Computational Models of Belief Propagation

Khin Hua Ng

Submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of Imperial College London and
the Diploma of Imperial College London

October 2015

Imperial College London
Department of Computing

Abstract

In this thesis we aim to gain better understanding on the working of the belief propagation algorithm designed for graphical models in other computational frameworks like the neural systems, as well as the error associated with loopy belief propagation in Bayesian networks.

In the first part, we examine a few recent neural computational models of belief propagation and highlight the significance of these models that demonstrate the viability of performing belief propagation using neural computations by transforming it into a dynamical system. We also propose the idea of implementing the belief propagation in computational models like the Hopfield network through free energy minimisation.

It is widely known that exact inference in loopy graphs is computationally difficult and thus there has been a lot of effort spent in the area of developing practical approximate inference algorithms. Loopy belief propagation is a widely used approximate inference algorithm for graphical models. In the second part of this thesis, we analyse the loopy error and propose two exact inference algorithms using belief propagation with loop correction for Bayesian networks with generic loops. We also propose a new approximate inference method called the 2-Pass loopy belief propagation and demonstrate empirically its potential for use as a fast approximate inference algorithm with comparable accuracy to standard loopy belief propagation. We also discuss issues related to its application as an approximate inference method.

Declaration of Originality

I hereby declare that this thesis is my own work. Any contribution by others has been explicitly acknowledged and a list of references is given in the bibliography.

Khin Hua Ng

Copyright Declaration

The copyright of this thesis rests with the author and is made available under a Creative Commons Attribution Non-Commercial No Derivatives licence. Researchers are free to copy, distribute or transmit the thesis on the condition that they attribute it, that they do not use it for commercial purposes and that they do not alter, transform or build upon it. For any reuse or redistribution, researchers must make clear to others the licence terms of this work.

Acknowledgements

First and foremost, I would like to thank my supervisor, Professor Duncan Gillies for his guidance and encouragement throughout this PhD study. I am extremely grateful to him for his invaluable advice, constant encouragement and support, especially during the difficult times when I was coping with the passing of my father. I also wish to thank Professor Murray Shanahan for his advice on the part related to graphical models and neural computations.

I am very grateful to the Barnet Overseas Students Housing Association for providing such an excellent accommodation for me and my family at Nansen Village. It is truly a home away from home and a wonderful place filled with warmth, happiness, and international friendships. Special thanks to the trustees, especially Kurt, Charlotte, and Johanna who have touched our lives, and many others.

I am greatly indebted to DSO National Laboratories (Singapore) for sponsoring this PhD study. I would like to thank my colleagues from the HR department responsible for administrating this overseas scholarship and my bosses Khee Yin and Gee Wah for their kind understandings and support. I would also like to thank Shihan for his help on libdai implementation issues.

Thanks to my family for their love and support, and for giving me a peace of mind to concentrate on my studies. I am especially thankful to Alvin, Fion and Michelle for handling my matters back in Singapore.

Lastly, I want to thank my lovely daughters Yu Xin and Yu Le for always cheering me on. I would also like to give my special thanks to my wife, Pearlyn, for her unwavering encouragement, love and support. Your constant moral support kept me going during the tough times. Thank you for taking such good care of us in London.

In memory of my parents

Contents

Abstract **3**

Acknowledgements **5**

List of Figures **9**

1 Introduction **11**

2 Graphical Models **14**

2.1 Bayesian networks 14

2.2 Markov random fields 23

2.3 Factor graphs 25

Part I **29**

3 Graphical Models and Bayesian Inference in Neural and Cognitive Architectures **30**

4 Belief Propagation and Neural Computations **34**

4.1 Belief propagation as a dynamical system 34

4.2 Belief propagation as free energy minimisation 40

4.3 Concluding remarks 43

Part II **48**

5 An Alternative Way to Improve Loopy Belief Propagation in Bayesian Networks **49**

5.1 Loopy belief propagation 51

5.2 Join tree algorithm 52

5.3 Using hidden nodes to compact network structure or to compensate for conditional dependency errors 66

5.4 Our proposal 82

6 Error Analysis of Loopy Belief Propagation in Bayesian Networks 84
6.1 Join tree algorithm versus loopy belief propagation on a simple loop 84
6.2 Error analysis on simple single loop with no instantiation 94
6.3 Exact inference in a generic single loop with no instantiation using standard belief propagation with loop correction 99
6.4 Loopy error due to evidence entering the loop through the sink node 106
6.5 Discussion 116
7 2-Pass Loopy Belief Propagation for Fast Approximate Inference 123
7.1 Empirical evaluations of the 2-Pass loopy belief propagation 131
7.2 Examples where 2-Pass loopy belief propagation performs better 143
7.3 Empirical study on 2-Pass loopy belief propagation in a single loop with evidence at the sink node 147
7.4 Discussions on using 2-Pass loopy belief propagation with loop correction for approximate inference 156
8 Conclusion 164
Bibliography 169

List of Figures

2.1	Graphical models ……...	15
2.2	Examples of Bayesian networks …….....	16
2.3	An illustrative example of Pearl's message passing algorithm [1] ……	22
2.4	Pair-wise MRF ……...	22
3.1	Examples of Bayesian models of cognitive and neural architectures [2] [3] …….	32
4.1	A Hidden Markov Model ……..……..	35
4.2	Bayesian inference in binary HMM using Spiking Neurons (Deneve [4]) ……...……...	37
5.1	(a) A multiply connected network (b) A clustered network ……..	50
5.2	(a) Original Bayesian network (b) "Marry" parents (c) Moral graph …...	53
5.3	Triangulating a graph using maximum cardinality search with fill-ins ….	56
5.4	Clique ordering and join tree …….	57
5.5	Logical control example ……..……	62
5.6	Two examples of altering a network structure by introducing a hidden node …….……...	67
5.7	The various propagation methods …….…….	70
5.8	Message passing for backward propagation ……..	71
5.9	Message passing for forward propagation …….……	75
5.10	Summary of hidden node training algorithm ……..……	76
6.1	(a) A simple loop. (b) Join tree of simple loop……..…….	85
6.2	A single loop with generic CPT entries ……..……..…….	95
6.3	A generic size single loop ..…….……...	100
6.4	Example of a single loop Bayesian network (a) and its join tree (b) …….	104

6.5 Example of a Bayesian network with multiple loops……..….... 104

6.6 A Bayesian network with double loops sharing the same source and sink nodes …….. 105

6.7 Virtual hybrid graph showing the effects of λ evidence received at node W …… 108

6.8 π evidence entering loop through intermediate node ……... 117

6.9 λ evidence entering loop through intermediate node ……... 118

7.1 The Asia network ……..…….. 132

7.2 A join tree for the Asia network …….…… 132

7.3 A Bayesian network (a) and its join tree (b) …….…….. 144

7.4 Examples of sub-networks connected to the sink node …….. 144

7.5 Results of Test 1 ……..…… 153

7.6 Results of Test 2 ……..…… 153

7.7 Results of Test 3 ……..…… 153

7.8 Results of Test 4 ……..…… 154

7.9 Error graphs for a fixed set of attractive CPT matrices with varying 154

7.10 Error graphs for a fixed set of arbitrary CPT matrices with varying $P(c_1)$ ……... 154

7.11 Results of Test 5 ……..…… 155

7.12 Results of Test 6 ……..…… 155

7.13 An example network (a), its join tree (b), and its modified network (c) using a hidden node. …….. 157

7.14 Constraints on range of p_{sr} given p_s and p_r …….. 161

Chapter 1

Introduction

In this thesis we aim to gain better understanding on the working of the belief propagation algorithm designed for graphical models in other computational models like the neural systems, as well as the error associated with the loopy belief propagation in Bayesian networks. In recent years, there has been increasingly more interdisciplinary research work on studying the brain, the mind and human intelligence through fields like philosophy, psychology, artificial intelligence, neuroscience, linguistics, and anthropology, with the goal of understanding how the brain works using its bio-mechanism to build a machine with human-like intelligence. In the first part of this thesis, we examine the possibility of implementing belief propagation used in generic graphical models in other computational models like Hopfield networks and Spiking neural networks. This was motivated by the concept of Bayesian model of cognition, and several other neural and cognitive architectures that make use of graphical models as their internal knowledge representations and Bayesian inference as the reasoning mechanism. We envisage the use of Bayesian models, or graphical models in general, as the bridge between neural and symbolic systems by acting as the middle layer in a cognitive architecture where there are lower level neuronal computations and higher level rule based knowledge representations. Specifically, we investigate the link between belief propagation and free energy minimisation, and belief propagation as a dynamical system.

It is widely known that exact inference in loopy graphs is computationally difficult and thus there has been a lot of effort spent in the area of developing practical approximate inference algorithms. Loopy belief propagation is a widely used approximate inference algorithm for graphical models. It works by applying the same belief propagation algorithm designed to compute exact marginal probabilities on singly connected Bayesian networks to those that contained loops. In the second part of this thesis, which presents the main contributions of our research, we analyse the loopy error and explore ways to improve the loopy belief propagation. We propose a new approximate inference method called the 2-Pass loopy belief propagation. Our idea of applying a 2-Pass control to the loopy belief propagation was born from understanding the effects of the 2-Pass evidence propagation of the join tree algorithm. We investigate the performance of the proposed 2-Pass method empirically and discuss issues related to its application. The ability to carry out loop correction is due to the findings from the error analysis we have conducted on applying loopy belief propagation to several different configurations of single loop Bayesian networks.

The remainder of this thesis is organised as follows. In chapter 2, we introduce the three main types of graphical models, namely, Bayesian networks, Markov random fields, and factor graphs and the two main types of inference problems. We present the different factorisations of the joint distributions modelled by the network structures of each type of graphical models, and their respective inference algorithms. We illustrate the d-separation property with a few examples of Bayesian networks. We give a brief summary of Pearl's message passing algorithm in Bayesian network, which also introduces the notion of λ evidence and π evidence. Then we present the message update operations of belief propagation (sum-product) for factor graphs and Markov random fields, and provide some remarks on the equivalence of Pearl's message passing algorithm and the sum-product algorithm. In chapter 3, we provide the motivation for the first part of our research. Chapter 4 presents our findings on the linkages between belief propagation and neural computations. In chapter 5, we present an alternative thinking on how we can improve the performance of loopy belief propagation when applied to Bayesian networks and to develop approximate belief propagation models for approximate inference. Our

proposal is the result of surveying some established algorithms for carrying out inference in Bayesian networks combined with inspirations from more recent research results related to understanding loopy belief propagation. In chapter 6, we present our findings on errors due to loopy belief propagation applied to several different configurations of single loop Bayesian networks. In chapter 7, we demonstrate the potential of using our proposed 2-Pass loopy belief propagation through empirical studies and also discuss issues related to its application as an approximate inference method. We conclude in chapter 8.

Chapter 2

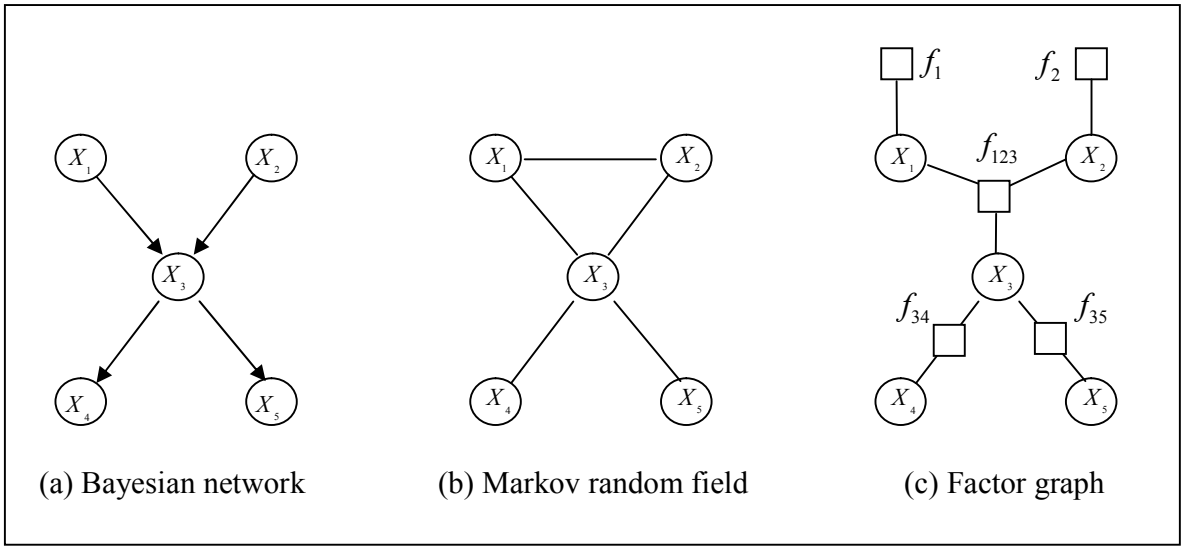
Graphical Models

Graphical models are generally categorized into directed and undirected models corresponding to a graph's directed and undirected edges respectively. Directed acyclic graphical models are known as Bayesian networks (BN) and undirected graphical models are known as Markov random fields (MRF).

2.1 Bayesian networks

Bayesian networks [5] have been increasingly gaining popularity since the early '90s for reasoning with uncertainty in knowledge-based systems. They are powerful tools that allow representation of domain knowledge and uncertain beliefs in a probabilistic way that also provide a reasoning mechanism for inference. A Bayesian network uses a combination of probability theory and graph theory. It is a graphical (probabilistic) model and is fully defined by two types of knowledge: qualitative and quantitative. Figure 2.1(a) shows an example of Bayesian network. The qualitative part refers to a directed acyclic graph $G = (V, E)$, where $V = \{X_1, ..., X_n\}$ is the set of nodes (vertices) where each node is labelled by a decision variable or state of observation, and E is the set of directed network arcs (edges or links) which indicate the strength of causal relationship / conditional dependency between variables. A network arc emanates from a parent node and points to a child node. A child node can have multiple parent nodes. We shall use the terms variables and nodes interchangeably throughout this thesis. A random variable is either discrete or continuous, and a Bayesian network can contain a mixture of both. Although

Bayesian networks are fully capable of handling continuous variables, for certain application, the continuous variables might be pre-processed into discrete ones (a process called discretisation). The value or state of a variable X_i is denoted by x_i .The quantitative part is a set of conditional probability functions (equivalently, conditional probability tables (CPTs) for program execution) of the form $P(x_i \mid Parents(x_i))$. In the case where a node has no parent, its unconditional probability distribution is provided instead. Conditional and unconditional probabilities are usually determined through learning from past data or consultation from domain experts.



(a) Bayesian network (b) Markov random field (c) Factor graph

Figure 2.1: Graphical models.

The beauty of the Bayesian networks framework lies with its factorisation equation that exploits the conditional dependence (or independence) between variables to yield a more compact representation and computation of the joint probability distributions. The factorisation equation defines the joint probability distribution across all nodes as a product of local conditional probability distributions:

$$P(x_1,...,x_n) = \prod_{i=1}^{n} P(x_i \mid Parents(x_i))$$

based on the assumption that the variable X_i is conditionally independent of its non-descendants when given the states of its $Parents(X_i)$. Two variables X_i and X_j are said

to be independent if $P(x_i, x_j) = P(x_i)P(x_j)$ or equivalently $P(x_i \mid x_j) = P(x_i)$. Similarly, X_i and X_j are conditionally independent given X_k if

$$P(x_i \mid x_j, x_k) = P(x_i \mid x_k)$$

and

$$\begin{aligned} P(x_i, x_j \mid x_k) &= P(x_i \mid x_j, x_k)P(x_j \mid x_k) \\ &= P(x_i \mid x_k)P(x_j \mid x_k) \end{aligned}.$$

By "given x_k", we mean an observation on the state of x_k has been made, e.g., $x_k = 1$, which is also called instantiation.

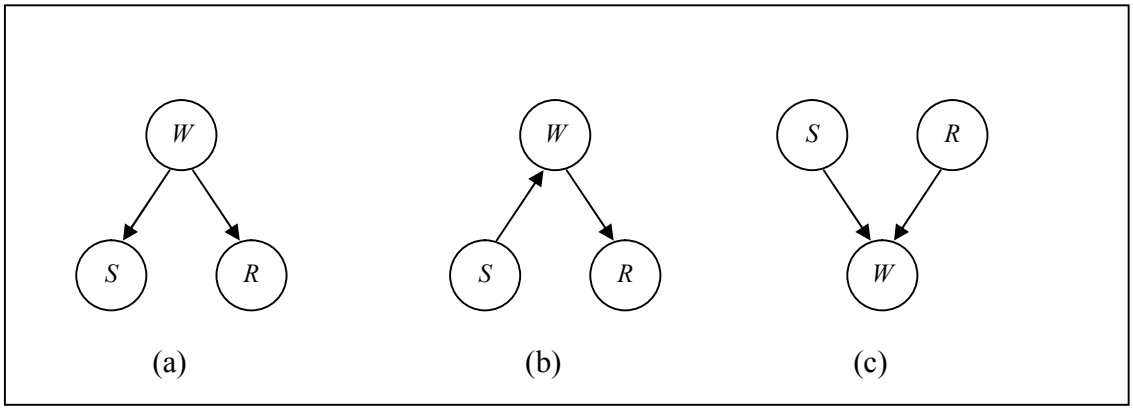


Figure 2.2: Examples of Bayesian networks.

Let us look at three simple examples of different probability distributions represented by different network structures. Suppose we have three variables *S*, *R* and *W*, and *S* is conditionally independent of *R* given *W*. Then we have $P(s, r \mid w) = P(s \mid w)P(r \mid w)$, and we can represent this conditional dependency using the Bayesian network shown in Figure 2.2(a). In this case, we can write the joint probability distribution as

$$\begin{aligned} P(s, r, w) &= P(s, r \mid w)P(w) \\ &= P(s \mid w)P(r \mid w)P(w) \end{aligned}.$$

Figure 2.2(b) shows another Bayesian network that also depicts *S* being conditionally independent of *R* given *W*. For this network, the joint probability distribution is

$$P(s, r, w) = P(r \mid w)P(w \mid s)P(s).$$

Figure 2.2(c) shows a different conditional dependency scenario to the previous two cases. In this case, the *S* and *R* are marginally independent but become conditionally dependent on given *W*, and the joint probability distribution is expressed as

$$P(s,r,w) = P(w \mid s,r)P(s)P(r) .$$

The formal statement on the dependencies among variables in a Bayesian network is known as the *d-separation* property [5]. A thorough introduction on the d-separation property is also covered in the book on graphical models by Jordan and Weiss [6]. Another related concept is that of Markov blanket – a node is conditionally independent of all other nodes in a Bayesian network given its Markov blanket, which is a set of nodes comprising of its parent nodes, children nodes, and other parent nodes sharing the same child node.

The purpose of building a Bayesian network is to allow a process called *inference* to be carried out which will provide answers to specific problems the network has been built on. There are two main types of inference processes, namely,

- marginal inference: compute

$$P(x_1), P(x_2), ..., P(x_n) , \text{ and}$$

- maximum a posterior probability (MAP): compute

$$\arg \max_{x_1, x_2, ..., x_n} P(x_1, x_2, ..., x_n) .$$

The task of marginal inference, also known as belief updating [5], is to update or compute the values of $P(x_i)$ given new evidence or observation, which is written as $P(x_i \mid evidence)$. Fundamentally, the inference process makes use of the Bayes Rule (hence the term "Bayesian"):

$$P(x_i \mid evidence) = \frac{P(evidence \mid x_i) * P(x_i)}{P(evidence)} .$$

Computing the marginal distribution by direct summation is computationally intensive as it involves summing over an exponential number of terms. For example, if we want to compute the marginal probability of the last node x_5 of the example shown in Figure 2.1(a), we have to compute the following

$$P(x_5) = \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4} P(x_1, x_2, x_3, x_4, x_5)\,.$$

If we make use of the factorised joint distribution, we obtain a more efficient way to compute

$$\begin{aligned} P(x_5) &= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4} P(x_5 \mid x_3)P(x_4 \mid x_3)P(x_3 \mid x_1, x_2)P(x_1)P(x_2) \\ &= \sum_{x_3} P(x_5 \mid x_3)\sum_{x_4} P(x_4 \mid x_3)\sum_{x_1, x_2} P(x_3 \mid x_1, x_2)P(x_1)P(x_2) \end{aligned}$$

where it has now become a product of sums, of which each sum involves a smaller subsets of variables. Now if we have received some evidence, say x_4 is in its *i*-th state (written as $x_4 = i$), then the inference problem becomes

$$\begin{aligned} P(x_5 \mid x_4 = i) &= \frac{P(x_4 = i, x_5)}{P(x_4 = i)} \\ &= \frac{P(x_4 = i, x_5)}{\sum_{x_5} P(x_4 = i, x_5)} \end{aligned}$$

In this case, we will first compute the numerator term

$$\begin{aligned} P(x_4 = i, x_5) &= \sum_{x_1}\sum_{x_2}\sum_{x_3} P(x_5 \mid x_3)[P(x_4 \mid x_3)\delta(x_4, i)]P(x_3 \mid x_1, x_2)P(x_1)P(x_2) \\ &= \sum_{x_3} P(x_5 \mid x_3)\sum_{x_4}[P(x_4 \mid x_3)\delta(x_4, i)]\sum_{x_1, x_2} P(x_3 \mid x_1, x_2)P(x_1)P(x_2) \end{aligned}$$

where

$$\delta(x_4, i) = \begin{cases} 1 & x_4 = i \\ 0 & \text{otherwise} \end{cases}$$

for all the states of X_5 then divide them by the normalisation constant

$$\sum_{x_5} P(x_4 = i, x_5)$$

to obtain the posterior probabilities. This example illustrates that generally the inference process is the same with or without evidence, except that with evidence input there are the additional steps of instantiating the variable to its observed state at the beginning and the normalisation at the end. In general we may also want to compute the posterior joint distribution of several variables given some observations on other variables. For example, we may want to compute $P(x_1, x_5 \mid x_2, x_3, x_4)$, which can be written as

$$P(x_1, x_5 \mid x_2, x_3, x_4) = \frac{P(x_1, x_2, x_3, x_4, x_5)}{P(x_2, x_3, x_4)}\,.$$

Therefore in general, the marginal inference problem is equivalent to solving $P(x_c)$ for some X_c subsets of V.

The strength of Bayesian networks lies in their strong modelling power and existence of many network construction tools. There are also algorithms that facilitate domain knowledge representation, for learning network models from data, as well as many different forms of inference algorithms to choose from to suit specific problems. Bayesian networks have been successfully applied in many different domains. For example, in solving diagnostic and troubleshooting problems, machine learning, data mining, pattern recognition, and more recently in bioinformatics and computational neuroscience. Bayesian networks are also increasingly being applied to medical diagnosis to model the relationship among diseases, intermediary variables and observable measurements from clinical tests. When modelling using Bayesian networks, the direction of an arc is at times interpreted as a "causal" link that indicates the influence of the parent node to the child node, the strength of "causation" being captured in the arc's CPT [7].

Much of the interest generated in the early 90's was in part due to the pioneering work by Pearl [5], in which he proposed an efficient algorithm for evidence propagation known as message passing or belief updating algorithm for the class of polytree (singly connected) networks. We give a brief summary of Pearl's message passing algorithm here. For any node X_i in a singly connected network, we use the d-separation property to partition the evidence into the set $\{e^-, e^+\}$, where e^- and e^+ are evidence reaching node X_i through its child nodes and parent nodes respectively. Then we can rewrite the term $P(x_i \mid evidence)$ as

$$
\begin{aligned}
P(x_i \mid evidence) &= P(x_i \mid e^-, e^+) \\
&= \frac{P(x_i, e^-, e^+)}{P(e^-, e^+)} \\
&= \frac{P(e^- \mid x_i, e^+) * P(x_i \mid e^+) * P(e^+)}{P(e^-, e^+)} \\
&= \alpha P(e^- \mid x_i) * P(x_i \mid e^+) \\
&= \alpha \lambda(x_i) \pi(x_i)
\end{aligned}
$$

where α is a normalizing constant, $\lambda(x_i) = P(e^- \mid x_i)$ is the likelihood, and $\pi(x_i) = P(x_i \mid e^+)$ is the conditional probability given evidence e^+. When performing the message passing algorithm, each node X_i then computes a term called the belief function $b_i(x_i) = \alpha \lambda(x_i) \pi(x_i)$ which converges to $P(x_i \mid evidence)$ as messages are passed between each pair of parent and child nodes. The rules for generating the messages are as follows. Node X_i receives a message $\lambda_{y_j}(x_i)$ from one of its child nodes Y_j and message $\pi_{x_i}(u_k)$ from one of its parent U_k to compute

$$\lambda(x_i) = \prod_j \lambda_{y_j}(x_i) \text{ and } \pi(x_i) = \sum_{u_k} P(x_i \mid \{u_k\}) \prod_k \pi_{x_i}(u_k).$$

Upon updating the belief $b_i(x_i)$, node x_i sends its parent u_l the updated message

$$\lambda_{x_i}(u_l) = \sum_{x_i} \lambda(x_i) \sum_{u_k, k \neq l} P(x_i \mid \{u_k\}) \prod_{k \neq l} \pi_{x_i}(u_k)$$

and to its child y_j

$$\pi_{y_j}(x_i) = \frac{b_i(x_i)}{\lambda_{y_j}(x_i)}.$$

Since the message passing algorithm is an iterative process, we have to perform the following initialisation step:

1. For each root node with no parent nodes, set $\pi(x_i) = P(x_i)$, the prior probabilities of X_i.
2. For each leaf node with no child nodes, set $\lambda(x_i) = 1$ for all x_i. In practice, we can set it to any fixed constant.
3. For each $X_i = e_j \in \{e^-, e^+\}$

$$\pi(x_i) = \lambda(x_i) = \begin{cases} 1 & x_i = e_j \\ 0 & \text{otherwise} \end{cases}.$$

Figure 2.3 gives an illustrative example, extracted from [1], of Pearl's message passing algorithm for updating a Bayesian Network.

While Pearl's algorithm is very efficient for polytree Bayesian networks, many real world applications are much more complex and entail a multiply connected network, for which the algorithm is not directly applicable. The inference algorithms for generic Bayesian networks are classified as exact or approximate depending on the type of posterior marginal distribution computed. It has been proven that exact inference computation in Bayesian networks is NP-hard in general [8], and subsequently it was discovered that approximate inference is NP-hard too [9]. Nevertheless, many researchers at that time envisaged the potential of Bayesian networks as useful probabilistic reasoning tools and the quest for more efficient algorithms went on.

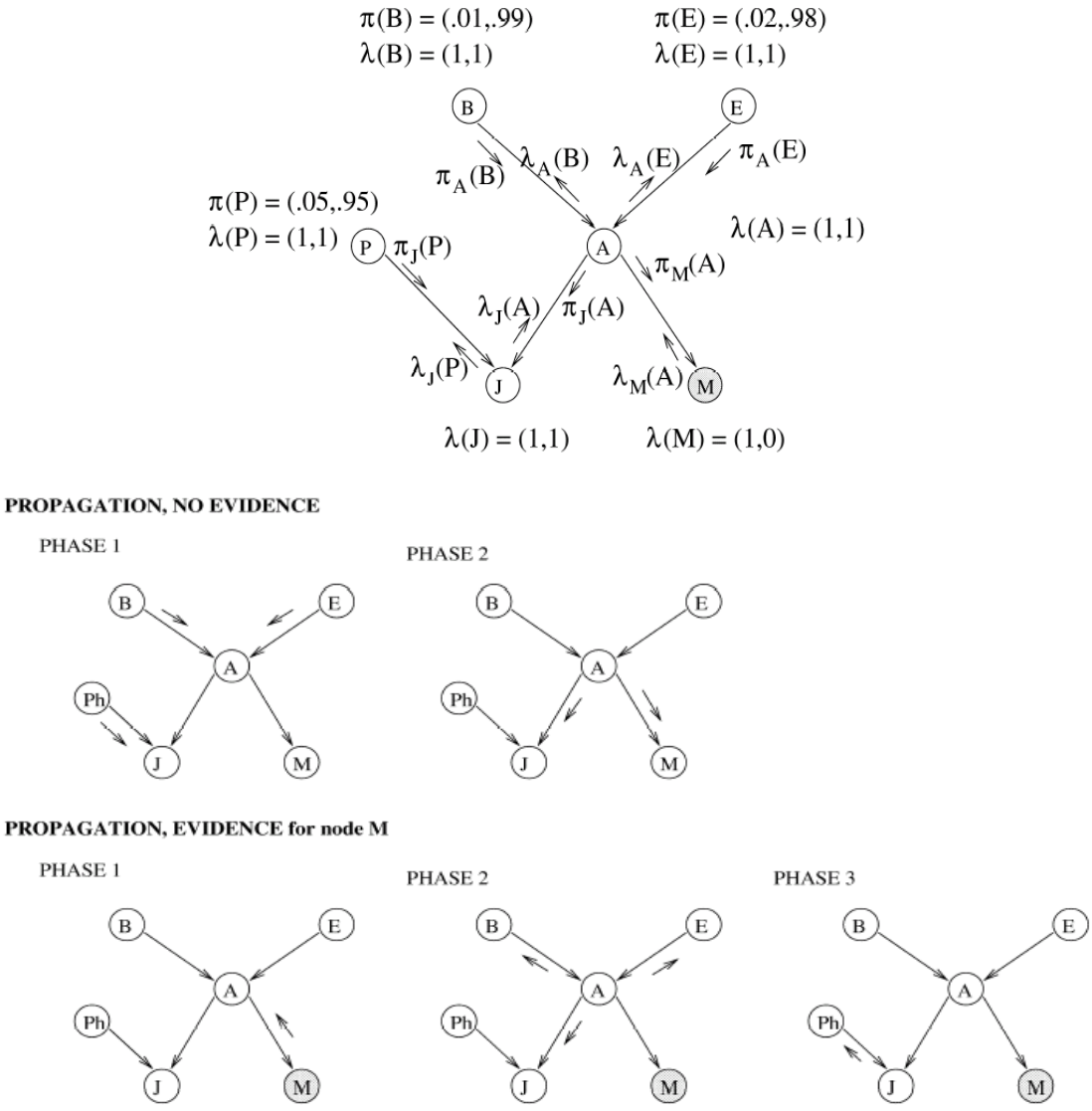


Figure 2.3: An illustrative example of Pearl's message passing algorithm [1].

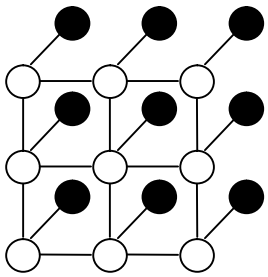


Figure 2.4: Pair-wise MRF.

2.2 Markov random fields

Markov random fields (MRF), also called Markov networks, are undirected graphical models widely used in statistical image processing applications [10]. Figure 2.1(b) shows an example of a MRF. We have seen that the graphical structure of a Bayesian network represents the factorisation of the joint probability distribution of all the random variables defined in the model. This is also the case for undirected models. In the case of Bayesian networks, the factorisation is defined by

$$P(x_1,...,x_n) = \prod_{i=1}^{n} P(x_i \mid Parents(x_i))$$

where the joint distribution is factored into a product of conditional distributions of smaller sets of variables. The Hammersley-Clifford theorem [11] states that the joint probability distribution of any MRF can be written as a Gibbs distribution of the form

$$P(x_1,...,x_n) = \frac{1}{Z}\prod_{C}\psi_C(x_C)$$

where C are maximal cliques of the graph and each ψ_C is a non-negative function called the potential function, and

$$Z = \sum_{x}\prod_{C}\psi_C(x_C)$$

is the normalization constant also known as the partition function. A clique is a fully connected sub-graph and it is maximal if it is not contained in another clique. Unlike the directed case, here Z is required to make the expression a valid probability distribution.

The differences between directed and undirected graphical models lie in their representations of different Markov properties, i.e. the relationship between graph separation and conditional independence. As discussed by Jordan and Weiss [6], these differences are important when examining the family of joint probability distributions that a particular graph can represent. However, in the inference problem, since we would have a specific fixed joint probability distribution to work on, the differences are less important.

A commonly used MRF model is the pair-wise MRF often found in computer vision applications. The model is "pair-wise" such that the maximal cliques are all of size two (and hence the potential function is defined on pairs of variables). If we denote the set of hidden and observed variables in a pair-wise MRF as $\{X\}$ and $\{Y\}$ respectively, then the factorisation of the joint distribution is given by

$$P(\{x\},\{y\}) = \frac{1}{Z}\prod_{(i,j)}\psi_{ij}(x_i,x_j)\prod_i \phi_i(x_i,y_i)\,.$$

Here the function $\psi_{ij}(x_i,x_j)$ represents the *compatibility* (dependencies) between variable X_i and variable X_j, and $\phi_i(x_i,y_i)$ provides the local evidence for variable X_i from the observed variable Y_i. For example, in image segmentation we typically work with a 2D array of pixel intensities represented by some numbers. The pair-wise MRF will then have a grid-like structure defining two types of variables, namely, X_i and Y_i where Y_i is the observed intensity of pixel *i* and X_i is the hidden variable indicating if pixel *i* belongs to some regions or objects of interest. Figure 2.4 shows an example of such pair-wise MRF where the filled-in circles represent Y_i and the empty circles represent X_i.

It turns out that the pair-wise MRF is closely related to the Potts and Ising models in physics, and this relationship facilitates the definition of energy functions in the graphical models. This further leads to the concept of *free energy* in belief propagation, which is an important idea in our research. Hence, we give a short account of this relationship extracted from Yedidia et al. [12] here. If we define the "interaction" between neighbouring nodes by $J_{ij}(x_i,x_j) = \ln\psi_{ij}(x_i,x_j)$ and the "field" at each node as $h_i(x_i) = \ln\phi_i(x_i,y_i)$, then the Potts model *energy* is given by

$$E(x_1,...,x_n) = -\sum_{i,j} J_{ij}(x_i,x_j) - \sum_i h_i(x_i)$$

Using Boltzmann's law from statistical physics

$$P(\{x\}) = \frac{1}{Z}e^{-E(\{x\})/T}$$

we see that the above pair-wise MRF is exactly the Potts model if we set the temperature to 1. Furthermore, if each node has exactly two states (binary pair-wise MRF), which

usually takes on the values of 1 and -1, then it is called the Ising model. If we further restrict to symmetric J_{ij} interactions, the energy function can be written as

$$E(x_1,...,x_n) = -\sum_{i,j} J_{ij} x_i x_j - \sum_i h_i x_i$$

Lastly, in the context of the Ising model, the inference problem of computing the beliefs $b_i(x_i)$ can be mapped onto the physics problem of computing local "magnetizations"

$$m_i \equiv b_i(x_i = 1) - b_i(x_i = -1).$$

2.3 Factor graphs

A third type of graphical model is known as *factor graph.* Kschischang et al. [13] introduced the factor graph and the sum-product algorithms and showed that many known algorithms developed in artificial intelligence, signal processing, and digital communications can be derived as specific instances of the sum-product algorithm. The factor graph is a popular model for decoding error-correcting codes in the domain of data transmission. While it is an undirected graph, it has an additional type of node that explicitly represents the potential functions. A factor graph is a bipartite graph that connects variables and factors, where each factor represents a function of the probability distribution over the variables it connects to. Figure 2.1(c) shows an example of a factor graph where the circles represents the variables as in BN and MRF, and the squares are the function nodes found only in factor graphs. The factor graph is useful when we need to describe models that involve a large number of overlapping relationships between variables. The factor graph is often seen as the unifying framework of both Bayesian networks and Markov random fields, and like these graphical models, it also provides a factorisation of the joint distribution given by

$$P(x_1,...,x_n) = \frac{1}{Z}\prod_a f_a(x_a)$$

where a is an index labelling the function nodes and x_a denotes the subset of variables connected to the function node. If we compare the factorisation equations between Bayesian networks and factor graphs, it is clear that we can easily map the functions f_a to the terms $P(x_i \mid Parents(x_i))$ and convert from one graphical model to the other. For an example, in Figure 2.1(c) we have

$$f_1 = P(x_1), \quad f_2 = P(x_2), \quad f_{123} = P(x_3 \mid x_1, x_2)$$
$$f_{35} = P(x_5 \mid x_3) \quad \text{and} \quad f_{45} = P(x_5 \mid x_4)$$

In general, we can always transform graphical models from one form to the other and more important is the fact that belief propagation algorithms, which have been developed specifically to each model of Bayesian networks, Markov random fields, and factor graphs, haven been shown to be mathematically equivalent [14]. Hence it gives us the flexibility in choosing the type of representation that suits our problem and breakthroughs in the inference algorithm for one type of model can be generalized to the others if required. For example, we often see the pair-wise MRF being used as the representation for research work reported in the literature related to belief propagation properties as it has the simplest form of one type of message to compute and propagate. On the other hand, in Bayesian networks, due to the direction of the arrow (causal direction), there is a π-message from parent node to child node and a separate λ-message from child node to parent node. Similarly, in the case of the factor graph, there are two types of messages: from factor node to variable node, and variable node to factor node.

For factor graphs, the belief propagation (sum-product) algorithm is a method to compute an approximation to the marginal probabilities of variables in the graphical model. As mentioned previously, computing the marginal distribution by direct summation is computationally intensive. Belief propagation is a more efficient iterative process that relies on local message passing (or distributed computations) between nodes making use of the factorisation of the joint distribution defined by the graph structure, just like Pearl's message passing algorithm. In fact, the sum-product algorithm is formally equivalent to Pearl's message passing algorithm. However, in this case since a factor graph is not a directed graph there is no differentiating of the π-message or λ-message. The key idea of the sum-product algorithm is to take advantage of the fact that multiplication is distributive over addition, which is

$$xy + xz = x(y + z),$$

to reduce the arithmetic operations. The idea is to compute what is called the belief $b_i(x_i)$ of each node (and factors in the case of factor graph), by combining the input messages received from a node's neighbours, and as the process converges, $b_i(x_i)$ converges to an approximation of the marginal distribution $P(x_i)$.

The message update rule in a factor graph is given by

(a) from variable node *i* to factor node *a*: $n_{i\to a}(x_i) \leftarrow \prod_{b\in N(i)\setminus a} m_{b\to i}(x_i)$, and

(b) from factor node *a* to variable node *i*: $m_{a\to i}(x_i) \leftarrow \sum_{x_a \setminus x_i} f_a(x_a) \prod_{j\in N(a)\setminus i} n_{j\to a}(x_j)$

where $N(a)$ denotes the neighbours of a factor node *a*.

The belief is computed as follows:

$$b_i(x_i) \propto \prod_{a\in N(i)} m_{a\to i}(x_i)$$

(Note: we have to normalize the beliefs for it to sum to one.)

In the case of pair-wise MRF, the above message update rule can be further simplified to a single message $m_{ij}(x_j)$ from variable *i* to variable *j*

$$m_{ij}(x_j) \leftarrow \sum_{x_i} \psi_{ij}(x_i, x_j)\phi_i(x_i, y_i) \prod_{k\in N(i)\setminus j} m_{ki}(x_i)$$

and the belief is computed by

$$b_i(x_i) \propto \phi_i(x_i, y_i) \prod_{k\in N(i)} m_{ki}(x_i).$$

So far we have concentrated our discussions mainly on marginal inference and seem to have neglected the MAP problem. The reason for this is that the same belief propagation algorithm for marginal inference can often be modified to solve the MAP problem. For example, the MAP's variant of sum-product algorithm is called the max-product. For pair-wise MRF, the max-product message update rule is

$$m_{ij}(x_j) \leftarrow \max_{x_i} \psi_{ij}(x_i, x_j)\phi_i(x_i, y_i) \prod_{k\in N(i)\setminus j} m_{ki}(x_i)$$

and the belief equation is unchanged, but in this case the beliefs are not marginal probabilities but the most probable states. Sometimes, the MAP problem is formulated in the log domain, and the max-product becomes max-sum. While both inference problems

often share common algorithms with minor modifications, it is important to note that there exist specialised algorithms that are tuned to solve each class of inference problem, and results from one may not be generalised to the other.

When implementing belief propagation, we need an update schedule to determine the order for each node to update their messages through time. There are many versions of update schedules. For example, using the *flooding schedule* all nodes update their messages using current messages received from their neighbours at each iteration, and thus it is suitable for parallel computations. On the other hand, in a *serial* schedule only one message is sent anywhere in the graph at a time. There are also more complex schedules like *residual belief propagation* [15] and *residual splash belief propagation* [16], which have a dynamic schedule that prioritise updates for messages according to the size of their residuals (the difference between updated and current messages according to some distance measure).

While the belief propagation algorithm is termed an approximation, it actually computes the true values of marginal probabilities (exact inference) if the graph has no loops, just as Pearl's message passing algorithm is exact for polytree networks. When belief propagation is applied to graphs with loops (called loopy belief propagation), convergence is not guaranteed, and even if it converges, the beliefs are not guaranteed to be close to the true values. Nevertheless, loopy belief propagation has been shown to be well defined and empirically it can often produce good approximations [17] and has also been used successfully in many applications such as turbo codes [18]. There are also various studies that reported conditions under which loopy belief propagation performs well and also on properties of the error bounds. There are also other related approximate inference algorithms like Generalised Belief Propagation [12], [14], [19], Survey Propagation [20], and Expectation Propagation [21].

Part I

Belief Propagation and Neural Computations

Chapter 3

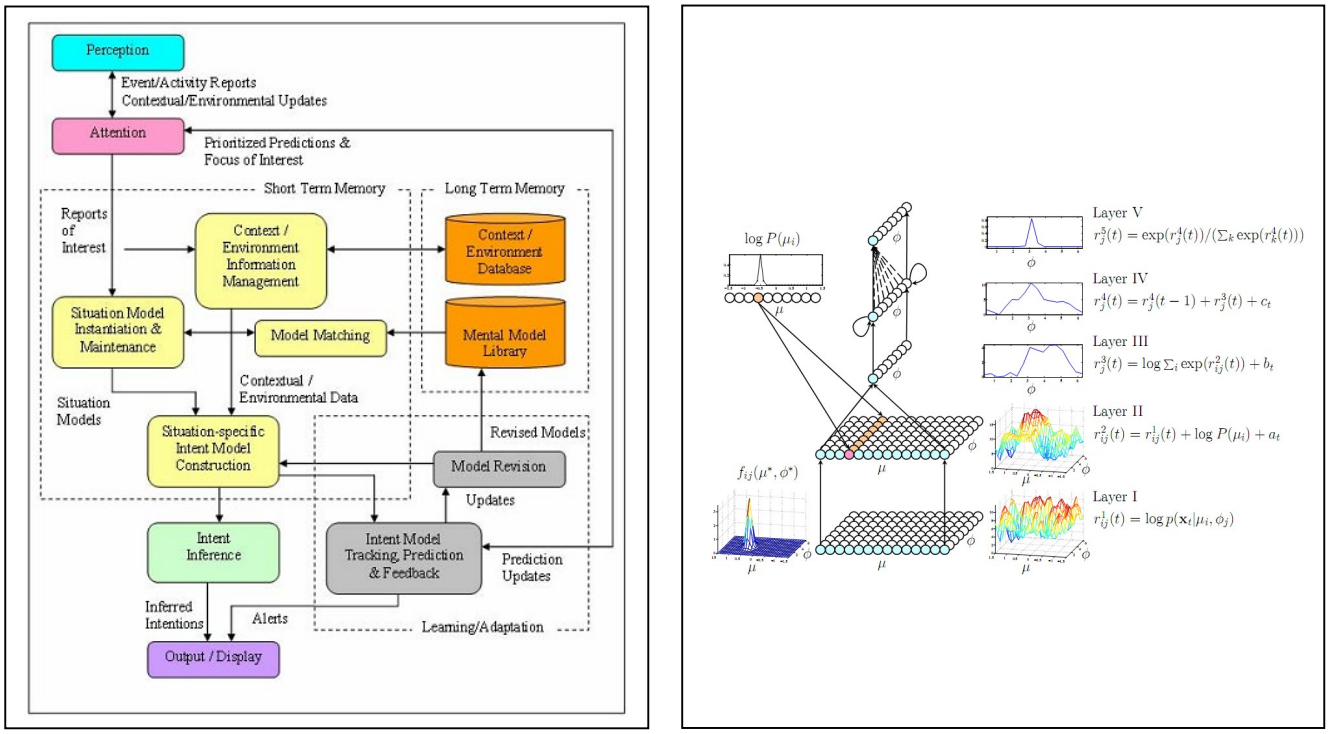
Graphical Models and Bayesian Inference in Neural and Cognitive Architectures

The broad aim of this part of our research is to study the use of computing theories from fields like Artificial Intelligence (AI), Cognitive Science and Computational Neuroscience to gain insights that will help the development of more robust intelligent systems, cognitive architectures, or algorithms for tasks like pattern recognition and decision making. In recent years, we are seeing increasingly more interdisciplinary research work on studying the brain, the mind and human intelligence through fields like philosophy, psychology, artificial intelligence, neuroscience, linguistics, and anthropology. Their research aim is simple - understand how the brain works with its bio-mechanism to build a machine with human-like intelligence, but the problem is far from being solved. For example, many cognitive scientists are still searching for the answer to the fundamental questions of what is intelligence, how it is produced in our brain, and if it can be modelled computationally. Through collaborations among the different disciplines, the interactions among different established theories of the brain, the mind and the computer have generated unifying ideas and prompted new theories in one discipline using evidence gathered in the others.

