investigating the possibility to reduce the time complexity to be strictly less than  $O(V^4)$ .
- Order-space sampling (OSS): one possible technique to reduce the amount of computation required to infer a BN is to use the OSS technique [124]. The OSS assumes knowledge of ordering of the nodes. Suppose the nodes are indexed according to some known ordering, given that the parents of node  $v_j$  can only come from the nodes  $v_1, \dots, v_{j-1}$ . That is, parents of  $v_j$  are only generated from nodes that precede it in the ordering. Note that this would eliminate the

---

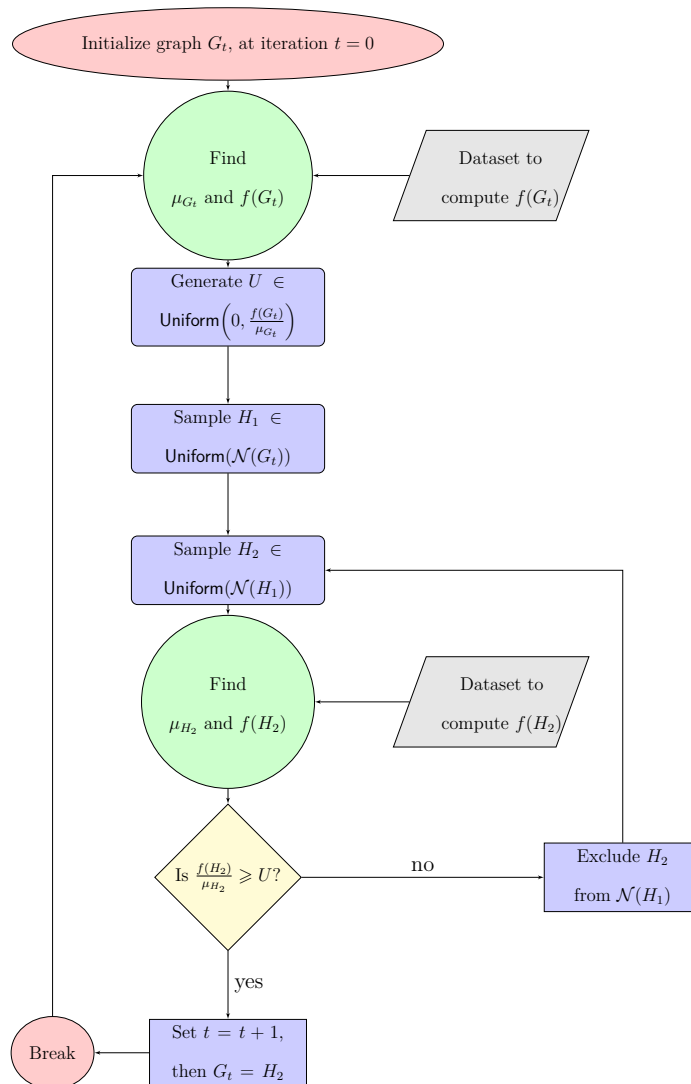
need for cycle checking. In this work, I aim to apply the MH, HAR and NS to real-life applications when the order of nodes is known, and compare the results in the absence of ordering.

- Full analysis of time complexity of adaptive algorithms and theoretical results concerning convergence and mixing times.

# Appendix A

## Using MCMC Samplers to Sample Bayesian Networks

### A.1 Flowchart describing the NS process



## A.2 All possible paths within three transitions of a graph of three nodes

**Example 11.** Consider connected BNs with three vertices. Recall that the graph space of all such BNs contains 18 graphs. The Figure A.1 shows all possible paths of length three, starting from the graph in the centre. Note that all the 18 graphs are within three transitions from the graph given in the centre.

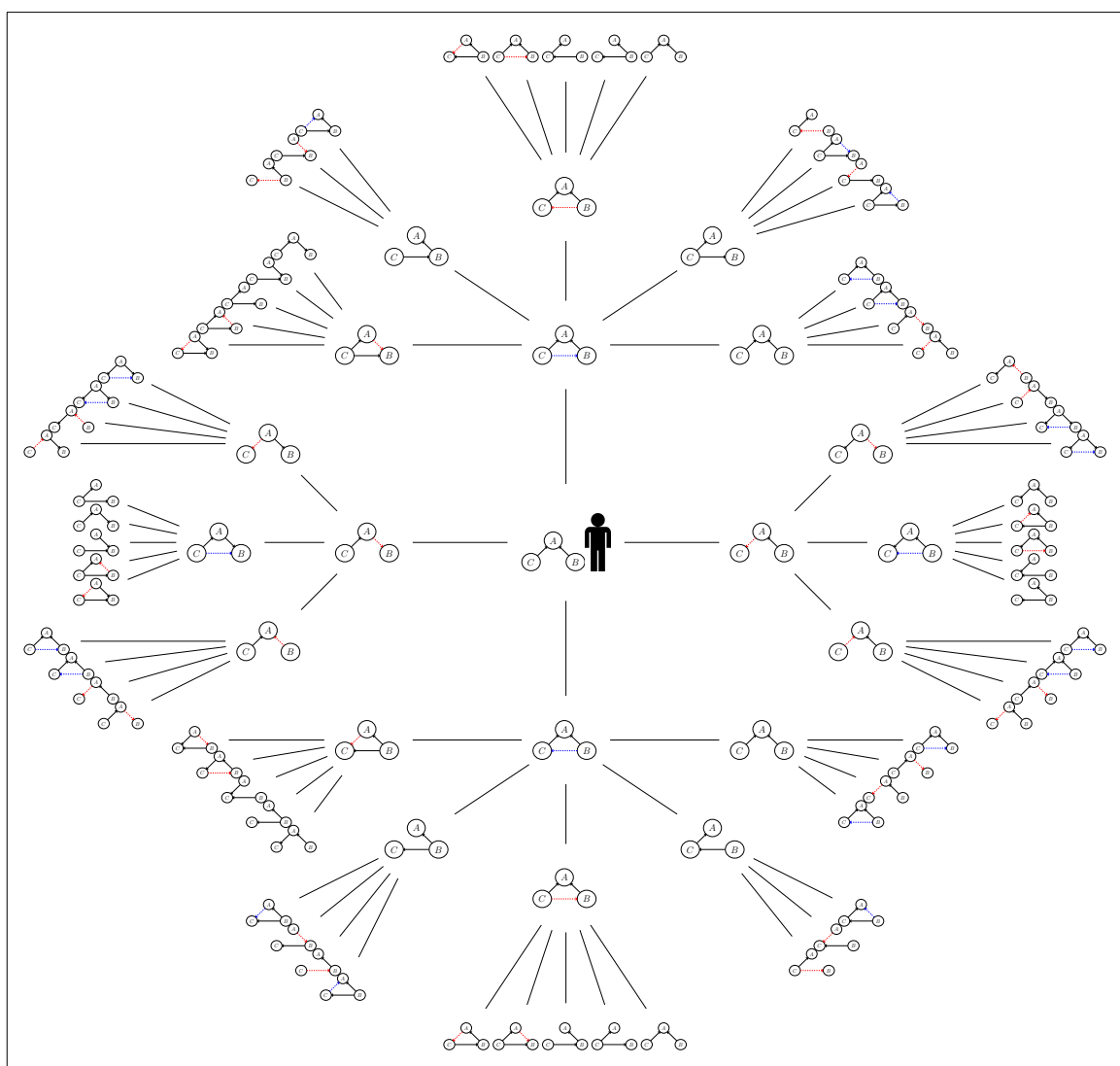


FIGURE A.1: All possible paths within 3 transitions of a graph of 3 nodes.

# Appendix B

## Sampling Bayesian Networks

### Uniformly

#### B.1 Checking normality

Figure B.1, Figure B.2, Figure B.3, Figure B.4, Figure B.5, and Figure B.6 plot the frequency histograms for the samples obtained by the MH, HAR and NS in Figure 5.5, Figure 5.6, Figure 5.7, Figure 5.8, Figure 5.9 and Figure 5.10, respectively.

The bars in the histograms are symmetrically distributed around the value obtained by dividing the number of iterations by the number of graphs in the target space, when the number of iterations is large. With all MCMC samplers, the bars in the histograms of four nodes have been normally distributed using only 10,000 iterations as illustrated in the top right panel of Figure B.1, Figure B.2, and Figure B.3. With all MCMC samplers, the bars in the histograms of five nodes have been normally distributed using only 500,000 iterations as illustrated in the middle left panel of Figure B.4, Figure B.5, and Figure B.6.

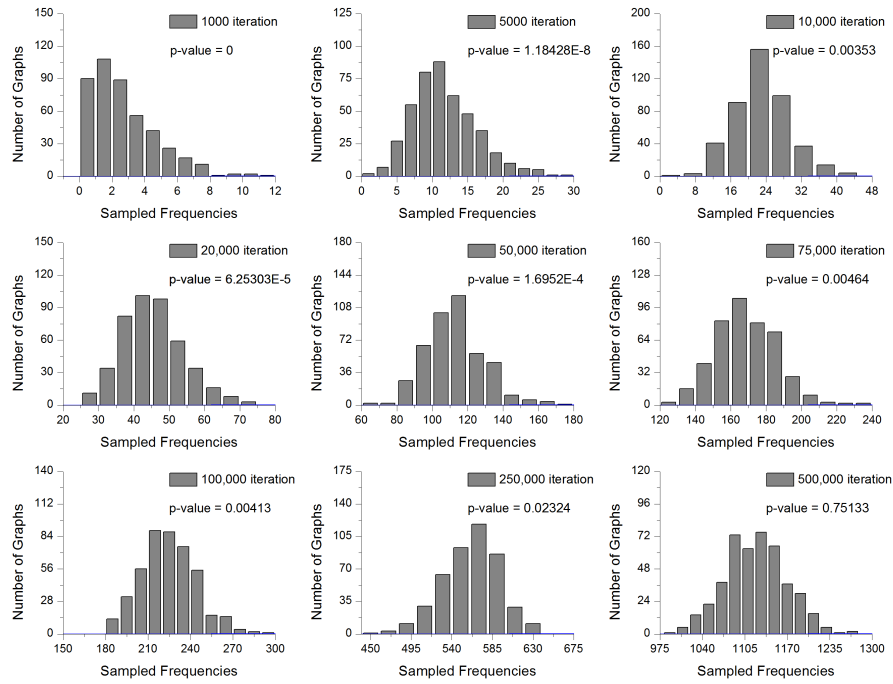


FIGURE B.1: MH with four nodes.

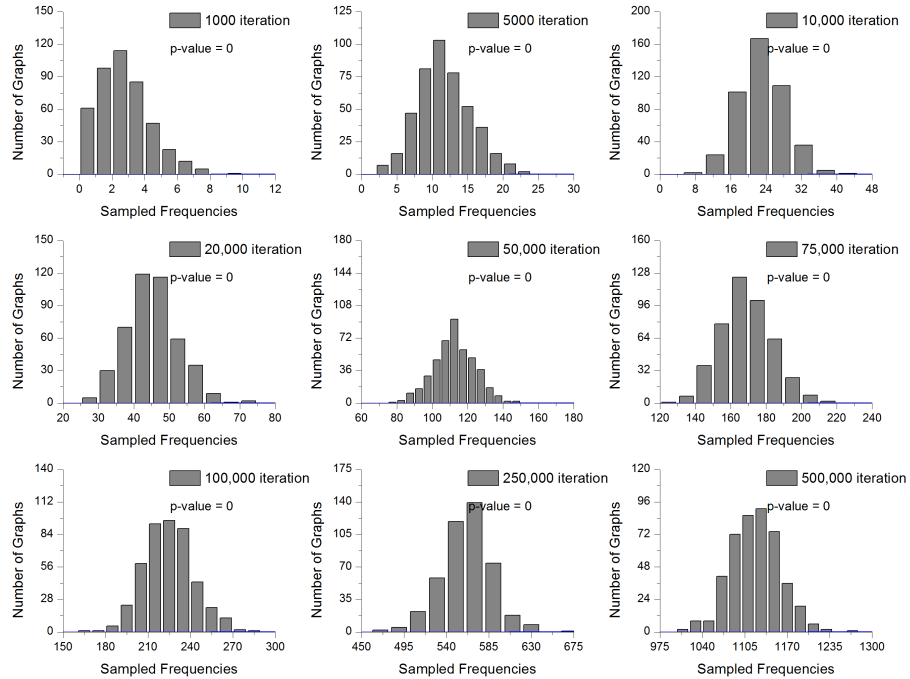


FIGURE B.2: HAR with four nodes.



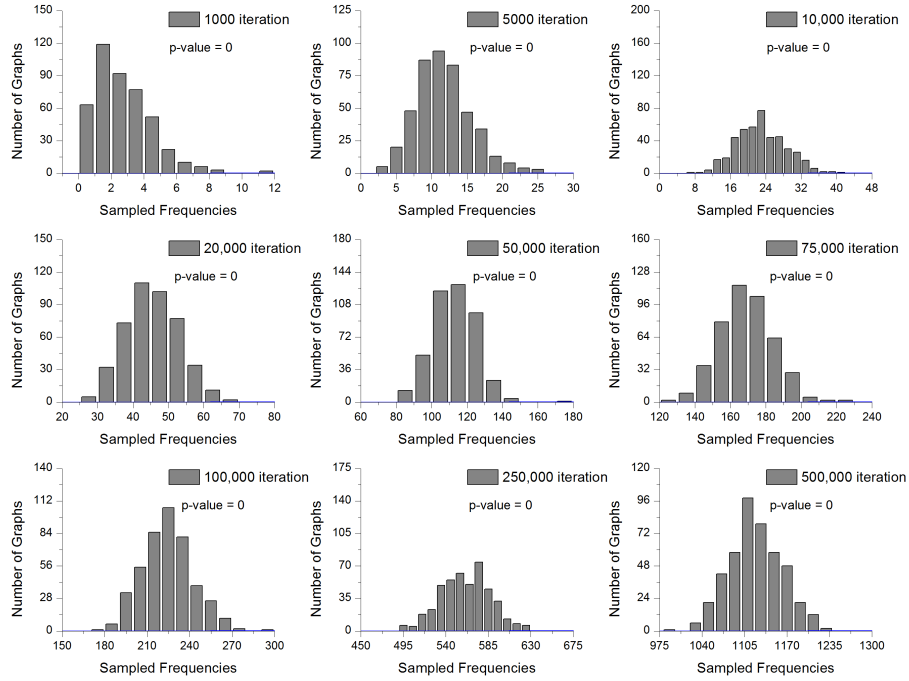


FIGURE B.3: NS with four nodes.

