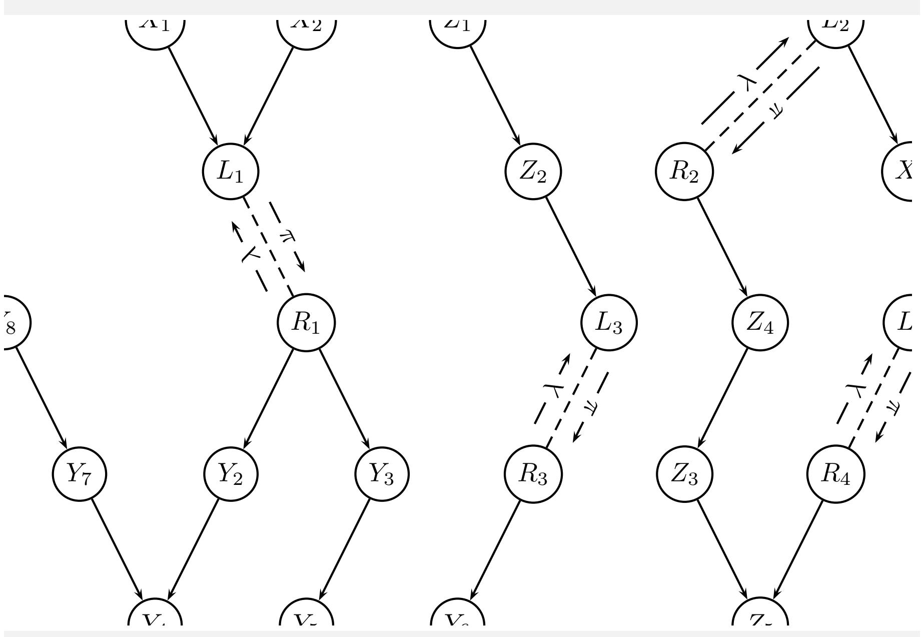
Belief Propagation in Distributed Probabilistic Models

Comparing different approaches to probabilistic inference in distributed Bayesian networks



Patrick de Oude

Supervisors
Dr. Gregor Pavlin
Prof. dr. ir. Frans C.A. Groen

Thesis

Master of Science

in

Artificial Intelligence

Patrick de Oude

Belief Propagation in Distributed Probabilistic Models
Comparing different approaches to probabilistic inference in distributed Bayesian networks

Supervisors:
Dr. Gregor Pavlin
Prof. dr. ir. Frans C.A. Groen

January 29, 2006

University of Amsterdam
Department of Computer Science
Intelligent Autonomous Systems Group
Kruislaan 403
1098 SJ Amsterdam
The Netherlands

Master thesis submitted in partial satisfaction of the requirements for the degree of

Master of Science

in

Artificial Intelligence

______________________________ ______ - ______ - _________

Dr. Gregor Pavlin Date of Approval

Abstract

THE main focus of this thesis is twofold: (i) introduction to Distributed Perception Networks and (ii) a comparison of distributed inference approaches.

We introduce Distributed Perception Networks (DPN) which are a MAS-approach to information fusion with distributed Bayesian networks. DPNs support fusion of very heterogeneous and noisy information. In addition, DPN agents can self organize into DPN systems that support correct and efficient information fusion. Since DPNs can automatically locate and integrate relevant information sources at runtime, they can be used in applications where information sources are not known prior to the operation. DPN systems are organized in such a way that fusion results correctly reflect the entire evidence set that has been injected through different parts (i.e. agents) of a DPN system, without using any synchronization of partial fusion processes. In addition, DPNs do not require compilation of the global inference structure at runtime. Instead, they use network fragments which are compiled prior to the operation. In this way they can adapt to dynamic systems of agents at runtime.

Next to DPNs, we briefly describe two well known approaches to distributed inference in Bayesian networks, namely Multiply Sectioned Bayesian Networks and Prior/Likelihood Decomposable Models. These three approaches are compared by focusing on relevant application aspects: problem domain, model flexibility, (self) configuration, compilation, inference and sequential processing of information. It turns out that each of the compared approaches has advantages and disadvantages with respect to these different application aspects. DPNs support a limited class of models, while they have self organization capabilities. Consequently, they can easily adapt to changing constellations of information sources and dynamic agent systems at runtime. MSBNs and PLDMs, on the other hand, can model very complex domains. However, the modeling flexibility requires computationally expensive preprocessing prior to the operation. Therefore, these approaches are not suitable for domains where constellations of information sources and processing nodes change frequently at runtime.

Acknowledgment

FIRST of all I would like to thank my supervisor Gregor Pavlin for his guidance, patience and assistance in writing this thesis. Without his help this thesis would not be at the current level. Besides that, I owe my gratitude to Jan Nunnink and Sicco Pier van Gosliga for attending the Tuesday afternoon theory discussions with Gregor and myself. I also want to thank Frans Groen and Leo Dorst for providing feedback on the structure and the content of the thesis, Hakan Elgin for solving these nasty DPN implementation problems and Jacob Gerritsen for checking the thesis on spelling and grammar errors. My admiration goes out to my girlfriend for not only being loving and supportive, but also for being understandable about the late night writing sessions and finally, I would like to thank my two cats for keeping me company during the lonesome working hours on this thesis.

