University of Windsor

Scholarship at UWindsor

Electronic Theses and Dissertations | Theses, Dissertations, and Major Papers

2010

Efficient Probabilistic Inference Algorithms for Cooperative Multiagent Systems

Hongxuan Jin
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation

Jin, Hongxuan, "Efficient Probabilistic Inference Algorithms for Cooperative Multiagent Systems" (2010). *Electronic Theses and Dissertations*. 414.
https://scholar.uwindsor.ca/etd/414

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

EFFICIENT PROBABILISTIC INFERENCE ALGORITHMS FOR COOPERATIVE MULTI-AGENT SYSTEMS

A DISSERTATION
SUBMITTED TO THE SCHOOL OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF UNIVERSITY OF WINDSOR
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Hongxuan Jin
May 2010

© Copyright by Hongxuan Jin 2010
All Rights Reserved

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Dan Wu) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(who who)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(who who
(Electrical Engineering))

Approved for the University Committee on Graduate Studies.

Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

Probabilistic reasoning methods, Bayesian networks (BNs) in particular, have emerged as an effective and central tool for reasoning under uncertainty. In a multi-agent environment, agents equipped with local knowledge often need to collaborate and reason about a larger uncertainty domain. Multiply sectioned Bayesian networks (MSBNs) provide a solution for the probabilistic reasoning of cooperative agents in such a setting.

In this thesis, we first aim to improve the efficiency of current MSBN exact inference algorithms. We show that by exploiting the calculation schema and the semantic meaning of inter-agent messages, we can significantly reduce an agent's local computational cost as well as the inter-agent communication overhead. Our novel technical contributions include 1) a new message passing architecture based on an MSBN linked junction tree forest (LJF); 2) a suite of algorithms extended from our work in BNs to provide the semantic analysis of inter-agent messages; 3) a fast marginal calibration algorithm, designed for an LJF that guarantees exact results with a minimum local and global cost.

We then investigate how to incorporate approximation techniques in the MSBN framework. We present a novel local adaptive importance sampler (LLAIS) designed to apply localized stochastic sampling while maintaining the LJF structure. The LLAIS sampler provides accurate estimations for local posterior beliefs and promotes efficient calculation of inter-agent messages.

We also address the problem of online monitoring for cooperative agents. As the MSBN model is restricted to static domains, we introduce an MA-DBN model based on a combination of the MSBN and dynamic Bayesian network (DBN) models. We show that effective multi-agent online monitoring with bounded error is possible in an MA-DBN through a new secondary inference structure and a factorized representation of forward messages.

Acknowledgement

First and foremost, I would like to express my sincere gratitude to my advisor Dr. Dan Wu for his support and research guidance throughout the course of this research. I would also like to thank the other members of my thesis committee for their valuable time and helpful suggestions. I would like to thank my colleague, Tania, with whom I have had the pleasure of working during her Master's study.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

An intelligent agent is usually defined as a computational or natural system that senses its environment and takes actions intelligently according to its own goals [92]. Such an agent can process local observations, generate appropriate decisions and execute the chosen action. Some examples include autonomous mobile robots, internet infobots and intelligent tutors. A *probabilistic agent* uses probabilistic knowledge representations and reasons explicitly with regard to the state of the domain. For instance, the driverless car, which won the DRAPA Grant Challenge in 2005, has demonstrated the power of a real world probabilistic application on a single-agent mobile robot.

In recent years, systems involving multiple agents that communicate with each other in a distributed fashion have become more prevalent. Cooperative agents need to reason collectively about the states of an uncertain domain based on their local knowledge and inter-agent communication. This can happen either in a static time-invariant or a dynamic temporal environment. For instance, one problem is how four driverless cars on a city street can collaborate with each other and coordinate their actions, in order to avoid any collision and safely pass a four-way-stop intersection. We are facing the challenge of how to fully utilize and extend the existing representation models and inference algorithms for a single probabilistic agent to multi-agent settings.

One well-studied model for cooperative multi-agent probabilistic reasoning is the *Multiply Sectioned Bayesian Network* (MSBN) extended from the traditional Bayesian network (BN) model. With an MSBN, we can decompose a larger problem domain into subdomains, each individually represented and managed by a relatively lightweight single agent. Multiple agents can collectively reason about the state of the global domain based on their local knowledge, local observation, and limited inter-agent communication. Existing inference calculation in MSBN is carried out in some secondary structures, typically a *linked junction tree forest* (LJF). An LJF consists of local junction trees (JT) each for an agent's local domain and linkage trees connecting a pair of neighboring agents.

In this thesis, we show that while an LJF provides a coherent framework for exact inference with MSBNs, it is too costly to carry out efficient computation with the current Hugin-based message passings. We introduce techniques extending the BN Shenoy-Shafer architecture to the LJF inference structure for improved efficiency of exact global propagation. Not only is our method able to avoid the repeated local updates, but it also avoids full rounds of local message passing completely. Still, in larger and more complex problem domains, the exponential computation of LJF global inference could render any exact representation and calculation mostly impractical. It is thus natural to consider the possibility of trading off exact inference against the calculation speed and communication cost with approximate approaches. Unfortunately, although approximate techniques have been well-developed in traditional BNs, their extension to MSBNs has been very limited. In the second part of this thesis, we thus focus on the design of alternative approximate solutions to the existing MSBN based multi-agent probabilistic inference. Last but not the least, we move on to the dynamic problem domain and present a novel model that describes the temporal evolvement of dynamic agents. Our new model supports effective online monitoring for a group of cooperative agents with bounded errors.

Overall, we propose solutions to the following three questions in this thesis:

1. How to improve the efficiency and robustness of existing exact inference algorithms;

2. How to apply practical approximation techniques in an MSBN model;

3. How to effectively model and reason with a group of dynamic probabilistic agents.

Our solutions are based on the issues and difficulties addressed below.

1.1 Improving Message Passing in LJFs

Most existing inference algorithms are applied on an LJF with Hugin-based recursive message scheduling schema for inter-agent communication. This results in excessive amount of local computation because each agent's local belief has to be updated repeatedly, and each update triggers a round of message propagation in the agent's LJF local JT. Extending from the BN Shenoy-Shafer message passing, we propose a new message oriented architecture for LJFs, such that all inter-agent messages are explicitly calculated and buffered. We will show that although the total number of external messages remains the same, it is much more efficient to compute these messages with our new architecture. This improvement is due to that repeated local updates are no longer needed, and local message passings are conducted more efficiently through a partial propagation. We completely avoid any full round passings of local messages.

With the traditional recursive methods, inter-agent message passing is very sensitive to unreliable communication channels. Also, periodical off-line times can prevent each agent from observing local evidence continuously. We thus try to support the exact MSBN belief updating with iterative message passing. We present an iterative version of our new architecture, along with a scheme that avoids repeated multiplications during message computation. We show that the convergence of iterative message passing to exact results is guaranteed. More importantly, temporary communication errors can be tolerated without causing global belief updating failures.

1.2 Marginal Calibration

The marginal distribution, or *prior marginal* distribution, of an MSBN subnet's local variables is essential for an agent to reason about its own problem subdomain. *Marginal calibration* refers to the process of forming the prior marginal in each local domain. The initial potential assignment of MSBN subnets does not provide such information. This is because the agent's local junction tree is not consistent after the construction, and more importantly, each subnet's initial potential does not necessarily contain all the required information to form the prior marginal.

A fast calibration ensures efficient global inference. With all existing algorithms, the marginal calibration is carried out through standard inter-agent messages passings. Such a calibration process is implicit and is usually expensive in both time and space. In this thesis, we introduce a marginal calibration algorithm based on the theories developed for the cluster calibration in traditional BN junction trees and our new LJF message passing architecture.

The *global propagation* (GP) method used in the Hugin architecture is arguably one of the best methods for exact probabilistic inference method in BNs. Passing messages between clusters (cliques) in a JT is the basic operation in the GP method. It is traditionally considered that the messages passed are simply potentials without any specific semantic meaning. We study the factorizations of a joint probability distribution defined by a BN *before* and *after* the GP method is performed, and we investigate the messages passed algebraically. We reveal that the messages are actually separator marginals or their factorizations, thus passing messages in the GP method can be equivalently considered as the problem of allocating separator marginals. This novel perspective of propagation gives rise to a more efficient way of computing cluster marginals with both the Hugin and the Shenoy-Shafer message passings.

Extending the above results, we design an MSBN marginal calibration algorithm that requires the minimum inter-agent message passing and local computation. We introduce the concept of prior marginal (PM) factors for a complete prior distribution of MSBN subnets. Based on a distributed analysis of these factors at the compile time, we can guide the actual runtime inter-agent communication by sending only the necessary messages.

1.3 Localized Stochastic Sampling in MSBNs

Although stochastic sampling has been successfully used in BN approximation, application of these techniques to the MSBN global context has been proven to be problematic. Earlier attempts of MSBN approximation algorithms forgo the LJF structure and sample an MSBN directly in the global context [93]. It has been shown that such approximation indeed requires more inter-agent message passing, and at the cost of revealing more private knowledge of each local subnet. Furthermore, MSBN global sampling schema tend to explore only a small part of the entire multi-agent domain space.

We thus aim to maintain the LJF framework and explore localized approximation, which is realized through an individually carried out sampling process at an agent's subnet. In a calibrated local JT, such local approximation is possible, but standard BN sampling algorithms can not be applied directly. One major obstacle is the need for supporting the inter-agent message calculation over linkage trees. We should be able to obtain an inter-agent message, which is composed of a set of extended linkage potentials, accurately and efficiently with the local sampling algorithm.

As we study the extension of BN importance sampling techniques to JTs, we present a novel LJF-based Local Adaptive Importance Sampler (LLAIS). We design our importance function as tables of posterior probabilities over the clusters of an LJF local JT. We adopt the adaptive importance sampling, such that the importance functions are learned sequentially to approach the optimal sampling distribution. One innovative feature of the LLAIS is that it facilitates inter-agent message calculation. We can obtain an approximation of a linkage tree message from the learned importance function before the local sampling is completed.

1.4 Multi-agent Probabilistic Reasoning in Dynamic Domains

Another problem investigated in this thesis is the representation models and inference algorithms for a dynamically evolving multi-agent system. Cooperative agents often need to reason about the states of a domain that changes over time. For example, in many applications, agents need to track the state of such systems, a problem known as *tracking*, or *monitoring*. Essentially, each agent needs to determine the posterior probability for nodes of interest, given a set of accumulated evidence from the agent's own observation and those of other agents. *Online monitoring* requires the calculation of monitoring results at each time step at runtime.

Our goal is to provide a fast and accurate online monitoring calculation for a group of cooperative agents. Although the MSBN model has been applied successfully in the multi-agent probabilistic reasoning, it is restricted to static problem domains. On the other hand, the dynamic Bayesian network (DBN) model is well known for modeling dynamic (temporal) domains involving a single agent. We are thus motivated to search for a possible combination of the two for representing dynamic uncertainty knowledge in a multi-agent setting. However, several obstacles to such integration exist. In particular, a decomposed representation of joint JPD does not guarantee efficient inference calculation in dynamic domains. The spatial distribution of the multi-agent systems could conflict with the temporal message passing for dynamic multi-agent probabilistic reasoning.

Our solution to a compact representation and effective inference framework is to exploit weak interactions between each dynamic agent's individual evolvement over time. By assuming certain level of independency among the temporal advance of the cooperative agents, we can take the advantage of both MSBN and DBN models and provide an approximate solution to the multi-agent online monitoring problem with a new model named as Multi-Agent Dynamic Bayesian Networks (MA-DBN). While agents are organized according to an underlying hypertree structure to facilitate inter-agent communication, each dynamic agent maintains an individual chain of evolution. We introduce a new secondary structure of an MA-DBN called LDJF, which enables a factorized and more efficient computation of the cooperative online monitoring.

1.5 Thesis Overview

The organization of the thesis is summarized below:

- **Chapter 2: Background** This chapter presents an introduction to probabilistic graphical models, particularly, the Bayesian network (BN) model and the multiply sectioned Bayesian network (MSBN) model. We discuss major exact and approximate inference algorithms for BNs. We also introduce the secondary structure of MSBNs, a linked junction tree forest (LJF), as well as existing algorithms for calculating the posterior probability distribution with an MSBN LJF.

- **Chapter 3: An Improved LJF Message Passing Architecture** In this chapter, we present a new message passing architecture for MSBN LJFs. Different from the traditional Hugin-based message passing, our new approach adopts from the Shenoy-Shafer architecture by utilizing linkage trees as message buffers. An inter-agent message can be originated from either a consistent or inconsistent local JT, and a full local update is never issued. The new architecture can be extended to allow asynchronized passing of iterative messages; such a scheme maintains the correctness of the exact message calculation with improved robustness of inter-agent communication.

- **Chapter 4: BN Prior Marginal Factors** In this chapter, we examine the problem of JPD factorization in traditional single-agent BNs. We investigate the semantic meaning of messages passed over the separator of each pair of neighbouring JT clusters. We present a procedure named Allocate Separator Marginal(ASM) to determine the actual information a cluster requires to form the marginal in the JT. We show how the ASM procedure can help to form the marginal with a minimum messages passing.

- **Chapter 5: Fast Marginal Calibration** Current MSBN calibration methods are performed implicitly and expensively in terms of both inter-agent messages passing and local computation. They are not suitable when an explicit prior marginal is needed for certain subnets. In this chapter, we present a new marginal calibration algorithm that is based upon informed message passing; not only does it provide a correct prior explicitly, but it also requires a minimum amount of inter-agent messages and local calculation.

- **Chapter 6: Local Adaptive Importance Sampling** In this chapter, we address the problem of approximate inference with an MSBN. We show that localized approximation can be combined with the existing MSBN LJF framework, thus providing a practical solution to inference in larger and more complex MSBNs. We present an LJF local importance sampler that delivers good approximation for the posterior distribution of an MSBN subnet's local JT, as well as for the estimates of inter-agent messages.

- **Chapter 7: MA-DBN: Modeling Agents' Dynamic Evolvement** In this chapter we turn to dynamic problem domains. We present a dynamic model, MA-DBN, that supports distributed multi-agent probabilistic inference. We model the dynamics of a group of cooperative agents approximately by utilizing weak interactions among them. We show that the error resulting from such assumption of independency is bounded over time during the course of online monitoring. We also introduce a method of re-factorization to reduce the correlation between two adjacent dynamic agents.

The thesis concludes with a summary and a discussion of directions for future research.

Chapter 2

Background

This chapter presents a brief introduction to probabilistic graphical models, particularly, the Bayesian network model and the multiply sectioned Bayesian network model. We discuss major exact and approximate inference algorithms for BNs. We also introduce the secondary structure of MSBNs, a linked junction tree forest, as well as its existing inference algorithms.

2.1 Probabilistic Graphical Models

Probabilistic graphical models have become an important tool in helping an intelligent agent to reason with its uncertainty knowledge and to take proper actions. They utilize graphs to compactly represent a complex probabilistic distribution, such that data are modeled as a set of nodes representing random variables, and their connecting arcs, directed or undirected, encode the dependencies between the variables. Probabilistic graphical models combine the representation and algorithmic powers of both the probability theory and the graph theory. We will present a brief introduction that is pertinent to our work in later chapters. A more comprehensive introduction can be found in [41] [45].

2.1.1 Basic Probability Theory

Based on probability theory, a *random variable* is a variable whose outcomes (values) are given by chance. The possible outcomes of a discrete random variable are mutually exclusive and collectively exhaustive, and together as a set, called the *domain* of the variable. The probability of a random variable is measured by a function that maps each possible outcome, or *instantiation*, of this variable into the interval of [0,1].

In this thesis, we restrict our discussion to multiple-valued discrete random variables. Capital letters or indexed capital letter, such as A, B, or X_i denote random variables, unless otherwise specified. Bold capital letters, such as $\mathbf{X}$, or $\mathbf{Y}$, denote sets of variables. Bold capital letter $\mathbf{E}$ is usually used to denote the set of evidence variables. Lower case letters, such as a and x denote particular instantiation of variable A and X respectively, unless specified otherwise. Bold lower case letters, such as $\mathbf{x}$ and $\mathbf{y}$, denote particular instantiations of sets $\mathbf{X}$ and $\mathbf{Y}$ respectively. Bold lower case letter $\mathbf{e}$ is used to denote the observation for the set of evidence variables $\mathbf{E}$.

Given a set of random variables $\mathbf{V} = \{V_1, V_2, ..., V_n\}$, the probabilities of all combinations of the possible outcomes of each variable in $\mathbf{V}$ is called the *joint probability distribution* (JPD) of $\mathbf{V}$, which is denoted as

$$P(\mathbf{V}) = P(V_1 = v_1, V_2 = v_2, ..., V_n = v_n) = P(v_1, v_2, ..., v_n),$$

where $v_1, v_2, ..., v_n$ are the respective values those variables take. The *domain* of $\mathbf{V}$ is the cross join of the domains of all variables in $\{V_1, V_2, ..., V_n\}$. Each element from the domain of a set of variables is referred to as an instantiation of these variables.

The probability distribution of a subset $\mathbf{X}$ of $\mathbf{V}$ can be obtained by summing out all variables in set of $\mathbf{V}$ excluding $\mathbf{X}$ (denoted by $\mathbf{V}\backslash\mathbf{X}$):

$$P(\mathbf{X}) = \sum_{\mathbf{V}\backslash\mathbf{X}} P(\mathbf{V}),$$

where $P(\mathbf{X})$ is called the *marginal probability distribution* (MPD) of $\mathbf{X}$ from $P(\mathbf{V})$. It can also be written as $P^{\downarrow x}(\mathbf{V})$. In general, the process of summing out some variables from a probability distribution is called *marginalization*.

Given that we have some random variables observed with certain values, the probability distribution of other random variables may change. This relationship of dependency is expressed by a *conditional probability distribution* (CPD). Let $\mathbf{X}$ and $\mathbf{Y}$ be two disjoint subsets of $\mathbf{V}$ and $\mathbf{x}$ and $\mathbf{y}$ be their instantiations. The CPD of $\mathbf{X} = \mathbf{x}$ given $\mathbf{Y} = \mathbf{y}$, denoted by $P(\mathbf{X} = \mathbf{x}|\mathbf{Y} = \mathbf{y})$ and abbreviated as $P(\mathbf{x}|\mathbf{y})$, is defined as

$$P(\mathbf{x}|\mathbf{y}) = \frac{P(\mathbf{x}, \mathbf{y})}{P(\mathbf{y})}. \tag{2.1}$$

where $P(\mathbf{y}) \neq 0$.

Equation 2.1 defines the probability of $\mathbf{X} = \mathbf{x}$ given $\mathbf{Y} = \mathbf{y}$, where $\mathbf{X}$ is the *head* and $\mathbf{Y}$ is the *tail* of this CPD.

The conditional probability distribution of some variables $\mathbf{X}$ with given evidence $\mathbf{e}$, denoted as $P(X|\mathbf{E} = \mathbf{e})$, is also known as the *posterior probability distribution* of $\mathbf{X}$. In this thesis, we will consider only *hard evidence* such that each evidence is an instantiation of a variable. The marginal probability distribution can be viewed as a special case of conditional probability distribution when evidence is not yet observed for any variables. Thus, it is also referred to as the *prior marginal distribution* or just the *prior* in this thesis.

2.1.2 Dependency Model

A complete specification of JPD defines a probabilistic model for a set of random variables. However, to specify a probability model using a full JPD table is impractical. For a domain described by n boolean variables, it requires a table of size $O(2^n)$ and takes $O(2^n)$ time to process the table. By taking advantage of the dependence and independence relationship among variables, this cost can be reduced greatly.

Let $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ be disjoint subsets of $\mathbf{V}$. $\mathbf{X}$ and $\mathbf{Y}$ are *unconditionally independent* if the following holds:

$$P(\mathbf{X}|\mathbf{Y}) = P(\mathbf{X}),\ P(\mathbf{Y}) \neq 0. \tag{2.2}$$

$\mathbf{X}$ and $\mathbf{Y}$ are *conditionally independent* given $\mathbf{Z}$ if the following holds:

$$P(\mathbf{X}|\mathbf{Y}, \mathbf{Z}) = P(\mathbf{X}|\mathbf{Z}),\ P(\mathbf{Z}) \neq 0. \tag{2.3}$$

The conditional independency relationship in Equation 2.3 can be denoted as a *conditional independency statement* (CIS) $I(\mathbf{X}, \mathbf{Z}, \mathbf{Y})$ or $I(\mathbf{X}, \mathbf{Y}|\mathbf{Z})$. The unconditional independency relationship in Equation 2.2 can be denoted as CIS $I(\mathbf{X}, \emptyset, \mathbf{Y})$ or $I(\mathbf{X}, \mathbf{Y}|\emptyset)$. A dependency model is any model M of a set of variables $\mathbf{V} = \{V_1, V_2, ..., V_n\}$ from which one can decide whether $I(\mathbf{X}, \mathbf{Y}|\mathbf{Z})$ is true or not for all possible disjoint subsets $\mathbf{X}, \mathbf{Y}$ and $\mathbf{Z}$.

An easy and intuitive approach to model some dependency models is the use of directed acyclic graphs (DAG). A DAG consists of a set of nodes as the random variables, and a set of directed links between nodes but with no directed cycles. The independency relationship in a DAG can be identified by a graphical criteria called *d-separation*.

Definition 1 *D-separation*

Variables X and Y in a DAG are d-separated if for all paths connecting X and Y, there is an intermediate variable Z such that one of the following statement is satisfied.

1. *Z is the middle variable in one or a serial of diverging connections, and Z is instantiated as an evidence.*

2. *Z is the middle variable in a converging connection, and neither Z nor any of its descendants have been instantiated.*

If $\mathbf{Z}$ d-separates $\mathbf{X}$ and $\mathbf{Y}$ in the graph G, then the CIS $I(\mathbf{X}, \mathbf{Z}, \mathbf{Y})$ is said to be derived from G. A *causal network* is a directed graph constructed based on a special list of CIS called *causal input list*, where the random variables are ordered such that a cause always precedes its effect.

2.2 Bayesian Networks

*Bayesian network*s (BNs) [70] is a probabilistic graphical model for reasoning under uncertainty. It has been well accepted as a coherent and effective framework for decision support systems that must function with uncertain knowledge, such as machine learning, speech recognition, bioinformatics, error-control codes, medical diagnosis and so on.

Denoted as a triplet $B = (\mathbf{V}, \mathcal{D}, P)$, a BN consists of a set of random variables $\mathbf{V}$, a DAG $\mathcal{D}$ where each variable in $\mathbf{V}$ corresponds one to one to a node in $\mathcal{D}$. Each variable V_i in $\mathbf{V}$ is represented as a node in the DAG and is associated with a CPD $P(V_i|Pa(V_i))$, where $Pa(V_i)$ denotes the parents of V_i in the DAG. The product of these CPDs defines a JPD as:

$$P(\mathbf{V}) \;=\; \prod_{V_i \in \mathbf{V}} P(V_i|Pa(V_i)), \tag{2.4}$$

and we call this factorization (in terms of CPDs) a *Bayesian factorization*. The BN model captures the independency among random variables and provides a compact representation of JPD. Alternatively and equivalently, a BN can be defined in terms of the CPD factorization of a JPD.

Definition 2 Let $\mathbf{V} = \{V_1, \ \ldots, \ V_n\}$. Consider the CPD factorization of $P(\mathbf{V})$ as below:

$$P(\mathbf{V}) \;=\; \prod_{V_i \in \mathbf{V}, \ V_i \notin \mathbf{A_i}, \ \mathbf{A_i} \subseteq \mathbf{V}} P(V_i|\mathbf{A_i}), \tag{2.5}$$

If (1) each $V_i \in \mathbf{V}$ appears exactly once as the head of one CPD in the above factorization, and (2) the graph obtained by depicting a directed edge from vertex X to V_i for each $X \in \mathbf{A_i}$ is a DAG, $i = 1, \ \ldots, \ n$, then the obtained DAG and the CPDs $P(V_i|\mathbf{A_i})$ in Equation (2.5) define a BN. In fact, the factorization in Equation 2.5 is a Bayesian factorization of the defined BN.

The graphical structure of DAG encodes CIs that are satisfied by the JPD defined by the Bayesian factorization. In particular, the *Markov independence statement* states that every vertex V_i in a DAG $\mathcal{D}$ is independent of its non-descendants (denoted $NonDesc(V_i)$) given its parents $Pa(V_i)$, i.e., $I(V_i, \ Pa(V_i), \ NonDesc(V_i))$. We will use $Markov(\mathcal{D})$ to denote all the Markov independence statements induced by a DAG $\mathcal{D}$. Any JPD $P(\mathbf{V})$ that satisfies each CI in $Markov(\mathcal{D})$ can be factorized as a Bayesian factorization with respect to $\mathcal{D}$ and vice versa.

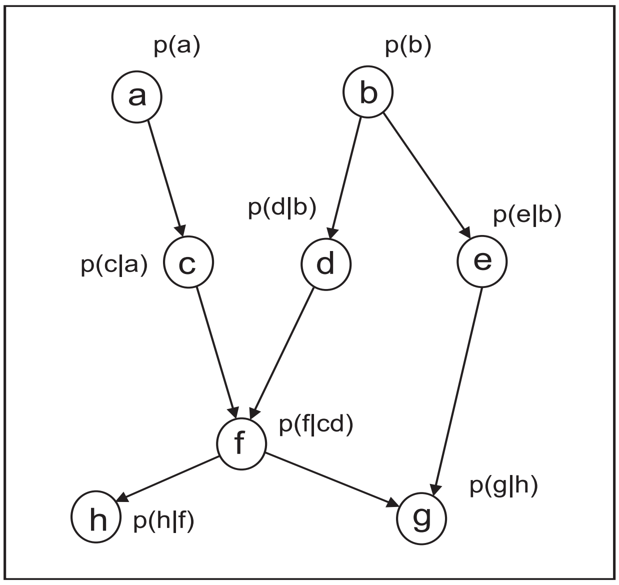

Figure 2.1: A simple BN: the Asia travel network.

Consider the Asia travel BN [52] defined over $\mathbf{V} = \{a, \ \ldots, \ h\}$. Its DAG and the CPDs associated with each node are shown in Figure 2.1. The JPD $P(\mathbf{V})$ is obtained as: $P(\mathbf{V}) = P(a) \cdot P(b) \cdot P(c|a) \cdot P(d|b) \cdot P(e|b) \cdot P(f|cd) \cdot P(g|ef) \cdot P(h|f)$. The DAG encodes CI information, for instance, given b, d and e are independent, i.e., $I(d, \ b, \ e)$; given f, h and $abcdeg$ are independent, i.e., $I(h, \ f, \ abcdeg)$.

2.2.1 Exact Inference with Junction Trees

A Bayesian network provides not only a natural and compact way to model causal structures, but also a computational basis for probabilistic inference [81]. The most common inference task performed on BNs is the calculation of posterior distribution $P(\mathbf{X}|\mathbf{E} = \mathbf{e})$ for a set of variables $\mathbf{X}$ given evidence set $\mathbf{E}$. We call the set of variables $\mathbf{H} = \mathbf{V} \backslash \mathbf{X} \backslash \mathbf{E}$ *hidden variables*. Since a Bayesian network specifies a complete representation of JPD over all random variables, any probabilistic inference can be calculated by summing out hidden variables with a sequence of multiplication and addition operations, should the full joint distribution obtained from Equation 2.4 become available. That is

$$P(\mathbf{X}|\mathbf{E}=\mathbf{e}) = \frac{P(\mathbf{X},\mathbf{e})}{P(\mathbf{e})} = \alpha P(\mathbf{X},\mathbf{e}) = \alpha \sum_{H} P(\mathbf{X},\mathbf{e},\mathbf{H}), \tag{2.6}$$

where α is a normalization value $1/P(\mathbf{e})$.

The above inference calculates the result exactly according to the probability theory. This is known as *exact* inference calculation. However, such brute force calculation is computationally infeasible. In fact, many algorithms have been developed that are based on the same notion of a BN, but with considerably different underlying concepts. A group of exact algorithms perform the task of probabilistic inference on a secondary structure of a JT. A JT is a tree graph whose nodes are subsets of the domain variables called *clusters*, or *cliques*. The steps of constructing a JT are sketched as follows.

Step 1. Moralizing the original graph: A moral graph is constructed by first connecting every pair of nodes in each node's parent set if they are not connected; then replacing the directed edges with undirected edges.

Step 2. Triangulating the moralized graph: Add necessary edges so the moral graph is triangulated. In a triangulated graph, every cycle of length 4 or more has at least one link between two non-adjacent nodes on the cycle. Different triangulation algorithm may have different results. The problem of finding an optimal triangulation is NP-complete [105], but fast algorithms that produce high quality results are available [19] [8].

Step 3. Identifying and joining cliques to construct a JT: A *clique* or just a *cluster* is a complete and maximal subgraph of a triangulated graph. Once the clusters are identified, they are arranged to form a JT. A method to construct an optimal JT is discussed in [37] where the clique intersection with largest state space (the number of configurations) is preferred at each construction step.

In a JT, an important property called the *running intersection property* holds. That is, if a variable belongs to two distinct JT clusters, then it must belong to every cluster on the path connecting the two clusters. Based on this property, we define the set of common nodes to a pair of neighboring clusters as their *separator*.

Once a JT is constructed, each JT cluster will be quantified with an initial function Φ, known as the *potential* of the cluster. [1] As each CPD $P(X_i|Pa(X_i))$ is assigned to a JT cluster containing $\{X_i\} \cup Pa(X_i)$, the initial potential of a cluster is then the product all CPD the cluster has received. If a cluster does not receive any CPD, it will be initialized to a unity potential 1.

After the JT initialization, the potential of each cluster does not represent the cluster marginal. In order to transform the cluster potential into the cluster marginal, the probability information of each cluster must be updated to be consistent to other clusters. In particular, when evidence is observed, they are entered into some clusters and need to be propagated throughout the JT. The property of running intersection ensures the coherent information exchange among JT clusters through message passings over the separators. The Hugin architecture [40] [59] and the Shenoy-Shafer architecture [82] [83] [84] are the two major variations for the JT-based exact inference calculation.

The Hugin Architecture

Under the Hugin architecture, there is a message buffer for each separator to enforce the consistency between adjacent clusters on common variables. The potential associated with each separator is initialized to unity potential 1, and the Hugin *global propagation* (GP) is carried out based on message passings. First, an arbitrary cluster in the JT is chosen as the root cluster and all the edges of the JT are directed toward the root. Then, messages between clusters are calculated and propagated in the cluster tree through two stages, namely, the inward and outward passing stages, also known as the Collect-Evidence stage, and the Distribute-Evidence stage. During the inward pass, each JT cluster passes a message to its neighbor towards the root's direction, beginning with the cluster farthest from the root. During the outward pass, each cluster in the JT passes a message to its neighbor away from the root's direction, beginning with the root itself.

[1] A potential is just a non-negative function.

Suppose a message is to be sent from cluster C_1 to cluster C_2, and the potential associated with sender C_1 is $\Phi(C_1)$. The message $M_{C_1 \rightarrow C_2}$ is calculated as

$$M_{C_1 \rightarrow C_2} = \Phi(C_1)^{\downarrow C_1 \cap C_2}, \tag{2.7}$$

where $C_1 \cap C_2$ is the set of nodes in the separator between C_1 and C_2.

In Hugin architecture, a message calculated by Equation 2.7 is not directly multiplied to the receiving neighbor, but is divided by the existing potential in the separator and then stored as a new separator potential. This updated potential on the separator is the actual value to be absorbed in the receiving cluster. We illustrate a single message passing under the Hugin architecture in Figure 2.2. Note that passing a Hugin message will result in updated potential values for the receiving cluster as well as the separator.

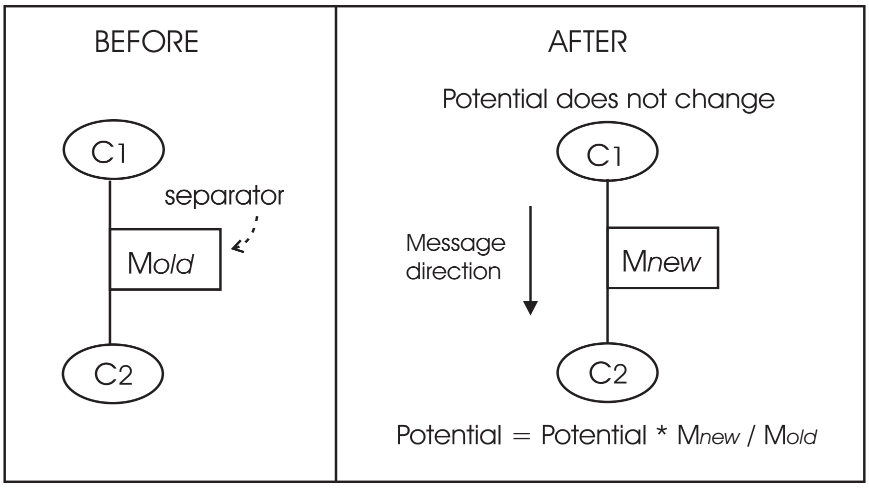
BEFORE
C1
separator
Mold
C2
AFTER
Potential does not change
C1
Message direction
Mnew
C2
Potential = Potential * Mnew / Mold

Figure 2.2: Before and after a single message passing in the Hugin architecture.

Once the GP is completed, the JPD of all variables is equal to the product of potentials on all clusters divided by the product of potentials on all separators. Meanwhile, the potential of each cluster and separator has transformed into marginal. The marginal of some variable of interest $\mathbf{X}$ can be calculated by first locating a cluster C that contains $\mathbf{X}$. Then, we marginalize the cluster potential $\Phi(C)$ on $\mathbf{X}$ to obtain

$$P(\mathbf{X}, \mathbf{e}) = \Phi(C)^{\downarrow \mathbf{X}}, \tag{2.8}$$

where $\mathbf{e}$ is the evidence incorporated before applying the GP method. Given $P(\mathbf{X}, \mathbf{e})$ and $P(\mathbf{e})$, The posterior distribution can then be calculated following Equation 2.6.

The message passing on the Hugin architecture can be improved by the method of Lazy inference [57] [58] [56]. Instead of maintaining a single potential for each cluster and separator in the Hugin architecture, we keep a multiplicative decomposition of potential tables for the clusters and separators in the Lazy inference. We can thus delay the actual combination of potential tables during the multiplication calculation, so we are able to exploit independence relations introduced by evidence during the message calculation.

The Shenoy-Shafer Architecture

In the Shenoy-Shafer architecture, the global propagation is also executed as message exchanges over the separators, but each separator is associated with two message buffers for storing messages passed in each direction between the two clusters.

With the Shenoy-Shafer message passing scheme, each cluster waits to send its message to a given neighbor until it has received messages from all *other* neighbors. Unlike the typical rooted recursive scheduling for Hugin message passings, no root cluster is selected in the Shenoy-Shafer architecture. When a cluster is ready to send its message to a particular neighbor, it computes the message by collecting all the buffered messages from its other neighbors, multiplying its own table by these messages and marginalizing the product over the separator between itself and the neighbor to whom it is sending [84].

Figure 2.3 shows the calculation of a Shenoy-Shafer message. Suppose we have a JT cluster C and N_C is the set of its neighboring clusters. The message C has received from any neighboring cluster N is denoted as $M_{N \to C}$, then the message C sent to a specific neighbor C' is given by

$$M_{C \to C'} = (\Phi(C) \cdot \prod_{N \in N_C \setminus C'} M_{N \to C})^{\downarrow C \cap C'}. \tag{2.9}$$

Note that in the Shenoy-Shafer architecture, a message calculated by Equation 2.9 is not immediately absorbed by the receiving cluster. Instead, it is stored in one of the two message buffers associated with the separator. The potential of the sender and receiver

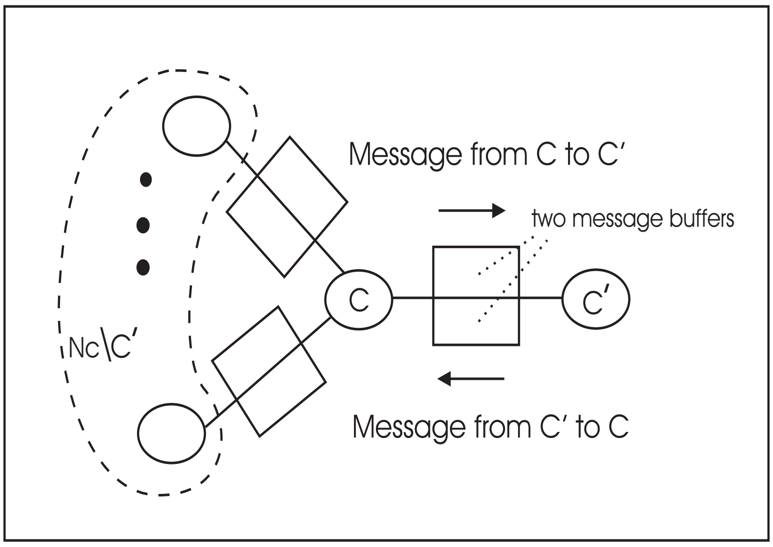

Figure 2.3: Message passing in the Shenoy-Shafer architecture.

clusters do not change during a single Shenoy-Shafer message passing.

The global propagation is completed when all clusters have sent and received messages from all their neighbors, or equivalently, when each message buffer is filled with a message. The marginal on a cluster C, with evidence $\mathbf{e}$ incorporated before the propagation, can be calculated as

$$P(C, \mathbf{e}) = P(\mathbf{V})^{\downarrow C} = \Phi(C) \prod_{N \in N_C} M_{N \to C}. \tag{2.10}$$

The original potential table and all the messages it receives from all its neighbors are multiplied together to obtain the cluster marginal. The marginal of some certain variables can be calculated by marginalizing the JT cluster that contains the variables, followed by normalization as described in Equations 2.8 and 2.6.