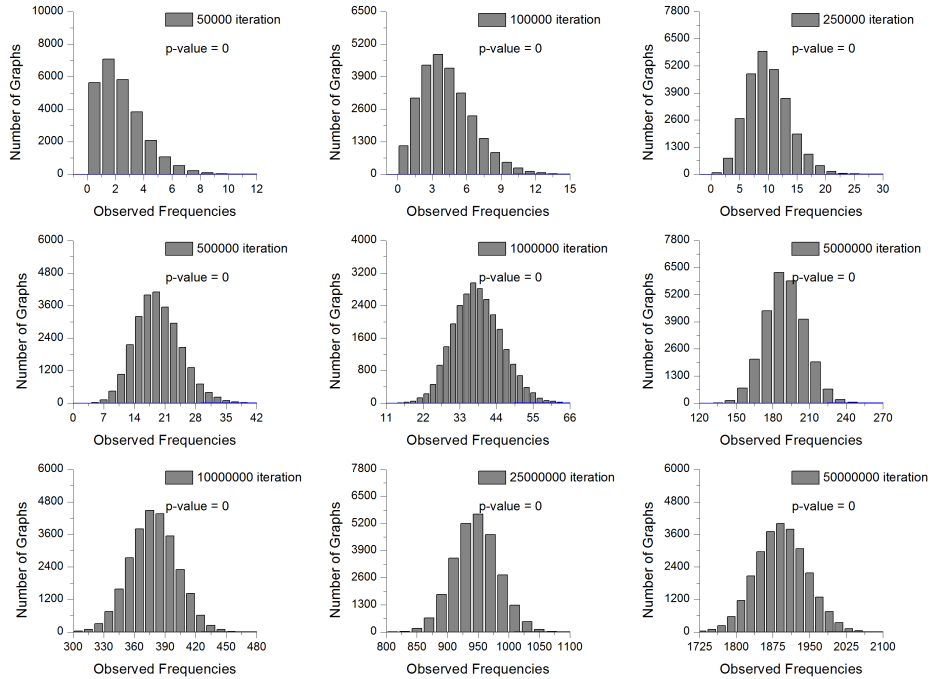


FIGURE B.4: MH with five nodes.

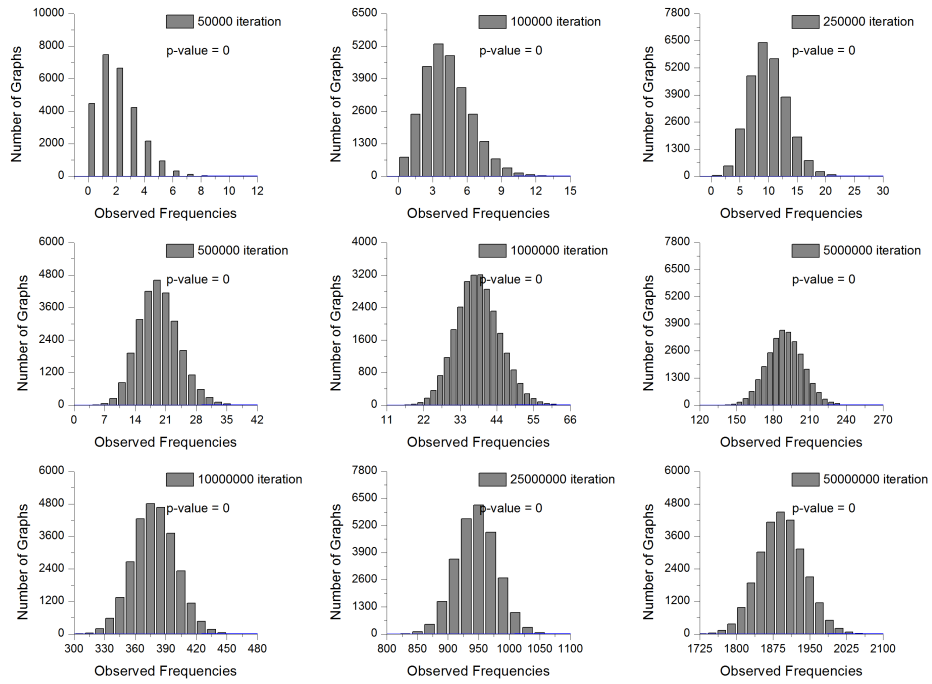


FIGURE B.5: HAR with five nodes.

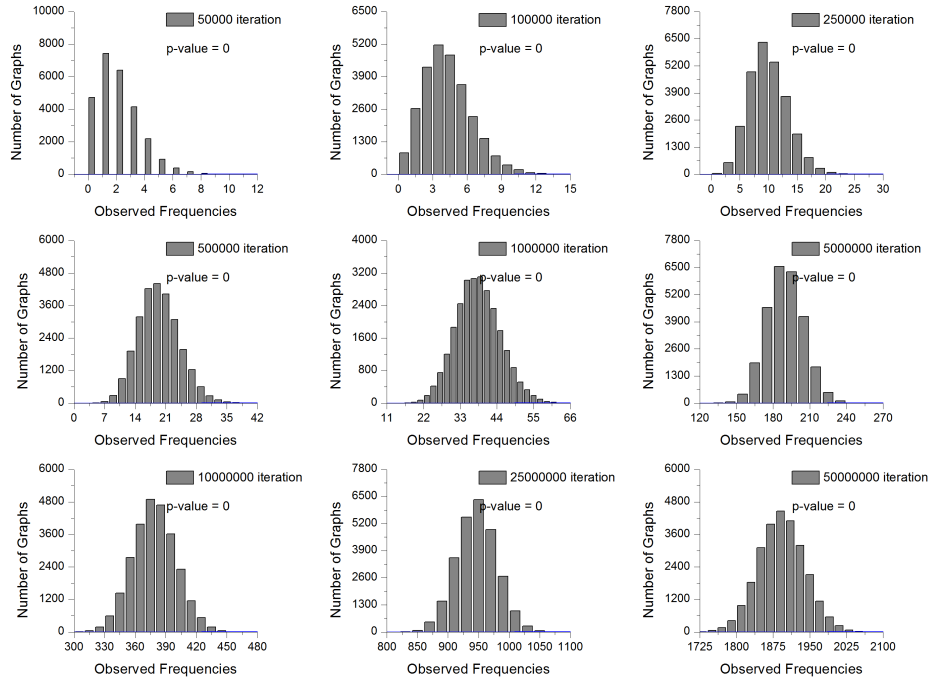


FIGURE B.6: NS with five nodes.

# Appendix C

## Applications

### C.1 Inferring structures from Microarray network

#### C.1.1 The true structure of Microarray network

Figure C.1 illustrates the true structure of Microarray network. There are three directed edges connecting the four genes. Each gene is represented by a binary random variable that takes two state values: ‘off’ or ‘on’. In the dataset file, these states are denoted ‘0’ and ‘1’ respectively.

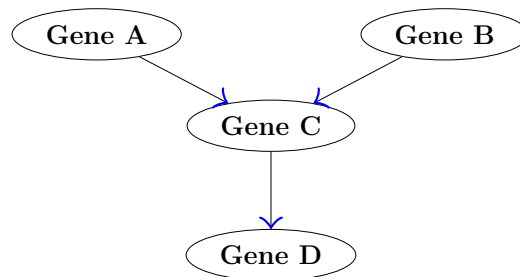


FIGURE C.1: The true structure of Microarray network.

#### C.1.2 The CPTs of Microarray network

Figure C.2 illustrates the conditional probability tables of Microarray network. These CPTs were used to simulate 5000 observations for each node in Figure C.1.

Gene A						Gene B	
$P(A=0)$		$P(A=1)$				$P(B=0)$	
0.50		0.50				0.50	

Gene C			
A	B	$P(C=0)$	$P(C=1)$
0	0	1	0
0	1	0.80	0.20
1	0	0.90	0.10
1	1	0.05	0.95

Gene D		
C	$P(D=0)$	$P(D=1)$
0	0.10	0.90
1	0.95	0.05

FIGURE C.2: The CPTs of Microarray network.

## C.2 Inferring the Mendel Peas network

### C.2.1 The CPTs of Mendel network

Figure C.3 illustrates the conditional probability tables of Mendel network. These CPTs were used to simulate 5000 observations for each of the six nodes.

P1		
$P(P1 = RR)$	$P(P1 = Rr)$	$P(P1 = rr)$
0.25	0.50	0.25

C				
P1	P2	$P(C = RR)$	$P(C = Rr)$	$P(C = rr)$
RR	RR	1	0	0
RR	Rr	0.50	0.50	0
RR	rr	0	1	0
Rr	RR	0.50	0.50	0
Rr	Rr	0.25	0.50	0.25
Rr	rr	0	0.50	0.50
rr	RR	0	1	0
rr	Rr	0	0.50	0.50
rr	rr	0	0	1

Color P1		
P1	$P(\text{Color P1} = red)$	$P(\text{Color P1} = white)$
RR	1	0
Rr	1	0
rr	0	1

Color C		
C	$P(\text{Color C} = red)$	$P(\text{Color C} = white)$
RR	1	0
Rr	1	0
rr	0	1

P2		
$P(P2 = RR)$	$P(P2 = Rr)$	$P(P2 = rr)$
0.25	0.50	0.25

Color P2		
P1	$P(\text{Color P1} = red)$	$P(\text{Color P1} = white)$
RR	1	0
Rr	1	0
rr	0	1

FIGURE C.3: The CPTs used to simulate 5000 datapoints.

### C.2.2 Summary statistics

Table C.1 summarises the outputs obtained from the three chains sampled after running 5000 iterations using the MH, HAR and NS in Section 7.2. The summary statistics consider all sampled graphs before applying a burn-in interval, so that the performances of the samplers are expected to vary. It has been noted that the NS and HAR have the potential to explore the target space by returning more graphs than the MH sampler as shown in the last column in Table C.1. In terms of the frequency of the true graph and its ranking, the NS and HAR again have returned higher frequencies with better rankings compared to the MH sampler.

MCMC Sampler	True Graph Frequency	True Graph Ranking	Total Sampled Graphs
MH	53	19	267
HAR	66	15	368
NS	78	6	570

TABLE C.1: Summary outputs obtained from running three chains of 5000 iteration using the MH, HAR and NS.

### C.2.3 Mendel Peas network based on adding and deleting edges

In this section, I briefly investigate the potential of the MH, HAR and NS to infer the Mendel Peas network structure when one considers adding and deleting edges only, that is, no transition between two adjacent graphs by reversing.

I ran four chains of 1000 iterations with each sampler. Each chain was run from a different initial network structure. The four initial networks are shown in Figure C.4. The maximum number of parents and children in the true network are two

each. Therefore, the graph space was reduced by imposing a maximum of three parents and three children for each node.

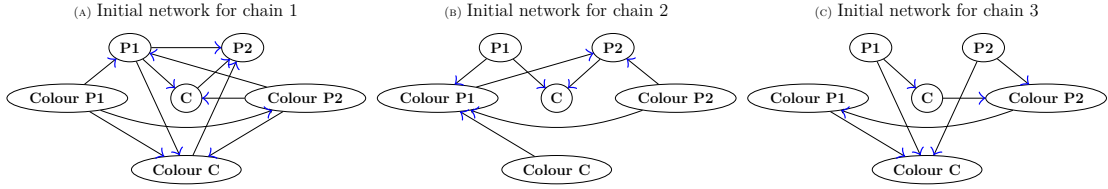


FIGURE C.4: Three initial graphs for running three Markov chains with lengths of 1000 iterations each using the MH, HAR and NS.

Figure C.5 shows the mean posterior edges probabilities of all 12 chains. The NS and HAR have shown better performance than the MH sampler. The latter is likely to sample the true graph only if it gets stuck in a local mode containing the true graph.

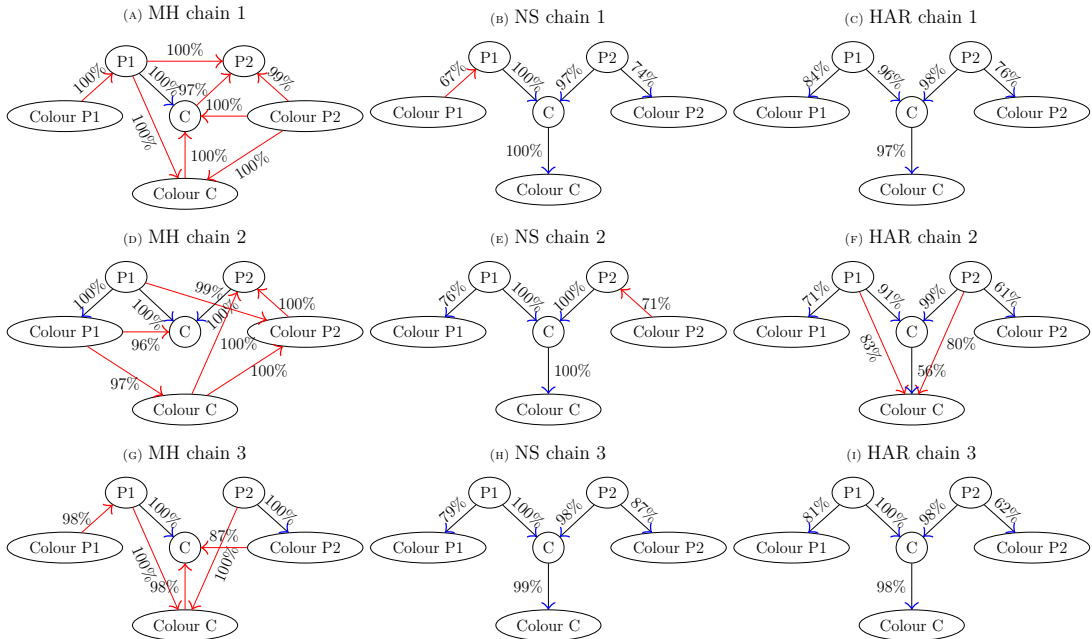


FIGURE C.5: Posterior edge probabilities at threshold  $\geq 50\%$ .