Broadly categorised, there are two main research approaches in Cognitive Science [22]. The symbolic approach is based on the belief that cognition can be explained using

operations on symbols, and the mind has mental representations corresponding to some types of computing data formats like rules, images, and concepts. This is usually the approach taken by cognitive psychologists, philosophers, and AI computer scientists. On the other hand, the sub-symbolic or connectionist approach is inspired by neuroscience where it holds that cognition can only be modelled and explained by using systems like artificial neural networks that are biologically inspired and plausible. Some researchers view the two approaches as competing while others see them as complementary. The conflicting views arise probably due to the gap that exists between the results produced by the two approaches thus far. For example, research on cognitive architecture (a computational infrastructure that supports intelligent behaviours) usually implements human knowledge through symbolic representation, like rules, suited for current generation of computing architectures instead of some form of neuronal representations that encode these rules. While there exist very elaborate cognitive architectures designed at the neuronal level, these architectures usually implement very specific functions or regions of the brain which itself does not constitute cognition. Furthermore the scaling up of neural circuits to achieve higher level of cognition remains a very challenging problem. As Barsalou [23] succinctly puts it, the problem with classic theories of cognition is not due to the inclusion of symbolic operations, but *how* these symbolic operations are being implemented. Hence, this has prompted some connectionists to develop neural network accounts of symbolic operations. However, at that time, these approaches did not gain popularity with either cognitive psychologists or with knowledge engineers possibly because to psychologists, neural network accounts of symbolic operations had little psychological plausibility and to knowledge engineers, they were difficult and inefficient to implement. Nonetheless, there has been a recent resurgence of interest in this area and it is called "Neural-Symbolic Integration". Proponents like Bader and Hitzler [20], and the many authors who contributed in the edited book titled "Perspectives of Neural-Symbolic Integration" [24], which covers recent developments, argue that Neural-Symbolic Integration is a key area for the advancement of artificial intelligence in creating machines with human-like abilities.

Our standpoint on the symbolic and connectionist approaches is that they can potentially complement each other and hence yield a more powerful and intelligent system if we direct effort to work on the underdeveloped link between the knowledge representation that is typically framed using AI techniques and the memory representation of neural circuits as well as the correspondences between human inference process and neural computation. Our research work is motivated in this direction.



(a) (b)

Figure 3.1: Examples of Bayesian models of cognitive and neural architectures [2] [3].

Recently, the concept of a "Bayesian Brain" [25] or Bayesian model [26] of cognition has generated a lot of interest in the fields of cognitive science and computational neuroscience. Researchers have shown that many cortical processes could be explained in a Bayesian way, and neurophysiological data that have been observed exhibit neural computations similar to Bayesian inference process. Ng et al. [2] have developed a cognitive architecture (Figure 3.1(a)) for activity and intent inference using Bayesian network knowledge fragments. George and Hawkins [27] propose a

hierarchical model of invariant visual pattern recognition of the human visual cortex where knowledge is encoded in Bayesian network structures and argue that their model largely matches available anatomical data. Yu and Dayan [3] present a five-layer hierarchical Bayesian neural architecture (Figure 3.1(b)) that implements optimal probabilistic integration of top-down information and sequentially observed data. Motivated by these research efforts, we are interested in the use of Bayesian models in developing intelligent systems, and we would like to investigate the possibility of implementing the inference processes using a connectionist approach. Specifically, we have identified a key area of bridging Bayesian models and neuronal implementations prompted by recent research work on Bayesian Inference using Spiking Neurons [28], [29], and we will present our study in the next chapter. We envisage the use of Bayesian models, or graphical models in general, as the bridging model between neural and symbolic systems by acting as the middle layer in a cognitive architecture where there are lower level neuronal computations and higher level rule based knowledge representations.

Chapter 4

Belief Propagation and Neural Computations

4.1 Belief propagation as a dynamical system

In recent years, we have seen several new ideas on neural computational models of Bayesian inference and belief propagation. The key idea behind these neural models is to transform the belief propagation equations into differential equations to allow computations to be carried out by a network of recurrent neurons. In this section, we will present some of these recent ideas that bring out the link between belief propagation and dynamical systems. To do that, we will first introduce the Hidden Markov Model (HMM) shown in Figure 4.1 using similar notations from [25]. The HMM is a special class of temporal Bayesian networks (called Dynamic Bayesian networks) for which the graphical models evolve over time. Let y_t denotes the observed inputs to the model at time *t* and x_t denotes the hidden cause or state that generates y_t. The inference process is to compute $P(x_t \mid y_t, ..., y_1)$. In a HMM, it is assumed that the hidden state x_t only depends on its previous state x_{t-1} according to its state transition $P(x_t \mid x_{t-1})$ (first-order Markov property). The observable y_t is generated by the conditional probability distribution $P(y_t \mid x_t)$. Then, using the belief propagation in the forward direction of the time sequence, we obtain the following equations for message passing:

$$m_t^i = P(y_t \mid x_t^i) \sum_j P(x_t^i \mid x_{t-1}^j) m_{t-1,t}^j$$

$$m^i_{t,t+1} = m^i_t / \sum_j m^j_t = P(x^i_t \mid y_t, ..., y_1),$$

where x^i_t denotes variable x in its i^{th} state at time t, and $m^i_{t,t+1}$ is the i^{th} element of the message from x_t to x_{t+1}. The above message is represented in the log domain as

$$\log m^i_t = \log P(y_t \mid x^i_t) + \log \sum_j P(x^i_t \mid x^j_{t-1}) m^j_{t-1,t}$$

by taking logarithm of both sides.

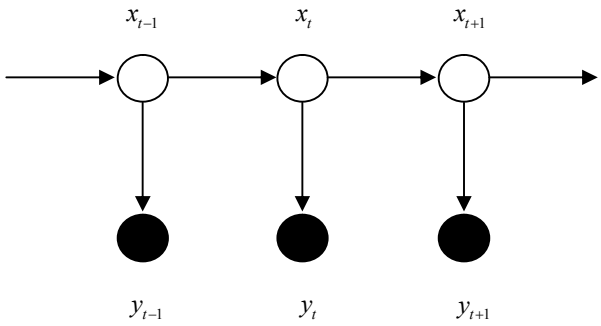


Figure 4.1: A Hidden Markov Model.

We will now present the main ideas of Rao's work ([25], [30]–[33]) on mapping belief propagation of a HMM to the neurodynamics of a network of noisy spiking neurons. The differential equation of a recurrent network of leaky integrate-and-fire neurons is given by

$$\tau \frac{dv_i}{dt} = -v_i + \sum_j w_{ij} I_j + \sum_j u_{ij} v'_j$$

where v_i is the membrane potential of neuron i, I_j is the synaptic current due to input neuron j, w_{ij} represents the strength of the synapse from input j to recurrent neuron i, v'_j denotes the synaptic current due to recurrent neuron j, u_{ij} is the corresponding synaptic strength, and τ is the membrane time constant. Considering the discrete form of the above equation together with nonlinear dendritic filtering functions f and g, we have

$$v_i(t+1) = f\left(\sum_j w_{ij} I_j(t)\right) + g\left(\sum_j u_{ij} v'_j(t)\right).$$

Mapping the above equation and the earlier log domain belief propagation equation, it gives

$$v_i(t+1) = \log m_t^i$$
$$f\left(\sum_j w_{ij} I_j(t)\right) = \log P(y_t \mid x_t^i)$$
$$g\left(\sum_j u_{ij} v_j'(t)\right) = \log \sum_j P(x_t^i \mid x_{t-1}^j) m_{t-1,t}^j$$

Then, using the assumption that $P(\text{neuron } i \text{ spikes at time } t) \propto e^{(v_i(t)-T)}$ where T is the threshold for spiking, it leads to

$$\begin{aligned} P(\text{neuron } i \text{ spikes at time } t+1) &\propto e^{(v_i(t+1)-T)} \\ &\propto e^{(\log m_t^i - T)} \\ &\propto m_t^i \end{aligned}$$

Therefore, the probability of spiking (or equivalently the instantaneous firing rate) of neuron i is directly proportional to the message m_t^i.

Deneve [4] also formulates the belief propagation equations into some form of differential equations to be computed by a recurrent process through a network of neurons (see Figure 4.2). However, in this case the log-likelihood ratio encoding is used to arrive at the following differential equation for a binary HMM:

$$\frac{dL_t}{dt} = r_{on}(1+e^{-L}) - r_{off}(1+e^{L}) + I_t$$

where

- $L_t = \log\left(\frac{P(x_t = 1 \mid s_{0\to t})}{P(x_t = -1 \mid s_{0\to t})}\right)$ is the log-likelihood ratio,
- $s_{0\to t}$ is the synaptic input received from time 0 to time t.
- r_{on} and r_{off} are the state transition rates, and
- I_t is the total contribution from sensory observations.

The synaptic input is represented by a vector of N binary variables $[s_t^i]_{i=1,\ldots,N}$ where $s_t^i = 1$ when the synapse i is activated during time t and $t+dt$. Each synapse is activated with a

probability $P(s_t^i = 1 \mid x_t = 1) = q_{on}^i dt$ if the state is 1 and $P(s_t^i = 1 \mid x_t = 0) = q_{off}^i dt$ if the state is 0. The total synaptic input is computed by

$$I_t = \sum_i w_i \delta(s_t^i, 1) - \theta$$

where w_i are the synaptic weights describing how informative a synapse i is about the state of the hidden variable (e.g., $w_i = \log \frac{q_{on}^i}{q_{off}^i}$), and θ is the bias, determined by how informative it is not to receive a spike, $\theta = \sum_i q_{on}^i - q_{off}^i$.

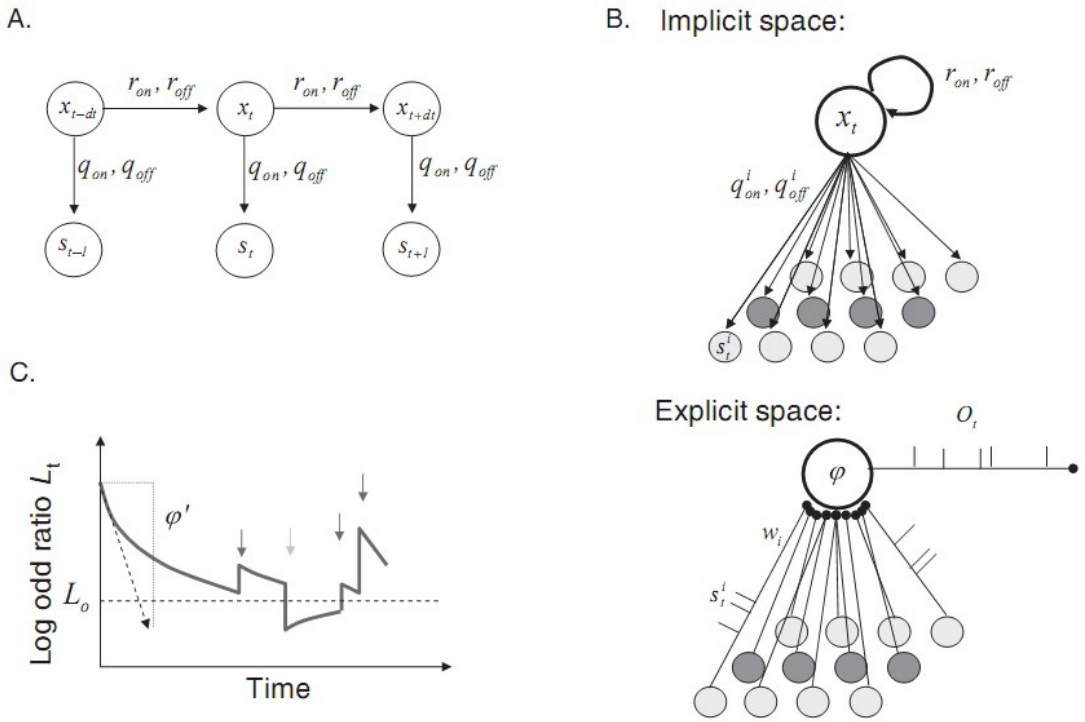


Figure 4.2: Bayesian inference in binary HMM using Spiking Neurons (Deneve [4]).

So far, we have seen that there could be more than one way of representing uncertainty (probability distribution) in neural computations. The notion of neural representation of uncertainty is a current research topic in computational neuroscience. For binary variables, there are generally two ways for neural coding (Knill and Pouget

[34]) of uncertainty, which are both supported by neurophysiological data. The first way is to encode using two populations of neurons: one for each state of the binary variable. The second method involves only one population encoding a likelihood ratio, like the encoding used in Deneve's formulation albeit in the log domain. From Rao [31], we see that the neurodynamics of a network is described by the differential equations of each neuron *i* in a recurrent network of leaky integrate-and-fire neurons:

$$\tau \frac{dv_i}{dt} = -v_i + \sum_j w_{ij} I_j + \sum_j u_{ij} v_j'$$

where v_i can either represent the firing rate of neuron *i* or its membrane potential. Generally in rate coding models, the neurodynamics of each neuron is described based on the single output variable v_i (i.e., $\frac{dv_i}{dt}$). On the other hand, each discrete variable in a graphical model has at least two states (say, $x_i = 1$ and $x_i = -1$). Hence, when using a neural circuit to represent a graphical model the direct mapping approach is to use a neuron to represent a state of a variable. This would then require a further step of normalization to be implemented for the group of neurons representing a single variable to sum. For example, in Rao [31], he suggests the normalization step should be computed using a separate group of neurons. In the case of a graphical model with only binary variables, we can make use of the trick that only one variable is needed to encode the full distribution since knowing $P(x_i = 1)$ leads to recovery of $P(x_i = -1) = 1 - P(x_i = 1)$. In fact, a common way of single-variable encoding of a binary distribution is to use the log-likelihood ratio given by

$$l = \log \frac{b(x_i = 1)}{b(x_i = -1)}$$

where the inverse is

$$b(x_i = 1) = \frac{e^l}{1 + e^l} = \frac{1 + \tanh(\frac{l}{2})}{2}, \text{ using } \tanh(\tfrac{l}{2}) = \frac{e^l - 1}{e^l + 1}.$$

Using the log-likelihood ratio we will only need to maintain a single value of *l* at each graphical node instead of a vector

$$\begin{pmatrix} b(x_i = 1) \\ b(x_i = -1) \end{pmatrix}.$$

Furthermore, for the case of binary pair-wise MRF, it helps to reparametrise the message updating equation from

$$m_{ij}(x_j) \leftarrow \sum_{x_i} \psi_{ij}(x_i, x_j)\phi_i(x_i, y_i) \prod_{k \in N(i)\setminus j} m_{ki}(x_i) \quad \text{(refer to Chapter 2)}$$

to the following form

$$\tanh(n_{ij}^{t+1}) = \tanh(\ln \psi_{ij}(x_i, x_j)) \tanh\left(\sum_{k \in N(i)\setminus j} n_{ki}^{t} + \ln \phi_i(x_i, y_i) \right)$$

by defining $\tanh(n_{ij}) = m_{ij}(x_j = 1) - m_{ij}(x_j = -1)$. For the proofs of the correctness of this reparametrisation, refer to Mooij and Kappen [35] and Richardson and Urbanke [36]. Note that n_{ij} is a single value. In addition, with reference to the Ising model presented in Chapter 2, the computation of the local magnetizations simplifies to

$$m_i = \tanh\left(\sum_{k \in N(i)} n_{ki} + \ln \phi_i(x_i, y_i) \right).$$

Ott and Stoop [37] make use of the above reparametrisation to show that the dynamical equations of a continuous Hopfield network can be derived from the message passing equations of belief propagation on a binary pair-wise MRF. This is done by first translating the difference equation

$$\tanh(n_{ij}^{t+1}) = \tanh(\ln \psi_{ij}(x_i, x_j)) \tanh\left(\sum_{k \in N(i)\setminus j} n_{ki}^{t} + \ln \phi_i(x_i, y_i) \right)$$

into its equivalent continuous time system:

$$\frac{dn_{ij}(t)}{dt} = -n_{ij}(t) + \tanh^{-1}\left(\tanh(\ln \psi_{ij}(x_i, x_j)) \tanh\left(\sum_{k \in N(i)\setminus j} n_{ki}(t) + \ln \phi_i(x_i, y_i) \right) \right).$$

Then through some algebraic manipulations (e.g., define $n_{ij} = w_{ij} v_i$) and approximations (based on certain assumptions), it matches the differential equations of the continuous Hopfield network of the form

$$\frac{dv_i(t)}{dt} = -v_i(t) + f\left(\sum_k w_{ki} v_k(t) \right) + K_i(t)$$

where v_i describes the activity of a neuron (e.g., membrane potential), f is a sigmoid function (in this case tanh()), w_{ij} are the symmetric synaptic weights, and $K_i(t)$ is an external signal or bias.

The significance of the research work reported in this section is that it suggests the viability of performing belief propagation using neural computations by transforming it into a dynamical system. However, it is still a fairly new idea restricted to simpler graphical models like HMM and binary pair-wise MRF, and there are many open problems yet to be solved. We felt that in order to extend the work to more generic classes of graphical models we would need to have a good understanding of inference algorithms (belief propagations) related to graphical models in general. From reviewing the various graphical models (Bayesian Networks, Markov Random Fields, and factor graphs) we learned that while the graphical representations vary, their respective belief propagation algorithms are mathematically equivalent. This is a useful fact because it gives us the flexibility to choose the graphical representation and belief propagation algorithm that would better suit neural implementation. Furthermore, understanding the concept of belief propagation and the different forms it takes in various graphical models is important when we look towards neural computations to implement it. For example, the pair-wise MRF has the simplest form of message update equations, and in the binary case it can be reparametrised to passing a single message between pairs of nodes.

The work by Ott and Stoop [37] prompted us to explore the concept of free energy and its relationship to belief propagation. We will defer the discussion on their link after presenting the free energy concept in the next section.

4.2 Belief propagation as free energy minimisation

In this section, we will present the idea due to Yedidia et al. [12], [14], [19], of casting belief propagation into the problem of minimising the *free energy* function defined in

statistical physics. Using Boltzmann's law and setting the temperature $T = 1$ (this only affects the scale of units here) the probability of a state x of a graphical model is

$$P(x) = \frac{1}{Z} e^{-E(x)}$$

where $Z = \sum_x e^{-E(x)}$ is the partition function and $-\ln Z$ is called the *Helmholtz* free energy.

For example, if the graphical model is a factor graph, we can define $E(x)$ of a state x as

$$E(x) = -\sum_a \ln f_a(x_a) .$$

Since we use belief propagation to compute $b_i(x_i)$ as an approximation to $P(x_i)$ for all variables x_i, we can determine the "goodness" of this approximation by defining a distance measure between the joint distributions $b(x)$ and $P(x)$. In this case, we use the Kullback Leibler distance and define

$$D(b(x) \,||\, P(x)) = \sum_x b(x) \ln \frac{b(x)}{P(x)} = \sum_x b(x) \ln b(x) + \sum_x b(x) E(x) + \ln Z .$$

This distance measure is always non-negative and is zero if and only if $b(x) = P(x)$.

Furthermore, if we define the *Gibbs free energy* by

$$F_{Gibbs}(b(x)) = U(b(x)) - H(b(x))$$

where

$U(b(x)) = \sum_x b(x) E(x)$ is the *average energy*

and

$H(b(x)) = -\sum_x b(x) \ln b(x)$ is the *entropy*,

then we have

$$F_{Gibbs}(b(x)) \geq -\ln Z .$$

In other words the Gibbs free energy is bounded below by the Helmholtz free energy and equality is achieved only when $b(x) = P(x)$. Hence, if we directly minimise the Gibbs free energy $F_{Gibbs}(b(x))$ we can obtain $P(x)$. However, in general where the number of variables is large, computing $F_{Gibbs}(b(x))$ becomes intractable as we need to sum over an exponential number of terms. Therefore a few approximations of $F_{Gibbs}(b(x))$ have been developed. Note: the advantage of describing the belief propagation using Gibbs free

energy instead of directly in terms of the joint probability distribution is that it allows analytically tractable approximation to $F_{Gibbs}(b(x))$ [12].

For example, the mean field approximation

$$b_{MF}(x) = \prod_i b_i(x_i)$$

assumes that $b(x)$ can be factorised as a product of individual node's beliefs. Thus, the mean field free energy is given by

$$F_{MF}(b(x)) = -\sum_{(ij)} \sum_{x_i, x_j} b_i(x_i) b_j(x_j) \ln f_{ij}(x_i, x_j) + \sum_i \sum_{x_i} b_i(x_i) \ln b_i(x_i) .$$

Another important free energy approximation is the Bethe approximation. The *Bethe* approximation decomposes the set of variables into smaller subset and sums the subsets' free energies to approximate $b(x)$. It takes advantage of the property that a singly connected graph can be represented by

$$P(x) = \frac{\prod_a P_a(x_a)}{\prod_i P_i(x_i)^{d_i - 1}}$$

where d_i is the number of neighbours of node x_i. For a factor graph, Bethe approximation yields

$$F_{Bethe}(b(x)) = U_{Bethe}(b(x)) - H_{Bethe}(b(x))$$

where

$$U_{Bethe}(b(x)) = -\sum_a \sum_{x_a} b_a(x_a) \ln f_a(x_a)$$

and

$$H_{Bethe}(b(x)) = -\sum_a \sum_{x_a} b_a(x_a) \ln b_a(x_a) + \sum_i (d_i - 1) \sum_{x_i} b_i(x_i) \ln b_i(x_i) .$$

For pair-wise MRF, Bethe approximation uses only the one-node and two-node beliefs. Since the entropy term computed by the Bethe approximation is exact for singly connected graph, it also computes the Gibbs free energy exactly. However, in general, the Bethe free energy does not form a bound on the Gibbs free energy. Furthermore, the beliefs computed by using the Bethe free energy may not be defined by any distribution.

Yedidia et al. [12] proved using Lagrange multipliers that the belief propagation algorithm can only converge to a fixed point that is also a stationary point of the Bethe approximation to the free energy. Following which, Heskes [38] further showed that *stable* fixed points of belief propagation must be local *minima* of the Bethe free energy (while the converse may not be true). The discovery of this equivalence between belief propagation and Bethe approximation resulted in the development of the generalized belief propagation based on *region-based* approximation [19]. The idea is to divide the graph into a set of regions and approximate the total free energy by summing over the regions' free energies. In generalized belief propagation, messages are passed between regions rather than between nodes, similar to the join tree algorithm where messages are passed between cliques of nodes.

It is important to note that the belief propagation does not directly correspond to a constrained minimisation of the free energy function. It is the stationary points between the two that are the same. Thus, prompted by this similarity, other researchers have developed algorithms that directly minimise the free energy (e.g., see Welling and Teh [39]), though these algorithms are slower than belief propagation they are guaranteed to converge. To date, research is still active on creating more practical and efficient belief propagation algorithms using formulations related to free energy.

4.3 Concluding remarks

The work by Ott and Stoop shows that the approach of transforming the belief propagation equations into differential equations can be applied, beyond HMM, to pair-wise MRF. It also gives us a novel idea that we could possibly implement belief propagation in a Hopfield network through free energy minimisation since we know the Hopfield network has an associated energy function of which the states of the network always converge to its local minima. The idea is if we could formulate the Bethe free energy function into an equivalent Hopfield network's energy function, then we could implement the belief propagation using a Hopfield network. This idea is further supported by the following observations. Interestingly, there exists an isomorphism

between the Hopfield network and the Ising model of magnetism at temperature zero [40]. It turns out that the energy function of an Ising model has the same form as the energy function of a Hopfield network:

Ising model: $E(x_1,...,x_n) = -\sum_{i,j} J_{ij} x_i x_j - \sum_i h_i x_i$

Hopfield network: $E(x_1,...,x_n) = -\frac{1}{2}\sum_{i,j} w_{ij} x_i x_j - \sum_i \theta_i x_i$

where θ_i is the threshold of unit x_i. Furthermore, we know that the binary pair-wise MRF is the Ising model at temperature one (refer to Chapter 2).

Returning to the topic on belief propagation and neural circuits, Rao ([25], [30]–[33], [41]) has shown that neural circuits could implement belief propagation for Bayesian Inference in two types of simple graphical models, namely a Hidden Markov Model and a three-level hierarchical model. This is done by transforming the belief propagation's message passing equations into the log domain and mapping it mathematically to the differential equations that describe the dynamics of neural circuits. In his earlier version of research [42], the mapping to linear recurrent networks with firing-rate dynamics allows inference to be implemented but it requires an approximation of a log-sum with a sum-of-logs and hence the inference is only approximate. In subsequent works, exact inference is achieved by applying the log function as dendritic filters to a network of leaky integrate-and-fire neurons. With log dendritic filters the approximation of log-sum is no longer required. Furthermore, unlike the linear recurrent networks which compute the target variables of firing rates, the target variables in the spiking networks are the membrane potentials of neurons.

Deneve [4] has also proposed the use of spiking neurons for Bayesian inference, but unlike Rao's model where each neuron encodes a state of a variable, each neuron encodes the log-odds ratio for a preferred binary-valued state (e.g., the ratio of the on and off states of a binary variable), which makes her model applicable only to binary variables. In Rao's model, the probability of spiking for each neuron is proportional to the likelihood function, whereas in Deneve's model a neuron spikes only when the difference between its log-odds ratio and a predicted value of the log-odds ratio reaches a

certain threshold. While Rao's model has an explicit representation of probabilities, Deneve's model does not. Nevertheless, she has shown that her model is similar to leaky integrate-and-fire neurons.

Note that both neural computation models encode the probabilities in the log domain. As Rao pointed out, there is growing evidence that the brain uses log likelihood ratios during decision making, and the advantage of implementing probabilistic inference in log domain is that it transforms the multiplication and division operations into additions and subtractions, which are readily implemented through excitation and inhibition in neural circuits.

In a related work, Beck and Pouget [43] derive a quadratic nonlinear, first-order dynamical system for exact Bayesian inference in a Hidden Markov Model, and show that under certain conditions, Rao's log-likelihood model and Deneve's log-odds ratio model follow naturally from their nonlinear equations. In contrast, their model is an explicit representation of the probability distribution (and hence likelihood), and they further argue that direct likelihood implementation for exact Bayesian inference for HMM only requires divisive normalization and dendritic multiplication, both of which have been observed in neurons (references given in their paper). Their standpoint is that not only can their model be mapped onto a biologically plausible network but it is also more consistent with physiological observations. They further argue that while log transformation eliminates multiplicative terms, this transformation can also result in a dynamical system that has an even more complicated nonlinear interaction associated with the recurrent terms.

The above groups of researchers are concerned primarily with developing biologically plausible neural computation models. If our focus is to make use of these findings and couple with the potential efficient computation of such models to implement Bayesian inference algorithms in more efficient ways, then the type of representation to use for probabilistic values becomes a choice in view of the inference algorithm rather than a constraint (the need to be consistent with physiological data). More importantly,

these research works have shown the potential of using neural computation for Bayesian inference and belief propagation. Nonetheless, these are fairly new works and the authors themselves acknowledge that there are many open problems and challenges. Among these challenges there are two problems that we found interesting. Firstly, while a mapping can be done from inference equations to neural computational models, there is the issue of how learning is carried out in these model to capture the equivalence of parameters like those of conditional probability tables in a Bayesian network. In other words, what is the associated learning mechanism of such neural models? Secondly, the neural models are restricted to mainly Hidden Markov Models, so how can we generalize or adapt the computation mechanism to allow Bayesian Inference algorithms like local message passing or belief propagation for generic Bayesian networks and other graphical models?

Lastly, we have to ask ourselves why the interest in spiking neuron models? Besides opening up a new paradigm for Bayesian Inference, spiking neurons have the potential of fast and efficient implementations, which will make it appealing to real time pattern recognition and decision making systems. Also, spiking neuron models have higher biological fidelity, and fruitful research in this area will help bridge the gap of higher level cognitive models of knowledge representations and mental models with a lower level of neuronal information processing. This is especially so if the existing knowledge base consists of Bayesian networks models, and it will make it easier and more accurate to integrate such high level knowledge into a bio-inspired cognitive architecture with neuronal level of processing units while retaining the integrity of the knowledge captured. In conclusion, we believe the followings are two good strategies for use in future research to extend neural computations to more generic graphical modes:

- Attempt to formulate the belief propagation in its free energy minimization form and make use of Hopfield network-like neural computations that converge to minima of its defined energy function.
- Translate the message updating equations of the belief propagation into a continuous form of differential equation (dynamical system) and make use of the neurodynamics of a network of neurons (for example, recurrent spiking neurons) to converge to the

approximate (possibly exact) solutions.

Part II

Loopy Belief Propagation in Bayesian Networks: Error Analysis, Correction and Improvements

Chapter 5

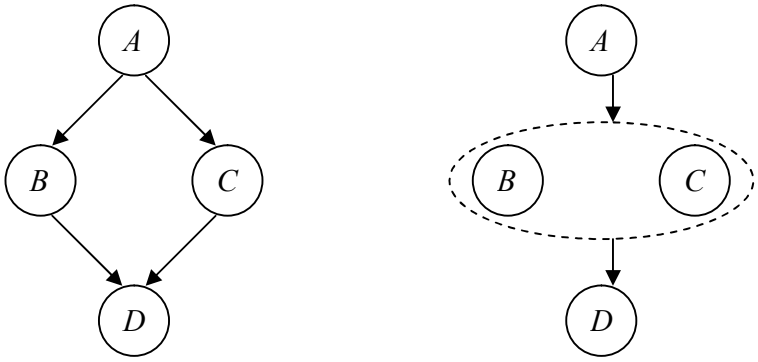
An Alternative Way to Improve Loopy Belief Propagation in Bayesian Networks

In this chapter we present an alternative thinking on how we can improve the performance of loopy belief propagation when applied to Bayesian networks and to develop approximate belief propagation models for approximate inference. Our proposal is the result of surveying some established algorithms for carrying out inference in Bayesian networks combined with inspirations from more recent research results related to understanding loopy belief propagation. We shall present the related algorithms and research results in the following sections and our proposal in the last section.

It is widely known that exact inference in loopy graphs is computationally difficult. Hence there has been a lot of effort spent in the area of developing practical approximate inference algorithms, which continues till today. Inference computation in Bayesian Networks has been proven to be NP-Hard [8] in general. Nevertheless, we believe there are more theories on the behaviour of belief propagation on loopy graphs that can be discovered. Furthermore, there is potential in using these discoveries to improve the performance of approximate inference using loopy belief propagation.

Many real world applications are much more complex than tree structured Bayesian network can model and they usually entail multiply connected networks, for which

Pearl's algorithm is not directly applicable. Figure 5.1(a) shows a multiply connected network, which we also term it as a loopy graph. The main problem with applying Pearl's algorithm is that the node (and consequently every other nodes) that sends out an update message to its neighbour receives unwanted 'feedback' resulting in non-convergence or wrong posterior probabilities being computed, e.g., $D \to C \to A \to B \to D \to C \to \cdots$.



(a) (b)

Figure 5.1: (a) A multiply connected network (b) A clustered network.

To solve this problem, we need to resort to a technique called clustering. Clustering in effect converts the original multiply connected network into a singly connected network, but the new network now has clusters of nodes at its vertices. Figure 5.1(b) shows such a cluster tree. Lauritzen and Spiegelhalter first proposed the Lauritzen-Spiegelhalter (LS) algorithm to build the tree in 1988 [44]. Jensen et al. [45] proposed a modified version and it was subsequently adopted in the Hugin [46] (a well-known Bayesian Networks commercial software) implementation, and thus known as the Hugin architecture. There are also other variants like the Shafer-Shenoy [47] architecture, and collectively they are known as join tree (also clique/junction tree) algorithms. Throughout this thesis we will use the terms join tree and junction tree interchangeably. They are one of the most popular classes of exact inference algorithms for Bayesian Network in practice. Other exact inference algorithms are available, for example, loop-cutset conditioning for Pearl's algorithm and variable eliminations. There are also a variety of approximate inference algorithms like the popular Markov Chain Monte Carlo (MCMC) method and Gibbs sampling method. But again, it has been proven that approximation algorithms for

inference are also NP hard. What this means is that approximate inference algorithms that run in polynomial time given a certain error bound will still fail on certain classes of inputs. The clustering algorithms work efficiently when the network is sparse. The approximation algorithms have been motivated to solve densely connected networks. As expected, there is no clear winner that works well for all kinds of networks, and with the inherent disadvantage of NP hard complexity there is not likely to be one. Therefore, researchers and engineers who wish to apply a specific inference algorithm should have a good understanding of its effects on accuracy and computational load, which have direct impact on different applications.

5.1 Loopy belief propagation

Loopy belief propagation is a widely used approximate inference algorithm for graphical models. It was first introduced by Pearl [5] as an approximation method by applying the same belief propagation algorithm designed to compute exact marginal probabilities on singly connected Bayesian networks to those that contained loops. The main idea of loopy belief propagation is to continue the message passing process until a stable state of beliefs are obtained or when some terminating conditions are met. Subsequently, many results that showcased the excellent performance of loopy belief propagation have been published, such as the success reported in iterative decoding of turbo codes [18]. Also, there were empirical studies like the one conducted by Murphy et al. [17] that demonstrated the good performance of loopy belief propagation when it had fast convergence. These results prompted many researchers to further investigate the properties of loopy belief propagation so that theoretical measures could be developed to assess whether it is appropriate to apply it to a particular problem.

The two main issues in the analysis of loopy belief propagation are convergence and accuracy. In general, loopy belief propagation may not converge, or could oscillate about some points, when applied to graphs that contain loops. There has been extensive research conducted on finding sufficient conditions for the convergence of loopy belief

propagation [48]–[51]. On the other hand, there were not many analytical findings on the accuracy of loopy belief propagation in its early days until the ground breaking discovery made by Yedidia et al. [12], that the loopy belief propagation can only converge to a fixed point that is also a stationary point of the Bethe free energy of the graphical model. Following which, Heskes [38] further showed that *stable* fixed points of belief propagation must be local *minima* of the Bethe free energy (while the converse may not true). However, it is important to note that the loopy belief propagation algorithm is not an iterative minimisation algorithm. It does not attempt to minimise the Bethe free energy function in its iterations. The equivalence is only at the converged fixed point that matches the stationary point. Nevertheless, this discovery of the equivalence has sparked off huge research interest in designing better approximation methods for belief propagation as well as prompting researchers to look towards alternative formulations to analyse the inference accuracy. For example, Tatikonda and Jordan [48] related loopy belief propagation to Gibbs measures and produced conditions under which it will converge. Wainwright et al. [52] gave an alternative interpretation of loopy belief propagation as tree-based reparametrisation of the original distribution on the graph and it also led to a derivation of bounds on the errors of loopy belief propagation. Many researchers have also attempted to define accuracy bounds on loopy belief propagation. Shi et al. [53] provided a good summary and comparisons of recent research development on accuracy bounds while presenting a tighter error bounds for loopy belief propagation.

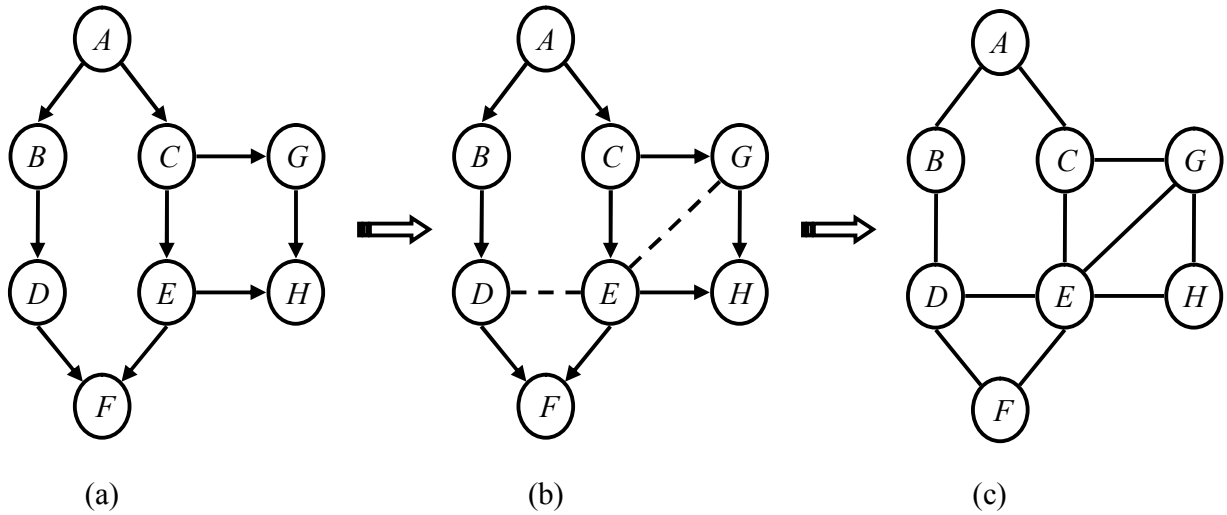
5.2 Join tree algorithm

In this section we shall first present the Lauritzen-Spiegelhalter (LS) join tree algorithm [44] and follow by a short comparison of it to the junction tree variant by Jensen et al. [45]. The LS algorithm belongs to a class of inference algorithms called clustering techniques. The clustering approach is able to handle a multiply connected Bayesian network by converting it into a singly connected secondary network (join tree) with each new node being a clique of nodes from the original network. Once the converted network is formed, the join tree propagation process is applied whenever new evidence needs to

be injected into the network. The join tree propagation process, based on the principle of “local computations”, propagates evidence through message passing between nodes in two phases. The phrase "local computation" refers to a computation that involves only a small number of variables. The adjective "local" is used because the variables involved in a given computation are near each other when the relations among the variables are represented graphically [47]. The general join tree algorithm has the following two main processes:

- Network transformation: a static process that transforms any given Bayesian Network into a suitable join tree structure. We can consider it to be an initialisation step, and it is required only once at the beginning.
- Evidence propagation: a dynamic process that updates the posterior probabilities of the whole network once evidence is entered into (instantiated) a particular node in the network. This process carries out a 2-Pass propagation process.

The process for the network transformation involves graph theoretic approaches, which are described next.



(a) (b) (c)

Figure 5.2: (a) Original Bayesian network (b) “Marry” parents (c) Moral graph.

Moral Graph

The moral graph of a directed graph $G = (V, E)$ is formed by 'marrying' (adding a link between) parent nodes of each common child node and once all links have been added, converting the directed graph into an undirected graph by dropping the directions. It is represented by $G_u = (V, E')$ where E' is the augmented list of original edges that now contains the 'married' links. An example of a moral graph is given in Figure 5.2.

Triangulated Graph

An undirected graph is called triangulated if every cycle of length four or greater contains a chord.

Maximum Cardinality Search

Given $G_u = (V, E)$ an undirected graph, the maximum cardinality search is the following process:

Initialisation: Label an arbitrary node as 1.

For i = 2 to N (total number of nodes in graph)

- *Compute for each remaining unlabelled node, the total number of labelled adjacent nodes it has.*
- *Label the node with largest number of labelled adjacent nodes with value i, breaking ties arbitrarily.*
- *If at any stage a node i has labelled adjacent nodes j and k that are not adjacent to each other, terminate with failure.*

If all nodes are labelled, terminate with success.

The maximum cardinality search was originally proposed by Tarjan and Yannakakis [54] to test the chordality of graphs. An undirected graph is triangulated if and only if the maximum cardinality search terminates with a success (i.e. all nodes are labelled). In the event that the graph is not triangulated, the algorithm can "triangulate" it by performing fill-ins. A single fill-in adds an arc between the pair of labelled adjacent nodes that would otherwise result in the failure condition being satisfied. It is clear that by repeatedly

performing fill-ins we can triangulate a graph. Hence, we can triangulate an undirected graph using the following procedure:

- *Step 1: Perform maximum cardinality search on graph.*
    - *If maximum cardinality search terminates with success, end.*
- *Step 2: Perform fill-ins on pairs of numbered adjacent nodes that contributed to failure condition*
    - *Repeat step 1 with the new graph.*

Figure 5.3 illustrates the triangulation procedure on the moral graph obtained in Figure 5.2(c). The first labelling iteration terminates at the 6^{th} node, which requires a fill-in. After the fill-in (dashed link) the labelling process is restarted from node 1. It then terminates at node 4 and requires another fill-in between node 1 and 3, after which the labelling process is started again and finally terminates successfully with all nodes labelled and the graph triangulated.

The above triangulation procedure is simple but may not be efficient and does not guarantee an optimal triangulated graph with minimum fill-ins added. We shall see after defining a clique that minimum fill-ins is desired since more fill-ins will increase the sizes of the cliques being formed and thus increase the computational cost of evidence propagation in the join tree. In the worst case, the triangulated graph is so densely connected that there is only one resulting clique that contains all nodes of the original network. Then again, it has been shown that search for optimal triangulation is NP hard. In fact, triangulation of graphs has wide applications and it is a research area of its own right.

Cliques

A complete graph is a graph where all the nodes are pair-wise connected. A clique is a maximal complete subgraph. For example, cliques for the triangulated graph in Figure 5.3 are $\{ABD\}$, $\{ACE\}$, $\{ADE\}$, $\{DEF\}$, $\{CEG\}$ and $\{EGH\}$. Note that $\{AB\}$ (and likewise $\{AC\}$, $\{GH\}$, etc) does not form a clique because it is not maximal.