Both the Hugin and the Shenoy-Shafer architectures avoid the explicit calculation of the joint probability distribution over the complete domain. Instead, marginals for variables of interest are obtained through message passing and cached computation. In the Hugin architecture, the division operation is needed to calculate each message, whereas the Shenoy-Shafer requires no division but needs additional multiplication operations for all messages originated from one cluster. Essentially, the message passed between two neighboring clusters, under either the Hugin or the Shenoy-Shafer architecture, contains essentially the same information. The difference lies in the message calculation schema adopted by the two architectures: the explicit form of messages which a cluster receives does not appear in the Hugin architecture, whereas each message is individually stored in the Shenoy-Shafer architecture.

According to the comparison conducted by Lepar [53], the overall computational efficiency between the Shenoy-Shafer and the Hugin architectures depends on the structure of the original BNs. The Hugin architecture is faster on arbitrary JTs. However, the Shenoy-Shafer architecture answers queries for a wider variety of applications, and it delivers better performance in a special tree structure, binary join trees, in which each cluster has at most three neighbors [67].

Exact inference in BN is NP-hard in the worse case [16]. JT algorithms, including both the Hugin and the Shenoy-Shafer architectures, do not solve the problem of NP-hardness as well. Exponential time and space are required when these algorithms are applied to JTs that are constructed from densely connected networks. The network density is usually captured by a graph parameter called *tree-width*, which represents the size of the largest JT cluster.

2.2.2 Approximation Methods

In larger BNs, exact inference, including the JT algorithms, become impractical. An alternative is to use approximate inference to obtain a result that is close to exact. Although approximation with guaranteed error bounds is also NP-hard in worse cases [18], the class of solvable problems is wider and some approximation strategies work well in practice. Two approximation methods pertinent to this thesis are briefly introduced as follows.

Stochastic Sampling

Stochastic sampling algorithms, also known as Monte Carlo algorithms, are the most well-known class of approximation techniques. These methods generate randomly selected instantiations of the network according to the probabilistic distribution of the model. Then, the frequencies of instantiations for the query nodes are calculated as an approximation of the inference task. The accuracy depends on the size of samples; the execution time is linear in the sample size and is mostly independent of the network topology. Stochastic algorithms have a nice any-time property such that the computation can be interrupted at any given time to yield an approximation. [33].

Probabilistic logic sampling is the first and simplest sampling algorithm [34]. Samples are obtained in a BN following the directed edges from the root; any samples that are inconsistent with the observed evidence values (if any) are discarded. The probabilities of query nodes are obtained by counting the frequencies with which relevant events occur in the sample set. Logic sampling works very well in the absence of evidence. But with evidence, the number of accepted samples decreases exponentially with the number of evidence variables, resulting in a large amount of wasted samples.

The algorithms of likelihood weighting or evidence weighting [80] [29] solve the problem of sample waste in logic sampling. In likelihood weighting, when an evidence node is encountered in the sampling process, the observed value of the evidence node is recorded, and the sample is weighed by the likelihood of evidence conditional on the samples. Likelihood weighting algorithm can be applied in larger networks and converges faster then logic sampling. However, the main difficulty with likelihood weighting, and indeed with most stochastic sampling algorithms, is that it takes a long time to converge for unlikely events [33] [15].

Importance sampling algorithms improve these sampling approaches by using a revised sampling distribution to approximate the posterior distributions. Such a sampling distribution can be generated in many ways. A successful method is the Adaptive Importance Sampling for Bayesian networks (AIS-BN) [15], which reduces the sampling variance by learning a sampling distribution that is as close as possible to the optimal importance sampling function.

The main problem of stochastic sampling algorithms is that the convergence deteriorates when given extreme probabilities in the CPTs and the probability of evidence is low. It is also difficult to judge how close of the simulation results to the exact results, especially when the probability of evidence is very low [33].

Structure Simplification

These methods approximate the inference calculation by applying a certain level of simplification to the original model. These algorithms try to weaken or ignore some of the networks dependencies, forcing the generated dependencies of the network to result in a bounded error during the inference calculation. Such a simplification could involve reducing the cardinality of the size of JT clusters [48], or fitting parameters to a simple logistic function [62]. Another widely applied method is to reduce edges of an original network. For example, the edges representing weak dependencies can be removed to simplify the inference calculation [43].

2.3 Multi-agent Probabilistic Reasoning with MSBNs

In a multi-agent setting, the problem domain is naturally distributed among agents, and typically with increased size and complexity. Modeling such a domain as a single BN and performing inference tasks have been known to be difficult [92]. It is thus natural to consider decomposing one single large and complex domain into subdomains, which can be individually represented and managed by a relatively lightweight single agent. We assume that agents are cooperative such that they provide only truthful information about their local domains to other agents.

Multiply Sectioned Bayesian Network (MSBN) [92] [100] extends the traditional BN model from single-agent oriented paradigm to distributed multi-agent paradigm. The MSBN model is introduced based on following five basic assumptions that describe some ideal knowledge representation formalisms for multi-agent uncertain reasoning [99] [101].

1. Agent's belief is represented as probability.

2. Agents communicate their beliefs based on a small set of shared variables.

3. A simpler agent organization is preferred.

4. A DAG is used to structure each agent's knowledge.

5. An agent's local JPD admits the agent's belief of its local variables and the shared variables with other agents.

It has been shown that based on this small set of requirements, the resultant representation of a cooperative multi-agent system is an MSBN [101]. Formally, an MSBN is defined as the followings [92].

Definition 3 *Let G = ($\mathbf{V}$,E) be a connected graph, with the set of random variables $\mathbf{V}$ and connecting edges E, sectioned into subgraphs $\{G_i = (\mathbf{V}_i, E_i)\}$. Let the subgraphs be organized into an undirected tree $\mathcal{H}$ where each node is uniquely labeled by a G_i and each link between G_k and G_m is labeled by the non-empty interface $\mathbf{V}_k \cap \mathbf{V}_m$ such that for each i and j, $\mathbf{V}_i \cap \mathbf{V}_j$ is contained in each subgraph on the path between G_i and G_j in $\mathcal{H}$. Then $\mathcal{H}$ is a hypertree over G. Each G_i is a hypernode and each interface is a hyperlink.*

Definition 4 *Let G be a directed graph such that a hypertree over G exists. A node x contained in more than one subgraph with its parents $Pa(x)$ in G is a d-sepnode if there exists at least one subgraph that contains $Pa(x)$. An interface I is a d-sepset if every $x \in I$ is a d-sepnode.*

Definition 5 *A hypertree MSDAG $G = \cup_i G_i$, where each G_i is a DAG, is a connected DAG such that (1) there exists a hypertree $\mathcal{H}$ over G, and (2) each hyperlink in $\mathcal{H}$ is a d-sepset.*

Definition 6 *An MSBN M is a triplet ($\mathbf{V}$, G, P). $\mathbf{V} = \cup_i \mathbf{V}_i$ is the domain where each $\mathbf{V}_i$ is a set of variables. $G = \cup_i G_i$ (a hypertree MSDAG) is the structure in which nodes of each DAG G_i are labled by elements of $\mathbf{V}_i$. Let x be a variable and $Pa(x)$ be all the parents of x in G. For each x, exactly one of its occurrences (in a G_i containing $\{x\} \cup Pa(x)$) is assigned $P(x|Pa(x))$, and each occurrence in other DAGs is assigned a uniform potential. $\mathcal{P} = \prod_i P_i$ is the JPD, where each P_i is the product of the potentials associated with nodes in G_i. A triplet $N_i = (\mathbf{V}_i, G_i, P_i)$ is called a subnet of M. Two subnets N_i and N_j are said to be adjacent if G_i and G_j are adjacent on the hypertree MSDAG.*

An MSBN is composed of a set of BN subnets organized into a hypertree. Each agent maintains its own local BN subnet that represents a partial view of a larger problem domain. The union of all subnet DAGs must be a DAG, and these subnets are organized into a tree structure. Each hypertree node corresponds to a subnet, and each hypertree link corresponds to a *d-sepset*, which is the set of shared variables between adjacent subnets. A hyperlink renders two sides of the network conditionally independent similar to a separator in a JT. A hypertree $\mathcal{H}$ is purposely structured so that (1) for any variable x contained in more than one subnet with its parents $Pa(x)$ in $\mathcal{G}$, there must exist a subnet containing $Pa(x)$; (2) the shared variables between any two subnets N_i and N_j are contained in each subnet on the path between N_i and N_j in $\mathcal{H}$.

One small example of MSBN for digital equipment monitoring, borrowed from Xiang's paper [94], is shown in Figure 2.4. The digital equipment consists of 5 individual physical components. Each component contains the logical gates along with the labels for its input and output signals as shown enclosed in a box in Figure 2.4 (a). A set of 5 agents, each maintaining one component, cooperate to monitor the equipment and trouble-shoot. The hypertree of the constructed MSBN is shown in Figure 2.4 (b). Each agent is responsible for its own subdomain knowledge regarding the gates of the component and local observation. For example, Figure 2.5 shows a subnet maintained by one agent. Agents share information over the gates that connects two components. Thus, through limited local observation and communication, agents can cooperate to determine the current functioning status of the equipment, and identify faulty gates when mal-functioning occurs [96].

MSBNs provide a framework for uncertainty reasoning in cooperative multi-agent systems, so that a distributed problem domain can be modeled modularly and the inference to be performed coherently. MSBNs have been successfully applied in areas such as building surveillance [31], medical and equipment diagnosis [103] [96], distributed collaborative design [95] and multi-agent ambiguous context clarification in pervasive environments [3]. MSBNs also provide support for object-oriented Bayesian networks [47].

2.3.1 Linked Junction Tree Forests (LJFs)

A derived dependence structure called *linked junction tree forest* (LJF) is typically adopted for distributed probabilistic inference in MSBNs. An LJF is constructed through a process of cooperative and distributed compilation so that each hypernode in a hypertree $\mathcal{H}$ is transformed into a local JT and each hyperlink into a *linkage tree*.

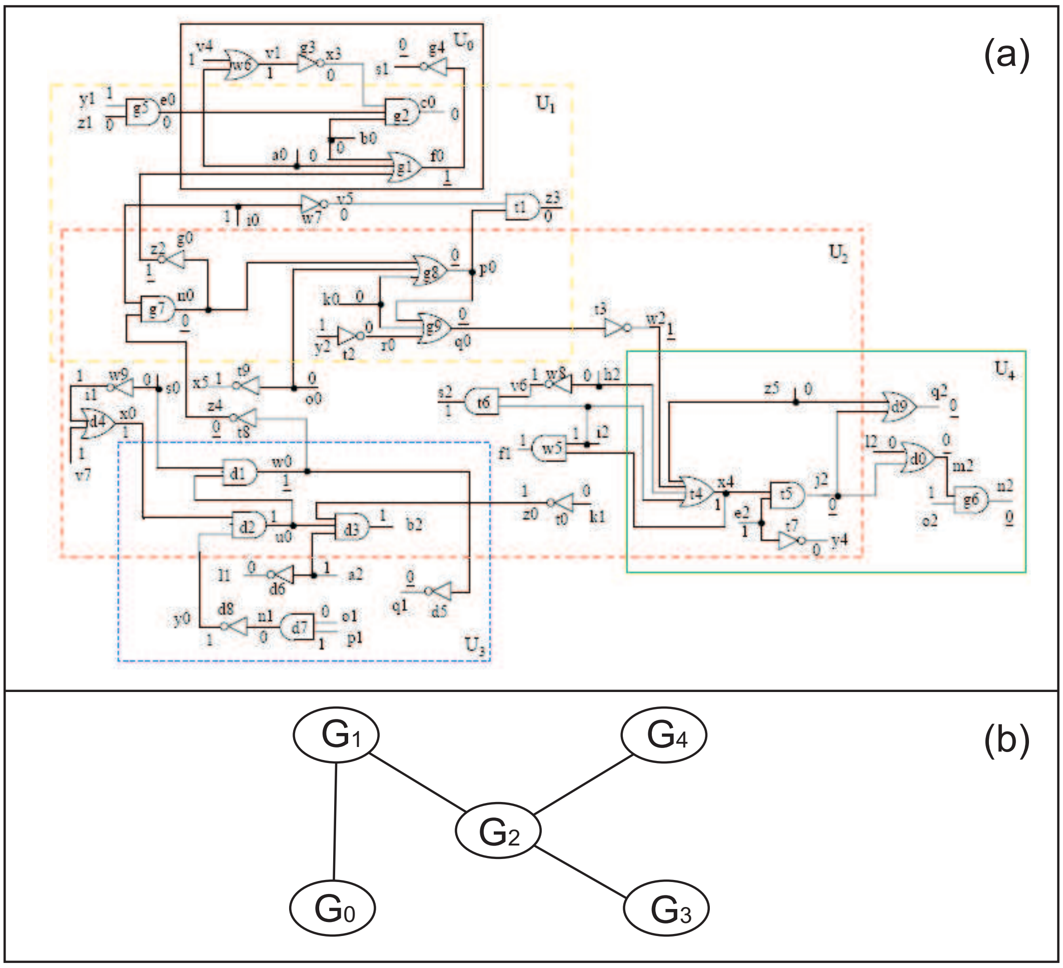

Figure 2.4: An example of digital equipment monitoring system. (a) The equipment, (b)The hypertree of corresponding MSBN.

A linkage tree is just a special name for the JT constructed from a d-sepset. In a linkage tree, each cluster is called a *linkage*, and each separator, a *linkage separator*. The cluster in a local JT that contains a linkage is called a *linkage host*. Two adjacent subnets will each maintain its own linkage tree that corresponds to the same d-sepset. These two linkage trees may be different, but only at their topologies. As they span over the same d-sepset and have an identical set of clusters and separators, it has been proven that the result of message passing is not affected [92].

For example, the three BN subnets, namely G_0, G_1, and G_2 in Figure 2.6 (b), together

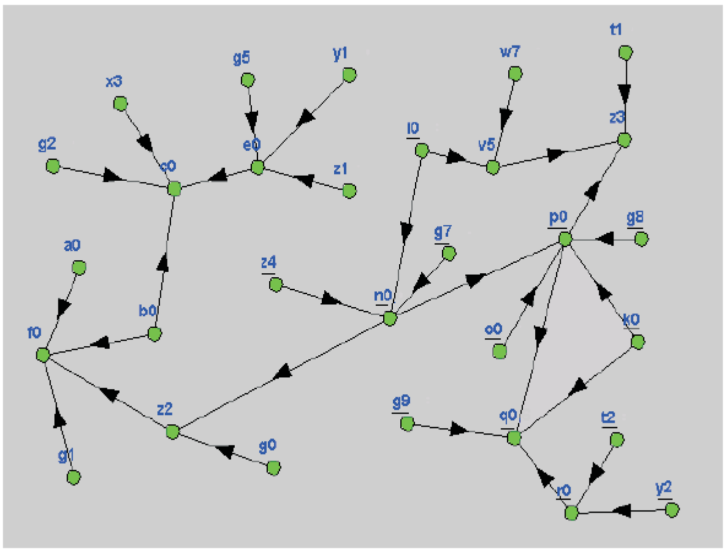

Figure 2.5: One subnet of the MSBN in Figure 2.4.

with the hypertree Figure 2.6 (c), comprise an MSBN. Two hyperlinks in Figure 2.6 (c) consist of the d-sepsets $\{a, b, c\}$ and $\{a, b, c, d\}$. The subnets are maintained by agents A_0, A_1 and A_2 respectively. Note that the union of the three DAGs in Figure 2.6 (b) gives rise to the DAG in Figure 2.6 (a).

Figure 2.7 shows an LJF constructed from the MSBN in Figure 2.6 (b) and Figure 2.6 (c). Local JTs, T_0, T_1 and T_2 constructed from BN subnet G_0, G_1 and G_2 respectively, are enclosed by boxes with solid edges. Linkage trees, converted from d-sepsets, are enclosed by boxes with dotted edges. Note that each pair of adjacent subnets maintain the identical linkage trees in this example. The linkage tree L_{02} contains two linkages abc and bcd and their linkage hosts in T_0 are the clusters $\{abc\}$ and $\{bcd\}$.

During the initialization of an LJF, exactly one of all occurrences of a variable x (in a subnet containing $\{x\} \cup Pa(x)$), is assigned the CPD $P(x|Pa(x))$. All other occurrences are assigned a unity potential. Also, a unity potential is assigned to each separator (in each local JT) and each linkage (in each linkage tree). Thus, the initial potential of a local JT cluster is either the product of all of its assigned CPDs, or unity potential 1 if no CPD is assigned. An example of the initialization of an LJF is shown in Figure 2.7. Within the total seven occurrences of the variable b in all three subnets, only one occurrence, which is in

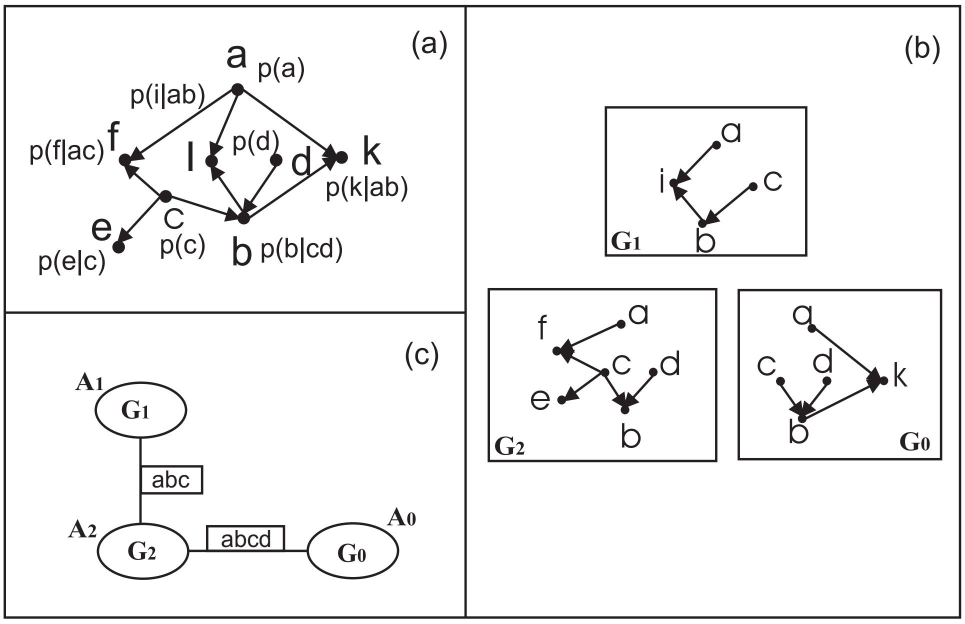

Figure 2.6: (a) A BN. (b) A small MSBN with three subnets. (c) The corresponding MSBN hypertree.

cluster bcd of G_2, is assigned the CPD $P(b|cd)$. All other occurrences are assigned a unity potential (not shown in the figure). In most cases, the initial potential does not provide the complete information for an agent to correctly reason about its own problem subdomain. This is because the local JTs are not yet consistent, but more importantly, the potential of each subnet does not represent the prior marginal distribution, which is the JPD of the local variables without any observed evidence.

For example, the initial potentials of all three subnets in Figure 2.7 are: $\Phi(G_0) = P(a) \cdot P(c) \cdot P(d) \cdot P(k|ab)$, $\Phi(G_1) = P(i|ab)$ and $\Phi(G_2) = P(b|cd) \cdot P(e|c) \cdot P(f|ac)$. None of these potentials forms the JPD over the corresponding local variables. Even though local consistence can be achieved through message passing in the local JT, inter-agent communication is necessary to provide each MSBN subnet the missing information to form the prior marginal. This process is known as *marginal calibration*.

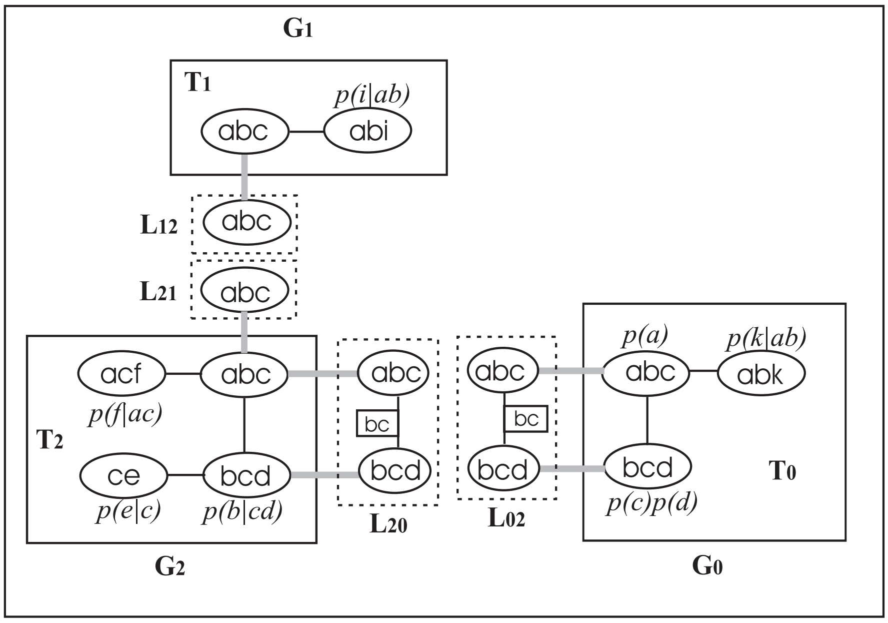

Figure 2.7: An LJF constructed for the MSBN in Figure 2.6.

2.3.2 MSBN Distributed Inference

A major task of MSBN inference is to supply the correct global posterior probabilistic knowledge to each agent given some locally observed evidence. The most dominant group of algorithms for distributed probabilistic inference in an MSBN utilize secondary inference structures [92, 93, 98]. Among them, the LJF-based algorithms, extending from JT-based inference methods for single-agent BNs, have proven to be the most successful [92]. LJF-based inference algorithms are superior to methods based on global loop-cutset conditioning and global stochastic sampling [93], as they provide a higher level of autonomy than those alternatives.

However, the existing LJF-based algorithms are based mostly on the extension from Hugin message passings. The typical propagation process requires two rounds of global messages exchange in the corresponding LJF. This can be viewed at a high level as the Hugin message propagation in the context of an MSBN.

Within an LJF framework, inter-agent messages passed between two adjacent agents are calculated over their connecting linkage tree. [2] During LJF global propagation, inter-agent

[2]A detailed discussion of the linkage tree construction will be presented in Chapter 3.

messages are passed recursively inward and outward, relative to a randomly selected root subnet. During the inward message passing, each agent passes a message to its neighbor toward the root's direction, starting with the leaf subnets. During the outward passing, each agent passes a message to its neighbor away from the root's direction, originating from the root itself.

With the existing Hugin-based methods, an agent's local computation is costly. The consistency of the local JT must be achieved in the sender before passing an inter-agent message, and also in the receiver afterward. This local consistency is obtained through the standard JT message passing in the LJF local JT. The number of such local updates in an MSBN subnet is not relative to the root selection during the LJF global propagation, but depends on the hypertree topology. In fact, the existing Hugin-based methods all require *repeated* updates of the local beliefs. Whenever an agent receives an inter-agent message, a full round of message passing in its local JT must be performed, which includes two complete stages of message passings (inward and outward) among the local JT clusters,

2.4 Discussion

A BN, as an example of graphical probabilistic model, has been well accepted as a powerful tool for uncertainty reasoning. Although exact inference as well as approximate inference with guaranteed precision are NP-hard, there have been many practical algorithms that can solve a wide range of inference tasks.

As an extension to the original BN model, the MSBN provides a specific model for probabilistic reasoning in multi-agent systems. A large and distributed problem can be modeled as a set of organized subdomains, following certain constraint of domain sectioning. As an MSBN maintains a hypertree structure, the BN JT inference algorithm is naturally extended to an MSBN secondary structure LJF. Essentially, the current inference algorithms in LJF follows the same principle of the Hugin message passing of a JT, but carries out the inference calculation in two levels, the LJF global level and the LJF local JT level. Although exact results are guaranteed, these algorithms could be too costly to be applied in larger systems. In particular, the local computational cost at some MSBN subnets may become significantly high, possibly halting the LJF global inference.

To date, there has been little research of extending non-Hugin message passings to the MSBN context. Despite the similarities of the tree structure in both a BN JT and an MSBN hypertree, current research has not offered effective solutions for LJF global inference. In the next chapters, we will investigate methods to improve the existing distributed inference calculation by adopting some novel inference architectures. Furthermore, we will use approximation techniques, such as stochastic sampling and model simplification, to trade in certain levels of accuracy for more efficient computation.

Chapter 3

An Improved LJF Message Passing Architecture

The existing LJF inference algorithms are extensions to the Hugin-based message passing architecture. They are similarly composed of rooted, recursive message passing schema at both the global and the local levels [92]. The global inference is first called upon a randomly selected agent, and two rounds of inter-agent messages passing are recursively carried out among all agents in a restricted order. A message passed between two adjacent agents, known also as an *external message* or a *global message*, is transmitted over the linkage trees between the agents. The update of local belief during the local propagation involves passing of intra-agent messages, each known as an *internal message* or a *local message* in the LJF local JT[1].

The original Hugin architecture supports the inference computation in a BN JT with exact results, but its extended version in MSBN LJFs could incur an extensive amount of internal messages passing. This is because each Hugin message is no longer passed between two JT clusters, but instead between two MSBN subnets each with its own internal structure of a local JT. Upon receiving an incoming inter-agent message, an agent must absorb it immediately, followed by an update of its local belief that involves two rounds of intra-agent message passings. Also, with the Hugin-based propagation, it is well-known

[1]Hereinafter, we use inter-agent, external and global message, intra-agent, internal and local message interchangeably

that periodical off-line time prevents each agent from observing local evidence continuously [88]. The recent improvement of the LJF Hugin-based method extends from the lazy inference algorithm [58]. The calculation of extended linkage potentials is more efficiently carried out with a lazy-based division [98]. Nevertheless, the local computation remains costly, as an agent with s neighbors will still conduct s times local belief updates, each with a complete round of inward and outward local message passings.

Our goal is to improve the local computational efficiency for LJF global propagations. We argue that, as an LJF represents a high level JT, the inference algorithm for LJFs should not be restricted to the extension of a certain class of JT inference algorithms, i.e. the Hugin architecture. In particular, by adopting some new message calculation and passing scheme, we could avoid the repeated local updates.

In this chapter, we introduce an LJF-based inference architecture extending from the Shenoy-Shafer message passing. The main obstacle to such an integration of the Shenoy-Shafer message passing and LJF is the usage of message buffers. Since Shenoy-Shafer messages must always be buffered, an earlier attempt of Shenoy-Shafer extension is applied on a special secondary structure of an MSBN, named double linked junction tree forest (DLJF) [97]. A DLJF provides the needed message buffers, but its construction requires a more sophisticated, message direction dependent compilation process compared to the construction of an LJF. More importantly, extra storage is needed in a DLJF since we need to maintain two sets of JTs (as the term "double" stands for) in order to pass the two messages between each pair of adjacent agents.

Our new approach aims to fully utilize the existing LJF structure. The reduction of local computation are realized through 1) the adopting of the Shenoy-Shafer architecture to avoid repeated local updates, and 2) the introduction of partial propagation to limit the number of needed local message passing during each local update. In order to support the Shenoy-Shafer message calculation, we treat each linkage tree as a message buffer. Given that an incoming message will be buffered, no repeated local propagation is conducted. Moreover, we introduce a new method of partial local update, which greatly reduces the amount of intra-agent messages while maintaining the correctness of inter-agent message passings and LJF global propagation. A non-rooted inter-agent message scheduling scheme is adopted so that the calculations of inter-agent messages are not carried out recursively based on a given root, but initiated simultaneously at all nodes. Our asymptotic analysis shows that the time complexity of our new message passing architecture is superior than the current Hugin-based as well as the Lazy extension to LJF inference methods.

Later in this chapter, we introduce an extension of our new global propagation architecture to iterative message passing. Since more than one message are passed toward each direction between a pair of adjacent agents, a temporary missing inter-message will not halt the global propagation. It has been shown that the current Hugin-based message passing architecture cannot be extended successfully to support exact calculation with an iterative scheme [2], whereas our message oriented LJF architecture provides the needed support for such extension.

Under our iterative scheme, each agent performs the local calculation in complete parallel with other agents. Inter-agent messages are delivered iteratively and as batch. An agent has more autonomy as to decide the time interval for processing received messages and perform local updates. Since the iterative message passing is conducted among agents organized into a hypertree structure, the convergence of inter-agent messages is guaranteed. Meanwhile, each agent's local belief can be obtained exactly. Such a scheme could be more costly in terms of message calculation, but it is more suitable in a multi-agent environment with unreliable communication channels due to its fault tolerance ability.

3.1 Hugin-based Recursive Inference

3.1.1 Linkage Tree as Separator

In an MSBN, the exchange of the shared information of adjacent agents is through messages passed over their corresponding d-sepset. A linkage tree is an alternative representation of the d-sepset between adjacent agents in an LJF [89]. A linkage tree is constructed with an explicit internal structure, containing clusters (linkages) smaller than the domain of d-sepset in order to reduce the cost of message calculation. A linkage tree, a linkage and a linkage host are defined by Definition 7 [92].

Definition 7 *Let I be the d-sepset between JTs T_i and T_j in an LJF. A linkage tree L of T_i with respect to T_j is constructed as follows:*

Initialize L to T_i. Repeat the following on a cluster of L until no nodes can be removed:
1. Remove a node $x \notin I$ if x is contained in a single cluster C.
2. If C becomes a subset of an adjacent cluster D after step 1., union C into D.

Each cluster l in L is a linkage. Define a cluster in T_i that contains l as its linage host and break the tie arbitrarily.

Current LJF inference algorithms [88, 92] are extensions of the Hugin architecture. Inter-agent messages passed between two adjacent agents are calculated over their connecting linkage trees, which are used as a Hugin separator. For an LJF local JT T_i to deliver a message to T_j over T_i's linkage tree L_{ij}, each linkage Q_i in L_{ij} is assigned first a *linkage potential*, which is

$$\Phi(Q_i) = \sum_{C_i \setminus Q_i} \Phi(C_i), \tag{3.1}$$

where C_i is Q_i's linkage host in T_i.

For example, consider the LJF with local JTs and linkage trees shown in Figure 3.1. The message originated from G_0 to be delivered to G_2 is calculated over the linkage tree L_{02}, and should consist of the potentials over two linkages abc and bcd.

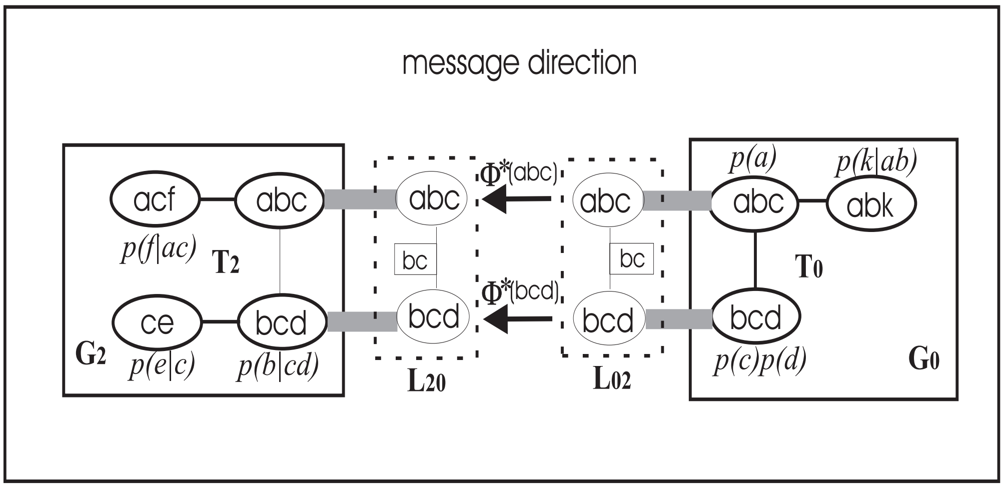

Figure 3.1: Message Calculation over an LJF.

As the linkage tree construction enables a more compact representation of d-sepset, it also introduces redundancy over linkage separators. This is because the separator nodes in the linkage tree may appear more than once in different linkages. For example, consider again the example in Figure 3.1. Using Equation 3.1, we have linkages abc and bcd both carrying information over bc, causing errors in the message propagation.

In order to solve this problem, the concept of *extended linkage potential* is introduced [92]. That is, we first randomly select a linkage as the root linkage and direct all linkages accordingly. Then, we associate each linkage separator with one of its two neighboring linkages, which is farther away from the root linkage, as the linkage's peer separator. The extended linkage potential is defined as follows:

Definition 8 *Suppose in a linkage tree, for each linkage Q with peer separator R, the extended linkage potential is defined as follows:*

$$\Phi^*(Q) = \Phi(Q)/\Phi(R) \tag{3.2}$$

for all non-root linkages, and

$$\Phi^*(Q) = \Phi(Q) \tag{3.3}$$

if Q is the root linkage.

Therefore, as the extended linkage potential for root linkage remains the same, the redundancy of the separator information over other linkages is removed by division [92]. For example, in linkage tree L_{02} from Figure 3.1, we can select the cluster abc as the root linkage, then associate the separator bc with the linkage bcd as the peer separator. Meanwhile, abc has no peer assigned. The extended linkage potentials on each linkage are $\Phi^*(abc) = \Phi(abc)$ and $\Phi^*(bcd) = \Phi(bcd)/\Phi(bc)$. The extended linkage potential defined in Equation 3.2 is actually used to calculate external messages.

The delivery of an inter-agent message is done through the passing of extended linkage potentials of each linkage over the corresponding linkage tree. An operation, named $Absorb\_Through\_Linkage\_Orig$ [96], is used to propagate the belief from one linkage host in the sender agent to the other linkage host in the receiver agent. The operation is described in the following algorithm.

Algorithm 1 $Absorb\_Through\_Linkage\_Orig$

Let Q be a linkage in a linkage tree L between two LJF local JTs T_i and T_j, and let C_a and C_b be the corresponding linkage host of Q in T_i and T_j. Let $\Phi^*_Q(Q)$ be the extended linkage potential associated with Q, and $\Phi^*_{C_b}(Q)$ be the extended linkage belief on Q defined in C_b. When Algorithm $Absorb\_Through\_Linkage\_Orig$ is called on C_a to absorb from C_b through Q, the following operations are performed.

1. Updating host belief: $\Phi'_{C_a}(C_a) = \Phi_{C_a}(C_a) * \Phi^*_{C_b}(Q)/\Phi^*_Q(Q)$

2. Updating linkage belief: $\Phi^{*\prime}_Q(Q) = \Phi^*_{C_b}(Q)$

Algorithm $Absorb\_Through\_Linkage\_Orig$ delivers a message, in the form of an extended linkage potential, through a single linkage. The message is immediately absorbed (multiplied) into the potential maintained at the linkage host of the receiver agent. The linkage potential receiver host is updated to be consistent with the sender host. Such consistency between the linkage host and the linkage is defined as follows.

Definition 9 *Let Q be a linkage between two LJF local JTs T_i and T_j. Let C_a be the linkage host of Q in T_i. C_a and Q are said to be consistent if $\sum_{C_a \setminus Q} \Phi(C_a) = \Phi_Q(Q)$.*

Note that in Definition 9, $\Phi_Q(Q)$ is linkage potential, not the extended linkage potential used during the message passing. After Algorithm $Absorb\_Through\_Linkage\_Orig$ is performed in the sender host, C_b and l are consistent. Also, if C_a and Q are consistent before the operation is performed, the same consistency is maintained afterward as well. In fact, although each of the two adjacent agents maintains its own linkage tree, the two linkage trees have exactly the same value. They are used in the place of a separator of the Hugin architecture.

An actual external message is delivered over a linkage tree, which is constructed typically with multiple linkages. Algorithm $Update\_Belief\_Orig$ propagates belief from a local JT to another adjacent local JT through a set of extended linkage potential.

Algorithm 2 $Update\_Belief\_Orig$

Let T_i and T_j be two adjacent local JTs, and L be the linkage tree between them. When $Update\_Belief\_Orig$ is called on T_i relative to T_j, the following is performed:

1. *For each linkage Q in L, absorb the corresponding extended linkage potential by calling the linkage host of Q in T_i to perform $Absorb\_Through\_Linkage\_Orig$.*
2. *Perform a full round of local propagation in T_i.*

It is important to note that the delivery of each inter-agent message will trigger local propagation, which is typically conducted as standard BN JT inference [96]. We call such local propagation in an LJF local JT as agent's *local updates*.

3.1.2 Rooted Recursive Scheduling

The current LJF-based inference algorithms are similarly composed of a rooted, recursive message passing schema. The global inference algorithm is first called upon a randomly selected agent, and two rounds of inter-agent messages passing are recursively carried out amongst all agents in a restricted order. During the inward message passing, each agent in the hypertree passes an external message to its neighbor *toward* the root's direction, beginning with the hypertree's leaf nodes. During the outward passing, each agent in the hypertree passes a message to its neighbor *away* from the root's direction, beginning with the root itself. This scheduling scheme is realized with the following set of algorithms [96].

Algorithm 3 $Collect\_Belief\_Orig$

Suppose an agent A with a local JT T. When $Collect\_Belief\_Orig$ is called in A, A does the following:

1. *If A has no neighbor except the caller, it performs local propagation and returns.*
2. *Otherwise, for each adjacent local JT T' except the caller, call $Collect\_Belief\_Orig$ in T'. After T' finishes, T performs $Update\_Belief\_Orig$ relative to T'.*

Algorithm 4 $Distribute\_Belief\_Orig$

Suppose an agent A with a local JT T. When $Distribute\_Belief\_Orig$ is called in A, A does the following:

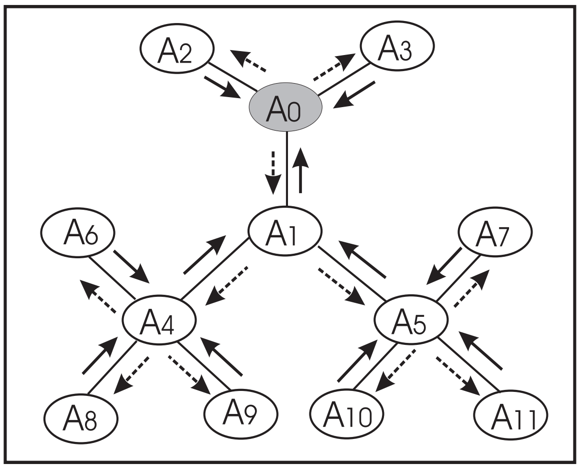

Figure 3.2: Inter-agent message passing in MSBNs.