## C.3 Inferring the Diagnostic Chest network

Table C.2 illustrates the conditional probability tables of the true structure of Diagnostic Chest network used to simulate 5000 data-points for each of the eight

variables.

Smoking			
		$P(\text{Smoking} = T)$	$P(\text{Smoking} = F)$
		0.50	0.50

Travel			
		$P(\text{Travel} = \text{yes})$	$P(\text{Travel} = \text{no})$
		0.01	0.99

Lung Cancer			
Smoking		$P(\text{Lung Cancer} = \text{present})$	$P(\text{Lung Cancer} = \text{absent})$
smoker		0.10	0.90
non smoker		0.01	0.99

Tuberculosis			
Travel		$P(\text{Tuberculosis} = \text{present})$	$P(\text{Tuberculosis} = \text{absent})$
yes		0.05	0.95
no		0.01	0.99

Bronchitis			
Smoking		$P(\text{Bronchitis} = \text{present})$	$P(\text{Bronchitis} = \text{absent})$
smoker		0.60	0.40
non smoker		0.30	0.70

Tuberculosis or Cancer			
Tuberculosis	Lung Cancer	$P(\text{TbOrCa} = \text{true})$	$P(\text{TbOrCa} = \text{false})$
present	present	1	0
present	absent	1	0
absent	present	1	0
absent	absent	0	1

Dyspnea			
TbOrCa	Bronchitis	$P(\text{Dyspnea} = \text{present})$	$P(\text{Dyspnea} = \text{absent})$
true	present	0.90	0.10
true	absent	0.70	0.30
false	present	0.80	0.2
false	absent	0.10	0.90

XRay Results		
Tuberculosis or Cancer	$P(\text{XRay} = \text{abnormal})$	$P(\text{XRay} = \text{normal})$
true	0.98	0.02
alse	0.05	0.95

TABLE C.2: The CPTs of the Chest Clinic network used to simulate 10,000 datapoints.

## C.4 Inferring the Raf-Signaling Pathway network

### C.4.1 Twelve random initial networks

Figure C.6 illustrates the twelve initial networks generated randomly using the technique proposed in Section 4.8 to learn the structure of Raf-Signaling Pathway using the MH, HAR and NS. Each initial network was used to run three chains of 10,000 iterations each using the three MCMC samplers. The posterior edge probabilities for all the resultant 36 chains are separately shown in Figures 7.20 - 7.31 in Chapter 7.

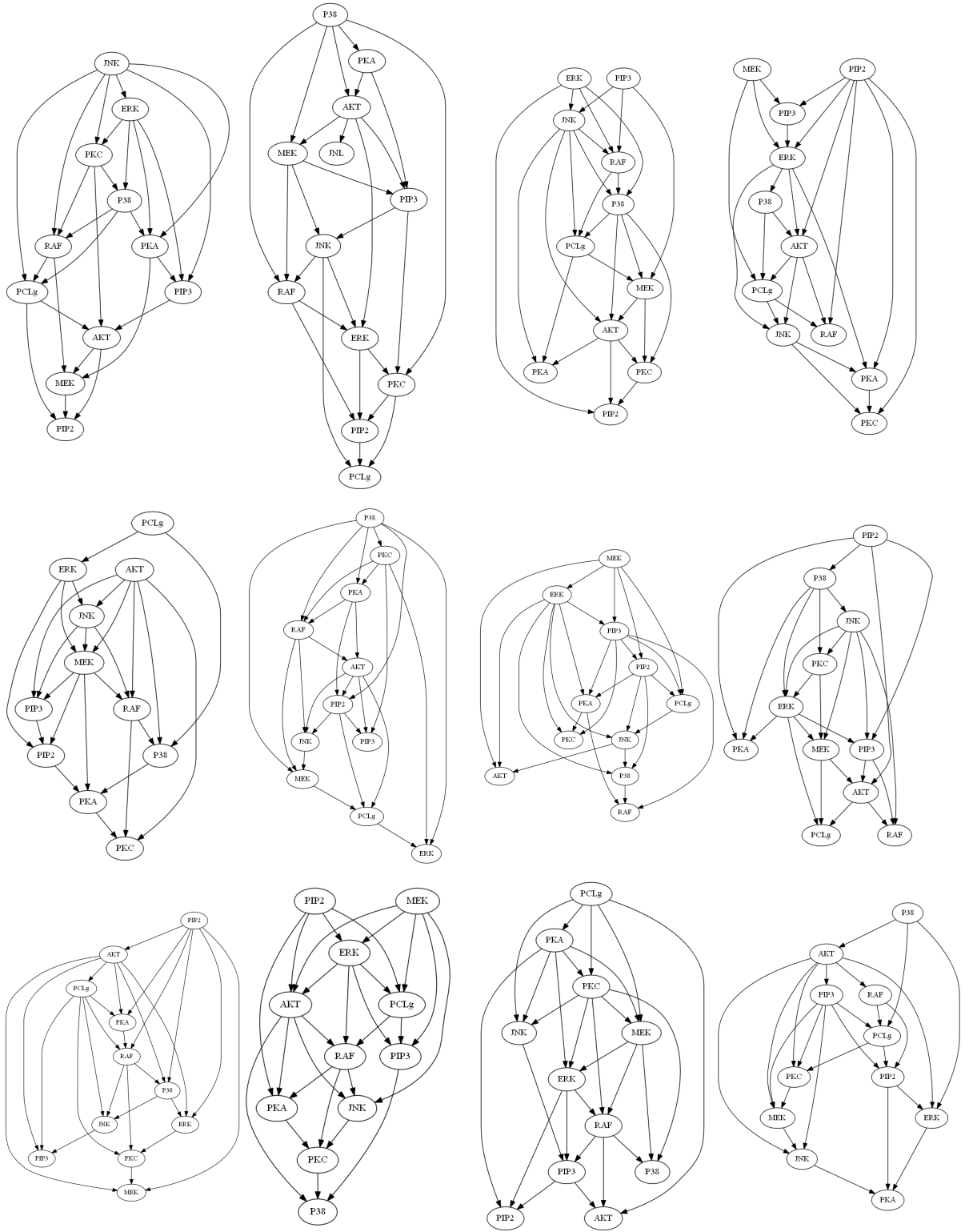


FIGURE C.6: Twelve random initial networks.



### C.4.2 Log posterior for 36 chains and 10,000 iterations each

This section uses the MH, HAR and NS approaches to run twelve Markov chains with 10,000 iterations each, given the twelve initial networks plotted in Section C.4.1. It then determines the point at which burn-in has occurred given the time-series plot of the log posterior at each iteration shown in Section 7.4.2.

Figure C.7, Figure C.8 and Figure C.9 plot separately the log posterior distributions for the twelve Markov chains and 10,000 iterations, produced by the MH, HAR and NS, respectively.

Figures in this section demonstrate that: with the MH, the chains are often stuck at local modes, with the HAR, the chains have ultimately returned the same common mode even though convergence has occurred a bit late with some chains, and with the NS, all chains have early converged to a single common mode.

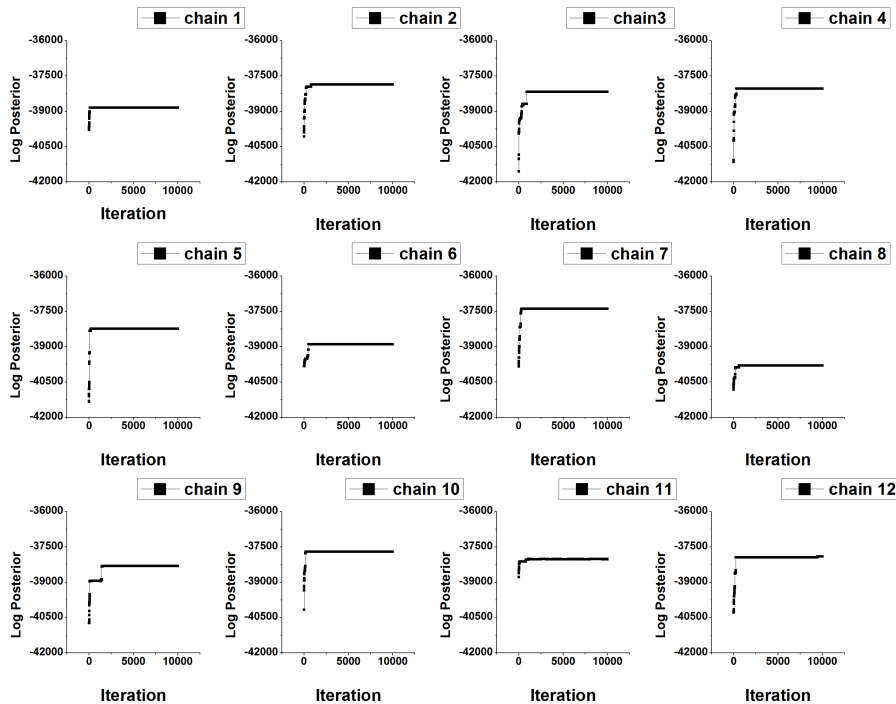


FIGURE C.7: MH: Log posterior for 12 chains and 10,000 iterations each.

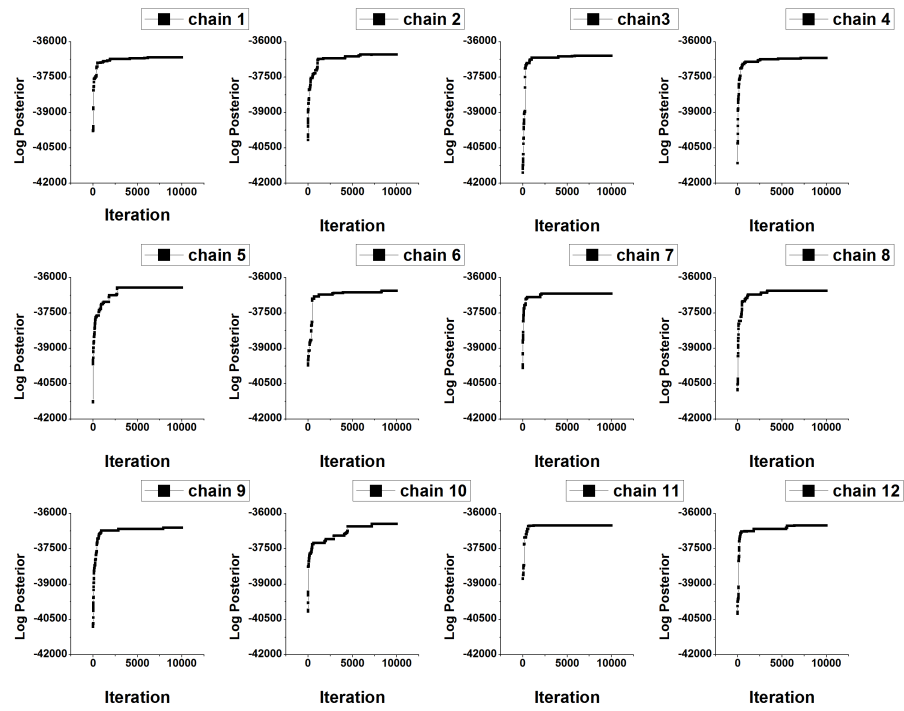


FIGURE C.8: HAR: Log posterior for 12 chains and 10,000 iterations each.

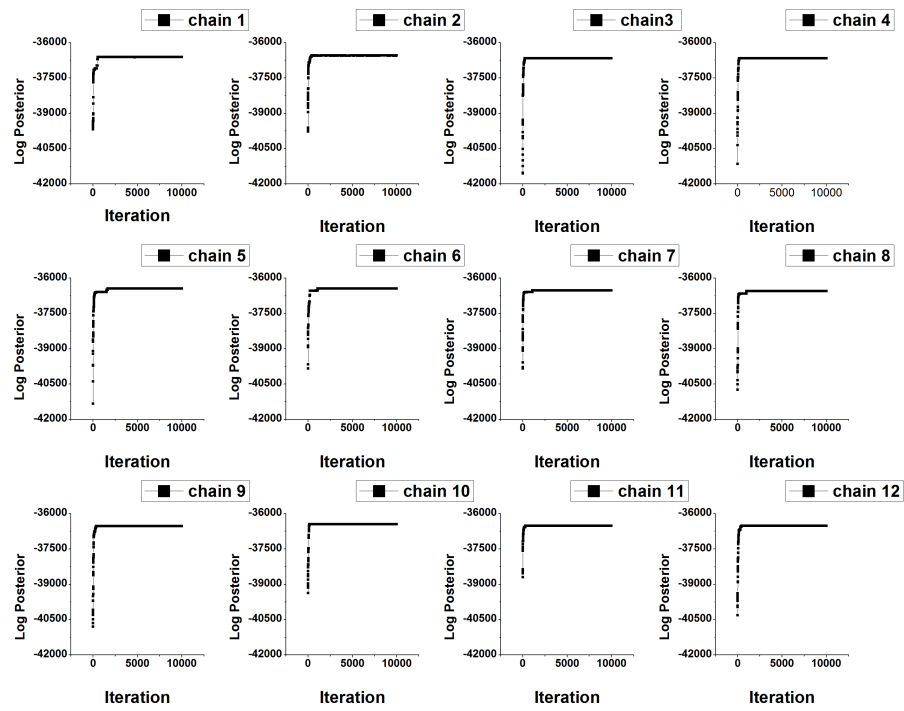


FIGURE C.9: NS: Log posterior for 12 chains and 10,000 iterations each.

Figure C.10, Figure C.11 and Figure C.12 provide all the posterior edge probabilities of Figures 7.20 - 7.31 including probabilities less than 50%.