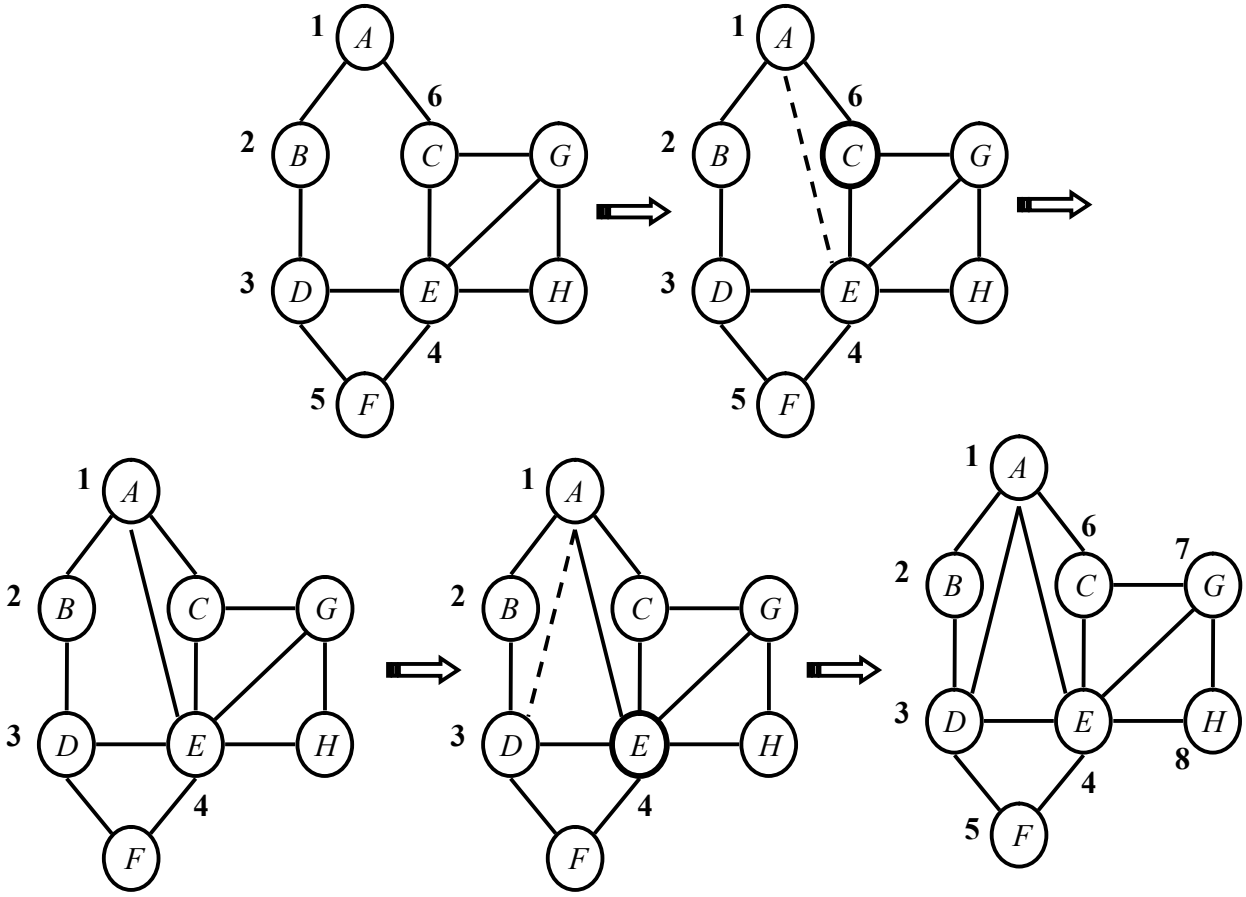


Figure 5.3: Triangulating a graph using maximum cardinality search with fill-ins.

Running Intersection Property

Let $G_u = (V, E)$ be an undirected graph with *n* cliques. An ordering of the cliques $(Clq_1, ..., Clq_n)$ is said to have the running intersection property if for every $i > 1$, there exists a $j < i$ such that

$$Clq_i \cap (Clq_1 \cup Clq_2 \cup \ldots \cup Clq_{i-1}) \subseteq Clq_j .$$

Once a moral graph has been triangulated and the cliques identified, an ordering of the cliques that satisfies the running intersection property can be obtained as follows:

- *Label the nodes by maximum cardinality search;*
- *Order the cliques according to their highest labelled nodes.*

With the running intersection property, assigning each clique Clq_i a parent clique Clq_j generates a join tree. When there is more than one choice for the parent clique, the assignment is arbitrary. Note that there will be at least a clique with no parent, which can be thought of as the root of the join tree. Figure 5.4 shows a table of the ordering of the cliques for the triangulated graph in Figure 5.3, which satisfies the running intersection property and the corresponding join tree diagram.

Clique	**Highest label**	**Clique Order**	**Inter-section**	**Parent Clique**
$\{ABD\}$	3	**1**	**-**	-
$\{ADE\}$	4	**2**	$\{A\}$	1
$\{DEF\}$	5	**3**	$\{DE\}$	2
$\{ACE\}$	6	**4**	$\{AE\}$	2
$\{CEG\}$	7	**5**	$\{CE\}$	4
$\{EGH\}$	8	**6**	$\{EG\}$	5

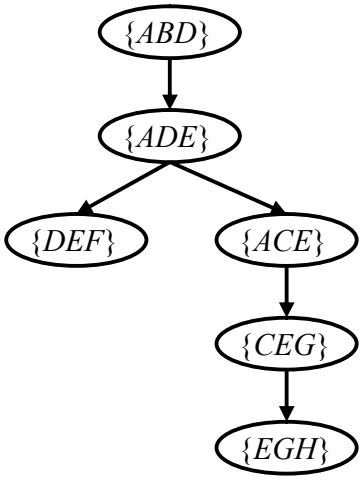


Figure 5.4: Clique ordering and join tree.

Potential Representation

To fully describe the LS algorithm, we need to introduce the notion of potential representation (function) of a probability distribution. Given the joint probability distribution of a Bayesian network with *n* nodes $(x_1,...,x_n)$ as

$$P(x_1,...,x_n)=\prod_{i=1}^{n} P(x_i \mid Parents(x_i)),$$

a potential representation of a joint probability distribution over $V=\{x_1,...,x_n\}$ is given by $(\{W_1,...,W_m\},\phi)$ such that, for some constant *k*,

$$P(V)=k\prod_{i=1}^{m}\phi(W_i)$$

where each W_i is a subset of V and ϕ is the potential function that maps each possible subset of V to a unique real number. In implementation, the potential functions are often viewed and handled in table form, similar to conditional probability tables, and represented by matrices. There are three operations on potential functions required in the LS algorithm:

- Multiplication

Let $W = W_1 \bigcup W_2$, then the multiplication of two potentials is denoted by

$$\phi(W) = \phi(W_1) * \phi(W_2)$$

where

$$\phi(w) = \phi(w_1) * \phi(w_2)$$

is such that w_1 and w_2 are the states of W_1 and W_2 respectively, which are consistent to the state of w. For example, $\phi(\{a_2,b_1,c_3\})=\phi(\{a_2,b_1\})*\phi(\{b_1,c_3\})$, $\phi(\{a_1,b_1,c_2\})=\phi(\{a_1,b_1\})*\phi(\{b_1,c_2\})$, $\phi(\{a_3,b_2,c_2\})=\phi(\{a_3,b_2\})*\phi(\{b_2,c_2\})$, etc.

- Division

Let $W = W_1 \setminus W_2$ and $W_2 \subseteq W_1$, then the division operation is denoted by

$$\phi(W) = \frac{\phi(W_1)}{\phi(W_2)}$$

where

$$\phi(w) = \frac{\phi(w_1)}{\phi(w_2)}$$

is such that w_1 and w_2 are the states of W_1 and W_2 respectively, which are consistent to the state of w. For example, $\phi(\{a_2\})=\phi(\{a_2,b_1\})/\phi(\{b_1\})$, $\phi(\{b_2,c_2\})=\phi(\{a_3,b_2,c_2\})/\phi(\{a_3\})$, $\phi(\{b_1\})=\phi(\{a_1,b_1,c_2\})/\phi(\{a_1,c_2\})$, etc. The case of 0/0 is defined as 0.

- Marginalisation

The marginalisation of potential functions is similar to the marginalisation of joint probability distributions. Let $S = W \setminus R$, then the marginalisation of $\phi(W)$ onto set S is denoted by

$$\phi_W^{\downarrow S} = \sum_{W/S} \phi(W) = \sum_R \phi(W)$$

where

$$\phi_W^{\downarrow S}(s_i) = \sum_j \phi(s_i \cup r_j)$$

and the summation over j means summing over all states of R for each state s_i of the set S. We can view marginalisation as a projection of a potential function onto the space of its subset.

Network Transformation

The first part of the LS algorithm, which is the transformation process, is carried out as follows:

Given a Bayesian network represented by $G = (V, E)$ *:*

1. *Create the moral graph* G_m.
2. *Triangulate the moral graph* G_m *to obtain* G_t *with cliques* $(Clq_1, ..., Clq_n)$ *ordered with running intersection property.*
3. *Construct the join tree by identifying one parent for each clique* Clq_i *where the notion of parent is induced by the running intersection property, i.e.* $Parent(Clq_i) = Clq_j$ *if* $S_i = Clq_i \cap (Clq_1 \cup Clq_2 \cup \ldots \cup Clq_{i-1}) \subseteq Clq_j, j < i$. *The assignment is arbitrary if there are two or more parents for each clique. On the other hand, a clique with no parent is a root.*
4. *Generate the potential representation (functions/tables):*
   a. *For each* $v \in V$ *(vertex in original graph* $G = (V, E)$*), assign* v *to a unique* Clq_i *such that*

      $$\{v\} \cup Parents(v) \subseteq Clq_i$$

      and denote this assignment by $f(v) = i$. *At the end of the assignment process, if a clique is not assigned with any v, then set it as*

      $$\phi(Clq_i) = 1 \text{ (vector).}$$

   b. *Compute potential function* $\phi(Clq_i)$ *for each clique by*

$$\phi(Clq_i) = \prod_{f(v)=i} P(v \mid Parents(v))$$

Note 1:

In step 4(a), it is always possible to find a Clq_i such that

$$\{v\} \bigcup Parents(v) \subseteq Clq_i \qquad \forall v \in V .$$

This is because:

- After moralisation, the parent nodes in original graph are linked in G_m. Therefore $\{v\} \bigcup Parents(v)$ is a complete subgraph of G_m.
- After triangulation, $\{v\} \bigcup Parents(v)$ is still a complete subgraph of G_t and since a clique is a maximal subgraph of G_t there is at least one clique that contains $\{v\} \bigcup Parents(v)$.

Note 2:

The set of potential functions $\{\phi(Clq_i)\}$ computed in step 4(b) is a potential representation of $P(V)$. The proof is as follows:

$$\begin{aligned} \prod_i \phi(Clq_i) &= \prod_i \prod_{f(v)=i} P(v \mid Parents(v)) \\ &= \prod_v P(v \mid Parents(v)) \\ &= P(V) \end{aligned}$$

The second equality is obtained from the fact that each *v* is assigned to exactly one clique.

With the termination of the transformation process, a new network structure (that is a join tree ordered with running intersection property) is formed, and a set of cliques and their corresponding potential functions determined.

2-Pass Evidence Propagation

The aim of the LS algorithm's next process - evidence propagation, is to update the whole join tree by some updating mechanism that operates on clique potential functions such that the resulting potential functions are marginal probability distributions over variables found in each clique, which are subsets (and thus smaller) of the original set of variables. Thus, the posterior probability distribution of any variable can be computed by further marginalising the potential function. This updating mechanism is

called 2-Pass propagation process. It consists of a Full Inward Pass and a Full Outward Pass, and is described next.

Evidence Propagation *{*

Full Inward Pass {

Logical control rule:

A clique performs a single inward pass to its parent only after all its child cliques have completed their single inward pass operations.

A Single Inward Pass: from child (Clq_j) to parent (Clq_k){

Compute $\phi(Clq_j)^{\downarrow S_{jk}} = \sum_{Clq_j \setminus S_{jk}} \phi(Clq_j)$ *and pass it as message to parent clique*

Update

$$Child\ \phi(Clq_j) = \frac{\phi(Clq_j)}{\phi(Clq_j)^{\downarrow S_{jk}}}$$

$$Parent\ \phi(Clq_k) = \phi(Clq_k) * \phi(Clq_j)^{\downarrow S_{jk}}$$

}

}

Full Outward Pass {

Logical control rule:

A clique performs a single outward pass to each of its child clique only after its parent clique has completed its single outward pass operation.

A Single Outward Pass: from parent (Clq_k) to child (Clq_j){

Compute $\phi(Clq_k)^{\downarrow S_{jk}} = \sum_{Clq_k \setminus S_{jk}} \phi(Clq_k)$ *and pass it as message to child clique*

Update

$$Child\ \phi(Clq_j) = \phi(Clq_j) * \phi(Clq_k)^{\downarrow S_{jk}}$$

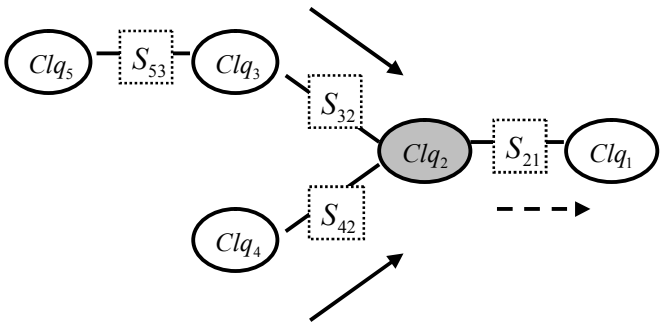
}

}

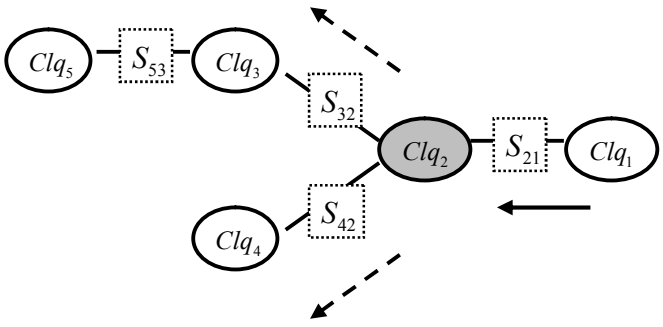
}

When the full inward pass terminates, we obtain the (updated) joint probability distribution of the root clique. This property plays a central role in the LS algorithm. In fact, this property is also important to the Hugin and Shafer-Shenoy algorithms. It is this property that provides a starting point for the full outward pass in the LS algorithm. The full outward pass is similar to the full inward pass. It recursively carries out single outward pass that starts at the root clique and ends at the leaf clique. The aim of each single outward pass is to marginalise onto the separator (which is the set of common variables between two cliques) that links it to its child (as compared to parent in the inward pass) clique to allow the child clique to update its potential function to become joint probability distribution.

Notice that besides the added logical control over the inward and outward pass processes, we have also replaced the notation of the separator between Clq_j and Clq_k by S_{jk}. Since a child clique is assigned a single parent clique whereas a parent clique can have several child cliques, we can use the new notation's indices to indicate the relationship. Note also that given the logical control rule for inward pass, it implicitly implies that the full inward pass starts off with any leaf clique since it does not have a child clique and ends at the root clique. Similarly, the logical control rule for outward pass implies that the full outward pass starts from the root clique and ends at the any leaf clique. Figure 5.5 shows how the logical control rule is applied to manage message propagation at a clique with two child cliques. During the full inward pass, Clq_2 performs inward pass towards Clq_1 only after receiving messages from Clq_3 and Clq_4. Likewise, an outward pass towards either Clq_3 or Clq_4 can be made only after Clq_1's outward pass.



(a) Inward pass at Clq_2



(b) Outward pass at Clq_2

Figure 5.5: Logical control example.

In the literature, the term "architecture" is often used in place of "algorithm" when comparing different join tree propagation implementations. It has the implication that although the implemented architectures vary, their propagation algorithms or theories still belong to the same class of algorithms and are fundamentally equivalent. Here, architecture refers to the propagation algorithm in relation to the storage design of data structures (usually matrices) for potential functions and the logical control of message passing and updating.

Hugin Architecture and Junction Tree

Jensen et al. [45] proposed a modified version of the LS architecture in 1990. It was developed into the software called Hugin [46] at University of Aalborg in Denmark. Hugin subsequently became a commercial product, which is widely used today. In what follows, we describe the Hugin architecture as found in the open literature. It may not be representative of the implementation of the current version of the software. Furthermore, we believe the software version could have optimisations or enhancements incorporated, as would any commercial software. One main difference between Hugin architecture and LS architecture is that it maintains a potential function/table $\phi(S_{jk})$ for each separator. All entries of the potential table for each $\phi(S_{jk})$ are initialised with 1's to admit the following potential representation

$$P(V) = \frac{\prod_i \phi(Clq_i)}{\prod_{j,k} \phi(S_{jk})}$$

which is equivalent to the LS architecture's potential representation. The Hugin architecture also adopts a similar 2-Pass evidence propagation process to update $\phi(Clq_i)$ into $P(Clq_i)$. Due to the presence of new potential functions on the separators, the Hugin propagation algorithm deviates from the LS propagation algorithm as follows:

- *Additional initialisation step: set all* $\phi(S_{jk}) = 1$

- *A Single Inward or Outward Pass (also called absorption process): from* Clq_j *to* Clq_k
  - *At clique* Clq_j*: compute* $\phi(Clq_j)^{\downarrow S_{jk}} = \sum_{Clq_j \setminus S_{jk}} \phi(Clq_j)$ *and pass it as a message to separator* S_{jk}
  - *At* S_{jk}*: compute* $\frac{\phi(Clq_j)^{\downarrow S_{jk}}}{\phi(S_{jk})}$ *and pass it as a message to clique* Clq_k*, then update* $\phi(S_{jk}) = \phi(Clq_j)^{\downarrow S_{jk}}$
  - *At clique* Clq_k*: update* $\phi(Clq_k) = \phi(Clq_k) * \phi(S_{jk})$

Note that the inward pass and outward pass are identical in this case. In Hugin architecture's terminology, this single message passing process is called *absorption* and the full inward and outward passes are known as Collect-Evidence and Distribute-Evidence respectively. The aim of the absorption process is to make a separator consistent. A separator is said to be consistent if it contains the information its two neighbouring cliques Clq_j and Clq_k have in common, i.e. $\phi(Clq_j)^{\downarrow S_{jk}} = \phi(S_{jk}) = \phi(Clq_k)^{\downarrow S_{jk}}$. If all separators are consistent, then the join tree is said to be consistent and we have $\phi(Clq_i) = P(Clq_i)$. The Collect-Evidence and Distribute-Evidence processes if carried out on a junction tree (which we will discuss next) will result in a consistent join tree and the proofs are given in [55].

We saw earlier that the transformed join tree must contain ordered cliques that satisfy the running intersection property for the evidence propagation to be correct. The Hugin architecture works on an alternate type of join tree that provides a more flexible logical control for message passing. This alternate join tree is called junction tree (thus the Hugin algorithm has also been widely referred to as the junction tree algorithm). A junction tree is a cluster tree that satisfies the following property:

(Junction Tree Property) For every pair of cliques U *and* V *with intersection* S*, all cliques on the unique path between* U *and* V *contain* S.

The construct of a junction tree induces a partial order on the cliques that satisfies the running intersection property [56]. This means that if we pick any vertex (clique) in a junction tree and carry out a well-ordered ordering process (e.g., breadth-first or depth-first) starting from this clique, the resulting total ordering on the set of cliques has the running intersection property. In other words, any clique in a junction tree can be designated as the root for evidence propagation. The LS architecture identifies and fixes the root clique upon construction of the join tree and determines the logical control required over the 2-Pass propagation process. The Hugin architecture has to implement a flexible logical control to cater to the dynamic designation of the root clique. In other words, the logical control must be able to ensure that the right sequence of full inward pass and full outward pass are being carried out at runtime given any root designation. We can achieve this by recursively requesting neighbouring cliques that are outward from the root to send their update messages inward. This will cause the requests to reach the leaf cliques eventually and the full inward pass can then be started. In short, by recursively sending requests outward from the root, it maps out the full path for message sending. The Hugin architecture's logical control for the 2-Pass evidence propagation process is described below:

Arbitrarily choose a clique as the root

Collect-Evidence {

- *starting from the root, recursively request all neighbouring cliques to send their update messages*
    - *Clq_j can send its update message to Clq_k only after receiving messages from all neighbouring cliques except Clq_k*
- *the absorption process is performed with each requested message sent*

}

Distribute-Evidence {

- *starting from the root, recursively send an update message to each neighbouring cliques*
- *the absorption process is performed with each requested message sent*

}

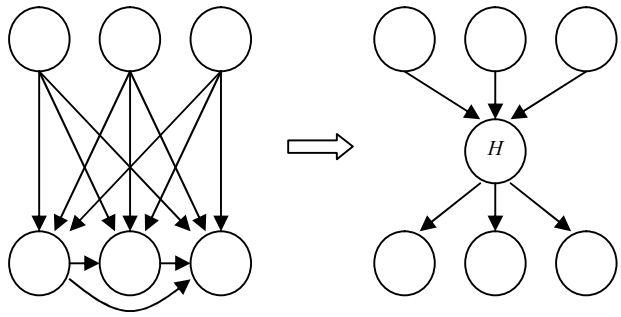
Once a junction tree has been constructed, altering the choice of the root clique does not reduce the amount of message passing or number of operations on potential tables. It only changes the sequence at which the potential table of a clique is updated to its marginal distribution and ensures that the selected root clique is the first to attain it. However, we can make good use of this observation to implement an optimised version of the propagation process called query-driven message passing [57] for diagnostic applications where small number of variables are queried over a Bayesian network with many variables. The idea is that if inference is required on single query variable or variables contained in a single clique, we can set the clique that contains the variable(s) as the root clique and call the Collect-Evidence process (full inward pass). By the end of the process we will obtain the potential table of the root clique as its marginal distribution and there is no need to perform the Distribute-Evidence process (full outward pass). We can further extend it to handle dynamic evidence by keeping track of the status (old/new with respect to new evidence input) of messages using a set of Boolean message flags. Finally, we end this section by pointing out that although the junction tree was initially closely associated with the Hugin architecture, it can also be used by the LS and Shafer-Shenoy architectures. Similarly, Hugin architecture can be applied to join trees with running intersection property too.

5.3 Using hidden nodes to compact network structure or to compensate for conditional dependency errors

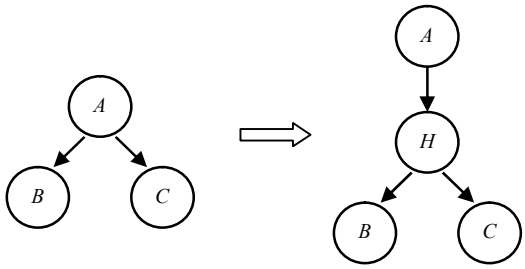
In this section, we present a few methods of training hidden nodes that are intentionally introduced into a Bayesian network to alter the network structure resulting in a more compact structure or for the purpose of accounting for conditional dependency errors. Binder et al. [58] illustrate using the example shown in Figure 5.6(a) that introducing hidden nodes can lead to more compact representation. In this example, assuming all variables are three-valued, the original network on the right requires 708 parameters while the network with a two-valued hidden node requires only 45 independent parameters. Furthermore, they assert that using hidden nodes on networks with

significant local structure leads to a more concise representation of the joint distribution on observable variables, which in turn requires fewer examples for learning.

Another motivation for introducing hidden nodes into a Bayesian network is to improve the conditional independency model of a network structure learned from data as proposed by Kwoh and Gillies [59]. In designing or learning Bayesian networks, the assumption that variables given the same parent are conditionally independent usually does not hold in practice and this leads to inaccurate inference results. They propose introduction of hidden node where there is conditional dependency between variables to address this issue. Figure 5.6(b) illustrates the introduction of hidden node in this way. They demonstrate the success of their hidden node methodology through application in an endoscopic guidance system.



(a) Using a hidden node for more compact representation



(b) Using a hidden node to handle conditionally dependent nodes

Figure 5.6: Two examples of altering a network structure by introducing a hidden node.

In the remaining of this section, we present two groups of gradient-based algorithms to learning hidden nodes in Bayesian networks:

1. B+F: Backward + Forward propagation method by Kwoh and Gillies [59]. B+D: Backward + Diagonal (Left and Right) propagation method by Bang and Gillies [60], which is an extension of the B+F propagation method.
2. APN: Adaptive Probabilistic Networks method by Russell et al. [61] and Binder et al. [58]. CAPN: Conditional Adaptive Probabilistic Networks method by Ramachandran [62], which is an extension of the APN method.

Gradient-descent is a widely used function optimisation algorithm. It is commonly found in training weights of neural networks. The idea of gradient-descent (ascent) is to iteratively step towards the minimum (maximum) of a function guided by the direction provided by the gradient computed at each iteration (also called epoch). Suppose we wish to minimise a certain differentiable function $f(W)$ defined on the parameters of a Bayesian network represented by W. Using the gradient-descent method, we start with some initial $W^{(0)}$ and iteratively update W by

$$W^{(k+1)} \leftarrow W^{(k)} - \eta \nabla W^{(k)} \qquad \text{where} \qquad \nabla W^{(k)} = (\nabla f|_{W^{(k)}})$$

and η is called the learning rate that defines the step size. With sufficiently small learning rate we expect the process to converge towards the minimum point. Note that the direction is taken in the negative of the gradient since this is a minimisation problem. Gradient-descent suffers from convergence towards local optimum points. Some possible remedies like using adaptive learning rate or introducing a momentum term in the above updating equation have been suggested. There are also many variants of gradient-descent available like line search or conjugate gradient methods. Bishop [63] provides a good treatment on the topic. In many ways, training a Bayesian network using gradient-descent methods is similar to training a Neural network. This is not surprising because of the many similarities between the two types of networks. Nonetheless, there is a slight twist in this case. The network parameters of a Bayesian network are subject to two probability constraints. Firstly, all elements of a CPT must be within the range $[0,1]$. Secondly, the conditional probabilities (these are elements in each row or column of a CPT depending on the implementation) must sum to one, i.e.

$$\sum_i P(X = i \mid Parents(X) = k) = 1 \qquad \forall k \,.$$

In his analysis, Kwoh [64] shows that the two constraints can be satisfied by setting elements with negative values to zero after each iteration follow by a renormalisation process:

$$P(X = i \mid Parents(X) = k) \;\leftarrow\; \frac{P(X = i \mid Parents(X) = k)}{\sum_j P(X = j \mid Parents(X) = k)} \quad \forall i,k$$

In the event that zero valued elements have negative gradients resulting in non-progressive iterations, he suggests adding a small constant perturbation to permit continuation of the training process. Russell et al. [61] also point out that by first taking a step in the gradient direction and then renormalising to the constraint surface achieves the same effect. Binder et al. [58] take a different tack to this constraint problem. Their approach is to project $\nabla W^{(k)}$ onto the constraint surface resulting in a renormalised vector such that

$$\sum_i P(X = i \mid Parents(X) = k)$$

remains unchanged for all values of k after every update. So, if the process starts out on the constraint surface it necessarily remains on the surface. There is also a commonly adopted practice to prevent extreme probabilities in a distribution by restricting all elements of a CPT to be within the range $[\varepsilon, 1-\varepsilon]$ where ε is a small configurable value.

Backward + Forward Propagation Method

Kwoh and Gillies formulate the problem of training hidden nodes given a specific network structure as a non-linear optimisation problem where the objective is to minimise the expectation of the squared-error cost function given by

$$E\{\xi\} = E\left\{\sum_{i=1}^{|X|}[D(x_i) - Bel(x_i)]^2\right\}$$

in which the variable X has $|X|$ states, $Bel(X)$ is the posterior probability (belief) of X given some evidence from the training database, $D(X)$ is the desired value of X, and

$E\{\bullet\}$ is the expectation operator. They further propose two ways to train the hidden node structure shown in Figure 5.7 (a) and (b):

1. Backward (B) propagation – instantiate node *B* and node *C* to calculate the belief of root node *A*. The error function is defined on node *A* as $\xi = \sum_{i=1}^{|A|}[D(a_i) - Bel(a_i)]^2$.
2. Forward (F) propagation – instantiate node A to calculate the belief of node *B* and node *C*. In this case, the error function is defined on both node *B* and node *C* as $\xi = \sum_{i=1}^{|B|}[D(b_k) - Bel(b_k)]^2 + \sum_{l=1}^{|C|}[D(c_l) - Bel(c_l)]^2$.

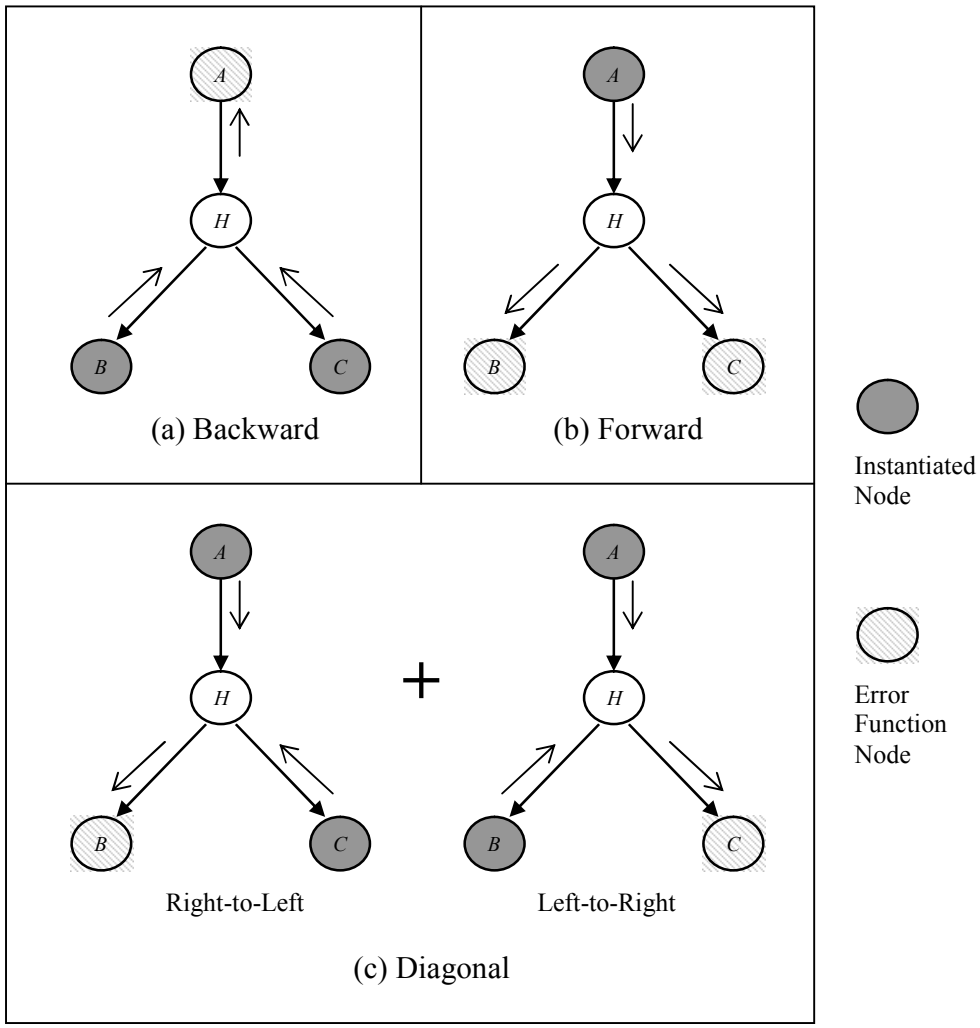


Figure 5.7: The various propagation methods.

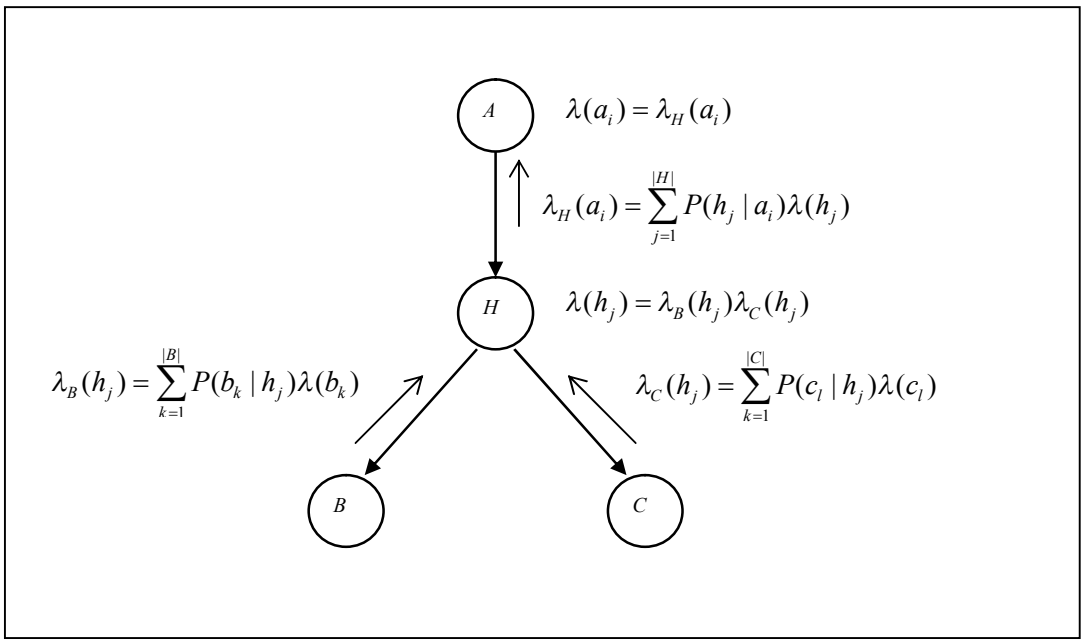


Figure 5.8: Message passing for backward propagation.

To apply the gradient-descent method, we need to determine the expected gradient for each epoch. We present their derivation of the gradient equations for the conditional probabilities of the hidden node structure for backward propagation. First, we need to introduce the operating equations for Pearl's [5] message passing algorithm for this specific case. Given a database of training data, we can specify

- $\pi(a_i)$ - prior probability distribution of root node *A* computed from the database;
- $D(a_i)$ - desired value of root node *A*, which is its instantiated value in each training data;
- $\lambda(b_k)$ - instantiated value of node *B* in each training data;
- $\lambda(c_l)$ - instantiated value of node *C* in each training data.

With reference to the message passing scheme shown in Figure 5.8, the inference process produces the posterior probability of root node *A* given instantiations of node *B* and node *C* as

$$bel(a_i) = \lambda(a_i)\pi(a_i)$$
$$Bel(a_i) = \frac{bel(a_i)}{\sum_{m=1}^{|A|} bel(a_m)}.$$

Next, we apply chain rule for partial derivatives to derive the gradient for a general entry $P(h_j \mid a_i)$ in the CPT of the link between node *A* and node *H* in the form

$$\frac{\partial \xi}{\partial P(h_j \mid a_i)} = \frac{\partial \xi}{\partial Bel(a_m)} \frac{\partial Bel(a_m)}{\partial bel(a_i)} \frac{\partial bel(a_i)}{\partial \lambda(a_i)} \frac{\partial \lambda(a_i)}{\partial P(h_j \mid a_i)}$$

in which each individual term works out as

$$\frac{\partial \xi}{\partial Bel(a_m)} = -2(D(a_m) - Bel(a_m));$$
$$\frac{\partial Bel(a_m)}{\partial bel(a_i)} = \frac{\delta_{im}}{\sum_{n=1}^{|A|} bel(a_n)} - \frac{bel(a_m)}{\left(\sum_{n=1}^{|A|} bel(a_n)\right)^2}$$
$$= \frac{1}{\sum_{n=1}^{|A|} bel(a_n)}\left(\delta_{im} - Bel(a_m)\right) \qquad \text{where} \qquad \delta_{im} = \begin{cases} 1 & i = m \\ 0 & i \neq m \end{cases};$$
$$\frac{\partial bel(a_i)}{\partial \lambda(a_i)} = \pi(a_i);$$
$$\frac{\partial \lambda(a_i)}{\partial P(h_j \mid a_i)} = \lambda(h_j).$$

Note that we have to account for the fact that the term $Bel(a_m)$ is dependent on all $bel(a_i)$ terms due to normalisation, and this is done by summing over all values indexed by *m*. Putting all these together and we obtain

$$\frac{\partial \xi}{\partial P(h_j \mid a_i)} = \frac{-2}{\sum_{n=1}^{|A|} bel(a_n)} \sum_{m=1}^{|A|} \left[\left(D(a_m) - Bel(a_m)\right)\left(\delta_{im} - Bel(a_m)\right)\right] \pi(a_i)\lambda(h_j).$$

In a similar fashion, we again apply the chain rule to derive the gradient for a general entry $P(b_k \mid h_j)$ in the CPT of the link between node *H* and node *B*:

$$\frac{\partial \xi}{\partial P(b_k \mid h_j)} = \left[\sum_{i=1}^{|A|} \frac{\partial \xi}{\partial Bel(a_m)} \frac{\partial Bel(a_m)}{\partial bel(a_i)} \frac{\partial bel(a_i)}{\partial \lambda(a_i)} \frac{\partial \lambda(a_i)}{\partial \lambda(h_j)}\right] \frac{\partial \lambda(h_j)}{\partial P(b_k \mid h_j)}.$$

The summation over states of A in this expression is necessary because $bel(a_i)$ is dependent on all $\lambda(h_j)$ terms due to evidence propagation. In this case, we only need to expand the new terms $\frac{\partial \lambda(a_i)}{\partial \lambda(h_j)}$ and $\frac{\partial \lambda(h_j)}{\partial P(b_k \mid h_j)}$. Hence, by substituting

$$\begin{aligned}\frac{\partial \lambda(a_i)}{\partial \lambda(h_j)} &= \frac{\partial \lambda(a_i)}{\partial \lambda_H(a_i)} \frac{\partial \lambda_H(a_i)}{\partial \lambda(h_j)} \\ &= P(h_j \mid a_i)\end{aligned}$$

and

$$\begin{aligned}\frac{\partial \lambda(h_j)}{\partial P(b_k \mid h_j)} &= \frac{\partial \lambda(h_j)}{\partial \lambda_B(h_j)} \frac{\partial \lambda_B(h_j)}{\partial P(b_k \mid h_j)} \\ &= \lambda_C(h_j)\lambda(b_k)\end{aligned}$$

into the expression, we arrive at the following equation

$$\frac{\partial \xi}{\partial P(b_k \mid h_j)} = \frac{-2}{\sum_{n=1}^{|A|} bel(a_n)} \sum_{i=1}^{|A|} \left[\sum_{m=1}^{|A|} \left[\left(D(a_m) - Bel(a_m) \right) \left(\delta_{im} - Bel(a_m) \right) \right] \pi(a_i) P(h_j \mid a_i) \right] \lambda_C(h_j)\lambda(b_k)$$

Furthermore, by virtue of symmetry of the network structure, we obtain

$$\frac{\partial \xi}{\partial P(c_l \mid h_j)} = \frac{-2}{\sum_{n=1}^{|A|} bel(a_n)} \sum_{i=1}^{|A|} \left[\sum_{m=1}^{|A|} \left[\left(D(a_m) - Bel(a_m) \right) \left(\delta_{im} - Bel(a_m) \right) \right] \pi(a_i) P(h_j \mid a_i) \right] \lambda_B(h_j)\lambda(c_l)$$

by swapping the symbols of B and C.

By now we have all the gradient equations required to run the gradient-descent process. However, Kwoh and Gillies point out that considering individual CPT's element in the above formulations results in rather cumbersome equations with multiple summations that make implementation a tedious task. They propose an equivalent yet elegant representation by exploiting the implicit summation of matrix multiplication. By using the following vector and matrix notations

$$\pi(X)=\begin{bmatrix}\pi(x_1)\\ \pi(x_2)\\ \vdots\\ \pi(x_{|X|})\end{bmatrix};\quad \lambda(X)=\begin{bmatrix}\lambda(x_1)\\ \lambda(x_2)\\ \vdots\\ \lambda(x_{|X|})\end{bmatrix};\quad D(X)=\begin{bmatrix}D(x_1)\\ D(x_2)\\ \vdots\\ D(x_{|X|})\end{bmatrix};$$

$$P_{Y|X}=\begin{bmatrix}P(y_1\mid x_1) & P(y_2\mid x_1) & \cdots & P(y_{|Y|}\mid x_1)\\ P(y_1\mid x_2) & P(y_2\mid x_2) & \cdots & P(y_{|Y|}\mid x_2)\\ \vdots & \vdots & \ddots & \vdots\\ P(y_1\mid x_{|X|}) & P(y_2\mid x_{|X|}) & \cdots & P(y_{|Y|}\mid x_{|X|})\end{bmatrix};$$

and the element-by-element multiplication operator $\otimes$:

$$\begin{bmatrix}u_{11} & u_{12} & \cdots & u_{1n}\\ u_{21} & u_{22} & \cdots & u_{2n}\\ \vdots & \vdots & \ddots & \vdots\\ u_{n1} & u_{n2} & \cdots & u_{nn}\end{bmatrix}\otimes\begin{bmatrix}v_{11} & v_{12} & \cdots & v_{1n}\\ v_{21} & v_{22} & \cdots & v_{2n}\\ \vdots & \vdots & \ddots & \vdots\\ v_{n1} & v_{n2} & \cdots & v_{nn}\end{bmatrix}=\begin{bmatrix}u_{11}v_{11} & u_{12}v_{12} & \cdots & u_{1n}v_{1n}\\ u_{21}v_{21} & u_{22}v_{22} & \cdots & u_{2n}v_{2n}\\ \vdots & \vdots & \ddots & \vdots\\ u_{n1}v_{n1} & u_{n2}v_{n2} & \cdots & u_{nn}v_{nn}\end{bmatrix},$$

the operating equations for the backward propagation method can now be written as:

$$\lambda_B(H)=P_{B|H}\lambda(B);\quad \lambda_C(H)=P_{C|H}\lambda(C);\quad \lambda(H)=\lambda_B(H)\otimes\lambda_C(H);$$

$$\lambda_H(A)=P_{H|A}\lambda(H);\quad \lambda(A)=\lambda_H(A);$$

$$bel(A)=\lambda(A)\otimes\pi(A);\quad Bel(A)=\frac{bel(A)}{|bel(A)|};$$

$$\xi=[D(A)-Bel(A)]^T[D(A)-Bel(A)];$$

$$\frac{\partial\xi}{\partial P_{H|A}}=\frac{-2}{|bel(A)|}\left\{\left[(D(A)-Bel(A))^T(I(A)-Bel(A)ones(A)^T)\right]^T\otimes\pi(A)\right\}\lambda(H)^T;$$

$$\frac{\partial\xi}{\partial P_{B|H}}=\frac{-2}{|bel(A)|}\left\{\left[P_{H|A}{}^T\left\{\left[(D(A)-Bel(A))^T(I(A)-Bel(A)ones(A)^T)\right]^T\otimes\pi(A)\right\}\right]\otimes\lambda_C(H)\right\}\lambda(C)^T;$$

$$\frac{\partial\xi}{\partial P_{C|H}}=\frac{-2}{|bel(A)|}\left\{\left[P_{H|A}{}^T\left\{\left[(D(A)-Bel(A))^T(I(A)-Bel(A)ones(A)^T)\right]^T\otimes\pi(A)\right\}\right]\otimes\lambda_B(H)\right\}\lambda(B)^T.$$

With the vector and matrix notations, a single gradient-descent step represented by

$$W^{(k+1)}\leftarrow W^{(k)}-\eta\nabla W^{(k)}$$

now becomes updating of three CPTs as follows:

$$P_{H|A}\leftarrow P_{H|A}-\eta E\left\{\frac{\partial\xi}{\partial P_{H|A}}\right\};\quad P_{B|H}\leftarrow P_{B|H}-\eta E\left\{\frac{\partial\xi}{\partial P_{B|H}}\right\};\quad P_{C|H}\leftarrow P_{C|H}-\eta E\left\{\frac{\partial\xi}{\partial P_{C|H}}\right\}.$$

In the above expressions, the expected gradient for each epoch is given by

$$E\left\{\frac{\partial \xi}{\partial P}\right\} = \frac{1}{N}\sum_{1}^{N}\frac{\partial \xi}{\partial P} \qquad \text{for} \qquad P = \{P_{H|A}, P_{B|H}, P_{C|H}\}$$

where N is the number of data points in the database and the summation is taken over $\frac{\partial \xi}{\partial P}$ evaluated at each data point.