1. *If the caller is a local JT, performs $Update\_Belief\_Orig$ relative to the caller.*
2. *For each adjacent local JT T' except the caller, call $Distribute\_Belief\_Orig$ in T'.*

Algorithm 5 $Communicate\_Belief\_Orig$

When $Communicate\_Belief\_Orig$ is initiated at an LJF, $Collect\_Belief\_Orig$ is called at any chosen local JT T, followed by a call of $Distribute\_Belief\_Orig$ at T.

Algorithm $Communicate\_Belief\_Orig$ brings an LJF into global consistency such that all local JTs are consistent and each linkage tree is consistent with each of the two corresponding local JTs as defined in Def. 9. The LJF global propagation is analogous to the Hugin-based GP method in a BN JT. While the above set of algorithms realize exact inference in an LJF, its drawback is the cost for each agent to maintain local consistency. An agent must perform multiply times of local updates during the LJF global message propagation.

For example, in Figure 3.2, suppose the shaded node represent an agent A_0 and is selected as the root node. The solid arrows indicate the direction of inward messages flow starting from all the leaf nodes, and the dashed arrows indicate the direction of outward messages flow originated from A_0. Consider the stage of inward message passing. A_4 receives one inter-agent message from each of the three neighboring agents, A_6, A_8 and A_9. These three nodes must maintain their local consistency through local message passing before the inter-agent message to A_4 can be calculated. The receiver node A_4 will need to update its local belief and maintain its own consistency upon the arrival of *each* inter-agent message, for a total of three times, in order to prepare an outgoing message to be delivered to A_1. Next, during the outward message passing initiated at A_0, A_4 will receive an incoming message from A_1, which is absorbed by A_4 by another round of local propagation. Afterward, since A_4 has received all incoming messages, its local belief become globally consistent as well locally consistent.

3.2 An Improved LJF Inference Architecture

Aiming at improving the efficiency of local computation, we introduce a new architecture for LJFs global inference, which extends from the Shenoy-Shafer architecture. Compared with the Hugin calculation, Shenoy-Shafer messages between JT clusters are stored in message buffers, rather than multiplied directed into the receiver cluster. The belief of each JT cluster is consistently updated in the Hugin architecture, whereas the Shenoy-Shafer architecture is message-oriented, and the original belief of each cluster remains unchanged. With the latter, the JPD of a JT cluster is obtained by an explicit call to multiply all buffered messages with the original potential of the cluster.

The Shenoy-Shafer architecture has been shown to out-perform the Hugin architecture, particularly in a special JT structure named binary joint tree [53]. Although an MSBN hypertree does not resemble a binary joint tree, we extend the Shenoy-Shafer message passing in order to take the advantage of delayed manipulation of inter-agent messages. Such an approach will results in reduced number of local updates currently needed in the Hugin architecture. So far, the only extension of the Shenoy-Shafer architecture to MSBNs is applied on a special secondary structure: a double linked junction tree (DLJF) [97]. We present a new architecture that allows the Shenoy-Shafer message passing based on a standard LJF.

3.2.1 Linkage Tree as Message Buffer

In an LJF, each of the two adjacent agents maintains its own linkage tree. The linkage trees are constructed over the same d-sepset, resulting in the same set of linkages and linkage separators. Currently, the potential stored in the two linkage trees are identical during the belief propagation, these two separate linkage trees are used to serve the purpose of a single Hugin separator. Indeed, this treatment guarantees the correctness of the Hugin message passing, but each incoming message must be immediately processed followed by a complete round of inference in the local JT, as described in Algorithm $Update\_Belief\_Orig$.

The main novel idea of our architecture lies in the usage of linkage trees as message buffers. In order to conduct the Shenoy-Shafer message calculation in JTs, two message buffers must be allocated for each pair of adjacent clusters. Similarly, we need two message buffers for each adjacent pair of agents in an LJF to store the messages for both directions. Since an agent maintains a linkage tree to each of its neighboring agents, there are two linkage trees associated with each pair of adjacent agents. Thus, naturally enough, we use each individual linkage tree as a message buffer, such that a message delivered from a neighboring agent can be buffered into the linkage tree corresponding to that agent. We describe the delivery of an inter-agent message over a linkage tree as follows.

Algorithm 6 $Deliver\_Through\_Linkage$

Let T_i and T_j be two adjacent local JTs in an LJF, and let their corresponding linkage trees be L_i and L_j. Let Q_i^k, Q_j^k (k=1,...,n) each be a pair of corresponding linkages in L_i and L_j, and C_i^k and C_j^k be their linkage hosts in T_i and T_j. $\Phi(Q_j^k)^{cur}$ is the current potential in linkage Q_j^k. When Algorithm $Deliver\_Through\_Linkage$ is called on T_i to obtain a message from T_j, the following is performed:

```
1. For each linkage $Q_j^k$ in the linkage tree $L_j$
2.       Calculate the extended linkage potential $\Phi^*(Q_j^k)$ from linkage host potential $\Phi(C_j^k)$;
3.       Compose the message potential as $\Phi(Msg) = \Phi^*(Q_j^k)/\Phi(Q_j^k)^{cur}$;
4.       Assign linkage potential $\Phi(Q_i^k) = \Phi(Msg)$;
5.       Maintain the current belief of linkage hosts $\Phi(C_i^k)$ and $\Phi(C_j^k)$;
6. End for
7. Flag the linkage tree $L_i$ as message received;
```

The operation defined in Algorithm $Deliver\_Through\_Linkage$ delivers an inter-agent message that is still composed of extended linkage potential. Compared with Algorithms $Absorb\_Through\_Linkage\_Orig$ and $Update\_Belief\_Orig$, a message passed with Algorithm $Deliver\_Through\_Linkage$ is only buffered without being absorbed into the potential of the corresponding linkage host. Note the actual calculation of the message includes a division to remove the potential initially assigned to the linkage tree. This is due to the new local update schema that will be introduced in the next sections.

We illustrate our inter-agent message calculation and buffering with an example. Consider an agent A_0 with its local JT T_0 and n neighbors A_1,..., A_n shown in Figure 3.3. The inter-agent message passings are marked with arrows. Linkage trees are used as buffers, e.g. buf_1, buf_2,..., buf_n to store messages coming from A_0's n adjacent agents.

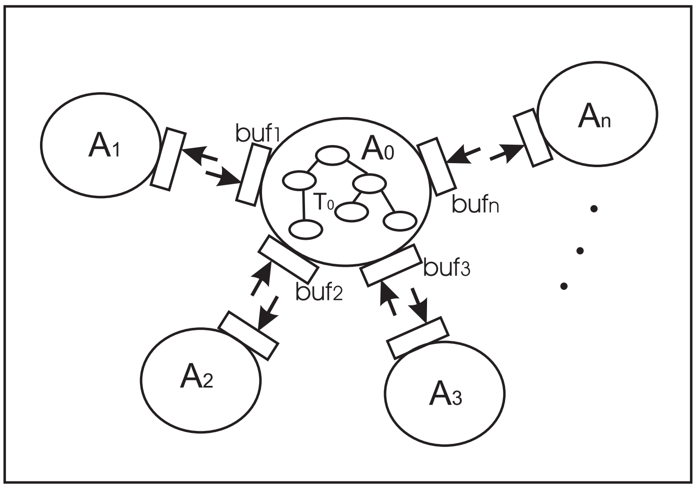

Figure 3.3: Incoming message buffers. The agent A_0 maintains n message buffers each responding to an adjacent agent.

In our new architecture, the local belief update is not triggered by a new incoming message. Instead, an agent decides when the local message calculation is to be performed following certain global message passing protocols (which will be presented Section 3.2.3). An update of local belief can be achieved with Algorithm $Update\_Belief$.

Algorithm 7 $Update\_Belief$

Let T be a local JT of an agent A with n adjacent JTs. Let L_i (i=1,...,n) be the n linkage trees maintained by T to its neighbors. In a linkage tree L_i, let Q_i be the set of linkages, and let their set of linkage hosts in T be C_{Q_i}. When Algorithm $Update\_Belief$ is called in T, A performs the following:

```
1. For each linkage tree $L_i$ that is marked with a message received
2.     For each linkage host in $C_{Q_i}$
3.         Absorb the extended linkage potential $\Phi(l)$ for each $Q \in Q_i$;
4.         $\Phi'(C_{Q_i}) = \Phi(C_{Q_i}) * \Phi(Q)$;
5.     End for
6. End for
7. Perform a full round of local propagation in T;
```

The call to Algorithm $Update\_Belief$ is issued by an agent to update the local belief with regards to the inter-agent messages that are received. The local JT will absorb all currently buffered messages, and a round of local belief update is performed. Note that although the JT is locally consistent after a call to Algorithm $Update\_Belief$, the local belief might not be globally consistent unless messages have arrived from all of the neighboring agents.

3.2.2 Message Calculation and JT Local Consistency

The goal of LJF inference is to propagate each agent's local belief over the whole network. An agent's coherent local belief, with regards to the LJF global domain, is obtained through incoming messages from its adjacent agents. Under the current Hugin-based architecture, an inter-agent message is always calculated in a locally consistent JT, as such a message is always obtained after a full round of local JT propagation is performed. However, since the agent might not have received messages from all its neighbors, local propagation performed at this moment does not guarantee global consistency. Repeated internal message passing is conducted only for the purpose of message calculation.

We try to reduce the local computational cost in terms of intra-agent (internal) message passing. We argue that local propagation, with a full round of inward and outward passing of internal message, is not always necessary. Although the correct belief of local JT depends on all the messages delivered to the JT, a message sent by that local JT actually depends on all but one of these messages. With a Shenoy-Shafer message passing scheme, we are able to obtain a message in a non-consistent JT, which is of the same content as one from a consistent local JT, through a partial belief update. We first introduce the concept of a *linkage host tree* and an *extended linkage host tree* to facilitate such message calculation.

Definition 10 *Let T_i be a local JT and a linkage tree L_i connecting to a neighboring local JT. In T_i, H is the set of linkage host clusters for L_i. Then a tree T_h constructed with H is called a linkage host tree of T_i with regards to L_i.*

Proposition 1 *A linkage host tree is a JT.*

Proof:

Based on the linkage tree property, a linkage tree is a JT representation of the d-sepset. Also, each of the linkage tree clusters is a linkage, and each linkage corresponds to a linkage host, which is a cluster of the local JT. A linkage host cluster may contain additional nodes to its corresponding linkages, and these nodes are not d-sepset nodes.

If we construct a tree T_h with linkage host clusters following the same structure of the linkage tree, the running intersection property for each d-sepset is preserved since the d-sepset is not changed in T_h. Also, as in T, all clusters are local JT clusters, so the running intersection property holds also for none d-sepset nodes. Therefore, the linkage host tree T_h is also a JT. □

Definition 11 *Let T_i be a local JT and its linkage host tree T_h that corresponds to a linkage tree L_i. The minimum subtree of T_i that contains all clusters in T_h is called an extended linkage host tree T_e of T_i with regards to L_i.*

Note that a linkage host tree is a conceptually defined structure that contains *only* linkage host clusters. It may not be a subtree of the local JT. Meanwhile, an extended linkage host tree is a subtree within the local JT, but it may contain clusters other than the linkage hosts for the connecting linkage tree.

For example, consider the local JT T_0 with the root cluster C_r shown in Figure 3.4. The linkage hosts for linkage tree L_0 are clusters C_r, C_1 and C_3 each with a bold border. These three clusters can construct a JT which is the linkage host tree of T_0 with regard to L_0 and is not shown in the figure. Also, C_r, C_1 and C_3 together with a non-host cluster C_2, construct an extended linkage host tree T_e as a subtree of T_0. T_e is shown in shaded area in T_0.

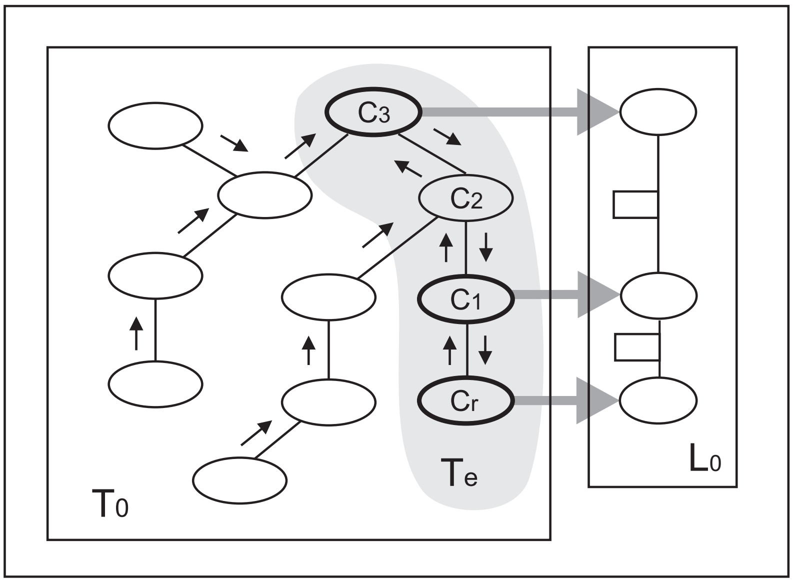

Figure 3.4: An example of partial propagation for calculating a single outgoing message. Shown with the local JT T_0, linkage tree L_0 and extended linkage host tree T_e.

In our new architecture, an agent A_i can deliver a single outgoing message to a neighboring agent A_j, if A_i has received all incoming message from its other neighbors except A_j. Such a message is composed over extended linkage potential from the corresponding linkage hosts that are locally consistent with all the messages that A_i has received. Typically, this can be achieve by calling Algorithm $Update\_Belief$. However, we show that such a complete round of local belief update can be avoided. An algorithm designed to obtain an outgoing inter-agent message with only a partial local propagation is shown as follows.

Algorithm 8 Cal_Single_MSG

Let A_i and A_j be two adjacent agents. A_i's local JT is T_i with the set of clusters C. T_i maintains a linkage tree L to A_j. In T_i,, H is the set of linkage hosts of L and the *extended linkage host tree is T_e. When called, an external message passed from A_i to A_j is calculated as the follow steps:*

1. For all linkage trees of A_i connecting to neighboring agents except A_j, absorb the extended linkage potential and update the belief on corresponding linkage hosts.

2. Randomly select a linkage host $C_r \in H \subset C$ as the root of T_e as well as T_i. Direct all clusters away from C_r in T_i.

3. Perform a full inward message passing on C_r in T_i, such that C_r calls recursively all child clusters to send an inward message.

4. Perform a partial outward message passing on C_r within the context of T_e, such that C_r sends outward messages to all linkage hosts recursively.

5. For each linkage in L, obtain corresponding extended linkage potential.

Consider again Figure 3.4. Suppose we need to calculate the outgoing message of local JT T_0 over linkage tree L_0. After step 1 of Algorithm Cal_Single_MSG, we select C_r and perform an inward message passing toward C_r in the local JT T_0. Each inward message, marked by an arrow in the figure, shows that the messages are passed from all leave nodes first recursively toward the root node in the whole local JT. Next, an outward message passing is originated from root C_r, but with outward messages reaching only the clusters within the extended linkage host tree T_e. Once this partial message passing is over, we can calculate the outgoing message over L_0 through the linkage host of each linkage in L_0 by step 5 of the algorithm, as shown with the shaded thick arrows in the figure.

The following proposition shows that an inter-agent message calculated with Algorithm Cal_Single_MSG is consistent to the JT's local belief.

Proposition 2 *Let T over the set of local variables N be a local JT of an agent A. Let L be T's linkage tree connecting to an adjacent agent A' over their d-sepset I. The extended linkage host tree is H, with C_r being the root cluster. For each linkage $Q \in L$, let $\Phi^*(Q)$ be the extended linkage potentials. After a call of Cal_Single_MSG to calculate an inter-agent message to A', we have*

$$\prod_{Q \in L} \Phi^*(Q) = const \sum_{N \setminus I} \Phi(N)$$

where $\Phi(N)$ is the local belief including its initial belief and all messages received except the one from A'.

Proof:

First, consider the root cluster C_r in linkage host tree H as well as local JT T. After step 2 of Cal_Single_MSG, the messages except from A' are absorbed. Next, an inward propagation is performed in the local JT with regard to C_r as the root. Therefore, the potential associated with root cluster C_r defines the marginal of $\Phi(N)$ onto C_r. For C_r's corresponding linkage Q_{C_r} in L, $\Phi(Q_{C_r}) = const \sum_{N \backslash Q_{C_r}} \Phi(N)$.

Next, consider all other linkage host clusters in H. Step 4 performs a partial outward JT message passing within H. After each $C_i \in H$ has received an outward message, the potential associated with C_i defines the marginal of $\Phi(N)$ onto C_i, and we have $\Phi(Q_{C_i}) = const \sum_{N \backslash Q_{C_i}} \Phi(N)$.

Since all clusters in H are consistent with the local belief, the correct linkage potentials can be obtained. Therefore, based on the definition of extended linkage potential, we have $\prod_{Q \in L} \alpha(Q) = const\Phi_L(I) = const \sum_{N \backslash I} \Phi(N)$.

$\square$

Note that we need to always choose a linkage host as the root cluster during the propagation. Given such a root, we conduct a full inward passing so the local messages carrying the probability information of all JT clusters flow towards the root. Next, a partial outward passing is performed only in the context of the extended linkage host tree to distribute belief originated from the root within the clusters of the extended linkage host tree. The local JT is not consistent because not all clusters are consistent with the local belief. Nevertheless, the extended linkage host tree is consistent, so that all linkage hosts are equipped with the complete local knowledge to form the correct outgoing message. Overall, no full round of local belief, with both inward and outward message passing, is required for the message calculation.

3.2.3 Global Belief Updates

We now describe the message passing protocol in our new architecture using a non-rooted message scheduling scheme in the LJF global context. Agents no longer follow a recursive call during the global inference, but each simultaneously starts to process incoming and outgoing inter-agent messages. The coordination of the inter-agent message passing is controlled by two simple rules, which elaborate the scheduling scheme of standard Shenoy-Shafer messages.

> **Rule 1.** When an agent has received all except one messages from its adjacent agents, the agent composes an outgoing message to that particular neighbor.
>
> **Rule 2.** When an agent has received the last message from its adjacent agents, the agent absorbs the message, updates the local belief and calculates all outgoing messages to its other neighbors.

Following the above rules, the message passing order is implicitly defined. We present a global propagation algorithm $Communicate\_Belief\_I$ as a set of operation defined for each individual agent.

Algorithm 9 $Communicate\_Belief\_I$
Let $T_i(i = 1, ..., n)$ be the local JTs and $\mathcal{H}$ the corresponding hypertree of an LJF L, which is populated by n agents with one at each subnet. Each agent A_i has K_i adjacent subnets.

When called, each agent A_i performs the following:

```
1. Set cnt= $K_i$;
2. while( $cnt \neq 0$)
3. { Wait for incoming messages;
4.    If an incoming message is received
5.       Set $cnt = cnt - 1$;
6.    If ( $cnt == 1$ && $\exists$ $A_j$ such that no incoming message has arrived from agent $A_j$ )
7.       Send a message to $A_j$ with $Cal\_Single\_MSG$; }
8. Call $Update\_Belief$ in $T_i$;
9. Send all remaining outgoing messages with $Deliver\_Through\_Linkage$;
```

The belief propagation with Algorithm $Communicate\_Belief\_I$ is controlled by an implicit order of inter-agent message calculation. An agent absorbs the buffered incoming message only when it has received all messages except one neighbor, followed by the calculation of the outgoing message based on **Rule 1** with Algorithm Cal_Single_MSG. Once the agent receives all the messages, the local belief is updated and all outgoing messages are calculated based on **Rule 2**. Note that as an incoming message is not guaranteed to be processed right way (if the agent has not received all except one), repeated local propagation is avoided.

In fact, we could even forgo completely the full local update, as described in Line 8 of Algorithm $Communicate\_Belief\_I$, during the global communication. Recall that the partial local propagation is conducted to send a message to a neighbor from whom the last incoming message is originated. When this last incoming message has arrived, we only need to conduct another partial local propagation, just enough to update the local belief with regards to this message. Algorithm Cal_Single_MSG consists of a full inward (in the context of local JT) and a partial outward (in the context of the extended linkage host tree) local message passing. Similarly, we can analogously design an algorithm to conduct a partial local propagation in order to achieve local consistence upon receiving the very last message.

Algorithm 10 $Update\_Belief\_onLastMSG$

Let A_i and A_j be two adjacent agents. A_i's local JT is T_i with the set of clusters C. T_i maintains a linkage tree L to A_j. In T_i,, H is the set of linkage hosts of L and the extended linkage host tree is T_e. Suppose A_i has calculated an outgoing message to A_j with Algorithm Cal_Single_MSG with selected root C_r. A_i, on receiving the last incoming message from A_j, performs the following.

1. A_i absorbs message from A_j by updating belief of all clusters in H with the extended linkage potential of each linkage in L.

2. Perform a partial inward message passing on C_r in T_e, such that C_r calls recursively all clusters in H to send an inward message.

4. Perform a full outward message passing on C_r in local JT T_i, such that C_r sends outward messages to all linkage hosts recursively.

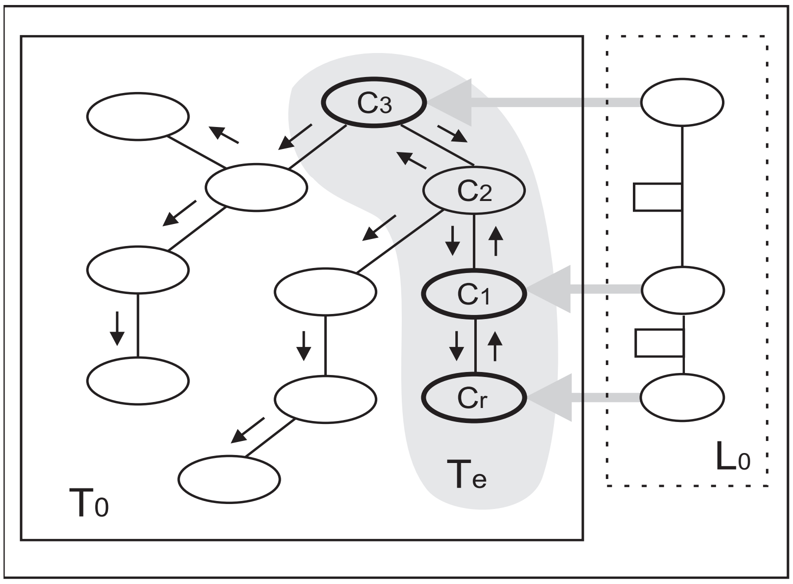

Figure 3.5: An example of partial propagation for updating local calculating a single outgoing message. Shown with the local JT T_0, linkage tree L_0 and extended linkage host tree T_e.

The process of the partial belief update is illustrated in Figure 3.5. Suppose the last incoming message arrives over linkage tree L_0. The linkage hosts C_r, C_1 and C_3 have absorbed the message and the updated potentials of these clusters need to be propagated in the local JT T_0. As the modified potentials are only for clusters in the extended linkage host tree T_e, we first issue a partial inward pass in T_e for C_r to collect all inward messages. Next, outward messages are propagated from C_r to all the cluster in T_0, which brings all clusters to locally consistent with regard to the received incoming message.

We now incorporate Algorithm $Update\_Belief\_onLastMSG$ to replace the call of a full local belief update in Algorithm $Communicate\_Belief\_I$ and obtain our final global propagation algorithm as follows.

Algorithm 11 $Communicate\_Belief$

Let $T_i (i = 1, ..., n)$ be the local JTs and $\mathcal{H}$ the corresponding hypertree of an LJF L, which is populated by n agents with one at each subnet. Each agent A_i has K_i adjacent subnets.

When called, each agent A_i performs the following: 1. Set cnt= K_i;

2. while($cnt \neq 0$)

```
3. { Wait for incoming messages;
4.     If an incoming message is received
5.         Set $cnt = cnt - 1$;
6.     If ( $cnt == 1$ && $\exists A_j$ such that no incoming message has arrived from agent $A_j$ )
7.         Send a message to $A_j$ with $Cal\_Single\_MSG$; }
8. Call $Update\_Belief\_onLastMSG$ in $T_i$;
9. Send all remaining outgoing messages with $Deliver\_Through\_Linkage$;
```

Theorem 1 *Let F be over domain $\mathcal{N}$ the LJF of an MSBN and $Communicate\_Belief$ is performed in F. Let A be the agent of a local JT T with clusters T_c over local variables N. A has n neighboring agents $A_1, A_2, ..., A_n$. Let*

$$\Phi(N) = \prod_{C \in T_c} \Phi(C) \prod_{i=1}^{n} \Phi(M_{A_i \to T}),$$

where $\Phi(C)$ is the potential initially assigned to a cluster C, and an incoming message received from T's one neighboring agent is denoted by $\Phi(M_{A_i \to T})$. Then,

$$\Phi(N) = const \sum_{\mathcal{N} \setminus N} \Phi_F(\mathcal{N})$$

,

where $\Phi_F(\mathcal{N})$ represents the global belief of F over $\mathcal{N}$.

Based on Theorem 1, after a call to $Communicate\_Belief$ in an LJF is finished with all inter-agent messages delivered, each agent's local belief can be obtained as the product of its initial local belief with all its incoming messages, and the local belief is also globally consistent. The most intuitive way to verify its validity is to view our global message passing in LJF as the Shenoy-Shafer message passings in JTs. Although partial propagations are utilized during local updates, we have shown that the message computed with Algorithm Cal_Single_MSG is consistent to local belief, and Algorithm $Update\_Belief\_onLastMSG$ updates the local belief coherently. Therefore, a proof to Theorem 1 can be simply lifted from the Shenoy-Shafer JT global propagation to our new LJF message passing architecture.

Complexity Analysis

Our global inference algorithm can be divided into three parts: the initialization of an LJF; the calculation of all external and internal messages; and the computation of the belief for each LJF local JT. The cost of constructing an LJF through the cooperative compilation is typically overshadowed by the cost to compute inter-agent messages and local beliefs. Extending the complexity analysis of message propagation JT, we can consider the asymptotic complexity as the cost of computing the most expensive inter-agent message. This cost is determined by the size of the largest local JT in the LJF. Thus, the time complexity of our new architecture is exponential in the largest local JT of the LJF.

In a more detailed analysis with a comparison to other LJF inference algorithms, we use the follow parameters:

- n: the total number of agents.
- c: the maximum number of clusters in a local JT.
- d: the maximum number of clusters in an extended linkage host tree.
- s: the maximum number of adjacent agents.
- q: the cardinality of the largest cluster.

First, we consider the complexity of the local calculation with our global propagation algorithm. During a call to Algorithm Cal_Single_MSG, as only partial propagation is performed, the time complexity with message passing among local clusters is between $O(c + d)$ and $O(c + d)2^q)$, which is also the cost during a call to Algorithm $Update\_Belief\_onlastMSG$. In a local JT, both Algorithms $Update\_Belief\_onlastMSG$ and Cal_Single_MSG are called only once. Therefore, the total cost for local calculation is between $O(2(c+d))$ and $O(2(c+d)2^q)$.

With Hugin-based LJF inference, Algorithm $Update\_Belief\_Orig$ is called to unify local belief. In local JT, each cluster sends a message to, and receives a message from

	Local JT	Global LJF
Hugin-based	$O(4cs2^q)$	$O(4ncs2^q)$
Lazy-based	$O(4cs)$ - $O(4cs2^q)$	$O(4ncs)$ - $O(4ncs2^q)$
Our architecture	$O(2(c+d))$ - $O(2(c+d)2^q)$	$O(2n(c+d))$ - $O(2n(c+d)2^q)$

Table 3.1: Comparison of time complexity: local and global cost.

each of its adjacent clusters. The complexity is linear in c. In the simplest case, the message is empty and its computation trivial. In the most complex case, the potential over the cluster needs to be obtained by multiplication and marginalization. Therefore, the time complexity of $Update\_Belief$ is $O(2c2^q)$ [98]. Due to the fact that an agent must call $Update\_Belief\_Orig$ whenever a message is delivered from its adjacent agent during the inward and outward pass. Thus, the local time complexity is $O(4cs2^q)$. The time complexity for the Lazy-based global inference in LJFs and DLJFs, as well as the Shenoy-Shafer extension in DLJFs, can be obtained similarly. That is between $O(4cs)$ and $O(4cs2^q)$. It is clear that the local computation in our new architecture is more efficient than the other methods, and particularly it does not depends on the topology of the network, e.g. the number of neighboring nodes.

Next, consider the time complexity of $Communicate\_Belief$. Each of the n agent will perform calls to Cal_Single_MSG and $Update\_Belief\_onlastMSG$. Thus, we have the time complexity of our global inference as between $O(2n(c + d))$ and $O(2n(c + d)2^q)$. This result can be compared with the Hugin-based global inference in LJFs, which has the time complexity of $O(4ncs2^q)$, and with the Lazy-based global inference and the Shenoy-Shafer extension in DLJFs, which has the complexity between $O(4ncs)$ and $O(4ncs2^q)$. It is obvious that the Hugin-based inference has a much higher complexity than the lower-bound result for our new architecture. As a linkage tree usually spans over a small number of clusters in a local JT, in larger networks, typically we have $c >> d$. Therefore, based on our analysis, the global inference in our new architecture is also more time efficient than all existing LJF or DLJF exact inference algorithms. The time complexity comparison of the three inference algorithms is listed in Table 3.1.

For space complexity, each agent maintains a linkage tree, following the standard LJF construction. The message buffers do not require extra memory, as the linkage trees are utilized for this purpose. We need additional storage to keep copies of the original local beliefs in order to form the correct posterior distribution during each global propagation. However, it is still less than the space required with DLJF-based methods. In a DLJF with n agents, there must be $2(n-1)$ sets of message JTs each over a local subdomain, whereas in an LJF, we only need to maintain $2(n-1)$ linkage trees, each over the d-sepset that is generally much smaller than the size of a local domain.

3.3 Towards Fault-Tolerant Exact Belief Propagation

Existing MSBN inference methods, extending either from the Hugin or the Shenoy-Shafer architecture, have difficulties in dealing with unreliable communication channels that often exist in a multi-agent environment. One prominent problem is that the loss of inter-agent messages can halt global message passings. For example, in the recursive Hugin-based architecture, if any message originated from an LJF leaf node is missing during the inward pass, the global belief updating fails immediately. A similar problem also exists in our new architecture. Although the messages are not initiated at a root node, there is exact one inter-agent message passed over each pair of adjacent agents at one direction. Therefore, a lost message will also prevent Algorithm $Communicate\_Belief$ from finishing successfully.

In fact, with the LJF message passing algorithms we have so far described, the message from an agent A_i to A_j could only be computed after all messages to A_i (except that from A_j) have been computed. Such message passing algorithms are *synchronized* since there is a partial order constrained by the message passing protocol with which messages are calculated. It is possible, however, to define an LJF *asynchronous* algorithm that forgoes the restricted message passing rules. In such an algorithm, the messages are initialized with arbitrary values and then updated iteratively. Essentially, there could be more than one message sent over a hyperlink in one direction.

In BNs, iterative algorithms can be applied to multiply connected networks to perform approximate inference, known as loopy belief propagation, or iterative belief propagation [65] [49] [23] [36]. The JT message propagation has iteratively applied to join-graph to perform approximate inference. Loopy belief propagation is extended to generalized belief propagation [106] by clustering some of the nodes in BNs into super nodes and apply message passing among the super nodes. Iterative algorithms have been developed for decoding problems which are viewed as an instance of the belief propagation [60] [79]. It is important to note that in a cycle-free graph, the iterative message passing among all vertices will eventually converge, but such convergence is not always possible in a multiply connected graph [50] [74] [86]. Applying this result to an MSBN LJF, the convergence of iterative passing of inter-agent message is also guaranteed to converge.

Recently, LJF fault-tolerant exact propagation [2] has been discussed. It is suggested that the iterative message passing is not suitable in the Hugin architecture, as an agent's belief may not converge correctly. It also has been shown that with the use of buffering, it is possible to conduct asynchronous message passing based on lazy propagation in both LJFs and DLJFs [2]. Although these methods are exact and less sensitive to message lost, they are computational expensive. Each agent needs to update the local belief and re-send all outgoing messages whenever triggered by one new incoming message, which is analogous to the *flooding schedule* [5] [50]. Moreover, the delivery of a single inter-agent message is also costly. In order to send a message to a neighboring agent, an agent's local belief must be combined with messages coming from all its other neighbors. This process must be repeated again for each of message during each iteration at a local JT. Such a schema causes agents to be interrupted frequently for message computation, which indeed limits the agent's autonomy.

Therefore, we consider an improved asynchronized message passing schema based on our extended Shenoy-Shafer architecture. The goal is to perform exact computation with better fault tolerance as to message loss, and to reduce local computational cost from existing iterative passing algorithms. Instead of allowing the message calculation to be triggered at each incoming message, agents now have a higher degree of control as how to process the incoming and outgoing messages.

3.3.1 Calculation of Iterative Messages

We first present our method for the calculation of iterative messages, which forms the foundation of our fault-tolerant message passing. We introduce a control factor for message processing. We consider the original synchronized schema as one extreme, with which an outgoing message is formed only after all except one incoming messages have arrived. Meanwhile, we consider the current LJF iterative schema [2] as the other extreme, such that one incoming message will trigger the recalculation of all outgoing messages. Our new method balances the both and provides an agent the flexibility to control its own schedule for message recalculation.

An agent can also decide on the number of incoming messages that must be received to start processing outgoing messages. In fact, such message calculation can be easily supported with our extended Shenoy-Shafer architecture, due to the use of linkage trees as buffers. An agent's outgoing messages (to all its adjacent agents) can be formed periodically and as a batch. We call an incoming message *valid* if the message carries additional information compared with the previous buffered message (from the same sender). The income messages are batch processed only when a certain number of valid messages have arrived. For each batch messages, we combine the current buffered incoming messages with the agent's local belief once to obtain the updated local belief, and then produce each outgoing message by removing the redundant factors.

Algorithm 12 Cal_Iter_Msg

Let A be an agent with local JT J and m adjacent agents. For each of A's neighboring subnet $A_i (i = 1, ..., m)$, the linkage trees L_i and L'_i are maintained respectively by A and A_i. Let l_i, l'_i (i=1,...,n) each be a corresponding pair of linkages in the linkage trees, and C_i and C'_i be the linkage hosts. $\Phi_{C_i}(l_i)^{cur}$ is the current potential in linkage l_i. When called, A does the following:

```
1.  Identify all valid incoming messages;
2.  Absorb the linkage potentials $\Phi_{C_i}(l_i)^{cur}$ of the messages to linkage hosts.
3.  Call $Update\_Belief$;
4.  For each $A_i(i = 1, ..., m)$
5.      Calculate the extended linkage $\Phi^*_{C_i}(l_i)$ from $C_i$
6.      Compose the message potential as $\Phi(Msg) = \Phi^*_{C_i}(l_i)/\Phi_{C_i}(l_i)^{cur}$
7.      Assign $\Phi^*_{C_i}(l_i)' = \Phi(Msg)$.
8.  End for.
```

With Algorithm Cal_Iter_Msg, each inter-agent message can be viewed as a Shenoy-Shafer message, calculated with a modified scheme to avoid the repeated multiplications. The batched message calculation requires only one call to $Update\_Belief$, and the division operation is required for each message instead of repeated multiplication. Clearly, compared to the repeated message passing of current methods, the local computational cost is much less. Even though the division operation introduces certain computational overhead, the saving is still significant considering the total number of local propagation and message calculation avoided.

3.3.2 Global Iterative Message Passing

It is well known that message passing in a factor graph is guaranteed to converge if the graph contains no cycles [50]. In a message-oriented architecture, if the messages are calculated iteratively over edges of a tree graph, each message will eventually become unchanged (converged) after enough round of message passings. The local belief of each node converges and the result is exact. This result is consistent with the original belief propagation with the Pearl's algorithm in BNs [69]. An LJF, as a high level JT, can be viewed as a factor graph where each local JT represents a node with a local function. Thus, the message passing over each hyperlink will converge, provided the messages are explicitly calculated and maintained. In the existing Hugin-based architecture, such message convergence is not possible as the Hugin messages are not individually maintained [92]. A detailed discussion of the non-convergence property of the Hugin architecture can be found in some recent literature [2].

With our extended Shenoy-Shafer architecture, the extension to LJF iterative message passing is simple and straightforward. Most importantly, the result of message passing will converge to exact values. As most practical systems are finite, in which the propagation is expected to finish after a certain time, we design our iterative message passing algorithm with finite message passing which is contrary to other existing LJF iterative algorithms [2]. The basic principle in our new message passing schema is that each agent batch-processes the buffered message in a predetermined interval until all messages converge. We keep track of all the converged messages and terminate the propagation after all messages have become stable. That is, the propagation is finished when no new incoming message is valid, e.g. it does not carry any additional information to effect an agent's local belief. The global propagation with iterative message passing is presented in Algorithm $Iter\_Msg\_Prop$.

Algorithm 13 $Iter\_Msg\_Prop$

An MSBN LJF is populated by multiple agents with one at each local JT. An arbitrary agent A_i has a JT T_i constructed locally. A_i has n adjacent local subnets. V is denoted as the number of valid messages, and $UPDATE\_INTERVAL$ is the update time interval. When called, agent A_i performs the following:

```
1.  Initialize all incoming message buffers ;
2.  Set V = n;
3.  while(true)
4.      Absorb local evidence if any;
5.      Sleep(UPDATE_INTERVAL);
6.      Update V with the current number of valid incoming messages;
7.      If V == 0
8.          Update local belief;
9.          Retrun;
10.     Else
11.         Call $Cal\_Iter\_Msg$ to calculate outgoing messages;
12.         For each of n adjacent agents
13.         Deliver corresponding inter-agent messages;
14.     End if
15. End while
```

When Algorithm $Iter\_Msg\_Prop$ is called, each agent processes incoming and outgoing messages in complete parallel with each other, till no more new message arrives from adjacent agents. This scheme is more desirable in a multi-agent setting as all agents carry out local computation and message calculation in parallel. Compared with the existing schema with which an agent must be interrupted by each incoming message, now the agent gains a higher level of autonomy as to batch-process all incoming messages with a self-determined interval.