	RAF	MEK	PCl <sub>g</sub>	PIP2	PIP3	ERK	AKT	PKA	PKC	P38	JNK
RAF	0	1	0	0	0	0.47	0	0	0.51	0.54	0.88
MEK	0	0	0	0	0	0	0	0	0	0	0
PCl <sub>g</sub>	0	0	0	1	1	0	1	0	0	0	0
PIP2	0	0	0	0	0	0	0	0	0	0	0
PIP3	0	0	0	1	0	0	0	0	0	0	0
ERK	0.53	0	0	0	0	0	1	0.54	0.56	0.9	0
AKT	0	1	0	0	0	0	0	0	0	0	0
PKA	0	1	0	0	0	0	0	0	0	0	0
PKC	0.49	0	1	0	0	0.46	1	0	0	0.53	0.89
P38	0.46	0	1	0	0	0.44	0	1	0.47	0	0
JNK	0.12	0	1	0	1	0.1	1	1	0.11	0	0

	RAF	MEK	PCl <sub>g</sub>	PIP2	PIP3	ERK	AKT	PKA	PKC	P38	JNK
RAF	0	0.31	0	0	0	1	0.93	0.23	1	0	0.27
MEK	0.69	0	1	0	0	0	0.89	0.38	0	0.65	0.41
PCl <sub>g</sub>	0	0	0	1	1	0	0	0	0	0	0
PIP2	0	0	0	0	0	0	0	0	0	0	0
PIP3	0	0	0	1	0	0	0	0	0	0	0
ERK	0	0	0	0	0	0	0	0	0	1	0
AKT	0.07	0.11	0	0	0	0	1	0.04	0	0	0
PKA	0.77	0.62	0	0	0	0	0.96	0	0	0.69	0.5
PKC	0	0	1	0	1	0	0	0	0	0	0
P38	0	0.35	0	0	0	0	0	0.31	1	0	0.27
JNK	0.73	0.59	1	0	0	1	0	0.5	0	0.73	0

	RAF	MEK	PCl <sub>g</sub>	PIP2	PIP3	ERK	AKT	PKA	PKC	P38	JNK
RAF	0	0.24	0	0	0	1	1	0	1	0	0.26
MEK	0.76	0	0.45	0	0	1	1	0	1	0	0.31
PCl <sub>g</sub>	0	0.55	0	0	0.85	0	0	1	0	0.81	0.32
PIP2	0	0	0	0	0	0	0	0	0	0	0
PIP3	0	0	0.15	1	0	0	0	0	0	0	0.14
ERK	0	0	0	0	0	0	0	0	0	0	0
AKT	0	0	0	1	0	1	0	0	0	0	0
PKA	0	0	0	0	0	0	0	0	0	0	0
PKC	0	0	0	1	0	0	1	1	0	0	0
P38	0	0	0.19	0	0	0	0	1	0	0.15	0
JNK	0.74	0.69	0.68	0	0.86	0	0	1	0	0.85	0

	RAF	MEK	PCl <sub>g</sub>	PIP2	PIP3	ERK	AKT	PKA	PKC	P38	JNK
RAF	0	0	0	0	0	0	0	0	0	0	0
MEK	0	0	0.62	0	0	1	0.74	0.55	0	0	1
PCl <sub>g</sub>	0	0.38	0	0.87	0	0	0.41	0	0	0	1
PIP2	0	0	0.13	0	1	0	0	1	1	0	0
PIP3	0	0	0	0	0	0	0	0	0	0	0
ERK	0	0.26	0	0	0	0	0.25	1	0	0	1
AKT	1	0.45	0.59	0	0	0.75	0	0	0	0	1
PKA	1	0	0	0	0	0	0	0	1	1	0
PKC	1	0	0	0	1	0	0	0	0	1	0
P38	0	0	0	0	0	0	0	0	0	0	0
JNK	0	0	0	0	0	0	0	1	1	0	0

	RAF	MEK	PCl <sub>g</sub>	PIP2	PIP3	ERK	AKT	PKA	PKC	P38	JNK
RAF	0	0									

FIGURE C.10: From left to right top to bottom: Posterior edges probabilities calculated for chains from one to twelve produced by the MH sampler to learn Raf-Signaling network.

## C.4.3.2 Hit-and-Run sampler

	RAF	MEK	PCLg	PIP2	PIP3	ERK	AKT	PKA	PKC	P38	JNK
RAF	0	1	0	0	0	0	1	0.33	0.12	0	0.28
MEK	0	0	0.47	0	0	0	1	0	0	0	0
PCLg	0	0	0	1	1	0.22	0	0.05	0	0.05	0.05
PIP2	0	0	0	0	0	0	0	0	0	0	0
PIP3	0	0	0	1	0	0	0	0	0	0	0
ERK	0	1	0	0	0	0	1	0.07	0.03	0.01	0.04
AKT	0	0	0.44	0	0	0	0	0	0	0	0
PKA	0.67	1	0.95	0	0	0.93	0	0	0.22	0.7	0.45
PKC	0.88	0	0	0	1	0.97	0	0.78	0	0.77	0.69
P38	0	0	0.48	0	0	0.25	0	0.3	0.23	0	0.26
JNK	0.72	0	0.47	0	0	0.48	0	0.55	0.31	0.74	0

RAF	0	0	0	0	0	0	0	0	0	0	0
MEK	1	0	0.42	0	0	0	0.75	0.69	0.92	0	0.54
PCLg	0	0.58	0	0.84	0.67	0	0.14	0.78	0	0.33	0.37
PIP2	0	0	0.16	0	0.14	0	0	0	0	0	0
PIP3	0	0	0.33	0.86	0	0	0	0	0	0	0.43
ERK	0	0	0	0	0	0	0	0	0	0	0
AKT	0	0.25	0.32	0	0	1	0	0.34	0.43	0	0
PKA	1	0.31	0.22	0	0	1	0.66	0	0.85	0.67	0.49
PKC	1	0.08	0	0	0.49	0	0.57	0.15	0	1	0.46
P38	0	0	0	0	0	0	0	0	0	0	0
JNK	0	0.46	0.13	0	0.08	1	0	0.51	0.54	1	0

RAF	0	0	0	0	0	0	0	0	0	0	0
MEK	0.63	0	0	0	0	0.47	0.48	0.77	1	0	0.98
PCLg	0	0	0	1	1	0	0	0	0	0	0
PIP2	0	0	0	0	0	0	0	0	0	0	0
PIP3	0	0	0	1	0	0	0	0	0	0	0
ERK	0.65	0.53	0	0	0	0	0.49	0.76	0	0	0
AKT	0.66	0.52	0	0	0	0.51	0	0.82	0	0.1	0.09
PKA	0	0.23	0	0	0	0.24	0.18	0	1	1	0.95
PKC	0	0	1	0	0.89	0	0	0	0.9	0.89	0
P38	0	0	1	0	0.01	0	0	0	0	0	0
JNK	0	0.02	1	0	0.1	0	0.02	0.05	0.11	1	0

RAF	0	0	0	0	0	0	0	0	0	0	0
MEK	1	0	0.47	0	0	0.97	0.46	0.86	0	0.02	0
PCLg	0	0.53	0	1	1	0	0.38	0.74	0	0.98	0
PIP2	0	0	0	0	0	0	0	0	0	0	0
PIP3	0	0	0	1	0	0	0	0	0	0	0
ERK	0	0.03	0	0	0	0	0.11	0.17	0	0	1
AKT	0	0.54	0.62	0	0	0.9	0	0.85	0	0	1
PKA	1	0.14	0.26	0	0	0.83	0.15	0	1	1	1
PKC	1	0	0	0	1	0	0	0	0	0	0
P38	0	0	0	0	0	0	0	0	1	0	0
JNK	0	0	0	0	0	0	0	0	1	1	0

RAF	0	0	0	0	0	0	0	0	0	0	0
MEK	0.32	0	1	0	0	0	0.6	0.18	1	0.75	0
PCLg	0	0	0	1	1	0	0	0	0	0	0
PIP2	0	0	0	0	0	0	0	0	0	0	0
PIP3	0	0	0	1	0	0	0	0	0	0	0
ERK	0.26	0.4	0	0	0	0	0.3	1	0	0	0
AKT	0.57	0.82	1	0	0	0.7	0	0.51	0.48	0	0
PKA	0	0	1	0	0	0	0	0	0.52	1	1
PKC	0.24	0.25	0	0	1	0	0	0.48	0	1	1
P38	0	0	0	0	0	0	0	0	0	0	0
JNK	0	0	0	0	0	0	0	0	0	1	0

RAF	0	0.25	0	0	0	0.16	0.21	0	1	0	0
MEK	0.75	0	0.73	0	0	0.53	0.26	0.6	1	1	0
PCLg	0	0.27	0	0.94	1	1	0	0.19	0.27	0	0
PIP2	0	0	0.06	0	0.06	0	0	0	0	0	0
PIP3	0	0	0	0.94	0	0	0	0	0	0	0
ERK	0.84	0.47	0	0	0	0	0.31	0.57	0	0	0
AKT	0.79	0.74	0.8	0	0	0.69	0	0.87	0	0	0
PKA	0	0.4	0.73	0	0	0.43	0.13	0	1	1	1
PKC	0	0	0	0	1	0	0	0	0	1	1
P38	0	0	0	0	0	0	0	0	0	0	1
JNK	0	0	0	0	0	0	0	0	0	0	0

RAF	0	0.56	0	0	0	0.53	0.56	0	1	0	0
MEK	0.44	0	0.95	0	0	0.48	0.45	0.91	0	1	0
PCLg	0	0.05	0	1	1	0	0.03	0.06	0	0	0.44
PIP2	0	0	0	0	0	0	0	0	0	0	0
PIP3	0	0	0	1	0	0	0	0	0	0	0
ERK	0.47	0.52	0	0	0	0	0.39	0.89	0	0	0
AKT	0.43	0.55	0.97	0	0	0.61	0	0.97	1	0	0
PKA	0	0.09	0.94	0	0	0.11	0.03	0	1	1	1
PKC	0	0	0	0	0.89	0	0	0	0	1	1
P38	0	0	0	0	0	0	0	0	0	0	0.56
JNK	0	0	0	0	0.11	0	0	0	0	0	0

FIGURE C.11: From left to right top to bottom: Posterior edges probabilities calculated for chains from one to twelve produced by the HAR sampler to learn Raf-Signaling network.

## C.4.3.3 Neighbourhood Sampler

	RAF	MEK	PCLg	PIP2	PIP3	ERK	AKT	PKA	PKC	P38	JNK
RAF	0	1	0	0	0	0	1	0.22	0.22	0	0.17
MEK	0	0	0	0	0	0	0	0	0.01	0	0
PCLg	0	0	0	0.92	0.9	0	0	0	0.33	0.28	0.31
PIP2	0	0	0	0.08	0	0.06	0	0	0	0	0
PIP3	0	0	0	0.1	0.94	0	0	0	0.16	0	0
ERK	0	1	0	0	0	0	1	0.23	0.23	0	0.17
AKT	0	0	0	0	0	0	0	0	0	0	0
PKA	0.78	1	0	0	0	0.77	1	0	0.29	0.51	0.16
PKC	0.78	0	0.67	0	0.84	0.77	0	0.71	0	0.72	0.42
P38	0	0	0.72	0	0	0	0	0.49	0.27	0	0.27
JNK	0.83	0	0.69	0	0.01	0.83	0	0.84	0.58	0.73	0

	RAF	MEK	PCLg	PIP2	PIP3	ERK	AKT	PKA	PKC	P38	JNK
RAF	0	0	0	0	0	1	0	0	0.99	0	0
MEK	1	0	0.23	0	0	1	0	1	0.01	0	0.35
PCLg	0	0.77	0	0.98	0.97	0	0.66	0	0	0	0.75
PIP2	0	0	0.02	0	0.02	0	0	0	0	0	0
PIP3	0	0	0.03	0.98	0	0	0	0	0	0	0.04
ERK	0	0	0	0	0	0	0	1	0	0	0
AKT	1	0	0.34	0	0	1	0	0	0	0	0.46
PKA	0	0	0	0	0	0	0	0	1	1	0
PKC	0	0	0	0	0.96	0	0	0	0	1	0
P38	0	0	0	0	0	0	0	0	0	0	0
JNK	1	0.65	0.25	0	0	0	0.54	1	1	1	0

	RAF	MEK	PCLg	PIP2	PIP3	ERK	AKT	PKA	PKC	P38	JNK
RAF	0	0.33	0	0	0	0.27	0.26	0	0.99	0	0
MEK	0.67	0	0.85	0	0	0.46	0.33	0.79	0.98	0	1
PCLg	0	0.15	0	1	1	0	0.08	0.18	0	0	0
PIP2	0	0	0	0	0	0	0	0	0	0	0
PIP3	0	0	0	1	0	0	0	0	0	0	0
ERK	0.73	0.54	0	0	0	0	0.4	0.69	0.02	0	0
AKT	0.74	0.67	0.9	0	0	0.6	0	0.87	0.01	0	0
PKA	0	0.21	0.76	0	0	0.31	0.13	0	1	1	1
PKC	0	0	0	0	1	0	0	0	0	1	1
P38	0	0	0.07	0	0	0	0	0	0	0	0
JNK	0	0	0	0	0	0	0	0	0	1	0

	RAF	MEK	PCLg	PIP2	PIP3	ERK	AKT	PKA	PKC	P38	JNK
RAF	0	0.29	0	0	0	0	1	0.43	0.4	0	0
MEK	0.71	0	1	0	0	0.93	0	0.61	0.56	0.08	0.58
PCLg	0	0	0	1	1	0	0	0.01	0	0	0
PIP2	0	0	0	0	0	0	0	0	0	0	0
PIP3	0	0	0	1	0	0	0	0	0	0	0
ERK	0	0.07	0	0	0	0	1	0.1	0.09	0	0
AKT	0	0	0.98	0	0	0	0	0	0	0	0
PKA	0.57	0.39	0.8	0	0	0.9	1	0	0.29	0.76	0.64
PKC	0.6	0.44	0	0	1	0.91	0	0.71	0	0.79	0.7
P38	0	0.02	0.2	0	0	0	0	0.24	0.21	0	0.21
JNK	0	0.32	0	0	0	0	0	0.36	0.3	0.79	0