The forward propagation gradients are derived in exactly the same manner. Using the vector and matrix notations the error function defined on node B and node C is given in the form

$$\xi = [D(B) - Bel(B)]^T [D(B) - Bel(B)] + [D(C) - Bel(C)]^T [D(C) - Bel(C)].$$

Figure 5.9 shows the propagation messages being passed when node A is instantiated. The required posterior probabilities are calculated as

$$Bel(B) = \pi(B);$$
$$Bel(C) = \pi(C).$$

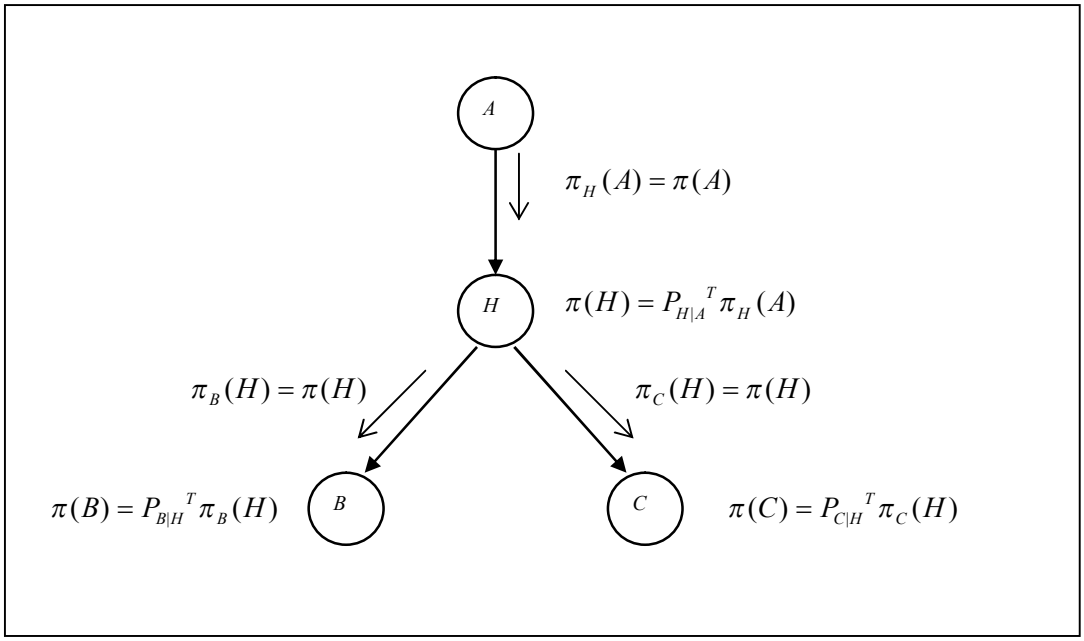


Figure 5.9: Message passing for forward propagation.

Given the set of nodes $\{A, B, C, H\}$ as shown in Figure 5.8
and a training database with *N* data points.

Initialisation:
{
- Set learning rate η and probability value limit ε
- Set hidden node size
- Initialise and normalise the CPTs - $\{\pi(A), P_{H|A}, P_{B|H}, P_{C|H}\}$
- Select combination of propagations for generating gradients
- Set error limit and epoch limit

}

Training Process:
{
While (mean squared error > error limit
| number of iterations < epoch limit)
{
For each propagation type selected do
{
For i = 1 to N, given i^{th} data point.

Compute $\left\{\frac{\partial \xi}{\partial P_{H|A}}, \frac{\partial \xi}{\partial P_{B|H}}, \frac{\partial \xi}{\partial P_{C|H}}\right\}$

Compute $E\left\{\frac{\partial \xi}{\partial P}\right\} = \frac{1}{N}\sum_{1}^{N}\frac{\partial \xi}{\partial P}$ for $P = \{P_{H|A}, P_{B|H}, P_{C|H}\}$.

Update $P \leftarrow P - \eta E\left\{\frac{\partial \xi}{\partial P}\right\}$ for $P = \{P_{H|A}, P_{B|H}, P_{C|H}\}$.

Normalise $\{P_{H|A}, P_{B|H}, P_{C|H}\}$.

}
}
}

Figure 5.10: Summary of hidden node training algorithm.

Note that in this case $\pi(A)$ is the instantiated value of node A and not the priors as in backward propagation. Furthermore, if we ensure that all $\pi(A)$ are normalised from the start, there is no need to normalise $Bel(B)$ and $Bel(C)$. This results in simpler gradient equations shown below

$$\frac{\partial \xi}{\partial P_{H|A}} = -2\pi(A)\left[(D(B) - Bel(B))^T P_{B|H}{}^T + (D(C) - Bel(C))^T P_{C|H}{}^T\right];$$

$$\frac{\partial \xi}{\partial P_{B|H}} = -2\left[P_{H|A}{}^T \pi(A)\right](D(B) - Bel(B))^T;$$

$$\frac{\partial \xi}{\partial P_{C|H}} = -2\left[P_{H|A}{}^T \pi(A)\right](D(C) - Bel(C))^T.$$

Figure 5.10 shows a summarised interpretation of the full training algorithm due to Kwoh and Gillies. Depending on the purpose of the Bayesian network, the training may be carried out using backward propagation only, forward propagation only, or combined backward/forward propagation. It all depends on which are the query nodes.

Backward + Diagonal Propagation Method

Kwoh and Gillies [59] show that using combined backward/forward propagation produces very good results in their application of endoscopic guidance system. Bang and Gillies [60] further extend this methodology to include a third type of propagation – diagonal propagation, see Figure 5.7(c). The idea is to replace the forward propagation with the diagonal propagation, which is actually made up of two propagations:

- Diagonal Left (L) propagation – instantiate node *A* and node *C* to calculate belief of node *B*. The error function is defined on node *B* as

  $$\xi = \sum_{k=1}^{|B|} [D(b_k) - Bel(b_k)]^2 \text{ ; and}$$

- Diagonal Right (R) propagation – instantiate node *A* and node *B* to calculate belief of node *C*. The error function is defined on node *C* as

  $$\xi = \sum_{l=1}^{|C|} [D(c_l) - Bel(c_l)]^2 .$$

In his analysis using synthetic data, Bang [65] shows that the combined backward/diagonal method performs better than the combined backward/forward method. Bang did not make use of the vector/matrix notations suggested by Kwoh and Gillies. We have worked out the respective gradient equations using the vector/matrix notations as follows:

$$\xi = [D(B) - Bel(B)]^T [D(B) - Bel(B)];$$

$$\frac{\partial \xi}{\partial P_{B|H}} = \frac{-2}{|bel(B)|}\varepsilon(H)\Big[(D(B) - Bel(B))^T (I(B) - Bel(B)ones(B)^T)\Big];$$

$$\frac{\partial \xi}{\partial P_{H|A}} = \frac{-2}{|bel(B)|}\pi(A)\Big[\Big\{P_{B|H}\Big[(D(B) - Bel(B))^T (I(B) - Bel(B)ones(B)^T)\Big]^T\Big\} \otimes \lambda(H)\Big]^T;$$

$$\frac{\partial \xi}{\partial P_{C|H}} = \frac{-2}{|bel(B)|}\Big[\Big\{P_{B|H}\Big[(D(B) - Bel(B))^T (I(B) - Bel(B)ones(B)^T)\Big]^T\Big\} \otimes \pi(H)\Big]\lambda(C)^T.$$

APN and CAPN Methods

We now move on to the learning method proposed by Russell et al. [61] called Adaptive Probabilistic Networks (APN). The APN method aims to learn the parameters of a Bayesian network that maximises the objective function $\xi = \ln P_w(D)$, the log of the probability assigned by the network to the observed data when the CPT parameters are set to *w*. With this, the APN method learns the maximum likelihood model. If there is missing data we can resort to the widely Expectation Maximization algorithm [66]. We present next the gradient derivation for APN method following [38]. Let w_{ijk} denote a specific CPT entry, the probability that variable X_i takes on its j^{th} possible value assignment given that its parents U_i take on their k^{th} possible value assignment:

$$w_{ijk} \equiv P(X_i = x_{ij} \mid U_i = u_{ik}).$$

Note that if we write w_{ijk} using previous notations, we will have

$$w_{ijk} \equiv P(X_i = j \mid Parents(X_i) = k).$$

Suppose the training database D has data points $D_1, \cdots, D_N$, which are generated independently from some underlying distribution. The contribution of each data point to the gradient can be computed separately and summed together to produce the required gradient:

$$\frac{\partial \xi}{\partial w_{ijk}} = \frac{\partial \ln P_w(D)}{\partial w_{ijk}}$$
$$= \frac{\partial}{\partial w_{ijk}}\left(\ln \prod_{l=1}^{N} P_w(D_l) \right)$$
$$= \sum_{l=1}^{N} \frac{\partial \ln P_w(D_l)}{\partial w_{ijk}}$$
$$= \sum_{l=1}^{N} \frac{1}{P_w(D_l)} \frac{\partial}{\partial w_{ijk}} \left(P_w(D_l) \right).$$

By averaging over the possible values of X_i and U_i we have

$$P_w(D_l) = \sum_{j',k'} P_w(D_l \mid x_{ij'}, u_{ik'}) P_w(x_{ij'}, u_{ik'})$$
$$= \sum_{j',k'} P_w(D_l \mid x_{ij'}, u_{ik'}) P_w(x_{ij'} \mid u_{ik'}) P_w(u_{ik'})$$
$$= \sum_{j',k'} P_w(D_l \mid x_{ij'}, u_{ik'}) w_{ij'k'} P_w(u_{ik'})$$

which leads to

$$\frac{\partial}{\partial w_{ijk}} \left(P_w(D_l) \right) = \frac{\partial}{\partial w_{ijk}} \left(\sum_{j',k'} P_w(D_l \mid x_{ij'}, u_{ik'}) w_{ij'k'} P_w(u_{ik'}) \right)$$
$$= P_w(D_l \mid x_{ij}, u_{ik}) P_w(u_{ik})$$
$$= \frac{P_w(x_{ij}, u_{ik} \mid D_l) P_w(D_l)}{P_w(x_{ij}, u_{ik})} P_w(u_{ik})$$
$$= P_w(x_{ij}, u_{ik} \mid D_l) P_w(D_l) \frac{1}{P_w(x_{ij} \mid u_{ik})}$$
$$= \frac{P_w(x_{ij}, u_{ik} \mid D_l) P_w(D_l)}{w_{ijk}}$$

Note the application of the Bayes Rule to the term $P_w(D_l \mid x_{ij}, u_{ik})$ in the antepenultimate equality. Finally, piecing all together we obtain the gradient equation in the form

$$\frac{\partial \xi}{\partial w_{ijk}} = \sum_{l=1}^{N} \frac{1}{P_w(D_l)} \frac{P_w(x_{ij}, u_{ik} \mid D_l) P_w(D_l)}{w_{ijk}}$$
$$= \sum_{l=1}^{N} \frac{P_w(x_{ij}, u_{ik} \mid D_l)}{w_{ijk}}.$$

The above equation specifies the gradient-ascent direction and it requires computing the term $P_w(x_{ij}, u_{ik} \mid D_l)$, the posterior probabilities given the instantiation due to each of $D_1, \cdots, D_N$. Russell et al. suggest the use of standard Bayesian network inference

algorithms for this task. But this can be computationally intensive for generic network. Nonetheless, for the purpose of training the hidden node network we can rely on Pearl's message passing algorithm.

The APN method can be implemented in a way similar to Kowh and Gillies's method partitioning w_{ijk} into $\{P_{H|A}, P_{B|H}, P_{C|H}\}$ like in the case of backward propagation. Using the APN gradient equation, we obtain

$$\frac{\partial \xi}{\partial P_{H|A}} = \left(\sum_{l=1}^{N} Bel(H, A) \right) \div P_{H|A}; \quad \frac{\partial \xi}{\partial P_{B|H}} = \left(\sum_{l=1}^{N} Bel(B, H) \right) \div P_{B|H}; \quad \frac{\partial \xi}{\partial P_{C|H}} = \left(\sum_{l=1}^{N} Bel(C, H) \right) \div P_{C|H};$$

where we have introduced a new element-by-element division operator $\div$:

$$\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ u_{21} & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n1} & u_{n2} & \cdots & u_{nn} \end{bmatrix} \div \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{n1} & v_{n2} & \cdots & v_{nn} \end{bmatrix} = \begin{bmatrix} u_{11}/v_{11} & u_{12}/v_{12} & \cdots & u_{1n}/v_{1n} \\ u_{21}/v_{21} & u_{22}/v_{22} & \cdots & u_{2n}/v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n1}/v_{n1} & u_{n2}/v_{n2} & \cdots & u_{nn}/v_{nn} \end{bmatrix},$$

and

$$Bel(Y, X) = \begin{bmatrix} P(y_1, x_1 \mid D_l) & P(y_2, x_1 \mid D_l) & \cdots & P(y_{|Y|}, x_1 \mid D_l) \\ P(y_1, x_2 \mid D_l) & P(y_2, x_2 \mid D_l) & \cdots & P(y_{|Y|}, x_2 \mid D_l) \\ \vdots & \vdots & \ddots & \vdots \\ P(y_1, x_{|X|} \mid D_l) & P(y_2, x_{|X|} \mid D_l) & \cdots & P(y_{|Y|}, x_{|X|} \mid D_l) \end{bmatrix}.$$

It should also be noted that in this case we are carrying out gradient-ascent, therefore we have

$$P_{H|A} \leftarrow P_{H|A} + \eta \frac{\partial \xi}{\partial P_{H|A}}; \quad P_{B|H} \leftarrow P_{B|H} + \eta \frac{\partial \xi}{\partial P_{B|H}}; \quad P_{C|H} \leftarrow P_{C|H} + \eta \frac{\partial \xi}{\partial P_{C|H}};$$

and there is no expectation operator $E\{\bullet\}$.

Ramachandran [62] points out that while the APN method learns the maximum likelihood model, when it comes to learning a network to be used as a classifier, it may be desirable to optimise the classification accuracy of the network instead. To elaborate further, this means that the learning algorithm should learn a network that best estimates the probability distribution of the target (class) variables conditioned on the evidence provided by the other variables. In view of this consideration, he proposes the

Conditional Adaptive Probabilistic Networks (CAPN) method where the objective is to maximise the function

$$\xi = \ln P_w(D_C \mid D_E)$$

in which D_C and D_E are partitions of the database pertaining to the set of class variables and the set of variables that provide the evidence respectively. For example, given the hidden node network shown in Figure 5.6b, we could have $D_C = \{A\}$ and $D_E = \{B, C\}$. By expanding the objective function we obtain

$$\xi = \ln P_w(D_C, D_E) - \ln P_w(D_E).$$

This allows us to write the gradient equation of the objective function as the difference of two separate gradient functions in the form

$$\frac{\partial \xi}{\partial w_{ijk}} = \frac{\partial \ln P_w(D_C, D_E)}{\partial w_{ijk}} - \frac{\partial \ln P_w(D_E)}{\partial w_{ijk}}.$$

In this way, the CAPN method can compute the required gradient by simply applying the APN method twice, first to $\ln P_w(D_C, D_E)$ and then to $\ln P_w(D_E)$. Clearly, the CAPN method can fully utilise the mechanism of APN and no additional operating equation is required.

There are some practical issues that have to be addressed when considering training hidden nodes in a Bayesian network. The first issue is that of identifying where to introduce the hidden node. Recall that there are three possible scenarios to hidden node creation:

1. Identified as a hidden cause by domain expert;
2. Introduced to gain compactness in network structure;
3. Introduced to handle violation of conditional independency assumption.

The first scenario solely relies on domain expert to identify the presence of a hidden node while the second scenario involves examining the network structure to determine if a hidden node should be added. In these two cases, there is no clear prescriptive approach to each process. For the third case, Kwoh and Gillies propose using the conditional mutual information measure to decide if a hidden node should be added. For example, the

mutual information measure could be used as the dependency measure between two variables B and C:

$$Dep(B,C) = \sum_{k=1}^{|B|}\sum_{l=1}^{|C|} P(b_k, c_l) \log \frac{P(b_k, c_l)}{P(b_k)P(c_l)}.$$

Here, the conditional dependency measure between variables B and C given A using the conditional mutual information measure is defined as

$$Dep(B,C \mid A) = \sum_{i=1}^{|A|}\sum_{k=1}^{|B|}\sum_{l=1}^{|C|} P(a_i, b_k, c_l) \log \frac{P(b_k, c_l \mid a_i)}{P(b_k \mid a_i)P(c_l \mid a_i)}$$

which is non-negative and zero only when B and C are conditionally independent over A. Once the hidden node is created, the next issue is to determine its size (the number of states). If a domain expert creates the hidden node and it is associated with some known physical properties that allow its size to be determined, then we are done. Otherwise, like the other two scenarios, we need to estimate the size of the hidden node. Kwoh [64] presents a good analysis on the lower and upper bounds for estimating the size of the hidden node. With reference to the hidden node network displayed in Figure 5.7, and assuming that A is the query node and $|A| \leq |B| \leq |C|$, the size of the hidden node should be between $|A|$ and $|A| \times |C|$. However, a precise way of estimating the size of the hidden node was not provided.

5.4 Our proposal

Motivated by the 2-Pass control of the join tree algorithm, we propose updating messages in a loopy belief propagation process in the schedule defined by the 2-Pass control of a corresponding join tree formed from the original Bayesian network. The idea is to update the nodes in the order of the join tree clique order. We would like to investigate the feasibility of applying the 2-Pass order to loopy belief propagation as a control rule. In this way, we can address the problem of convergence issues of loopy belief propagation.

Next, we will conduct error analysis of loopy belief propagation applied to Bayesian networks. Most of the error analysis work on loopy belief propagation has been

carried out on undirected graphs. It seems that it is easier to analyse an undirected graph because there is no need to handle the upward and downward pass, however, we believe there could be advantages in a directed graph setup since we could identify the sink and source nodes of the error, where the sink and source nodes are defined by the upward/downward pass. The findings from the error analysis can also provide some insights to the behaviour of loopy belief propagation on Bayesian networks. Furthermore, if we can derive the error in terms of network parameters, then we can attempt to perform loop correction and combine it with the 2-Pass control mentioned above to improve its approximation accuracy.

In chapter 6 we will present our error analysis on the loopy error in Bayesian networks, and present two algorithms that make use of the error terms to perform exact inference using belief propagation on networks with single generic loop(s). In chapter 7 we will introduce a 2-Pass loopy belief propagation algorithm with empirical studies and discussions.

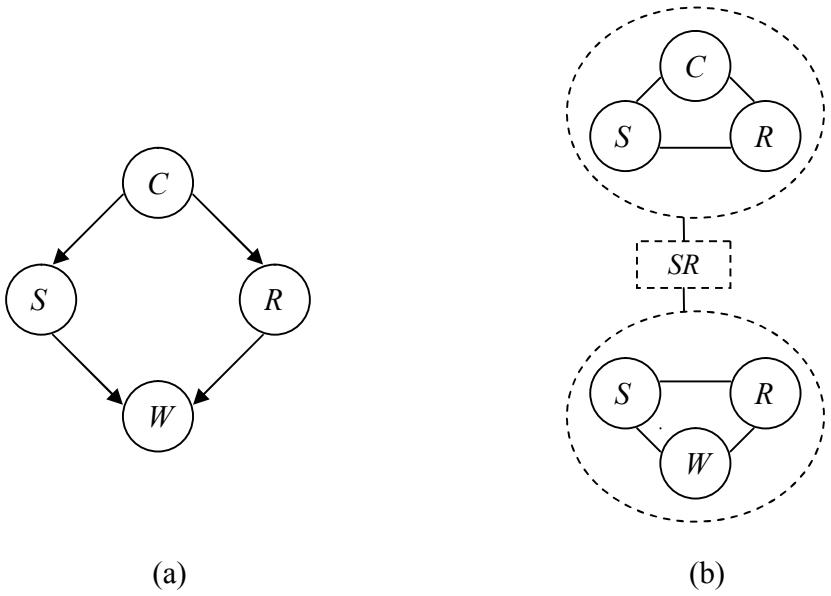
Chapter 6

Error Analysis of Loopy Belief Propagation in Bayesian Networks

In this chapter we will present our findings on errors due to loopy belief propagation applied to Bayesian networks with loop.

6.1 Join tree algorithm versus loopy belief propagation on a simple loop

Let us begin the error analysis by first examining the differences between the join tree algorithm and loopy belief propagation when they are applied to the simple loop shown in Figure 6.1(a).



(a) (b)

Figure 6.1: (a) A simple loop. (b) Join tree of simple loop.

The joint distribution of the network is given by

$$P(c,s,r,w) = P(w \mid s,r)P(s \mid c)P(r \mid c)P(c).$$

We will first walk through the loopy belief propagation algorithm starting from the leaf node W just as in Pearl's standard message passing algorithm. Since node W is the leaf node and it has no λ evidence, we set $\lambda(W)$ to a uniform vector, e.g., [1, 1] for a binary node. Note that in this case, there is a slight problem with the initial message passing starting from node W as it requires the π message ($\pi_W(S)$) from node S to generate the λ message ($\lambda_W(R)$) to node R and vice versa. The standard practice is to initialise both π messages ($\pi_W(S), \pi_W(R)$) to a uniform vector. Now we can step through the generated messages as follows:

1. Node W sends the messages $\lambda_W(R)$ and $\lambda_W(S)$ to node R and node S respectively, which turn out to be uniform vectors too. This is to be expected as node W has no evidence to encode in its messages to its parent nodes S and R. This effect can also be understood using the d-separation property on the converging network structure ($S \rightarrow W \leftarrow R$) of S, R and W, which states that S and R are marginally independent.

2. Node S and node R send the messages $\lambda_S(C)$ and $\lambda_R(C)$ to node C respectively and again these are uniform vectors as there is no evidence to encode.
3. Node C sends the messages $\pi_S(C) = \pi(C) \otimes \lambda_R(C) = \pi(C)$ and $\pi_R(C) = \pi(C) \otimes \lambda_S(C) = \pi(C)$ to node S and node R respectively. Note that effectively $\pi(C)$ is the only piece of evidence required to update the posteriors of all nodes in this network and furthermore, we have $bel(C) = \lambda(C) \otimes \pi(C) = \lambda_S(C) \otimes \lambda_R(C) \otimes \pi(C) = \pi(C) = P(C)$.
4. Node S first computes $\pi(S) = P_{S|C}{}^T \pi_S(C)$ and then sends it as the message $\pi_W(S) = \pi(S)$ to node W. Similarly, node R first computes $\pi(R) = P_{R|C}{}^T \pi_R(C)$ and then sends it as the message $\pi_W(R) = \pi(R)$ to node W. By this stage of the message passing, we have effectively obtained the correct marginal probabilities of both node S and node R:

$$bel(S) = \lambda(S) \otimes \pi(S) = \lambda_W(S) \otimes (P_{S|C}{}^T \pi_S(C)) = P_{S|C}{}^T \pi(C) = P(S)$$

$$bel(R) = \lambda(R) \otimes \pi(R) = \lambda_W(R) \otimes (P_{R|C}{}^T \pi_R(C)) = P_{R|C}{}^T \pi(C) = P(R)$$

5. Node W computes $bel(W) = \lambda(W) \otimes \pi(W)$ and the algorithm terminates since the messages $\lambda_W(R)$ and $\lambda_W(S)$ remains unchanged (uniform vectors) from Step 1.

The computation of the marginal probability of node W at Step 5 is what goes wrong with the loopy belief propagation for this network. If we look at the joint distribution of the network, the correct way to compute $P(w)$ is

$$\begin{aligned} P(w) &= \sum_{s,r,c} P(w \mid s,r) P(s \mid c) P(r \mid c) P(c) \\ &= \sum_{s,r} P(w \mid s,r) \sum_{c} P(s \mid c) P(r \mid c) P(c) \\ &= \sum_{s,r} P(w \mid s,r) P(s,r) \end{aligned}$$

However, the loopy belief propagation algorithm computes the term $\pi(w_i)$ at node W as

$$\pi(w_i) = \sum_{s_j, r_k} P(w_i \mid s_j, r_k) \left[\pi_w(s_j) \pi_w(r_k) \right]$$

where $\pi_W(s_j)$ is in fact equal to $P(s_j)$ since $\lambda_W(S)$ is a uniform vector and similarly $\pi_W(r_k) = P(r_k)$, both of which are obtained from Step 4. This is equivalent to computing

$$P(w) = \sum_{s,r} P(w \mid s,r)P(s)P(r).$$

The original belief propagation algorithm was designed to produce exact marginal probabilities for a singly connected network on which the marginal independency assumption between node *S* and node *R* with converging arcs ($S \rightarrow W \leftarrow R$) on node *W* holds true. However, when the standard belief propagation is applied to a network with loop, it produces the above error since node *S* and node *R* are now dependent through the root node *C* due to the network structure ($S \leftarrow C \rightarrow R$), that is $P(s,r) \neq P(s)P(r)$.

To summarise, when we apply loopy belief propagation to the single loop network shown in Figure 6.1(a), the only error lies with the computation of the marginal probability of node *W* due to the assumption that $P(s,r) = P(s)P(r)$. The marginal probabilities of nodes *C*, *S*, and *R* are all correct. Note that throughout the belief propagation process, the computing of the messages only involves terms of $P(S)$ and $P(R)$, and not $P(S,R)$. We will show in later sections that we can indeed reconstruct the true $P(S,R)$ from individual $P(S)$ and $P(R)$ values by analysing the dependency between them due to node *C*, thereby allowing the loopy belief propagation to perform an additional step of loop correction in order to compute the exact marginal probability of node *W*. But before that, to help us gain a better understanding of the loop error problem, let us further investigate how the join tree algorithm avoids this error for this specific example.

With reference to the join tree shown in Figure 6.1(b), we have $Clq_1 = \{C,S,R\}$, $Clq_2 = \{S,R,W\}$, and the separator set $Sep = \{S,R\}$. We shall assign the potential functions using the Hugin architecture as follows:

$$\phi_1(Clq_1) = P(s \mid c)P(r \mid c)P(c) \equiv f,$$

$$\phi_2(Clq_2) = P(w \mid s,r) \equiv g, \text{ and}$$

$$\phi_{Sep}(Sep) = 1,$$

such that

$$P(V) = \frac{\phi_1(Clq_1) * \phi_2(Clq_2)}{\phi_{Sep}(Sep)}$$
$$= \frac{f * g}{1}$$
$$= P(s \mid c)P(r \mid c)P(c)P(w \mid s, r).$$

In this case, the full inward pass is just a single inward pass from Clq_1 to Clq_2, where we have set Clq_2 as the root node. The operations of the inward pass are:

- At Clq_1: compute $\phi_1(Clq_1)^{\downarrow Sep} = \sum_c f$ and pass it as message to separator $Sep = \{S, R\}$. Update $\phi_1^*(Clq_1) = \phi_1(Clq_1) = f$.
- At $Sep = \{S, R\}$: compute $\frac{\phi_1(Clq_1)^{\downarrow Sep}}{\phi_{Sep}(Sep)} = \frac{\sum_c f}{1}$ and pass it as message to clique Clq_2, then update $\phi_{Sep}^*(Sep) = \phi_1(Clq_1)^{\downarrow Sep} = \sum_c f$
- At clique Clq_2: update $\phi_2^*(Clq_2) = \frac{\sum_c f}{1}\phi_2(Clq_2) = (\sum_c f)g$.

Similarly, the full outward pass is just a single outward pass from Clq_2 to Clq_1, and the operations are:

- At Clq_2: compute $\phi_2^*(Clq_2)^{\downarrow Sep} = \sum_w (\sum_c f)g = (\sum_w g)(\sum_c f)$ and pass it as message to separator $Sep = \{S, R\}$. Update $\phi_2^{**}(Clq_2) = \phi_2^*(Clq_2) = (\sum_c f)g$.
- At $Sep = \{S, R\}$: compute $\frac{\phi_2^*(Clq_2)^{\downarrow Sep}}{\phi_{Sep}^*(Sep)} = \frac{(\sum_w g)(\sum_c f)}{\sum_c f} = \sum_w g$ and pass it as message to clique Clq_1, then update $\phi_{Sep}^{**}(Sep) = \phi_2^*(Clq_2)^{\downarrow Sep} = (\sum_w g)(\sum_c f)$.
- At clique Clq_1: update $\phi_1^{**}(Clq_1) = (\sum_w g)\phi_1^*(Clq_1) = (\sum_w g)(f)$.

To verify the correctness of the results produced by the join tree algorithm, first note that

$$\sum_w g = \sum_w P(w \mid s, r) = 1,$$

$$
\begin{aligned}
P(s,r) &= \sum_{c,w} P(w \mid s,r)P(s \mid c)P(r \mid c)P(c) \\
&= \sum_{c} P(s \mid c)P(r \mid c)P(c)\sum_{w} P(w \mid s,r) \\
&= \sum_{c} P(s \mid c)P(r \mid c)P(c),
\end{aligned}
$$

and

$$
\begin{aligned}
P(c,s,r) &= \sum_{w} P(w \mid s,r)P(s \mid c)P(r \mid c)P(c) \\
&= P(s \mid c)P(r \mid c)P(c)\sum_{w} P(w \mid s,r) \\
&= P(s \mid c)P(r \mid c)P(c).
\end{aligned}
$$

By the end of the 2-Pass propagation, we have

$$
\begin{aligned}
\phi_1^{**}(Clq_1) &= (\sum_{w} g)(\sum_{c} f) \\
&= \sum_{c} f \\
&= \sum_{c} P(s \mid c)P(r \mid c)P(c) \\
&= P(s,r)
\end{aligned}
$$

and

$$
\begin{aligned}
\phi_2^{**}(Clq_2) &= (\sum_{w} g)(f) \\
&= f \\
&= P(s \mid c)P(r \mid c)P(c) \\
&= P(c,s,r).
\end{aligned}
$$

In addition, we also have

$$
\begin{aligned}
\phi_{Sep}^{**}(Sep) &= (\sum_{w} g)(\sum_{c} f) \\
&= P(s,r).
\end{aligned}
$$

This completes the verification that the resulting potential functions are indeed the true joint distributions for each clique.

The join tree algorithm circumvents the need to explicitly compute $P(S,R)$ from $P(S)$ and $P(R)$, at the expense of higher computational costs and additional storage by maintaining the joint probability(belief) of cluster nodes $\{C,S,R\}$ and $\{C,S,W\}$, and then marginalizing to obtain the beliefs of $\{S,R\}$ for the separator set. In general, the join tree

algorithm always computes the joint belief of nodes that are dependent as they are required to correctly update potential functions in cliques that they are a subset of.

Next, we will further investigate the difference between loopy belief propagation and join tree algorithm using the free energy fixed point approach. The Bethe free energy for the network shown in Figure 6.1(a) is given by

$$F_B = U_B - H_B$$

where

$$U_B = -\left\{ \begin{array}{l} \sum_c bel(c) \ln P(c) + \sum_{c,s} bel(c,s) \ln P(s \mid c) + \sum_{c,r} bel(c,r) \ln P(r \mid c) \\ + \sum_{s,r,w} bel(s,r,w) \ln P(w \mid s,r) \end{array} \right\}$$

and

$$H_B = -\left\{ \begin{array}{l} \sum_c bel(c) \ln bel(c) + \sum_{c,s} bel(c,s) \ln bel(c,s) + \sum_{c,r} bel(c,r) \ln bel(c,r) \\ + \sum_{s,r,w} bel(s,r,w) \ln bel(s,r,w) \end{array} \right\}$$
$$+ \left\{ 2\sum_c bel(c) \ln bel(c) + \sum_s bel(s) \ln bel(s) + \sum_r bel(r) \ln bel(r) \right\}$$

Suppose at a certain stage of some iterative process that minimises the Bethe free energy or by applying the loopy belief propagation, we have obtained the following beliefs:

$$bel(C) = P(C)$$
$$bel(C,S) = P(C,S)$$
$$bel(C,R) = P(C,R)$$

Our aim is to find out how the loopy belief propagation's fixed point for $bel(S,R)$ is related to the beliefs mentioned above. First, note that computing $bel(S,R,W)$ is essentially equivalent to computing $bel(S,R)$ since $bel(s,r,w) = P(w \mid s,r) bel(s,r)$ where $P(w \mid s,r)$. For simplicity, we shall consider the case where the nodes are binary. Let us denote

$$bel(S) = \sum_c bel(c,S) = \begin{bmatrix} p_s \\ 1 - p_s \end{bmatrix} \quad \text{and} \quad bel(R) = \sum_c bel(c,R) = \begin{bmatrix} p_r \\ 1 - p_r \end{bmatrix},$$

then we can fully represent $bel(S,R)$ using a single parameter

$$p_{sr} = bel(s_1, r_1)$$

by writing it as

$bel(S,R)$	r_1	r_2
s_1	p_{sr}	$p_s - p_{sr}$
s_2	$p_r - p_{sr}$	$1-(p_s + p_r - p_{sr})$

.

Now we are ready to take the partial derivatives on the Bethe free energy with respect to p_{sr}. We have

$$\frac{\partial U_B}{\partial p_{sr}} = -\frac{\partial}{\partial p_{sr}}\left\{\begin{array}{l}\sum_c bel(c)\ln P(c) + \sum_{c,s} bel(c,s)\ln P(s\mid c) + \sum_{c,r} bel(c,r)\ln P(r\mid c) \\ + \sum_{s,r,w} bel(s,r,w)\ln P(w\mid s,r)\end{array}\right\}$$

$$= -\frac{\partial}{\partial p_{sr}}\left\{\sum_{s,r,w} bel(s,r,w)\ln P(w\mid s,r)\right\}$$

and it is easy to show that

$$\frac{\partial H_B}{\partial p_{sr}} = -\frac{\partial}{\partial p_{sr}}\left\{\sum_{s,r,w} bel(s,r,w)\ln bel(s,r,w))\right\}.$$

Putting the two terms together, we have

$$\frac{\partial F_B}{\partial p_{sr}} = \frac{\partial U_B}{\partial p_{sr}} - \frac{\partial H_B}{\partial p_{sr}}$$

$$= -\frac{\partial}{\partial p_{sr}}\left\{\sum_{s,r,w} bel(s,r,w)\ln P(w\mid s,r)\right\} + \frac{\partial}{\partial p_{sr}}\left\{\sum_{s,r,w} bel(s,r,w)\ln bel(s,r,w))\right\}$$

$$= \frac{\partial}{\partial p_{sr}}\left\{\sum_{s,r,w} bel(s,r,w)\ln bel(s,r,w)) - bel(s,r,w)\ln P(w\mid s,r)\right\}$$

$$= \frac{\partial}{\partial p_{sr}}\left\{\sum_{s,r,w} bel(s,r,w)\ln \frac{bel(s,r,w)}{P(w\mid s,r)}\right\}$$

$$= \frac{\partial}{\partial p_{sr}}\left\{\sum_{s,r,w} bel(s,r,w)\ln bel(s,r)\right\}$$

$$= \frac{\partial}{\partial p_{sr}}\left\{\sum_{s,r} bel(s,r)\ln bel(s,r)\right\}$$

Recall that

$bel(S,R)$	r_1	r_2
s_1	p_{sr}	$p_s - p_{sr}$
s_2	$p_r - p_{sr}$	$1-(p_s + p_r - p_{sr})$

,

so we have

$$\frac{\partial}{\partial p_{sr}}\{bel(s_1,r_1)\ln bel(s_1,r_1)\} = \frac{\partial}{\partial p_{sr}}\{p_{sr}\ln p_{sr}\},$$
$$= \ln p_{sr} + 1$$

$$\frac{\partial}{\partial p_{sr}}\{bel(s_1,r_2)\ln bel(s_1,r_2)\} = \frac{\partial}{\partial p_{sr}}\{(p_s - p_{sr})\ln(p_s - p_{sr})\}$$
$$= -\ln(p_s - p_{sr}) + (p_s - p_{sr})\frac{1}{(p_s - p_{sr})}(-1),$$
$$= -\ln(p_s - p_{sr}) - 1$$

$$\frac{\partial}{\partial p_{sr}}\{bel(s_2,r_1)\ln bel(s_2,r_1)\} = \frac{\partial}{\partial p_{sr}}\{(p_r - p_{sr})\ln(p_r - p_{sr})\}$$
$$= -\ln(p_r - p_{sr}) + (p_r - p_{sr})\frac{1}{(p_r - p_{sr})}(-1),$$
$$= -\ln(p_r - p_{sr}) - 1$$

and

$$\frac{\partial}{\partial p_{sr}}\{bel(s_2,r_2)\ln bel(s_2,r_2)\} = \frac{\partial}{\partial p_{sr}}\{(1-(p_s + p_r - p_{sr}))\ln(1-(p_s + p_r - p_{sr}))\}.$$
$$= \ln(1-(p_s + p_r - p_{sr})) + 1$$

Hence, $\frac{\partial F_B}{\partial p_{sr}} = 0$ implies

$$\frac{\partial}{\partial p_{sr}}\left\{\sum_{s,r} bel(s,r)\ln bel(s,r)\right\} = 0$$
$$(\ln p_{sr} + 1) + (-\ln(p_s - p_{sr}) - 1) + (-\ln(p_r - p_{sr}) - 1) + (\ln(1-(p_s + p_r - p_{sr})) + 1) = 0$$
$$\ln\left\{\frac{p_{sr}(1-(p_s + p_r - p_{sr}))}{(p_s - p_{sr})(p_r - p_{sr})}\right\} = 0$$
$$p_{sr}(1-(p_s + p_r - p_{sr})) = (p_s - p_{sr})(p_r - p_{sr})$$
$$p_{sr} = p_s p_r$$

The significance of the above derivations is that we have successfully shown that $bel(s_1,r_1) = p_{sr} = P(s_1)P(r_1)$, which is the same error produced by the loopy belief propagation.

We have also carried out the same analysis on the Gibbs free energy (F_G) for the join tree algorithm. Since $U_G = U_B$, we obtain the same expression for the average energy as before,

$$\frac{\partial U_G}{\partial p_{sr}} = \frac{\partial U_B}{\partial p_{sr}} = -\frac{\partial}{\partial p_{sr}}\left\{\sum_{s,r,w} bel(s,r,w)\ln P(w\,|\,s,r)\right\}.$$

To determine the entropy term of the Gibbs free energy, we have to make use of the join tree factorisation

$$P(c,s,r,w) = \frac{P(c,s,r)P(s,r,w)}{P(s,r)} = P(c,s,r)P(w\,|\,s,r)$$

which allows us to rewrite

$$bel(c,s,r,w) = bel(c,s,r)P(w\,|\,s,r)$$

to yield

$$\begin{aligned}
H_G &= -\left\{\sum_{c,s,r,w} bel(c,s,r,w)\ln bel(c,s,r,w)\right\} \\
&= -\left\{\sum_{c,s,r,w} bel(c,s,r)P(w\,|\,s,r)\ln bel(c,s,r) + \sum_{c,s,r,w} bel(c,s,r)P(w\,|\,s,r)\ln P(w\,|\,s,r)\right\} \\
&= -\left\{\sum_{c,s,r} bel(c,s,r)\ln bel(c,s,r) + \sum_{s,r,w} bel(s,r)P(w\,|\,s,r)\ln P(w\,|\,s,r)\right\} \\
&= -\left\{\sum_{c,s,r} bel(c,s,r)\ln bel(c,s,r) + \sum_{s,r,w} bel(s,r,w)\ln P(w\,|\,s,r)\right\}
\end{aligned}$$

Putting the average energy and entropy terms together, we have

$$\begin{aligned}
\frac{\partial F_G}{\partial p_{sr}} &= \frac{\partial U_G}{\partial p_{sr}} - \frac{\partial H_G}{\partial p_{sr}} \\
&= -\frac{\partial}{\partial p_{sr}}\left\{\sum_{s,r,w} bel(s,r,w)\ln P(w\,|\,s,r)\right\} \\
&\quad + \frac{\partial}{\partial p_{sr}}\left\{\sum_{c,s,r} bel(c,s,r)\ln bel(c,s,r) + \sum_{s,r,w} bel(s,r,w)\ln P(w\,|\,s,r)\right\} \\
&= \frac{\partial}{\partial p_{sr}}\left\{\sum_{c,s,r} bel(c,s,r)\ln bel(c,s,r)\right\}
\end{aligned}$$

Contrasting the above to $\frac{\partial F_B}{\partial p_{sr}} = \frac{\partial}{\partial p_{sr}}\left\{\sum_{s,r} bel(s,r)\ln bel(s,r)\right\}$, it clearly shows the difference between join tree algorithm and loopy belief propagation's handling of the dependency between node *S* and node *R*. In this case, the Gibbs free energy correctly accounts for the dependency between node *S* and node *R* due to node *C* by maintaining the belief function $bel(c,s,r)$.

From the analysis presented in this section, we see that the loopy belief propagation can in fact compute the true marginal probabilities for nodes *C*, *S* and *R* for the single loop network shown in Figure 6.1(a). The only problem is with node *W*, where the joint probability of nodes *S* and *R* is required but it is hampered by the message passing process which only provides beliefs of single nodes. In some sense, we can think of node *C* being the "culprit" that causes the problem at node *W*. We shall label the root node of a loop the source node to indicate that it is the source of where the unaccounted dependency originated from. Likewise, we shall label the converging node of a loop as the sink node, and the remaining nodes as intermediate nodes. Hence, node *C* is the source node, node *W* is the sink node, and nodes *S* and *R* are the intermediate nodes. By identifying the type of nodes in a loop it allows us to pinpoint where the loopy error originates from and which type of nodes are problematic according to the state of instantiations.

6.2 Error analysis on simple single loop with no instantiation

In this section we derive for the single loop Bayesian network shown in Figure 6.2 with no instantiation the error term

$$Error(S,R) = P_{true}(S,R) - P_{indep}(S,R)$$

where $P_{true}(S,R)$ denotes the true joint probability $P(S,R)$ and $P_{indep}(S,R)$ denotes the joint probability computed by the loopy belief propagation with the assumption that node *S* and node *R* are marginally independent, i.e. $P_{indep}(S,R) = P(S)P(R)$. Once we know how to compute the error term, we can reconstruct the true joint probability $P(S,R)$ from the individual $P(S)$ and $P(R)$ that are sent as messages in loopy belief propagation.