Contents

1 Introduction **1**

1.1 Probability Theory and Bayesian Networks 1

1.2 Thesis Focus 2

1.2.1 Distributed Perception Networks 2

1.2.2 Comparison of Distributed Inference Approaches 2

1.3 Thesis Overview 3

1.4 DPN Publications 3

I Bayesian Inference Principles **5**

2 Bayesian Networks **7**

2.1 Bayesian Probability Theory 7

2.1.1 Basics of Probability Theory 7

2.1.2 Conditional Probability Tables (CPTs) 9

2.1.3 Projections 10

2.1.4 Potential Algebra 10

2.1.5 Complexity of Probabilistic Reasoning 11

2.2 Bayesian Networks 12

2.2.1 The chain rule 12

2.2.2 Evidence 13

2.2.3 D-Separation and Conditional Independence 14

3 Bayesian Inference **16**

3.1 Cluster Graphs 16

3.1.1 Cluster Graphs with Cycles 17

3.1.2 Junction Trees 18

3.1.3 Independence Map 19

3.1.4 Potential Assignment 20

3.2 Belief Updating in Junction Trees 20

II Distributed Probabilistic Inference 23

4 Distributed Perception Networks 26

4.1 Causal Models 26

4.1.1 Temporal Aspects 28

4.1.2 Problem Decomposition 29

4.2 Distributed Models 30

4.2.1 Input Concepts and Service Concepts 31

4.2.2 Organization Constraints 32

4.3 Self Organization Principles 33

4.3.1 CNET Protocol 33

4.3.2 Organization Algorithms 33

4.4 Distributed Inference through DPN Network Fragments 36

4.4.1 Static fusion algorithm 36

4.4.2 Dynamic fusion algorithm 36

4.4.3 Appendable fusion algorithm 38

4.4.4 Distributed Inference Process 40

4.4.5 Resetting 42

5 Alternative Approaches to Distributed Probabilistic Inference 45

5.1 Multiply Sectioned Bayesian Networks (MSBNs) 45

5.1.1 Architecture 45

5.1.2 Compilation 48

5.1.3 Inference 49

5.2 Prior/Likelihood Decomposable Models 52

5.2.1 Inference Problem 53

5.2.2 Distributed Sensor Calibration 53

5.2.3 Architecture 54

5.2.4 Model Decomposition 56

5.2.5 Distribution of the Model 56

5.2.6 Robust Message Passing 56

6 Comparison 59

6.1 Problem Domains 59

6.2 Modeling Complexity 60

6.2.1 Distributed Perception Networks 60

6.2.2 Multiply Sectioned Bayesian Networks 61

6.2.3 Prior/Likelihood Decomposable Models 61

6.3 Configuration and Compilation 61

6.3.1 Distributed Perception Networks 61

6.3.2 Multiply Sectioned Bayesian Networks 61

6.3.3 Prior/Likelihood Decomposable Models 61

6.4 Self Configuration and Dynamic Agent Systems 62

6.4.1 Distributed Perception Networks 62

6.4.2 Multiply Sectioned Bayesian Networks 62

6.4.3 Prior/Likelihood Decomposable Models 63
6.5 Exact Propagation 63
6.5.1 Distributed Perception Networks 63
6.5.2 Multiply Sectioned Bayesian Networks 64
6.5.3 Prior/Likelihood Decomposable Models 64
6.6 Frequent Evidence Processing 64
6.6.1 Distributed Perception Networks 65
6.6.2 Multiply Sectioned Bayesian Networks 65
6.6.3 Prior/Likelihood Decomposable Models 66
6.7 Concise Overview 66

7 Conclusions 68
7.1 Distributed Perception Networks 68
7.2 Comparison of Distributed Inference Approaches 68

A Examples 70
A.1 Visit to Asia 70
A.2 Evidence Updating 70
A.3 D-Separation 72
A.4 Absorption 73
A.5 Independence Map 75

B Why BNs Encode Conditional Independence? 77

C Building a Junction Tree 79
C.1 Moralization of a BN 79
C.2 Triangulation 79
C.3 Organizing cliques into a JT 81

D Sensor Data Simulation 84

E List of Notations 86

F List of Abbreviations 88

1 Introduction

REASONING in intelligent systems requires appropriate models that can reflect the uncertain nature of real world problems. Such models should represent the problem domain as concisely as possible without losing relevant information. Including all details about a problem domain in a model turns out to be often undoable, because uncertain events are subject to numerous exceptions. Not only is the discovery of all these exceptions difficult but also covering all of them is inefficient with respect to computation. We simply cannot afford to include all exceptions. Reasoning about uncertainty requires models that rely on a limited knowledge representation of the problem domain.