More importantly, the iterative message passing with Algorithm $Iter\_Msg\_Prop$ will terminate and messages are guaranteed to converge. This is due to the non-cyclic agent organization structure of an LJF, as well as the message buffering in our extended Shenoy-Shafer architecture. Even though an agent's belief may be skewed temporally by a missing or corrupted message, the correct result will eventually be recovered once the communication is restored. The convergence of an inter-agent message can be easily verified by compared the new message with the current value in the buffer, and flag the buffer accordingly. Once all messages have converged, an agent's updated local belief is obtained exactly by multiplying the messages into the original local belief.

3.4 Discussion

Exact posterior calculation is one of the most important tasks of multi-agent probabilistic inference. The existing algorithms, based on an MSBN secondary structure call LJF, typically extend the Hugin-based message scheduling which requires an extensive amount of local computation in each LJF local JT. In this chapter, we have presented an improved LJF-based global inference architecture based on the Shenoy-Shafer message passing schema. As the passing of a JT Shenoy-Shafer message requires the buffering of each message over a separator toward each direction, we use the LJF linkage trees for this purposed. Our new architecture is message-oriented as all inter-agent messages are explicitly calculated and buffered. We have discovered that, although the total number of external messages remains the same in our extended Shenoy-Shafer architecture, the calculation of such messages have become much less expensive.

This reduction in local calculation cost is first realized through the elimination of repeated local updates. Whereas an incoming Hugin message is immediately absorbed and propagated in the local JT, an incoming Shenoy-Shafer message is buffered and absorbed when all messages (or except one) have arrived. Secondly, we have designed algorithms to conduct only partial local propagation during the LJF global inference. The calculation of the first outgoing message does not invoke a full round of local propagation, either does the calculation of all other outgoing messages. A local JT becomes locally consistent as well as globally consistent without the need to conduct any full round of local message passings in our new architecture.

Our complexity analysis has shown that the cost for internal message passing in a local JT is significantly lower than other existing methods. Moreover, the global inference in our new architecture is also more time efficient than all existing LJF or DLJF exact inference algorithms. Beside a small cost incurred from keeping a copy of the original local belief, our improved architecture does not require additional memory for the message buffers. Moreover, we have maintained the existing LJF which is a more compact and easier-to-built structure than a DLJF.

Another advantage of our new architecture is its support for asynchronized message passings. Although non-rooted, the current message passing scheme is still considered synchronized because there is an implicit order of how the inter-agent messages are calculated. However, our architecture can be easily extended to exact belief updating with iterative message passing. We take advantage of the buffered message calculation and the LJF tree-like structure so that the iterative updates of inter-agent message are guaranteed to converge. During iterative global propagation, each agent is able to reason about its locally observed evidence continuously, and update the local belief with regards to all incoming inter-agent messages at self-determined intervals. It is more robust compared to the recursive methods in that temporary communication errors can be tolerated without causing global belief updating failure. Also, comparing to other recent LJF iterative algorithms, our iterative algorithm is finite and each agent is provided with a higher level of autonomy.

In our next work, an implementation of the proposed architecture is called for, along with an empirical comparisons with the existing recursive and iterative algorithms. In particular, we will evaluation our method as a concurrent program, for which the issues like the deadlocks need to be addressed.

Chapter 4

BN Prior Marginal Factors

In this chapter, we examine the problem of JPD factorization for single-agent oriented BNs, with a goal to develop more efficient JT inference algorithms. The exact inference calculation of a BN is typically carried out in a secondary structure, a JT. We will focus on the message passing scheme of the Hugin architecture, which is described by the Hugin global propagation (GP) [40] [59]. The GP method has been well received and widely applied. However, the semantic meaning of the messages has never been under close investigation.

We will take a new algebraical approach to analyze the GP messages. By studying the factorizations of a JPD both before and after the GP method, we are able to reveal the actual meaning of the GP messages. This result will help us in designing a procedure named Allocate Separator Marginal (ASM). The procedure determines the actual information a cluster requires to form the cluster marginal. The concept of *marginal factors*, defined in this chapter for BN JT clusters, allows us to design an algorithm to calculate the prior marginal for each JT cluster in an informed and more efficient way. This result also prepares the theoretical basis of the fast LJF marginal calibration technique that will be introduced in Chapter 5.

Moreover, the current initialization of a BN JT assigns CPDs randomly to one qualified cluster. An initial CPD assignment could affect the composition of the marginal factors in each of the clusters, resulting in variations of the message composition and flows. Therefore, we present a simple heuristic for the JT initialization phase, such that we assign the initial CPDs in a more controlled way, resulting in further reduced number of required messages.

Incorporated with the initialization heuristic and the ASM procedure, we are able to realize the minimum messages passing to calculate the prior marginal distribution. Our experiments have shown that our algorithms can significantly improve the standard Hugin global propagation.

4.1 Semantic Meaning of GP Messages

The viability of BNs very much depends on the development of its inference algorithms. Among various methods developed, the GP algorithm in the Hugin architecture [40] [59] has been very well received and implemented. With the GP method, BN inference is realized on a JT through a coordinated series of local manipulations called *message passings*. The following highlights pertinent facts of the GP method relevant to the discussions in this chapter, with an example.

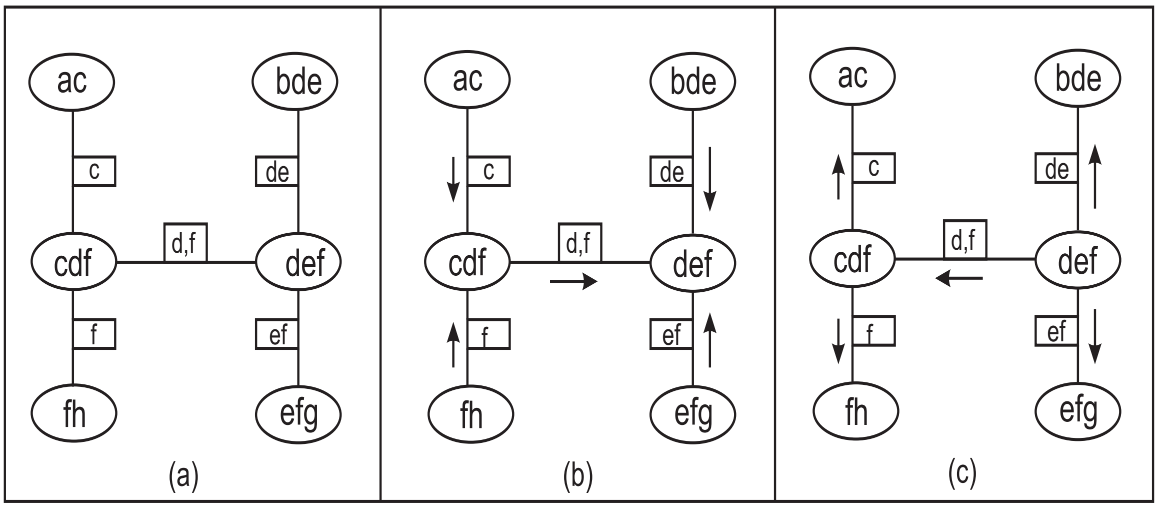

Figure 4.1: (a) The JT constructed from the BN in Figure 2.1. (b) Inward message passing, and (c) Outward message passing with cluster def as the root.

Consider again the Asia travel BN. The DAG in Figure 2.1 is moralized and triangulated so that a JT such as the one in Figure 4.1 (a) is constructed. This JT consists of six clusters (shown with rounded boxes) that are denoted as $C_1 = ac$, $C_2 = bde$, $C_3 = cdf$, $C_4 =$

def, $C_5 = fh$, $C_6 = efg$. Five separators (shown with smaller boxes) are attached to the edge connecting two neighboring clusters, and are denoted as $S_1 = c$, $S_2 = de$, $S_3 = df$, $S_4 = f$, $S_5 = ef$.

Every CPD $P(a_i|Pa(a_i))$ in Figure 2.1 is assigned to a cluster C_j if $\{a_i\} \cup Pa(a_i) \subseteq C_j$ to form the cluster potential $\Phi(C_j)$ before the GP method is applied. If $\{a_i\} \cup Pa(a_i)$ is a subset of two or more clusters, then we arbitrarily assign $P(a_i|Pa(a_i))$ to one of the clusters. If no CPD is assigned to a cluster C_j, then $\Phi(C_j) = 1$. In our example, the following cluster potentials will be obtained before the GP is applied:

$$\begin{array}{rcl} \Phi_{C_1}(ac) = & P(a) \cdot P(c|a), & \Phi_{C_4}(def) = 1 \\ \Phi_{C_2}(bde) = & P(b) \cdot P(d|e) \cdot P(e|b), & \Phi_{C_5}(fh) = P(h|f) \\ \Phi_{C_3}(cdf) = & P(f|cd), & \Phi_{C_6}(efg) = P(g|fe) \end{array} \tag{4.1}$$

Meanwhile, a separator potential is also formed for each separator with an initial value 1. That is, $\Phi(S_i) = 1$, $(i = 1, \ \ldots, \ 5)$. It is easy to verify that the following equations hold before the GP method is performed on the JT:

$$\begin{aligned} P(V) &= \Phi(C_1) \cdot \Phi(C_2) \cdot \Phi(C_3) \cdot \Phi(C_4) \cdot \Phi(C_5) \cdot \Phi(C_6). \\ &= \frac{\Phi(C_1) \cdot \Phi(C_2) \cdot \Phi(C_3) \cdot \Phi(C_4) \cdot \Phi(C_5) \cdot \Phi(C_6)}{\Phi(S_1) \cdot \Phi(S_2) \cdot \Phi(S_3) \cdot \Phi(S_4) \cdot \Phi(S_5)}. \end{aligned} \tag{4.2}$$

Note here the separator potentials $\Phi(S_i)(\cdot)$ are unity potential ($\Phi(S_i) = 1$).

Message passing is the basic operation in the GP method. Consider two adjacent clusters C_i and C_j with the separator S_{ij}, that C_i passes a message to C_j (or C_j absorbs the message from C_i) means a two-step computation: (1) updating the separator cluster $\Phi(S_{ij})$ by setting $\Phi(S_{ij}) = (\sum_{C_i \setminus S_{ij}} \Phi(C_i))/\Phi(S_{ij})$; (2) updating the cluster potential $\Phi(C_j)$ by setting $\Phi(C_j) = \Phi(C_j) \cdot \Phi(S_{ij})$, where the potential $\Phi(S_{ij})$ is an actual Hugin "message" passed from C_i to C_j.

The GP method typically consists of a coordinated sequence of message passes. Consider a JT with n clusters. First, a cluster in the JT is randomly selected as the root. Then a sequence of message passes is performed through the two stages, the *inward* stage and the *outward* stage. The *inward* stage causes $n-1$ messages to be passed. Similarly, the *outward* stage causes another $n-1$ messages to be passed. Altogether, there are exactly $2(n-1)$ messages to be passed [35, 39]. The sequence of message passes is shown in Figure 4.1 (b) and (c) when the cluster $C_4 = def$ is the root.

With the GP message passing, there are exactly two messages passed over each separator: one in the inward stage and one in the outward stage. After passing a total of $2(n-1)$ messages, the potentials $\Phi(C_i)$ and $\Phi(S_j)$ will be transformed into marginals $P(C_i)$ and $P(S_j)$ respectively. Thus, the following equation holds in our example [35].

$$P(V) = \frac{P(C_1) \cdot P(C_2) \cdot P(C_3) \cdot P(C_4) \cdot P(C_5) \cdot P(C_6)}{P(S_1) \cdot P(S_2) \cdot P(S_3) \cdot P(S_4) \cdot P(S_5)}. \tag{4.3}$$

Although the mechanism of the GP method is well understood, the semantic content of the messages $\Phi(S_i)$ is not clearly defined, and the messages are passed as meaningless potentials. By comparing Equation 4.2 with Equation 4.3, however, we can view the message passing of the GP method as a process to transform Equation 4.2 to Equation 4.3. This unique view of transformation provides us a chance to analyze the messages *algebraically*, which indeed leads to the demystification of the semantic meanings of GP messages.

Recall that the cluster potentials in Equation 4.2 are in fact composed of the original CPDs from the BN shown in Figure 2.1. If we substitute the actual contents for the cluster potentials in Equation 4.2, we obtain the following:

$$\begin{aligned} P(V) = \ & \overbrace{[P(a) \cdot P(c|a)]}^{\Phi(C_1)} \cdot \overbrace{[P(b) \cdot P(d|e) \cdot P(e|b)]}^{\Phi(C_2)} \cdot \\ & \overbrace{[P(f|cd)]}^{\Phi(C_3)} \cdot \overbrace{[1]}^{\Phi(C_4)} \cdot \overbrace{[P(h|f)]}^{\Phi(C_5)} \cdot \overbrace{[P(g|fe)]}^{\Phi(C_6)} \end{aligned} \tag{4.4}$$

Essentially, Equation 4.4 shows an initial potential assignment in each JT cluster. Comparing Equation 4.4 with Equation 4.3, one may immediately notice that Equation 4.4 does not have any denominators while Equation 4.3 does. Thus, we multiply and divide the term

$\Pi_{j=1}^{5} P(S_j)$ at the same time to Equation 4.4, and we obtain

$$P(V) = \frac{P(c) \cdot P(de) \cdot P(df) \cdot P(f) \cdot P(ef) \cdot \overbrace{P(a)P(c|a)}^{\Phi(C_1)} \cdot \overbrace{P(b)P(d|e)P(e|b)}^{\Phi(C_2)} \cdot \overbrace{P(f|cd)}^{\Phi(C_3)} \cdot \overbrace{1}^{\Phi(C_4)} \cdot \overbrace{P(h|f)}^{\Phi(C_5)} \cdot \overbrace{P(g|fe)}^{\Phi(C_6)}}{P(c) \cdot P(de) \cdot P(df) \cdot P(f) \cdot P(ef)} \tag{4.5}$$

Now, the Equation 4.5 and Equation 4.3 both have exactly the same denominators but differ in the numerators. In particular, we now have some extra dangling marginals in the numerator of Equation 4.5, which are added as the term $\Pi_{j=1}^{5} P(S_j)$ is multiplied to Equation 4.4. In order to reach Equation 4.3 from Equation 4.5, we need to transform the initial cluster potential in Equation 4.5 into a cluster marginal on its respective cluster. The extra marginals thus play an important role in the transformation. Essentially, such transformation can be viewed as the multiplication of separator marginals to appropriate cluster potentials in Equation 4.1.

Given the initial CPD assignment, our goal is to form the cluster marginal. First of all, it is possible to obtain directly the cluster marginal from the initial assignment in some clusters. In the Asia example, we have $\Phi_{C_1}(ac) = P(a) \cdot P(c|a) = P(ac)$ and $\Phi_{C_2}(bde) = P(b) \cdot P(d|b) \cdot P(e|b) = P(bde)$. In other words, no separator marginal is needed to be mingled with the initial cluster potentials.

Secondly, some cluster potentials can also be easily transformed into the cluster marginal by multiplying whole terms of separator potential. For the cluster potential $\Phi_{C_5}(fh) = P(f|h)$, we can multiply it with the separator marginal $P(f)$ which results in $\Phi_{C_5}(fh) = P(h|f) \cdot P(f) = P(fh)$. For the cluster potential $\Phi_{C_6}(efg) = P(g|ef)$, we can multiply it with the separator marginal $P(ef)$ which results in $\Phi_{C_6}(efg) = P(g|ef) \cdot P(ef) = P(efg)$. So far, we have successfully obtained marginals for clusters C_1, C_2, C_5, and C_6. The separator marginals $P(f)$ and $P(ef)$ have been used during this process.

Finally, we need to investigate the factorization of separator potential in order to achieve certain cluster potentials, which is different than the above two situations. For example, we need to transform the cluster potentials $\Phi_{C_3}(cdf) = P(f|cd)$ and $\Phi_{C_4}(def) = 1$ into marginals, with the remaining separator marginals, i.e., $P(c)$, $P(de)$ and $P(df)$ at hand. In order to make $\Phi_{C_3}(cdf) = P(f|cd)$ marginal, we need to multiply it with $P(cd)$, which is not available as separator marginals. However, if we factorize the separator potential

potential	initial assignment	receives	result
$\Phi(C_1)$	$P(a)$, $P(c\|a)$	nothing	$P(a) \cdot P(c\|a) = P(ac)$
$\Phi(C_2)$	$P(b)$, $P(d\|b)$, $P(e\|b)$	nothing	$P(b) \cdot P(d\|b) \cdot P(e\|b) = P(bde)$
$\Phi(C_3)$	$P(f\|cd)$	$\underline{P(c)}$, $\underline{P(d)}$	$P(f\|cd) \cdot \underline{P(c)} \cdot \underline{P(d)} = P(cdf)$
$\Phi(C_4)$	1	$\underline{P(de)}$, $\underline{P(f\|d)}$	$\underline{P(de)} \cdot \underline{P(f\|d)} = P(def)$
$\Phi(C_5)$	$P(h\|f)$	$\underline{P(f)}$	$P(h\|f) \cdot \underline{P(f)} = P(fh)$
$\Phi(C_6)$	$P(g\|ef)$	$\underline{P(ef)}$	$P(g\|ef) \cdot \underline{P(ef)} = P(efg)$

Table 4.1: Allocating separator marginals, the underlined terms are either the separator marginal or from the factorization of a separator marginal.

$P(df)$ as the multiplication of two terms $P(d)$ and $P(f|d)$, we are able to assemble all necessary terms, $P(d)$ from the factorization and $P(c)$ from the separator marginal. Given CI $I(d,\ \emptyset,\ c)$ holds in the original DAG in Figure 2.1, $\Phi_{C_3}(cdf) = P(cdf)$, we thus obtain the marginal potential $\Phi_{C_3}(cdf) = P(f|cd) \cdot P(d) \cdot P(c)$. The remaining term $P(f|d)$ from the separator factorization and the separator $P(de)$ can now be multiplied to $\Phi_{C_4}(def) = P(de) \cdot P(f|d)$. As the CI $I(f,\ d,\ e)$ holds in the original DAG in Figure 2.1, we thus obtain marginal $P(cdf)$. So far we have successfully and algebraically used all separate marginals to transform each cluster potential $\Phi(C_i)$ into a marginal $P(C_i)$. Table 4.1 summarizes the allocation scheme for the multiplied separator marginals in our example.

4.2 JT Marginal Factors

4.2.1 Allocate Separator Marginals

At the JT initialization stage, every JT cluster C_i is associated with an initial cluster potential $\Phi(C_i$. During the course of propagation, the GP method transforms the cluster potential $\Phi(C_i$ into a cluster marginal $P(C_i)$. This algorithmic phenomena of the GP method can be explained algebraically. Consider Equation 4.5, in which the numerators are the original cluster potentials together with the multiplied separator marginals. The messages received by all the clusters in the GP method as a whole, which *algorithmically* transform each cluster potential into a cluster marginal, have the same effect as the separator marginals we multiplied in Equation 4.5, which *algebraically* transform each cluster potential into a cluster marginal. This analysis leads to the following proposition.

Proposition 3 *The product of the messages received by every cluster in the GP method equals to the product of all separator marginals.*

The example in the previous section has demonstrated that each separator marginal or its factorization was used exactly once and allocated to an appropriate cluster potential. This perfect arrangement of separator marginals is not a simple coincidence. Assigning either a separator marginal or its factorization to a cluster potential, as shown in Sect 4.1, must satisfy one necessary condition, namely, condition (1) of Definition 2, in order for the product of the cluster potential with the allocated separator marginal or its factorization to be a marginal. That is, for each $\Phi(C_i)$, we need a CPD with a_j as head for each $a_j \in C_i$. If a variable, say a_j, appears m times in m clusters in the JT, then each of these m clusters will need a CPD with a_j as head. However, the original BN provides only one CPD with a_j as head, and we are short of $m - 1$ CPDs (with a_j as head). Fortunately, m clusters containing a_j implies the JT must have exactly $m - 1$ separators containing the variable a_j [35], therefore the $m - 1$ needed CPDs with a_j as the head will be supplied by the $m - 1$ separator marginals (or their factorizations). Based on this analysis, we present the following procedure.

Procedure: Allocate Separator Marginals (ASM)

Step 1. Consider each CPD $P(a_i|Pa(a_i))$ assigned to a cluster C_k to form $\Phi(C_k)$. If the variable a_i appears in a separator S_{kj} between C_k and C_j, then draw a small arrow originating from a_i in the separator S_{kj} and pointing to the cluster C_j. If variable a_i also appears in other separators in the JT, draw a small arrow on a_i in those separators and point to the neighboring clusters away from C_k's direction. Repeat this for each CPD $P(a_i|Pa(a_i))$ of the given BN.

Step 2. Examine each separator S_i in the JT. If the variables in S_i all point to one neighboring cluster, then the separator marginal $P(S_i)$ will be allocated to that neighboring cluster.

Step 3. For a separator $P(S_i)$ that has nodes allocated to both connecting clusters, then $P(S_i)$

has to be factorized so that the obtained factors can be assigned to appropriate cluster indicated by the arrows in the separator.

The procedure for allocating separator marginals is illustrated in Figure 4.2. In Step 1, a total of eight CPDs are originally assigned to respective clusters to form the cluster potentials as in Equation 4.1. As the CPDs $P(a)$, $P(b)$, $P(g|f)$ and $P(h|f)$ are not contained in any separators, they are not considered in the allocation procedure. For the remaining CPDs, we first consider the CPD $P(c|a)$ assigned to cluster $C_1 = ac$. Since the variable c is contained only in the separator between cluster C_1 and cluster $C_3 = cdf$, we draw a directed arrow from c in the separator, pointing away from to cluster C_1 to cluster C_3. The other three CPDs, $P(d|b)$, $P(e|b)$ and $P(f|cd)$, are all contained in more than one separator. Thus, we draw arrows over all their occurrences in all the separators, away from the cluster to which each CPD is originally assigned. Following Step 2 in the procedure, we combine all arrows that point in the same direction. For example, for separator de and ef, they are allocated completely to the cluster def and efg respectively. In other words, no factorization is needed for both separators. This complete allocation, however, does not apply to separator df following Step 3. The separator marginal of df must be factorized to accommodate both connecting clusters.

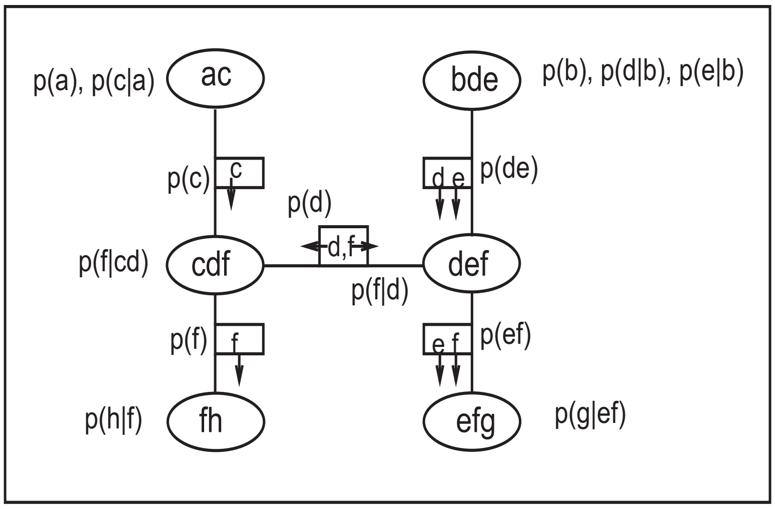

Figure 4.2: Allocating separate marginals according to the ASM procedure.

For each separator in a JT, one can always assign either the separator marginal or some factors in its factorization to an appropriate cluster C_i as dictated by the procedure, such that for each variable $a_j \in C_i$, there is a CPD assigned or allocated to the cluster $\Phi(C_i)$ in which a_j is the head. Essentially, the ASM procedure provides a high level understanding of how the marginal information of separators flows during the GP propagation process. Indeed, each arrow over a separator node c indicates which cluster potential requires a CPD, one with c as head, to complete the cluster marginal calculation. If a CPD $P(a_i|Pa(a_i))$ is already assigned to a cluster C_k and a_i appears in a separator of C_k connecting to other clusters, the cluster potential $\Phi(C_k)$ obviously does not need an extra CPD with a_i as head, and the arrow over a_i in the separators indicates this CPD information must be incorporated to the other clusters.

The ASM procedure, however, does not provide the actual messages flowing over a separator if arrows of both direction present in the separator. In this situation, we need to determine the proper factorization as for the proper information to be sent to each of the two clusters. For example, according to the ASM procedure, the separator *df* must be factorized into two CPDs, each containing d or f as the head. As *df* can be factorized as either $P(d)\dot{P}(f|d)$ or $P(f)\dot{P}(d|f)$, both satisfy the requirement based on the ASM procedure. This leads to the question: which factorization represents the correct message information?

Proposition 4 *If the procedure ASM indicates that a separator marginal $P(S_i)$ has to be factorized before it can be allocated to its neighboring clusters, then $P(S_i)$ must be factorized based on a topological ordering of the variables in S_i with respect to the original DAG.*

This proposition provides a method to determine the correct factorization of the separator marginals. Although an appropriate allocation of the separator marginals can always be guaranteed to satisfy condition (1) of Definition 2, one still needs to show that such an allocation will not produce a directed cycle when verifying condition (2) of Definition 2. It is important to note that a directed cycle can be created in a directed graph if and only if one draws a directed edge from the descendant of a node to the node itself.

Consider a cluster C_i in a JT and its neighboring cluster C_j; their connecting separator is S_{ij} with separator marginal $P(S_{ij})$. Suppose the separator marginal is allocated as a whole to C_i. Following the rule of condition (2) in Definition 2, we can obtain directed edges based on the original CPDs assigned to C_i and the newly allocated separator marginal $P(S_{ij})$. The edges will not result in any directed cycle. This is because 1) the original CPDs assigned to C_i are from the given BN without any cycles, and 2) the variables in S_{ij} will be ancestors with regard to all other cluster variables, which create no cycles either.

It is important to determine the proper factorization based on proposition 2 if the separator marginal $P(S_{ij})$ has to be factorized as a product of CPDs. Consider again the example shown in Figure 4.2. If we decompose the separtor as $P(df) = P(f) \cdot P(d|f)$ and assign the factor $P(d|f)$ to C_3, this would result in $\Phi(C_3(cdf) = P(c) \cdot P(d|f) \cdot P(f|cd)$. It is easy to verify that $\Phi(C_3)$, after incorporating the allocated CPD $P(d|f)$, satisfies the condition (1) but not (2) of Definition 2, which means that $\Phi(C_3(cdf) = P(c) \cdot P(d|f) \cdot P(f|cd) \neq P(cdf)$ and it is not a Bayesian factorization.

In fact, the factorization $P(df) = P(f) \cdot P(d|f)$ does not follow the topological ordering of the variables d and f (d should precede f in the ordering) with respect to the original DAG, in which f is a descendant of d. Drawing a directed edge from f to d, as dictated by the CPD $P(d|f)$, would mean a directed edge from the descendant of d, namely, the variable f to the variable d itself, and this is exactly the cause of creating a directed cycle. However, if we follow the topological ordering of the variables d and f with regard to the original DAG, the resulting factorization of $P(df)$ is $P(df) = P(d) \cdot P(f|d)$. This way, the heads of the CPDs in the factorization are guaranteed to be the non-ancestors of their respective tails in the original DAG.

4.2.2 Marginal Factors for JT Clusters

In Proposition 3, we have established a rough connection between the messages passed in the GP method and the separator marginals. We have pointed out that the product of all the messages is equal to the product of all separator marginals. Proposition 4 has further explored this rough connection and suggested that all the separator marginals or their factorizations can be appropriately allocated to cluster potentials. By doing this, each cluster potential, multiplying with the allocated, results also in the desired cluster marginal. The messages received by each cluster algorithmically in the GP method are equal to the allocated separator marginal or its factors received by each cluster potential algebraically. That is, the message received by a cluster C_i from its neighbor C_j is in fact the separator marginal $P(S_{ij})$ or factors in its factorization or 1, where S_{ij} is the separator between C_i and C_j.

Let C_i and C_j be two clusters in a JT constructed from the DAG of a BN, and S_{ij} be the separator between C_i and C_j. Regardless of which cluster in the JT is chosen as the root, there are two messages that will be passed between C_i and C_j. Without loss of generality, suppose a message denoted $M_{i \to j}$ is passed from C_i to C_j in the inward stage, and another message denoted $M_{i \leftarrow j}$ is passed in the outward stage.

Theorem 2 *Consider the result of applying the ASM procedure to the JT. There are three possible outcomes regarding the separator marginal $P(S_{ij})$.*

(a) If $P(S_{ij})$ as a whole is allocated to C_j, then $M_{i \to j} = P(S_{ij})$ and $M_{i \leftarrow j} = 1$.

(b) If $P(S_{ij})$ as a whole is allocated to C_i, then $M_{i \to j} = 1$ and $M_{i \leftarrow j} = P(S_{ij})$.

(c) If $P(S_{ij})$ must be factorized (following a topological ordering of variables in S_{ij}), then $M_{i \to j}$ = the product of factors allocated to C_j and $M_{i \leftarrow j}$ = the product of factors allocated to C_i, both based on the results from the ASM procedure.

The proof of above theorem is straightforward based on the JT message passing scheme. We use an example to illustrate this theorem. Consider the JT in Figure 4.1 (a). If cluster $C_4 = def$ is chosen as the root for the GP method, then $C_3 = cdf$ will send a message to C_4 during the inward stage and C_4 will send a message to C_3 during the outward stage. Before C_3 can send the message to C_4, clusters $C_1 = ac$ and $C_5 = fh$ have to pass messages to C_3. The message from C_1 to C_3 is $\Phi(c) = (\sum\limits_{a} P(a) \cdot P(c|a))/1 = P(c)$, which coincides with (a) in the above theorem. The message from C_5 to C_3 is $\Phi(f) = (\sum\limits_{h} P(h|f))/1 = 1$, which coincides with (b) in the above theorem . The cluster C_3, after absorbing these two messages, becomes $\Phi(C_3) = P(f|cd) \cdot P(c) \cdot 1 = P(f|cd) \cdot P(c)$. The message sent from C_3

to C_4 is $\Phi(df) = (\sum_c P(f|cd) \cdot P(c))/1 = \sum_c \frac{P(fcd)}{P(cd)} \cdot P(c) = \sum_c \frac{P(fcd)}{P(c) \cdot P(d)} \cdot P(c) = \sum_c \frac{P(fcd)}{P(d)} = P(fd)/P(d) = P(f|d)$, which coincides with (c) in the above theorem.

The following corollary can be immediately established if we consider messages of unity potential as trivial messages.

Corollary 1 *The ASM procedure provides the minimum set of non-trivial messages for marginal calibration.*

Also, one may expect that messages passed in the GP method should be different if a different root if chosen. The following corollary, based on Theorem 2 reveals an interesting property of the messages.

Corollary 2 *Let $M_{i \leftarrow j}$ and $M_{i \rightarrow j}$ be the messages passed between C_i and C_j when C_k is chosen as the root. Let $M^{'}_{i \leftarrow j}$ and $M^{'}_{i \rightarrow j}$ be the messages passed when $C^{'}_k$ ($\neq C_k$) is chosen as root. Then we have*

$M_{i \leftarrow j} = M^{'}_{i \leftarrow j}$ and $M_{i \rightarrow j} = M^{'}_{i \rightarrow j}$.

That is, given a rooted message passing scheme, the messages passed between a given pair of neighboring clusters remains the same no matter which cluster is chosen as the root.

4.3 JT Cluster Prior Calculation

4.3.1 Minimum Messages for Prior

As we have revealed the semantic meaning of the messages in the GP method, it is straightforward to determine how many messages a cluster requires to form the cluster marginal from its neighbouring clusters. We apply this result with the Hugin GP message passing to establish the correctness of our marginal calibration method. Then, we present an algorithm based on the Shenoy-Shafer message passing architecture for computing the cluster prior marginal.

First we consider the Hugin GP message passings, which originally requires the passing of two messages over each separator. By applying the ASM procedure before any evidence is incorporated, we are able to determine the messages required passing over a Hugin separator for forming the prior marginal. We call one such message an ASM message, which contains the potential of either the complete separator marginal, or the factorization of the separator marginal.

We can thus form the prior marginal for each cluster in a JT following the same message calculation procedure, as described in the *inward* and *outward* stages, but with only ASM messages calculated.

Theorem 3 *Given the result of applying the ASM procedure to a JT T. After the Hugin GP method with only ASM messages passed, the potential of each cluster C of T forms the prior marginal.*

The proof is straightforward based on Theorem 3 and Corollary 1. During the Hugin GP message passing, all non-ASM messages can be ignored as they contain unity potential 1. Also, the content of an ASM message is the same as a Hugin GP message, thus the correctness of each message calculation holds. This results in a consistent JT with each cluster representing the prior after passing only ASM messages during the GP propagation.

We now turn to the Shenoy-Shafer architecture with the application of the ASM procedure. The result of Theorem 3 holds in the Shenoy-Shafer architecture as each message between a pair of neighboring clusters, no matter Hugin or Shenoy-Shafer, contains essentially the same potential. The difference between the two approaches lies in just how these messages are absorbed and manipulated over the separator. Thus, the Shenoy-Shafer message passing with ASM messages will guarantee prior marginal in each JT cluster as well. Moreover, the calculation of an ASM message is the same as one for a regular Shenoy-Shafer message.

Algorithm 14 $Cal\_JT\_Marginal$

Let $C_i (i = 1, ..., n)$ be a cluster of a JT T. Based on the result of the ASM procedure, the set of C_i's incoming messages and outgoing ASM messages are P^i_{in} and P^i_{out} respectively. When called, the calculation is carried out at each cluster C_i as follows:

```
1. while( $\{P^i_{in}\} \neq \emptyset$)
2. {   wait for incoming messages;
3.   if received an incoming message $P_i$
4.       set $\{P^i_{in}\}$ = $\{P^i_{in}\}\backslash\{P_i\}$;
5.   if ( $|\{P^i_{in}\}| == 1$ with $P_j \in P^i_{in}$ from cluster $C_j$ &&
6.           $\exists\, P_k \in P^i_{out}$ to agent $C_j$ )
7.       send a Shenoy-Shafer message to $C_j\}$
8. send remaining outgoing Shenoy-Shafer messages;
9. output cluster prior marginal
```

By maintaining the two message buffers between each pair of adjacent clusters, we can form the marginal for each cluster by combining the originally assigned CPDs of a cluster with the ASM messages. This informed message passing allows more efficient message propagation among all clusters under the Shenoy-Shafer architecture.

Consider the example in Figure 4.2, it is noted that for every cluster in the JT, either it needs to send the separator marginal or the factors in its factorization to its neighboring clusters once the cluster marginal is known, or it needs to receive the allocated separator marginal or the factors in its factorization from its neighboring clusters in order to transform the cluster potential into the cluster marginal. For the former case, cluster $C_1 = ac$ must send $P(c)$ to cluster $C_3 = cdf$ if $P(ac)$ is known. For the latter case, cluster $C_3 = cdf$ must receive $P(c)$ and $P(f|d)$ from clusters C_1 and $C_4 = def$, respectively.

Some cluster potentials are cluster marginals automatically, without the need to receive anything from its neighboring separators. For example, the cluster potentials for $C_1 = ac$ and $C_2 = bde$ in Equation 4.1 are already marginals, as shown in the first two rows in Table 4.1. Once $P(ac)$ and $P(bde)$ are available, they can then send the needed separator marginals $P(c)$ and $P(de)$ to the cluster potentials C_3 and C_4, respectively.

At this point, cluster potentials for C_3 and C_4 further need the factors in the factorization of the separator marginal $P(df)$ from each other. Cluster C_3 needs the factor $P(d)$ to transform $\Phi(C_3$ into $P(C_3)$, and cluster C_4 needs $P(f|d)$ to transform $\Phi(C_4$ into $P(C_4)$. If $P(C_4)$ is known, then $P(d)$ can be supplied to C_3; if $P(C_3)$ is known, then $P(f|d)$ can be supplied to C_4. Both $P(C_3)$ and $P(C_4)$ are unknown at this point, but based on the Shenoy-Shafer message calculation the messages between C_3 and C_4 can be easily obtained. Recall that in order to obtain a Shenoy-Shafer message from cluster C_j to C_k, we multiply together the original assigned CPDs at C_j and all messages from C_j's neighboring clusters except the one from C_k, then we sum out of this product all attributes that are not in the connecting separator between C_j and C_k. Thus, the messages passed between C_3 and C_4 are computed directly with this rule counting only the ASM messages.

Once $P(C_3)$ and $P(C_4)$ are available, they can then send the separator marginals $P(f)$ and $P(ef)$ to clusters C_5 and C_6 respectively. Receiving the needed separator marginals $P(f)$ and $P(ef)$, $\Phi(C_5)$ and $\Phi(C_6)$ become $P(C_5)$ and $P(C_6)$ as shown in the 5th and 6th row in Table 4.1.

4.3.2 Informed JT Initialization

The current initialization of a BN JT assigns the CPD of a node randomly to any qualified clusters, which contain the node itself and its parents. If multiple options exist, assign arbitrarily. For example, consider the BN and its JT in Figure 4.3. The CPD for some nodes can be assigned optionally. For example, the CPD of a, $P(a)$, can be assigned to clusters abc, ace or ade, and the CPD of e, $P(e|c)$ can be assigned to either cluster ace or cluster ceg. With the existing initialization, the choice is made randomly.