As discussed in the previous section, the source of error for loopy belief propagation comes from the source node *C* when there is no instantiation. To derive the error equations let us refer to the generic CPT entries of the single loop Bayesian network shown in Figure 6.2. In this case, it is not necessary to specify the details of the CPT

$P(W \mid S, R)$ as discussed in previous section. Note that the number of states of nodes *C*, *S* and *R* are denoted by |*C*|, |*S*| and |*R*| respectively, and the generic CPT entries are $P(s_j \mid c_i) = \alpha_i^j$ and $P(r_k \mid c_i) = \beta_i^k$.

$$P(C) = \pi(C) = \begin{pmatrix} \pi(c_1) \\ \pi(c_2) \\ \vdots \\ \pi(c_{|C|}) \end{pmatrix}$$

$P(S \mid C)$	s_1	s_2	$\cdots$	$s_{|S|}$
c_1	α_1^1	α_1^2	$\cdots$	$\alpha_1^{|S|}$
c_2	α_2^1	α_2^2	$\cdots$	$\alpha_2^{|S|}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$c_{|C|}$	$\alpha_{|C|}^1$	$\alpha_{|C|}^2$	$\cdots$	$\alpha_{|C|}^{|S|}$

$P(R \mid C)$	r_1	r_2	$\cdots$	$r_{|R|}$
c_1	β_1^1	β_1^2	$\cdots$	$\beta_1^{|R|}$
c_2	β_2^1	β_2^2	$\cdots$	$\beta_2^{|R|}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$c_{|C|}$	$\beta_{|C|}^1$	$\beta_{|C|}^2$	$\cdots$	$\beta_{|C|}^{|R|}$

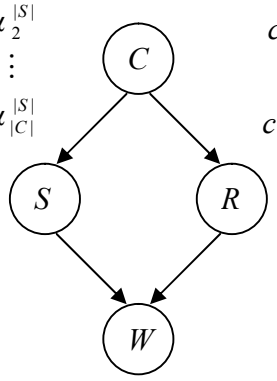


Figure 6.2: A single loop with generic CPT entries.

Let us recapitulate that the error at node *W* is due to the term $\left[\pi_w(s_j)\pi_w(r_k)\right]$ in the following equation:

$$\pi(w_i) = \sum_{s_j, r_k} P(w_i \mid s_j, r_k)\left[\pi_w(s_j)\pi_w(r_k)\right],$$

and with no evidence from node *W* to node *S* and node *R*, we have $\pi_W(s_j) = P(s_j)$ and $\pi_W(r_k) = P(r_k)$ as explained in the previous section. Furthermore, the above equation is equivalent to computing

$$P(w) = \sum_{s,r} P(w \mid s, r)P(s)P(r).$$

Let us now derive the error term $Error(S, R)$ by studying its (*j*,*k*) entry:

$$Error(s_j, r_k) = P_{true}(s_j, r_k) - P_{indep}(s_j, r_k)$$

where

$$\begin{aligned} P_{true}(s_j, r_k) &= \pi_{true}(s_j, r_k) \\ &= \sum_{c,w} P(w \mid s_j, r_k) P(s_j \mid c) P(r_k \mid c) P(c) \\ &= \sum_{c} P(s_j \mid c) P(r_k \mid c) P(c) \sum_{w} P(w \mid s_1, r_1) \\ &= \sum_{c} P(s_j \mid c) P(r_k \mid c) P(c) \\ &= \sum_{i} \alpha_i^j \beta_i^k \pi(c_i) \end{aligned}$$

and

$$\begin{aligned} P_{indep}(s_j, r_k) &= P_{true}(s_j) * P_{true}(r_k) \\ &= \pi_w(s_j) * \pi_w(r_k) \\ &= \left(\sum_{c} P(s_j \mid c) P(c) \right) \left(\sum_{c} P(r_k \mid c) P(c) \right). \\ &= \left(\sum_{i} \alpha_i^j \pi(c_i) \right) \left(\sum_{l} \beta_l^k \pi(c_l) \right) \end{aligned}$$

Hence, we have

$$\begin{aligned} & Error(s_j, r_k) \\ &= \sum_{i} \alpha_i^j \beta_i^k \pi(c_i) - \left(\sum_{i} \alpha_i^j \pi(c_i) \right) \left(\sum_{l} \beta_l^k \pi(c_l) \right) \\ &= \sum_{i} \alpha_i^j \beta_i^k \pi(c_i) - \left(\sum_{i<l} (\alpha_i^j \beta_l^k + \alpha_l^j \beta_i^k) \pi(c_i) \pi(c_l) + \sum_{i} \alpha_i^j \beta_i^k \pi(c_i) \pi(c_i) \right) \\ &= \left(\sum_{i} \alpha_i^j \beta_i^k \pi(c_i) - \sum_{i} \alpha_i^j \beta_i^k \pi(c_i) \pi(c_i) \right) - \sum_{i<l} (\alpha_i^j \beta_l^k + \alpha_l^j \beta_i^k) \pi(c_i) \pi(c_l) \\ &= \left(\sum_{i} \alpha_i^j \beta_i^k \pi(c_i)(1 - \pi(c_i)) \right) - \sum_{i<l} (\alpha_i^j \beta_l^k + \alpha_l^j \beta_i^k) \pi(c_i) \pi(c_l) \end{aligned}$$

To further simplify the above expression, we observe that since $\sum_i \pi(c_i) = 1$, we can then rewrite $(1 - \pi(c_i)) = \sum_{l \neq i} \pi(c_l)$, which means

$$\sum_i \alpha_i^j \beta_i^k \pi(c_i)(1-\pi(c_i)) = \sum_i \left(\alpha_i^j \beta_i^k \pi(c_i) \sum_{l \neq i} \pi(c_l) \right)$$
$$= \sum_{l<i} \alpha_i^j \beta_i^k \pi(c_i)\pi(c_l) + \sum_{l>i} \alpha_i^j \beta_i^k \pi(c_i)\pi(c_l)$$
$$= \sum_{i<l} \alpha_l^j \beta_l^k \pi(c_l)\pi(c_i) + \sum_{i<l} \alpha_i^j \beta_i^k \pi(c_i)\pi(c_l)$$
$$= \sum_{i<l} (\alpha_i^j \beta_i^k + \alpha_l^j \beta_l^k)\pi(c_i)\pi(c_l)$$

By substituting it into the previous expression, we obtain

$$Error(s_j, r_k)$$
$$= \left(\sum_i \alpha_i^j \beta_i^k \pi(c_i)(1-\pi(c_i)) \right) - \sum_{i<l} (\alpha_i^j \beta_l^k + \alpha_l^j \beta_i^k)\pi(c_i)\pi(c_l)$$
$$= \sum_{i<l} (\alpha_i^j \beta_i^k + \alpha_l^j \beta_l^k)\pi(c_i)\pi(c_l) - \sum_{i<l} (\alpha_i^j \beta_l^k + \alpha_l^j \beta_i^k)\pi(c_i)\pi(c_l)$$
$$= \sum_{i<l} (\alpha_i^j - \alpha_l^j)(\beta_i^k - \beta_l^k)\pi(c_i)\pi(c_l) \qquad \text{---------- equation (6.1)}$$

and this completes our derivation of the error term $Error(S,R)$. It turns out that the error term takes on a much simpler form if the nodes are binary, as we illustrate in the following examples.

Example 6.2.1. Consider the case where $|C| = 2$, then we have

$$Error(S,R)$$
$$= \begin{bmatrix} (\alpha_1^1-\alpha_2^1)(\beta_1^1-\beta_2^1) & (\alpha_1^1-\alpha_2^1)(\beta_1^2-\beta_2^2) & \cdots & (\alpha_1^1-\alpha_2^1)(\beta_1^{|R|}-\beta_2^{|R|}) \\ (\alpha_1^2-\alpha_2^2)(\beta_1^1-\beta_2^1) & (\alpha_1^2-\alpha_2^2)(\beta_1^2-\beta_2^2) & \cdots & (\alpha_1^2-\alpha_2^2)(\beta_1^{|R|}-\beta_2^{|R|}) \\ \vdots & \vdots & \ddots & \vdots \\ (\alpha_1^{|S|}-\alpha_2^{|S|})(\beta_1^1-\beta_2^1) & (\alpha_1^{|S|}-\alpha_2^{|S|})(\beta_1^2-\beta_2^2) & \cdots & (\alpha_1^{|S|}-\alpha_2^{|S|})(\beta_1^{|R|}-\beta_2^{|R|}) \end{bmatrix} \pi(c_1)\pi(c_2).$$

If we also assume $|S| = 2$ and $|R| = 2$, the error term will further simplify to

$$Error(S,R) = \begin{bmatrix} (\alpha_1^1-\alpha_2^1)(\beta_1^1-\beta_2^1) & (\alpha_1^1-\alpha_2^1)(\beta_1^2-\beta_2^2) \\ (\alpha_1^2-\alpha_2^2)(\beta_1^1-\beta_2^1) & (\alpha_1^2-\alpha_2^2)(\beta_1^2-\beta_2^2) \end{bmatrix} \pi(c_1)\pi(c_2)$$
$$= \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} (\alpha_1^1-\alpha_2^1)(\beta_1^1-\beta_2^1)\pi(c_1)\pi(c_2)$$

where the simplification is due to

$$(\alpha_1^2 - \alpha_2^2) = (1-\alpha_1^1) - (1-\alpha_2^1) = -(\alpha_1^1 - \alpha_2^1)$$

and

$$(\beta_1^{\,2} - \beta_2^{\,2}) = (1 - \beta_1^{\,1}) - (1 - \beta_2^{\,1}) = -(\beta_1^{\,1} - \beta_2^{\,1}) .$$

Example 6.2.2. Suppose we have the case where $|S| = 2$, $|R| = 2$, and $|C| = n$, then by applying the same simplification as used in previous example, we obtain

$$Error(S,R) = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \times \sum_{i<l}^{n} (\alpha_1^1 - \alpha_2^1)(\beta_1^1 - \beta_2^1)\pi(c_i)\pi(c_l) .$$

It is interesting to see, from the above examples where both node *S* and node *R* are binary, that the error matrix is symmetric with only a flip of sign for the off-diagonal entries and all four error terms share the same error magnitude regardless of the number of states of the source node *C*. This leads to the property that the sum of each column or each row of the error matrix is always zero. The error matrix also brings out the point that the magnitude of the error is dependent on two factors, namely, the network parameters appearing in the term $((\alpha_i^j - \alpha_l^j)(\beta_i^k - \beta_l^k))$ and the evidence entering the loop through the source node C as $\pi(C)$. The term $(\alpha_i^j - \alpha_l^j)$ actually measures the degree of dependency of node *S* on node *C*. In the binary case, if $(\alpha_1^1 = \alpha_2^1)$ then effectively we have $P(S \mid C) = P(S)$, the error becomes zero and $P_{true}(S,R) = P_{indep}(S,R)$. The same can also be said of $(\beta_1^1 - \beta_2^1)$. On the other hand, the error also approaches zero as the value of any one of the terms $\pi(c_i)$ approaches value 1. When $\pi(c_i)$ equals 1, it means that node *C* has been instantiated to one of its i^{th} state and thus making node *S* and node *R* marginally independent. Lastly, given fixed CPT parameters $(\alpha_i^j, \alpha_l^j, \beta_1^1, \beta_2^1)$, the error is largest when $\pi(c_1) = \pi(c_2) = 0.5$, where node *C* is binary.

Finally, we demonstrate how we can make use of the derived error term to compute the exact marginal probability of node *W* for the case where nodes *C*, *S*, and *R* are all binary. First, we compute the error matrix $Error(S,R)$ using equation (6.1) and use it to recover the true $P_{true}(S,R)$:

$$P_{true}(S,R) = P_{indep}(S,R) + Error(S,R) .$$

Once we have recovered $P_{true}(S,R)$, we can obtain $P_{true}(W)$ as follows

$$
\begin{aligned}
P_{true}(W) &= \pi(W) \\
&= \begin{bmatrix} \pi(w_1) \\ \vdots \\ \pi(w_n) \end{bmatrix} \\
&= \begin{bmatrix} P(w_1 \mid s_1, r_1) & P(w_1 \mid s_1, r_2) & P(w_1 \mid s_2, r_1) & P(w_1 \mid s_2, r_2) \\ \vdots & \vdots & \vdots & \vdots \\ P(w_n \mid s_1, r_1) & P(w_n \mid s_1, r_2) & P(w_n \mid s_2, r_1) & P(w_n \mid s_2, r_2) \end{bmatrix} \times \\
&\qquad \left\{ \begin{bmatrix} \pi_w(s_1)\pi_w(r_1) \\ \pi_w(s_1)\pi_w(r_2) \\ \pi_w(s_2)\pi_w(r_1) \\ \pi_w(s_2)\pi_w(r_2) \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} (\alpha_1^1 - \alpha_2^1)(\beta_1^1 - \beta_2^1)\pi(c_1)\pi(c_2) \right\}
\end{aligned}
$$

Note that node W is not restricted to be binary, and is shown as having n states in the equation above. Furthermore, the CPT entries of $P(W \mid S,R)$ are not required for the error term computation and are only required for computing marginal distribution of node W.

6.3 Exact inference in a generic single loop with no instantiation using standard belief propagation with loop correction

Building on the analysis result presented in the previous section, it turns out that extending the error term derivation to a generic single loop like the one shown in Figure 6.3 is quite straight forward. In this network, there are n S-nodes and m R-nodes. In this case we observe that we can consolidate the conditional probabilities of all nodes between node C and node S_n, and node C and node R_m, prior to the start of the belief propagation process, as indicated by the dotted arrows in Figure 6.3. In effect, we are pre-computing the additional CPTs $P(S_n \mid C)$ and $P(R_m \mid C)$. Alternatively, we can think of it as transforming the larger loop into a smaller loop involving only the nodes C, S_n, R_m, and W. Once this is done, the error term derived in the previous section (using equation (6.1)) can be directly applied.

Let us use an example to illustrate this point before we present the exact inference algorithm using standard belief propagation with loop correction.

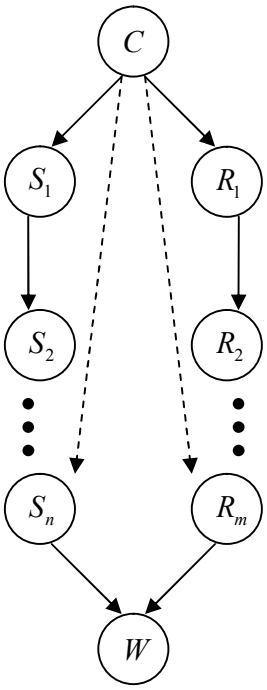


Figure 6.3: A generic size single loop.

Example 6.3.1. Consider a case of the generic single loop Bayesian network shown in Figure 6.3 where all the variables are binary and with the following CPTs:

- $P(C) = \pi(C) = \begin{pmatrix} \pi(c_1) \\ \pi(c_2) \end{pmatrix}$, $\begin{array}{c|cc} P(S_1 \mid C) & s_1 & s_2 \\ \hline c_1 & a_1^1 & a_2^1 \\ c_2 & b_1^1 & b_2^1 \end{array}$, $\begin{array}{c|cc} P(R_1 \mid C) & r_1 & r_2 \\ \hline c_1 & p_1^1 & p_2^1 \\ c_2 & q_1^1 & q_2^1 \end{array}$

- for i = 1 to n-1: $\begin{array}{c|cc} P(S_{i+1} \mid S_i) & S_{i+1} = state_1 & S_{i+1} = state_2 \\ \hline S_i = state_1 & a_1^{i+1} & a_2^{i+1} \\ S_i = state_2 & b_1^{i+1} & b_2^{i+1} \end{array}$,

- for i = 1 to m-1: $\begin{array}{c|cc} P(R_{i+1} \mid R_i) & R_{i+1} = state_1 & R_{i+1} = state_2 \\ \hline R_i = state_1 & p_1^{i+1} & p_2^{i+1} \\ R_i = state_2 & q_1^{i+1} & q_2^{i+1} \end{array}$.

In this case, we pre-compute the additional CPTs $P(S_n \mid C)$ and $P(R_m \mid C)$ as follows:

- $$P(S_n \mid C) = \begin{bmatrix} a_1^n & b_1^n \\ a_2^n & b_2^n \end{bmatrix} \times \cdots \times \begin{bmatrix} a_1^1 & b_1^1 \\ a_2^1 & b_2^1 \end{bmatrix}$$
- $$P(R_m \mid C) = \begin{bmatrix} p_1^m & q_2^m \\ p_1^m & q_2^m \end{bmatrix} \times \cdots \times \begin{bmatrix} p_1^1 & q_2^1 \\ p_1^1 & q_2^1 \end{bmatrix}$$

Once we have the above CPTs, we can apply error equation (6.1) to recover the term $P_{true}(S_n, R_m)$, where $[\alpha_i^j] = P(S_n \mid C)$ and $[\beta_i^k] = P(R_m \mid C)$. On further inspection, we discovered that a much simpler expression for the error term can be derived for this binary case in a more compact way by operating using matrix representations:

$$\begin{aligned}
&Error(S_n, R_m) \\
&= P_{true}(S_n, R_m) - P_{indep}(S_n, R_m) \\
&= \left(\begin{bmatrix} a_1^n & b_1^n \\ a_2^n & b_2^n \end{bmatrix} \cdots \begin{bmatrix} a_1^1 & b_1^1 \\ a_2^1 & b_2^1 \end{bmatrix} \right) \left(\begin{bmatrix} \pi(c_1) & 0 \\ 0 & \pi(c_2) \end{bmatrix} - \begin{bmatrix} \pi(c_1)\pi(c_1) & \pi(c_1)\pi(c_2) \\ \pi(c_1)\pi(c_2) & \pi(c_2)\pi(c_2) \end{bmatrix} \right) \times \\
&\qquad \left(\begin{bmatrix} p_1^m & p_2^m \\ q_1^m & q_2^m \end{bmatrix} \cdots \begin{bmatrix} p_1^1 & p_2^1 \\ q_1^1 & q_2^1 \end{bmatrix} \right)
\end{aligned}$$

which can be further simplified to

$$\begin{aligned}
&Error(S_n, R_m) \\
&= \left(\begin{bmatrix} a_1^n & b_1^n \\ a_2^n & b_2^n \end{bmatrix} \cdots \begin{bmatrix} a_1^1 & b_1^1 \\ a_2^1 & b_2^1 \end{bmatrix} \right) \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \left(\begin{bmatrix} p_1^m & p_2^m \\ q_1^m & q_2^m \end{bmatrix} \cdots \begin{bmatrix} p_1^1 & p_2^1 \\ q_1^1 & q_2^1 \end{bmatrix} \right) \pi(c_1)\pi(c_2) \\
&= \left(\begin{bmatrix} a_1^n & b_1^n \\ a_2^n & b_2^n \end{bmatrix} \cdots \begin{bmatrix} a_1^2 & b_1^2 \\ a_2^2 & b_2^2 \end{bmatrix} \right) \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \left(\begin{bmatrix} p_1^m & p_2^m \\ q_1^m & q_2^m \end{bmatrix} \cdots \begin{bmatrix} p_1^1 & p_2^1 \\ q_1^1 & q_2^1 \end{bmatrix} \right) (a_1^1 - b_1^1)\pi(c_1)\pi(c_2) \\
&\vdots \\
&= \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \left(\begin{bmatrix} p_1^m & p_2^m \\ q_1^m & q_2^m \end{bmatrix} \cdots \begin{bmatrix} p_1^1 & p_2^1 \\ q_1^1 & q_2^1 \end{bmatrix} \right) \pi(c_1)\pi(c_2) \prod_{i=1}^{n} (a_1^i - b_1^i) \\
&= \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \left(\begin{bmatrix} p_1^{m-1} & p_2^{m-1} \\ q_1^{m-1} & q_2^{m-1} \end{bmatrix} \cdots \begin{bmatrix} p_1^1 & p_2^1 \\ q_1^1 & q_2^1 \end{bmatrix} \right) (p_1^m - q_1^m)\pi(c_1)\pi(c_2) \prod_{i=1}^{n} (a_1^i - b_1^i) \\
&\vdots \\
&= \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \pi(c_1)\pi(c_2) \prod_{i=1}^{n} (a_1^i - b_1^i) \prod_{j=1}^{m} (p_1^j - q_1^j)
\end{aligned}$$

By comparing this error term to the one derived in Example 6.2.1, it is interesting to see that the effect of each additional intermediate node in the loop contributes an extra term of either $(a_1^i - b_1^i)$ or $(p_1^j - q_1^j)$ to the total error. The remarks from the previous section that the magnitude of the error is dependent on the network parameters appearing in the term $((\alpha_i^j - \alpha_l^j)(\beta_i^k - \beta_l^k))$ are again applicable here, and in addition, the loop error decreases as the size of the loop increases since $\left|(a_1^i - b_1^i)\right| \le 1$ and $\left|(p_1^j - q_1^j)\right| \le 1$, thus $\prod_{i=1}^{n}(a_1^i - b_1^i)\prod_{j=1}^{m}(p_1^j - q_1^j) \to 0$ as $n, m \to \infty$. This last observation has also been reported by Bidyuk and Dechter [67], where a similar error expression for the specific case of binary nodes was derived by using mathematical induction for the purpose of studying the epsilon-cutset effects in Bayesian networks.

We are now ready to present the algorithm next.

Algorithm 6.3 Single Downward Pass Belief Propagation with Loop Correction for Single Loop Network without Instantiations

Input: *A Bayesian network G as shown in Figure 6.3*

Output: *Marginal probability of all nodes in G*

1. *Compute the additional CPTs* $P(S_n \mid C)$ *and* $P(R_m \mid C)$.
2. *Downward pass:*

   For i = 1 to n,

   Perform standard belief propagation from parent node to child node S_i, *i.e., node* S_i *will receive the update message* $\pi(S_i)$.

   End {For}

   For i = 1 to m,

   Perform standard belief propagation from parent node to child node R_i, *i.e., node* R_i *will receive the update message* $\pi(R_i)$.

   End {For}
3. *Error Correction Step:*

Compute $Error(S_n, R_m)$ *using equation (6.1) from Section 6.2, which requires the CPTs* $P(S_n \mid C)$ *and* $P(R_m \mid C)$ *from Step 1.*

Compute $\pi(S_n, R_m) = \pi(S_n)\pi(R_m) + Error(S_n, R_m)$

4. *Compute marginal probability:*

   - $P(C) = \pi(C)$

   - *For i = 1 to n,* $P(S_i) = \pi(S_i)$ *End {For}*

   - *For i = 1 to m,* $P(R_i) = \pi(R_i)$ *End {For}*

   - $P(W) = P(W \mid S_n, R_m) \times \pi(S_n, R_m)$

5. *Output marginal probability of all nodes in G.*

Note that there is no need to perform any upward pass belief propagation in this case as the messages will all be uniform. Furthermore, for the case where the network contains only binary source and immediate nodes, we can apply the simpler error term expression presented in Example 6.3.1 to Step 1 and Step 4. The computational cost of algorithm 6.3 is the same as a single outward pass of standard belief propagation starting from the root node C to the leaf nodes S_n and R_m plus the additional cost of error correction, as well as the extra storage space for holding the additional CPTs at Step 1. Note that the computation of the error term coefficients $(\alpha_i^j - \alpha_l^j)(\beta_i^k - \beta_l^k)$ in equation (6.1) can also be done offline prior to the start of the downward pass to speed up Step 3.

It is quite obvious that algorithm 6.3 will be far more efficient than the join tree algorithm when applied to a generic single loop, but nevertheless, we will use the following example to highlight this point.

Example 6.3.2. Figure 6.3 shows an 8-node single loop Bayesian network and its corresponding join tree. Applying algorithm 6.3 to this network only requires sending 8 downward pass messages and the additional step of error correction. On the other hand, the inward pass of the join tree algorithm from clique {*A,B,C*} to clique {*F,G,H*} becomes an overkill process as it involves sending 5 inward messages each requiring

marginalization of a 3-node potential function to the 2-node separator, as well as the higher storage space required for clique variables. Of course in this case, one could also argue that perhaps loop-cutset conditioning method will be a more efficient method to use instead of join tree. Hence, we shall compare algorithm 6.3 to loop-cutset in the next example.

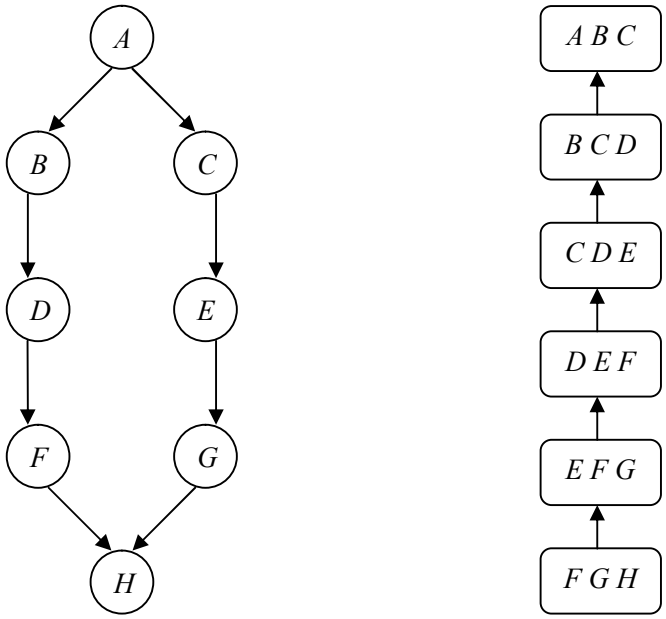


Figure 6.4: Example of a single loop Bayesian network (a) and its join tree (b).

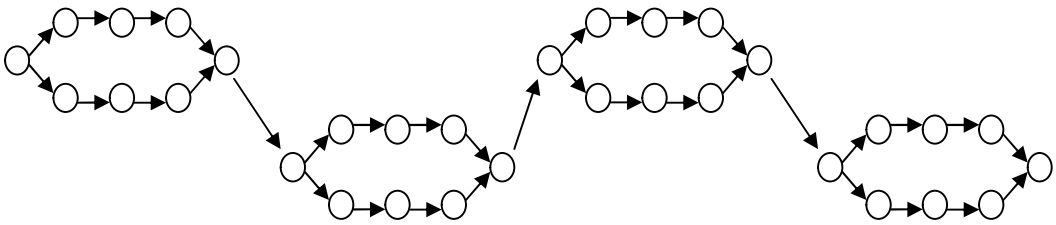


Figure 6.5: Example of a Bayesian network with multiple loops.

Example 6.3.3. In this example let us first consider applying loop-cutset conditioning to the network in Figure 6.4(a) by conditioning on node *A*. For each state of node *A*, 8 downward pass messages are required. If node *A* is binary, then a total of 16 messages

will be sent, and also there are the extra steps of computing the marginal probability through summing over the instantiations of node A. Perhaps the advantage of algorithm 6.3 over loop-cutset conditioning is more evident if we consider a network with several connected loops. An example of such a network is shown in Figure 6.5. Let us suppose that there are n such loops and all the variables are binary, then the loop-cutset will have a size of n nodes and will require to run belief propagation on 2^n instantiations. On the other hand, we only need to run algorithm 6.3 n times to handle the loops in sequence and standard belief propagation along the connecting arcs among nodes that link the loops.

We wish to highlight that algorithm 6.3 can actually be used together with the join tree algorithm or loop-cutset conditioning, and possibly other belief propagation algorithms, on networks that contain one or more single loops, provided these algorithms can subsequently make use of the single variable marginal probability or the respective π and λ messages produced.

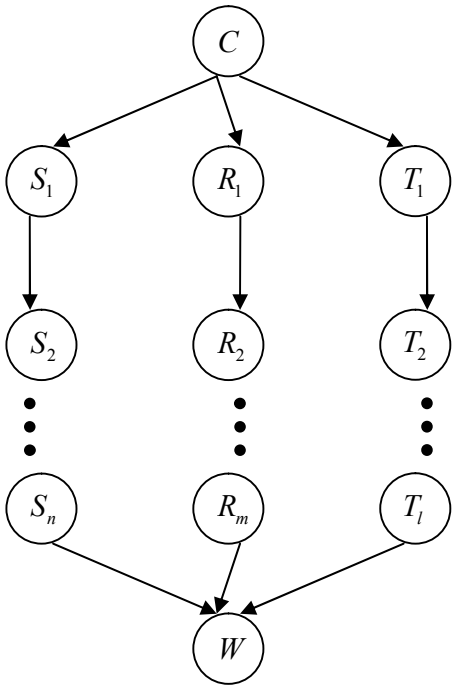


Figure 6.6: A Bayesian network with double loops sharing the same source and sink nodes.

Example 6.3.4. In this example, we will discuss how we can extend the error term and thus algorithm 6.3 to handle a network with parallel loops like the one shown in Figure 6.5. In this double loop example, the sink node now has 3 parent nodes instead of 2, which means we will have to compute $Error(S_n, R_m, T_l)$. One approach to this problem is to repeat the same error term derivation in section 6.2 and expand it to handle 3 nodes. The other approach is to recursively apply the single loop error correction starting from, say, the left most loop and moving right. For example, we can first compute $Error(S_n, R_m)$ to recover $\pi(S_n, R_m)$ and then apply it to compute $Error(\{S_n, R_m\}, T_l)$ by treating the cluster node $\{S_n, R_m\}$ as a single node. To do this we will also need to compute the additional CPT of $P(\{S_n, R_m\} | C)$ by combining $P(S_n | C)$ and $P(R_m | C)$.

6.4 Loopy error due to evidence entering the loop through the sink node

In the section, we will examine the loopy error due to evidence that enters the loop through the sink node. Evidence can enter a loop through the sink node when it is instantiated or when it has received a λ message that contains evidence from its child node. In either case, we shall say that the sink node has received λ evidence. The nature of the loopy error due to evidence entering through the sink node is very different from the preceding cases we have presented so far. Recall that in a single loop, if the source node or any of the intermediate nodes is instantiated, it effectively reduces the network to a singly connected one, and the loopy belief propagation will compute the exact marginal posterior probabilities for all the nodes in the loop. Furthermore, as long as the sink node has not received any λ evidence, even with other evidence entering the loop through the source node and intermediate nodes, the loopy belief propagation process will always converge. For example, in the single loop network shown in Figure 6.2, if node *W* has not received any λ evidence, the messages $\lambda_W(R)$ and $\lambda_W(S)$ sent between node *S* and node

R through node W will always be uniform vectors. This is due to the fact that node S and node R are marginally independent in the converging network structure ($S \rightarrow W \leftarrow R$). In other words, for a single loop Bayesian network where the sink node has not received any λ evidence, the loopy belief propagation process will always converge to the true marginal probabilities for all the nodes except the sink node, for which we have derived the error terms. On the other hand, if the sink node has received λ evidence, it causes node S and node R to become conditionally dependent. In this case, the messages $\lambda_W(R)$ and $\lambda_W(S)$ sent between node S and node R through node W are no longer uniform vectors, causing incorrect 'feedback' of evidence circulating in the loop and resulting in wrong posterior probabilities being computed. It has been proven that loopy belief propagation will always converge to a unique fixed point when applied to a single loop in an undirected graphical model with finite-strength potentials [50], [68].

Recall that the joint probability distribution of this network is given by $P(c,s,r,w) = P(w \mid s,r)P(s \mid c)P(r \mid c)P(c)$, and we have shown in section 6.2 that when node W has not received any λ evidence, the joint probability $P(s,r)$ does not contain the terms $P(w \mid s,r)$, which is

$$P(s,r) = \sum_c P(s \mid c)P(r \mid c)P(c)$$

and let us consider the binary case where we can write in matrix form

$$P_{true}(S,R) = \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \begin{bmatrix} c_1 & 0 \\ 0 & c_2 \end{bmatrix} \begin{bmatrix} p_1 & p_2 \\ q_1 & q_2 \end{bmatrix}.$$

Note that to help in improving the readability of the expressions we have made a slight abuse of notation here by using c_1 to denote the first state of node C as well as $P(C = c_1) = \pi(c_1) = c_1$, and using c_2 in a similar fashion. In any case, there should be no ambiguity to what c_1 and c_2 represent given the context of the expression they appear in. Also, note that $c_2 = 1 - c_1$, $a_2 = 1 - a_1$, $b_2 = 1 - b_1$, $p_2 = 1 - p_1$, and $q_2 = 1 - q_1$. Now, in this case where node W has received some λ evidence, we have to compute

$$
\begin{aligned}
P(s,r \mid w) &= \alpha \sum_{c} P(w \mid s,r) P(s \mid c) P(r \mid c) P(c) \\
&= \alpha P(w \mid s,r) \sum_{c} P(s \mid c) P(r \mid c) P(c) \\
&= \alpha \lambda(s,r) \pi(s,r)
\end{aligned}
$$

where α is a normalizing constant and

$$
\pi(S,R) = \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \begin{bmatrix} c_1 & 0 \\ 0 & c_2 \end{bmatrix} \begin{bmatrix} p_1 & p_2 \\ q_1 & q_2 \end{bmatrix}.
$$

Note that the additional factor in this case is the term $\lambda(s,r)$ due to the λ evidence of node *W*. If node *W* has been instantiated to a particular state, say w_i, then $\lambda(s,r) = P(w_i \mid s,r)$, where $P(w_i \mid s,r)$ is a CPT entry of node *W*. On the other hand, if node *W* has received a λ message, then $\lambda(s,r) = \sum_{w_i} \lambda(w_i) P(w_i \mid s,r)$. Whichever case let us denote

$$
\lambda(S,R) = \begin{bmatrix} x & y \\ u & v \end{bmatrix}
$$

and hence we have

$$
P_{true}(S,R \mid W) = \alpha \left(\begin{bmatrix} x & y \\ u & v \end{bmatrix} \right) \otimes \left(\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \begin{bmatrix} c_1 & 0 \\ 0 & c_2 \end{bmatrix} \begin{bmatrix} p_1 & p_2 \\ q_1 & q_2 \end{bmatrix} \right).
$$

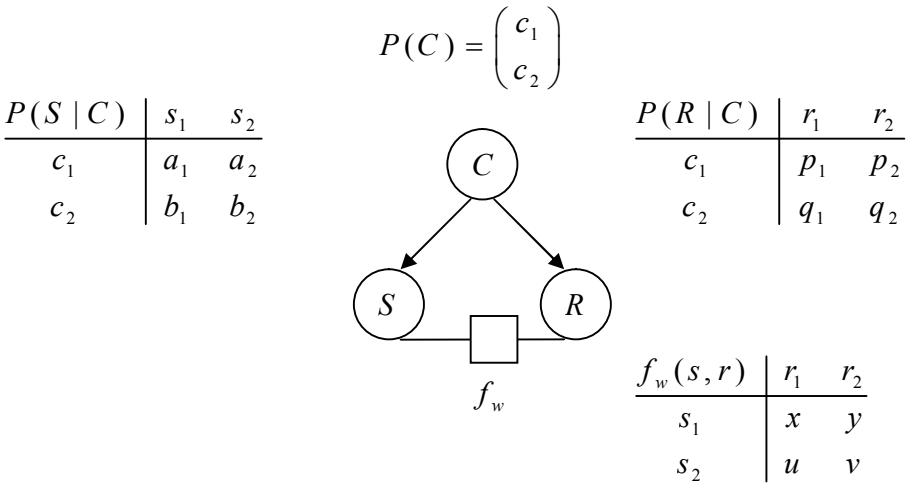


Figure 6.7: Virtual hybrid graph showing the effects of λ evidence received at node *W*.

In this case, we can think of $\lambda(s,r)$ as a function of the factor node f_w in the virtual hybrid graph shown in Figure 6.7. This hybrid graph depicts the effects on the message passing process due to the λ evidence received at node *W*. What it illustrates is that now there is a direct link of message passing between node *S* and node *R* through the factor node f_w. Specifically, node *S* sends the following message to node *R*:

$$\lambda_W(R) = \left(\begin{bmatrix} \pi(s_1) & \pi(s_2) \end{bmatrix} \begin{bmatrix} x & y \\ u & v \end{bmatrix} \right)^T$$

and likewise, node *R* sends the following message to node *S*:

$$\lambda_W(S) = \begin{bmatrix} x & y \\ u & v \end{bmatrix} \begin{bmatrix} \pi(r_1) \\ \pi(r_2) \end{bmatrix}.$$

When analysing the loopy error in previous cases where node *W* has not received any λ evidence, we have ignored the order in which the messages are being passed in the loop because the belief propagation process always converges to the true marginal probabilities of node *C*, *S*, and *R*, which allows us to recover $P_{true}(s,r)$ from $P_{true}(s)$ and $P_{true}(r)$ using the derived error term. Furthermore, by recovering $P_{true}(s,r)$ it leads to the correct computation of $P_{true}(w)$. However, in this case, while convergence is guaranteed, the belief propagation process might not produce the terms $P_{true}(s)$ and $P_{true}(r)$. This is due to the erroneous circulating of evidence within the loop through the factor node f_w. In fact, there are two directions of evidence circulation. In the clockwise direction, we have:

- node *R* sends $\lambda_W(S) = \begin{bmatrix} x & y \\ u & v \end{bmatrix} \begin{bmatrix} \pi(r_1) \\ \pi(r_2) \end{bmatrix}$ to node *S*,
- node *S* sends $\lambda_S(C)$, which contains the term $\lambda_W(S)$, to node *C*,
- node *C* sends $\pi_R(C)$ to node *R*, which is used to update $\pi(R)$,
- the updated $\pi(R)$ causes node *R* to send a new $\lambda_W(S)$ to node *S* (which leads to erroneous compounding of λ evidence at node *S*) and the process continues until it converges.

Similarly, there is an anti-clockwise circulation of evidence from node *S* to node *R*, node *R* to node *C*, and then from node *C* back to node *S*. The true marginal probability of node *S* can be computed as follows

$$P_{true}(s \mid w) = \alpha \sum_{c,r} P(w \mid s,r)P(s \mid c)P(r \mid c)P(c)$$
$$= \alpha \sum_{r} \left(P(w \mid s,r) \sum_{c} P(s \mid c)P(r \mid c)P(c) \right)$$

which in matrix form is given by

$$P_{true}(S \mid w) = \alpha \begin{bmatrix} x(a_1p_1c_1 + b_1q_1c_2) + y(a_1p_2c_1 + b_1q_2c_2) \\ u(a_2p_1c_1 + b_2q_1c_2) + v(a_2p_2c_1 + b_2q_2c_2) \end{bmatrix}$$
$$= \alpha \begin{bmatrix} (xp_1 + yp_2)a_1c_1 + (xq_1 + yq_2)b_1c_2 \\ (up_1 + vp_2)a_2c_1 + (uq_1 + vq_2)b_2c_2 \end{bmatrix}$$
$$= \alpha \begin{bmatrix} (xp_1 + yp_2)a_1 & (xq_1 + yq_2)b_1 \\ (up_1 + vp_2)a_2 & (uq_1 + vq_2)b_2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$
$$= \alpha \left(\begin{bmatrix} (xp_1 + yp_2) & (xq_1 + yq_2) \\ (up_1 + vp_2) & (uq_1 + vq_2) \end{bmatrix} \otimes \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \right) \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$
$$= \alpha \left(\left(\begin{bmatrix} x & y \\ u & v \end{bmatrix} \begin{bmatrix} p_1 & q_1 \\ p_2 & q_2 \end{bmatrix} \right) \otimes \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \right) \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

Now let us analyse the marginal probability of node *S*, $P_{loopy}(S \mid w)$, computed by the loopy belief propagation by the end of the first iteration. The following are the messages node *S* will receive by the end of the first iteration:

- $\pi_C(S) = \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$ from node *C*, and
- $\lambda_W(S) = \begin{bmatrix} x & y \\ u & v \end{bmatrix} \begin{bmatrix} \pi(r_1) \\ \pi(r_2) \end{bmatrix}$ from node *R* through node *W*, where $\begin{bmatrix} \pi(r_1) \\ \pi(r_2) \end{bmatrix} = \begin{bmatrix} p_1 & q_1 \\ p_2 & q_2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$ is due to the message from node *C*.

Thus, we have

$$P_{loopy}(S \mid w) = \alpha \lambda_W(S) \otimes \pi_C(S)$$
$$= \alpha \left(\begin{bmatrix} x & y \\ u & v \end{bmatrix} \begin{bmatrix} p_1 & q_1 \\ p_2 & q_2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \right) \otimes \left(\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \right).$$

From the above expression, it becomes clear that deriving the error term for $P_{loopy}(S \mid w)$ will be a tedious process, which is further complicated by the circulating of evidence within the loop that leads to varying messages of $\pi_C(S)$ and $\lambda_W(S)$ to be received by node *S* in subsequent iterations. Nevertheless, there are two interesting observations we can make by comparing the two expressions. Firstly, we see that even at the first iteration, the loopy belief propagation has double counted the evidence due to the prior probability of node *C* since there are two occurrences of the term $\begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$ in $P_{loopy}(S \mid w)$. Secondly, in computing the true marginal $P_{true}(S \mid w)$, the entry-wise multiplication takes place in the 2D matrix space of the CPT parameters $\begin{bmatrix} x & y \\ u & v \end{bmatrix}, \begin{bmatrix} p_1 & q_1 \\ p_2 & q_2 \end{bmatrix}$, and $\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}$, before the final reduction to a binary vector by multiplying with the vector $\begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$. In contrast, the computation of $P_{loopy}(S \mid w)$ first performs two reductions of the CPT parameters into two binary vectors ($\lambda_W(S)$ and $\pi_C(S)$) both by multiplying with the vector $\begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$, and the entry-wise multiplication takes place in the reduced binary vector space. The second observation hints at the possibility of accounting for the error term by reconstructing the binary vectors to their respective 2D matrix space, which we will describe later in this section. Another approach for further analysis is to make use of the results established in Weiss [68] for the properties of loopy belief propagation in a pair-wise binary Markov random field. The way to adopt the established results is by converting our virtual hybrid network (Bayesian network + factor graph), shown in Figure 6.6, into a pair-wise binary Markov random field, which is a straight forward process. Then following the same arguments in Weiss [68], we can compute for each node a single matrix that summarises all the message passing operations around the loop starting and ending at that node. For

example, for node *S*, we can compute the summary matrix for the anticlockwise direction of message passing as

$$M_{S\to C\to R\to S} = \begin{bmatrix} x & y \\ u & v \end{bmatrix}\begin{bmatrix} p_1 & q_1 \\ p_2 & q_2 \end{bmatrix}\begin{bmatrix} c_1 & 0 \\ 0 & c_2 \end{bmatrix}\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}$$

and similarly

$$M_{S\to R\to C\to S} = \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \end{bmatrix}\begin{bmatrix} c_1 & 0 \\ 0 & c_2 \end{bmatrix}\begin{bmatrix} p_1 & p_2 \\ q_1 & q_2 \end{bmatrix}\begin{bmatrix} x & u \\ y & v \end{bmatrix}$$

for the clockwise direction. Note that $M_{S\to C\to R\to S}$ is actually the transpose of $M_{S\to R\to C\to S}$. Then, according to Weiss [68], if all the elements of $M_{S\to C\to R\to S}$ are non-zero, we have the following results:

1. The message $\lambda_W(S)$ that node *R* sends to node *S* through node *W* will converge to the principal eigenvector of $M_{S\to C\to R\to S}$.
2. The message $\pi_C(S)$ that node *C* sends to node *S* will converge to the principal eigenvector of $M_{S\to R\to C\to S}$.
3. The convergence rate of the messages is governed by the ratio λ_1 / λ_2 where λ_1 and λ_2 are the largest and second-largest eigenvalue of $M_{S\to C\to R\to S}$ respectively.
4. The diagonal elements of $M_{S\to C\to R\to S}$ give the correct posteriors, that is $P_{true}(s_i \mid w) = \alpha M_{S\to C\to R\to S}(i,i)$.