In general, several different reasoning systems are proposed for handling uncertain domains. These reasoning systems can roughly be classified into two classes: *procedure-based systems* and *model-based systems* (see [18]). Procedure-based systems use uncertainty as a truth value attached to formulas where the uncertainty computation is based on a combination of the truth values of the subformulas. In model-based systems uncertainty is assigned to possible states of an event. In both classes the computational efficiency and semantic perspicuousness differ considerably. Procedure-based systems are computationally efficient, but semantically unclear while model-based systems are computationally inefficient, but semantically clear. In this thesis we will investigate the model-based approach *Bayesian networks (BNs)* used in a distributed environment.

1.1 Probability Theory and Bayesian Networks

It is generally believed that probability theory was started by the French mathematicians Blaise Pascal (1623 - 1662) and Pierre Fermat (1601 - 1665) in the 17th century, while the concepts of chance and uncertainty date back a couple of millenniums B.C. Nowadays, probability theory is an important mathematical tool in a variety of disciplines like for example: psychology, astronomy, biology, economics, medicine, meteorology, marketing, computer science, genetics, etc. Bayesian networks (also called *belief networks*) are theoretically rigorous graphical representations of probabilistic knowledge. BNs can be used to describe probabilistic causal relations between uncertain events. BNs are based on graphs whereas *nodes* correspond to *stochastic variables* that represent events or states and directed links represent causal relations. Every directed arrow in a BN is associated with a probability distribution that reflects conditional uncertainty of relations between events.

Usually, inference in BNs is based on centralized approaches, which might not be suitable for a significant class of real world problems, because:

(i) BNs describing real world domains can be complex, which means that inference can be computationally very expensive;

(ii) Modeled phenomena can be dispersed over a large geographical area. Using a monolithic BN to describe such a problem domain would be awkward, because large amounts of information about the observations must be transported to a central processing unit.

(iii) single point of failures must be avoided and failing of one system may not jeopardize the overall functioning of the system.

In order to be able to deal with these challenges efficiently, we can distribute BNs and inference processes throughout systems of networked devices. However, *Distributed Bayesian networks* are challenging, since there exist trade offs between modeling flexibility and processing efficiency.

1.2 Thesis Focus

The main purpose of this thesis is twofold. Firstly, it introduces a distributed inference approach called Distributed Perception Networks (DPNs). Secondly, DPNs and two other well known approaches to Bayesian inference in a distributed environment are compared in the context of different application aspects.

1.2.1 Distributed Perception Networks

DPNs are a multiagent systems (MAS) approach to distributed information fusion of large quantities of very heterogeneous and noisy information to do state estimation. DPNs support decentralized estimation of the events/states that cannot be observed directly. Every agent in a DPN system captures part of the problem domain and is able to communicate with other agents to perform a global fusion task. DPNs focus on applications where the information sources are unknown prior to operation and huge quantities of information have to be processed sequentially. Consider, for example, a situation where fire fighters have to respond effectively to a spreading forest fire. In order to do so, a large sensor network is deployed to measure smoke, heat, wind direction, etc. Moreover, fire watchers will enter the scene with a helicopter and observe the spreading of the fire. In this way they can provide crucial information about the location of the fire. All this information can be fused in a DPN system to inform the fire fighters about the spreading of the fire. In such applications DPNs are a useful and unique tool to do robust state estimation.

1.2.2 Comparison of Distributed Inference Approaches

Beside DPNs, there are also other approaches to inference in distributed BNs. In general, the applicability of different approaches is domain specific. We analyze applicability of DPNs and Multiply Sectioned Bayesian Networks (MSBNs) and Prior/Likelihood Decomposable Models (PLDMs) with respect to the following application aspects:

- Problem domains - what are the characteristics of the problem domain where the approach is most suitable;

- Model complexity - every approach places different constraints on the possible distributed graphical models. This in turn affects the complexity of the allowed models;

- Configuration - configuration is needed to connect different agents together to perform global task. Configuration methods differ in their ability to use autonomous configuration and support for changing agent systems where agents can join and leave the system of agents;
- Compilation - the graphical model of a BN needs to be converted to another probabilistic structure that supports efficient inference;
- Propagation - information is propagated through the inference structure in order to compute probabilities over modeled events;
- Frequent evidence processing - it turns out that some approaches are not suitable for sequential processing of large amounts of information.

This comparison should give the reader insight into the applicability of each approach in specific problem domains and should also demonstrate the trade-off between complexity and flexibility of different aspects in specific problem domains.

1.3 Thesis Overview

This thesis is organized into two parts: in the first part we review the basic principles of inference with Bayesian networks in order to facilitate description and comparison of different approaches to inference in distributed probabilistic models. In chapter 2 the concise representation of probabilities with the help of Bayesian networks is explained. In chapter 3 Bayesian inference using junction trees is discussed.

In the second part distributed inference approaches will be explained. In chapter 4 DPNs will be introduced. In chapter 5 MSBNs and PLDMs are briefly described.

In chapter 6 the three distributed inference approaches are compared with respect to different application aspects.

Finally, this thesis is concluded in chapter 7 with a review of the most important observations of DPNs and the comparison between different approaches to distributed Bayesian inference.