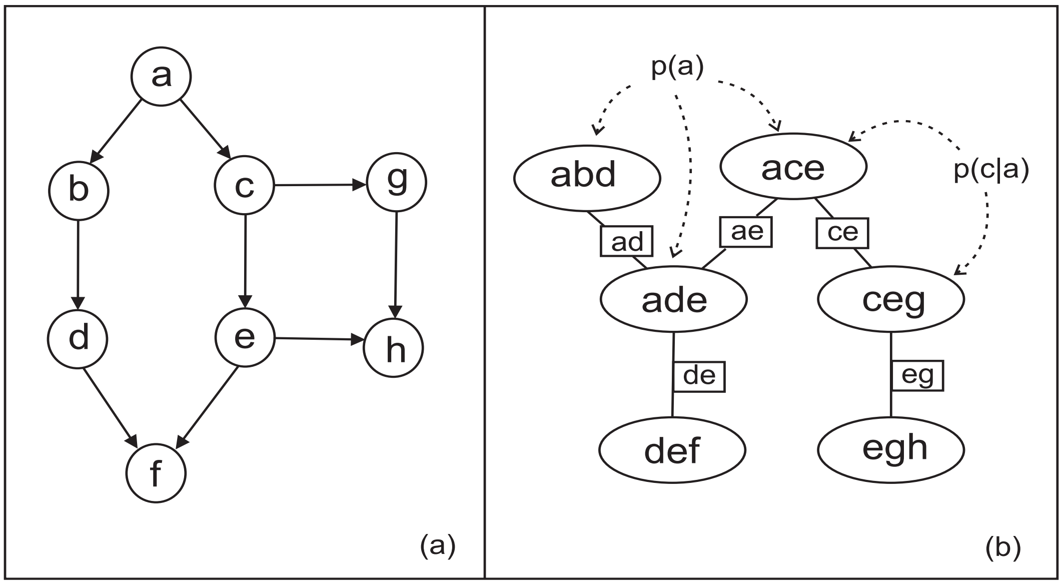

Figure 4.3: (a) A sample BN. (b) The corresponding JT with multiple initialization options.

In order to improve its efficiency, many modifications on both the Hugin and the Shenoy-Shafer inference architectures have been proposed [21] [78] [67]. However, the initialization stage has remained the same. We are thus motivated to exploit further savings on message passing by assigning the CPD more deterministically. For example, if we assign the CPDs $P(a)$, $P(c|a)$ and $P(e|c)$ to cluster ace, its cluster potential forms the marginal over the three nodes, a, c, and e, immediately. This observation leads to the following proposition whose validity is straightforward.

Proposition 5 *If a cluster C in a JT has n nodes, and during the initialization phase if it receives n CPDs then the cluster potential will be the cluster marginal automatically before the GP method is performed. That is*

$$\Phi(C) = P(C) = \prod_{i=1}^{n} P(x_i|Pa(x_i))$$

Our heuristic is based on the above proposition. When a node's CPD can be assigned to more than one cluster of the JT, i.e. the node itself and its parents are present in more than one cluster of the JT, we assign the node according to the rule below.

Initialization heuristic

Among the clusters of the JT where the node itself and its parents are present, the CPD of that node should be assigned to a cluster that has the least number of nodes, but with the most CPDs already assigned.

During the initialization phase, with this heuristic, we first try to identify the CPDs of nodes that can be assigned to more than one cluster. Meanwhile, the CPDs of nodes, which have only one cluster to be qualified for receiving the node, are assigned to that particular cluster immediately. Then, for each node with optional clusters, we search for the cluster that is smallest in terms of number of nodes but has the highest number of CPDs already assigned to it. This simple rule helps us to form the marginal for certain clusters before the GP method is applied, and simplifies message passing to the greatest extent.

If we consider the allocated marginal or the factors in its factorization received by a cluster from its neighboring cluster as a message, then it is easy to verify that there is no need to pass $2(n-1)$ messages as in the GP method (recall that n denotes the number of clusters in a JT). For example, applying the GP method on the example in Figure 4.3 requires passing 10 messages. The GP method neglects the fact that messages can be saved because the semantic meanings of the messages in the GP method are irrelevant. By applying the ASM procedure on the same example only 8 messages are needed. Then applying the informed assignment of CPDs with our heuristic, we need only 7 messages for the same example. In a real life network, this saving of the number of messages could result in significantly less arithmetic computation compared with the GP method.

To summarize, our method of marginal calculation is based on the procedure of ASM and informed initialization. The ASM procedure provides us with the actual potential needed to form the marginal so there is no need to pass blindly two messages over each separator. We can achieve the minimum message passing given a certain JT initialization. Moreover, the initialization heuristic increases the chance of a cluster to form the marginal automatically, thus further reducing the messages to be passed between the clusters. Combining the two techniques, our method can significantly improve the cost for JT marginal calculation.

4.4 Experimental Results

We conducted experiments on a number of BNs, collected from the publicly available HUGIN repository. These networks can be located at the http://www.cs.huji.ac.il/labs/compbio /Repository/ and http://forum.HUGIN.com/index.php?board=12.0. All the implementations were done with C/C++ Eclipse IDE on Windows operating system, with 512MB of RAM and an Intel 1.4 GHz processor.

As the message passing is the most important operation in the HUGIN architecture, the number of messages occurring during GP has a direct impact on the performance. Thus, in our first experiment, we used the criteria of the total message required to evaluate the efficiency of our method. We implemented the traditional HUGIN architecture as the base for the comparison, the ASM procedure alone applied on HUGIN architecture, and our complete method with ASM and initialization heuristic applied.

Network			**Total Messages**			**% of savings**
	nodes	clusters	Hugin Method	ASM Method(Only)	Our Method	**Hugin v.s. Our Method**
Asia	8	6	10	6	6	40 %
Diabetes	413	337	672	621	609	9.38 %
Car_ts	12	6	10	5	5	50 %
Mildew	35	29	56	47	44	21.43 %
4sp	58	41	80	58	55	31.25 %
Barley	48	36	70	59	56	20 %
Munin2	1003	866	1730	1190	1171	32.31 %
Munin3	1044	904	1806	1220	1207	33.17 %
Munin4	1041	872	1742	1163	1143	34.39 %
water	32	19	36	22	18	50 %
studfarm	12	9	16	15	13	18.75 %
pigs	441	368	734	713	698	4.91 %

Table 4.2: Comparison of message counts on various networks

From the experiment results presented in Table 4.2, it is clear that when ASM is applied, the total amount of message passing was reduced considerably. Further saving was achieved if our initialization heuristic was also used. The results also confirmed that by utilizing the semantic meaning of the messages, we saved up to 50% of messages compared to the use of the Hugin method. As fewer message counts would result in fewer computations, our methods definitely have improved the performance of the HUGIN architecture.

In our second experiment, we looked into the actual arithmetic operation in the Hugin and our method. A single message pass in HUGIN requires three arithmetic operations: summations, multiplications and division. The summation operation sums out the variables in the given set and returns a potential defined over a smaller set of variables. The division operator acts on two potentials and returns a quotient potential. By simply applying the ASM procedure, a single message passing is carried out by multiplying the originally assigned CPDs, with the allocated separator marginals or the factor in its factorization. That is, whereas HUGIN requires a substantial number of the divisions along with summation and multiplication, our method with ASM procedure only requires 2 operations: summations and multiplications. Table 4.3 shows the number of arithmetic operations needed by

Network	Hugin Method(+, *, /) (+, *, /)	Our Method(+, *) (+, *)	**% of savings**
Asia	159	86	45.66%
Diabetes	64233781	57615326	10.30%
Car_ts	202	96	52.48%
Mildew	21157394	16589402	21.59%
4sp	199732	135640	32.09%
Munin2	27797088	18392860	33.83%
Munin3	33698532	22130570	34.33%
Munin4	661845513	249624860	35.34%
water	21323694	10657956	50.02%
studfarm	504	371	26.49%
pigs	6343971	5952050	6.18%

Table 4.3: Comparison of arithmetic operation counts on various networks

the GP method of HUGIN architecture and our method. Clearly, fewer messages indeed result in less computation as demonstrated with this experiment. It is worthwhile to note that, while the saving of message count was no more than 50%, the saving of actually arithmetic operation can exceed that limit, as in Car_ts and Water networks.

Finally, we measured the time efficient of the Hugin propagation and our method in the calculation of the prior marginal with all the networks. The experimental data is shown in Table 4.4. We can see that the saving in percentage is consistent with the result comparing arithmetic operations, and the saving could be over 50% in certain networks. Again, it is confirmed that the propagation based on the separator analysis and informed initialization is considerably more efficient than the GP method.

4.5 Discussion

The *global propagation* (GP) [35] method used in the Hugin architecture [53] is arguably one of the best methods for probabilistic inference in Bayesian networks. Passing messages between clusters in a JT is the basic operation in the GP method. It is traditionally considered that the messages passed are simply potentials without any specific semantic meaning. We studied the factorizations of a joint probability distribution defined by a

Network	**Propagation Time(sec)**		**% of savings**
	Hugin Method	Our Method	
Asia	0.003	0.0016	46.67%
Diabetes	29.037	25.29	12.91%
Car_ts	0.003	0.0013	56.67%
Mildew	5.057	3.51	30.60%
4sp	0.05	0.02	60%
Munin2	16.494	11.16	32.34%
Munin3	14.317	10.204	28.73%
Munin4	221.353	195.095	11.86%
water	16.394	8.03	51.02%
studfarm	0.006	0.0042	30%
pigs	4.186	3.09	26.18%

Table 4.4: Comparison of propagation time on various networks

Bayesian network *before* and *after* the GP method is performed, we investigated the messages passed algebraically, and we make the following two contributions. (a) We reveal that the messages passed are not mere potentials, but in fact separator marginals or factors in their factorizations. (b) We demonstrate that the revealed semantics of the messages can be utilized to avoid passing up to half of the messages that could have required passing by the GP method during prior calculation.

We have studied the messages passed in the GP method algebraically. It was revealed that the messages are actually separator marginals or factors in their factorizations. Passing messages in the GP method can be equivalently considered as the problem of allocating separator marginals. This different perspective of propagation gives rise to a different idea of computing cluster marginals, which is realized in the ASM procedure. When applied, the ASM procedure provides the detailed information of the messages a cluster requires to form the marginal, thus avoiding passing two messages over each JT separator blindly. Also, we have presented an initialization heuristic, based on the observation that cluster potential could be the cluster marginal automatically during initialization. This enables us to pass even fewer messages during propagation.

In our next work, we will extend the ASM procedure to the calculation of posterior distribution. Observed evidence affects the CIs in the original network, such that the d-separation between sets of node may change based on Definition 1. This also influences the semantic meaning of messages, which will contain different information than the ones passed during the prior calculation. In fact, the content of such messages will vary depends on the given set of evidence. The Lazy inference [57] has used some induced dependency to facilitate message passing during posterior calculation. Our plan is to extend an algebraical approach from our current results to conduct a formal analysis of these messages.

Chapter 5

Fast Marginal Calibration

In Chapter 4, we described how to calculate the marginal in JTs of single agent Bayesian networks, and presented an algorithm that forms the prior in a JT cluster with informed message passing. The results will be extended in this chapter to solve the problem of MSBN LJF marginal calibration.

Although the existing MSBN LJF-based exact inference algorithms [91, 92, 98] use different message calculation schema, or storage allocation, they typically consist of an LJF initialization process conducted before the global propagation. During this initialization stage, the CPDs of all domain variables are assigned to appropriate LJF subnets and in each of them forms an initial potential. However, this initial potential does not provide the complete information, thus preventing the agent to reason about its own problem subdomain correctly at this stage. Obviously, the agent's local JT is not yet consistent. Even with local message passing, each subnet's local potential still does not represent the JPD, or prior marginal, over the local variables.

MSBN LJF *marginal calibration* refers to the process during which each LJF subnet is supplied with the necessary information to form a complete prior marginal representation for local variables. The calibration process must be conducted before the global propagation can be performed in an MSBN LJF. Furthermore, an MSBN subnet equipped with its local prior can potentially admit a wide range of approximation techniques. For example, a calibrated subnet is essential if we intend to apply stochastic sampling techniques in an agent's local subnet.

All MSBN LJF inference algorithms perform an implicit marginal calibration process, which is typically expensive in terms of external and internal message passings. With the existing Hugin-based global propagation architecture, the calibration process requires the calculation of two inter-agent messages over each LJF hyperlink. Moreover, each of such message passings requires local consistency in the sender's local subnet, which calls for repeated calculation of intra-agent messages. Overall, extensive amount of inter-agent runtime communication and local calculation is necessary. Even with our extended Shenoy-Shafer architecture, although the total number of local messages is reduced, the same amount of inter-agent messages, which are two over each hyperlink in the LJF, are still needed during the calibration process.

In this chapter, we present a method of fast MSBN LJFs calibration which optimizes the messages passing among agents. Based on the concept of LJF local *prior marginal* (PM) factors, we can greatly reduce the amount of inter-agent messages required. These factors, which are information required to form a complete prior marginal distribution in each LJF local subnet, consist of the initially assigned CPDs and some factorization of the subnet's hyperlinks connecting to its neighbors. Based on a compile time analysis of these factors, called *hyperlink analysis*, we can guide the actual message passing at runtime with the minimum inter-agent communication. Moreover, we utilize our new LJF inference architecture for the message calibration process, so we are able to compose inter-agent messages more efficiently with partial local updates. [1]

5.1 Hyperlink Analysis

5.1.1 Local PM Factors

Recall that during initialization of an LJF, exactly one of all occurrences of a variable x (in a subnet containing $\{x\} \cup Pa(x)$) is assigned the CPD $P(x|Pa(x))$. All other occurrences are assigned a unity potential. Also, a unity potential is assigned to each separator in each local JT and each linkage in each linkage tree. The initial potential of a local JT is either

[1]For clarity, the term *global propagation* or global message passing used for a JT in Chapter 3 will be used to refer to the inter-agent message passing in the LJF.

the product of all of its assigned CPDs, or 1 if no CPD is assigned. In general, the initial potential does not provide the complete information for an agent to correctly reason about its own problem subdomain.

The main purpose of inter-agent message passing is to perform MSBN multi-agent belief update with some observed evidence. However, with all existing algorithms, the same method is also used to calibrate the prior marginal in each local subnet. The inter-agent messages passed for marginal calculation and for evidence propagation are not distinguished. Agents are thus penalized with the high cost of message calculation during the process of marginal calibration, which indeed should be treated as part of the LJF initialization stage. This motivated us to extend our results from Chapter 3 in order to investigate the composition of the prior marginal of each subnet and the semantic meaning of messages between agents. Given the saving on the local computational cost already achieved with our extended Shenoy-Shafer architecture, we hope to further reduce the $2(n-1)$ inter-agent message passings in an MSBN of n agents during the marginal calibration process.

We first introduce the concept of prior marginal (PM) factors and explain how these factors contribute to the marginal calibration of each MSBN subnet. We show that the inter-agent message passing during calibration is essentially the transmission of the PM factors. Through a process of hyperlink analysis at compile time, an agent is provided with the knowledge of the exact composition of its incoming and outgoing PM factors. This information will facilitate the actual message passings during the calibration.

Essentially, an MSBN hypertree can be viewed as a JT which is constructed from a high level BN. This JT integrates all MSBN subnets, such that the JT clusters correspond to the hypernodes and the JT separators correspond to the hyperlinks. For example, the three MSBN subnets in Figure 2.6 (b) can be combined into a BN like the one shown in Figure 2.6 (a). The hypertree shown in Figure 2.6 (c) may also be considered a JT constructed from this BN. For simplicity, we ignore the internal structure of local JTs and linkage trees, and explain the idea of MSBN marginal calibration using this BN example.

Consider Figure 5.1 (a), which shows a JT of the BN in Figure 2.6 (a) and its initial CPD assignment. Let V represent all the random variables. The JPD $P(V)$ can be factorized as follows:

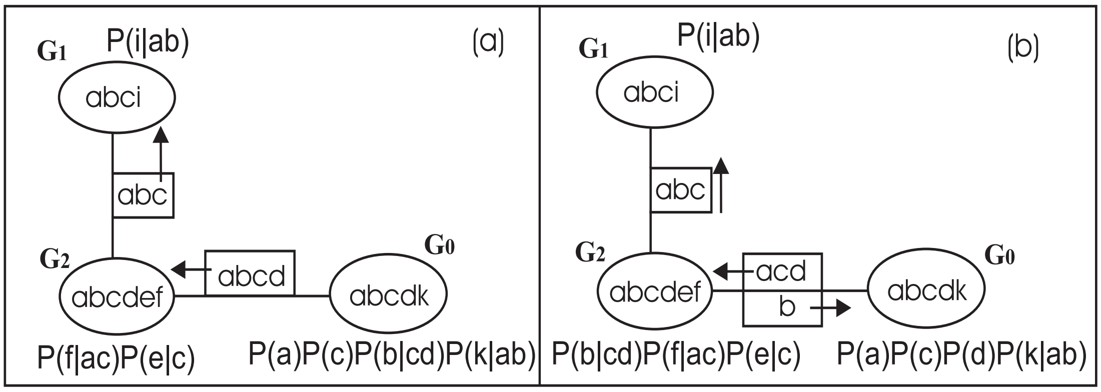

Figure 5.1: Conceptual view of PM factors in an LJF.

$$P(V) = \overbrace{[P(i|ab)]}^{\Phi(G_1)} \cdot \overbrace{[P(f|ac) \cdot P(e|c)]}^{\Phi(G_2)} \cdot \overbrace{[P(a) \cdot P(b|cd) \cdot P(c) \cdot P(d) \cdot P(k|ab)]}^{\Phi(G_0)}. \quad (5.1)$$

Each square bracket in Equation 5.1 expresses the composition of a potential for the clusters in the JT.

Furthermore, $P(V)$ can also be factorized as follows [92]:

$$P(V) = \frac{P(abci) \cdot P(abcdef) \cdot P(abcdk)}{P(abc) \cdot P(abcd)}. \quad (5.2)$$

In order to transform Equation 5.1 into Equation 5.2 algebraically, one first needs to multiply $\frac{P(abc) \cdot P(abcd)}{P(abc) \cdot P(abcd)}$ to the right hand side of Equation 5.1. By doing so, we get

$$P(V) = \overbrace{[P(i|ab)]}^{\Phi(G_1)} \cdot \overbrace{[P(f|ac) \cdot P(e|c)]}^{\Phi(G_2)} \cdot \overbrace{[P(a) \cdot P(b|cd) \cdot P(c) \cdot P(d) \cdot P(k|ab)]}^{\Phi(G_0)} \cdot \frac{P(abc) \cdot P(abcd)}{P(abc) \cdot P(abcd)}. \quad (5.3)$$

Obviously, Equation 5.3 is almost identical to Equation 5.2, except that it has a different numerator. Therefore, we try to group the terms $P(abc)$ and $P(abcd)$ with other terms of the numerator in Equation 5.3, in order to reach Equation 5.2. A careful examination reveals that in Equation 5.1 the potential $\Phi(G_0) = P(abcdk)$ already. Next, once the term $P(abcd)$ in Equation 5.3 is assigned to $\Phi(G_2)$, we have $\Phi(G_2) \cdot P(abcd) = P(abcdef)$. Finally, the term $P(abc)$ in Equation 5.3 is assigned to $\Phi(G_1)$, which results in $\Phi(G_1) \cdot P(abc) = P(abcdi)$. These assignments are shown below, where the newly assigned terms are underlined:

$$\overbrace{[P(i|ab) \cdot \underline{P(abc)}]}^{\Phi(G_1)=P(abci)} \cdot \overbrace{[P(f|ac) \cdot P(e|c) \cdot \underline{P(abcd)}]}^{\Phi(G_2)=P(abcdef)} \cdot$$
$$\overbrace{[P(a) \cdot P(b|cd) \cdot P(c) \cdot P(d) \cdot P(k|ab)]}^{\Phi(G_0)=P(abcdk)}. \qquad (5.4)$$

From this example, we can see that the prior marginal $P(G)$ of a JT cluster G consists of terms from G's initially assigned CPDs and terms from the separator factorization. We define the *prior marginal* (PM) factors of an MSBN subnet N as all the terms required in order to form a marginal representation, or JPD, over N's local variables. They include N's original CPD assignment and/or some factorizations of N's hyperlinks.

Definition 12 *Prior Marginal(PM) Factors*

Let L be an MSBN LJF and N be a subnet of L with local variables V. Let B be a BN with the identical DAG of N. The factorized JPD representation of B is $P(V) = \prod_{N}^{i=1} \Phi_i$, where Φ_i, $i \in N$ is the set of CPDs in the form of $P(X_i|Pa(X_i))$.

The prior marginal factors $\Psi_i (1 \le i \le k)$ is the set of CPD or product of CPDs of Φ_i, such that $P(V) = \prod_{k}^{i=1} \Psi_i$.

If each variable in an MSBN subnet is assigned a CPD during the initialization process, the prior marginal of this subnet can be obtained immediately once it is locally consistent. In this case, the subnet's PM factors are complete and no other terms are required. Indeed, such a local JT represents the prior distribution of the corresponding BN for the DAG of the MSBN subnet. This observation is described in the following proposition, whose proof is trivial.

Proposition 6 *Suppose a local JT in an LJF contains n nodes, and during the LJF initialization phase it receives n CPDs. Then, the local JT will be calibrated as marginal automatically after a round of local propagation, and without any inter-agent message passings.*

Unfortunately, in most situations, the assigned CPDs compose only partial PM factors of an MSBN subnet. The rest of the PM factors are in the form of hyperlink factorization and must be provided by adjacent subnets through inter-agent message passing. That is, even the local JT can become consistent with a round of local propagation, the resulting potential of each cluster is not a prior marginal distribution.

For example, the Equations 5.3 and 5.4 demonstrate the algebraic transformation from Equation 5.1 to Equation 5.2. From an algorithmic perspective, this algebraic transformation amounts to two external message passings for the missing PM factors, one from G_0 to G_2 and the other from G_2 to G_1 over their corresponding hyperlinks. Therefore, by analyzing algebraically the PM factors of each agent's subnet, we can identify all non-unify messages that need to be calculated during the marginal calibration process. This enables us to carry out the inter-agent communication more efficiently and in a well-informed manner.

5.1.2 Hyperlink Analysis: Centralized v.s. Distributed

By ignoring the internal structure of local JTs and linkage trees during the discussion of this above example, we have followed the idea of the ASM procedure in Chapter 3 in our analysis. Indeed, we can naturally extend the ASM procedure to the context of an LJF. The goal of hyperlink analysis is to provide us information at the compilation time regarding how external messages flow, in order to guide actual message passings at runtime. This process can be conducted either through a system coordinator with a certain amount of centralized control, or through the collaboration of all agents.

With the first solution, we conduct the hyperlink analysis similar to the ASM procedure by treating each hyperlink as a JT separator in an MSBN hypertree. Then hyperlink analysis is initiated by an MSBN system coordinator, who first selects candidate nodes in each subnet that are assigned a CPD and contained in at least one hyperlink connecting to an adjacent subnet. For each of these nodes, the system coordinator marks the direction of the node over the hyperlink toward the adjacent subnet, and repeats the process in the adjacent subnet until the node is no longer on any hyperlink connecting to other neighboring subnets. All nodes over each hyperlink will be marked with a direction. The final result is obtained by simply combining all nodes of the same direction over each hyperlink. It provides the total number of external messages required and the direction of each of them.

Algorithm 15 *Hyperlink_Analysis_Centralized*
Suppose an agent A with local JT T. When called after the LJF initialization process, A performs the following:

1. For each CPD $P(a_i|Pa(a_i))$, locate the local JT T to which it has been assigned. In all hyperlinks containing variable a_i, draw an arrow pointing away from T over a_i in the hyperlink.
2. Examine each hyperlink l in the LJF. If the variables in l_i all point to one neighboring subnet, then the linkage will be allocated as a whole to that neighboring subnet.
3. For a linkage l that has nodes pointing to both connecting subnets, then the linkage must be factorized so the resulting factors will be assigned to the appropriate subnet indicated by the arrows in the hyperlink.

For example, let us consider the initial CPD assignment shown in Figure 5.1(b). The CPD $P(b|cd)$ is assigned to the variable b in the subnet G_2. Therefore, we direct the variable b away from G_2 toward its two adjacent subnets G_0 and G_1. The CPDs $P(a)$ and $P(c)$ are assigned to the variables a and c in G_0 and exist on the hyperlink connecting G_0 and G_2 as well as G_2 and G_1. Thus, we direct the variables a and c away from G_0 to G_1. The final result of linkage analysis is shown with arrows in Figure 5.1(b), which indicates that two external messages are required between G_0 and G_2 during inference, but only one external message is needed from G_2 to G_1.

For the second solution, we forgo the centralized control. By examining the locally assigned CPDs and analyzing the hyperlinks connecting to their adjacent agents, agents can determine collaboratively the exact flow of PM factors.

Algorithm 16 *Hyperlink_Analysis_Distributed*

Suppose an agent A with its local JT T. When called after the LJF initialization process, A performs the following:

1. Determine based on proposition 6, if the T's assigned CPDs have a guaranteed marginal. If yes, exit.
2. In T, for each assigned CPD $P(a_i|Pa(a_i))$, if the head a_i presents in any of T's hyperlinks, send a_i over these hyperlinks to adjacent agents.
2. Upon receiving a node x from an adjacent agent, sent x to other agents if x presents on the corresponding hyperlinks.
3. Combine all the incoming nodes from a single neighboring agent. The set indicates a set of PM factors.

With Algorithm $Hyperlink\_Analysis\_Distributed$, an agent first determines if its set of PM factors is completed by simply matching all the head nodes of the assigned CPDs to its local variables. Consider the three subnets from the hypertree shown in Figure 5.1 (a). G_0's PM factors are complete with all the assigned CPDs, since the head nodes of all the CPDs are already equal to the set of local variables. On the other hand, the assigned CPDs of subnets G_1 and G_2 contribute only the partial PM factors, and the rest will be provided by hyperlink factorizations. For example, the PM factor over variable $\{a, b, c\}$ is missing in G_1 and needs to be obtained from the factorization of its hyperlink to G_2.

In order to decide the actual flow of PM factors, agents collaborate to provide each other their already obtained PM factors through corresponding hyperlinks. First, each agent selects all the local variables that are assigned CPDs and are also contained in at least one hyperlink. Then the agent will "push" these variables over the hyperlinks towards the adjacent agents. The set of variables that are "push"ed away represent the PM factors the agent will provide to its neighbors. Upon receiving some "push"ed-in variables over a hyperlink, an agent checks if the variables are contained in its *other* hyperlinks. If so, it means the same factors are also required by other agents, and the agent will "push" these variables further toward them. This whole process can be viewed as broadcasting each agent's relevant CPD information to its immediate neighbors, and eventually across the whole network.

Once the PM factor analysis is finished, every variable on each hyperlink is "push"ed toward one of the hyperlink's two connecting agents. The variables that are going in the same direction over a hyperlink represent a single inter-agent message. From an agent's point of view, this analysis result provides the number and context of the agent's runtime inter-agent messages. This process is illustrated in Figure 5.1 (a). Once G_2 receives the information over $\{a, b, c, d\}$ from G_0, it will push further to its neighbor G_1 the variables $\{a, b, c\}$ contained in the hyperlink. The flow of PM factors is shown by arrows over the hyperlinks. A total of two runtime inter-agent message passings will be transmitted.

In Algorithm $Hyperlink\_Analysis\_Distributed$, no centralized control is needed, and thus it is more preferable in a multi-agent environment. Although some inter-agent communication is required, the amount of information exchanged is negligible compared to actual runtime cost of inter-agent message passings.

5.2 LJF Marginal Calibration

5.2.1 PM Messages

Once agents have the knowledge of their current and missing PM factors, we can carry out the LJF marginal calibration process to supply each agent the corresponding missing factors. We call the inter-agent message passed during this process the *PM messages*. A PM message is the message that needs to be delivered over one direction of a hyperlink during the marginal calibration. It can consist the potential of one PM factor, or the product of a set of PM factors. Given each LJF hyperlink and the hyperlink analysis result, we further define two types of PM messages to facilitate message calculation.

- PM message type I: A PM message that is composed of all variables over the corresponding hyperlink.
- PM message type II: A PM message that is composed of a subnet of variables of the corresponding hyperlink.

For example, based on the result of hyperlink analysis in Figure 5.2(a), we obtain three PM messages, which are marked by the arrows. The PM message over the hyperlink between G_1 and G_2 is a type I message. The two PM messages over the hyperlink between G_2 and G_0 are of type II.

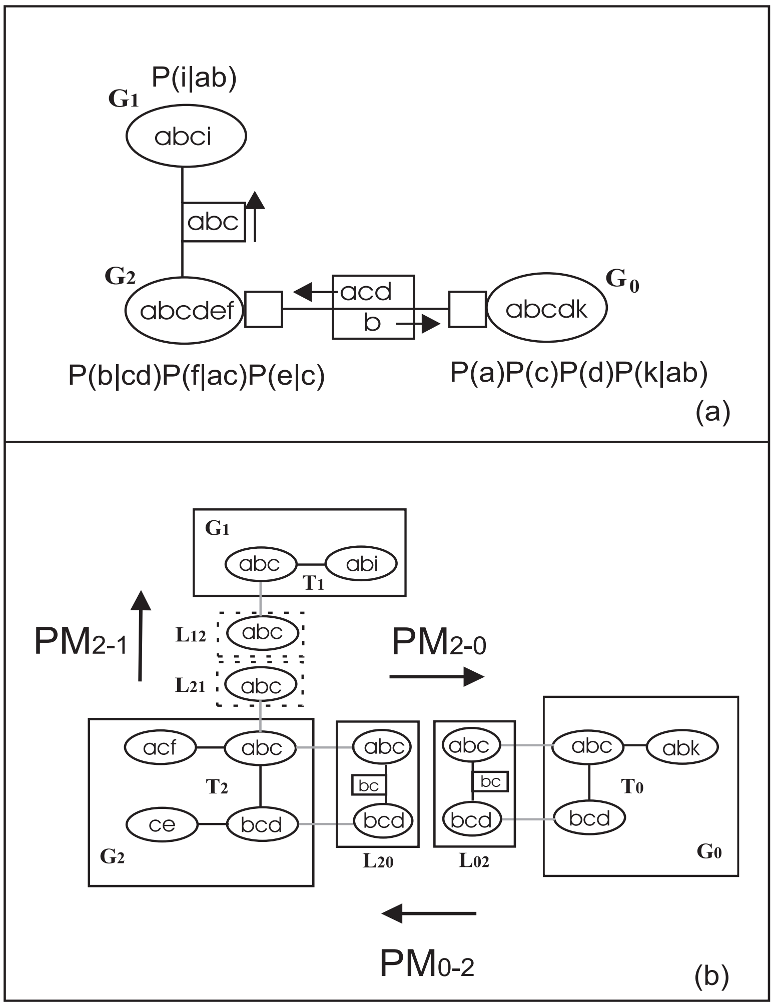

Figure 5.2: Passing of PM messages. (a) A conceptual view, and (b) Actual calculation.

A type II message needs to be prepared when the local JT has not received the complete set of incoming PM Messages. We can easily notice the similarity between the situations when a type II message is calculated and Rule 2 of the LJF global propagation in our extended Shenoy-Shafer architecture. Therefore, we use the operation described in Algorithm Cal_Single_MSG from Chapter 3, with only a minor modification. That is, we change step 1 of Algorithm 8 Cal_Single_MSG, such that we absorb only the buffered PM messages. We present the algorithm for calculating a type II PM message as follows:

Algorithm 17 $Cal\_PM\_MSG\_II$

Let A_i and A_j be two adjacent agents. A_i's local JT is T_i with the set of clusters C. T_i maintains a linkage tree L to A_j. In T_i,, H is the set of linkage hosts of L and the extended linkage host tree is T_e. $L(i = 1, ..., n)$ are T_i's linkage trees that are connecting to adjacent agents except A_j, and over which PM messages will arrived. Based on the result of hyperlink analysis, A_i calculates a type II PM message to A_j is calculated as follows.

1. For all linkage trees $l \in L$, A_i absorb the extended linkage potential and update host belief:

2. In T_i, randomly select a linkage host $C_r \in H \subset C$ as the root of T_e as well as T_i. Direct all clusters away from C_r in T_i.

3. Perform a full inward message passing on C_r in T_i, such that C_r calls recursively all child clusters to send an inward message until reaching the leaf clusters.

4. Perform a partial outward message passing on C_r within the context of extended linkage host tree T_e, such that H_e sends outward messages to all linkage hosts recursively.

5. For each linkage in L, calculate corresponding extended linkage potential to compose the PM message to A_j.

5.2.2 Calibration with Minimum PM Messages

Guided with the result of PM factor analysis, agents process only the necessary PM messages during the marginal calibration. We have presented how to calculate a type II PM message without maintaining the local consistency. Here we present our fast marginal calibration algorithm described as follows.

Algorithm 18 *Marginal_Calibration*

Let $N_i(i = 1, ..., n)$ be the subnets and $\mathcal{H}$ the corresponding hypertree of an MSBN M, which is populated by multiple agents with one at each subnet. Each agent A_i has a JT T_i

constructed in its local subnet N_i. Based on the result of hyperlink analysis, the set of A_i's incoming PM messages and outgoing PM messages are P_{in} and P_{out} respectively. When called, each agent A_i performs the following:

```
1. while( $\{P_{in}\} \neq \emptyset$)
2. {  wait for incoming PM messages;
3.      if received an incoming PM message $P_i$
4.          set $\{P_{in}\}$ = $\{P_{in}\} \backslash \{P_i\}$;
5.      if( $|\{P_{in}\}| == 1$ with $P_j \in P_{in}$ from agent $A_j$ &&
6.              $\exists\ P_k \in P_{out}$ to agent $A_j$ )
7.          send a message to $A_j$ with $Cal\_PM\_MSG\_II$ ;  }
8. perform local message passing in $A_i$;
9. send all remaining outgoing PM messages;
```

In our calibration algorithm, agents with missing PM factors wait for the arrival of these factors as incoming PM messages. A type II message can be calculated at this stage if an agent has received all except one PM message. When an agent has received all PM messages, it has complete its set of PM factors, which includes the ones that are initially assigned and ones that arrive as PM messages. Then, the local JT is able to send out its outgoing PM messages, which could consist both type I and type II messages. A simple solution is to perform local propagation in the JT to achieve local consistency, then deliver all the messages through the standard message passing operation described in Algorithm $Deliver\_Through\_Linkage$ under our new architecture. The complete local propagation, which is to be carried out to form the local prior marginal, is only performed after all PM factors have arrived.

For example, consider the MSBN LJF shown in Figure 5.2 (b). Even though none of the three subnets is initialized with a complete set of PM factors, G_0 and G_2 are able to send each other a type II message with Algorithm $Cal\_PM\_MSG\_II$. The messages, consisting of PM factors over $\{a, c, d\}$ and $\{b\}$, bring the potential of subnets $\Phi(G_0)$ and $\Phi(G_2)$ into prior marginal $P(abcdk)$ and $P(abcdef)$ respectively. Next, G_2 will pass a type I message to G_1 so the potential $\Phi(G_1)$ becomes marginal $P(abci)$ as well. A total of three PM messages are passed, compared to the four messages required by Hugin-based LJF inference architectures.

The following theorem states the correctness of our algorithm.

Proposition 7 *All local JT subnets in an LJF will be calibrated with a complete prior marginal when Algorithm $Marginal\_Calibration$ terminates.*

Proof:

The proof can be easily extended from JT cluster marginal calculation in Chapter 4. As an LJF can be viewed as a high level JT, each local JT as a whole can also be viewed as a JT cluster. The process of hyperlink analysis correctly produces the PM factors missing from each cluster, similar to the ASM procedure. The correctness of PM message calculation, particularly Algorithm $Cal\_PM\_MSG\_II$, is based on the extended Shenoy-Shafer message calculation in Chapter 3. Algorithm Cal_Single_MSG differs from Algorithm $Cal\_PM\_MSG\_II$ only at the number of incoming message buffers that are incorporated during the calculation, and thus its proof extends immediately to Algorithm $Cal\_PM\_MSG\_II$. □

Proposition 8 *Algorithm $Marginal\_Calibration$ requires only the minimum inter-agent message passing in an LJF for marginal calibration.*

We can verify the validity of the above proposition extending from Corollary 1 in Chapter 4. It is easy to verify that the messages that we have avoided calculation indeed consists of all unity potential. The inter-agent messages that are required passing with Algorithm $Marginal\_Calibration$ are the minimum set for the calculation of prior marginal marginal.

It is worthwhile to mention that, we could utilize an optimal CPD initialization to further reduced the number of required messages. For example, the assignment of CPD $P(b|cd)$ to either G_2 or G_0 in Figure 5.1 will result in different number of PM messages. If $P(b|cd)$ is assigned to G_0 as shown in (a), there will be total two messages, whereas three messages are needed if $P(b|cd)$ is assigned to G_2 as shown in (b). Based on our initialization heuristic in Chapter 4, we can extend a similar method in the context of an LJF. We omit further discussion as such an extension is straightforward.

MSBNs			No. of External Messages		No. of Local Propagation	
	Subnets	Variables	GP Method	Our Method	GP Method	Our Method
(a)	3	5	4	2	6	3
(b)	3	15	4	2	5	3
(c)	4	16	6	3	9	4
(d)	5	12	8	4	11	5

Table 5.1: Comparison of the MSBN GP and the MSBN SA-GP methods on various MSBN networks

5.3 Experimental Results

The computational efficiency of different propagation algorithms may be compared by the actual number of arithmetic operations. In [53], the number of additions, multiplications and divisions are used as a measure for the comparison of different BN propagation algorithms. We apply a similar approach and use the number of internal and external messages as a crude measurement to the real calculation. In particular, we count the number of external messages generated over the whole network and the number of local propagations required. We have conducted preliminary experiments on four MSBN networks to compare our method to the Hugin-based method.

Table 5.1 shows the counting of external messages and local propagation. Given an MSBN network with n subnets, the number of external messages is $2(n-1)$ in the MSBN GP method. With our method, this number is reduced by 50% in all 4 networks. Considering the number of local propagations, our method requires local propagations to be performed once only in each subnet. However, since Algorithm $Cal\_PM\_MSG\_II$ involves internal message passings, we roughly count each of its invocations as half local propagation for the purpose of fair comparison. The result shows that the total number of local propagations has also been reduced by an average of 50% in our SA-GP method compared to the GP method with a randomly selected root. These preliminary tests on small networks have shown substantial savings of external and internal messages using our fast calibration method.