With results (1) and (2) we can now work on the converged messages that node *S* will receive to compute the converged value of $P_{loopy}(S \mid w)$, and result (4) yields an alternative form to $P_{true}(S \mid w)$, which could be useful for deriving the error term.

Based on the insights gained from our error analysis presented in previous sections, we see that if we control the order in which the nodes are updated, we can still compute the true marginal $P_{true}(w)$ for node *W*. The idea here is to update the nodes according to the clique order of its join tree. For example, if we update node *C*, *S*, and *R* through belief propagation, we will then be able to recover

$$\pi(S,R)=\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}\begin{bmatrix} c_1 & 0 \\ 0 & c_2 \end{bmatrix}\begin{bmatrix} p_1 & p_2 \\ q_1 & q_2 \end{bmatrix}$$

at node W from the messages $\pi_W(S)$ and $\pi_W(R)$ it receives from node S and node R respectively. We can then compute the exact marginal probability of node W as follows

$$\begin{aligned} P_{true}(W) &= \alpha\lambda(w)\pi(w) \\ &= \alpha\begin{bmatrix} \lambda(w_1) \\ \vdots \\ \lambda(w_n) \end{bmatrix}\otimes\begin{bmatrix} \pi(w_1) \\ \vdots \\ \pi(w_n) \end{bmatrix} \\ &= \alpha\begin{bmatrix} \lambda(w_1) \\ \vdots \\ \lambda(w_n) \end{bmatrix}\otimes\begin{bmatrix} P(w_1\mid s_1,r_1) & P(w_1\mid s_1,r_2) & P(w_1\mid s_2,r_1) & P(w_1\mid s_2,r_2) \\ \vdots & \vdots & \vdots & \vdots \\ P(w_n\mid s_1,r_1) & P(w_n\mid s_1,r_2) & P(w_n\mid s_2,r_1) & P(w_n\mid s_2,r_2) \end{bmatrix}\times \\ &\qquad \left\{\begin{bmatrix} \pi_w(s_1)\pi_w(r_1) \\ \pi_w(s_1)\pi_w(r_2) \\ \pi_w(s_2)\pi_w(r_1) \\ \pi_w(s_2)\pi_w(r_2) \end{bmatrix}+\begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}(a_1-b_1)(p_1-q_1)c_1c_2\right\} \end{aligned}$$

In addition, since we have now recovered the term $\pi(S,R)$, we will also be able to compute $P_{true}(s,r\mid w)$, followed by $P_{true}(c\mid w)$. Also, $P_{true}(s\mid w)$ and $P_{true}(r\mid w)$ are obtained by marginalising $P_{true}(s,r\mid w)$.

By controlling the belief propagation in the order of the join tree cliques, we are trying to prevent the erroneous compounding of evidence in the loop. In a way, we can think of it as mimicking the join tree algorithm's 2-Pass process, but instead of maintaining a larger potential or belief function defined over all the node variables within a clique, we maintain individual belief functions updated through loopy belief propagation based on the original network structure. The main problem with this method is that it does not produce the required potential or belief function of the separator set that is required when moving from one clique to another. As analysed in section 6.1, the join tree algorithm circumvents the need to explicitly compute $P(S,R)$ from $P(S)$ and $P(R)$, at the expense of higher computational costs and additional storage by maintaining the joint probability(belief) of cluster nodes $\{C,S,R\}$ and $\{C,S,W\}$, and then marginalizing to obtain the beliefs of $\{S,R\}$ for the separator set. Hence, if we are able to recover the

belief function of the separator set exactly from individual belief functions of nodes within it, then we will have a new exact inference algorithm based on loopy belief propagation. However, it is intuitive that such recovery process is likely to be computationally intensive in a generic Bayesian network, especially when the number of nodes in the separator set is big. Thus, a more practical alternative will be to consider a more efficient recovery computation that only approximates the belief function of the separator set. Nevertheless, we can extend algorithm 6.3 to also perform exact inference using belief propagation with loop correction even the case where there is evidence entering the sink node, which we will present next. We will provide a more in-depth discussion on the effects of using the 2-Pass control on loopy belief propagation as approximate inference in Chapter 7.

Algorithm 6.4 2-Pass Belief Propagation with Loop Correction for Single Loop Network without Instantiations

Input: *A Bayesian network G (assume n = m) as shown in Figure 6.3 with evidence* $\lambda(W)$ *at the sink node*

Output: *Marginal probability of all nodes in G*

1. *Perform the Downward pass of single node belief propagation as in algorithm 6.3.*
2. *For i = 1 to n,*

   Compute $\pi(S_i, R_i) = \pi(S_i)\pi(R_i) + Error(S_i, R_i)$

   End {For}
3. *Upward pass:*

   Compute $\lambda(S_n, R_n)$ *by multiplying* $\lambda(W)$ *into CPT* $P(W \mid S_n, R_n)$

   For i = n-1 to 1,

   Compute $\lambda(S_i, R_i)$ *by multiplying CPTs* $P(S_{i+1} \mid S_i)$ *and* $P(R_{i+1} \mid R_i)$ *into* $\lambda(S_{i+1}, R_{i+1})$

   End {For}
4. *Compute marginal probability:*

   $P(C) = \pi(C) \otimes \lambda(C)$, *where* $\lambda(C) = (P(S_1 \mid C) \times P(R_1 \mid C))\lambda(S_1, R_1)$

   For i = 1 to n,

$$P(S_i, R_i) = \pi(S_i, R_i) \otimes \lambda(S_i, R_i)$$

Marginalise $P(S_i, R_i)$ *to yield* $P(S_i)$ *and* $P(R_i)$

End {For}

$$P(W) = (P(W \mid S_n, R_m) \times \pi(S_n, R_m)) \otimes \lambda(W)$$

5. *Output marginal probability of all nodes in G.*

The assumption of $n = m$ in the algorithm 6.4 is just for easier presentation. Without loss of generality, suppose $m > n$, then all we need to do is to process the additional terms $\lambda(S_n, R_{n+1})$, $\lambda(S_n, R_{n+2})$, ..., $\lambda(S_n, R_m)$, using the CPTs $P(R_{n+1} \mid R_n)$, $P(R_{n+2} \mid R_{n+1})$,…, $P(R_m \mid R_{m-1})$ respectively.

Algorithm 6.4 is designed based on the Shafer-Shenoy [47] join tree architecture where at each separator set the architecture maintains 2 separate potential functions to store the belief update messages passed in each of the inward and outward passes. In this way, it removes the need to discount for double counted evidence through division. However, algorithm 6.4 deviates from the Shafer-Shenoy algorithm in two ways. Firstly, for the downward pass, the belief propagation is carried out on single nodes as opposed to message passing from clique to clique through marginalisation onto the clique separator. Secondly, while the upward pass in step 3 of algorithm 6.4 produces identical results on the clique separator sets, i.e. $\lambda(S_i, R_i)$,as those in the Shafer-Shenoy algorithm, it only needs to maintain a storage for 2 variables, while Shafer-Shenoy algorithm, as well as the HUGIN and LS architectures, requires maintaining the 3-variable clique potentials. The correctness of algorithm 6.4 to perform exact inference on the single loop is evident from the fact that at both of the passes, it produces the same $\pi(S_i, R_i)$ *and* $\lambda(S_i, R_i)$ messages as the Shafer-Shenoy algorithm. In this case, the savings in computing time as compared to join tree algorithm is again mainly due to the downward pass as is the case for algorithm 6.3.

We use the next example to illustrate an instance of running algorithm 6.4.

Example 6.3.4. Let us walkthrough algorithm 6.4 on the network shown in Figure 6.4(a). The first part of the algorithm it will compute using the downward pass with error correction to produce: $\pi(B,C)$, $\pi(D,E)$, $\pi(F,G)$. Note that we need not produce π values for the other separator sets of the join tree shown in Figure 6.3(b), which are $\{C,D\}$ and $\{E,F\}$. For the second part of the algorithm, in the upward pass, the following sequence of λ values will be produced:

- $\lambda(F,G)$ by multiplying $\lambda(H)$ into CPT $P(H \mid F,G)$
- $\lambda(D,E)$ by multiplying $\lambda(F,G)$ with $P(F \mid D)$ and $P(G \mid E)$
- $\lambda(B,C)$ by multiplying $\lambda(D,E)$ with $P(D \mid B)$ and $P(E \mid C)$

The marginal probability can then be computed using both λ and π values using step 4.

Note that similar to algorithm 6.3, algorithm 6.4 can also be extended to handle multiple parallel loops like the one shown in Figure 6.5. The downward pass extension is as discussed in previous section. The modification to algorithm 6.4 in this case is simply the need to handle a bigger λ message storage for the increased separator set size.

6.5 Discussion

So far we have presented analysis on the error due to evidence entering the loop through the source and sink nodes of the loop. Now we will expand our error analysis further to consider the two possible cases when evidence enters the loop through intermediate nodes (e.g., *S* and *R*):

1. evidence from parent (π message)
2. evidence from child (λ message)

Case of evidence through π message:

Consider the Bayesian network shown in Figure 6.8. In this case, external evidence from node *D* can enter the loop through node *S* as π messages. There are two possible

scenarios: either node *D* is instantiated or uninstantiated. If node *D* has been instantiated, say to state d_i, then the outcome is that the error term will be using the terms from the corresponding CPT, which is $P(S \mid C, D = d_i)$. For example, suppose node *D* has been instantiated to state d_2, then the error term becomes

$$P_{true}(S, R) - P_{indep}(S, R) = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} (a_{21} - b_{21})(p_1 - q_1) c_1 c_2$$

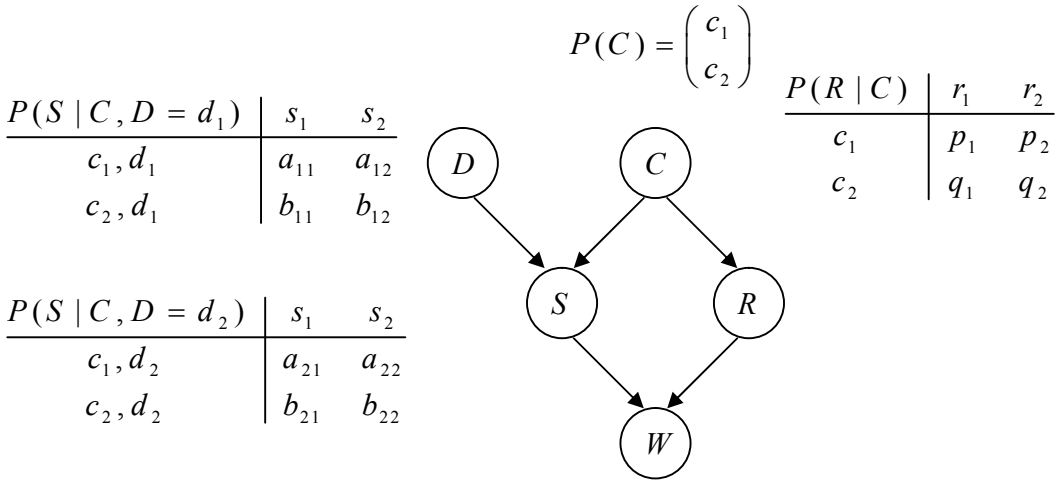


Figure 6.8: π evidence entering loop through intermediate node.

On the other hand, if node *D* has not been instantiated, then the π message it sends to node *S* contains evidence due to its unconditional probability. In this case, it is not necessary to derive the error term from scratch as we can apply a little neat trick by grouping node *C* and node *D* into a single source node. Let us denote this clustered source node as node *E*, and the updated CPT entries become

$P(S \mid E)$	s_1	s_2
c_1, d_1	a_{11}	a_{12}
c_2, d_1	b_{11}	b_{12}
c_1, d_2	a_{21}	a_{22}
c_2, d_2	b_{21}	b_{22}

and

$P(R \mid E)$	r_1	r_2
c_1, d_1	p_1	p_2
c_2, d_1	q_1	q_2
c_1, d_2	p_1	p_2
c_2, d_2	q_1	q_2

Note that we have to create two redundant entries in $P(R|E)$. Now we can derive the error term by applying the error equation (6.1) as before. Note that the forming of the node E is only for theoretical formulation of the error term. In practice, we can continue to perform the loopy belief propagation on each individual node C and node D, followed by the loopy correction step at node W.

Case of evidence through λ message:

Let us consider the case of λ evidence entering the loop using the Bayesian network shown in Figure 6.9.

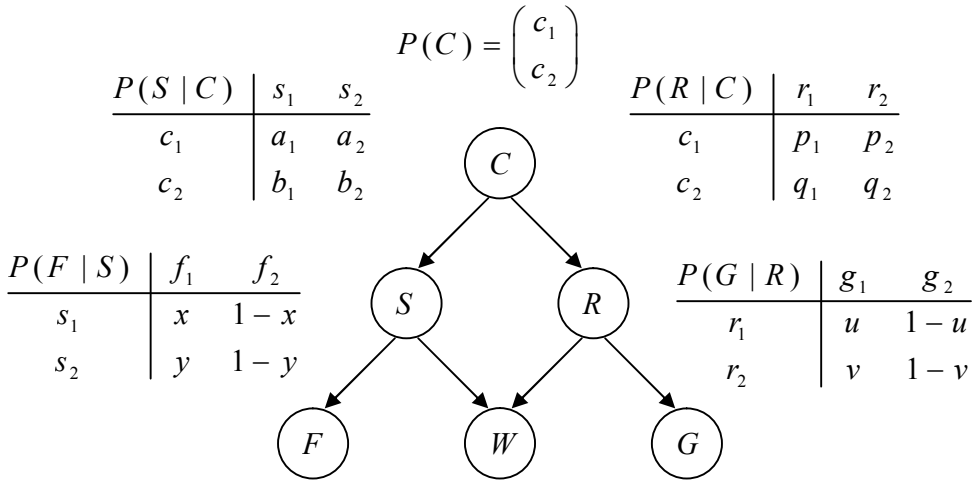


Figure 6.9: λ evidence entering loop through intermediate node.

Without loss of generality, let us assume that node F is instantiated to state f_1 where

$$P_{true}(s_i, r_j, F = f_1) = \sum_c P(s_i|c)P(f_1|s_i)P(r_j|c)P(c) .$$

Unlike in all our previous cases, the normalising constant (or partition function) in this case is no longer 1. For instance,

$$P_{true}(S, R, F = f_1) \propto \begin{bmatrix} x & 0 \\ 0 & y \end{bmatrix} \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \begin{bmatrix} c_1 & 0 \\ 0 & c_2 \end{bmatrix} \begin{bmatrix} p_1 & p_2 \\ q_1 & q_2 \end{bmatrix}$$

and

$$P_{true}(S,R \mid F = f_1) = \frac{1}{P(f_1)}\begin{bmatrix} x & 0 \\ 0 & y \end{bmatrix}\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}\begin{bmatrix} c_1 & 0 \\ 0 & c_2 \end{bmatrix}\begin{bmatrix} p_1 & p_2 \\ q_1 & q_2 \end{bmatrix}$$

where

$$P(f_1) = (a_1 x + a_2 y)c_1 + (b_1 x + b_2 y)c_2$$

is the normalising factor.

Similarly, we have

$$P_{true}(S \mid F = f_1) = \frac{1}{P(f_1)}\begin{bmatrix} x & 0 \\ 0 & y \end{bmatrix}\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}\begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

and

$$P_{true}(R \mid F = f_1) = \frac{1}{P(f_1)}\begin{bmatrix} p_1 & q_1 \\ p_2 & q_2 \end{bmatrix}\left(\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \otimes \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}\right)$$

where $\otimes$ denotes entry-wise multiplication.

Multiplying $P_{true}(S)$ with transpose of $P_{true}(R)$ yields

$$\begin{aligned}
&P_{indep}(S,R \mid F = f_1) \\
&= \frac{1}{\left(P(f_1)\right)^2}\left(\begin{bmatrix} x & 0 \\ 0 & y \end{bmatrix}\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}\begin{bmatrix} c_1 \\ c_2 \end{bmatrix}\right)\left(\begin{bmatrix} p_1 & q_1 \\ p_2 & q_2 \end{bmatrix}\left(\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \otimes \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}\right)\right)^T \\
&= \frac{1}{\left(P(f_1)\right)^2}\begin{bmatrix} x & 0 \\ 0 & y \end{bmatrix}\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}\begin{bmatrix} c_1 \\ c_2 \end{bmatrix}\begin{bmatrix} (a_1 x + a_2 y)c_1 & (b_1 x + b_2 y)c_2 \end{bmatrix}\begin{bmatrix} p_1 & p_1 \\ q_1 & q_2 \end{bmatrix} \\
&= \frac{1}{\left(P(f_1)\right)^2}\begin{bmatrix} x & 0 \\ 0 & y \end{bmatrix}\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}\begin{bmatrix} (a_1 x + a_2 y)c_1^2 & (b_1 x + b_2 y)c_1 c_2 \\ (a_1 x + a_2 y)c_1 c_2 & (b_1 x + b_2 y)c_2^2 \end{bmatrix}\begin{bmatrix} p_1 & p_1 \\ q_1 & q_2 \end{bmatrix}
\end{aligned}$$

Now, the error term is given by

$$\begin{aligned}
&P_{true}(S,R \mid F = f_1) - P_{indep}(S,R \mid F = f_1) \\
&= \frac{1}{\left(P(f_1)\right)^2}\begin{bmatrix} x & 0 \\ 0 & y \end{bmatrix}\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}\Delta\begin{bmatrix} p_1 & p_1 \\ q_1 & q_2 \end{bmatrix}
\end{aligned}$$

were

$$\Delta = P(f_1)\begin{bmatrix} c_1 & 0 \\ 0 & c_2 \end{bmatrix} - \begin{bmatrix} (a_1 x + a_2 y)c_1^2 & (b_1 x + b_2 y)c_1 c_2 \\ (a_1 x + a_2 y)c_1 c_2 & (b_1 x + b_2 y)c_2^2 \end{bmatrix}.$$

Since we know that $P(f_1) = (a_1x + a_2y)c_1 + (b_1x + b_2y)c_2$, substituting it into Δ, gives

$$\begin{aligned}\Delta &= \left((a_1x+a_2y)c_1+(b_1x+b_2y)c_2\right)\begin{bmatrix} c_1 & 0 \\ 0 & c_2 \end{bmatrix} - \begin{bmatrix} (a_1x+a_2y)c_1^2 & (b_1x+b_2y)c_1c_2 \\ (a_1x+a_2y)c_1c_2 & (b_1x+b_2y)c_2^2 \end{bmatrix} \\ &= \begin{bmatrix} (b_1x+b_2y)c_1c_2 & -(b_1x+b_2y)c_1c_2 \\ -(a_1x+a_2y)c_1c_2 & (a_1x+a_2y)c_1c_2 \end{bmatrix} \\ &= \begin{bmatrix} b_1x+b_2y & 0 \\ 0 & a_1x+a_2y \end{bmatrix}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}c_1c_2\end{aligned}$$

Thus, we have

$$\begin{aligned}&P_{true}(S,R \mid F=f_1) - P_{indep}(S,R \mid F=f_1) \\ &= \frac{1}{\left(P(f_1)\right)^2}\begin{bmatrix} x & 0 \\ 0 & y \end{bmatrix}\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}\left(\begin{bmatrix} b_1x+b_2y & 0 \\ 0 & a_1x+a_2y \end{bmatrix}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}c_1c_2\right)\begin{bmatrix} p_1 & p_1 \\ q_1 & q_2 \end{bmatrix} \\ &= \frac{1}{\left(P(f_1)\right)^2}\begin{bmatrix} a_1x & b_1x \\ a_2y & b_2y \end{bmatrix}\begin{bmatrix} b_1x+b_2y & 0 \\ 0 & a_1x+a_2y \end{bmatrix}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}c_1c_2(p_1-q_1) \\ &= \frac{1}{\left(P(f_1)\right)^2}\begin{bmatrix} a_1x(b_1x+b_2y) & b_1x(a_1x+a_2y) \\ a_2y(b_1x+b_2y) & b_2y(a_1x+a_2y) \end{bmatrix}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}c_1c_2(p_1-q_1) \\ &= \frac{1}{\left(P(f_1)\right)^2}\begin{bmatrix} xy(a_1-b_1) & -xy(a_1-b_1) \\ -xy(a_1-b_1) & xy(a_1-b_1) \end{bmatrix}c_1c_2(p_1-q_1) \\ &= \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}(a_1-b_1)(p_1-q_1)c_1c_2\left(\frac{xy}{\left(P(f_1)\right)^2}\right)\end{aligned}$$

Recall that for the single loop with only nodes *C*, *S*, *R* and *W*, we have

$$P_{true}(S,R) - P_{indep}(S,R) = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}(a_1-b_1)(p_1-q_1)c_1c_2 .$$

By comparing the two error terms it is easy to see that the difference in this case is the additional term

$$\left(\frac{xy}{\left(P(f_1)\right)^2}\right)$$

due to instantiation of $F = f_1$. Hence, we can again apply algorithms 6.3 and 6.4 with the modified error term shown above to handle the case where evidence enters the loop

through its intermediate notes. More importantly, step 2 (downward pass) of algorithm 6.3 has to be modified slightly to allow evidence entering the intermediate node to be updated along the path between node S_n and node R_m through source node *C*. The modified step 2 will then be to start the belief propagation from node S_n to node R_m through source node *C* and back, or vice versa.

It turns out that if we go through the same derivation process as above, we can also show in the case where the instantiations are $F = f_1$ and $G = g_1$,

$$P_{true}(S, R \mid F = f_1, G = g_1) - P_{indep}(S, R \mid F = f_1, G = g_1)$$
$$= \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} (a_1 - b_1)(p_1 - q_1)c_1c_2 \times \left(\frac{xyuv}{\left(P(f_1, g_1)\right)^2} \right)$$

The same error analysis can also be extended to a bigger loop with more intermediate nodes by applying the same arguments as those in section 6.3.

The significance of this discussion is that we have shown that algorithms 6.3 and 6.4 are able to handle the different cases of evidence instantiations in a generic loop.

After the derivation of the error term, the next natural question to ask is if we can make use of them to compute a bound on the loopy error, or at least for the case of a single loop? It turns out that the bound is quite easy to compute for a single loop without instantiation. To answer the question, let us begin by looking at the simplest case presented in section 6.2. In the concluding remarks of section 6.2, we discussed the conditions on which the error term becomes zero. Since the error term is given by

$$Error(S, R) = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} (a_1 - b_1)(p_1 - q_1)c_1c_2$$

we obtain

$$\frac{\partial Error(S,R)}{\partial c_1} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} (a_1 - b_1)(p_1 - q_1)(1 - 2c_1)$$

and

$$\frac{\partial Error(S,R)}{\partial c_1} = 0$$
$$\Rightarrow (a_1 - b_1)(p_1 - q_1)(1 - 2c_1) = 0$$
$$\Rightarrow c_1 = 0.5$$

Therefore, we can conclude that

$$\begin{aligned} \left|Error(S,R)\right| &= \left|(a_1 - b_1)(p_1 - q_1)c_1(1 - c_1)\right| \\ &\le \left|(a_1 - b_1)(p_1 - q_1)\right| 0.25 \end{aligned}$$

for all possible values of the network parameters (a_1, b_1, p_1, q_1). Hence, this answers the question that we can indeed compute a bound on the error term. Following the same arguments, we will arrive at the following bound for the case of a larger binary loop presented in section 6.3, which has the following error term

$$Error(S,R) = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} c_1 c_2 \prod_{i=1}^{n} (a_1^i - b_1^i) \prod_{j=1}^{m} (p_1^j - q_1^j).$$

It is easy to see that $\left|Error(S,R)\right| \le \prod_{i=1}^{n} |(a_1^i - b_1^i)| \prod_{j=1}^{m} |(p_1^j - q_1^j)| 0.25$ for this case.

One way of making use of this error bound is to help decide in advance if it is worth to spend the additional computational cost on loop correction error bound value is small. In other words, if we pre-compute the term $\prod_{i=1}^{n} |(a_1^i - b_1^i)| \prod_{j=1}^{m} |(p_1^j - q_1^j)|$ and it is less than a certain threshold, then we might not want to invoke the loop correction step.

While algorithms 6.3 and 6.4 are proposed to address the issue of loop correction for a single generic loop, their application is definitely beyond a network with a single loop. As we have seen in previous examples, it can easily be reused for networks with multiple single loops and multiple parallel loops sharing the same sink and source nodes. Furthermore, the ability to perform exact inference for a single loop with lower computational cost than join tree opens up the possibility to improve existing approximate inference algorithm that relies on tree-based approximation to also incorporate loops and thus account for more dependencies in the original network. Similarly, for learning efficient network from data for fast inference, the choice of network can now also include those with loops that algorithms 6.3 and 6.4 can handle.

Chapter 7

2-Pass Loopy Belief Propagation for Fast Approximate Inference

The join tree (also known as junction tree) algorithm is often the first choice solution for exact inference in a Bayesian Network, due to its solid underlying theoretical foundation, as well as many years of research work poured into it by pioneering researchers in the area of directed graphical models, which have produced a well understood algorithm with many practical implementation enhancements. The join tree algorithm works by transforming the original Bayesian network into a secondary graph structure called the clique tree where message passing is carried out on this secondary structure involving cliques of variables. It has been shown that the join tree algorithm is indeed a special case of belief propagation [66].

The computational cost of join tree algorithm grows exponentially with the size of the cliques formed. In fact, it has been proven that the complexity is exponential in its tree-width. For a triangulated graph, the tree-width is the size of the largest clique minus one. Hence, it is important that the join tree constructed has small clique sizes, but again, the problem of finding a join tree with minimal clique size is NP-hard. Thus for more complex problems, we seek solutions from approximate inference algorithms.

A widely used alternative to the join tree algorithm is an approximation algorithm known as Loopy Belief Propagation (LBP). LBP is Pearl's message passing algorithm (designed to perform exact inference on singly-connected networks) applied iteratively until convergence or when some terminating conditions are met. Hence, LBP computes the exact marginal posteriors if the network is singly-connected (i.e. without directed loops), and it has also been shown empirically to provide good quality approximations to multiply-connected networks (i.e. those that contain directed loops).

The join tree algorithm, while not being an optimisation algorithm, has been proven to converge to the Gibbs free energy associated with the Bayesian Network, on the other hand the loopy belief propagation algorithm will either converge to local minima of the Bethe free energy (which is an approximation to the Gibbs free energy) or oscillates among the local minima.

Prompted by the new insights gained on the relationships between belief propagation methods and their convergence properties with respect to the associated free energies defined by the network structure and parameters, many researchers have been working on improving the accuracy and efficiency of approximate algorithms based on the free energy formulations. For example, the TRW-BP (Tree Reweighted Belief Propagation) [52] method tries to approximate the original BN as a combination of weighted spanning trees. It was originally inspired by the problem of trying to maximize the lower bound of the free energy. The problem with LBP is that the Bethe free energy, while approximating the Gibbs free energy, does not bound it, hence there is no guarantee on the performance in generic networks, and in some cases it can perform badly, and this motivated research on TRW-BP approaches. Another popular approximation method is known as the Generalized Belief Propagation (GBP), which aims to work on more generalized free energy approximation beyond Bethe free energy, for example the Kikuchi free energy.

One of the biggest problems with the above mentioned state of art approximate inference algorithms is that they are iterative methods where convergence may be slow and is not

guaranteed (unlike the join tree algorithm which terminates in 2 passes). In some cases, usually for small to mid-size problems, the join tree algorithm might outperform the other iterative methods due to fewer overheads on checking for convergence.

In this chapter, we introduce a new approximate inference algorithm that combines the merits of the join tree algorithm (convergence governed by a 2-Pass control) and iterative belief propagation methods (computations within a single iteration is often very fast). We shall name our method 2P-LBP (2-Pass loopy belief propagation). The algorithm of 2P-LBP is given next.

Algorithm 2P-LBP

Input: *A Bayesian network G*

Output: *Marginal probability of all nodes in G*

1. *Construct a join tree of the Bayesian network G and produce the clique ordering* $\{Clq_1, Clq_2, ..., Clq_m\}$, *where* Clq_1 *is the root clique.*
2. *Inward pass:*

   For i = m to 1,

   Perform standard belief propagation on the sub-network of G defined by nodes within Clq_i

   End {For}
3. *Outward pass:*

   For i = 1 to m,

   Perform standard belief propagation on the sub-network of G defined by nodes within Clq_i

   End {For}
4. *Output marginal probability of all nodes in G.*

The 2-Pass loopy belief propagation was motivated by observations we made while studying the working of join tree algorithm on loopy graphs and the error analysis of

loopy belief propagation as presented in Chapter 6. In a way, we can view it as a convergent modification to the standard loopy belief propagation guided by the 2-Pass order of a join tree, which will minimize the over-counting of evidence, particularly in cases when leaf nodes have been instantiated. On the other hand, we can also view it as a modified (approximate) join tree algorithm that, instead of maintaining a big potential function for all nodes in the clique so that it can marginalise to the separator set, only maintains individual belief functions for each node in the clique. The default operations of the belief propagation will then compute the joint potential function by treating the nodes in the separator set as independent. In this way, we will then have the option to recover the required joint potential function for the separator set by error correction if applicable. In cases where the sub-network defined by nodes within Clq_i (in Step 2 and Step 3 of 2P-LBP) contains loops, we have a few options to tackle the problem. For fast approximate inference, we propose using tree approximation on the sub-network structure by performing a breadth-first search on an arbitrarily selected root node within the clique. For better accuracy, we can apply loopy belief propagation with loop correction. We will provide a more detailed discussion on application of error correction in 2P-LBP in section 7.4.

It is clear, by virtue of updating the beliefs following the clique order of a join tree, that if the input Bayesian network is singly-connected or when the loop-cutset nodes have been instantiated, the algorithm 2P-LBP will produce exact inference marginal probability. This is because the single variable marginal probability obtained by standard belief propagation on the sub-network defined on the nodes in each clique Clq_i will be the same as that obtained from marginalising from the clique potential function $\phi(Clq_i)$. 2P-LBP has the same offline computational cost as the join tree algorithm in terms of creating the clique tree. For the runtime cost of 2P-LBP, if the tree approximation scheme is used, then it is equal to the sum of the times required to perform standard belief propagation on *m* singly-connected sub-networks.

The 2P-LBP algorithm bears some resemblance to the "Inference with Approximate Messages" method described in the book by Koller and Friedman [69] (Chapter 11,

section 11.4.3). The approximate messages here refer to the messages computed through a process that has less computational cost, which also introduces error as compared to the junction tree algorithm, hence the terms "approximate messages". The basic concept is to approximate messages in the join tree propagation whereby instead of representing messages in the join trees as large factors, they use more compact representations of approximate factors represented in factored forms.

The main difference between the junction tree algorithm and inference by approximate messages lies in the way the messages between cliques are computed. Instead of computing the clique messages by marginalising over the larger product of factors, it computes their M-projection. The M-projection (moment projection) of P onto Q is given by the distribution

$$Q^M = \arg\min_{Q \in \mathbb{Q}} D(P \,||\, Q)$$

where P is the true distribution we want to approximate with the distribution Q from a convex set of distributions represented by $\mathbb{Q}$, and D is the Kullback Leibler distance measure discussed previously in Section 4.2. To illustrate the differences between the two methods, let us consider the case where there is a message to be computed between two cliques, say Clq_j and Clq_k with separator set S, within a certain join tree. Then, the standard junction tree algorithm, as presented in Section 5.2, will compute the message to be sent from Clq_j to Clq_k as follows:

$$\phi(Clq_j)^{\downarrow S} = \sum_{Clq_j \setminus S} \phi(Clq_j)$$

where $\phi(Clq_j)$ has already absorbed all messages from its neighbouring cliques excluding that from Clq_k. On the other hand, the inference with approximate messages method computes the message as follows:

$$\phi(Clq_j)^{\downarrow S} = \text{M-project-distr}_{j,k}(\phi(Clq_j))$$

where the term M-project-distr denotes the combined operation of marginalising variables in the set $Clq_j \setminus S$ and computing the M-projection onto the set of distributions according to some factorisation.

Now that we have contrasted the inference with approximate messages method with the junction tree algorithm, it is clear that the 2P-LBP algorithm shares the same commonality that it is also guided by the clique order of the join tree. In fact, as mentioned earlier, it was motivated to use the clique order to minimise the over-counting of evidence in loopy belief propagation. However, in contrast, the 2P-LBP algorithm does not attempt to approximate the clique messages. Its message passing scheme is based on loopy belief propagation among singleton variables. There is also no projection of the true distribution onto an approximate distribution involved in the message computations.

Since the discovery of the relationship between loopy belief propagation and its convergence properties with respect to the Bethe free energy formulation, there have been several attempts to also quantify the error made by LBP. Tatikonda [70] analysed the LBP error using mathematical tools from Gibbs measure theory on a computation tree called Bethe tree, which represents an unwrapping of the original graph with respect to the sum-product algorithm. The effect of N iterations of LBP at any singleton node S is equivalent to carrying out exact inference on the unwrapped tree-structured network with depth N and node S as its root (Tatikonda and Jordan [48]). The Dobrushin's theorem was used to derive a bound on the LBP marginals' errors for the special case of graphs with large girths. The girth of a graph is the number of edges in the shortest cycle in the graph. Graphs with large girths are often studied in low density parity check (LDPC) codes because they correspond to locally tree-like structures of the graphs found in the codes. Taga and Mase [71] also made use of the Dobrushin's theorem to derive another bound on the error of LBP marginals. The costs of computing both error bounds are exponential in the size of the Markov blanket of the singleton node. Ihler [72] introduced a dynamic-range measure to derive an error bound using self-avoiding-walk (SAW) trees. The SAW tree of a graph is a sub-tree of the Bethe tree and it contains all walks that do not backtrack. It corresponds to unwrapping of the original graph up to the point at which each path forms a loop. While the computational cost of computing this error bound is linear in the size of the network, it is only formulated for the special case of pair-wise

Markov random fields. Shi et al. [53] derived a tighter error bound to that of Ihler [72] by extending the derivation with an improved non-uniform distance bound.

Mooij and Kappen [73] also provided an error bound on single-variable marginal distributions, which is related to that of Ihler's [72]. They proposed a "Box Propagation (BoxProp)" algorithm to compute the error bound by propagating local bounds over a general sub-tree of the original graph instead of the SAW tree. The BoxProp algorithm first chooses a particular sub-tree rooted in the singleton variable of interest, and then it propagates messages containing the bounding boxes over this sub-tree, from leaf nodes towards the root node. The updating equations resemble those of LBP and the final "belief" at the root node is a box that bounds the exact marginal of the root node. The choice of sub-tree is arbitrary, which will lead to different bounds being computed. An interesting corollary provided in the paper is that the bounds on the exact marginal computed by BoxProp are also bounds for the approximate LBP marginal of the root node. This is because any sub-tree of the original network is also a sub-tree of the computation tree for any node selected. This result is of interest to us because we can use the same argument to conclude that 2P-LBP is also bounded by the bounds produced by the BoxProp algorithm, by choosing the sub-tree to contain the singleton nodes found in the clique that it belongs to. Unfortunately, while the computational cost of computing the error bound using BoxProp is not exponential in the sizes of a node's Markov Blanket it is still exponential in the number of possible values of the variables in a factor. More importantly, in the concluding section of [73], the authors highlighted that although their derived error bounds are a step forward in quantifying the error of LBP, the actual error made by LBP is often at least one magnitude lower than the tightness of these bounds. In general, the analysis of the error produced by variants of LBP is complicated by the fact that the iterative process might get trapped in a local minima of the Bethe free energy, or not converge at all.

From a historical perspective, the standard loopy belief propagation algorithm was used in the early 1990s without strong theoretical insights as to why it worked reasonably well in practice and most analysis on its convergent properties and errors were conducted

empirically. It was not until early 2000s when the link between loopy belief propagation and minimization of Bethe free energy was established that there have been more theoretical foundations on which error analysis could be based. The point is that in the 90s the usefulness of standard LBP or its significance was not ignored due to the lack of theoretical foundation but on the contrary it was quite well received due to its practical performances and empirical test results. Therefore it is not surprising to see that many researchers working on novel approximate inference algorithms have devoted much effort to demonstrate the potentials of their algorithms at least empirically through experiments on generated random networks, toy problems, or open source benchmark datasets. In the next section, we shall demonstrate the potential of the 2P-LBP algorithm through some empirical studies.

7.1 Empirical evaluations of the 2-Pass loopy belief propagation

We will make use of a well known Bayesian network example, called the "Asia network" [44], to illustrate how we can carry out a 2-Pass loopy belief propagation, and demonstrate its potential as an alternative method of approximate inference. After which, we will provide further empirical results on comparing the algorithm to loopy belief propagation using standard benchmark networks. The Asia network shown in Figure 7.1 is based on a fictitious medical diagnostic model and we use the same CPT parameters presented in [44]. To apply the 2-Pass control to the loopy belief propagation, we have to first construct its join tree. Figure 7.2 shows the join tree that we have worked out. Note that we have designated the clique $\{B,W,D\}$ as the root node of the join tree.

In our proposed 2-Pass loopy belief propagation, we control the message passing schedule following the inward and outward passes of a join tree. Here is the order of the 2-Pass loopy belief propagation's inward pass based on the join tree shown in Figure 7.2:

1. At clique $\{A,T\}$: update node A and node T through message passing along the arc $A \rightarrow T$.
2. At clique $\{W,X\}$: update node W and node X through message passing along the arc $W \rightarrow X$.
3. At clique $\{S,L,B\}$: update node *S,* node L and node B through message passing on the structure $L \leftarrow S \rightarrow B$.
4. At clique $\{T,L,W\}$: update node *T,* node L and node W through message passing on the structure $T \rightarrow W \leftarrow L$.
5. At clique $\{L,B,W\}$: no update is carried out in this clique since node L and node W are already updated in previous clique and node B is not directly connected to either node L or node W.
6. At clique $\{B,W,D\}$: update node *B,* node W and node D through message passing on the structure $W \rightarrow D \leftarrow B$.

The order of the outward pass is just the reverse of the inward pass.

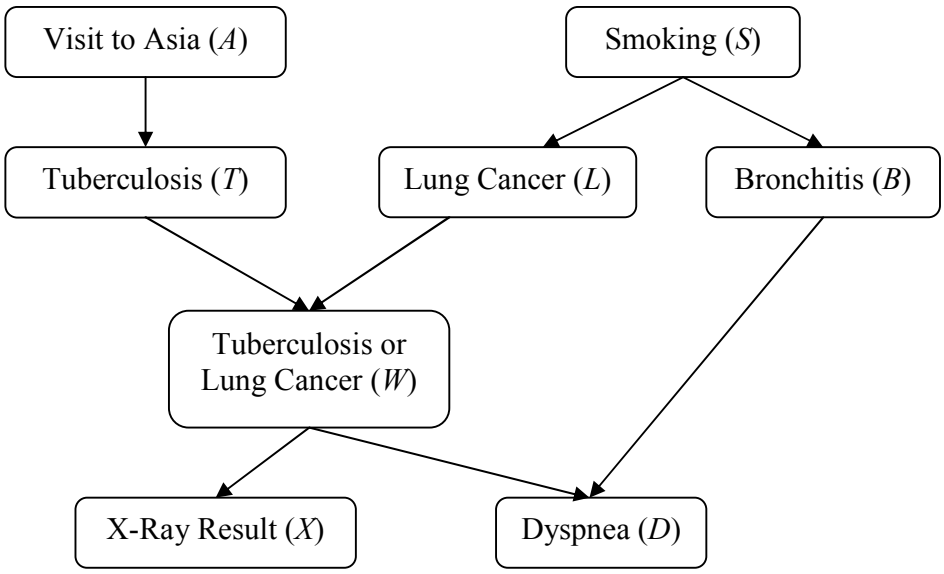


Figure 7.1: The Asia network.

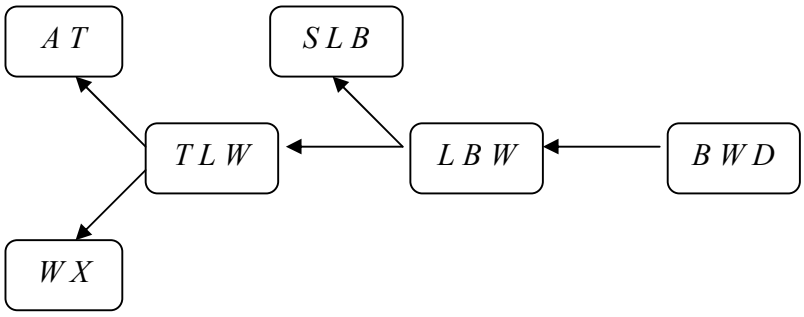


Figure 7.2: A join tree for the Asia network.

Instantiation: $D = d_1$					
Node	**Join Tree**	**Loopy BP**	**2P-LBP**	$B \rightarrow D$ **removed**	$S \rightarrow B$ **removed**
A	0.0103	0.0103 **(0.0)**	0.0103 **(0.0)**	0.0100 (-0.0003)	0.0102 (-0.0001)
T	0.0188	0.0184 (-0.0004)	0.0187 **(-0.0001)**	0.0160 (-0.0028)	0.0158 (-0.0030)
S	0.6340	0.6297 (-0.0043)	0.6318 **(-0.0022)**	0.5030 (-0.1310)	0.5120 (-0.1220)
L	0.1028	0.1153 (+0.0125)	0.0989 **(-0.0039)**	0.0614 (-0.0414)	0.0836 (-0.0192)
W	0.1205	0.1327 (+0.0122)	0.1166 **(-0.0039)**	0.0723 (-0.0482)	0.0985 (-0.0220)
X	0.1621	0.1734 (+0.0113)	0.1584 **(-0.0037)**	0.1170 (-0.0451)	0.1420 (-0.0201)
B	0.8340	0.8234 (-0.0106)	0.8261 **(-0.0079)**	0.4510 (-0.3830)	0.8970 (+0.0630)

Table 7.1: Experimental results for the case with instantiation $D = d_1$.

Instantiation: $D = d_1$ and $X = x_1$					
Node	**Join Tree**	**Loopy BP**	**2P-LBP**	$B \rightarrow D$ **removed**	$S \rightarrow B$ **removed**
A	0.0140	0.0137 (-0.0003)	0.0138 **(-0.0002)**	0.0133 (-0.0007)	0.0138 **(-0.0002)**
T	0.1139	0.1077 (-0.0062)	0.1100 **(-0.0039)**	0.0970 (-0.0169)	0.1090 (-0.0049)
S	0.7856	0.7693 **(-0.0163)**	0.7652 (-0.0204)	0.6980 (-0.0876)	0.7270 (-0.0586)
L	0.6213	0.6139 **(-0.0074)**	0.5280 (-0.0933)	0.5130 (-0.1083)	0.5780 (-0.0433)
W	0.7287	0.7159 **(-0.0128)**	0.6860 (-0.0427)	0.6050 (-0.1237)	0.6820 (-0.0467)
B	0.6819	0.6715 **(-0.0104)**	0.7289 (+0.0470)	0.5090 (-0.1729)	0.7430 (+0.0611)

Table 7.2: Experimental results for the case with instantiation $D = d_1$ and $X = x_1$.