1.4 DPN Publications

The described work on DPNs in this thesis is partially published in proceedings. DPNs are initially presented in *An Agent-Based Approach to Distributed Data and Information Fusion* by Pavlin et al. [15]. The inference technique of DPNs is published in *Information Fusion with Distributed Probabilistic Networks* by Oude et al. [3]. The comparison between DPNs and MSBNs is published in *Distributed Bayesian Networks in Highly Dynamic Agent Organizations* by Oude et al. [2]. Moreover, a demo paper about the DPN system is published in *A MAS Approach to Fusion of Heterogeneous Information* by Pavlin et al. [16].

Most ideas presented in chapter 4 will be used for a future journal paper about DPNs.

Part I

Bayesian Inference Principles

2 Bayesian Networks

In this chapter we review basic principles of Bayesian networks in order to facilitate description and comparison of different approaches to inference in distributed probabilistic models.

Bayesian Networks (BNs) are graphical representations of probability distributions. They facilitate modeling of causal relations between events with acyclic directed graphs (DAGs). The direction of the arrows indicates which node is the *effect* and which node is the *cause*. A node with an outgoing arrow is called the cause and a node with a incoming arrow is called the effect. Because we are dealing with uncertainty the occurrence of the effect is, in general, not deterministic given that the cause has occurred. In other words, a certain cause will not always result in the effect. There is a chance that the effect will happen given the cause.

BNs can be used to reason about uncertain domains. Say that we have modeled an event A in this problem domain. Event A has a finite number of states and for every state some probability value is assigned. If we know that some state of event A has taken place we can infer the probability of other events that are causally related to event A. This means that the states of these causally related events get updated probabilities (belief updating).

This chapter is organized as follows: in section 2.1 we review basics of modeling with BNs. This discussion is followed by a formal definition of Bayesian networks and the chain rule in section 2.2. Moreover, the notions of d-separation and conditional independence will be clarified.

2.1 Bayesian Probability Theory

We begin the discussion with Bayesian calculus (see [6]) which is also known as classical probability calculus (like in [4]).

2.1.1 Basics of Probability Theory

In this section we review the basics of probability theory which are needed in order to understand the principles of Bayesian networks. Let us first list the three basic axioms of probability calculus for BNs.

Basic Axioms Probability theory is based on the three axioms:

Definition 2.1 (Three Basic Axioms) *The three basic axioms for Bayesian networks are:*

axiom 1: $P(A) \geq 0$;

axiom 2: $P(A = a_i) = 1$ *if and only if state a_i is certain;*

axiom 3: $P\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} P(A_i)$ *only when $\bigcap_{i=1}^{\infty} A_i = \emptyset$*

Assume an example, where we have a *sample space A* with a discrete probability distribution[*] $P(A)$ defined over the events $\{a_1, \ldots, a_n\}$ we can write $P(A)$ as a vector:

$$P(A) = \begin{pmatrix} P(a_1) \\ \vdots \\ P(a_n) \end{pmatrix}$$

If we say that $P(A = a_1) = 1$[†] we know for certain that event a_1 has occurred and we also know that events $a_2, \ldots, a_n$ did not occur given that $\sum_i a_i = 1$ and $\forall a_i : a_i \geq 0$. A probability value between 0 and 1 means that we are uncertain if an event has taken place. For example, if $P(A = a_2) = 0.3$ we know that with a chance of 30% event A will result in a_2 and with a chance $1 - P(A = a_2)$ event A will not result in a_2. This leads us to the two basic axioms, namely

$$0 \leq P(A) \leq 1 \tag{2.1}$$

$$P(A = a_i) = 1 \quad \text{if and only if } a_i \text{ is certain} \tag{2.2}$$

The first axiom states that probability values are only defined in the interval $[0, 1]$ and the second axiom states that when a probability is 1 we know for certain that this event will take place.

The third axiom states that the belief assigned to any set of events is the sum of the belief assigned to the *mutually exclusive* (also disjoint) events. If we have an event A and another event B that are mutually exclusive, meaning that they are non-intersecting, the probability $P(A \cup B)$ is equal to:

$$P(A \cup B) = P(A) + P(B) \qquad \text{where } A \cap B = \emptyset \tag{2.3}$$

The Fundamental Rule The fundamental rule of probability calculus states that we can rewrite a joint probability as

$$P(A, B) = P(A|B)P(B) \tag{2.4}$$

where the probability $P(A, B)$ denotes the *joint probability* of the joint event $A \cap B$. Probability $P(A|B)$ is called the *conditional probability*. It is called conditional because it is the probability of event A that is conditioned on the occurrence of event B. From equation 2.4 we can conclude the following:

$$P(B|A)P(A) = P(A|B)P(B) \tag{2.5}$$

This can be rewritten to

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \tag{2.6}$$

[*] In this thesis only discrete probability distributions are considered
[†] The probability $P(A = a)$ can also be written as $P(a)$ if there cannot be any confusion.

The equation in formula 2.6 is known as the *Bayes' rule* given in theorem 2.1. This rule is the most important rule in Bayesian inference. It allows us to infer the probabilities of other events in a BN. Equation 2.6 states that we can calculate the *posterior* belief of B, reflected in $P(B|A)$, by multiplying the *prior* belief of B, reflected in $P(B)$, with the likelihood of A given that event B occurred. The denominator $P(A)$ is just a normalization constant.