Networks			Actual Exec Time(sec)	
	No. of Variables	No. of Clusters	GP Method	Our Method
(a) 4sp	58	41	0.02	0.01
(b) link	724	581	69.219	54.568
(c) mildew	35	29	2.764	2.023
(d) munin3	1044	904	5.838	3.725

Table 5.2: Fast marginal calibration: performance comparison in execution time.

In our second experiment, we compared our method to the standard LJF message passing algorithm in [92]. Recall that an MSBN hypertree can be viewed as a high level BN JT, where each cluster corresponds to a hypernode and each separator to a hyperlink. Therefore, we have implemented a partial version of both calibration algorithms and tested them on several BN networks as preliminary experiments.

The two implementations, both written in C, were tested on a WindowsXP platform. In our experiments, we focused on the comparison of inter-agent communication costs, which can be indicated by the amount of message passing among BN JT clusters. We randomly initialized all networks in order to avoid any specific CPD assignments that would favor our algorithm. We counted the number of messages generated during runtime communication, and the actual CPU time of running the two programs.

The test results are reported in Table 5.2 with the standard algorithm named as the LJF-GP method. It shows that, in each network, the reduced number of inter-agent messages are consistent with the savings of actual execution time. The saving on number of all PM messages as well as the execution time is listed in Table 5.3.

Networks	Savings of our method	
	No. of PM MSG	Exec Time
(a) 4sp	31%	50%
(b) link	29%	21.17%
(c) mildew	17%	26.81%
(d) munin3	31%	36.19%

Table 5.3: Fast marginal calibration: Savings in total inter-agent message and execution time.

5.4 Discussion

In multi-agent probabilistic systems, such as an MSBN, agents need to be calibrated with a complete marginal in order to reason about their own problem subdomains immediately and correctly. In the existing algorithms, however, this process has been carried out simply through standard inter-agent message passings, resulting in costly inter-agent communication and local calculation.

In this chapter, we have introduced the concept of PM factors for a complete prior marginal distribution BN subnets. Based on a distributed analysis of these factors at compile time, we were able to guide the actual runtime inter-agent communication by sending only the necessary messages. Although the result of hyperlink analysis applies to both Hugin and Shenoy-Shafer message passing architecture, the latter is more suitable to a fast calibration algorithm. In particular, an agent could send an outgoing message with an inconsistent local JT. Given an MSBN LJF with an arbitrary CPD initialization, our distributed calibration algorithm requires the minimum inter-agent communication and local computation.

The improved time efficiency of our proposed algorithm was confirmed in the preliminary experiments. In our experiments, we were not able to investigate the local computational cost in our preliminary test. Nevertheless, considering that the current savings are obtained without taking into account the improved message calculation without local consistency, we would expect a more favorable result from a complete implementation.

Chapter 6

Local Adaptive Importance Sampling

The MSBN provides an exact model for cooperative agents to reason about the states of a distributed uncertain domain. Such a problem domain can be decomposed into subdomains, each individually represented and managed by a relatively lightweight single agent. Typically, inference in MSBN is carried out in an LJF. Agents communicate through messages passed over LJF linkage trees, and belief updates in each LJF local JT are performed upon the arrival of a new inter-agent message. The LJF provides a coherent framework for exact inference with MSBNs, and is known to support consistent local inference in the absence of MSBN system-wide message passing [93]. However, the computational costs may render such exact calculation impractical for larger and more complex problem domains. For example, a network may contain subnets that are too large to admit exact local representation. In fact, the global inference is exponential to the largest node of the LJF, which is the largest local JT. It is natural to consider the possibility of trading off exact inference against the calculation speed and communication cost with approximate approaches.

Although approximate techniques have been well developed in traditional BNs, the extension of these solutions to MSBNs has been very limited. The methods of two stochastic sampling techniques, forward sampling and Markov sampling, have been extended and compared with the LJF-based exact inference algorithms [93]. Both proposed algorithms forgo the LJF structure and sample an MSBN directly in the global context. It has been shown that such approximation indeed requires more inter-agent messages passing, and at the cost of revealing more private knowledge of each local subnet. Furthermore, MSBN global sampling schema tend to explore only a small part of the entire multi-agent domain space.

We thus aim to maintain the LJF framework and explore localized approximation, a technique that has been applied in the propagation of hybrid BNs [54], and in large networks to approximate hard to compute messages [46] [20] [44]. In an MSBN, the computational cost for global inference is exponential to the size of the largest node in the hypertree [92]. Therefore, approximation in the larger nodes of an MSBN would improve the global inference efficiency. Local sampling in MSBNs would be straightforward if the subnets were valid BNs. Unfortunately, we have either an original subnet of a DAG structure with no marginal representation guaranteed, or an LJF local JT that is calibrated, but in the form of a JT. In the case of the former, local sampling is not feasible due to the lack of prior marginal information. We can only resort to the local sampling of calibrated local JTs.

Standard BN JTs algorithms have been combined with sampling in order to perform fast and accurate approximate inference algorithms. In one group of algorithms, samples can be obtained with a Markov chain Monte Carlo sampler combined with some exact calculations. Among them, we can have a Gibbs sampler [30] combined with JT exact message passing [44] [38] or use Rao-Blackwellized approximation [25] to improve the sampling generation [68] [7] [6]. A different approach is to use importance sampling based technique to sample a JT [11] [51]. In our case, we find motivation for an LJF local JT-based sampler that operates in a multi-agent context. We prefer an important sampler with an explicit form of importance function so that it can be integrated with the existing inter-agent communication schema, and to support the efficient message calculation over the LJF linkage trees.

Importance sampling algorithms have been well applied in BN approximations [29] [80] [13] [66] [15] [108]. As we study the extension of BN importance sampling techniques to JTs, we present an LJF-based local adaptive importance sampler(LLAIS), which is viewed as the main contribution of this chapter. We design our importance function as tables of posterior probabilities over the clusters of an LJF local JT. We adopt the adaptive importance sampling [15], such that the importance functions are learned sequentially to approach the optimal distribution. One innovative feature of the LLAIS is that it facilitates inter-agent message calculation. We obtain an approximation of messages over linkage trees from the learned importance function. Our experiments results have shown that the LLAIS algorithm converges much faster compared to the other two local JT-based importance samplers we implemented. Also, with the LLAIS, a good approximation of inter-agent messages is available before the local sampling is completed.

6.1 Importance Sampling for BNs

A prominent subclass of BN approximate algorithm is the family of stochastic sampling algorithms, also known as the Monte Carlo algorithms. These algorithms sample the probability distribution and compute the probability required based on the obtained samples by calculating the frequencies of instantiations of the interest. The execution time is mostly independent of the topology of the network and is linear in the number of samples. Furthermore, these algorithms have an any-real-time property such that the computation can be interrupted at anytime with a guaranteed result.

Importance sampling is a class of Monte Carlo algorithms for approximate reasoning in BNs. As a commonly used simulation technique, importance sampling samples a modified distribution, known as the *importance function*, to estimate a hard-to-sample target distribution. The underlying idea of importance sampling is to approximate the average over a set of numbers by an average over a set of *sampled* numbers.

In order to evaluate a sum $I = \sum_{x \in X} g(x)$ for some real function g, samples are generated from an importance function f such that $g(x) \neq 0 \implies f(x) \neq 0$. We have

$$I = \sum_{x \in X} g(x) = \sum_{x \in X} \frac{g(x)}{f(x)} f(x) = E_f \left[\frac{g(x)}{f(x)} \right]$$

by the definition of expected value. We estimate I as

$$\hat{I} = \frac{1}{N} \sum_{i=1}^{N} w(x^i),$$

where

$$w(x^i) = \frac{g(x^i)}{f(x^i)}$$

is called the *sample weight* or the *score*.

In order to compute the probability of evidence $P(\mathbf{E} = \mathbf{e})$ from a JPD $P(\mathbf{X}) = \prod_{i=1}^{n} P(X_i|Pa(X_i))$ of a BN model, we need to sum over all non-evidence nodes:

$$\begin{aligned} P(\mathbf{E} = \mathbf{e}) &= \sum_{\mathbf{X}\backslash\mathbf{E}} P(\mathbf{X}\backslash\mathbf{E}, \mathbf{E}) \\ &= \sum_{\mathbf{X}\backslash\mathbf{E}} \prod_{i=1}^{n} P(X_i|Pa(X_i), \mathbf{E} = \mathbf{e}) \end{aligned} \tag{6.1}$$

Let $\mathbf{Z} = \mathbf{X}\backslash\mathbf{E}$, we simplify Equation 6.1 as

$$P(\mathbf{E} = \mathbf{e}) = \sum_{Z\in\mathbf{Z}} P(Z = z, \mathbf{E} = \mathbf{e}) \tag{6.2}$$

and we can apply the principle of importance sampling.

Suppose we choose a distribution Q as the importance function such that $P(Z = z, \mathbf{E} = \mathbf{e}) \neq 0 \implies Q(Z = z) \neq 0$. Such an importance function is also known as the *sampling distribution* or the *proposal distribution*. Then, Equation 6.2 can be rewritten as

$$P(\mathbf{E} = \mathbf{e}) = \sum_{Z\in\mathbf{Z}} \frac{P(Z = z, \mathbf{E} = \mathbf{e})}{Q(Z = z)} Q(Z = z). \tag{6.3}$$

By the definition of expected value, we have

$$E_Q[\mathbf{Z}] = \sum_{Z\in\mathbf{Z}} z\, Q(Z = z)$$

and from Equation 6.3

$$P(\mathbf{E} = \mathbf{e}) = E_Q\left[\frac{P(Z = z, \mathbf{E} = \mathbf{e})}{Q(Z = z)}\right] = E_Q[w(Z = z)]$$

where $w(Z = z)$ is the score of each sample and

$$w(Z = z) = \frac{P(Z = z, \mathbf{E} = \mathbf{e})}{Q(Z = z)}.$$

Thus, if we sample from Q and obtain a sample set $(z_1, ..., z_n)$, then

$$\hat{P}(\mathbf{E} = \mathbf{e}) = \frac{1}{N} \sum_{i=1}^{N} \frac{P(Z = z_i, \mathbf{E} = \mathbf{e})}{Q(Z = z_i)} = \frac{1}{N} \sum_{i=1}^{N} w(Z = z_i).$$

As the sample size increases, the expect value approaches the true average. That is, as $N \to \infty$, $\hat{P}(\mathbf{E} = \mathbf{e}) = P(\mathbf{E} = \mathbf{e})$. Thus, such an estimator is unbiased.

We can obtain the posterior probability distribution $P(\mathbf{X}|\mathbf{E})$ by separately computing the two terms $P(\mathbf{X}, \mathbf{E})$ and $P(\mathbf{E})$, and then combining them by the definition of conditional probability.

$$\hat{P}(X_i = x_i|\mathbf{E} = \mathbf{e}) = \frac{\hat{P}(X_i = x_i, \mathbf{E} = \mathbf{e})}{\hat{P}(\mathbf{E} = \mathbf{e})} = \frac{\sum_{j=1}^{N} \delta(x_i, z_j) w(z_j)}{\sum_{j=1}^{N} w(z_j)} \tag{6.4}$$

where $\delta(x_i, z_j) = 1$ if and only if the sample z_j contains $X_i = x_i$. Otherwise, $\delta(x_i, z_j) = 0$. It is important to note that while the two terms $\mathbf{P}(\mathbf{E} = \mathbf{e})$ and $P(X_i = x_i|\mathbf{E} = \mathbf{e})$ can be separately estimated unbiasedly, the estimation obtained by combining them through Equation 6.4 is not a unbiased estimator [15].

The quality of an importance sampling estimator mostly depends on how close is the sampling distribution to the target distribution [45]. Essentially, various importance sampling algorithms for BNs only differ in the way they obtain the importance function, which represents the sampling distribution. Many successful importance sampling algorithms [15, 61, 108, 12] have been proposed in recent years. Several choices of the importance function are available ranging from the prior distribution as in the likelihood weighing algorithm [45], to more sophisticated alternatives. The latter includes algorithms that update the importance function through a learning process [15], or calculate the importance function directly with loopy belief propagation [108]. These methods try to gradually approach the optimal importance function, which is usually a function proportional to the posterior distribution, and preferably with a thick tail [55] [107].

6.2 Basic Importance Sampling for LJF local JT

Earlier research has suggested difficulties in applying stochastic sampling to MSBNs at a global level [93]. Direct local sampling in an MSBN subnet is also not feasible due to the absence of a valid BN structure. However, an LJF local JT, the secondary structure of a subnet, can be calibrated with a marginal over all the local variables, making local sampling possible. Algorithms have been proposed to combine sampling with JT belief propagation [44, 68, 45]. Although generally applicable to a calibrated LJF local JT, these algorithms are based on Markov chain Monte Carlo, thus do not support efficient inter-agent message calculation in the context of MSBNs.

We now introduce a JT-based importance sampler, which will be extended in the next sections. Importance sampling in JTs was previously studied [51], such that the importance function was composed of some factors of JT clusters. In our case, however, an explicit form of the importance function is necessary as it facilitates the learning of the optimal sampling distribution, as well as the efficient calculation of inter-agent messages.

The JPD over all the variables in a calibrated local JT can be recovered as a decomposable model similar to the BN DAG factorization. Let $C_1, ..., C_m$ be the m JT clusters given in an ordering that satisfies the running intersection property. The *separator* is $S_i = \emptyset$ for $i = 1$ and $S_i = C_i \cap (C_1 \cup ... \cup C_{i-1})$ for $i = 2, ..., m$. Since $S_i \subset C_i$, we have the *residual* defined as $R_i = C_i \backslash S_i$. The JT running intersection property guarantees that the separator S_i separates the residual R_i from the set $(C_1 \cup ... \cup C_{i-1}) \backslash S_i$ in the JT.

We apply the chain rule to partition residuals given by the separators and have the JPD expressed as

$$P(C_1, ..., C_m) = \prod_{i=1}^{m} P(R_i|S_i). \tag{6.5}$$

Essentially, we select a root from the JT clusters and direct all links(separators) away from the root to form a directed *sampling JT*. This directed tree is analogous to a BN due to their similar forms of recursive factorization.

Given a JPD factorization of an LJF local JT, we define the importance function P' in our basic sampler as

$$P'(\mathbf{X}\backslash\mathbf{E}) = \prod_{i=1}^{m} P(R_i\backslash\mathbf{E}|S_i)|_{\mathbf{E}=\mathbf{e}} \tag{6.6}$$

Here, the vertical bar in $P(R_i\backslash\mathbf{E}|S_i)|_{\mathbf{E}=\mathbf{e}}$ indicate the substitution of $\mathbf{e}$ for $\mathbf{E}$ in $P(R_i\backslash\mathbf{E}|S_i)$. This importance function is factored into a set of local components, each corresponding to a JT cluster. Given the calibrated potential on each JT cluster C_i, we can calculate $P(R_i|S_i)$ for each cluster directly. For the root cluster, that is

$$P(R_i|S_i) = P(R_i) = P(C_i), i = 0. \tag{6.7}$$

We traverse a sampling JT and sample variables of the residue set in each cluster corresponding to the local conditional distribution. This is done similarly to the sampling of BNs, except that we now sample a group of nodes in a cluster instead of an individual node. If we encounter a cluster that contains a node in the evidence set $\mathbf{E}$, we simply assign to the node the value given by the evidence assignment. A complete sample consists of assignments to all non-evidence nodes according to the local JT's prior distribution. The score of each sample s_i is calculated as

$$Score_i = \frac{P(s_i, \mathbf{E})}{P'(s_i)}. \tag{6.8}$$

Consider the example shown in Figure 6.1 in order to generate a sample for the local JT T_0 with cluster bce as the root cluster and the evidence observed at g, we first sample over variables bce according to its local importance function. For the cluster bcg, no sampling is necessary since g is an evidence node and b and c have already been sampled. Next, we need to sample ad and f from the local importance functions for clusters abd and bfg, given the already determined values of b and g.

Unfortunately, using the prior distribution as the sampling distribution, our basic sampler may perform poorly if the posterior distribution of the network bears little resemblance

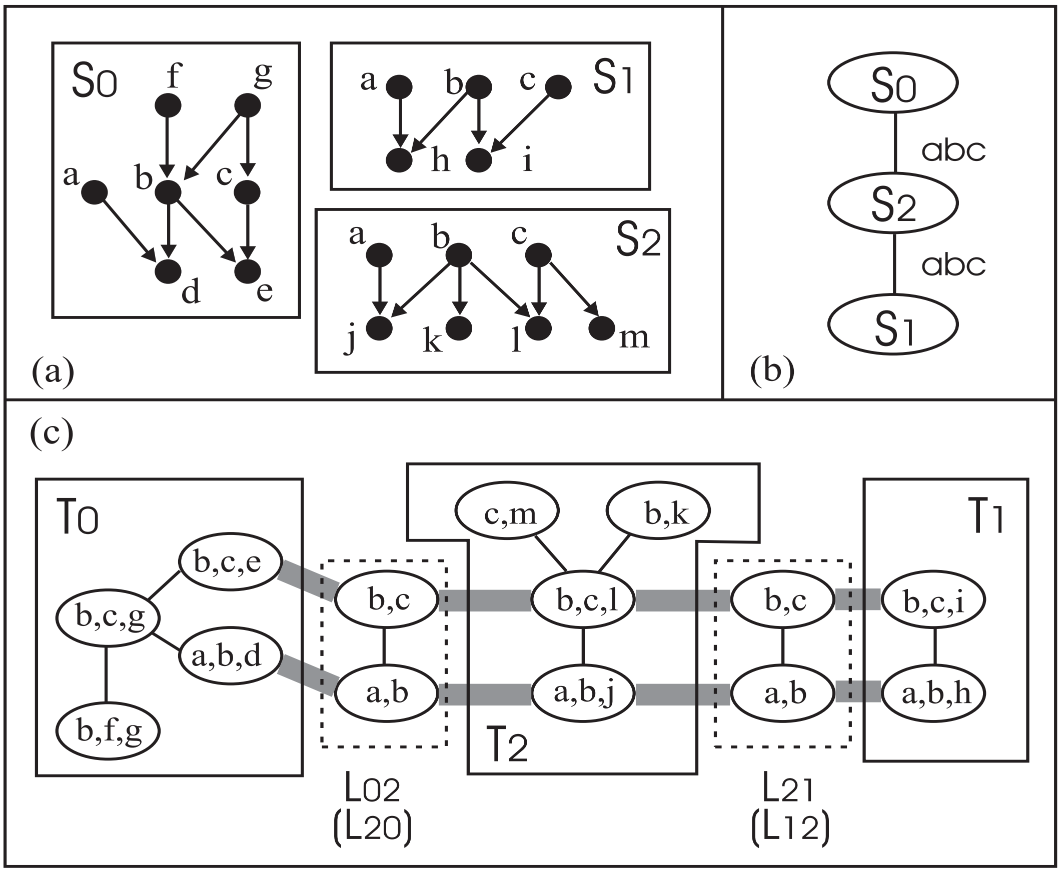

Figure 6.1: An LJF with (a) the subnets, (b) the hypertree and (c) the LJF with linkage trees and local JTs.

to the prior. It is proven that the optimal importance function for BN importance sampling is the posterior distribution $P(\mathbf{X}|\mathbf{E} = \mathbf{e})$ [15]. Applying this result to JTs, we can define the corresponding optimal importance function as

$$\rho(\mathbf{X}\backslash\mathbf{E}) = \prod_{i=1}^{m} P(R_i\backslash\mathbf{E}|\mathbf{E} = \mathbf{e}). \tag{6.9}$$

Equation (6.9) takes into account the influences of all evidence from all clusters in the sample of the current cluster, whereas Equation (6.6) only counts the influence from the precedent cluster, thus causing poor sampling results. Moreover, as there are potentially large differences between the two distributions, we can not exploit the form of the importance function Equation 6.6 in our basic sampler for inter-agent message estimation.

6.3 LJF Local Adaptive Importance Sampler (LLAIS)

Our objectives in designing an LJF local JT importance sampler are: 1) search for a good importance function for the best approximation, and 2) facilitate inter-agent message calculation over LJF linkage trees. There are several methods in BN importance sampling to adaptively approach the optimal importance function. The loopy belief propagation can be conducted to calculate an approximate version of local posterior distribution [108]. The updating process can also be viewed as one of learning a separate BN, by minimizing some error criterion, and use the learned BN for sampling [66]. In this section, we introduce the LLAIS sampler which follows the principle of adaptive importance sampling for learning factors of the importance function. We incorporates a learning process to update the importance function in Equation 6.6 of our basic sampler. We also show that inter-agent messages can be composed directly from the learned importance function.

6.3.1 Updating the Sampling Distribution

Although we know that the posterior distribution is the optimal sampling distribution, it is usually difficult to compute the optimal importance function in Equation 6.9 directly. We can, however, parameterize the sampling distribution to be as close as possible to the posterior distribution. We choose a sub-optimal importance function

$$\rho(\mathbf{X}\backslash\mathbf{E}) = \prod_{i=1}^{m} P(R_i\backslash\mathbf{E}|S_i, \mathbf{E} = \mathbf{e}) \tag{6.10}$$

and represent it as a set of local tables which is learned to approach the optimal sampling distribution. These tables are called the *Clustered Importance Conditional Probability Table*(CICPT).

The CICPT tables, one for each local JT cluster, are tables of probabilities indexed by the separator to the precedent cluster (based on the cluster ordering in the sampling tree) and conditioned by the evidence. For non-root JT clusters, they are in the form of

$$P(R_i|S_i, \mathbf{E}), \tag{6.11}$$

and for the JT root cluster, the CICPT table is

$$P(R_i|S_i, \mathbf{E}) = P(C_i|\mathbf{E}). \tag{6.12}$$

The CICPT tables have a similar structure to the factored importance function in our basic importance sampling algorithm. However, the CICPT tables are updated periodically by the scores of samples generated from the previous tables. A CICPT table is analogous to an ICPT table of BN adaptive importance sampling [15], but applied in the context of LJF local JTs.

A simple learning strategy is to re-calculate the CICPT table based on the most recent batch of samples, so we count the influence of all evidence through the current sample set. But such a learning process could oscillate as we completely ignore the previous CICPT tables at the calculation of new ones. Therefore, we adopt a smooth learning function and our algorithm takes the form:

Algorithm 19 *Algorithm LLAIS*

Step 1. *Specify the total sample number M, total updates K and update interval L. Initialize CICPT tables as in Equation 6.6.*
Step 2. *Generate L samples with scores according to the current CICPT table. Estimate $P'(R_i|S_i, \mathbf{e})$ by normalizing the score for each residue set given the states of the separator set.*
Step 3. *Update the CICPT tables based on the following learning function [15]:*
$P^{k+1}(R_i|S_i, \mathbf{e}) = (1 - \eta(k))P^k(R_i|S_i, \mathbf{e}) + \eta(k)P'(R_i|S_i, \mathbf{e})$*,*
where $\eta(k)$ is the learning function.
Step 4. *Modify the importance function if necessary, with the heuristic of ϵ-cutoff. For the next update, go to step 2.*

Step 5. *Generate the M samples from the learned importance function and calculate scores as in Equation 6.8.*
Step 6. *Output posterior distribution for each node.*

In LLAIS, the importance function is dynamically tuned from the initial prior distribution. New samples are obtained from the current importance function and then used to gradually refine the distribution. The learning overhead is expected to be compatible with that of the BN adaptive importance sampling [15].

It is well known that thick tails are desirable for importance sampling in BNs. This is because the quality of approximation deteriorates in the presence of zero probabilities due to the generation of a large number of samples having zero weights [15] [108] [32]. We solve this issue by a simple heuristic of ϵ-cutoff [14]. If less than a threshold ϵ, the small probabilities will be replaced by ϵ, and the change will be compensated by subtracting the difference from the largest probability entry.

6.3.2 Handling Evidence

In BN JTs, if an observed node is contained in more than one cluster, the evidence is typically inserted randomly into any of the clusters. With our LLAIS sampler, however, we enter the observation into a local JT cluster which contains the evidence node and is also the nearest to the local JT's root cluster. This simple rule is based on the following theorem.

Theorem 4 *In a sampling tree T, $Anc(\mathbf{E})$ is the ancestor cluster(s) to the clusters that contain evidence $\mathbf{E}$. Then, for a cluster $C_i \notin Anc(\mathbf{E}) \Longrightarrow P(R_i|S_i, \mathbf{E}) = P(R_i|S_i)$*

Proof: Suppose for cluster C_i, the values of its corresponding S_i of R_i are set. Then R_i is dependent on evidence $\mathbf{E}$ given S_i only when R_i is d-connecting with $\mathbf{E}$ given S_i. Since T is a directed tree, this happens only when there exists a cluster of C_i's descendants that belongs to the clusters containing evidence $\mathbf{E}$. That is, $C_i \notin Anc(\mathbf{E})$.

Based on Theorem 4, if a cluster is not the ancestor of clusters with evidence entered, its CICPT table remains unchanged. That is, after the CICPT tables are initialized in Step

1 of our algorithm, we simply need to update the tables for clusters that are the ancestors of the evidence.

For example, if all the evidence are observed for nodes in the root cluster of the sample JT, then we already have the CICPT table for each cluster. Our LLAIS algorithm becomes the same as the basic importance sampler with no learning steps and the initial prior distribution as the sampling distribution. In general, by entering new evidence into a cluster nearest to the root, we maximize the number of CICPT tables that require no updates with regard to the evidence node. This will result in considerable savings in the learning process of the importance function.

6.3.3 Calculating Inter-agent Message over Linkage Tree

In an MSBN LJF, agents propagate the impact of their local observations through inter-agent messages passing. Originated from one LJF local JT to one of its adjacent local JTs, an inter-agent message consists of extended linkage potentials over their corresponding linkage tree. With the basic importance sampler, we can only estimate these potentials from the complete sample set. By exploiting the adaptive feature of our LLAIS sampler, however, we are able to obtain an approximation of the extended linkage potentials directly from the learned importance function.

Theorem 5 *Suppose we have a linkage tree L that spans over a set of linkage hosts including the root cluster C_r of a local JT T. For each linkage Q in L, there exists at least one linkage host C_Q with a CICPT table $P'(R_Q|S_Q, \mathbf{E})$, such that the extended linkage potential of Q can be estimated as*

$$\Phi^*(Q) \approx \sum_{N_i \notin Q} P'(R_Q|S_Q, \mathbf{E}) \tag{6.13}$$

Proof: Based on Definition 10 in Chapter 3, a linkage host tree T_h can be formed over all linkage hosts. As the local JT root cluster C_r is included in the T_h, we select the linkage associated with C_r as the root linkage Q_r in L.

First, we consider the root linkage Q_r in L. Based on the definition of extended linkage potential, we have $\Phi^*(Q_r) = \Phi(Q_r)$. Since C_r is the linkage host for Q_r, and Equation 6.12

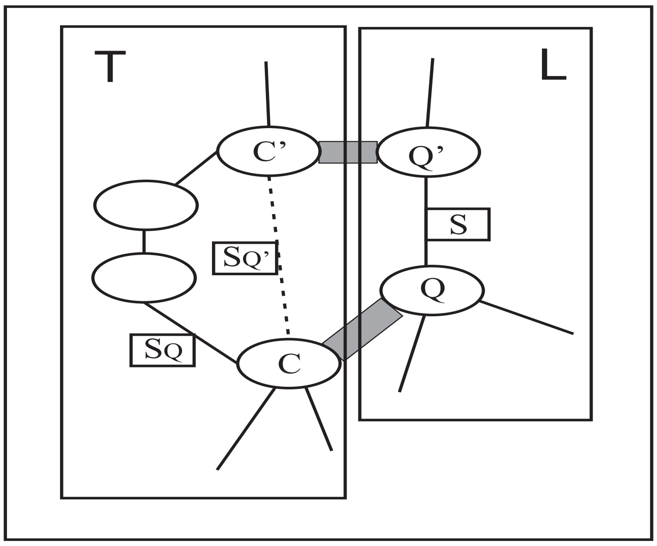

Figure 6.2: Estimation of extended linkage potentials for non-root linkages.

holds, the linkage potential given observed evidence can be obtain as follows:

$$\Phi^*(Q_r) = \Phi(Q_r) = \sum_{C_r \notin Q_r} P(C_r|\mathbf{E}) \approx \sum_{C_i \notin Q_r} P'(C_r|\mathbf{E})$$

Next, consider any non-root linkage Q in L with peer separator S connecting to its parent linkage Q'. The linkage host for Q and Q' are C and C' respectively. Suppose the linkage host tree T_h is an actual subtree of T. Based on the definition of linkage trees, we have C and C' as two neighboring clusters in T, and separator S_Q between C and C' is an actual separator of T. Thus, $S_Q = C' \cap C$, and $S \subset S_Q$.

If the linkage host tree T_h is not a subtree of T as shown in Figure 6.2, then C and C' may not be neighboring clusters T and their intersection S'_Q between C and C' may not be an actual separator of T. However, for each node $x \in S$, $x \in Q$ and also $x \in Q'$ as L is a JT. Moreover, because C and C' are the linkage hosts for Q and Q', we have $x \in C$ and $x \in C'$ as well. Thus, $S \subset S'_Q$ still holds. Based on the running intersection property of a JT, any node in both C and C' will be on the path between C and C'. Therefore, we have $S \subset S'_Q \subset S_Q$, where S_Q is the separator between C and its parent cluster in T.

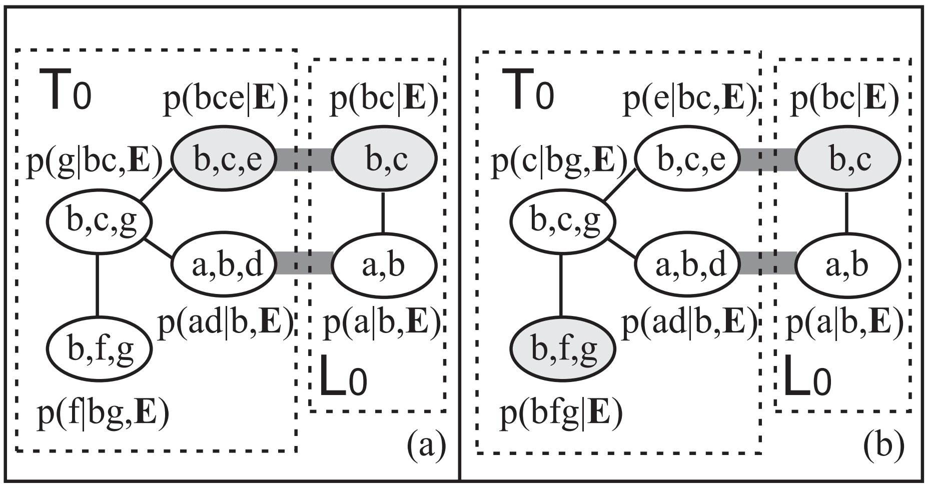

Figure 6.3: Examples of extended linkage potentials calculation.

Thus, for the non-root linkage Q*,*

$$\Phi^*(Q) = \Phi(Q)/\Phi(S) = \sum_{N_i \notin Q} P(C|S_Q, \mathbf{E}) = \sum_{N_i \notin Q} P(R_Q|S_Q, \mathbf{E}) \approx \sum_{N_i \notin Q} P'(R_Q|S_Q, \mathbf{E}).$$

$\square$

Based on Theorem 6.13, we always have a linkage host C_Q for each linkage Q from which an approximation of the extended linkage potential of Q can be obtained by summing out all irrelevant variables from C_Q's CICPT table. Consider the example shown in Figure 6.3(a). T_0 is the local JT and L_0 is the linkage tree. The root cluster of T_0 is $\{b, c, e\}$ (marked as shaded). L_0 spans over two linkage hosts $\{b, c, e\}$ and $\{a, b, d\}$ of T_0. By selecting the linkage $\{b, c\}$ as the root linkage of L_0, we can estimate the extended linkage potential of both linkages $\{b, c\}$ and $\{a, b\}$ by marginalization from each corresponding linkage host's CICPT table.

If a linkage tree is not hosted in the local JT's root cluster, we can still apply the same method of estimation for all linkages except the root linkage. As shown in Figure 6.3(b), the local JT T_0 is now rooted at cluster $\{b, f, g\}$, instead of $\{b, c, e\}$ from Figure 6.3(a). This change of rooting in local JT will affect the calculation of L_0's root linkage $\{b, c\}$. While we can still obtain the extended linkage potential of L_0's all non-root linkages directly with Equation 6.13, the root linkage $\{b, c\}$'s extended linkage potential $P(bc|\mathbf{E})$ cannot be marginalized directly from $P(e|bc, \mathbf{E})$, which is the CICPT table of $\{b, c\}$'s linkage host $\{b, c, e\}$. However, several solutions are available to solve this special case. One option is to obtain the estimation from the most recent update of the linkage host's CICPT table.

The main advantage of our message estimation schema is that we can estimate inter-agent messages before the complete set of samples is available. The approximation error decreases as the importance function approaches the optimal distribution. Essentially, the closer the CICPT table is to the true posterior distribution, the less error there is in our message estimation. Overall, the compromised accuracy can be properly compensated by the increased efficiency of LJF global communication.

6.4 Experimental Results

We conducted our preliminary experiments by comparing the LLAIS algorithm with two other variations of LJF local JT importance samplers, which are the basic importance sampler described in Section 6.2 and the adaptive importance sampler described in Section 6.3. We have not located in literature any previous application of importance sampling to MSBNs LJFs, or JT-based importance sampling with explicit forms of importance function. We implemented the algorithms in Matlab under Kevin Murphy's Bayesian network toolbox [64]. We performed initial tests on a sampling JT constructed from the Alarm network (total 37 nodes).

We evaluated the approximation accuracy in terms of the Mean Square Error(MSE)

$$MSE = \sqrt{\frac{1}{\sum_{X_i \in X \backslash E} N_i} \sum_{x_i \in x \backslash E} \sum_{j=1}^{n} (P'(x_{ij}) - P(x_{ij}))^2}$$

where N is the set of all nodes, E is the set of evidence and N_i is the number of outcomes of node i. $P'(X_{ij})$ and $P(X_{ij})$ are the sampled and exact marginal probability of the state j of a node i. We obtained the gold standard potential using the standard JT propagation.

We generated a total of 30 test cases which include three sequences of 10 test cases each.

	LLAIS	LLS1	LLS2
Minimum	0.0056	0.0098	0.0307
Median	0.0094	0.0297	0.1401
Maximum	0.0270	0.1874	0.2465

Table 6.1: Summary of all 30 test cases for comparing LLAIS, LLS1 and LLS2.

The three sequences had 9, 11 and 13 evidence nodes respectively. Most evidence nodes were in the leaf clusters of the sampling JT. Each algorithm was evaluated with $M = 5000$ samples. With LLAIS, we used the learning function [15] $\eta(k) = a(\frac{b}{a})^{k/k_{max}}$ and set $a = 0.4, b = 0.14$ and the total updates $K = 5$. In each updating step, $L = 2000$. We also separately ran the basic local JT importance sampler without evidence in the same network with 5000 samples for 10 times, resulting in an average MSE of 0.006. This result reflected the optimal accuracy since the results of probabilistic logic sampling without evidence approach the limit of how well stochastic sampling can perform.

Figure 6.4 and Table 6.1 shows the results for all test cases of our first experiment. Each test case was run 10 times and the average MSE was recorded as a function of the probability of the evidence. As far as the magnitude of difference was concerned, the LLAIS performed much better and with significantly better stability than the other two importance samplers, named as LLS1 and LLS2 respectively. In particular, the performance of LLAIS does not degenerate with less likely evidence, which is consistent with the results reported with the BN adaptive importance sampling. The minimum MSE of 0.0056 is within the range of the optimal result. The average MSE of LLAIS is 0.0106 with a medium of 0.0093, which is much smaller than the average MSE of 0.1376 and 0.0551 with the other two samplers. Although the average result for the LLAIS was larger than the optimal accuracy, it was understandable since we updated our importance function for only 5 times, and used a small set of 2000 samples. This short process imposed a small learning overhead, but might not have included enough iterations required for converging to the optimal distribution.

We also performed simulations to evaluate the accuracy of inter-agent message estimation. We randomly selected a subtree from the sampling JT and treated it as a linkage tree. We assumed each linkage host contained the same nodes as the corresponding linkage, and

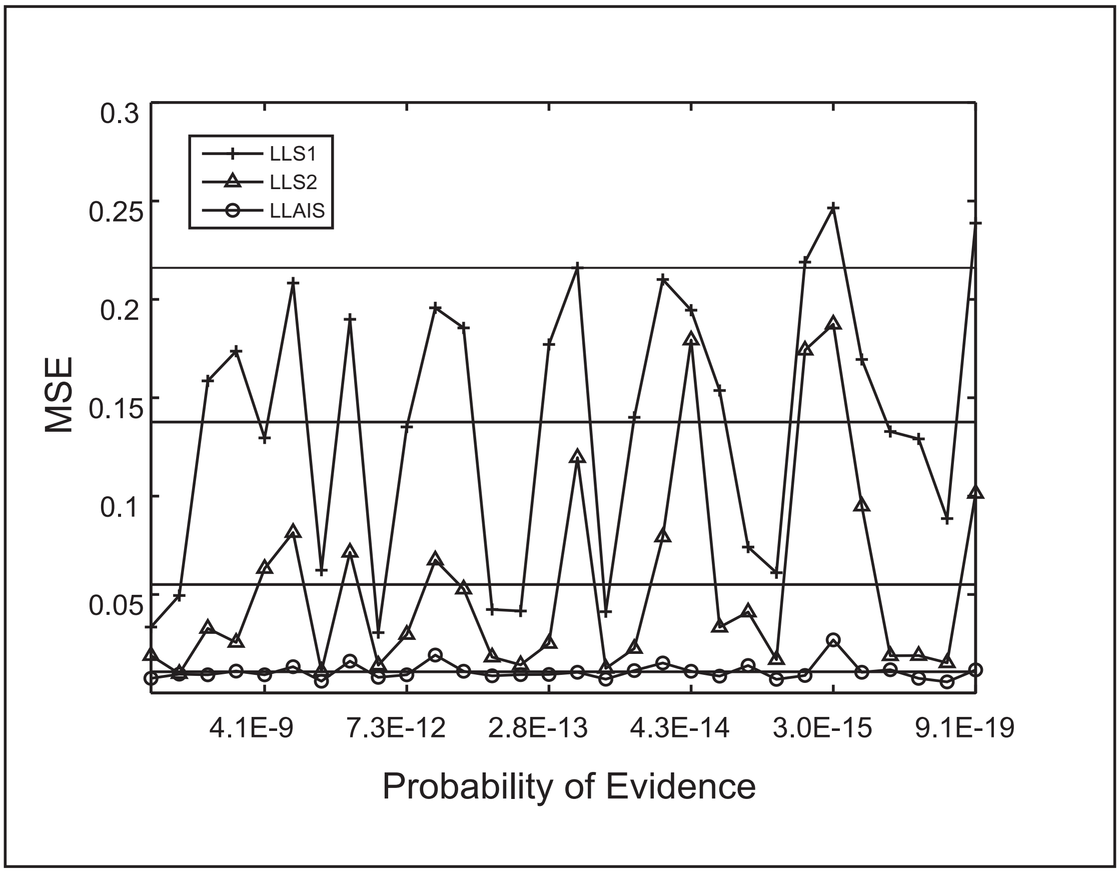

Figure 6.4: Performance of LLAIS, compared with two variations of LJF local importance samplers: MSE for each of 30 test cases plotted against the probability of evidence.

the JT root cluster was included as a linkage host. We used a test case from the previous experiment, which contained 11 evidence nodes. We compared the estimates with the exact results of extended linkage potentials for a total of 4 linkages.