	RAF	MEK	PCLg	PIP2	PIP3	ERK	AKT	PKA	PKC	P38	JNK
RAF	0	0.31	0	0	0	0	0.28	0.29	0	1	0
MEK	0.69	0	0.8	0	0	0.58	0.49	0.76	0	0	1
PCLg	0	0.2	0	1	1	0	0.2	0.25	0	0	0
PIP2	0	0	0	0	0	0	0	0	0	0	0
PIP3	0	0	0	1	0	0	0	0	0	0	0
ERK	0.72	0.42	0	0	0	0	0.38	0.57	0	0	0
AKT	0.71	0.51	0.78	0	0	0.62	0	0.76	0	0	0
PKA	0	0.24	0.7	0	0	0.33	0.24	0	1	1	1
PKC	0	0	0	0	1	0	0	0	0	1	0
P38	0	0	0.05	0	0	0	0	0	0	0	0
JNK	0	0	0	0	0	0	0	0	1	1	0

	RAF	MEK	PCLg	PIP2	PIP3	ERK	AKT	PKA	PKC	P38	JNK
RAF	0	0.14	0	0	0	0	1	0.2	0.22	0	0
MEK	0.86	0	0	0	0	0.84	0	0.44	0.55	0	0.55
PCLg	0	0	0	1	1	0	0	0.23	0.15	0.15	0
PIP2	0	0	0	0	0	0	0	0	0	0	0
PIP3	0	0	0	1	0	0	0	0	0.01	0	0
ERK	0.01	0.15	0	0	0	0	1	0.22	0.27	0	0
AKT	0	0	0	0	0	0	0	0	0	0	0
PKA	0.78	0.56	0	0	0	0.78	1	0	0.54	0.63	0.56
PKC	0.78	0.45	0.77	0	0.99	0.73	0	0.46	0	0.68	0.56
P38	0	0	0.85	0	0	0	0	0.37	0.32	0	0.16
JNK	0	0.45	0.85	0	0	0	0	0.44	0.44	0.84	0

	RAF	MEK	PCLg	PIP2	PIP3	ERK	AKT	PKA	PKC	P38	JNK
RAF	0	0.03	0	0	0	0	1	0.05	0.06	0	0
MEK	0.97	0	0	0	0	0.97	0.01	0.13	0.16	0	0.15
PCLg	0	0	0	0.84	0.84	0	0	0	0.66	0.56	0.54
PIP2	0	0	0.16	0	0.17	0	0	0	0	0	0
PIP3	0	0	0.16	0.83	0	0	0	0	0	0.38	0
ERK	0	0.03	0	0	0	0	1	0.06	0.06	0	0
AKT	0	0	0	0	0	0	0	0	0	0	0
PKA	0.95	0.87	0	0	0	0.94	0.99	0	0.25	0.23	0.16
PKC	0.94	0.84	0.34	0	0.62	0.94	0	0.75	0	0.64	0.55
P38	0	0	0.44	0	0	0	0	0.77	0.36	0	0.38
JNK	0	0.85	0.46	0	0	0	0	0.84	0.45	0.62	0

FIGURE C.12: From left to right top to bottom: Posterior edges probabilities calculated for chains from one to twelve produced by the NS sampler to learn Raf-Signaling network.

## C.5 Inferring the KEGG Pathways network

### C.5.1 More simulation run using the MH sampler

This section investigates the performance of the MH sampler to learn the KEGG Pathways network when the number of iteration increased to one million. The networks learned by the GES and HCS are set as initial networks to run two Markov chains using one million iteration each. Figure C.13 plots the log posterior values produced by each chain against one million iteration. The highest log posterior values achieved by the first chain and second chain are  $-1653.58$  and  $-1695.43$ , respectively, which are less far from the highest values achieved by the NS and HAR.

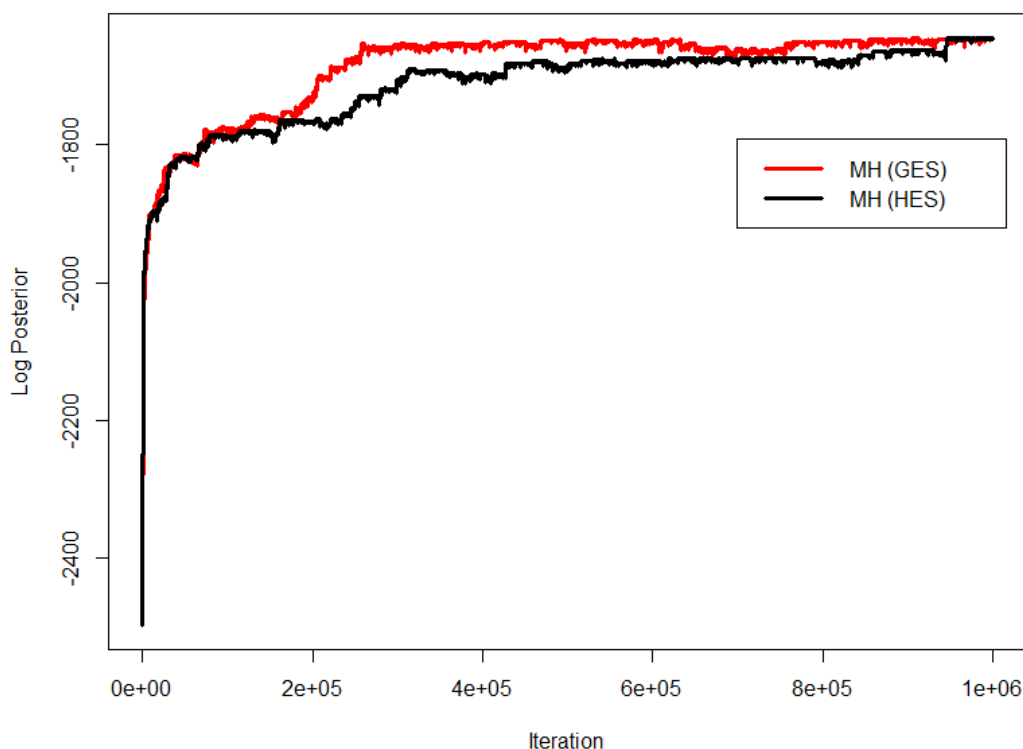


FIGURE C.13: MH: Log posterior for two chains of one million iterations each.

### C.5.2 Figures of best scoring networks

Figure C.14, Figure C.15, Figure C.16, and Figure C.17 plot the BNs with best scoring values learned by the HAR (GES), HAR (HCS), NS (HCS), and NS (GES), respectively. The common edges among the best networks are summarised in Figure 7.39 in Chapter 7.

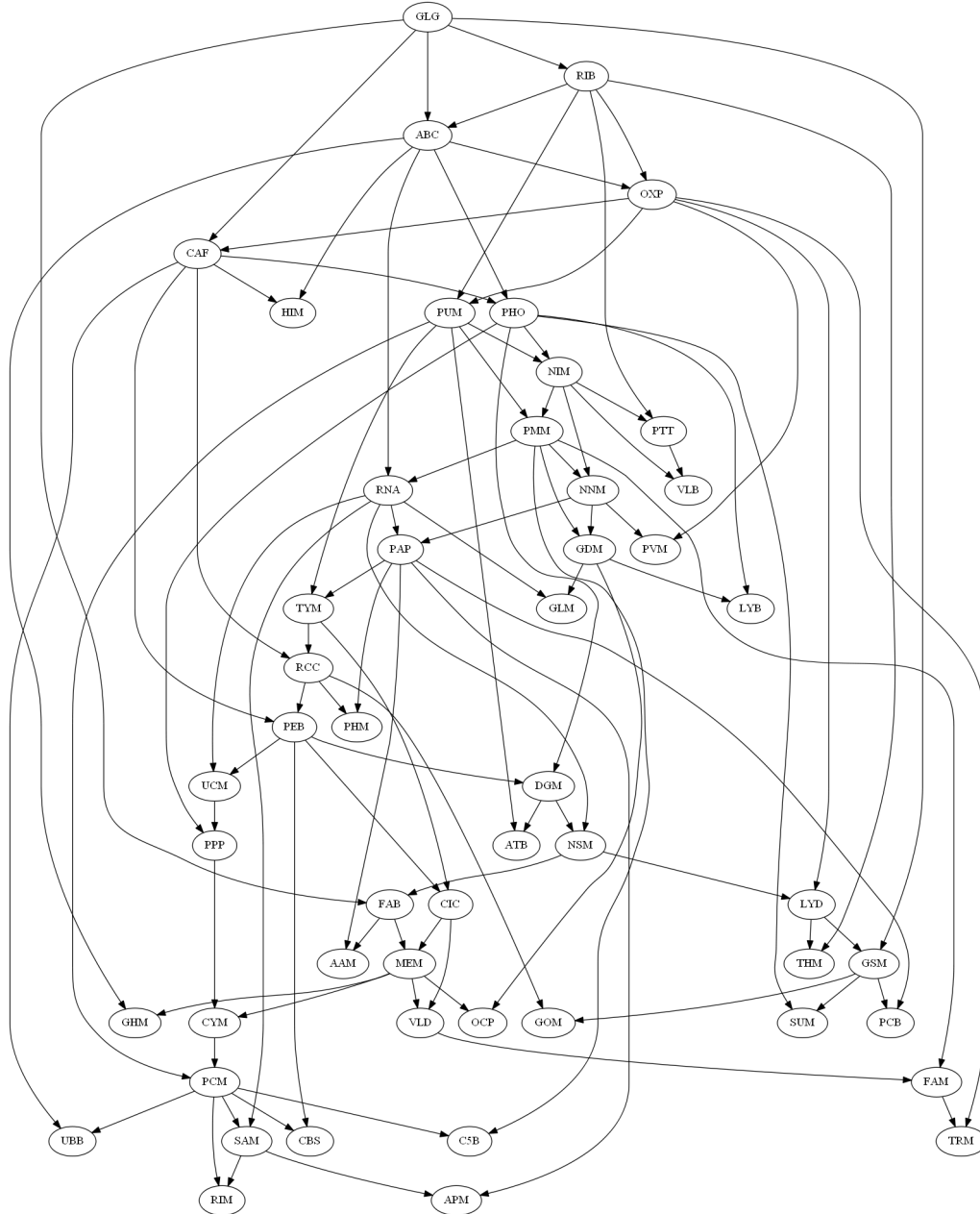


FIGURE C.14: Best scoring network learned by the HAR when the network learned by the GES is initialised.

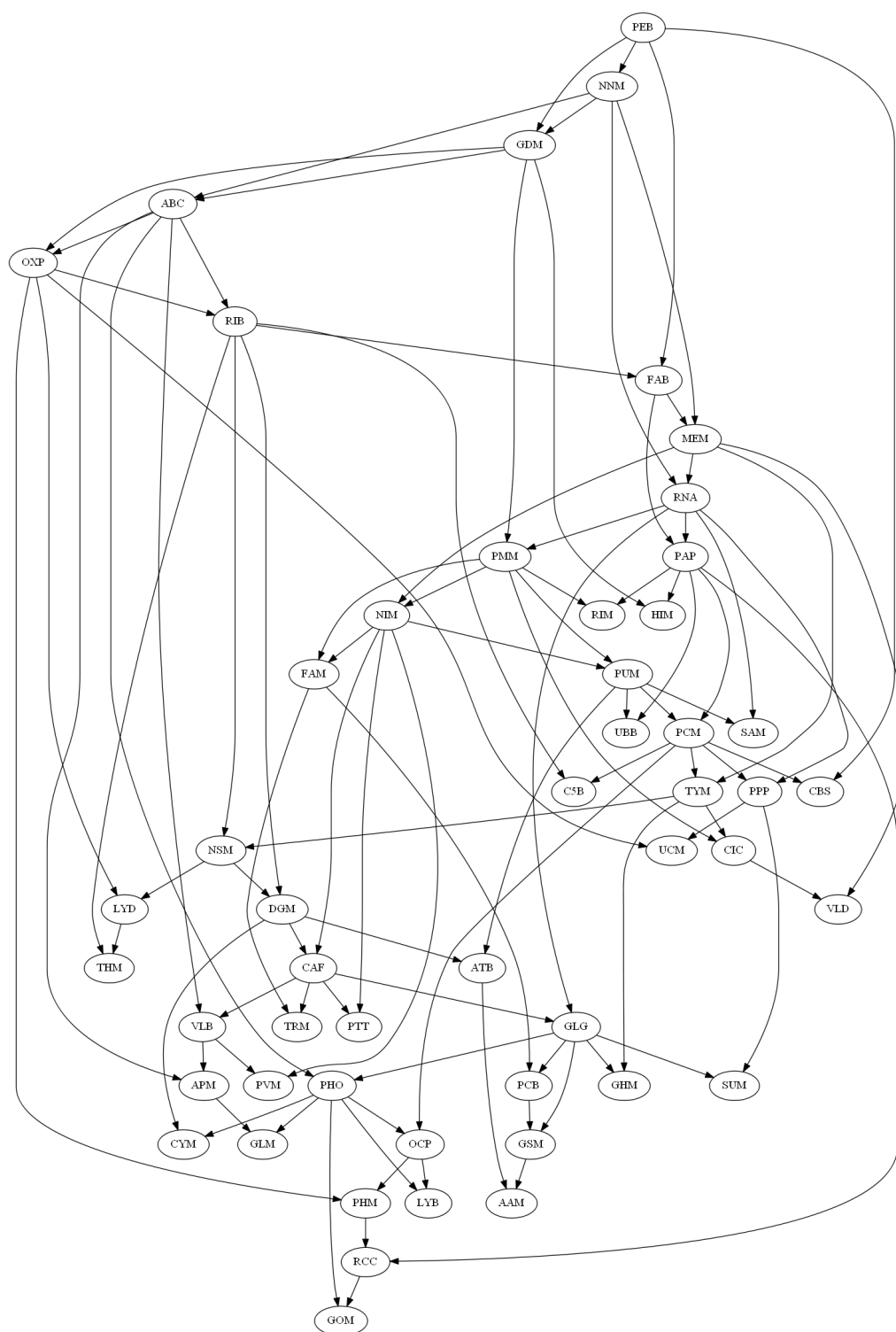


FIGURE C.15: Best scoring network learned by the HAR when the network learned by the HCS is initialised.