Theorem 2.1 (Bayes' Rule) *Let A and B be two events then Bayes' rule can be defined as:*

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \tag{2.7}$$

Bayes' rule supports *diagnostic reasoning* (or *retrospective reasoning*), where we reason about the probability of the cause given the effects.

2.1.2 Conditional Probability Tables (CPTs)

If we have a variable A with states a_1, a_2 and a variable B with states b_1, b_2, b_3, then we can write $P(A|B)$ as a *conditional probability table (CPT)* that describes a *conditional probability distribution (CPD)*:

Table 2.1: Conditional probability table of $P(A|B)$

	b_1	b_2	b_3
a_1	$P(a_1\|b_1)$	$P(a_1\|b_2)$	$P(a_1\|b_3)$
a_2	$P(a_2\|b_1)$	$P(a_2\|b_2)$	$P(a_2\|b_3)$

For example, the CPT of CPD $P(A|B)$ is given in table 2.2. Notice that the columns sum up to one, which is, a requirement for (conditional) probability distributions.

Table 2.2: Conditional probability table of $P(A|B)$

	b_1	b_2	b_3
a_1	0.3	0.1	0.4
a_2	0.7	0.9	0.6

Given a CPT, we can easily calculate the *joint probability table (JPT)* specifying the *joint probability distribution (JPD)* by using the fundamental rule that was given in section 2.1.1. If we have $P(A|B)$ we can calculate $P(A, B)$

$$P(A, B) = P(A|B)P(B) \tag{2.8}$$

To be able to compute the JPT based on the CPT given in table 2.2 the values of probability distribution $P(B)$ have to be known. Let us assume that the probability distributions of $P(B) = \begin{pmatrix} 0.3 \\ 0.2 \\ 0.5 \end{pmatrix}$. Using the probability distribution of B the JPD can be calculated and will result in the following JPT:

Table 2.3: Joint probability table of $P(A, B)$

	b_1	b_2	b_3
a_1	0.09	0.02	0.2
a_2	0.21	0.18	0.3

All values in a JPT sum up to one. Assume that we also want to compute the probability distribution over A. This distribution can be computed by summing over all states of variable B in the JPT of $P(A, B)$.

$$P(A) = \sum_{j}^{n} P(A, B = b_j) \tag{2.9}$$

This calculation is called *marginalization* over variable B. We can also say that B was marginalized out of the joint probability $P(A, B)$. Marginalization is an important computation because it allows us to calculate *marginal probabilities*, as for example $P(A)$ in equation 2.9.

2.1.3 Projections

In the Bayesian network literature *projections* are used in two different ways:

- Projection corresponding to marginalization (see [6]);
- Projection as potential (see section 2.1.4) entry selection (see [21]).

The meaning of projections corresponding to marginalization is straightforward. For example, if we have the joint probability $P(A, B, C)$ and we want to calculate the marginal probability $P(A, B)$, we can write this as:

$$P(A, B) = P(A, B, C)^{\downarrow\{A,B\}}$$

In this example we can also say that we are *projecting* the probability distribution $P(A, B, C)$ onto the probability distribution $P(A, B)$. So, the following expression is valid:

$$P(A, B, C)^{\downarrow\{A,B\}} = \sum_{C} P(A, B, C)$$

In other words, projecting down to the set of variables $\{A, B\}$ is the same as marginalizing C out of the expression $P(A, B, C)$.

The second type of projection differs from the first in that it is not marginalizing any variables out of the probability distribution, but is used to select entries in the probability distribution or potential. For example, if we have the set of stochastic variables $D = \{A, B, C\}$ with an entry $d = (a_2, b_3, c_1)$ and a set of stochastic variables $E = \{A, C\}$ then the projection $E^{\downarrow d} = (a_2, c_1)$. This example shows that the entry (a_2, c_1) in the set of stochastic variables E is selected by projecting d onto E. In the next section we will show that projections are a very convenient way to describe potential operations like multiplication and division.

2.1.4 Potential Algebra

Potentials are used to specify real-valued unnormalized probability tables. Potentials can specify prior probabilities, joint probabilities, conditional probabilities or any combination[‡] of these. Every stochastic variable that is defined in a potential has an attached domain that is defined over the states of the stochastic variables. For example, if we have the following CPD $P(B|A)$ that is captured by the potential $\phi(A, B) = P(B|A)$ then the stochastic variables of the potential $\phi(A, B)$ are $\{A, B\}$. We assume that stochastic variable A has the states $\{a_1, a_2\}$ and stochastic variable B has the states $\{b_1, b_2, b_3\}$. The corresponding domains are defined by $\mathcal{D}_A = \{a_1, a_2\}$ and by $\mathcal{D}_B = \{b_1, b_2, b_3\}$, respectively. The potential $\phi(A, B)$ uses a combination of the states (also configurations) of A and B and is defined over domain $\mathcal{D}_{\{A,B\}} = \{(a_1, b_1), (a_2, b_1), (a_1, b_2), \ldots\}$.