Figure 6.5 shows the convergence of the extended linkage potentials with K=10 and L=2000. At each update, the average MSE of 10 runs was recorded for each linkage. It showed that although minor conciliation occurred at the early stage, all 4 linkage potentials had converged to an average of 0.0314. The error for non-root linkages L_2, L_3 and L_4, however, was larger than we had seen in the first experiment. We believe it was due to the simple method we used to calculate $P(R_i|S_i, E = e)$. We estimated $P(C_i, E)$ and $P(S_i, E)$ separately from the same set of samples, and combine them by conditional probability. This

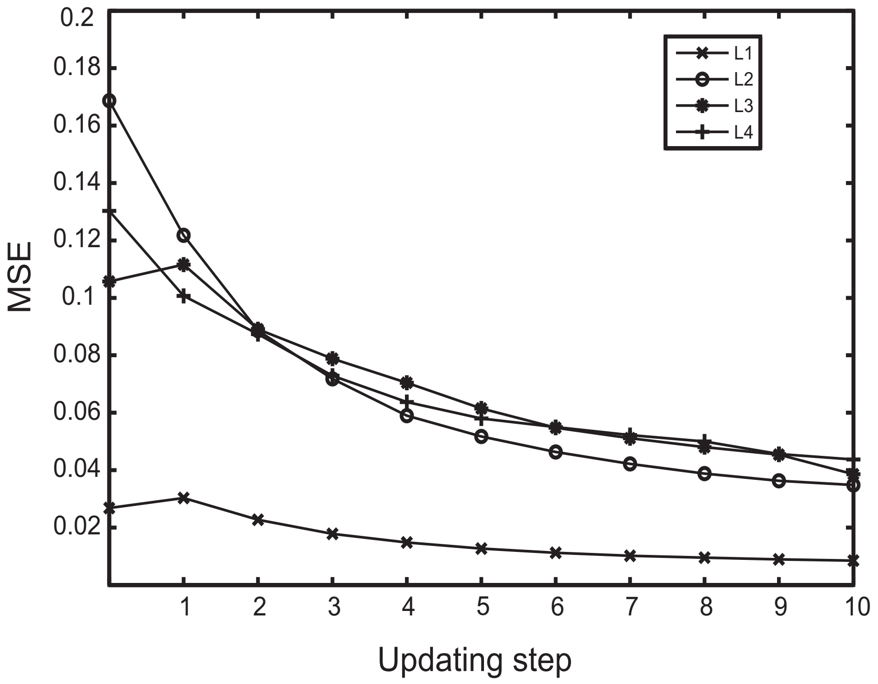

Figure 6.5: Convergence of extended linkage potentials of 4 linkages simulated in a test run of LLAIS.

method may introduce a large variance if the numeric value of $P(R_i|S_i, E = e)$ is extreme. Nevertheless, our message approximation scheme enables the propagation of local beliefs at a much earlier stage of the whole sampling process, which promotes efficient inter-agent communication at the LJF global level.

6.5 Discussion

In this chapter, we have studied the application of importance sampling in MSBN subnets. We have presented the LLAIS sampler, which integrates local importance sampling with the existing LJF framework. The LLAIS sampler adopts the adaptive importance sampling technique for improved sampling accuracy. The dynamic tuning of our importance function also facilitates inter-agent message calculation over LJF linkage trees. In our preliminary experiments, the LLAIS sampler demonstrated promising results for the estimations of both local posterior belief and linkage tree messages. We believe our algorithm represents an important step in solving MSBN communication bottlenecks and realizing practical inference for larger scale multi-agent probabilistic systems.

One direction of our future work will be improving the LLAIS algorithm, which includes methods to estimate posterior distribution with better accuracy, and to improve the learning process for importance functions.

Moreover, an important question that remains unanswered is how local accuracy will affect the overall performance of the entire network. As currently we have only simulated LJF local JTs from BN JTs, further experiments are necessary in full scale MSBNs.

Chapter 7

MA-DBN: Modeling Agents' Dynamic Evolvement

In many applications, cooperative agents need to reason about the states of a complex uncertain domain that evolves over time. An agent often needs to determine the posterior probability for some local nodes of interest, given a set of accumulated evidence from the agent's own observation and those of other agents. A similar problem under the single agent paradigm is often known as the *monitoring* problem [63] [45]. *Online monitoring* requires the calculation of monitoring results at each time step at runtime.

Although the MSBN model has been applied successfully in multi-agent probabilistic reasoning [92], it is restricted to static problem domains, and is insufficient when it comes to modeling dynamic temporal systems. Meanwhile, dynamic Bayesian network (DBN) model is a standard representation for dynamic systems, but it does not provide sufficient support for cooperative reasoning in a distributed multi-agent setting. For example, a coupled HMM [77] [85] can be used to model several parallel chains of evolvement over time, but chain(agent)s must be arranged linearly and they interact only with the neighboring upstream and downstream chains. In fact, reasoning techniques under the DBN model typically have been centralized.

We find motivation to extend the original MSBNs model to represent and reason about the states of a distributed multi-agent dynamic domain. It is natural to search for a combination of the MSBN and DBN models, hoping to take the best of the two worlds. However, the extension of MSBN to the temporal problem domain leads to special challenges. We need to model both the organization and the temporal evolvement of agents. More importantly, as agents' local domains are organized with an underlying structure, variables become highly correlated very quickly. The online monitoring based on the calculation of a forward message over the global forward interface is infeasible since such a forward message must be represented as a joint probabilistic distribution for all temporal variables of all agents.

One solution to approximate monitoring of dynamic system is based on structure factorization. The Boyen-Koller algorithm [9] [10] [64] approximates the exact belief states with a set of independent factors. It has been confirmed that accurate monitoring is possible by decomposing a larger network into weakly interacting subsystems [73][72]. These works are mostly focused on solving efficient monitoring for single-agent DBNs.

Another approximation approach is to utilize distributed particle filtering. Basic BN particle filter [26] has been extended to a distributed multiple platform environment, e.g. sensor networks. Zhao [109] used mostly independent particle filters to track moving objects. Rosencrantz [75] ran particle filters in parallel, sharing measurements as appropriate in a query-response protocol. However, since the sensors (agents) are usually loosely structured, it is difficult to decide the exact information to be shared to reach a global agreement.

Dynamic MSBN(DMSBN) [1] [4] is the most recent extension of MSBNs to dynamic problem domains. A DMSBN models one time slice of a dynamic multi-agent domain as an MSBN, with the assumption that each slice is stationary. Although temporal dependencies are only allowed within an agent's subdomain in a DMSBN, an agent's individual evolution is not explicitly represented. More importantly, DMSBNs do not support efficient monitoring of a group of cooperative agents as a mega MSBN needs to be constructed period by period with a preselected time interval. The online calculation is difficult as the multi-period MSBN usually covers several time steps. Recently, a method has been proposed to solve the time forecast (monitoring) problem with DMSBN by making the interface observability assumption [104]. That is, variables of all agents' d-sepsets are always observed. Such an assumption simplifies the online monitoring calculation by the introduced independencies among agents, but is generally too strong to hold in practical problems.

Our goal is to provide a modeling tool that supports effective online monitoring calculations for a group of cooperative agents. Since exact inference is often impractical in dynamic multi-agent environment, we focus on approximate techniques. In this chapter, we describe a model for dynamic multi-agent probabilistic inference from the original MSBN. Named as the Multi-Agent Dynamic Bayesian Networks(MA-DBN), our new framework aims at modeling the dynamics of a multi-agent domain by individual chains of evolvement at each agent. The temporal advance of each agent is individually and explicitly represented with a set of *local transition graphs*, while agents communicate through the existing MSBN organization structure represented as a *system organization graph*. An MA-DBN can be applied to systems with a mixture of dynamic and non-dynamic agents.

An MA-DBN allows a more efficient and a natural calculation of forward messages through the factorization of an agents' global forward interface. Inspired by the Boyen-Koller algorithm [9] for dynamic Bayesian networks, we also make the assumption of weak interactions among agents' dynamic evolution. In an MA-DBN, each dynamic agent will first calculate and forward the message over its own local forward interface individually. Then, arriving into the new time slice, agents exchange their local information through common nodes similar to the global propagation in static MSBNs, and obtain an approximate cooperative monitoring result. Although the MA-DBN model factorization is based on conditional independency among neighboring agents, we show that the monitoring error is expected to be bounded over time in terms of mutual information shared among two neighboring dynamic agents. Moreover, we propose a method of re-factorization to improve accuracy by enhancing weak correlations among dynamic agents.

Since an MA-DBN models the dynamic of a multi-agent domain with individual chains of evolution, we can adopt existing DBN approximate monitoring techniques in an agent's local domain. As an example, we present an MA-DBN online monitoring algorithm that consists of a set of particle filters, each running individually in a dynamic agent's local domain.

7.1 Dynamic Bayesian Networks

In a single agent setting, the state of a temporal domain can be represented as a dynamic Bayesian network (DBN) [22][63]. A DBN, like a BN, admits a compact representation of the belief state by utilizing conditional independency assumptions among state variables. Although the dynamics of a stochastic system are usually unpredictable, we can assume the system to be Markovian and time-invariant.

Consider a distribution over domain X during the time period $t = 0, ..., T$, which is denoted as $P(X_{(0:T)})$. With the Markov assumption, for all t, we have $I(X_{t+1}, X_{(0:t-1)}|X_t)$. That is, the future is independent of the past given the present. We also assume such a Markovian dynamic system is *stationary* such that the rules that govern the change do not change over time [76]. With these two assumptions, we can model the temporal distribution with the following three components: a prior probability distribution representing the initial state, a *transition model* representing the probability that the system will evolve in a single time step, and an *observation model* describing the observation entered at each time slice.

As defined in [42, 90], a DBN is a quadruplet

$$G^T = (\bigcup_{t=0}^{T} V_t, \bigcup_{t=0}^{T} E_t, \bigcup_{t=0}^{T} E_t^{\rightarrow}, \bigcup_{t=0}^{T} P_t). \tag{7.1}$$

Each V_t is a set of nodes labelled by random variables. V_t represents the state of a dynamic domain at time interval t $(0 \leq t < T)$, where T is the total number of clock times. Each E_t is a set of arcs between nodes in V_t, representing conditional dependencies among domain variables at the given time interval t. Each $E_t^{\rightarrow}$ is a set of temporal arcs, each arc connecting a node in V_{t-1} to a node in V_t $(t = 1, ..., T)$. These arcs represent *temporal dependencies* between two consecutive time slices. The subset of V_t $(0 \leq t < T)$, $FI_t = \{x \in V_t | (x, y) \in E_{t+1}^{\rightarrow}\}$ is called the *forward interface* of V_t. Each P_t is a set of probability distributions such that $P_t = \{P(v|Pa(v)) | v \in V_t\}$. A *slice* of the DBN is the pair $S_t = (D_t, P_t)$, where D_t is a DAG and $D_t = (V_t \bigcup FI_{t-1}, E_t \bigcup E_t^{\rightarrow})$. Collectively, consecutive slices of a DBN define a BN, whose structure is the union of slice structures and whose joint probability distribution is the product of probability tables in all slices.

Based on the assumption of time-invariant, the variables and their links are exactly replicated from slice to slice. Therefore, a DBN can be represented alternatively as a pair $(B_0, B_{\rightarrow})$. Suppose each DBN slice is represented in terms of a set of random variables V_t^i, $i \in \{1, ..., N\}$ B_0 is a standard one slice BN representing the initial state distribution and $B_{\rightarrow}$ is a two-slice temporal Bayesian network (2TBN) which defines the transition model. Overall, a 2TBN represents the conditional distribution

$$P(V_{t+1}|V_t) = P(V_{t+1}|FI_t) = \prod_{i=1}^{N} P(V_{t+1}^i|\pi(V_{t+1}^i)) \tag{7.2}$$

where V_t^i is the i'th node in slice t, and $\pi(V_t^i)$ are the parents of V_t^i in the same or previous time slice. Thus, B_0 and $B_{\rightarrow}$ together define the DBN. The joint probability distribution for time length T can be obtained by unrolling the network until T slices, and then multiplying all CPDs.

7.1.1 BK Approximation

One major task with a DBN is to track the state of a system over time as new evidence becomes available, which is known as *monitoring*. Suppose at each time slice t, evidence e_t is observed for the set of *observable variables*. We need to calculate the belief state $P(X_t|e_1, ..., e_t)$ at time t, where $X_t = V_t \backslash e_t$, representing the set of *non-observable variables*.

The monitoring problem can be solved exactly with the naive approach of unrolling a 2TBN T and applying BN variable elimination algorithms. In order to reduce storage requirements, Kjaerulff [42] proposed to dynamically add new slices and cut off old slices during monitoring. Xiang [90] utilized a temporally invariant template of a fixed length to be reused at runtime, thus saving on the cost of dynamic expansion and reduction. Murphy's interface algorithm [63] creates slice fractions known as the 1.5DBN, which is a slice plus its forward interface to the next slice. The interface algorithm is based on the Markov assumption which implies that all the historical information needed to monitor the system's evolvement is contained in its present state. Thus, we have the DBN forward interface encapsulates all necessary information about previous time slices to perform online monitoring.

While exact monitoring is simple in principle, the computation could be very expensive in practice. The state variables, unless totally decoupled as non-interacting nodes, will all become correlated over time. This results in a belief state that needs to be represented as an explicit joint distribution, rendering exact calculation impractical. Even with the Markov assumption, sometimes the forward interface is too large to be compactly represented.

The Boyen-Koller algorithm [9] utilizes factorization during the process of online monitoring. Given a discrete-time finite-state Markov process, BK approximation factors the exact belief state into a product of clusters based on the assumption of independence between chosen sets of variables [64]. That is, the belief state at time t,

$$\varphi_t = P(X_t|_{y_{1:t}}) \approx \prod_{c=1}^{C} P(X_t^c|_{y_{1:t}}), \tag{7.3}$$

where X_t^c is a subnet of the variables $\{X_t^i\}$. With such approximation, the single potential over all variables is replaced by a product of potentials over some subsets. A *BK cluster* is a set of nodes that forms such a partition. Based on this factorized belief representation, the forward interface of the DBN can thus be represented as factors as well. That is,

$$P(FI_t) \approx \prod_{c=1}^{N} P(FI_t^c), \tag{7.4}$$

where FI_t^c is the set of N clusters partitioning the original forward interface. The single potential over a DBN's forward interface is replaced approximately by a product of potentials. The BK approximation scheme assumes independence among the BK clusters, thus resulting in faster monitoring calculation at each time step.

For example, consider the simple DBN in Fig 7.1. With the BK approximation, the forward interface in time slice 1 is $\{a_1, c_1, d_1\}$ and a set of chosen BK cluster is $\{a_1, d_1\}$ and $\{c_1\}$. The belief state $\Phi(a_1c_1d_1)$ is approximated as $\Phi(a_1d_1) \times \Phi(c_1)$. Thus, the belief table over 3 inference nodes can be more compactly represented by two tables of smaller domains, with the assumption of independency between a_1d_1 and c_1.

The main steps of the BK algorithm are summarized as follows.

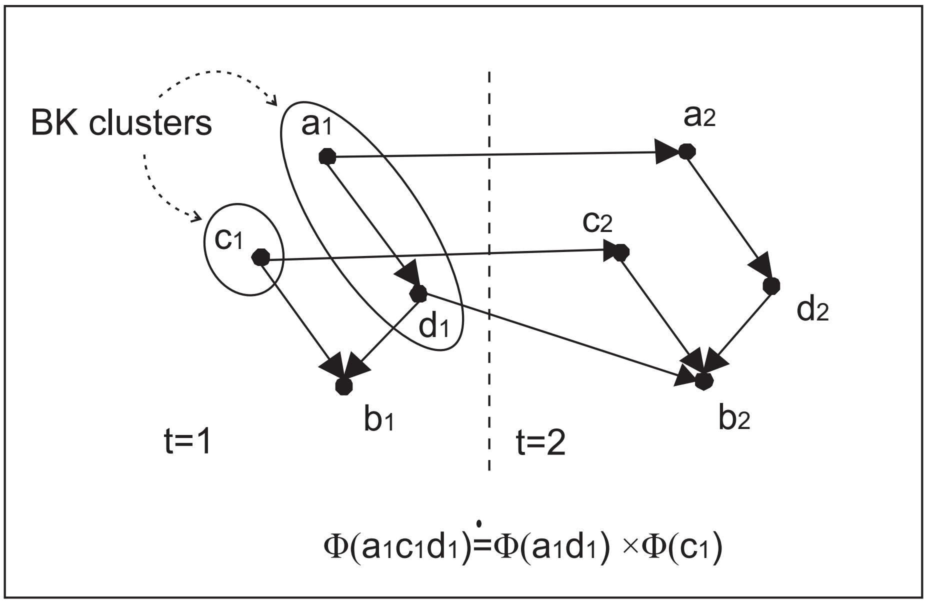

Figure 7.1: BK approximation in a small DBN with a forward interface as $\{a_1, c_1, d_1\}$

Step 1. A factorized representation of the exact belief state is chosen as input to the algorithm. Such representation is an approximate belief state based on the decomposition of the true belief state, resulting in a set of BK clusters.

Step 2. The approximate belief state is propagated forward at time slice t through the transition model for each BK cluster.

Step 3. Given a factored prior in time slice t, the posterior in time slice $t + 1$ is computed with exact Bayesian updating, while taking into consideration the evidence at the new time slice.

The BK algorithm exploits factorized belief states by momentarily ignoring the weak correlations among state variables during the propagation of forward interfaces. Intuitively, although some error is introduced by propagating the factorized forward interface at each time slice, the stochastic nature of the process and the informative nature of the evidence prevent the error from building up.

The errors in a properly factorized belief state contract exponentially as the process evolves with BK approximation. The errors taken over the lifetime of the process are bounded under the two conditions. First, the process is stochastic, so the errors from the past can be forgotten to certain extend. *Mixing rate* is used to bound the rate at which error with approximation from the past are forgotten. It measures the amount of similarity between two distributions [9].

Definition 13 *Let Y be a BK cluster at time $t+1$, and X be set of parent node of Y in time slice t. The mixing rate of the generalized transition $X \mapsto Y$ is defined as*

$$\gamma[X \mapsto Y] \triangleq min_{x_1,x_2} \sum_y [[y|x_1],[y|x_2]]$$

Intuitively, the mixing rate represents the minimal amount of mass that two distributions over a cluster Y are guaranteed to have in common: one is the distribution we would get starting at x_1, and the other starting at x_2 [9]. Secondly, the error introduced at each propagation step is bounded. A set of BK clusters with less degree of dependency will result in better approximation accuracy. Empirical evaluation has shown that huge computation savings is obtained at the cost of a very low error with the BK algorithm [9] [28].

7.2 Online Monitoring for Organized Agents

In a multi-agent probabilistic system, agents often need to deal with uncertainties that evolve over time. One major task is to track a cooperative belief state during runtime. It is similar to the task of monitoring in DBNs, but with considerably increased complexity due to the multi-agent setting.

For a group of cooperative agents $A_0, A_1, ..., A_i$, suppose at time t, evidence e_t^i is observed at agent A_i. The multi-agent online monitoring problem is to calculate the current belief state $p(X_t^i)$ at time t for an agent i's local domain, given partial observation from A_i's own past and the past of all other agents. [1] We use ob_i^t to denote the agent i's observation in its local domain up to time t, such that $ob_i^t = \{e_i^0, e_i^1, ..., e_i^t\}$. Thus, the task is to calculate $p(X_i^t|ob_1^t, ..., ob_n^t)$ at each time slice t where n is the total number of agents.

[1] In the following definition, subscripts are used to index spatial distribution and superscripts are used to index temporal evolution.

For the problem of online monitoring in a multi-agent setting, the most challenging issue is the problem of high correlation. Since temporal dependencies exist in each agent's local domain and agents communicate with each other for their shared information, the domains of all agents will become correlated quickly during the monitoring process. Therefore, the global belief of the system, which passed over consecutive time slices through a global system transition model, can only be represented as an explicit joint distribution over all system variables. This results in prohibitive amount of computation, and renders attempts to track exactly a general multi-agent system impractical.

The dynamic MSBN (DMSBN), first proposed in [92] and formalized in [4], is the most recent extension from the original MSBNs. A DMSBN models one time slice of a dynamic multi-agent domain as an MSBN, with the assumption that each of such clustered MSBN slice is stationary. Agents are organized by a hypertree structure, and maintain temporal dependencies in their local domains. In a DMSBN, a mega forward interface needs to be maintained for all temporal dependencies.

With an DMSBN, the calculation for all system variables over this mega forward interface is mostly impossible. Observing such difficulty, a method was proposed by An [4] to combine a dynamic multi-agent domain over a selected period of time into a global cluster of static MSBNs, and then reason about its state period by period. The calculation is approximated by considering a graphical observable Markov boundary (GOMB), which captures all relevant and observable variables regarding the state of a set of interested variables. The main limitation of this method is that the monitoring of system states is not performed online at each time step. Moreover, a significant amount of centralized control is required as to maintain the super MSBN clusters and to adjust the optimal length of each reasoning period. Recently, a method has been proposed to solve the time forecast (monitoring) problem in DMSBN by making the interface observability assumption [104]. By assuming all interface variables (d-sepsets) are observed, we can have each agent's local evolvement as totally decoupled from others. Such an assumption enables a fast calculation of the online monitoring result among cooperative agents, but it is generally too strong to hold in practical problems.

7.3 Multi-Agent Dynamic Bayesian Network(MA-DBN)

We are motivated to search for a more suitable model to support the online calculation of the multi-agent monitoring task. As we aim to achieve approximate results with bounded accuracy, such model should exploit structured representations based on some assumption of independency, both for agent organization and temporal evolution.

As previously discussed, the key to many DBN inference algorithms, e.g., the interface algorithm, is the Markov assumption. A set of Markovian separators, e.g., the forward interface, separate the future of the process from its past. Unfortunately, the nodes in such a separator could be of a large amount in a complex system in order to block the flow of influence. As a consequence, inference algorithms must maintain a joint distribution over all the separator nodes, rendering the cost exponential in the number of nodes. With the multi-agent online monitoring problem, the cost to maintain the joint distribution over the forward interface of the global domain could be extremely high. Thus, simply using the Markovian assumption is not enough to achieve a practical algorithm.

The work of Boyen and Koller [9] is particularly relevant to our discussion. The BK approximation provides an approximate solution for DBN inference based on the factorization of a complex system into subprocesses that weakly interact. Such a decomposition is utilized to approximate the joint distribution over the forward interface by assuming the subprocesses are independent. They provide bounds on the error incurred by such approximation, and it has been confirmed that accurate monitoring is possible by such decomposition of a larger system into weakly interacting subsystems [72]. For example, BK approximation has been successfully applied in dynamic object-oriented Bayesian network [27], which combines the object-oriented Bayesian network [47] and dynamic Bayesian network.

In static domains, the MSBN framework provides a structure representation based on a set of conditional independent sub-domains. These local sub-domains can be viewed as a form of factorization that naturally occur in the composition of a multi-agent system. They are formed with the MSBN model construction, instead of being a selected set of input in the BK approximation algorithm. Nevertheless, we can apply the same underlying idea of BK approximation to represent a dynamic evolving multi-agent system as a set of weakly interacting subprocesses. We thus take a factorization approach to model a dynamic multi-agent domain.

Definition 14 *MA-DBN*

Given a set of n agents $A_0, A_1, ..., A_i$ each populated with a local domain represented by a BN. An agent A_i is a dynamic agent if temporal dependencies D_i exist between its local domain V_i^t at time t and V_i^{t+1} at time time $t+1$; otherwise, the agent is non-dynamic and its local domain V_i is static. The agents compose a Multi-Agent Dynamic Bayesian Networks(MA-DBN) if the combination of all agents' local domains at each time slice constructs an MSBN and the temporal evolution of each dynamic agent is represented as a DBN.

An MA-DBN has the following properties:

- *Static property of MA-DBN:* In any given time slice, an MSBN hypertree structure exists among all agents. We use *spatial interface*, or simply *interface* to refer to the interface between a pair of adjacent agents in the hypertree.
- *Dynamic property of MA-DBN:* Temporal dependencies exist only within a dynamic agent's local domain. Each temporal dependency is represented by a temporal arc from the agent's local domains V_i^t to V_i^{t+1} in the agent's *local DBN*. The forward interface of the agent's local DBN is thus called the agent's *local forward interface*.

Based on the definition 14, we represent an MA-DBN with a *system organization graph* and a set of agent *local transition graphs*. The system organization graph, analogous to an MSBN hypertree, describes the static property of the MA-DBN. That is, how agents are organized and the interface between pair of adjacent agents. The agent local transition graph describes the dynamic property of the MA-DBN, which is each agent's local structure and how it evolves over two consecutive time slices. A two-slice DBN is used for the representation based on the assumptions of Markovian and time invariant.

Figure 7.2 shows a simple MA-DBN for three agents A_0, A_1 and A_2, with the organization graph in Figure 7.2(a), and the agent transition graphs in Figure 7.2(b). Figure 7.2(a) also shows the hypertree structure, and the interface variables, $\{a, b\}$ and $\{e, g\}$, are marked in curly brackets on the corresponding hyperlinks. Each local transition graph in Figure 7.2(b) describes an agent's local evolution. The local forward interface for each agent is $\{a, c\}$, $\{a\}$ and $\{h\}$ respectively. All observable variables are marked with an underscore.

By definition, an MA-DBN admits an MSBN at each time slice. Thus, a static BN can be composed by combining each agent's MSBN subnet in the same time slice. Moreover, each dynamic agent in an MA-DBN maintains a local DBN. Hence, temporal dependencies exist between two consecutive time slices in the composed BN, resulting a DBN. Such DBN describes the agents' cooperative belief based on the underlying MSBN, and also describes the dynamic evolution of the whole system since it maintains all the temporal dependencies. For clarity, we use the name *global DBN* to refer to this DBN constructed from all agents' local domains and the name *global forward interface* to refer to the forward interface of the global DBN.

Although both MA-DBNs and DMSBNs are extensions of MSBNs, an MA-DBN differs from a DMSBN as an MA-DBN is a more general model which can be used for problem domains with a mixture of static and dynamic agents. Also, it is important to note that

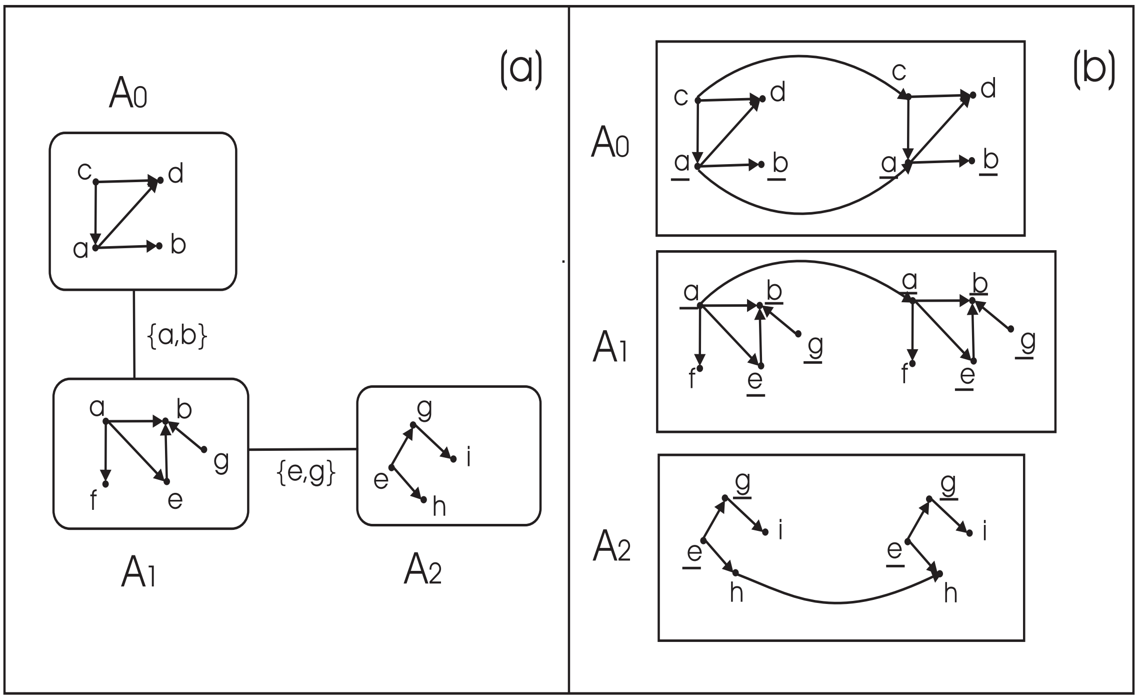

Figure 7.2: A sample MA-DBN.(a) The organization graph, and (b) the local transition graphs.

the global DBN and its forward interface are only used as a conceptual term, and they are never explicitly represented in an MA-DBN.

7.4 Approximate Online Monitoring

As a global DBN represents the dynamic evolution of the entire multi-agent temporal domain, the task of online monitoring translates into the calculation of the message over the global forward interface at each time advance. With an MA-DBN, we approximate this calculation through factorization. That is, we approximate the belief state of the global forward interface as a product of the belief state in each dynamic agent's local forward interface. Let V_G^t denote the state of the multi-agent system and V_i^t the state of a dynamic agent i at time t. FI_G and FI_i are the global forward interface and the agent i's local forward interface respectively. We have

$$P(V_G^t|FI_G^{t-1}) \approx \prod_{i=1}^{n} P(V_i^t|FI_i^{t-1}), \tag{7.5}$$

where n is the number of dynamic agents.

This way, a message passed over the global forward interface is decomposed into a set of messages each over a dynamic agent's local forward interface. Based on Equation 7.5, online monitoring in an MA-DBN can be approximated by a number of individual chains of monitoring in each dynamic agent's local DBN. At each time step after the advance of dynamic agent is finished, global inference is then conducted over the spatial interface of all agents, similar to the global updates in MSBNs.

Obviously, the MSBN LJF structure is no more suited for the calculation of online monitoring with an MA-DBN, because an LJF local JT does not support the advance of each time step in an dynamic agent's local domain. Our solution is to construct a special secondary structure of MA-DBN, named as Linked Dynamic Junction Forest (LDJF). An LDJF consists of two LJFs: an $LDJF_1$ representing the time slice $t = 1$, and an $LDJF_T$ representing the time slice $t > 1$. A special JT, similar to the 1.5DBN [63], is constructed in each dynamic agent's local domain in the $LDJF_T$, and used to propagate the message over the corresponding local forward interface.

The construction of an LDJF from an MA-DBN is similar to the construction of a standard LJF, with distributed moralization and triangulation. The major modification is that we need to complete the nodes from the local forward interface during the triangulation stage, so that they are contained in one JT cluster. In $LDJF_1$, the cluster that contains the forward interface is used as both an *in-cluster* and *out-cluster* for propagating the forward message. In $LDJF_T$, the forward interface in time slices 1 and 2 are included in the two clusters that contains the set of interface nodes. The two clusters are also named as the in-cluster and out-cluster respectively.

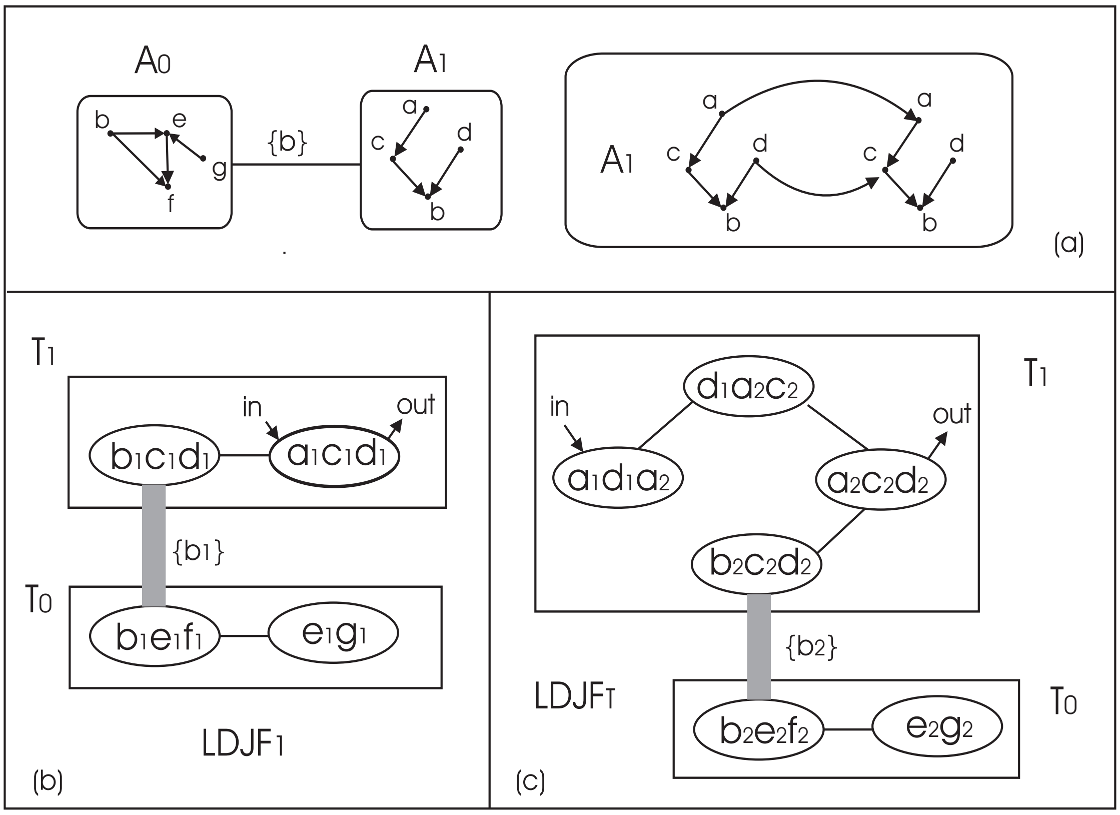

Figure 7.3: An MA-DBN compiled into an LDJF. (a) the MA-DBN (b) $LDJF_1$ (c) $LDJF_T$

For example, consider the MA-DBN with two agents in Fig 7.3. The structure of the MA-DBN is shown in (a) such that A_0 is a static agent and A_1 is a dynamic one. The LDJF is shown in (b) and (c), representing the two LJFs: $LDJF_1$ and $LDJF_T$. While the local JT remains the same for A_0 in both LJFs, the dynamic agent A_1 maintains two different local JTs which together define a local 1.5DBN with the local forward interface $\{a, d\}$. The spacial interface between A_0 and A_1 is $\{b\}$.

Given a constructed LDJF, our online monitoring algorithm is described as follows:

Algorithm 20 *Online_Monitoring_Basic*
An LDJF is populated by cooperative agents including n dynamic agents.

```
1. At time $t = 0$, load $LDJF_1$;
2. For each dynamic agent $A_i(i = 0, 1,..., n - 1)$
3.      update local evidence and update local belief;
4.      obtain the forward message from the out-cluster of local JT;
5. End for
6. At time $t >= 1$, load $LDJF_T$;
7. For each dynamic agent $A_i(i = 0, 1,..., n - 1)$;
8.      multiply forward message into the in-cluster of each local JT;
9.      Condition local belief on local evidence;
10.     Respond to a call for global propagation on $LDJF_T$;
11.     Calculate posterior probability for nodes of interest;
12.     Obtain the forward message from the out-cluster;
13.     Set t=t+1; and go to step 6;
14. End for
```

Our proposed online monitoring algorithm avoids the calculation of messages over the global forward interface. This approach utilizes a factorized approximation which can significantly reduce the cost for maintaining the potential over the global forward interface. An MA-DBN also takes the advantage of MSBN organization structure. At each time step, once all dynamic agents have advanced in their local dynamic domains, inter-agent communication is carried out over the underlying hypertree to propagate an agent's own beliefs throughout the network and thus establish a consistent global belief.

7.4.1 On Approximation Quality

An MA-DBN models the dynamic evolution of a multi-agent system by assuming weak interaction among agents' local processes. Hence, the cooperative evolution of all agents is carried out as a parallel set of local evolutions. This assumption of independency exploits the intuition that a complex system can be decomposed into weakly interacting subsystems. However, during each propagation step, errors are introduced by approximating the single message of global forward interface with a set of message each over the local forward interface. We need to access whether the errors accumulate unboundedly during the whole course of monitoring.

The measure of *KL distance* [17] will be used as a measure to quantify the error between two distributions.