To evaluate the performance of our proposed 2-Pass loopy belief propagation, we have conducted two experiments in Matlab using the Bayes Net Toolbox written by Kevin Murphy [74]. In the first experiment, we instantiated $D = d_1$, and carried out the join tree algorithm, loopy belief propagation, and 2P-LBP. In addition, we also experimented with the tree-approximation method that removes arcs from the original network to yield a singly connected network. In our experiment, we considered two separate cases of removing the arcs $B \rightarrow D$ and $S \rightarrow B$ respectively. In the second experiment, we instantiated $D = d_1$ and $X = x_1$. The results of both experiments are shown in Table 7.1 and Table 7.2. The values in each table represent the marginal posterior probability of the first state of each node, computed by the respective inference methods grouped by columns. For example, for the join tree algorithm we have $P(A = a_1 \mid D = d_1) = 0.0130$ and $P(A = a_1 \mid D = d_1, X = x_1) = 0.0140$. The performances of the various inference methods are evaluated by comparing their marginal posterior probabilities to the exact ones computed by the join tree algorithm. The difference is given in the tables in brackets. For each node (row), the inference method that produced the lowest absolute error is highlighted in bold. The standard loopy belief propagation successfully converged in both experiments. From Table 7.1, we see that in the first experiment, the 2-Pass loopy belief propagation came in as a clear winner by producing the lowest error for all nodes. This is somewhat unexpected as the standard loopy belief propagation took 9 iterations to converge, while the 2-Pass loopy belief propagation took only 1 iteration. Nevertheless, the magnitudes of errors between the two methods are still comparable. In contrast, the arc removal methods produced relatively larger errors. In particular, for the case where arc $B \rightarrow D$ has been removed, it caused a large error of 0.3830 for node *B*. In the second experiment, the 2-Pass loopy belief propagation's performance is not as good as in the first experiment. Nevertheless, it still managed to produce lowest errors for node *A* and node *T*. Its worst performance is at node *L*, at which both arc removal methods also performed poorly as compared to loopy belief propagation. In this case, the loopy belief propagation took 10 iterations to converge.

The results we have presented in this section provide a good motivation for proposing the 2-Pass loopy belief propagation as an alternative to standard loopy belief propagation. At the baseline of performance, both methods will produce the same exact inference results when the loop-cutset nodes have been instantiated and the same approximate inference results when none of the sink nodes of the loops have been instantiated. If we apply results from our loopy error analysis presented in chapter 6 and incorporate the loop corrections with the 2-Pass control, we can further improve its inference accuracy. Furthermore, in cases with simple disconnected loops, the 2-Pass loopy belief propagation with loop corrections can actually perform an exact inference. We will further expand on this point in the next section. Besides the possibility of producing more accurate inference results, the 2-Pass loopy belief propagation also has a few other advantages over the standard loopy belief propagations. Firstly, it is computationally less expensive since there is only one iteration of operations involving a single inward pass followed by a single outward pass. Secondly, it does not have the problem of non-convergence issues faced by standard loopy belief propagation. Thirdly, because it terminates in one iteration, it can be used as a quick approximation method for applications that require fast approximation with a reasonably good inference accuracy that accounts for more dependencies than the arc removal method.

Now that we have a better idea of the working of the 2-Pass loopy belief propagation using the toy problem of Asia Network, we are ready to evaluate its performance on some standard benchmark test sets and comparisons to LBP. We have implemented the 2P-LBP algorithm using the libdai (https://staff.fnwi.uva.nl/j.m.mooij/libDAI/) open source C++ library that provides implementations of various inference methods for graphical models. The algorithms tested are:

- JTree: Junction Tree algorithm (libdai implementation)
- 2P-LBP: Our proposed algorithm (made use of some BP classes in libdai)
- LBP: Standard Loopy Belief Propagation (libdai implementation)
- DL-GBP: Double Loop Generalized Belief Propagation (libdai implementation)

We have chosen to use the DL-GBP algorithm [75] as a reference in our testing because it is a convergent iterative belief propagation algorithm that is guaranteed to converge towards a (possibly local) minimum of the Bethe free energy, although often at the expense of higher computation time. The test case networks were obtained from the *bnlearn Bayesian Network repository*[1] and the *2011 Pascal Probabilistic Inference Challenge*[2]. These networks vary in sizes and complexity, and their details are provided next.

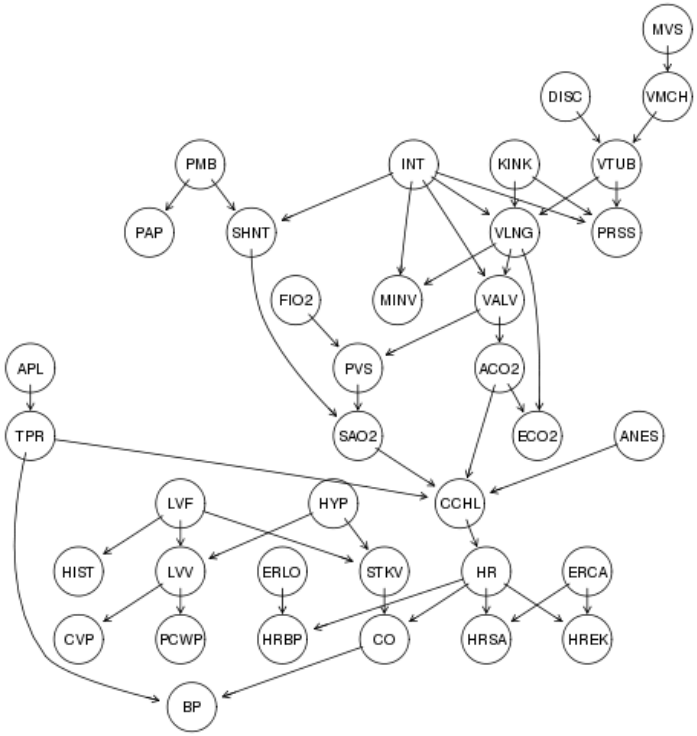
Description of Networks used: (source of network graphs: *bnlearn Bayesian Network repository*)

Network Name	Number of Nodes	Number of Arcs	Average Markov Blanket Size	Average degree	Maximum in-degree
Alarm (BN)	37	46	3.51	2.49	4
Andes (BN)	223	338	5.61	3.03	6
Barley (BN)	48	84	5.25	3.5	4
Insurance (BN)	27	52	5.19	3.85	3
Water (BN)	32	66	7.69	4.12	5
GridWrap-10x10 (MRF)	100	200	4	4	4

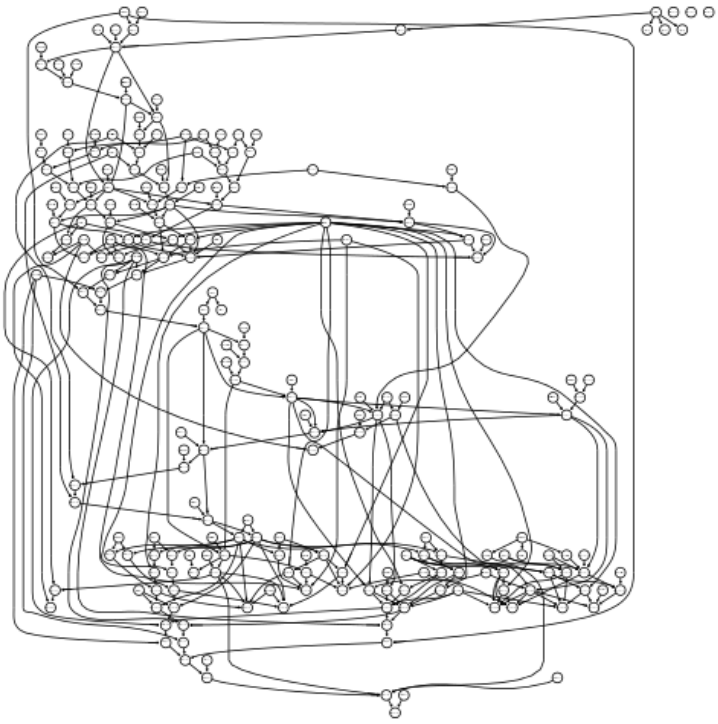
[1] http://www.bnlearn.com/bnrepository/

[2] http://www.cs.huji.ac.il/project/PASCAL/index.php

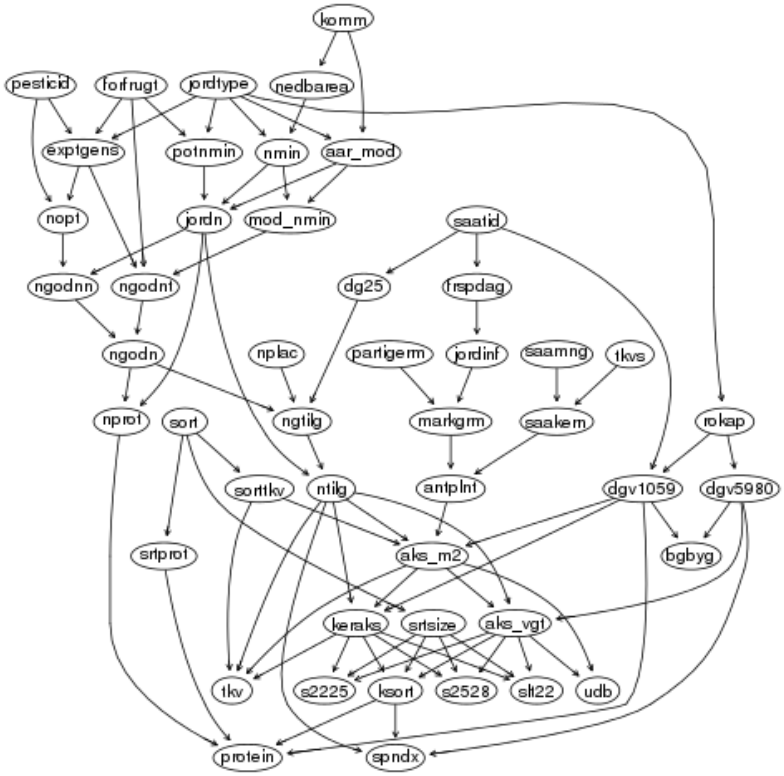
Network 1: Alarm



Network 2: Andes



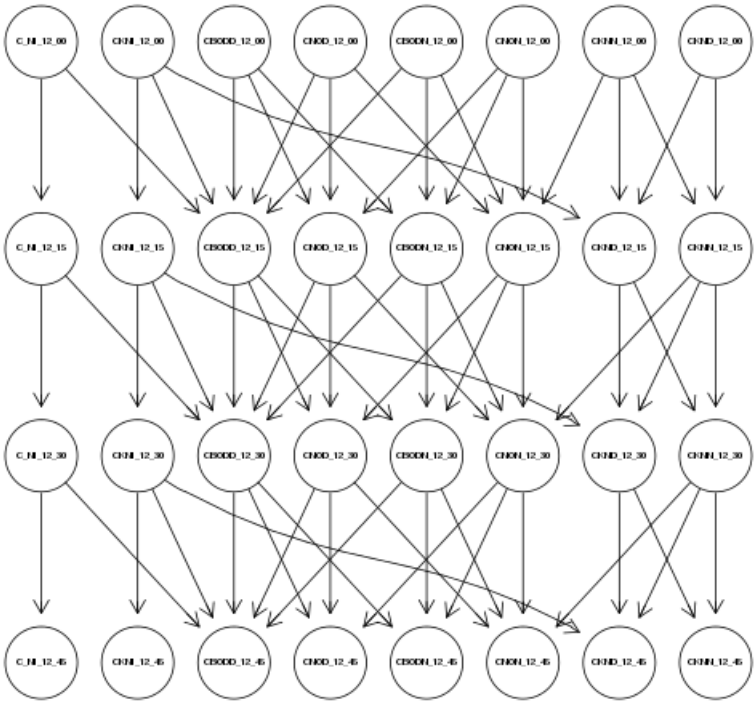
Network 3: Barley



Network 4: Insurance



Network 5: Water



Network 6: GridWrap -10x10 (Markov network)

A 10 by 10 grid with nodes at the top of the grid connected to the bottom and nodes at the left of the grid connected to the right.

For JTree, LBP, and DL-GBP, the default options in libdai were used, that is, no damping, maximum iterations set at 10000, and termination if change in messages or beliefs is less that 1E-9. We evaluated the performances in terms of CPU time in seconds and belief errors as compared to the exact marginal $P(x_i)$ produced by the Jtree algorithm. We report both the ℓ_∞ error and the average of the absolute error of the approximate single variable marginal b_i, calculated as $\underset{i \in V}{Max}\ \underset{x_i \in X_i}{Max} | b_i(x_i) - P(x_i) |$

and $Avg(\sum_{i \in V} \sum_{x_i \in X_i} | b_i(x_i) - P(x_i) |)$ respectively. The algorithms were tested on runs with no instantiation and instantiation of some of the leaf nodes. All instantiations were

set to the first state of the instantiated nodes. The results of the runs are presented in Table 7.3 and we present the pseudo code of the implementation below:

2P-LBP algorithm pseudo code implemented in libdai

```
1.      Read in Bayesian network BN.
2.      Create Join Tree cliques JTree.
3.      Run Init(JTree, BN) to determine the update sequence of belief propagation
4.      Clamp evidence if there are instantiations.
5.      Perform two-pass belief propagation based on the update sequence formed in
        Step 3.

Init(JTree, BN) {
        Vector <Clique> processOrder = topological sort the cliques of JTree in
        order of the inward pass towards an arbitrarily selected root clique
        FOR all cliques in processOrder, starting from the leaf clique
                Vector <Var> updateOrder = findOrder(clique, BN)
                Add updateOrder  to two-pass belief propagation update sequence
        END FOR
}

findOrder(clique, BN) {
        Construct undirected graph from BN (drop directions)
        Choose a variable in clique as the root arbitrarily
        Return a breadth-first search ordering of nodes in the clique based on the
undirected graph
}
```

Network Name (Instantiated Node Name [Node Index])	Maximum Belief Error JTree / 2P-LBP / LBP / DL-GBP	Average Belief Error JTree / 2P-LBP / LBP / DL-GBP	Time Taken (s) JTree / 2P-LBP / LBP / DL-GBP	Iterations JTree / 2P-LBP / LBP / DL-GBP
Alarm (-)	0 / **0.239** / 0.239 / 0.232	0 / **0.00533** / 0.00533 / 0.00368	0.000 / **0.000** / 0.016 / 5.772	- / - / 10 / 878
Alarm (BP [37])	0 / **0.274** / 0.274 / 0.267	0 / **0.0133** / 0.00966 / 0.00614	0.000 / **0.016** / 0.016 / 5.288	- / - / 21 / 912
Andes (-)	0 / **0.0644** / 0.0663 / 0.0389	0 / **0.00278** / 0.00271 / 0.00119	0.156 / **0.141** / 0.390 / 122.261	- / - / 15 / 1089
Andes (snode_151 [221])	0 / **0.0648** / 0.0667 / 0.0389	0 / **0.00285** / 0.00275 / 0.00121	0.156 / **0.140** / 0.452 / 126.041	- / - / 18 / 1089
Barley (-)	0 / **0.102** / 0.102 / 0.103	0 / **0.00725** / 0.00725 / 0.00597	7.535 / **0.608** / 0.344 / 341.504	- / - / 10 / 1281
Barley (protein [41], spndx [43])	0 / **0.101** / 0.106 / 0.103	0 / **0.00898** / 0.00928 / 0.00776	7.456 / **0.655** / 1.154 / 447.503	- / - / 35 / 1712
Insurance (-)	0 / **0.0781** / 0.0858 / 0.0716	0 / **0.0114** / 0.0120 / 0.00679	0.016 / **0.031** / 0.015 / 131.725	- / - / 7 / 1106
Insurance (propcost [19], medcost[22])	0 / **0.0732** / 0.0654 / 0.0385	0 / **0.0146** / 0.0150 / 0.00987	0.016 / **0.032** / 0.031 / 107.889	- / - / 21 / 880
Water (-)	0 / **0.00480** / 0.00480 / 0.00480	0 / **0.000169** / 0.000169 / 0.000169	1.264 / **0.047** / 0.063 / 55.098	- / - / 5 / 530

Water (Nodes ending with 12_45) [25 to 32]	0 / **0.00253** / 0.00252 / 0.00252	0 / **0.000237** / 0.000147 / 0.000151	1.201 / **0.063** / 0.203 / 43.47	- / - / 13 / 425
GridWrap10x10 (-)	0 / **0.100** / DNC / 0.190	0 / **0.0316** / DNC / 0.0497	8.736 / **0.031** / DNC / 86.348	- / - / DNC / 765
GridWrap10x10 ([100])	0 / **0.246** / DNC / 0.236	0 / **0.0747** / DNC / 0.0774	8.736 / **0.047** / DNC / 83.554	- / - / DNC / 771

Table 7.3: Belief difference and timing results on Bayesian networks and Markov networks. DNC = Did not converge after 10000 iterations.

In most of the test cases, 2P-LBP produced marginals with errors comparable to those produced by LBP and DL-GBP, and in some cases smaller errors than LBP. Furthermore, it did so with generally shorter computing timings. For example, in the case of the "Water" network, 2P-LBP achieved the same accuracy as LBP and DL-GBP, but in a much shorter time of 0.047s, compared to 0.063s, 55.098s, and 1.264s, scored by LBP, DL-GBP, and JTree respectively.

The good performance of 2P-LBP is clearly evident in the more complex test case of "GridWarp 10X10". In this case, the standard LBP could not even converge after 10000 iterations and hence produced no meaningful approximate marginals for comparisons. In fact, 2P-LBP out-performed DL-GBP in terms of both time and average error, and is also significantly faster than JTree.

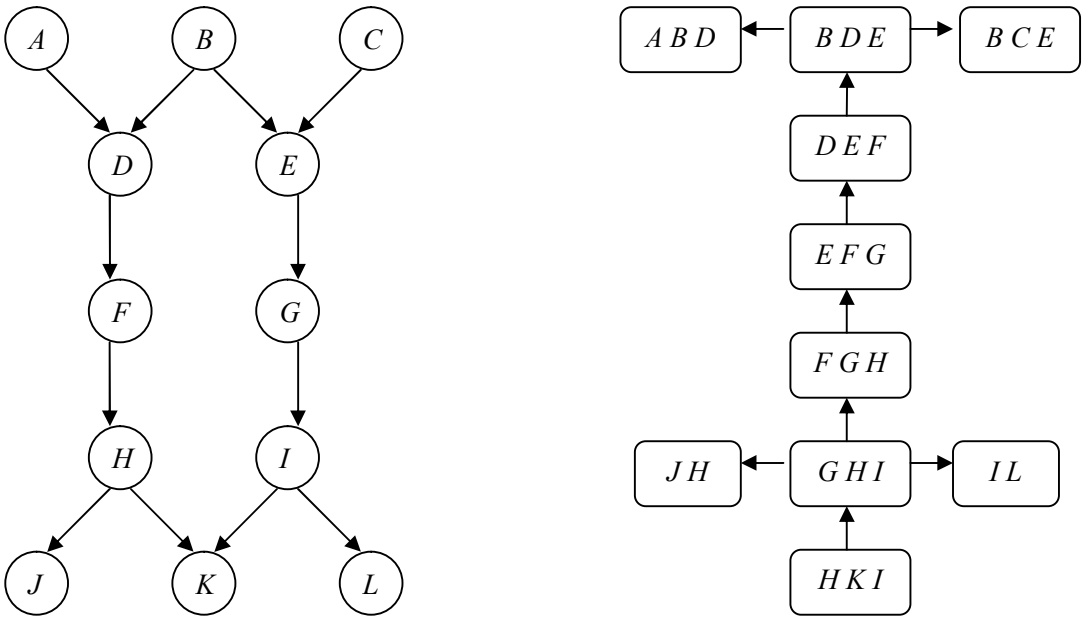
There is still much room to improve the 2P-LBP algorithm in terms of further reducing the computing time and achieving better inference accuracy. For example, as a quick approximation in this experiment, we only perform belief propagation on the singly connected sub-network structure for nodes within each clique in the join tree. In this way, we are indirectly making the assumption that they are singly-connected which is not true in general. For such cases, we can improve the inference accuracy by applying the same 2P-LBP algorithm recursively to the sub-network formed by the variables in the clique.

We will discuss more on this point in section 7.4. Also, the version of 2P-LBP implemented in this experiment did not account for the redundant message passing when moving from clique to clique. This happens when there is significant overlap of sub-network structures among connected cliques within the join tree. For example, in the case of Barley with no instantiation, 2P-LBP took 0.608s while LBP took only 0.344s (although with instantiation, 2P-LBP maintained the timing of 0.655s while LBP's timing increased to 1.154s). The way to account for such unnecessary message passing is to perform a pre-processing step at the time when the join tree was formed (Step 3 of pseudo code) to prune out these messages in advance.

In conclusion, we have demonstrated that the 2P-LBP has great potential as a fast approximate inference algorithm with comparable, and in some cases better, accuracy than standard LBP.

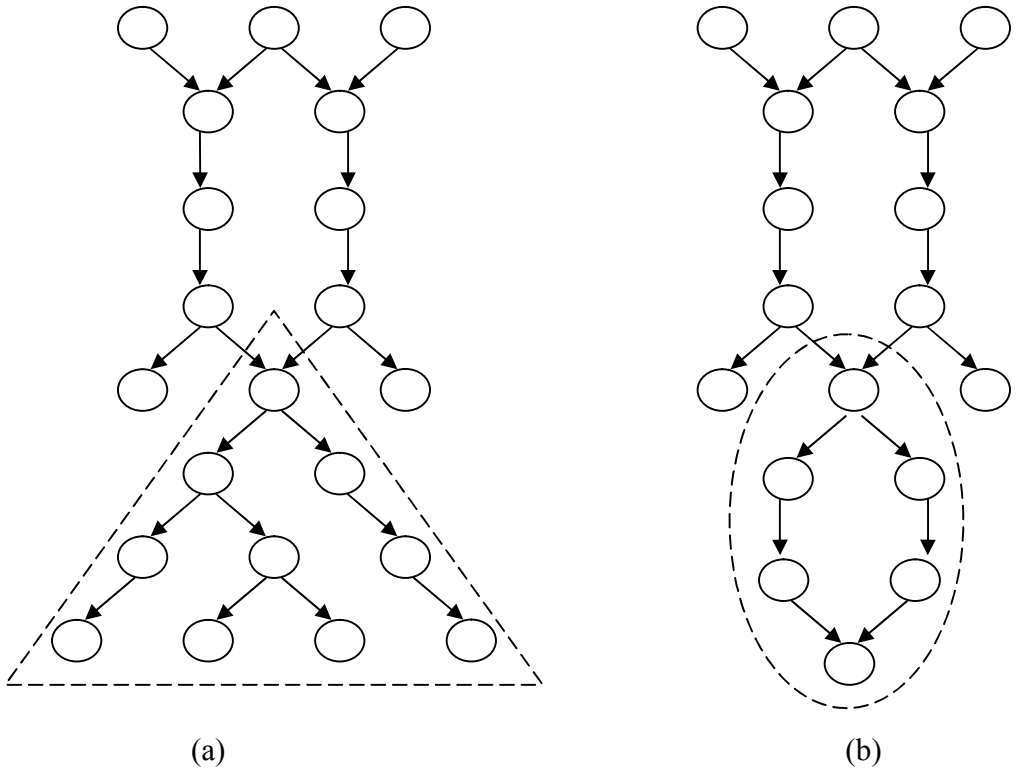
7.2 Examples where 2-Pass loopy belief propagation performs better

In this section, we will discuss the effects of using the 2-Pass control on loopy belief propagation as compared to the join tree algorithm and standard loopy belief propagation. Let us consider the Bayesian network shown in Figure 7.3(a) and its possible join tree in Figure 7.3(b).



(a) (b)

Figure 7.3: A Bayesian network (a) and its join tree (b).



(a) (b)

Figure 7.4: Examples of sub-networks connected to the sink node.

Assuming that there has been no instantiation to the network, both the 2-Pass and standard loopy belief propagation will produce the same inference result. In this case, both methods will compute the correct marginal probabilities for all nodes except node *K*. Based on the join tree clique order, the 2-Pass control's inward pass will result in the following sequence of messages being passed:

- $A \rightarrow D$, $B \rightarrow D$, $B \rightarrow E$, $C \rightarrow E$, $D \rightarrow F$, $E \rightarrow G$, $F \rightarrow H$, $G \rightarrow I$, $H \rightarrow K$, $I \rightarrow K$.

Note that in the above sequence, we have left out the uniform messages that do not provide any evidence update, for example, $J \rightarrow H$ and $L \rightarrow I$. Also, we use the symbol $\rightarrow$ to denote the direction in which a message is being passed instead of the direction of the arc in the original network, for example, $J \rightarrow H$ means node *J* sends a message to update node *H*. The outward pass is redundant as it does not produce any further updates to the messages. In this case, we can further apply the loopy correction using results established in Chapter 6, to obtain the true marginal probability of node *K*. In other words, the 2-Pass control with loopy correction can perform exact inference, which is an advantage it holds over the standard loopy belief propagation. In contrast, the join tree algorithm has to carry out much more computationally expensive operations and also requires more storage space to achieve exact inference at node *K*. In comparison to the loop correction alternative, it is definitely an overkill to maintain several clusters of nodes to account for the dependency between node *H* and node *I* due to the prior evidence at nodes *A*, *B*, and *C*.

Next, let us consider the case where node *K* has received λ evidence, which will cause the standard loopy belief propagation to iterate until it converges. For this case, let us suppose that there is a tree-structured sub-network connected to node *K* as shown in Figure 7.4(a), and the λ evidence received at node *K* is due to some instantiations of nodes within this sub-network. In this case, similar to what we have discussed in section 6.9, we can apply the loop correction during the inward pass of the 2-Pass loopy belief propagation to recover the true $\pi(H, I)$ at node *K* from the messages $\pi_K(H)$ and $\pi_K(I)$ it receives from node *H* and node *I* respectively. The significance of recovering

the true $\pi(H, I)$ at node K is that the 2-Pass method will produce exact inference results for the whole tree-structured sub-network. Note that the exact inference results for the sub-network hold true even if we do not apply loop corrections on the outward pass to the intermediate and source nodes above node K. On the other hand, for the standard loopy belief propagation, due to the circulating of evidence in the loop, node K will receive updated messages $\pi_K(H)$ and $\pi_K(I)$ at every iteration. This in turn causes additional updating to be carried out on for the whole of the tree-structured sub-network. Furthermore, even with these additional updates, it is unlikely that the converged messages $\pi_K(H)$ and $\pi_K(I)$ will yield the true $\pi(H, I)$ and hence the marginal posterior probabilities of nodes in the sub-network are only approximate. In this case, the 2-Pass method will clearly perform better than the standard loopy belief propagation both in terms of lower computational cost and better inference accuracy. Extending the same argument further, let us suppose that instead of having a tree structured sub-network we now have another loop connected to the sink node K. If the sink node of this sub-network has no λ evidence, then the behaviour of the 2-Pass method will be the same as the tree-structured sub-network case. However, if the sink node has received λ evidence, then the standard loopy belief propagation will need even more iterations to converge since the two individual loops' evidence updating are now intertwined and influencing each other. On the other hand, the 2-Pass method with loop correction is able to produce the true $\pi(H, I)$, which helps to break the connection between the two loops. This is because given the true $\pi(H, I)$, which acts as the prior probability of node K, the inference process of the sub-network below node K is now unaffected by any updates to the loop above it. Note that in this case, the inference process for the loop above node K will require the converged λ messages to be sent from the sub-network to node K. In a way, we can think of the 2-Pass method as partitioning the inference problem of the network with two connected loops into two separate single loop inference problems. In this case, it is obvious that the savings on the computational cost by using the 2-Pass method will be even more significant than the previous case. However, we cannot claim that the 2-Pass method will outperform the standard loopy belief propagation in terms of inference accuracy because the sub-network is now a loop and not a tree. Nevertheless, in the next

section, we will show empirically that the 2-Pass method can perform as well as, if not better than, the standard loopy belief propagation for a certain type of CPT parameters.

7.3 Empirical study on 2-Pass loopy belief propagation in a single loop with evidence at the sink node

The inference accuracy of the 2-Pass loopy belief propagation hinges on whether the sink node has received any λ evidence. If there is no λ evidence at the sink node, the 2-Pass method produces the same result as the standard loopy belief propagation, and we further propose to apply loop correction to recover the true marginal probability of the sink node. However, if the sink node has λ evidence, the standard loopy belief propagation will have to go through several iterations before it can converge. In this case, we do not have a theoretical foundation on which we can claim that the 2-Pass method will yield better approximate inference results although we did provide some theoretical insights in section Chapter 6 which will aid in future research on it. In fact, we can also extend the loop correction to the intermediate and source nodes, which will make the 2-Pass method an exact inference process for a single loop network. But this will come at the trade off of higher computational cost. Motivated by the good performance of the 2-Pass method in the experiments presented in section 7.1, we are interested to find out if it can outperform the standard loopy belief propagation even without loop correction for the case when the sink node has λ evidence. In this section we present results from the empirical study we have conducted to discover the conditions under which the 2-Pass method performs better. Specifically, by "conditions" we refer to the type of CPT parameters that could affect the inference accuracy of the 2-Pass method.

We setup our experiments in Matlab to solve for the inference problem on the single loop network shown in Figure 6.2. We implemented the 2-Pass loopy belief propagation for this single loop network and made use of the standard loopy belief propagation algorithm provided in the Bayes Net Toolbox [74]. Our approach in this empirical study is to vary the CPT parameters to see if we can observe some trends that

lead to better performance of the 2-Pass method. The CPT parameters are $P(C)$, $P(S \mid C)$, $P(R \mid C)$, and $\lambda_W(S, R)$, where $\lambda_W(S, R)$ is extracted from the relevant entries of $P(S, R \mid W)$ according to the instantiated state of node W or aggregated by applying the λ evidence it has received. For the purpose of our experiments, we generated the matrix $\lambda_W(S, R)$ directly instead of computing it from the matrix $P(S, R \mid W)$.

Test 1: In this first test, we generated 1000 cases of random CPT parameters $P(C)$, $P(S \mid C)$, $P(R \mid C)$, and $\lambda_W(S, R)$. We performed the 2-Pass and standard belief propagations on each of these cases and compared their results using two error measures, namely, the average absolute error for nodes *C*, *S* and *R*, and the maximum absolute error across all three nodes. The results of this test are presented as two graphs, shown in Figure 7.5, plotted based on the average error (on the left) and the maximum error (on the right), with standard loopy belief propagation error on the X-axis and the 2-Pass loopy belief propagation on the Y-axis. The result graphs of the other tests in this section will be presented in the same way. There is a line $y = x$ plotted on each graph. The coordinates of each point on the graph represents the pair of values (loopy belief propagation's error, 2-Pass method's error). The way to interpret the graph is: if a point falls below the line $y = x$ then for that case the 2-Pass method has produced a smaller error compared to the standard loopy belief propagation. Conversely, if the point is above the line then the standard loopy belief propagation has produced a smaller error. We can also view the graph as a correlation plot for both methods. From the graphs shown in Figure 7.5, we see that the standard belief propagation has performed better in more cases than the 2-Pass method for both error measures. Also, both methods produced relatively small errors in most of the cases, as seen from the many occurrences of points near the origin.

As the CPT parameters in Test 1 were randomly generated, we tried to sift through the test cases to look for patterns in the CPT parameters in cases that produced lower 2-Pass error. We discovered that for cases where the CPT parameters $P(S \mid C)$,

$P(R \mid C)$, and $\lambda_W(S, R)$ contained dominating diagonal entries, the 2-Pass method produced lower errors than the standard loopy belief propagation. Here is an example of a CPT matrix with dominating diagonal entries:

$$\begin{bmatrix} 0.7 & 0.3 \\ 0.2 & 0.8 \end{bmatrix}.$$

A CPT matrix with dominating diagonal entries models a strong dependency between 2 nodes. With further research, we found out that such CPT matrices belong to a class of potential functions known as *attractive* or *submodular* functions. Formally, a binary potential function f, represented in the following matrix form

$$\begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{bmatrix},$$

is attractive if $f(0,0) + f(1,1) \geq f(0,1) + f(1,0)$. In statistical physics, Ising models that have attractive potential functions are called *ferromagnetic*. There exist several interesting findings in the area of research on loopy belief propagation for graphical models with attractive potential functions. For example, it has been proven in [76] that for attractive binary Markov random fields, the Bethe free energy approximation associated with any fixed point of the loopy belief propagation always lower bounds the partition function. Note that in general, the loopy belief propagation does not provide a bound for the partition function unlike the naive mean field approximation. In the same paper, the authors also showed that empirically, the Bethe approximation's bound is tighter than the naive mean field's bound. In another work [77], it has been proven that the maximum a posterior probability (MAP) problem can be solved efficiently by graph cuts if the binary Markov random fields have attractive potential functions. In a more recent work, Weller and Jebara [78] have derived a fully polynomial time approximation (optimisation) scheme that is guaranteed to return an ε -approximation to the global minimum of the Bethe free energy for the class of attractive binary pair-wise Markov random fields.

Test 2: In this test, we generated 1000 cases of CPT parameters $P(S \mid C)$, $P(R \mid C)$, and $\lambda_W(S, R)$ such that they are all attractive in the following way:

- $P(s_1 \mid c_1) \in [l, 0.99]$, $P(s_2 \mid c_2) \in [l, 0.99]$
- $P(r_1 \mid c_1) \in [l, 0.99]$, $P(r_2 \mid c_2) \in [l, 0.99]$
- $\lambda_W(s_1 \mid r_1) \in [l, 0.99]$, $\lambda_W(s_2 \mid r_2) \in [l, 0.99]$
- $\lambda_W(s_1 \mid r_2) \in [0.01, 1-l]$, $\lambda_W(s_2 \mid r_1) \in [0.01, 1-l]$

By $P(s_1 \mid c_1) \in [l, 0.99]$, we mean that the value $P(s_1 \mid c_1)$ is randomly generated in the range $[l, 0.99]$. Note that the value of l dictates the strength of attractiveness. The value of l was set to 0.7 in this test, which means the dominating entries will have values in the range of $[0.7, 0.99]$. The parameters of $P(C)$ were randomly generated. Here is an example of the CPT parameters generated as described above:

$$P(S \mid C) = \begin{bmatrix} 0.73 & 0.27 \\ 0.15 & 0.85 \end{bmatrix}, \qquad P(R \mid C) = \begin{bmatrix} 0.88 & 0.12 \\ 0.25 & 0.75 \end{bmatrix},$$

$$\lambda_W(S, R) = \begin{bmatrix} 0.87 & 0.29 \\ 0.16 & 0.95 \end{bmatrix}, \qquad P(C) = \begin{bmatrix} 0.39 \\ 0.61 \end{bmatrix}.$$

Note that the row entries of $\lambda_W(S, R)$ are not required to sum to one since it is not a conditional probability distribution. From the graphs shown in Figure 7.6, we see that the accuracy of the 2-Pass methods has drastically improved in this test. In fact, for a vast majority of the 1000 cases the 2-Pass methods outperformed the standard belief propagation, and in cases which it lost, the magnitudes of the errors were comparable and most importantly very small.

Tests 3 and 4: Having observed the good performance of the 2-Pass method in Test 2, we wanted to further examine the effects of increasing the strength of attractiveness of the generated CPT parameters. Hence, we repeated Test 2 in Test 3 and Test 4 by increasing the values of l to 0.8 and 0.9 respectively. It is clearly shown in Figure 7.7 and Figure 7.8 that the increased strengths of attractiveness caused the 2-Pass method to perform even better than the standard loopy belief propagation with the best result achieved in the case $l = 0.9$. An interesting observation is that the inference errors

actually increased as the value of *l* increased for both the 2-Pass and standard loopy belief propagations. This seems to suggest that as the value of *l* increases, the inference problem becomes more difficult for the loopy belief propagation. A second interesting observation is that there seems to be some sort of underlying pattern that resembles a loop in the plot of the loopy belief propagation's error versus the 2-Pass method's error, which becomes increasingly more obvious as shown in Figure 7.6, Figure 7.7, and Figure 7.8. This phenomenon can be explained by the graphs shown in Figure 7.9, where the two graphs were plotted by varying the value of $P(c_1)$ from zero to one while maintaining the following attractive CPT matrices:

$$P(S \mid C) = \begin{bmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{bmatrix}, \quad P(R \mid C) = \begin{bmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{bmatrix},$$

$$\lambda_W(S, R) = \begin{bmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{bmatrix}.$$

For comparison, Figure 7.10 shows another set of error plots from varying the value of $P(c_1)$ from zero to one but using the following arbitrarily created CPT matrices instead:

$$P(S \mid C) = \begin{bmatrix} 0.8 & 0.2 \\ 0.3 & 0.7 \end{bmatrix}, \quad P(R \mid C) = \begin{bmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{bmatrix},$$

$$\lambda_W(S, R) = \begin{bmatrix} 0.1 & 0.6 \\ 0.8 & 0.9 \end{bmatrix}.$$

By comparing the graphs in Figure 7.9 and Figure 7.10, we see that the patterns of the correlation plots between the 2 methods are more regular.

Tests 5 and 6: In Tests 2, 3 and 4, all the CPT matrices were attractive. Now, we would like to further investigate if either set of the CPT matrices $\{ P(S \mid C),\ P(R \mid C) \}$ and $\{ \lambda_W(S, R) \}$ contributed more to the better performance of the 2-Pass method. In Test 5, we generated matrices of $\{ P(S \mid C),\ P(R \mid C) \}$ to be attractive and $\{ \lambda_W(S, R) \}$ to be random. In Test 6, we generated matrices of $\{ P(S \mid C),\ P(R \mid C) \}$ to be random and $\{ \lambda_W(S, R) \}$ to be attractive. The results illustrated in Figure 7.11 and 7.12 clearly

show that the attractiveness of the matrix $\lambda_W(S, R)$ played a more significant role in contributing to the better performance of the 2-Pass method.

Consolidating the results of all the tests in this section, we have shown empirically that the 2-Pass loopy belief propagation produced more accurate inference results than the standard loopy belief propagation for the class of attractive CPT matrices in a single loop. Furthermore, the accuracy increased as the strength of attractiveness increased. We end this section by highlighting that while the tests have been conducted on a small loop, the results will hold even if bigger loops have been used. This is because as long as there are no instantiations on the intermediate nodes, we can always compact the big loop to the size of the small loop used in this section by following the approach we have presented in section 6.3.

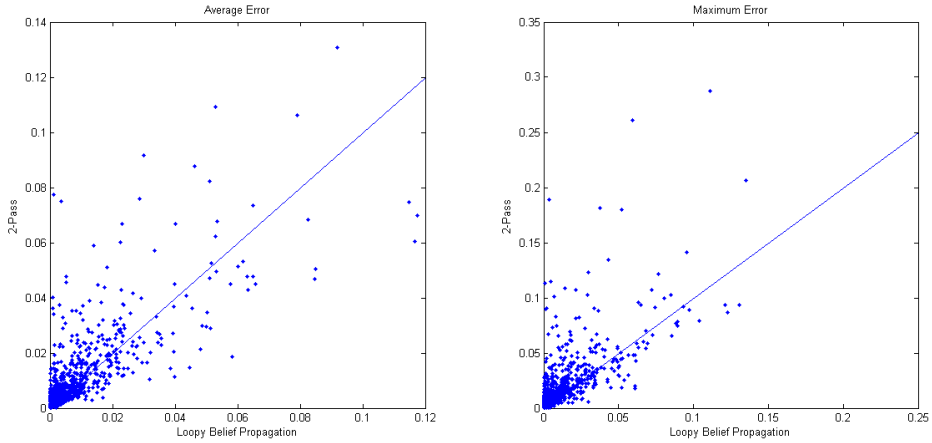


Figure 7.5: Results of Test 1.

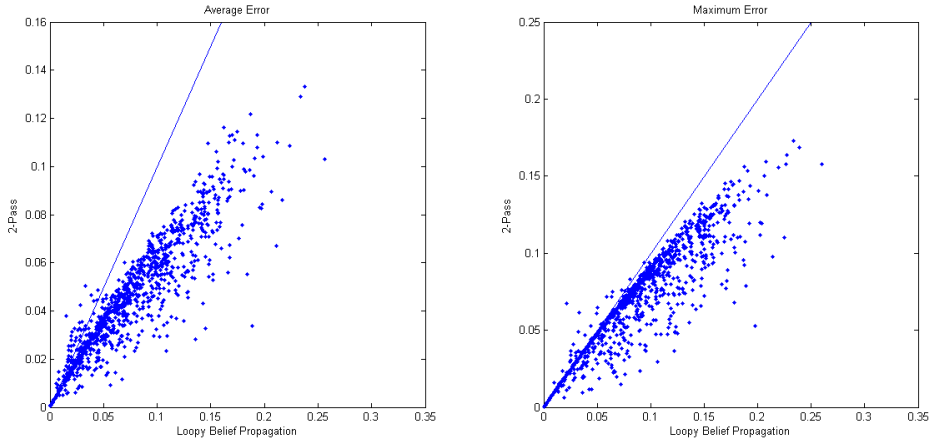


Figure 7.6: Results of Test 2.

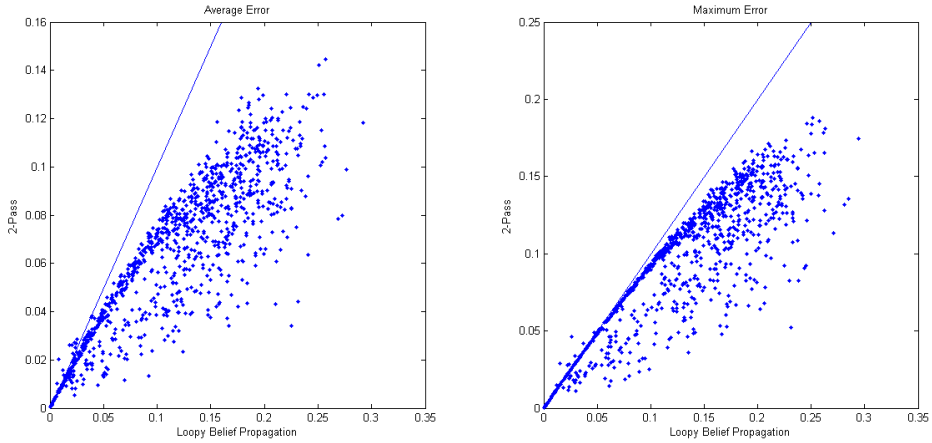


Figure 7.7: Results of Test 3.