The multiplication of potentials can be defined through *projections* described in section 2.1.3.

Definition 2.2 (Potential multiplication) *Let $\mathcal{X}, \mathcal{Y}$ be distinct sets of variables and $\mathcal{Z} = \mathcal{X} \cup \mathcal{Y}$. Let $\phi(\mathcal{X}), \phi(\mathcal{Y})$ be two potentials. The multiplication $\phi(\mathcal{X}) \cdot \phi(\mathcal{Y})$ is a potential $\phi(\mathcal{Z})$ such that for each configuration $z \in \mathcal{D}_{\mathcal{Z}}$,*

$$\phi(z) = \phi(\mathcal{X}^{\downarrow z}) \cdot \phi(\mathcal{Y}^{\downarrow z}) \tag{2.10}$$

[‡] Combination of probabilities is defined through multiplication

Let's assume the potential product $\phi(\mathcal{Z}) = \phi(\mathcal{X}) \cdot \phi(\mathcal{Y})$ that results in $\phi(\mathcal{Z})$ where $\mathcal{X} = \{A, B\}$, $\mathcal{Y} = \{B, C\}$ and $\mathcal{Z} = \mathcal{X} \cup \mathcal{Y}$. In table 2.4 the values of the potentials $\phi(\mathcal{X})$ and $\phi(\mathcal{Y})$ are given.

Table 2.4: Potentials of $\phi(\mathcal{X})$ and $\phi(\mathcal{Y})$

(A,B)	$\phi(\mathcal{X})$	(B,C)	$\phi(\mathcal{Y})$
(a_1,b_1)	0.20	(b_1,c_1)	0.11
(a_1,b_2)	0.45	(b_1,c_2)	0.94
(a_2,b_1)	0.38	(b_2,c_1)	1.22
(a_2,b_2)	0.22	(b_2,c_2)	0.76

Following definition 2.2 we can compute $\phi(\mathcal{Z})$ by projecting $\mathcal{X}$ and $\mathcal{Y}$ down to a combination of states $z \in \mathcal{Z}$ (i.e. configuration).

Table 2.5: Potential product $\phi(\mathcal{Z}) = \phi(\mathcal{X}) \cdot \phi(\mathcal{Y})$

(A,B,C)	$\phi(\mathcal{Z})$	(A,B,C)	$\phi(\mathcal{Z})$
(a_1,b_1,c_1)	0.022	(a_1,b_1,c_2)	0.188
(a_1,b_2,c_1)	0.549	(a_2,b_1,c_2)	0.357
(a_2,b_1,c_1)	0.042	(a_1,b_2,c_2)	0.342
(a_2,b_2,c_1)	0.268	(a_2,b_2,c_2)	0.167

The potential multiplication is *commutative* and *associative* which is defined in the theorems 2.2 and 2.3.

Theorem 2.2 (Potential product commutativity) *Let $\phi(\mathcal{X})$ and $\phi(\mathcal{Y})$ be two potentials, then the product between these two potentials is commutative and the following holds:*

$$\phi(\mathcal{X}) \cdot \phi(\mathcal{Y}) = \phi(\mathcal{Y}) \cdot \phi(\mathcal{X}) \tag{2.11}$$

Theorem 2.3 (Potential product associativity) *Let $\phi(\mathcal{X})$, $\phi(\mathcal{Y})$ and $\phi(\mathcal{Z})$ be three potentials, then the products of these three potentials are associative and the following holds:*

$$(\phi(\mathcal{X}) \cdot \phi(\mathcal{Y})) \cdot \phi(\mathcal{Z}) = \phi(\mathcal{Y}) \cdot (\phi(\mathcal{X}) \cdot \phi(\mathcal{Z})) \tag{2.12}$$

The proof of potential product commutativity and associativity is straightforward.

In some Bayesian inference algorithms potentials are also divided by each other. One problem with this operation is dividing by zero (which is undefined). In order to deal with dividing by zero two potentials have to be *zero-consistent*. Zero-consistency means that if for a configuration $z \in \mathcal{D}_\mathcal{Z}$ the potential $\phi(\mathcal{X}^{\downarrow z}) = 0$ also $\phi(\mathcal{Y}^{\downarrow z}) = 0$. When this is the case, potential $\phi(\mathcal{X}^{\downarrow z})$ is zero-consistent with $\phi(\mathcal{Y}^{\downarrow z})$.

Because of zero-consistency the quotient for potentials can be defined as follows:

Definition 2.3 (Potential quotient) *Let $\phi(\mathcal{X})$ be zero-consistent with $\phi(\mathcal{Y})$ and $\mathcal{Z} = \mathcal{X} \cup \mathcal{Y}$. The quotient of $\phi(\mathcal{X})$ divided by $\phi(\mathcal{Y})$ denoted as $\frac{\phi(\mathcal{X})}{\phi(\mathcal{Y})}$ is a potential $\phi(\mathcal{Z})$ such that for each $z \in \mathcal{D}_\mathcal{Z}$*

$$\phi(\mathcal{Z}) = \begin{cases} \frac{\phi(\mathcal{X}^{\downarrow z})}{\phi(\mathcal{Y}^{\downarrow z})} & \textit{if} \quad \phi(\mathcal{Y}^{\downarrow z}) > 0 \\ 0 & \textit{if} \quad \phi(\mathcal{Y}^{\downarrow z}) = 0 \end{cases} \tag{2.13}$$

2.1.5 Complexity of Probabilistic Reasoning

Calculating beliefs over different combinations of variables can be performed by using the full JPD $P(\mathcal{V})$. This computation is known to be NP-hard (see [1]) since the complexity increases exponentially with the number of variables in $\mathcal{V}$, i.e. cardinality.