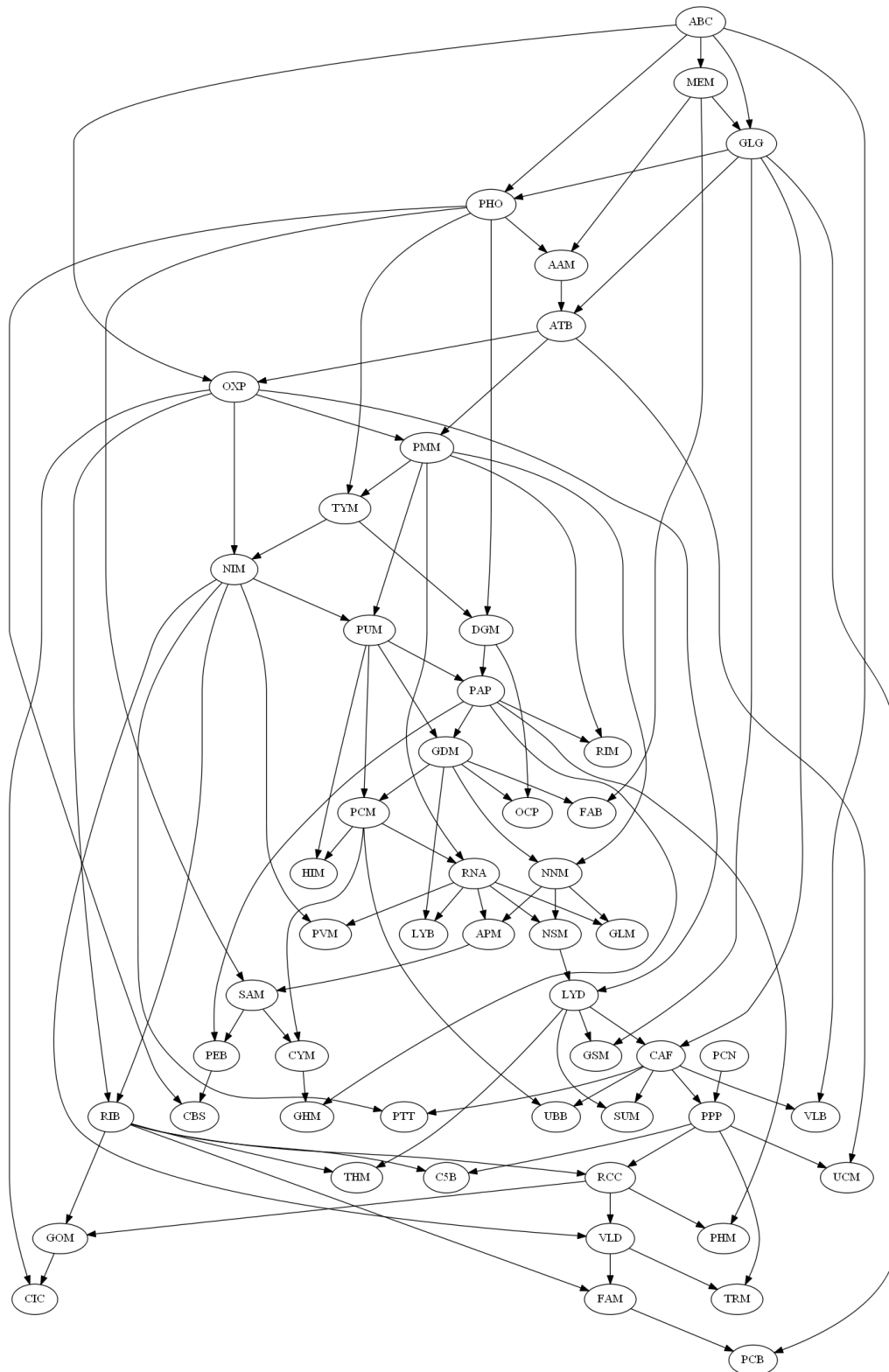


FIGURE C.16: Best scoring network learned by the NS when the network learned by the HCS is initialised.

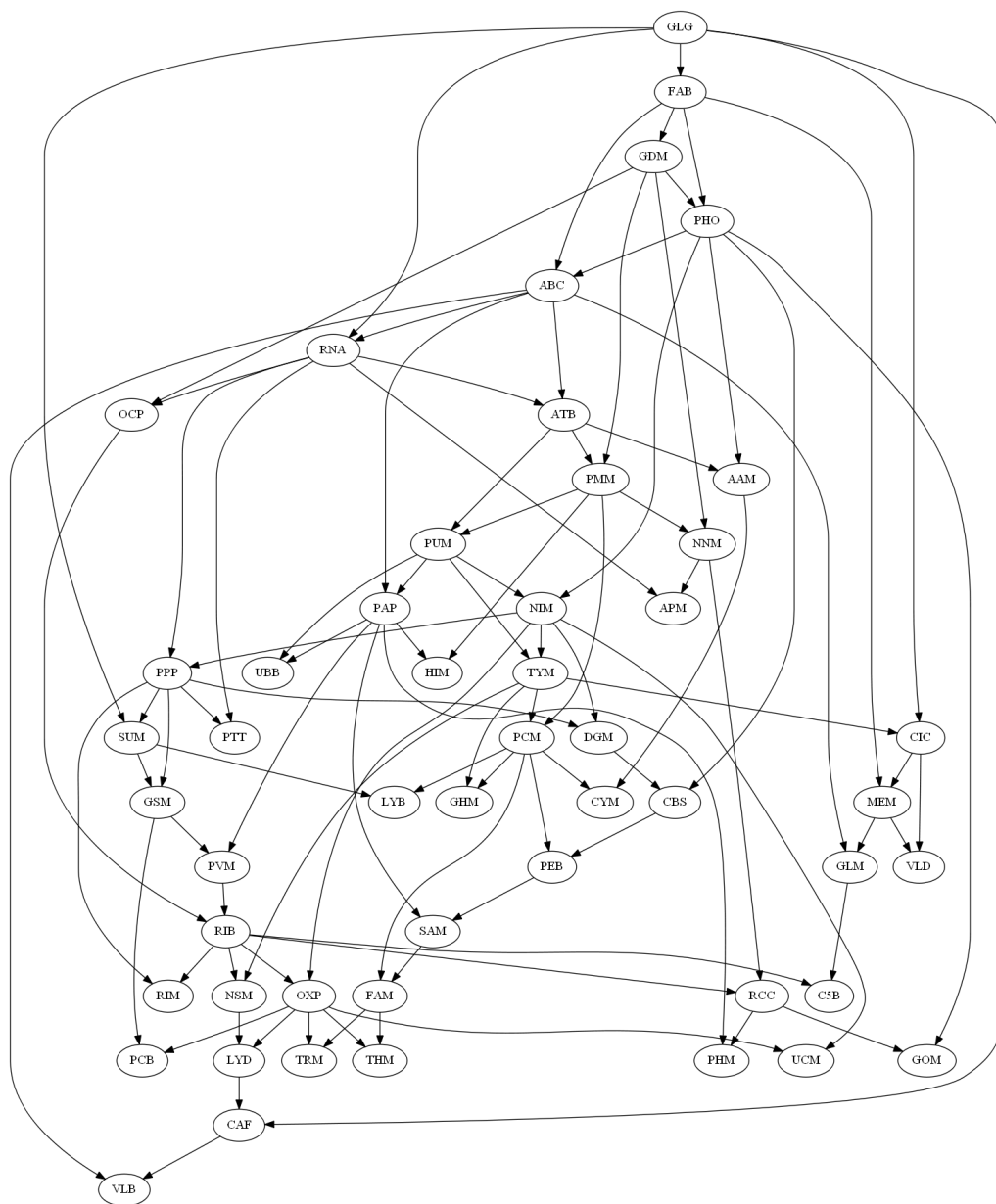


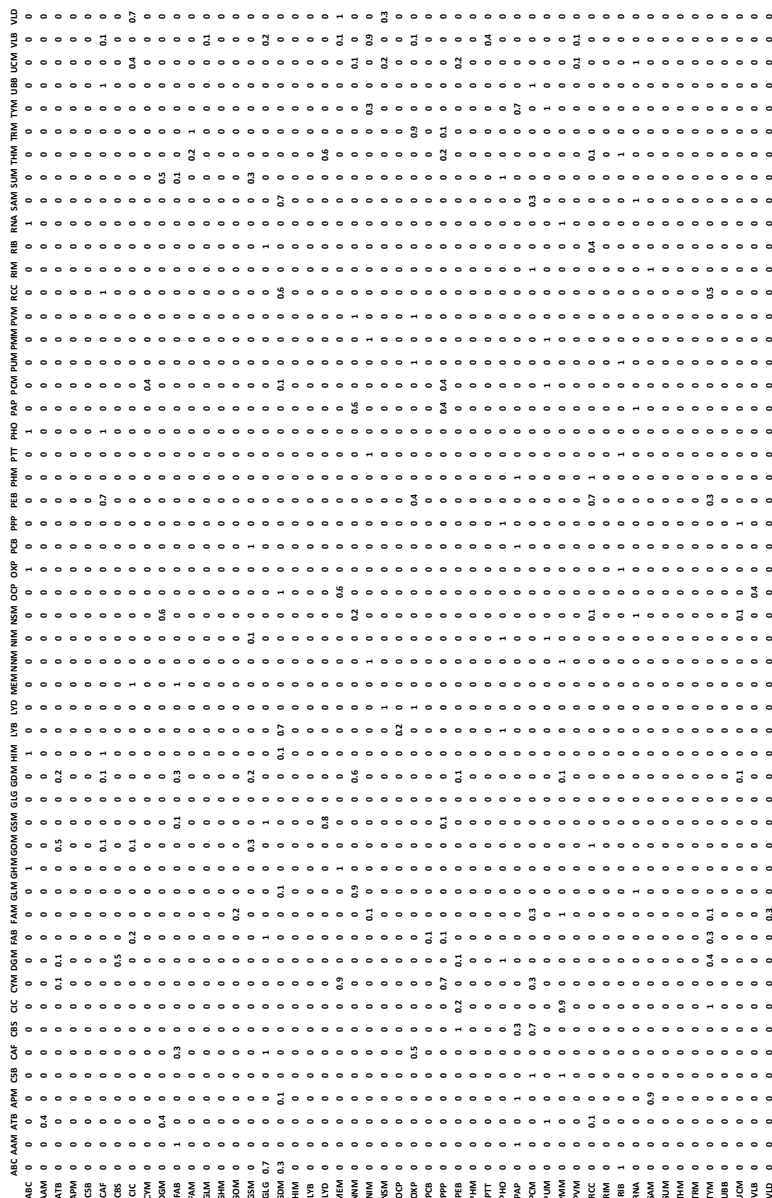
FIGURE C.17: Best scoring network learned by the NS when the network learned by the GES is initialised.

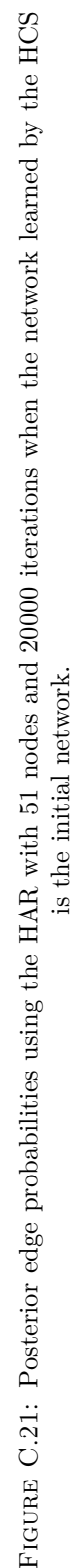
Figure C.18, Figure C.19, Figure C.20 and Figure C.21 provide all the posterior edge probabilities of Figure 7.35, Figure 7.36, Figure 7.37 and Figure 7.38 including probabilities less than 50%.



[illegible]

FIGURE C.19: Posterior edge probabilities using the NS with 51 nodes and 20000 iterations when the network learned by the HCS is the initial network.





# Appendix D

## Conclusion

### D.1 Sampling CUDGs uniformly

To construct the adjacent graphs  $\mathcal{N}_G$  for a particular CUDG, a sampler considers all possible edges that can be added or deleted while preserving the condition of connectivity. (Naturally any missing edge can be added, since adding an edge will never destroy connectivity) All of these valid adjacent graphs with the original graph itself are defined as neighbourhoods  $\mathcal{N}_G$  of  $G$ . Figure D.1 shows how all possible addable and deletable edges are identified to obtain the corresponding adjacent graphs of a particular initial CUDG.

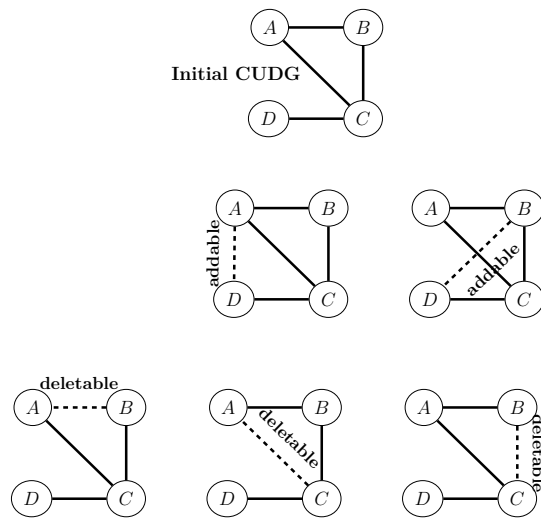


FIGURE D.1: All possible addable edges, and all possible deletable edges given an initial graph.

For CUDGs, I considered the space of all such graphs with four nodes; this graph space consists of only 38 CUDGs. I applied the NS to sample from this graph space. Consistently, this entire graph space can be visited at least once in only 250 iterations. I next considered the larger but still feasible space of CUDGs with five nodes; there are 728 such CUDGs. This entire graph space is typically explored using only 5000 iterations and an execution time of less than a second. The plots shown in Figure D.2 compare the sampled frequencies to the uniform distribution as the number of iterations grows. The SSDs at 20 000, 200 000, and 2 000 000 iterations are 0.00006828, 0.00000726 and 0.00000068. It is clear that the SSDs are converging to 0 as the number of iterations increases, validating both the algorithm and the software.