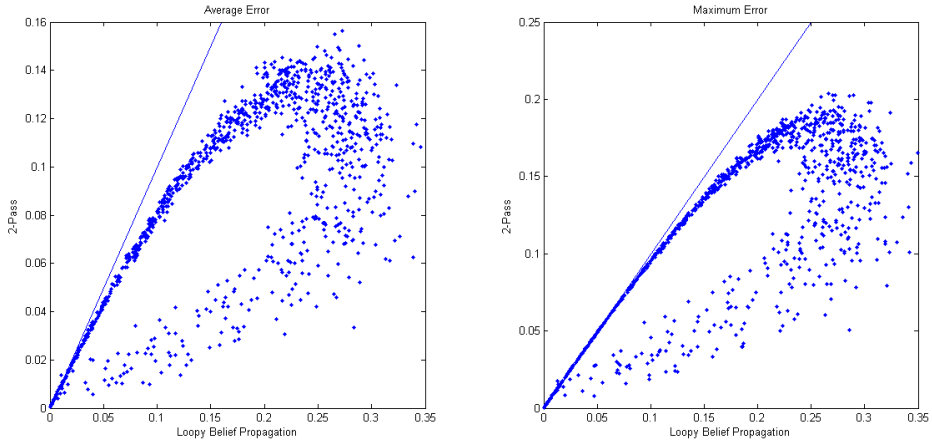


Figure 7.8: Results of Test 4.

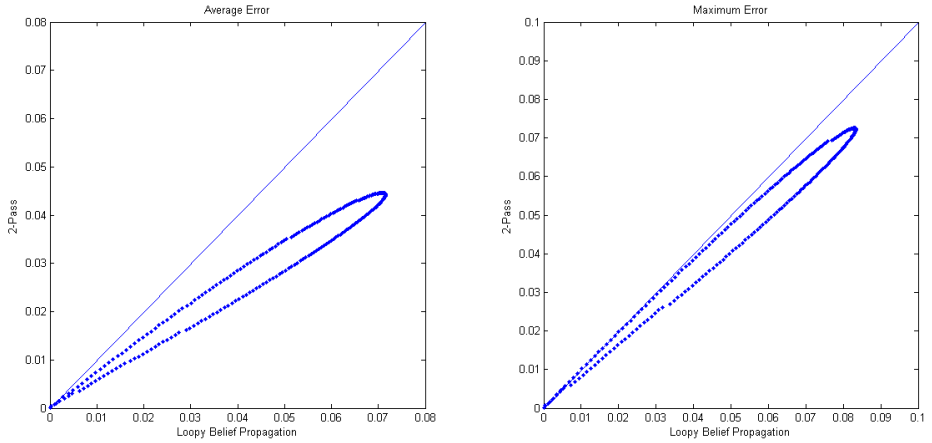


Figure 7.9: Error graphs for a fixed set of attractive CPT matrices with varying $P(c_1)$.

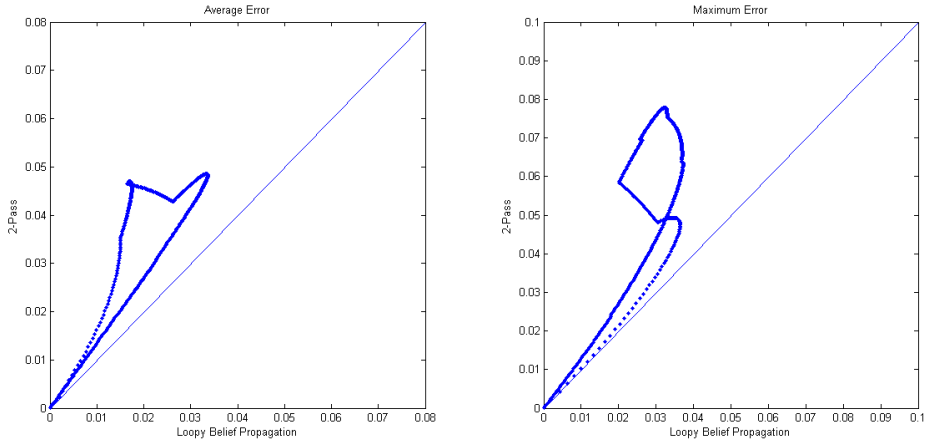


Figure 7.10: Error graphs for a fixed set of arbitrary CPT matrices with varying $P(c_1)$.

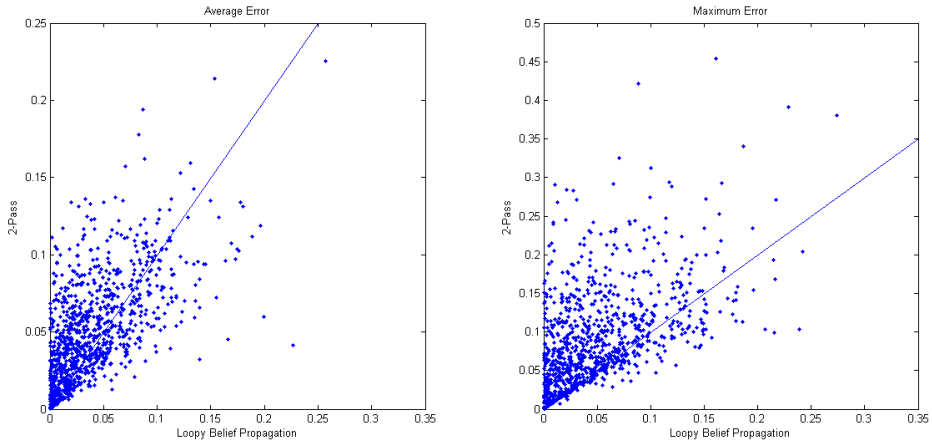


Figure 7.11: Results of Test 5.

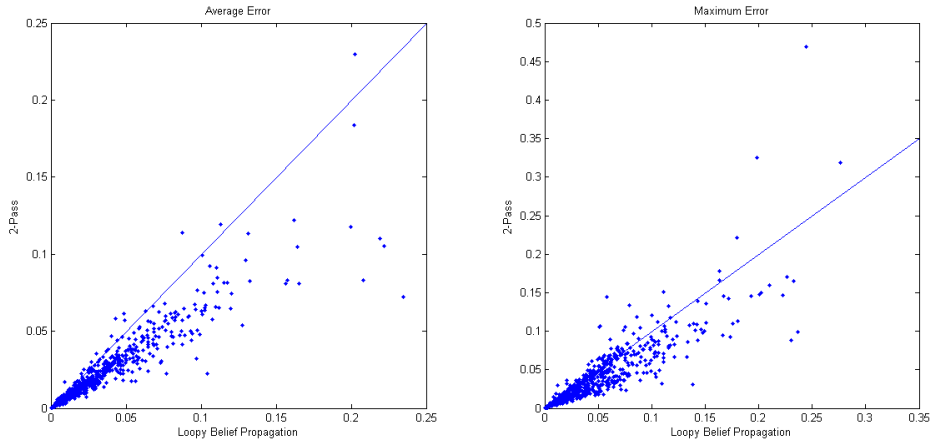


Figure 7.12: Results of Test 6.

7.4 Discussions on using 2-Pass loopy belief propagation with loop correction for approximate inference

In this last section, we advocate using the 2-Pass loopy belief propagation with loop correction as an approximate inference method and we will discuss some issues related to applying it to more generic loopy Bayesian networks. Most of the Bayesian networks modelled in practical problems that are multiply connected will be more complex than the type of single loop networks we have examined so far. Therefore, a direct application of the 2-Pass method will likely produce only approximate inference results. Furthermore, there is an additional issue, which we provide more details later, of the need to handle loops within a clique referenced by the join tree when controlling the message passing sequence with the 2-Pass control. Nevertheless, we will demonstrate using concrete examples that some of these more complex loopy networks can be transformed or modified to new network structures suitable for the application of the 2-Pass approximation. Through these examples it will demonstrate the potentials of using the 2-Pass approximation as an alternative method to the standard loopy belief propagation. The central idea to the network transformation or modification is to convert the original loopy network into one that contains possibly disjoint simple loops such that the 2-Pass control with loop correction can be applied. The good performance of the 2-Pass method is based upon the foundation that it could perform exact inference through loop correction for a single loop with no evidence at the sink node. When there is evidence at the sink node, we have the option to also produce exact inference results for the intermediate and source nodes through loop corrections, albeit at the price of higher computational costs.

Let us consider applying the 2-Pass method to the Bayesian network shown in Figure 7.13(a). While this network looks only slightly more complex than a single loop of the same size, it actually contains 6 overlapped simple loops, namely, $\{A, B, C, D\}$, $\{A, B, C, E\}$, $\{B, D, E, F\}$, $\{C, D, E, F\}$, $\{A, B, C, D, E, F\}$, and $\{B, C, D, E\}$.

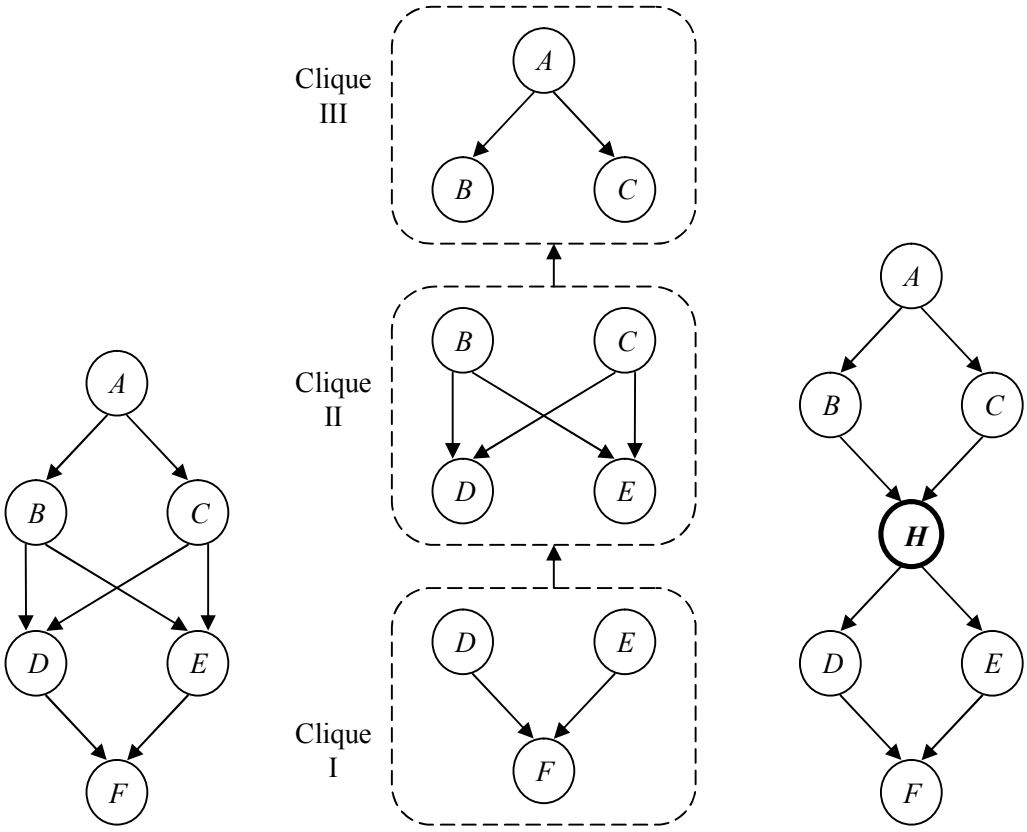


Figure 7.13: An example network (a), its join tree (b), and its modified network (c) using a hidden node.

We will use this example to bring out the issue on loops occurring within a clique referenced by the corresponding join tree, which the 2-Pass method will now have to address. Recall that in applying the 2-Pass method, we control the sequence of node updates guided by the 2-Pass order of the clique order of a join tree. For example, with reference to the constructed join tree shown in Figure 7.13(b), it means that for the inward pass we will update the nodes through message passing in the order of Clique III, Clique II, and Clique I. When updating the nodes in a clique, messages are passed along the structure of the sub-network found in the original Bayesian network. For example, we use the network structure $B \leftarrow A \rightarrow C$ when updating the nodes in Clique III. We shall call such network structure the clique structure. While the 2-Pass method is seen as a convergent modification to the standard loopy belief propagation guided by the 2-Pass

order of a join tree, we can also view it from another perspective. That is, it is a modified (approximate) join tree algorithm that, instead of maintaining a big potential table for all nodes in the clique so that it can marginalise to the separator set, only maintains individual belief functions for each node in the clique and tries to recover the required joint potential function for the separator set by error correction. Note that if we do not apply error correction, the default operations of the belief propagation will compute the joint potential function by treating the nodes in the separator set as independent. It is fairly obvious that the clique structures of all the cliques in a join tree formed from a single loop network will always be singly connected. Thus, the message passing process will be efficient when updating nodes in each clique controlled by the 2-Pass method. However, for a more complex network, the structure of a clique might be multiply connected. An example is the loop structure of Clique II shown in Figure 7.13(b). We propose handling clique structures with loops by solving them as sub-problems using either the 2-Pass method or standard loopy belief propagation. For the example shown in Figure 7.13(b), if we solve the clique structure of Clique II using the 2-Pass method, then it leads to a further partitioning of the clique structure into 2 sub-cliques of $\{B, C, D\}$ and $\{B, C, E\}$. Then the updating of nodes in Clique II is done by performing both the inward and outward passes between the two sub-cliques. It is important to note that the order of the sub-cliques is not integrated into the original clique order. In other words, we have to perform the two passes between the sub-cliques of Clique II twice – during the inward pass from Clique III to Clique I, and the outward pass from Clique I to Clique III. Through this example, we see that the computational cost of the 2-Pass method increases with the complexity of the clique structures. Nevertheless, its computation cost will still be lower than applying the standard loopy belief propagation. Furthermore, the presence of additional updating order on the sub-cliques also demonstrates the principled control used by the 2-Pass method, which is not obvious in previous discussions.

When the original network contains more complex loops it often leads to bigger cliques in the join tree. Furthermore, the clique structures are also likely to be multiply connected, with the possibility of having more than a single loop in each clique. One way of solving the inference problem for such networks is to use the 2-Pass method as an

approximation by applying it to a modified network that approximates the original network. We propose using the hidden node approach presented in section 5.3 to modify the network structure into simpler forms that contain disjoint single loops, which is suitable for applying the 2-Pass method. Let us consider the same example network shown in Figure 7.13(a). In this case, we can simplify the structure of Clique II by introducing a new hidden node *H* as shown in Figure 7.13(c). Suppose we define node *H* as a cluster node of nodes *B* and *C*, that is $H = \{B, C\}$, then we could actually perform exact inference using the 2-Pass method. However, in cases where the number of parent nodes is high, the number of states of node *H* will become too large to permit efficient computations. Thus, we have to resort to approximating the original network structure by using a smaller number of states for node *H*. In this case, we will have to learn the CPT parameters of the hidden node structure using the learning methods presented in section 5.3. Nevertheless, through this example, we can see the usefulness of altering the network structure using hidden nodes to form simpler loops for the purpose of applying the 2-Pass method. We believe more research can be conducted in this direction to explore the full potential of it leading to the development of a good approximate inference algorithm.

We observed in section 6.1 that we can actually analyse the Bethe free energy for loopy belief propagation using a single parameter $p_{sr} = bel(s_1, r_1)$, and from our findings presented in section 6.2 we know that we can recover the true joint distribution of *S* and *R* using the following error equation:

$$P_{true}(S, R) = P_{indep}(S, R) + \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \times const .$$

This suggests that we only need to compute a single parameter *const* to recover $P_{true}(S, R)$. These observations prompted us to take a slightly different approach to our error analysis steps, which we will discuss next.

Suppose we are given

$$P_{true}(S) = \begin{bmatrix} p_s \\ 1 - p_s \end{bmatrix} \quad \text{and} \quad P_{true}(R) = \begin{bmatrix} p_r \\ 1 - p_r \end{bmatrix},$$

and we want to compute $P_{true}(S, R)$. We can fully specify $P_{true}(S, R)$ using a single parameter

$$p_{sr} = P_{true}(s_1, r_1)$$

by writing it as

$P(S, R)$	r_1	r_2
s_1	p_{sr}	$p_s - p_{sr}$
s_2	$p_r - p_{sr}$	$1 - (p_s + p_r - p_{sr})$

Therefore

$$P(S, R) = \begin{bmatrix} p_{sr} & p_s - p_{sr} \\ p_r - p_{sr} & 1 - (p_s + p_r - p_{sr}) \end{bmatrix}$$
$$= \begin{bmatrix} 0 & p_s \\ p_r & 1 - (p_s + p_r) \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} p_{sr}$$

Recall that we have shown

$$P_{true}(s_1, r_1) - P_{indep}(s_1, r_1) = (a_1 - b_1)(p_1 - q_1)\pi(c_1)(1 - \pi(c_1)),$$

hence

$$p_{sr} = p_s p_r + (a_1 - b_1)(p_1 - q_1)\pi(c_1)(1 - \pi(c_1))$$

Now let us suppose that we do not have the above answer to compute p_{sr} from p_s and p_r, we will show that we can still attempt to "guess" the correct value of p_{sr} using the following sets of constraints:

- $0 \le p_s \le 1, 0 \le p_r \le 1, 0 \le p_{sr} \le 1$
- $p_{sr} \le p_s$, $p_{sr} \le p_r$
- $0 \le p_s + p_r - p_{sr} \le 1 \Rightarrow p_s + p_r - 1 \le p_{sr} \le p_s + p_r$

Combining the above constraints into a single constraint, we obtain

$$\max(0, p_s + p_r - 1) \le p_{sr} \le \min(p_s + p_r, p_s, p_r)$$

which is equivalent to

$$\max(0, p_s + p_r - 1) \le p_{sr} \le \min(p_s, p_r).$$

If we define

$$\alpha = \min(p_s, p_r) \text{ and } \beta = \max(p_s, p_r),$$

we have

$$\max(0, \alpha + \beta - 1) \leq p_{sr} \leq \alpha \ .$$

Note that

- if $\alpha + \beta \geq 1$, then $\alpha + \beta - 1 \leq p_{sr} \leq \alpha$,
- If $\alpha + \beta \leq 1$, then $0 \leq p_{sr} \leq \alpha$.

The possible range of value for p_{sr} given p_s and p_r is depicted in Figure 7.14.

Let us represent the range of p_{sr} by using a single *k* value as follows

$$p_{sr} = \alpha - k(1 - \beta)$$

where

$$0 \leq k \leq \min(\tfrac{\alpha}{1-\beta}, 1) \ .$$

For the case of $P_{indep}(S, R)$, $k = \alpha$ since

$$\begin{aligned} P_{indep}(s_1, r_1) &= p_s p_r \\ &= \alpha\beta \\ &= \alpha - \alpha(1 - \beta) \end{aligned}$$

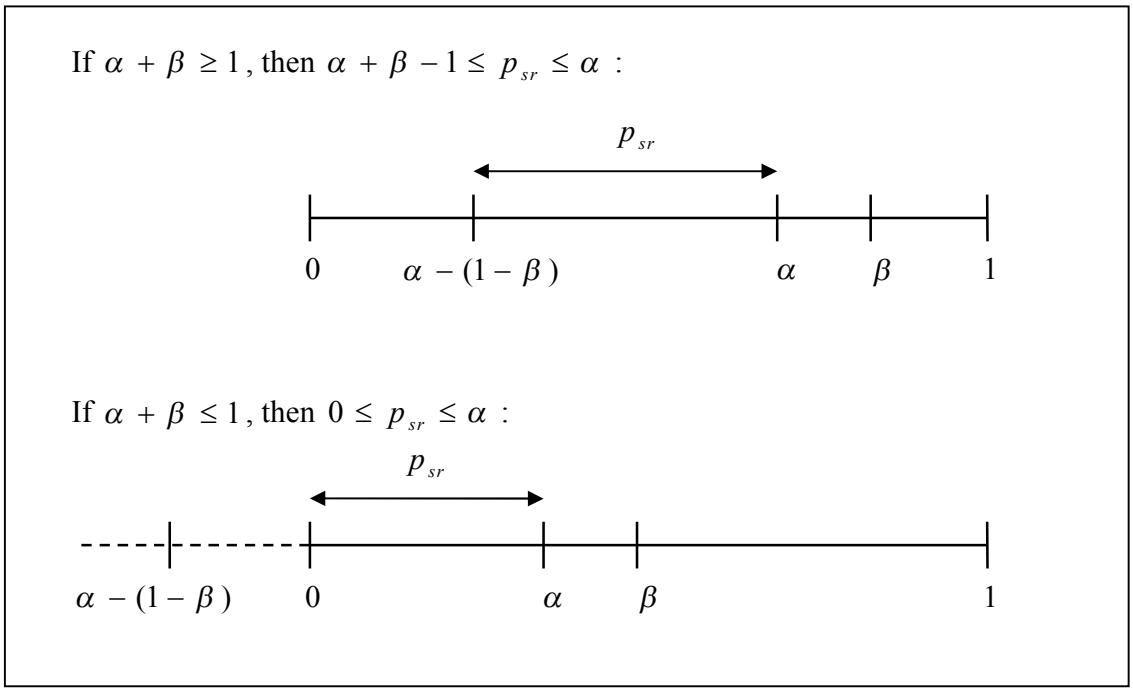


Figure 7.14: Constraints on range of p_{sr} given p_s and p_r .

By using the k value representation, we see that the loopy belief propagation algorithm always attempt to "guess" the true value of p_{sr} by choosing $k = \alpha$ by default, which give rise to the loopy error. Furthermore, the advantage of writing $P_{true}(S, R)$ in terms of k is that we can then develop an approximation scheme by learning a function for k based on (p_s, p_r), i.e. $k = f(p_s, p_r)$, or heuristics to choose the value of k with better accuracy than $k = \alpha$ (the case of loopy belief propagation). To illustrate this point, let us consider the error term presented in section 6.2

$$\begin{aligned}
\big(\alpha - k(1-\beta)\big) - \big(\alpha - \alpha(1-\beta)\big) &= (a_1 - b_1)(p_1 - q_1)\pi(c_1)(1-\pi(c_1)) \\
\big(\alpha - k\big)\big(1-\beta\big) &= (a_1 - b_1)(p_1 - q_1)\pi(c_1)(1-\pi(c_1)) \\
k &= \alpha - \frac{1}{1-\beta}(a_1 - b_1)(p_1 - q_1)\pi(c_1)(1-\pi(c_1))
\end{aligned}$$

In this case of a simple loop, we can in fact compute the value of $\pi(c_1)$ solely from p_s or p_r, since

$$p_s = \pi(c_1)a_1 + (1-\pi(c_1))b_1$$

we have

$$\pi(c_1) = \frac{p_s - b_1}{a_1 - b_1} \text{ and } (1-\pi(c_1)) = \frac{a_1 - p_s}{a_1 - b_1}.$$

Hence, we can compute the exact value of k in this case by

$$\begin{aligned}
k &= \alpha - \frac{1}{1-\beta}(a_1 - b_1)(p_1 - q_1)\frac{p_s - b_1}{a_1 - b_1}\frac{a_1 - p_s}{a_1 - b_1} \\
&= \alpha + \frac{1}{1-\beta}\frac{p_1 - q_1}{a_1 - b_1}(p_s - a_1)(p_s - b_1) \\
&= f(p_s, p_r) \\
&= f(\alpha, \beta)
\end{aligned}$$

The main point we want to bring out in this discussion is that we can actually recover the initial $\pi(C)$ message that enters the source node just by examining the top down messages received at the parents of the sink node. More importantly, for more complex network, we might want to learn an approximate function $f^*(p_s, p_r)$ offline which is more efficient to compute online by studying the possible range of value of p_{sr}.

Lastly, a similar approach to using hidden nodes to simplify the network structure for the purpose of approximate inference is to perform arc removal [79][80]. One simple approach used by arc removal methods is to remove arcs in the original network with weak dependencies resulting in a singly connected network on which exact inference can be performed efficiently, using the standard belief propagation algorithm. Here, when applying arc removal to transform the original network into one suitable for applying the 2-Pass method, we do not require the modified network to be singly connected. All we need is for the modified network to have simpler disjoint loops. Clearly, in this case, fewer arcs will be removed and the modified network will account for more dependencies among the nodes in the network.

Chapter 8

Conclusion

In the first part of this thesis, we have examined recent neural computational models of belief propagation. We highlighted the significance of these models that demonstrated the viability of performing belief propagation using neural computations by transforming it into a dynamical system. However, we noted that it is still a fairly new idea restricted to simpler graphical models like HMM and binary pair-wise MRF, and there are many open problems yet to be solved. We have also suggested the idea of implementing the belief propagation in computational models like the Hopfield network through free energy minimisation.

In Part II, we have presented an error analysis of loopy belief propagation on networks with single loops. We have derived the error terms for a single loop network with generic CPT entries for varying cases of node instantiations. With the discovery of the error terms, we then proposed algorithms 6.3 and 6.4 as exact inference algorithms using belief propagation with loop correction to handle single loop network with and without evidence entering at the sink node. We have also discussed on the merits of using these algorithms with several examples and highlighted points related to their applications. We have also proposed a new approximate inference algorithm called 2-Pass loopy belief propagation that combines the merits of the join tree algorithm (convergence governed by a 2-pass control) and iterative belief propagation methods (computations within a single iteration is often very fast) for fast approximate inference.

The 2-Pass loopy belief propagation was motivated by observations we made while studying the working of join tree algorithm on loopy graphs and the error analysis of loopy belief propagation that we have conducted. We have investigated it performance empirically and discussed several issues related to its application.

We started our error analysis by examining the differences between the join tree algorithm and loopy belief propagation when they are applied to a simple loop. The analysis was conducted by stepping through both methods' message passing processes as well as using the free energy fixed point approach. We gained the insight that the loopy belief propagation's error occurs at the sink node due to the wrong assumption that its parent nodes are marginally independent. On the other hand, the join tree algorithm circumvents this problem by maintaining the joint probability (belief) of the parent nodes. This insight prompted us to research on deriving the error term to allow us to recover the true joint distribution of the parent nodes.

By analysing the dependency between the intermediate nodes due to the source node, we successfully derived the loopy error terms for varying configurations of single loop networks with no instantiations to the sink node. The error terms not only allow us to perform loop corrections, they also provide interesting insights into the behaviour of the loopy belief propagation. For example, they show mathematically that the loop error decreases as the size of the loop increases. We have also analysed the bound provided by the error terms, which can be used to decide if loop correction is required at the expense of additional computational time. For the more complex case where the sink node has received λ evidence, we have shown how the circulating of evidence in the loop causes double counting of evidence. Nevertheless, we have shown that if we control the order in which the nodes are updated, according to the clique order of its join tree, we can still compute the true marginal for the whole loop. This fact played an important role in motivating our design of the 2-Pass loopy belief propagation as a variant to the standard loopy belief propagation.

In our initial investigation on the performance of the 2-Pass loopy belief propagation, we tested it on the Asia network and compared its accuracy against the exact solution provided by the join tree algorithm as well as the approximations produced by the standard loopy belief propagation and the arc removal method. Both experiments with instantiations to the sink node clearly showed that the 2-Pass method performed comparably to the standard loopy belief propagation but with the advantage of lower computational costs. In addition, the 2-Pass method's inference results closely matched those produced by the join tree algorithm, and in this case it is obvious that the reduction in computational cost by the 2-Pass method is even larger.

We further extended our empirical tests to open source benchmark test sets using the libdai C++ library that provides implementations of several inference methods for graphical models. We tested 2P-LBP against the JTree, LBP and DL-GBP methods. In most of the test cases, 2P-LBP produced marginals with errors comparable to those produced by LBP and DL-GBP, and in some cases smaller errors than LBP, and it did so with generally shorter computing timings. The good performance of 2P-LBP is more evident in the more complex test case of "GridWarp 10X10". In this case, the standard LBP could not even converge after 10000 iterations and hence produced no meaningful approximate marginals for comparisons. In fact, 2P-LBP out-performed DL-GBP in terms of both time and average error, and is also significantly faster than JTree.

An important argument we put forth is that at the baseline of performance, both methods will produce the same exact inference results when the loop-cutset nodes have been instantiated and the same approximate inference results when none of the sink nodes of the loops have been instantiated. Then if we apply results from our loopy error analysis and incorporate the loop corrections with the 2-Pass control, we can further improve its inference accuracy. Furthermore, in cases with simple disconnected loops, the 2-Pass loopy belief propagation with loop corrections can actually perform an exact inference. We supported this argument with two concrete examples where the sink node of a loop is connected with i) a tree-structured sub-network and ii) a sub-network with loop. We further argued that it is computationally less expensive to apply the 2-Pass method since

there is only one iteration of operations involving a single inward pass followed by a single outward pass. It also does not have the problem of non-convergence issues faced by the standard loopy belief propagation. Furthermore, because it terminates in one iteration, it can be used as a quick approximation method for applications that require a fast approximation with a reasonably good inference accuracy that accounts for more dependencies than the arc removal method.

Through empirical study, we have discovered that the 2-Pass method produced much better inference results than the standard loopy belief propagation when the CPT parameters of the loop were attractive. We also observed that the accuracy increased as the strength of attractiveness increased.

We have discussed issues related to applying the 2-Pass method as an approximate inference method in networks with slightly more complex loop structures. A natural extension is to apply the 2-Pass method recursively to cliques with complex structures. This will cause the computational cost to increase as the complexity of the clique structures increase, but we believe it will still be lower than applying the standard loopy belief propagation, which has to iterate through rounds of message passing until it converges. Hence, we have proposed an alternative approach using the hidden node method to modify a complex network structure into simpler forms that contain disjoint single loops, which is suitable for applying the 2-Pass method. As another alternative, we also mentioned the advantage of using the 2-Pass method on a network modified through arc removal, and explored the possibility of using the single *k*-value representation for approximating the error term.

Lastly, we conclude the thesis with the following suggestions for future work:

- Extend the loopy error term derivation beyond a single loop network. For example, consider overlapping loops that might not share the same source or sink node.
- Derive good approximation scheme to estimate the error correction term that is efficient to compute for more complex loopy network

- Investigate the improvement in accuracy if 2-Pass loopy belief propagation is applied recursively to cliques with complex network structures and if the increase in accuracy justifies the increase in computing time.

Bibliography

[1] K. Korb and A. Nicholson, *Bayesian Artificial Intelligence*. Chapman & Hall/CRC, 2003.

[2] G. W. Ng, K. H. Ng, K. H. Tan, and C. H. K. Goh, "Novel Methods for Fusing Bayesian Network Knowledge Fragments in D'Brain," in *Proceedings of the 10th International Conference on Information Fusion*, 2007.

[3] A. Yu and P. Dayan, "Inference, attention, and decision in a Bayesian neural architecture," *Advances in neural information processing systems*, vol. 17, pp. 1577–1584, 2005.

[4] S. Deneve, "Bayesian spiking neurons I: Inference," *Neural Computation*, vol. 20, no. 1, pp. 91–117, 2008.

[5] J. Pearl, *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Morgan Kaufmann Publishers, Inc., 1988.

[6] M. Jordan and Y. Weiss, "Graphical models: Probabilistic inference," *The Handbook of Brain Theory and Neural Networks, 2nd edition; MIT Press, Cambridge, MA, USA*, 2002.

[7] J. Pearl, *Causality: models, reasoning and inference*, vol. 29. Cambridge University Press, 2000.

[8] G. Cooper, "The computational complexity of probabilistic inference using Bayesian belief networks (research note)," *Artificial intelligence*, vol. 42, no. 2–3, pp. 393–405, 1990.

[9] P. Dagum and R. Chavez, "Approximating probabilistic inference in Bayesian belief networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 3, pp. 246–255, 1993.

[10] R. Kindermann, J. Snell, and A. M. Society., *Markov random fields and their applications*. American Mathematical Society Providence, RI, 1980.

[11] J. M. Hammersley and P. Clifford, "Markov fields on finite graphs and lattices," in *unpublished*, 1971.

[12] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Generalized belief propagation," *Advances in Neural Information Processing Systems 13*, vol. 13, pp. 689–695, 2001.

[13] F. Kschischang, B. Frey, and H. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, 2001.

[14] J. Yedidia, W. Freeman, and Y. Weiss, "Understanding Belief Propagation and its Generalizations," *TR-2001-22, Mitsubishi Electric Research Laboratories*, 2002.

[15] G. Elidan, I. Mcgraw,, and D. Koller, “Residual belief propagation: Informed scheduling for asynchronous message passing,” in *Proceedings of the Twenty-second Conference on Uncertainty in AI (UAI)*, Boston, Massachussetts, 2006.
[16] J. Gonzalez, Y. Low, and C. Guestrin, “Residual splash for optimally parallelizing belief propagation,” presented at the Artificial Intelligence and Statistics (AISTATS), 2009, pp. 177–184.
[17] K. P. Murphy, Y. Weiss, and M. I. Jordan, “Loopy Belief Propagation for Approximate Inference: An Empirical Study,” in *Proceedings of Uncertainty in AI*, 1999, pp. 467–475.
[18] R. McEliece, D. Mackay, and J. Cheng, “Turbo decoding as an instance of Pearl’s ‘belief propagation’ algorithm,” *IEEE Journal on Selected Areas in Communications*, pp. 140–152, 1998.
[19] J. S. Yedidia, W. T. Freeman, and Y. Weiss, “Constructing free-energy approximations and generalized belief propagation algorithms,” *IEEE Transactions on Information Theory*, vol. 51, no. 7, pp. 2282–2312, 2005.
[20] S. Bader and P. Hitzler, “Dimensions of neural-symbolic integration-a structured survey,” *Arxiv preprint cs.AI/0511042*, 2005.
[21] T. Minka, “Expectation Propagation for approximate Bayesian inference,” in *Proceedings of the Seventeenth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-01)*, San Francisco, CA, 2001, pp. 362–369.
[22] R. Wilson and F. Keil, *The MIT encyclopedia of the cognitive sciences*. MIT Press, 1999.
[23] L. Barsalou, “Grounding symbolic operations in the brain’s modal systems,” *Embodied grounding: Social, cognitive, affective, and neuroscientific approaches*, pp. 9–42, 2008.
[24] B. Hammer and P. Hitzler, *Perspectives of Neural-Symbolic Integration*. Springer Verlag, 2007.
[25] K. Doya, S. Ishii, A. Pouget, and R. Rao, *Bayesian brain: probabilistic approaches to neural coding*. MIT Press, 2007.
[26] J. B. Tenenbaum, C. Kemp, T. L. Griffiths, and N. D. Goodman, “How to Grow a Mind: Statistics, Structure, and Abstraction,” *Science*, vol. 331, no. 6022, pp. 1279–1285, Mar. 2011.
[27] D. George and J. Hawkins, “A hierarchical Bayesian model of invariant pattern recognition in the visual cortex,” in *Proceedings of the International Joint Conference on Neural Networks. IEEE*, 2005, pp. 1812–1817.
[28] W. Gerstner, “Spiking Neurons,” in *Pulsed Neural Networks*, W. Maass and C. M. Bishop, Eds. MIT Press, 1999, pp. 3–53.
[29] W. Maass, “Computing with spikes,” *Special Issue on Foundations of Information Processing of TELEMATIK*, vol. 8, no. 1, pp. 32–36, 2002.
[30] R. Rao, “Hierarchical Bayesian inference in networks of spiking neurons,” *Advances in neural information processing systems*, vol. 17, p. 1113?120, 2005.
[31] R. Rao, “Neural models of Bayesain belief propagation,” *Bayesian brain*, pp. 239–267, 2007.
[32] R. P. N. Rao, “Bayesian inference and attentional modulation in the visual cortex,” *Neuroreport*, vol. 16, no. 16, pp. 1843–1848, 2005.

[33] A. P. Shon and R. P. N. Rao, “Implementing belief propagation in neural circuits,” *Neurocomputing*, vol. 65, pp. 393–399, 2005.
[34] D. Knill and A. Pouget, “The Bayesian brain: the role of uncertainty in neural coding and computation,” *TRENDS in Neurosciences*, vol. 27, no. 12, pp. 712–719, 2004.
[35] J. Mooij and H. Kappen, “On the properties of the Bethe approximation and loopy belief propagation on binary networks,” *J. Stat. Mech*, vol. 11012, 2005.
[36] T. Richardson and R. Urbanke, *Modern Coding Theory*. New York: Cambridge University Press, 2008.
[37] T. Ott and R. Stoop, “The Neurodynamics of Belief Propagation on Binary Markov Random Fields,” *Advances in neural information processing systems*, vol. 19, p. 1057, 2007.
[38] T. Heskes, “Stable Fixed Points of Loopy Belief Propagation Are Local Minima of the Bethe Free Energy,” in *NIPS*, 2002, p. 343—350.
[39] M. Welling and Y. W. Teh, “Belief Optimization for Binary Networks: A Stable Alternative to Loopy Belief Propagation,” in *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2001, pp. 554–561.
[40] R. Rojas, *Neural networks: a systematic introduction*. Springer, 1996.
[41] R. Rao, “Hierarchical Bayesian inference in networks of spiking neurons,” *Advances in neural information processing systems*, vol. 17, p. 1113?120, 2005.
[42] R. P. N. Rao, “Bayesian computation in recurrent neural circuits,” *Neural Computation*, vol. 16, no. 1, pp. 1–38, 2004.
[43] J. M. Beck and A. Pouget, “Exact inferences in a neural implementation of a hidden Markov model,” *Neural Computation*, vol. 19, no. 5, pp. 1344–1361, 2007.
[44] S. L. Lauritzen and D. J. Spiegelhalter, “Local computations with probabilities on graphical structures and their application to expert systems (with discussion),” *Journal of the Royal Statistical Society, Series B*, vol. 50, pp. 157–224, 1988.
[45] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen, “Bayesian updating in causal probabilistic networks by local computations,” *Computational Statistics Quarterly*, vol. 4, pp. 269–282, 1990.
[46] HUGIN, “Bayesian Networks development software,” *http://www.hugin.com*.
[47] G. Shafer and P. Shenoy, “Probability propagation,” *Annals of Mathematics and Artificial Intelligence*, vol. 2, no. 1–4, pp. 327–351, 1990.
[48] S. C. Tatikonda and M. I. Jordan, “Loopy belief propagation and Gibbs measures,” in *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, San Francisco, CA, USA, 2002, pp. 493–500.
[49] T. Heskes, “On the uniqueness of loopy belief propagation fixed points,” *Neural Computation*, vol. 16, no. 11, pp. 2379–2413, Nov. 2004.
[50] A. Ihler, J. Fischer, and A. Willsky, “Loopy Belief Propagation: Convergence and Effects of Message Errors,” *J. Mach. Learn. Res.*, vol. 6, pp. 905–936, 2005.
[51] J. Mooij and H. Kappen, “Sufficient Conditions for Convergence of the Sum–Product Algorithm,” *Information Theory, IEEE Transactions on*, vol. 53, no. 12, pp. 4422–4437, 2007.
[52] M. Wainwright, T. Jaakkola, and A. Willsky, “Tree-based reparameterization framework for analysis of sum-product and related algorithms,” *IEEE Transactions on Information Theory*, vol. 49, no. 5, pp. 1120–1146, May 2003.

[53] X. Shi, D. Schonfeld, and D. Tuninetti, "Message error analysis of loopy belief propagation for the sum-product algorithm," presented at the 2010 IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP), 2010, pp. 2078–2081.
[54] R. E. Tarjan and M. Yannakakis, "Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs," *SIAM J. Comput.*, vol. 13, no. 3, pp. 566–579, Jul. 1984.
[55] F. V. Jensen, *An introduction to Bayesian Networks*. UCL Press, 1996.
[56] S. L. Lauritzen, *Graphical Models*. UK: Clarendon Press, 1996.
[57] C. Huang and A. Darwiche, "Inference in Belief Networks: A Procedural Guide," *International Journal of Approximate Reasoning*, vol. 15, pp. 225–263, 1994.
[58] J. Binder, D. Koller, S. J. Russell, and K. Kanazawa, "Adaptive probabilistic networks with hidden variables," *Machine Learning*, vol. 29, no. 2–3, pp. 213–244, 1997.
[59] C.-K. Kwoh and D. F. Gillies, "Using hidden nodes in Bayesian networks," *Artificial Intelligence*, vol. 88, no. 1–2, pp. 1 – 38, 1996.
[60] J.-W. Bang and D. Gillies, "Using Bayesian Networks with Hidden Nodes to Recognise Neural Cell Morphology," in *PRICAI 2002: Trends in Artificial Intelligence*, vol. 2417, M. Ishizuka and A. Sattar, Eds. Springer Berlin Heidelberg, 2002, pp. 385–394.
[61] S. Russell, J. Binder, D. Koller, and K. Kanazawa, "Local learning in probabilistic networks with hidden variables," in *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2*, San Francisco, CA, USA, 1995, pp. 1146–1152.
[62] S. Ramachandran, *Theory Refinement of Bayesian Networks with Hidden Variables*. Ph.D. Thesis, Department of Computer Sciences, University of Texas, Austin, 1998.
[63] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc., 1995.
[64] C.-K. Kwoh, *Probabilistic Reasoning from Correlated Objective Data*. PhD Thesis, Imperial College of Science, Technology and Medicine, University of London, 1995.
[65] J.-W. Bang, *Hidden Nodes in Bayesian Networks and their application to Prognostic Analysis of Hepatitis C*. PhD Thesis, Imperial College of Science, Technology and Medicine, University of London, 2002.
[66] A. Darwiche, *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.
[67] B. Bidyuk and R. Dechter, "Epsilon-cutset in Bayesian networks of arbitrary topology," University of California, Irvine, 2001.
[68] Y. Weiss, "Correctness of Local Probability Propagation in Graphical Models with Loops," *Neural Computation*, vol. 12, no. 1, pp. 1–41, Jan. 2000.
[69] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.
[70] S. C. Tatikonda, "Convergence of the sum-product algorithm," in *Proceedings 2003 IEEE Information Theory Workshop*, 2003, pp. 222–225.
[71] N. Taga and S. Mase, "Error Bounds Between Marginal Probabilities and Beliefs of Loopy Belief Propagation Algorithm," in *MICAI 2006*, A. Gelbukh and C. A. Reyes-Garcia, Eds. 2006, pp. 186–196.

[72] A. Ihler, “Accuracy bounds for belief propagation,” in *Proceedings of UAI 2007*, 2007, pp. 183–190.
[73] J. M. Mooij and H. J. Kappen, “Bounds on marginal probability distributions,” in *Advances in Neural Information Processing Systems 21 (NIPS*2008)*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. 2009, pp. 1105–1112.
[74] K. P. Murphy, “Bayes Net Toolbox,” *https://code.google.com/p/bnt/*.
[75] T. Heskes, K. Albers, and B. Kappen, “Approximate inference and constrained optimization,” in *19th Annual Conference on Uncertainty in Artificial Intelligence*, 2003, pp. 313–320.
[76] E. B. Sudderth, M. J. Wainwright, and A. S. Willsky, “Loop Series and Bethe Variational Bounds in Attractive Graphical Models.,” in *NIPS*, 2007.
[77] V. Kolmogorov and R. Zabih, “What energy functions can be minimized via graph cuts?,” *IEEE TPAMI*, vol. 26, no. 2, pp. 147–59, 2004.
[78] A. Weller and T. Jebara, “Approximating the Bethe Partition Function,” in *30th Annual Conference on Uncertainty in Artificial Intelligence*, 2014.
[79] U. Kjaerulff, “Reduction of computational complexity in Bayesian networks through removal of weak dependencies,” in *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, 1994, pp. 374–382.
[80] A. Choi, H. Chan, and A. Darwiche, “On Bayesian Network Approximation by Edge Deletion,” *CoRR*, vol. abs/1207.1370, 2012.