Let us denote the cardinality of $\mathcal{V}$ by $n = |\mathcal{V}|$. The cardinality of the largest variable space is denoted as $d = max_{\mathcal{V}}|\mathcal{D}_{\mathcal{V}}|$. In *worst case* we need to acquire $O(d^n)$ parameters in order to describe $P(\mathcal{V})$ fully. This problem is referred to as the *acquisition complexity*.

Moreover, suppose that we observe state a_1 of variable A and we want to calculate the posterior probability $P(B|a_1)$ for $B \in \mathcal{V}$. We can do this by updating the JPD to $P(\mathcal{V}|a_1)$ and then marginalizing it to get $P(B|a_1)$. To calculate $P(\mathcal{V}|a_1)$ we can use the product rule:

$$P(\mathcal{V}|a_1) = \frac{P(\mathcal{V}, a_1)}{P(a_1)} \tag{2.14}$$

the denominator $P(a_1)$ is calculated by just summing over all remaining terms. Each remaining term is then divided by the sum. This process is called *normalization*. Notice however, that this operation has complexity $O(d^n)$ and is quite expensive. This problem is referred to as *updating complexity*.

Finally, marginalization has to be performed by:

$$P(B|a_1) = \sum_{\mathcal{V}\backslash\{B\}} P(\mathcal{V}\backslash\{B\}, B|a_1) \tag{2.15}$$

this will result in summing over $O(d^n)$ probability values. This problem is called *marginalization complexity*.

2.2 Bayesian Networks

We have seen in section 2.1.5 that belief updating using the full JPDs is not very efficient. BNs on the contrary can be used to perform far more efficient belief updating by exploiting conditional independence. Graphical models of BNs capture conditional independence of variables, that is, given a conditional probability $P(A|B, C)$ we know that $P(A|B, C) = P(A|B)$ if variables A and C are conditionally independent and vice versa. Exploiting conditional independence between variables can significantly reduce the number of computations needed for belief updating.

The formal description of BNs is given in definition 2.4.

Definition 2.4 (Bayesian Network (BN)) *A* **Bayesian Network** *is a triplet of the following form $(\mathcal{V}, G, \mathcal{P})$. $\mathcal{V}$ is a set of variables. G is a Directed Acyclic Graph (DAG) whose nodes correspond to members of $\mathcal{V}$ such that they are connected by directed edges. $\mathcal{P}$ is a set of probability distributions:*

$$P = \{P(V_i|\pi(V_i))|V_i \in \mathcal{V}\} \tag{2.16}$$

where V_i denotes a element of $\mathcal{V}$ and $\pi(V_i) \subset \mathcal{V}$ denotes the parents of V.

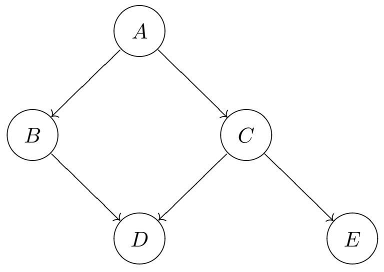
In figure 2.1 an example of a BN is given.

In this example the graph is a DAG, because there are no directed cycles and $\mathcal{V} = \{A, B, C, D, E\}$. The directed arrows in the BN have attached CPTs to represent the conditional uncertainties of variables $\{B, C, D, E\}$. Variable A does not have a CPT, but instead, a prior probability distribution.

2.2.1 The chain rule

Given the structure of the BN in figure 2.1 we can calculate the joint probability distribution $P(A, B, C, D, E)$ by using definition 2.5.

Figure 2.1: A Bayesian Network



Definition 2.5 (Chain rule for Bayesian networks) *Let a BN be defined over universe $\mathcal{V} = \{V_1, \ldots, V_n\}$, then the joint probability distribution $P(\mathcal{V})$ can be calculated by multiplying all (conditional) probabilities that are defined in the BN.*

$$P(\mathcal{V}) = \prod_i P(V_i|\pi(V_i)) \tag{2.17}$$

where $\pi(V_i)$ is the parent set of V_i.

The formula of the joint probability distribution of $P(A, B, C, D, E)$ will be

$$P(A, B, C, D, E) = P(A)P(B|A)P(C|A)P(D|B, C)P(E|C)$$

Notice that node A does not have any parents so that $P(A|\pi(A))$ is reduced to $P(A)$.