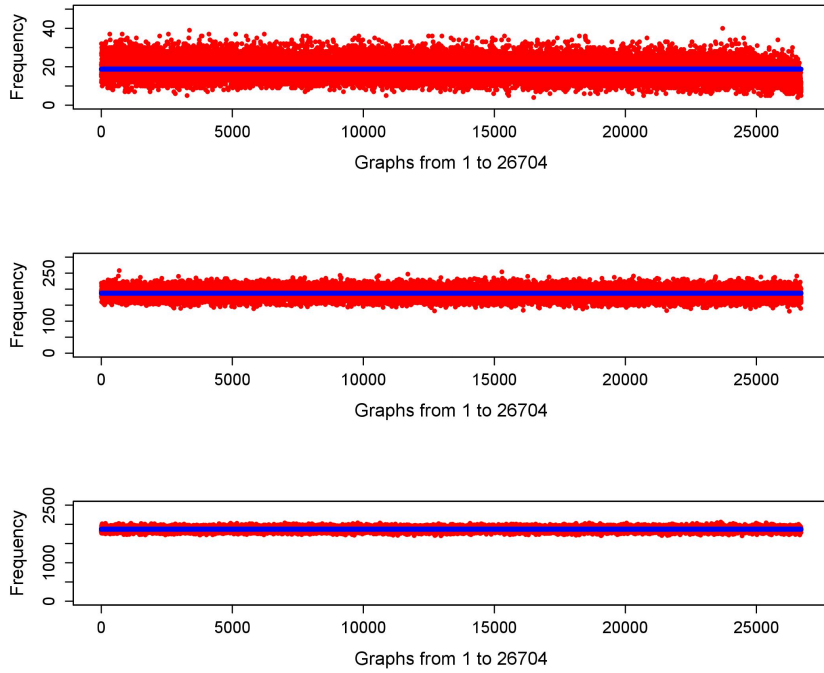


FIGURE D.2: Neighbourhood Sampler (Red) vs Uniform Distribution (Blue) for CUDGs. From top to bottom: with six nodes using 500 000, 5 000 000 and 50 000 000 iterations, respectively.



# Bibliography

1. Keith, J. M., Sofronov, G. Y. & Kroese, D. P. The generalised Gibbs sampler and the neighborhood sampler. In *Monte Carlo and Quasi-Monte Carlo Methods 2006. Springer Berlin Heidelberg* **31**, 537–547 (2008).
2. Lauritzen, S. L. & Spiegelhalter, D. J. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B (Methodological)* **50**, 15–224 (1988).
3. Pearl, J. *Probabilistic reasoning in intelligent systems: networks of plausible inference* 1st (Morgan Kaufmann Publishers Inc., 1988).
4. Zhang, N. L. & Poole, D. A simple approach to Bayesian network computations. In *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, 171–178 (1994).
5. Park, J. D. & Darwiche, A. Solving mAP exactly using systematic search. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI03)*, 403–410 (2003).
6. Tanner, M. A. *Tools for statistical inference: Methods for the exploration of posterior distributions and likelihood functions* (New York: Springer-Verlag, 1993).
7. Smith, A. F. M. & Roberts, G. O. Bayesian computation via the Gibbs sampler and related Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society. Series B* **55**, 3–23 (1993).

8. Gilks, W. R., Richardson, S. & Spiegelhalter, D. J. *Markov chain Monte Carlo in practice* (Boca Raton, FL: Chapman and Hall, 1996).
9. Gilks, W. & Roberts, G. "Strategies for improving MCMC." In W Gilks, S Richardson, D Spiegelhalter (eds). *Markov Chain Monte Carlo in Practice*, Chapman and Hall, Boca Raton, FL. 89–114 (1996).
10. Brooks, S. P. Markov chain Monte Carlo method and its application. *Journal of the Royal Statistical Society* **47**, 69–100 (1998).
11. Chen, M. H., Shao, Q. M. & Ibrahim, J. G. *Monte Carlo methods in Bayesian computation* (New York: Springer-Verlag, 2000).
12. Liu, J. S. *Monte Carlo strategies in scientific computing* 1st (Springer, 2001).
13. Robert, C. P. & Casella, G. *Monte Carlo statistical methods* (Springer, 2004).
14. Robert, C & Casella, G. A short history of Markov chain Monte Carlo: Subjective recollections from incomplete data. *Statistical Science* **26**, 102–115 (2011).
15. Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. & Teller, E. Equations of state calculations by fast computing machines. *Journal of Chemical Physics* **21**, 1087–1092 (1953).
16. Markov, A. A. Extension of the law of large numbers to dependent events. *svestia Soc. Phys. Math. Kazan* **15**, 135–156 (1906).
17. Hastings, W. K. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* **57**, 97–109. ISSN: 1464-3510 (1970).
18. Ross, S. M. *Stochastic processes* 2nd (John Wiley and Sons, 1980).
19. Tierney, L. Markov chains for exploring posterior distributions. *Annals of Statistics* **22**, 1701–1762 (1994).

20. Bremaud, P. *Markov chains: Gibbs fields, Monte Carlo simulation, and queues* (Springer, 2008).
21. Meyn, S. & Tweedie, R. L. *Markov chains and stochastic stability* 2nd (Cambridge University Press, 2009).
22. Wilmer, E. L., Levin, D. A. & Peres, Y. *Markov chains and mixing times* ISBN: 978-0-8218-4739-8 (American Mathematical Society, 2009).
23. Gardiner, C. *Handbook of stochastic methods: for physics, chemistry and the natural sciences* 3th. ISBN: 3540208828 (Springer, 2004).
24. Allen, L. J. S. *An introduction to stochastic processes with applications to biology* 2th. ISBN: 1-4398-1882-7 (Chapman and Hall, 2010).
25. Krishnamurthy, V. *Partially observed Markov decision processes* (Cornell University, 2016).
26. Breuer, L. & Baum, D. *An introduction to queueing theory and matrix-analytic methods* (Springer Netherlands, 2005).
27. Kulkarni, V. G. *Modeling and analysis of stochastic systems* 3rd (Chapman and Hall/CRC, 2016).
28. Häggström, O. *Finite Markov chains and algorithmic applications* 1st (London Mathematical Society Student Texts, 2002).
29. Kelly, F. P. Reversibility and stochastic networks. **2B**, 21–25 (1979).
30. Neumann, J. V. Various techniques in connection with random digits. *National Bureau of Standard Applied Mathematics Series* **12**, 36–38 (1951).
31. Devroye, L. Non-uniform random variate generation. *Springer* (1986).
32. Gilks, W. R. & Wild, P. Adaptive rejection sampling for Gibbs sampling. *J. R. Stat. Soc., Ser. C Appl. Stat.* **41**, 337–348 (1992).

33. Gilks, W. R. Derivative-free adaptive rejection sampling for Gibbs sampling. *Bayesian Stat.* **4**, 641–649 (1992).
34. Gilks, W. R., Best, N. G. & Tan, K. K. C. Adaptive rejection Metropolis sampling within Gibbs sampling. *Applied Statistics* **44**, 455–472 (1995b).
35. Martino, L. & Míguez, J. A generalization of the adaptive rejection sampling algorithm. *Statistics and Computing* **21**, 633–647 (2011).
36. Chen, Y. Another look at rejection sampling through importance sampling. *Statistics and Probability Letters* **72**, 277–283 (2005).
37. Marshall, A. W. "The use of multi-stage sampling schemes in Monte Carlo computations" In *Meyer, M.A. (Ed.), Symposium on Monte Carlo Methods*. Wiley, 123140 (1956.).
38. Liu, J. S. Metropolized independent sampling with comparisons to rejection sampling and importance sampling. *Statistics and Computing* **6**, 113–119 (1996a).
39. Kirkpatrick, S., Gelatt Jr, C. D. & Vecchi, M. P. Optimization by Simulated Annealing. *Science* **220**, 671–680 (1983).
40. Dowsland, K. A. "Simulated annealing. In *modern heuristic techniques for combinatorial problems* Reeves (McGraw-Hill, 1995).
41. Van Laarhoven, P. J. M. & Aarts, E. H. L. *Simulated annealing: theory and applications* (Springer, 1988).
42. Aarts, E. H. L. & Korst, J. H. M. *Simulated annealing and Boltzmann machines* (John Wiley and Sons, Ltd, 1989).
43. Rutenbar, R. A. Simulated annealing algorithms : an overview. *IEEE Circuits and Devices Magazine* **5**, 19–26 (1989).

- 
44. Metropolis, N. & Ulam, S. The Monte Carlo method. *J. Amer. Statist. Assoc.* **44**, 335–341 (1949).
  45. Gelman, A., Carlin, J. B., Stern, H. S. & Rubin, D. B. *Bayesian data analysis* 2nd (Chapman and Hall/CRC, 2003).
  46. Congdon, P. *Bayesian statistical modeling* (John Wiley and Sons, 2001).
  47. Congdon, P. *Applied Bayesian modeling* (John Wiley and Sons, 2003).
  48. Congdon, P. *A Bayesian models for categorical data* (John Wiley and Sons, 2005).
  49. Haario, H., Saksman, E. & Tamminen, J. An adaptive Metropolis algorithm. *Bernoulli* **7**, 223–242 (2001).
  50. Diaconis, P. & Saloff-Coste, L. What do we know about the Metropolis algorithm? *Journal of Computer and System Sciences* **57**, 20–36 (1998).
  51. Casella, G. & George, E. Explaining the Gibbs sampler. *The American Statistician* **46**, 167–174 (1992).
  52. Gelman, A. *Iterative and non-iterative simulation algorithms* Technical Report 347 (University of California, Dept. of Statistics, 1992).
  53. Chib, S. & Greenberg, E. Understanding the Metropolis-Hastings algorithm. *The American Statistician* **49**, 327–335 (1995).
  54. Liu, J. S. *Monte Carlo strategies in scientific computing* (2004).
  55. Roberts, G. O., Gelman, A. & Gilks, W. R. Weak convergence and optimal scaling of random walk Metropolis algorithms. *Ann. Appl. Probab.* **7**, 110–120 (1997).
  56. Schilling, R. L. *Measures, integral and martingales* (Cambridge University Press, 2005).

- 
57. Turchin, V. F. On the computation of multidimensional integrals by the Monte Carlo method. *Theory of Probability and its Applications* **16**, 720–724 (1971).
  58. Smith, R. L. Efficient Monte-Carlo procedures for generating points uniformly distributed over bounded regions. *Operations Research* **32**, 1296–1308 (1984).
  59. Hall, B. **LaplacesDemon**: Software for Bayesian inference. *R package version 12.05.07*, URL <http://cran.r-project.org/web/packages/LaplacesDemon/index.html>. 2012.
  60. Lovasz, L. Hit-and-Run mixes fast. *Mathematical Programming* **86**, 443–461 (1999).
  61. Lovasz, L. & Vempala, S. *Hit-and-Run is fast and fun* Technical Report (Microsoft Research, MSR-TR-2003-05, 2003).
  62. Chen, M. & Schmeiser, B. Performance of the Gibbs, Hit-and-Run and Metropolis samplers. *Journal of Computational and Graphical Statistics* **2**, 251–272 (1992).
  63. Boneh, A. & Golan, A. *Constraints redundancy and feasible region boundedness by random feasible point generator (RFPG)* in (Amsterdam, 1979).
  64. Romeijn, H. E. & Smith, R. L. *Sampling through random walks* Technical Report (The University of Michigan, Department of Industrial and Operations Engineering, 1990).
  65. Schmeiser, B. & Chen, M. *On Hit-and-run Monte Carlo sampling for evaluating multidimensional integrals* <<https://books.google.com.au/books?id=u9TntgAACAAJ>> (Purdue University, Department of Statistics, 1991).
  66. Kroese, D. P., Taimre, T. & Botev, Z. I. *Handbook of Monte Carlo methods* ISBN: 0-470-17793-4 (New York: John Wiley and Sons, 2011).
  67. Webb, A. R. & Copsey, K. D. *Statistical pattern recognition* 3rd (WILEY, 2011).

- 
68. Gelfand, A. E. & Smith, A. F. M. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association* **85**, 398–409 (1990).
  69. Gelman, A. & Rubin, D. B. Inference from iterative simulation using multiple sequences. *Statistical Science* **7**, 457–472 (1992).
  70. Geyer, C. J. Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments. In *Bayesian Statistics 4* (eds J. M. Bernardo, J. Berger, A. P. Dawid and A. F. M. Smith), 169–193 (1992).
  71. Klein, B. M. *State space models for exponential family data* PhD thesis (Department of Statistics, University of Southern Denmark, 2003).
  72. Gilks, W. R. & Roberts, G. O. Strategies for improving MCMC. In *Markov chain Monte Carlo in practice* (eds W. R. Gilks, S. Richardson and D. J. Spiegelhalter), 89–114 (1995).
  73. Roberts, G. O. Markov chain concepts related to sampling algorithms. In *Markov chain Monte Carlo in practice* (eds W. R. Gilks, S. Richardson and D. J. Spiegelhalter), 45–57 (1995).
  74. Fox, J. *Bayesian item response modeling: theory and applications* (New York: Springer, 2010).
  75. Brooks, S. P. & Morgan, B. J. T. Automatic starting point selection for function optimisation. *Statist. Comput.* **4**, 173–177 (1994).
  76. Bryan, F. J. M. *Randomization and Monte Carlo methods in biology* 233–258. ISBN: 978-1-4899-2995-2. doi:[10.1007/978-1-4899-2995-2\\_11](https://doi.org/10.1007/978-1-4899-2995-2_11) (Springer, 1991).
  77. Link, W. A. & Eaton, M. J. On thinning of chains in MCMC. *Methods in Ecology and Evolution* **3**, 112–115 (2012).

- 
78. Raftery, A. E. & Lewis, S. M. Implementing MCMC. In *Markov chain Monte Carlo in practice* (eds W. R. Gilks, S. Richardson and D. J. Spiegelhalter), 115–130 (1995).
  79. Atchade, Y., Fort, G., Moulines, E. & Priouret, P. Adaptive Markov chain Monte Carlo: theory and methods, chapter 2. In *Bayesian Time Series Models*, 32–51 (2011).
  80. Swendsen, R. H. & Wang, J. S. Non-universal critical dynamics in Monte Carlo simulation. *Physical Review Letters* **58**, 86–88. (1987).
  81. Merrilee, H. Difficulties in the use of auxiliary variables in Markov chain Monte Carlo methods. *Statistics and Computing* **7**, 35–44 (1997).
  82. Geweke, J. Evaluating the accuracy of sampling based approaches to the calculation of posterior moments. *Bayesian Statistics* **4**, 169–193 (1992).
  83. Cowles, M. K. & Carlin, B. P. Markov chain Monte Carlo convergence diagnostics: a comparative review. *J. Amer. Statist. Assoc.* **91**, 883–904 (1996).
  84. Brooks, S. P. & Gelman, A. General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics* **7**, 434–455 (1997).
  85. Plummer, M., Best, N., Cowles, K. & Vines, K. CODA: Convergence diagnostic and output analysis for MCMC. *R News* **6**, 7–11 (2006).
  86. Smith, B. J. boa: An R package for MCMC output convergence assessment and posterior inference. *Journal of Statistical Software* **21**, 1–37 (2007).
  87. Heidelberger, P. & Welch, P. Simulation run length control in the presence of an initial transient. *Operations Research* **31**, 1109–1144 (1983).
  88. Neapolitan, R. E. *Learning Bayesian networks* (Prentice-Hall, Inc., 2003).
  89. Bishop, C. M. *Pattern recognition and machine learning* 1st (Springer, 2006).