Definition 15 *Given two probability distribution μ and $\tilde{\mu}$ over the same space Ω, the KL distance, or relative entropy of μ to $\tilde{\mu}$ is*

$$\mathbf{D}[\mu \parallel \tilde{\mu}] \triangleq \mathbf{E}_{\mu}\left[ln\frac{\mu(s)}{\tilde{\mu}(s)}\right] = \sum_{s\in\Omega}\mu(s)\cdot ln\frac{\mu(s)}{\tilde{\mu}(s)}.$$

We first outline the error bounds for BK approximation as our error analysis for MA-DBN extends from Boyen and Koller's work [9] [10].

Suppose at time t, $\tilde{\mu}(t)$ is the approximate belief state of the true state $\mu(t)$. The BK approximation results in a *projection error* ϵ_{μ} with respect to $\mu(t)$ at each time step.

Definition 16 *The projection error of approximating φ by ψ with respect to a true distribution $\mu(t)$ is defined as*

$$\epsilon_{\mu}(\varphi \mapsto \psi) \triangleq \mathbf{D}[\varphi \parallel \psi] = \mathbf{E}_{\mu}\left[ln\frac{\varphi(s)}{\psi(s)}\right]$$

It has been shown that the error resulting from BK approximation is bounded in terms of projection error and the mixing rate of each subprocess. For L independent subprocesses T_1, ..., T_L, assume each process depends on at most r others, and each influences at most q

others. Let γ be the minimum mixing rate for T_l, ..., T_L, then

$$\gamma^* = (\frac{\gamma}{r})^q.$$

Theorem 6 *Boyer-Koller Bounded Error Theorem [9]*

Let T be stochastic process with mixing rate γ^*. Suppose we have an approximation scheme that, at each time slice t, incurs a projection error which is bounded by ϵ^* for all time t, Then, on expectation over the sequence of observations, by the process T for all t,

$$\mathbf{E}_T\mathbf{D}[\mu(t) \parallel \tilde{\mu}(t)] \leq \frac{\epsilon^*}{\gamma^*}.$$

The above theorem guarantees the error of BK approximation in terms of KL distance bounds between the entire true and approximate distributions. In fact, the property of belief being bounded indefinitely over time applies to any approximation schema for belief state representation. The term of mixing rate indicates that higher the stochastic of the subprocesses, the more likely old errors from BK factorization will be reduced. The term of projection error depends on the configuration of BK clusters. Indeed, properly selected BK clusters result in significantly smaller error bounds, and it has been confirmed with empirical evaluations [9].

Extending the BK Theorem to the analysis of MA-DBN model factorization, we focus on the project error resulting from the factoring of an MA-DBN global forward interface. Since agents in an MA-DBN are organized by an MSBN hypertree and temporal dependencies exist only in dynamic agents' local domains, an MA-DBN can also be viewed as a single global DBN in which each dynamic agent's local domain corresponds to a BK cluster. Our MA-DBN model approximation is thus analogous to the application of BK approximation. However, agents' local domains overlap over the d-sepsets, rather than being a disjoint set of variables as in BK clusters. Indeed, the MA-DBN model factorization results in an approximation distribution that is conditionally independent. Therefore, the key to our error analysis is to prove that the projection error through such a conditional independent decomposition is still bounded.

First consider the situation when d-sepset variables of the spacial interface connecting all dynamic agents are always observed. As the d-sepset variables d-separate the adjacent pair of local domains, having these variables observed will result in totally independent local process in each dynamic agent, and thus removing the undesirable domain correlation. In fact, such a configuration of MA-DBN results in an optimal BK factorization with which the bound of projection error holds tight. A recent study has confirmed that in such situation, the monitoring with totally independent chains of evolution produces optimal results [104].

In more general cases, however, the variables over an MA-DBN spatial interface are non-observable or partially observable. We base our formal analysis on the relationship between the projection error and the mutual information between the adjacent subnets. Without lost of generality, here we assume an MA-DBN contains only dynamic agents.

The *conditional mutual information* between two sets of variables X and Y given Z, is defined as [17]:

$$\mathbf{I}[X;Y|Z] \triangleq \mathbf{E}_Z\mathbf{D}[P[X,Y|Z] \parallel P[X|Z] \otimes P[Y|Z]],$$

where $\otimes$ is the outer product.

Given a factorization of the global forward interface with an MA-DBN at each time slice, we use φ to represent the distribution of the global forward interface, and use ψ to represent the factorized distribution. Let G_i and G_j be two adjacent local subnets. Then, the projection error defined in Definition 16 can be obtained as

$$\epsilon(\varphi \mapsto \psi) = \sum_{(i,j)\in R} \mathbf{I}[G_i; G_j|M_{ij}]$$

where M_{ij} denotes the intersection $G_i \bigcap G_j$.

Thus, we have the projection error decomposed as a sum of conditional mutual information, each for a pair of adjacent subnets. The bounds of the error can be derived based on the terms of conditional mutual information.

Suppose the *conditional entropy* of X given Y, denoted as $\mathbf{H}[\mathbf{X}|\mathbf{Y}]$, is defined as [17]

$$\mathbf{H}[X|Y] \triangleq \mathbf{E}_Y \mathbf{E}_{X|Y}\left[ln\frac{1}{P[X|Y]}\right]$$

Extending the Theorem of conditional weak interaction [10], we arrive to the following theorem.

Theorem 7 *Let M be an MA-DBN and G_i and G_j be two adjacent dynamic local subnets. Let $\tilde{\mu}$ be the factorized distribution according to MA-DBN factorization. Let φ be the distribution obtained from $\tilde{\mu}$ by propagation each local forward message through the transition model. Then, with respect to φ,*

$$\mathbf{I}[G_i, G_j|M_{ij}] \leq 3 \cdot ln|dom(G_{i\backslash j} \cup G_{j\backslash i})| \cdot (1 - min[\gamma_{ij}, \gamma_{ji}]) + \mathbf{H}[M_{ij}|M'_{ij}] + \mathbf{H}[M'_{ij}|Mij],$$

where the mixing rate γ_{ij}, reflecting the mutual influence between G_i and G_j, is defined as

$$\gamma_{ij} \triangleq \gamma[(G_i \backslash M_{ij}) \mapsto (G'_i \backslash M'_{ij})]$$

Based on Theorem 7, the projection error is bounded at each time step. Intuitively, either two adjacent dynamic agents interact directly, or only weakly, their d-sepset usually evolves more slowly than the two local processes it separates. In real world, this translates to many example of processes whose primary interaction is through some more slowly evolving process [10]. In fact, in a general MA-DBN with static agents, the interaction between dynamic agent is usually weaker, resulting much lower error than the presented bound.
Applying this result to the BK Theorem, we thus establish error bounds for the MA-DBN global factorization during the whole course of online monitoring calculation. That is, even our approximation of global forward interface introduces error at each time step, such error does not accumulate over time.

7.4.2 Method of Re-factorization

For the best approximation quality in BK algorithm, one needs to choose an optimal BK decomposition among all possible ones at the initial step of the BK algorithm. That is, within a single time slice, no node in any BK cluster is the parent of a node in a different BK cluster. This way, an approximation structure is chosen in advance, in which each BK cluster contains most likely the highly correlated temporal dependencies. In the case of MA-DBN, the factorization is decided on the network static structure. Nevertheless, we propose a method of *re-factorization* for improving approximation accuracy in an MA-DBN. Our method refactorizes temporal dependencies in order to promote weak interaction among all agents. Our situation is different from the one seen with the BK algorithm, since each agent's local composition is static such that no nodes can be added or removed from its local domain. Still, the static property of an MA-DBN allows us to maintain highly correlated temporal dependencies within one agent's local domain.

An MA-DBN admits a static MSBN in each time slice, so the following MSBN property holds: for any node contained in more than one subnet with its parents, there must exist a subnet containing the node and its parents. This ensures the minimum interaction between two neighboring MSBN subnets. We utilize this property to reduce the influence between two neighboring chains of temporal evolution.

Definition 17 *Interface temporal dependency DI_x is a temporal dependency represented by a temporal arc connecting a (spatial) interface node x in time slice t to another interface node in time slice $t+1$.*

Definition 18 *Given an interface temporal dependency DI_x, the set of temporal dependency for node x and the parents of x from time slice t to time slice $t+1$ are called close temporal dependencies.*

Given the above definitions, the method of re-factorization is described as follows:

Algorithm 21 *MADBN_Refactorization*
Given an MA-DBN, for each pair of neighboring agents,
1. Duplicate the set of interface temporal dependencies in the two neighboring local domains.

2. Choose one local domain that contains more close temporal dependencies.

3. Remove all redundant interface temporal dependencies in other local domains.

For example, the MA-DBN shown in Figure 7.2 has one set of close dependency over the interfaces, $\{c_t \rightarrow d_{t+1}, a_t \rightarrow a_{t+1}\}$. This MA-DBN can be refactorized as shown in Figure 7.4. Note that temporal dependencies only exist in agents A_0 and A_2 after re-factorization.

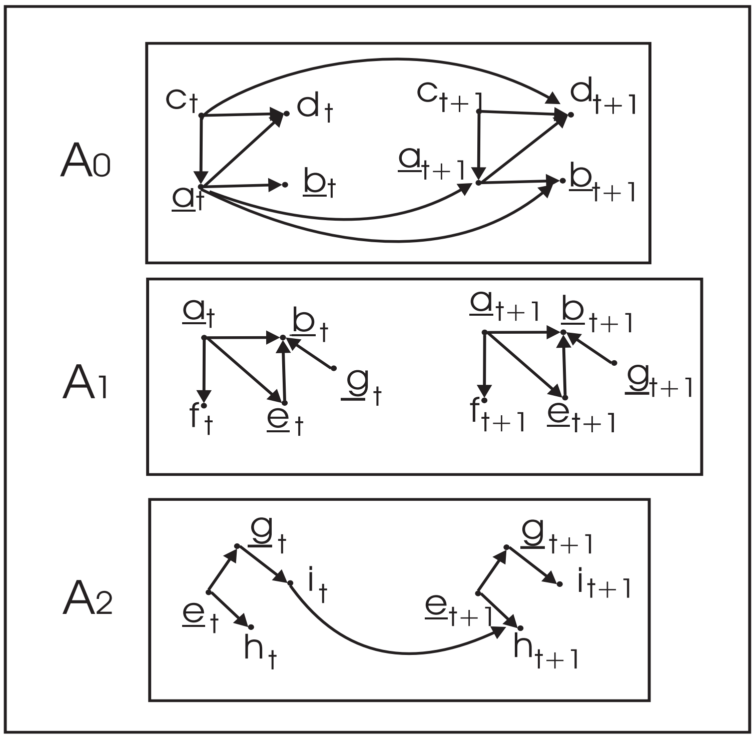

Figure 7.4: MA-DBN re-factorization.

Clearly, the temporal dependencies in the global multi-agent temporal domain are not affected by the procedure of re-factorization. It is due to the fact that the global DBN for the multi-agent domain consists the same temporal dependencies after re-factorization. This operation can be easily implemented through standard message based inter-agent communication. In fact, it requires only the exchange of information over the public spatial interface nodes without revealing an agent's private knowledge.

7.4.3 Distributed Particle Filters

In an MA-DBN, agents gain an increased level of autonomy for their local computation. Instead of performing an exact calculation for its temporal advance as described in Algorithm Basic Online Monitoring, one can adopt various DBN approximation algorithms, combined with the MA-DBN model approximation. In this section, we present an example of this method: a general algorithm that incorporates particle filtering [24] under the MA-DBN framework.

The particle filter algorithm is an important approximation technique based on stochastic sampling; it approximates a belief state by a set of weighed samples. For online monitoring in DBNs, the algorithm starts by generating N particles(samples) according to the prior distribution. Then, at each time slice, the algorithm generates the next state for each particle by sampling from the transition model. Next, it weighs these samples according to the likelihood that they assign to the observation model, and resamples N particles from this weighted distribution. The particles will thus tend to stay clustered in more probable regions of the state space according to the observation at each time slice.

The main advantage of particle filtering is that it provides consistent estimates with a proper size of particles. Therefore, it can be applied to problems of moderately-high dimensions with better performance than traditional numerical methods [24]. We hope to take advantage of particle filtering by reducing local computation and communication costs for an MA-DBN agent, especially if the agent's local domain is non-trivial. We propose to use a set of particle filters, each running in a dynamic agent's local domain. Our basic algorithm is described as follows:

Algorithm 22 *Dist_Particle_Filtering*

An MA-DBN is populated by n cooperative agents. Each agent $A_i(i = 0, 1, ..., n-1)$ maintains a particle filter $PF_i(i = 0, 1, ..., n-1)$. When called, A_i does the following:

1. At time t=0, generate an initial set of particles with PF_i according to the local evidence.

2. Set t=t+1 and extend the particle set to next time slice.

3. Incorporate local evidence and perform global propagation to update local belief.

4. Resample and calculate posterior probability for nodes of interest.

5. Go to step 2.

In our proposed algorithm, three main operations need to be implemented after the generation of an initial set of particles: incorporation of new evidence, forward propagation and inter-agent communication. The first two operations are the same as standard DBN particle filter operations [24]. For the last operation, the inter-agent communication, special consideration is needed in designing an appropriate scheme.

Although the local beliefs are now represented as a set of particles, we can still compose an inter-agent message as an ordinary potential. This enables us to reuse the standard MSBN linkage tree communication schema. Particularly, if we generate samples based on our local JT importance sampler, we can calculate separate messages efficiently over the dynamic agent's linkage trees. Given two adjacent dynamic agents, an alternative solution is to compose a message over the d-sepset as a list of samples together with their weights. Thus, we could forgo the linkage tree structure and transmit messages directly over the agent interface. We need, however, to define additional operations, e.g., the join operation over sets of samples, in order to manipulate directly in the sample space.

7.5 Discussion

The main contribution of this chapter is the extension of the MA-DBN model to approximate online monitoring of a cooperative multi-agent dynamic system. Our work is based on the idea of factorization among weakly interacting subdomains, similar to the successful BK approximation algorithm applied in DBNs. An MA-DBN models the dynamics of a multi-agent domain by individual chains of evolution at each agent, thus resulting in a factorized and more efficient calculation of the global forward message. We have shwon that the approximation errors to be bounded during the monitoring process. A method of re-factorization is proposed to improve the quality of the MA-DBN model approximation. in many situations of practical interest, our proposed MA-DBN model gives rise to approximate online monitoring algorithms that provide effective and accurate estimates for dynamic multi-agent systems.

Although both MA-DBNs and DMSBNs are extensions of MSBNs, an MA-DBN differs from a DMSBN in several aspects. First, an MA-DBN is a more general model such that it can be used to model problem domains with a mixture of static and dynamic agents. A DMSBN can be viewed as a special case of MA-DBN such that all agents must be dynamic. Secondly, an MA-DBN no longer represents an explicit mega forward interface between consecutive slices of the global MSBNs. Finally, MA-DBNs support effective online monitoring through a factorization based approximation that can be naturally distributed, whereas a centralized computation of the global GOMB is typically adopted in the monitoring calculation of a DMSBN.

An MA-DBN provides the flexibility of adopting existing DBN monitoring algorithm for an agent's local evolution. For example, an agent may choose from exact computation or stochastic sampling according to the accuracy requirements and local resources. In particular, we presented an approximate online monitoring algorithm with a set of distributed particle filters under the MA-DBN framework.

In our future work, the quality of MA-DBN model factorization and the method of re-factorization need to be evaluated through experiments. One straightforward approach is to use the exact monitoring result of the global DBN as the baseline, comparing the cooperative monitoring of the MA-DBN with individual chains of exact evolution in local DBNs. We also hope to extend our work to the case of reasoning backward in time, which allows to apply approximate inference to the task of computing the belief state of a time slice given both the past and future evidence.

Moreover, we need to further investigate the interaction corresponding to the level of correlation between agents. In DBNs, a property of separability for a representational structure has been proposed [71]. It has shown that based on the separability between subsystems, some decomposed dynamic system can allow exact calculation of marginal probabilities without propagating the complete joint distribution, but such result does not extend to online monitoring. Once we are able to identify the level of dependency among agents, given an MA-DBN, we can cluster the highly correlated subdomains during the monitoring process. That is, agents in a cluster will propagate forward their belief into the next time slice as a whole. Arriving in the new time slice, all agents cooperate to arrive to an updated global belief through message passing. Although the current monitoring method momentarily overlooks the weak correlations, we use clustering to improve the approximation quality when the interaction between some subsystem too large to be ignored.

Further analysis is also required to investigate the combined errors resulting from incorporating DBN approximation schemes for each agent's local evolution. For example, we need to consider the errors introduced by typical particle filtering. The basic algorithm described so far may result in large variances in the particles' importance weights. Such variances can cause some weights to drop to zero, reducing the effective number of particles in the filter and causing waste of computation [75]. This problem could be solved by periodically restarting each agent's particle filter. Thus, by generating a new set of particles with the evidence in the current time slice, we can discount part of the filter's history and run the filter forward again from the present time. This method is related to the existing methods for improving sample diversity in standard particle filters [87].

Chapter 8

Conclusion

After summarizing the thesis, we will discuss some additional directions for future research and then give some closing remarks.

Thesis Summary

In this thesis, we have presented several improvements over the current inference algorithms of MSBN-based multi-agent probabilistic systems.

We presented an improved message passing architecture for MSBN LJFs. Different from the traditional Hugin-based message passing, our new approach utilized the linkage tree as a message buffer so that an inter-agent message is not absorbed immediately. As a consequence, no repeated local updates are needed. Moreover we always issue partial local updates to calculate an outgoing message or to absorb the last incoming message. Therefore, we are able to forgo any complete local updates, each consisting of two rounds of local message passings. Our new architecture can easily be extended to the iterative messages passing, which maintains the correctness and exactness of the message calculation with improved robustness and fault tolerance.

We then examined the problem of JPD factorization in traditional single-agent BNs. Through the discovery of the semantic meanings of messages passed over JT separators, we presented a procedure to determine the actual information a cluster requires to form the marginals. We have designed a method to form the clique marginal with a minimum set of messages. A simple heuristic was also introduced to further optimize required message passings.

We went on to solve the problem of LJF marginal calibration. Current MSBN calibration methods are performed implicitly and expensively in both inter-agent messages passing and local computation. This makes them unsuitable when an explicit prior marginal is needed for certain nodes. We have presented a new marginal calibration algorithm based upon informed message passing; not only does it provide a correct prior explicitly and exactly, but also it requires then minimum amount of inter-agent message passing and local calculation given an LJF initialization.

Next, we applied approximate inference techniques within an MSBN framework. We have shown that localized approximation can be combined with the existing LJF structure, thus providing a practical solution to the inference in larger MSBNs. We designed a novel LJF local importance sampler with an adaptive feature. Our sampler delivers good approximation on the posterior distribution of an MSBN subnet's local JT. Moreover, the inter-agent messages, originated from the subnet and to be propagated globally in the LJF, can be obtained directly from the learned importance function, before the sampling process is fully completed.

Finally, we addressed the problem of cooperative online monitoring. We extended a model, MA-DBN, from the original MSBN to support distributed multi-agent probabilistic inference in dynamic domains. We modeled the dynamics of a group of cooperative agents approximately by assuming not only the Markov assumption, but also the weak interaction among all agents' local evolvements. The online monitoring is calculated based on the existing conditional independency of agents' local domains, which resulted in a factorized representation of the global forward interface. Our MA-DBN model supports online monitoring calculation with an approximate result and the error is bounded over time. Meanwhile, we have also introduced a re-factorization technique to further improve the approximation quality.

Direction for Future Research

Many of the preceding chapters have concluded with a discussion of directions for future work on their specific topics. Here we discuss some additional directions that are more general in scope.

- As we have been focused on the inference calculation based on an LJF, there have been issues concerning the effective usage of the current single form of an LJF. In the case of our extended Shenoy-Shafer architecture, a modified LJF with a hypertree structure resembling a binary join tree would be more suitable and more efficient for the computation of inter-agent messages with our new schema. Another example is the calculation of the online monitoring problem in MA-DBNs. We have shown that the local JT must be built to support the passing of the forward message in a local domain, resulting in a special LJF which is indeed an LDJF consisting of two LJFs. Therefore, we will investigate the distribution compilation of some special forms of LJFs to serve the purpose of various inference tasks.

- The current agent organization based on a hypertree guarantees the exact probabilistic inference calculation in static domains. However, it is sometimes difficult for a large system to admit a tree-like representation without modifying its specific structure [102]. As approximations are crucial for complex systems in either static or dynamic domains, it seems natural to relax the hypertree restriction in MSBNs. This opens up a vast number of research possibilities for the more flexible modeling of a multi-agent system and effective inference algorithms.

Concluding Remarks

Observing the limitation of the current exact calculation, we have introduced several BN-based techniques to the current MSBN modeling and its exact inference algorithms. We have shown that a Shenoy-Shafer message passing scheme is indeed a more suitable inference architecture for MSBN LJFs. By exploiting the semantic meaning of inter-agent messages and informed scheduling of global message passings, we are able to improve significantly the exact calculation for the marginal calibration problem.

We have applied some approximation techniques while maintaining the MSBN underlying agent organization. To the best of our knowledge, our local importance sampler is the first to realize practical approximation inference with an MSBN-based representation. Our proposed dynamic model, MA-DBN, also gives rise to a new effective method of on-line monitoring of cooperative agents, which is based on approximations from assumed structural independencies.

In spite of the concerns regarding the restriction of the MSBN model construction, we believe that a proper combination of exact and approximate inference, either at an agent's local level or a global agents societal level, will result in a much more effective and practical inference calculation in larger and more complex domains.

Bibliography

[1] X. An. *Towards dynamic multiagent probabilistic inference: testbeds and methods*. PhD thesis, University of Waterloo, Ontario, Canada, 2005.

[2] X. An and N. Cercone. Iterative multiagent probabilistic inference. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 240–246, 2006.

[3] X. An, D. Jutla, and N. Cercone. Privacy preserving multiagent probabilistic reasoning about ambiguous contexts: A case study. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 801–807, 2006.

[4] X. An, Y. Xian, and N. Cercone. Dynamic multiagent probabilistic inference. *International Journal Approximate Reasoning*, 48(1):185–213, 2008.

[5] P G Gulak B J Frey, F R Kschischang. Concurrent turbodecoding. In *IEEE Int. Symp. on Inform. Theory*, 1997.

[6] B. Bidyuk and R. Dechter. Cycle-cutset sampling for Bayesian networks. In *In The Canadian AI Conference, CAAI03*, pages 297–312, 2003.

[7] B. Bidyuk and R. Dechter. Empirical study of w-cutset sampling for Bayesian networks. In *Uncertainty in Artificial Intelligence*, pages 37–46, 2003.

[8] J. Blair, P. Heggernes, and J. Telle. A practical algorithm for making filled graphs minimal, 2001.

[9] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Fifteen Conference on Uncertainty in Artificial Intelligence*, 1998.

[10] X. Boyen and D. Koller. Exploiting the architecture of dynamic systems. In *In Proc. of the National Conference on Artificial Intelligence (AAAI)*, pages 313–320, 1999.

[11] J. Cano, L. Hernandez, and S. Moral. Importance sampling algorithms for the propagation of probabilities in belief networks. *International Journal of Approximate Reasoning*, 15(3):77–92, 1996.

[12] K. C. Chang and Donghai He. Particle filter with iterative importance sampling for Bayesian networks inference. In *Proceedings of SPIE, the International Society for Optical Engineering.*, volume 5809, pages 313–321, 2005.

[13] J. Charnes and P. Shenoy. A forward monte carlo method for solving influence diagrams using local computation. 2000.

[14] J. Cheng. *Efficient stochastic sampling algorithms for Bayesian networks*. PhD thesis, University of Pittsburgh, 2001.

[15] J. Cheng and M. J. Druzdzel. BN-AIS: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. *Artificial Intelligence Research*, 13:155–188, 2000.

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

[17] T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley & Sons, Toronto, 1991.

[18] P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153, 1993.

[19] E. Dahlhaus. Minimal elimination ordering inside a given chordal graph, 1997.

[20] A.P. Dawid, U. Kjaerulff, and S.L. Lauritzen. Hybrid propagation in junction trees. In *the fifth international Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 965–971, 1994.

[21] A.P. Dawid and S.L. Lauritzen. Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, 2(1):25–36, 1992.

[22] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

[23] Rina Dechter, Kalev Kask, and Robert Mateescu. Iterative join-graph propagation. In *18th Conference on Uncertainty in Artificial Intelligence*, pages 128–136. Morgan Kaufmann, 2002.

[24] A. Doucet. On sequential simulation-based methods for Bayesian filtering. Technical Report CUED/FINFENG/TR. 310, Department of Engineering, Harvard University, Cambridge, MA, 1998.

[25] A. Doucet. Rao-blackwellised particle filtering for dynamic Bayesian networks. In *16th Conference on Uncertainty in Artificial Intelligence*, 2000.

[26] A. Doucet, J. de Freitas, and N. Gordon. *Sequential monte carlo methods in practice*. Springer Verlag, 2001.

[27] N. Friedman, D. Koller, and A. Pfeffer. Structured representation of complex stochastic systems. In *In Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 157–164, 1999.

[28] C. Frogner and A. Pfeffer. Discovering weakly-interacting factors in a complex stochastic process. In *Neural Information Processing Systems*, 2007.

[29] R. Fung and K. C. Chang. Weighting and integrating evidence for stochastic simulation in Bayesian networks. In *Preceedings of the Fifth Conference on Uncertainty in Artificial Intelligence*, pages 209–220, 1990.

[30] S Geman and D Geman. Stochastic relaxation, gibbs distribution, and the Bayesian restoration of images. 6(6):721–741, 1984.

[31] A. Ghosh and S. Sen. Agent-based distributed intrusion alert system. In *the 6th International Workshop on Distributed Computing-IWDC 2004, Kolkata, India*.

[32] V. Gogate. Approximate inference in probabilistic graphical models with determinism. In *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, pages 1927–1928, 2007.

[33] H. Guo and W. Hsu. A survey of algorithms for real-time Bayesian network inference. In *In In the joint AAAI-02/KDD-02/UAI-02 workshop on Real-Time Decision Support and Diagnosis Systems*, 2002.

[34] M. Henrion. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In *Proceedings of the Uncertainty in Artificial Intelligence 2 Annual Conference on Uncertainty in Artificial Intelligence (UAI-86)*, Amsterdam, NL, 1986. Elsevier Science.

[35] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15(3):225–263, October 1996.

[36] A. Ihler, J. Fischer III, and A. Willsky. Loopy belief propagation: Convergence and effects of message errors. *Journal of Machine Learning Research*, 6:905–936, 2005.

[37] F. V. Jensen and Frank Jensen. Optimal junction trees. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, pages 360–366, San Francisco, CA, USA, 1994.

[38] F. V. Jensen, A Kong, and U. Kjaerulff. Blocking gibbs sampling in very large propabilistic expert systems. In *International Journal of Human Computer Studies. Special Issue on Real-World Applications of Uncertainty Reasoning*, pages 647–666, 1995.

[39] F.V. Jensen. *An Introduction to Bayesian Networks*. UCL Press, 1996.

[40] F.V. Jensen, S.L. Lauritzen, and K.G. Olesen. Bayesian updating in causal probabilistic networks by local computation. *Computational Statistics Quarterly*, 4:269–282, 1990.

[41] Jordan. *An Introduction to Variational Methods for Graphical Models*. The MIT Press, 2004.

[42] U. Kjaerulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Eighth Conference on Uncertainty in Artificial Intelligence*, pages 121–129, 1992.

[43] U. Kjaerulff. Reduction of computational complexity in Bayesian networks through removal of weak dependences. In *Tenth Conf. on Uncertainty in Artificial Intelligence*, pages 374–382. Morgan Kaufmann, 1994.

[44] U. Kjaerulff. Hugs: Combining exact inference and gibbs sampling in junction trees. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 368–375, 1995.

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

[46] D. Koller, U. Lerner, and D. Angelov. A general algorithm for approximate inference and its application to hybrid bayes nets. In *Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 324–333. Morgan Kaufmann Publishers, 1999.

[47] D. Koller and A. Pfeffer. Object-oriented Bayesian networks. In *Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 302–313. Morgan Kaufmann Publishers, 1997.

[48] A Kozlov and J. Singh. Computational complexity reduction for bn2o networks using similarity of states. In *Proceedings of the Proceedings of the Twelfth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pages 357–36, San Francisco, CA, 1996. Morgan Kaufmann.

[49] F. Kschischang, B. J. Frey, and H. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47:498–519, 2001.

[50] F. Kschischang and Brendan J. Frey. Iterative decoding of compound codes by probability propagation in graphical models. *IEEE Journal on Selected Areas in Communications*, 16:219–230, 1998.

[51] S. Moral L. D. Hernndez and A. Salmern. A monte carlo algorithm for probabilistic propagation in belief networks based on importance sampling and stratified simulation techniques. *International Journal of Approximate Reasoning*, 18:53–91, 1998.

[52] S.L. Lauritzen and D.J. Spiegelhalter. Local computation with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, 50:157–244, 1988.

[53] V. Lepar and P. P. Shenoy. A comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer architectures for computing marginals of probability distributions. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 328–337, San Francisco, 24-26 1998. Morgan Kaufmann.

[54] Uri N. Lerner. Hybrid Bayesian networks for reasoning about complex systems. Technical report, PhD thesis, Standford University, 2002.

[55] D. MacKay. Introduction to monte carlo methods. In *Proceedings of the NATO Advanced Study Institute on Learning in graphical models*, pages 175–204, 1998.

[56] A. Madsen. An empirical evaluation of possible variations of lazy propagation. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 366–373, 2004.

[57] A. Madsen and F. Jensen. Lazy propagation in junction trees. In Gregory F. Cooper and Serafín Moral, editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 362–369, San Francisco, 1998.

[58] A. Madsen and F. Jensen. Parallelization of inference in Bayesian networks, 1999.

[59] A. Madsen, F. Jensen, U. B. Kjaerulff, and M. Lang. The hugin tool for probabilistic graphical models. *International Journal on Artifical Intelligence Tools*, 14(3):507–543, 2005.

[60] R. J. Mceliece, D. J. C. Mackay, and J. Cheng. Turbo decoding as an instance of pearl's belief propagation algorithm. *Selected Areas in Communications, IEEE Journal on*, 16(2):140–152, 1998.

[61] S. Moral and A. Salmeron. Dynamic importance sampling computation in Bayesian networks. In *Seventh European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-03)*, page 137C148, 2003.

[62] K. Murphy. A variational approximation for Bayesian networks with discrete and continuous latent variables. In *Uncertainty in Artificial Intelligence(UAI)*, pages 457–466, 1999.

[63] K. Murphy. Dynamic Bayesian networks. In *Probabilistic Graphical Models*, 2002.

[64] K. Murphy and Y. Weiss. The factored frontier algorithm for approximate inference in dbns. In *Uncertainty in Artificial Intelligence(UAI)*, 2001.

[65] K. Murphy, Y. Weiss, and M. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Uncertainty in Artificial Intelligence(UAI)*, 1999.

[66] L. Ortiz and L. Kaelbling. Adaptive importance sampling for estimation in structured domains. In *In Proceedings of the 16th Annual Conference on Uncertainty in Artificial Intelligence*, pages 446–454, 2000.

[67] J. D. Park and A. Darwiche. Morphing the hugin and shenoy-shafer architectures. In *ECSQARU*, pages 149–160, 2003.

[68] M. Paskin. Sample propagation. In S. Thrun, L. Saul, and B.Scholkoph, editors, *Advances in Neural Information Processing Systems 16*, Cambridge, MA, 2004. MIT Press.

[69] J. Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29:241–288, 1986.

[70] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, San Francisco, California, 1988.

[71] A. Pfeffer. Sufficiency, separability and temporal probabilistic models. In *UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, pages 421–428, 2001.

[72] A. Pfeffer. Approximate separability for weak interaction in dynamic systems. In *Uncertainty in Artificial Intelligence*, 2006.

[73] A. Pfeffer and T. Tai. Asynchronous dynamic Bayesian networks. In *Uncertainty in Artificial Intelligence*, pages 467–476, 2005.

[74] T. Roosta, M. J. Wainwright, and S. Sastry. Convergence analysis of reweighted sum-product algorithms. In *In International Conference on Acoustic, Speech and Signal Processing*, 2007.

[75] M. Rosencrantz, G. Gordon, and S. Thrun. Decentralized sensor fusion with distributed particle filters. In *In Proc. of Uncertainty in Artificial Intelligence(UAI)*, 2003.

[76] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[77] L. Saul and M. Jordan. Boltzmann chains and hidden markov models. In *Advances in Neural Information Processing Systems 7*, pages 435–442, 1995.

[78] T. Schmidt and P. P. Shenoy. Some improvements to the shenoy-shafer and hugin architectures for computing marginals. *Artificial Intelligence*, 102(2):323–333, 1998.

[79] Y. Sella, A. Beery. Convergence analysis of turbo decoding of product codes. In *Information Theory, IEEE Transactions on*, volume 47.

[80] R. Shachter and M. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Fifth Conference on Uncertainty in Artificial Intelligence*, pages 221–231, 1990.

[81] G. Shafer. *Probabilistic Expert Systems*. Society for Industrial and Applied Mathematics, 1996.

[82] P.P. Shenoy. Binary join trees for computing marginals in the shenoy-shafer architecture,. *International Journal of Approximate Reasoning*, 17(1):1–25, 1997.

[83] P.P. Shenoy and G. Shafer. Propagating belief functions using local computations. 1(3):43–52, 1986.

[84] P.P. Shenoy and G. Shafer. An axiomatic framework for Bayesian and belief-function propagation. In *Fourth Conference on Uncertainty in Artificial Intelligence*, pages 307–314, St. Paul, MN, 1988.

[85] P. Smyth, D. Heckerman, and M. I. Jordan. Probabilistic independence networks for hidden markov probability models. *Neural Computation*, 9:227–270, 1997.

[86] S. Tatikonda and M. Jordan. Loopy belief propagation and gibbs measures. In *In Uncertainty in Artificial Intelligence*, pages 493–500. Morgan Kaufmann, 2002.

[87] R. Van der Merwe, N. deFreitas, A. Doucet, and E. Wan. The unscented particle filter. *Advances in Neural Information Processing System*, 13, 2001.

[88] Y. Xiang. A probabilistic framework for cooperative multi-agent distributed interpretation and optimization of communication. *Artificial Intelligence*, 87:295–342, 1996.

[89] Y. Xiang. Cooperative triangulation in msbns without revealing subnet structures. Technical report, Networks, 1998.

[90] Y. Xiang. Temporally invariant junction tree for inference in dynamic Bayesian network. In *AI '98: Proceedings of the 12th Biennial Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*, pages 363–377, 1998.

[91] Y. Xiang. Belief updating in multiply sectioned Bayesian networks without repeated local propagations. *International Journal of Approximate Reasoning*, 23, 2000.

[92] Y. Xiang. *Probabilistic Reasoning in Multuagent Systems: A Graphical Models Approach*. Cambridge, 2002.

[93] Y. Xiang. Comparison of multiagent inference methods in multiply sectioned Bayesian networks. *Approximate Reasoning*, 33(3):235–254, 2003.

[94] Y. Xiang, X. An, and N. Cercone. Simulation of graphical models for multiagent probabilistic inference. *Simulation*, 79(10):545–567, 2003.

[95] Y. Xiang, J. Chen, and W. S. Havens. Optimal design in collaborative design network. In *4th Inter. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pages 241–248, 2005.

[96] Y. Xiang and H. Geng. Distributed monitoring and diagnosis with multiply sectioned Bayesian networks. In *AAAI Spring symposium on AI in Equipment Service Maintenance and Support*, pages 18–25, 1999.

[97] Y. Xiang and F. V. Jensen. Inference in multiply sectioned Bayesian networks with extended shafer-shenoy and lazy propagation. In *UAI 99*, pages 680–687, 1999.

[98] Y. Xiang, F. V. Jensen, and X. Chen. Inference in multiply sectioned Bayesian networks: Methods and performance comparison. In *IEEE Transaction on Systems, Man, and Cybernetics*, volume 36, pages 546–558, 2006.

[99] Y. Xiang and V. Lesser. Justifying multiply sectioned Bayesian networks. In *6th International Conference on Multi-agent Systems*, pages 249–356, 2000.

[100] Y. Xiang and V. Lesser. A constructive graphical model approach for knowledge-based systems: A vehicle monitoring case study. *Computational Intelligence*, 19(4):284–309, 2003.

[101] Y. Xiang and V. Lesser. On the role of multiply sectioned Bayesian networks to cooperative multiagent systems. *IEEE Trans. Systems, Man, and Cybernetics-Part A*, 33(4):489–501, 2003.

[102] Y. Xiang, K.G. Olesen, and F.V. Jensen. Practical issues in modeling large diagnostic systems with multiply sectioned Bayesian networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 14(1), 2000.

[103] Y. Xiang, B. Pant, A. Eisen, M.P. Beddoes, and D. Poole. Multiply sectioned Bayesian networks for neuromuscular diagnosis. *Artificial Intelligence in Medicine*, 2:293–314, 1993.

[104] J. Smith Y. Xiang and J. Kroes. Multiagent Bayesian forecasting of time series with graphical models. In *Florida Artificial Intelligence Research Society Conference*, pages 565–570, 2009.

[105] M. Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal of Algebraic and Discrete Mathematics*, 2:77–79, 1981.

[106] J. Yedidia, W. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. pages 239–269, 2003.

[107] C. Yuan. *Importance Sampling for Bayesian Networks: Principles, Algorithms, and Performance*. PhD thesis, University of Pittsburgh, 2006.

[108] C. Yuan and M. J. Druzdzel. An importance sampling algorithm based on evidence pre-propagation. In *19th Conference on Uncertainty in Artificial Intelligence*, pages 624–631, 2003.

[109] F. Zhao, J. Liu, J. Liu, L. Guibas, and J. Reich. Collaborative signal and information processing: An information directed approach. In *Proceedings of the IEEE*, volume 91, pages 1199–1209, 2003.

Appendix A

Vita Auctoris

Hongxuan Jin was born in China and graduated from Shanghai University in 1997 with a B.Eng. in Electrical Engineering. She later studied in the University of Windsor, Canada where she obtained an M.Sc. in Computer Science in 2001 and a Ph.D in Computer Science in 2010.