- 
90. Mittal, A. *Bayesian network technologies: applications and graphical models* 2nd (IGI Global, 2007).
  91. Barber, D. *Bayesian reasoning and machine learning* 1st (Cambridge University Press, 2012).
  92. Kjaerulff, U. B. & Madsen, A. L. *Bayesian networks and influence diagrams: a guide to construction and analysis* 2nd (Springer, 2013).
  93. Gelman, A., Carlin, J. B., Stern, H. S. & Rubin, D. B. *Bayesian data analysis* 3rd (Chapman and Hall/CRC, 2013).
  94. Friedman, N., Linial, M., Nachman, I. & Peer, D. Using Bayesian networks to analyze expression data. *Journal of Computational Biology* **7**, 601–620 (2000).
  95. Friedman, N. Inferring cellular networks using probabilistic graphical models. *Science* **303**, 799–805 (2004).
  96. Beinlich, I., Suermondt, H., Chavez, R. & Cooper, G. The ALARM Monitoring System: A case study with two probabilistic inference techniques for belief networks. In *Proceedings of the Second European Conference on Artificial Intelligence in Medicine (AIME 89). Lecture Notes in Medical Informatics* **38**, 247–256 (1989).
  97. Abramson, B., Brown, J., Edwards, W., Murphy, A. & Winkler, R. L. Hailfinder: A Bayesian system for forecasting severe weather. *International Journal of Forecasting* **12**, 57–71 (1996).
  98. Pearl, J. *Causality* 2nd (Cambridge University Press, 2009).
  99. Russell, S. J. & Norvig, P. *Artificial intelligence: a modern approach* 3rd (Prentice Hall, 2009).
  100. Pe’er, D. Bayesian network analysis of signalling networks: A primer. *Science Signaling* **281**, 14 (2005).

101. Chickering, D. M., Heckerman, D. & Meek, C. Large-sample learning of Bayesian networks is NP-hard. *Journal of Machine Learning Research* **5**, 287–1330 (2004).
102. Robinson, R. Counting labeled acyclic digraphs. In *New Directions in the Theory of Graphs*, 239–273 (1973).
103. Lee, C. Y. An algorithm for path connections and its applications. *Electronic Computers, IRE Transactions. EC* **10**, 346–465 (1961).
104. Heckerman, D., Geiger, D. & Chickering, D. M. Learning Bayesian networks: the combination of knowledge and statistical data. *Machine Learning* **20**, 197–243 (1995).
105. Riggelsen, C. *MCMC learning of Bayesian network models by Markov blanket decomposition* in *Proceedings of the 16th European Conference on Machine Learning* (Springer-Verlag, Porto, Portugal, 2005), 329–340. ISBN: 3-540-29243-8, 978-3-540-29243-2. doi:[10.1007/11564096\\_33](https://doi.org/10.1007/11564096_33). <[http://dx.doi.org/10.1007/11564096\\_33](http://dx.doi.org/10.1007/11564096_33)>.
106. De Campos, L. M. & Castellano, J. G. Bayesian network learning algorithms using structural restrictions. *International Journal of Approximate Reasoning* **45**, 233–254 (2007).
107. Ebert-Uphoff, I. *Measuring connection strength and link strengths in discrete Bayesian networks* Technical Report (Georgia Institute of Technology, 2007).
108. Ram, R. & Chetty, M. *MCMC based Bayesian inference for modeling gene networks* in *Proceedings of the 4th IAPR International Conference on Pattern Recognition in Bioinformatics* (Springer-Verlag, Sheffield, UK, 2009), 293–306. ISBN: 978-3-642-04030-6. doi:[10.1007/978-3-642-04031-3\\_26](https://doi.org/10.1007/978-3-642-04031-3_26). <[http://dx.doi.org/10.1007/978-3-642-04031-3\\_26](http://dx.doi.org/10.1007/978-3-642-04031-3_26)>.

- 
109. Schmidt, J. M. A simple test on 2-vertex and 2-edge-connectivity. *Information Processing Letters* **113**, 241–244 (2013).
  110. Friedman, N., Nachman, I. & Pe’er, D. *Learning Bayesian network structure from massive datasets: the sparse candidate algorithm* in *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence* (Stockholm, Sweden, 1999), 206–215.
  111. De Campos, L. M. & Castellano, J. G. *On the use of restrictions for learning Bayesian networks* in *Godo L. (eds) Symbolic and Quantitative Approaches to Reasoning Uncertainty. Lecture Notes in Computer Science* **3571** (Springer, Berlin, Heidelberg, 2005), 174–185.
  112. Bollobas, B. *Modern graph theory* (New York, Springer-Verlag, 1998).
  113. Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. *Introduction to algorithms* 2nd (MIT Press and McGraw-Hill, 2001).
  114. Opsah, T., Agneessens, F. & Skvoretz, J. Node centrality in weighted networks: Generalizing degree and shortest paths. *Methodology And Computing In Applied Probability* **32**, 245 (2010).
  115. Frigyik, B. A., A., K. & Gupta, M. R. *Introduction to the Dirichlet distribution and related processes* Technical Report (University of Washington, 2010).
  116. Hrycej, T. Gibbs sampling in Bayesian networks. *Artificial Intelligence* **46**, 351–364 (1990).
  117. Cousins, S. B., Chena, W. & Frisse, M. E. A tutorial introduction to stochastic simulation algorithms for belief networks. *Artificial Intelligence in Medicine* **5**, 315–340 (1993).
  118. Dagum, P. & Horvitz, E. A Bayesian analysis of simulation algorithms for inference in belief networks. *Networks* **23**, 499–516 (1993).

- 
119. Dagum, P. & Luby, M. An optimal approximation algorithm for Bayesian inference. *Artificial Intelligence* **93**, 1–27 (1997).
  120. Liang, F. & Zhang, J. Learning Bayesian networks for discrete data. *Computational Statistics and Data Analysis* **53**, 865–876 (2009).
  121. Madigan, D. & York, J. Bayesian graphical models for discrete data. *International Statistical Review* **63**, 215–232 (1995).
  122. Giudici, P. & Castelo, R. Improving Markov chain Monte Carlo model search for data mining. *Machine Learning* **50**, 127–158 (2003).
  123. Friedman, N. & Koller, D. Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning* **50**, 95–125 (2003).
  124. Ellis, B. & Wong, W. H. Learning causal Bayesian network structures from experimental data. *Journal of the American Statistical Association* **103**, 778–789 (2008).
  125. Niinimäki, T., Parviainen, P. & Koivisto, M. *Partial order MCMC for structure discovery in Bayesian networks* in *Proceeding of the Twenty-Seventh International Joint Conference on Uncertainty in Artificial Intelligence* (2012).
  126. Grzegorzczak, M. & Husmèreier, D. Improving the structure MCMC sampler for a Bayesian networks by introducing a new edge reversal move. *Machine Learning* **71**, 265–305 (2008).
  127. Koivisto, M. & Sood, K. Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research* **5**, 549–573 (2004).
  128. Masegosa, R. & Moral, S. New skeleton-based approaches for Bayesian structure learning of Bayesian networks. *Applied Soft Computing* **13**, 1110–1120 (2013).

129. Ramoni, M. & Sebastiani, P. *Learning conditional probabilities from incomplete datasets: an experimental comparison* in *Proceeding of the Seventh International Workshop on Artificial Intelligence and Statistics* (Heckerman, D. and Whittaker, J. (eds). Morgan Kaufmann, 1999).
130. Wong, M. L. & Guo, Y. Y. Learning Bayesian networks from incomplete datasets using a novel evolutionary algorithm. *Decision Support Systems* **45**, 368–383 (2008).
131. Koller, D. & Friedman, N. *Probabilistic graphical models: principles and techniques* (MIT Press, 2009).
132. Brownlee, J. *Clever algorithms: nature-inspired programming recipes* 1st. ISBN: 978-1446785065 (LULU, 2012).
133. Daly, R., Shen, Q. & Aitken, S. Learning Bayesian networks: approaches and issues. *The Knowledge Engineering Review* **26**, 99–157 (2011).
134. Castillo, L. P. & Wrobel, S. *A comparative study on methods for reducing myopia of hill-climbing search in multirelational learning* in *Proceedings of the Twenty-first International Conference on Machine Learning* **3571** (ACM, Banff, Alberta, Canada, 2004).
135. Chickering, D. M., Geiger, D. & Heckerman, D. Learning Bayesian networks: search methods and experimental results. In *Learning from Data: Artificial Intelligence and Statistics V* (eds Fisher, D. and Lenz, H.-J.) *Lecture Notes in Statistics* **112**, 112–128 (1996).
136. Montermanni, R., Moon, J. N. J. & Smith, D. H. An improved tabu search algorithm for the fixed-spectrum frequency-assignment problem. *IEEE Transactions on Vehicular Technology* **52**, 891–901 (2003).
137. Glover, F. Tabu Search - Part 1. *ORSA Journal on Computing* **1**, 190206 (1989).

- 
138. Glover, F. Tabu Search - Part 2. *ORSA Journal on Computing* **2**, 432 (1990).
139. He, Z., Wang, N. & Liu, R. in. Chap. The multi-mode capital-constrained net present value problem (Springer, 2015). ISBN: 978-3-310-05442-1.
140. Margaritis, D. *Learning Bayesian network model structure from data* PhD thesis (Carnegie Mellon University, 2003).
141. Margaritis, D. & Thrun, S. *Bayesian network induction via local neighborhoods* Technical Report (DTIC Document, 2000).
142. Burnett, M. Blocking brute force attacks. *UVA Computer Science* (2007).
143. Paar, C., Pelzl, J. & Preneel, B. *Understanding Cryptography: a textbook for students and practitioners* (Springer, 2010).
144. Sedgewick, R. & Wayne, K. *Bridge.java from 4.1 undirected graphs* 2016. [algs4.cs.princeton.edu/41graph/Bridge.java.html](http://algs4.cs.princeton.edu/41graph/Bridge.java.html).
145. Gilbert, E. N. Random graphs. *Annals of Mathematical Statistics* **30**, 1141–1144 (1959).
146. Erdős, P. & Renyi, A. On random graphs. *Publicationes Mathematicae* **6**, 290–297 (1959).
147. Jordan, J. The degree sequences and spectra of scale-free random graphs. *Random Structures Algorithms* **29**, 226–242 (2006).
148. Britton, T., Deijfen, M. & Martin-Lof, A. Generating simple random graphs with prescribed degree distribution. *J. Stat. Phys.* **124**, 1377–1397 (2006).
149. Newman, M. E. J. Random graphs with clustering. *Phys. Rev. Lett.* **103**, 058701 (2009).
150. Chen, N. & Olvera-Cravioto, M. Directed random graphs with given degree distributions. *Stoch. Syst.* **3**, 147–186 (2013).

- 
151. Xiang & Miller, T. A well-behaved algorithm for simulating dependence structure of Bayesian networks. *International Journal of Applied Mathematics* **1**, 923–932 (1999).
  152. Spirtes, P., Glymour, C. & Scheines, R. *Causation, Prediction, and Search* 2nd. The MIT Press. (Adaptive Computation and Machine Learning, 2000).
  153. Bollobas, B. *Random graphs* 2nd (Cambridge University Press, 2001).
  154. Pinto, P. C., Nagele, A., Dejori, M., Runkler, T. A. & ao M.C. Sousa, J. Using a local discovery ant algorithm for bayesian network structure learning. *IEEE Transactions on Evolutionary Computation* **13**, 767–779 (2009).
  155. Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. *Introduction to algorithms* 3rd (The MIT Press, 2009).
  156. Le Phillip, P., Bahl, A. & Unga, L. H. Using prior knowledge to improve genetic network reconstruction from microarray data. *Silico Biology* **4**, 335–353 (2004).
  157. Scutari, M. Learning Bayesian networks with the bnlearn R package. *Journal of Statistical Software*. <[arXiv:0908.3817v2](https://arxiv.org/abs/0908.3817v2)> (2010).
  158. Sachs, K., Perez, O., Pe’er, D., A., L. D. & Nolan, G. P. Causal protein-signaling networks derived from multiparameter single-cell data. *Science* **308**, 523–529 (2005).
  159. Singh, A. K. *et al.* Integrative analysis of large scale expression profiles reveals core transcriptional response and coordination between multiple cellular processes in a cyanobacterium. *BMC Systems Biology* **4** (2010).
  160. Chickering, D. M. Optimal structure identification with greedy search. *Journal of Machine Learning Research* **3**, 507–554 (2002).

- 
161. Elvitigala, T., Singh, A. K., B., P. H. & Ghosh, B. *Bayesian network approach to understand regulation of biological processes in cyanobacteria* in *IEEE Conference on Decision and Control* (Shanghai, China, 2009).
  162. Wilczynski, B. & Dojer, N. BNFinder: exact and efficient method for learning Bayesian networks. *Bioinformatics* **25**, 286–287 (2009).
  163. *Bayes Server Ltd Company: Bayes Server package* 2011. <[www.bayesserver.com](http://www.bayesserver.com)>.
  164. *BayesFusion, LLC: GeNie & SMILE library* <[www.bayesfusion.com](http://www.bayesfusion.com)>.
  165. Warnes, G. R. HYDRA: a Java library for Markov chain Monte Carlo. *Journal of Statistical Software* (2002).
  166. Bonawitz, K., Mansinghka, V. & Cronin, B. Blaise: a toolkit for high-performance probabilistic inference. *CSAIL Digital Library* (2007).
  167. Murphy, K. Software packages for graphical models - Bayesian networks. *Bull. Int. Soc. Bayesian Anal* **14** (2007).