Definition 2.5 can also be used to compute JPDs over subsets of the universe $\mathcal{V}$. For example, if we want to calculate $P(A, C)$ we can multiply all potentials and marginalize over the variables that are not defined in the desired marginal probability.

$$P(A, C) = P(A)P(C|A) \sum_B P(B|A) \sum_D P(D|B, C) \sum_E P(E|C)$$

2.2.2 Evidence

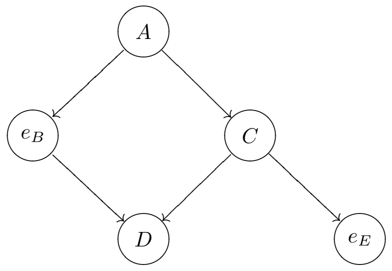
There are different types of observations, namely observations in the form of *hard evidence* and *soft evidence* (also called likelihood evidence).

Definition 2.6 (Hard Evidence) *When we receive evidence for a stochastic variable X we call it* **hard evidence** *if the probability $P(x_i)$ associated with one state x_i from the domain $\mathcal{D}_X = \{x_1, \ldots, x_n\}$ is set to one and the probabilities of other states are set to zero. In other words, if x_i is observed with certainty, we set $P(x_i) = 1$ and $\forall x_j \neq x_i : P(x_j) = 0$. We represent this through a hard evidence potential $\phi(e_X)$, where the entry corresponding to x_i is a positive number while other entries are set to 0.*

Definition 2.7 (Soft Evidence) *When we receive evidence for a stochastic variable X we call it* **soft evidence** *if no state from the domain $\mathcal{D}_X = \{x_1, \ldots, x_n\}$ is associated with probability 1. In other words, we do not know which of the possible states was observed with certainty. We represent this through a soft evidence potential $\phi(\overline{e}_X)$, where the entries have arbitrary positive numbers.*

In figure 2.2 we can see the network from figure 2.1 with two hard evidence observations e_B and e_E. This means that one state in variable B and one state in variable E are associated with probability $P(B = b_i) = 1$ and $P(E = e_j) = 1$, respectively while the probabilities of other states from the domains $\mathcal{D}_B \backslash b_i$ and $\mathcal{D}_E \backslash e_i$ are all set to zero (i.e. events corresponding to states b_i and e_j have taken place).

Figure 2.2: A Bayesian Network with hard evidence observations



With soft evidence, on the other hand, we do not know for certain what we have observed. For example, when we want to determine the color of a car in a dark alley. We cannot determine if the car is black or blue with certainty, but we can merely gather evidence about the degree of the car being blue or black.

Updating beliefs in BNs can be performed by multiplying the evidence vectors e_{X_i} with the potentials of a BN.

$$P(\mathcal{V}|\mathcal{E}) \cdot P(\mathcal{E}) = \phi(\mathcal{V}) \cdot \overbrace{\prod_i^n \phi(e_{X_i})}^{\text{Evidence set } \mathcal{E}} \tag{2.18}$$

where $\mathcal{E} = \{e_{X_1}, \ldots, e_{X_n}\}$ is a set of observations and $\mathcal{V}$ are all variables defined in the domain. Remember that potential multiplication is not defined through a matrix multiplication (see definition 2.2 how potentials should be multiplied).

In section A.2 in the appendix an example is described how updating of beliefs can be performed after the instantiation of hard evidence.

2.2.3 D-Separation and Conditional Independence

A very important concept in BNs is d-separation, which captures independencies between different variables. Variables A and C are independent given B if instantiation of A does not effect the belief of C and vice versa.

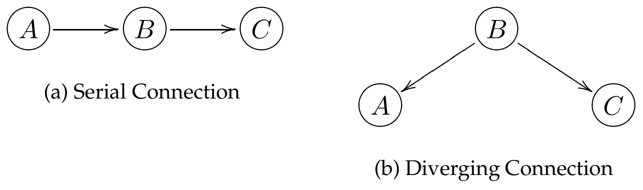
Definition 2.8 (Conditional Independence) *Variable A and variable C are conditional independent given variable B if the following equation holds:*

$$P(A|B) = P(A|B, C) \tag{2.19}$$

This can also be written as $\langle \mathcal{X}, \mathcal{Z}, \mathcal{Y} \rangle$ where $A \in \mathcal{X}$, $B \in \mathcal{Z}$ and $C \in \mathcal{Y}$. $\langle \mathcal{X}, \mathcal{Z}, \mathcal{Y} \rangle$ means that all pairs from $\mathcal{X}$ and $\mathcal{Y}$ are conditionally independent given variables from $\mathcal{Z}$ (see also definition 3.4 in section C).

In BNs three types of connections can be identified, namely serial connections (see figure 2.3 (a)), diverging connections (see figure 2.3 (b)) and converging connections (see figure 2.4).

Figure 2.3: A serial and diverging connection



If we inspect figure 2.3 (a) we can say that variable A is *d-separated* from variable C by variable B, given that B is observed (i.e. $P(B = b_i) = 1$). Variable B is blocking evidence from variable A to variable C and that is the reason why variables A and C are d-separated. A result of this is that evidence that is entered in variable A will not have any impact on the belief over variable C. We can say the same for diverging connections. In figure 2.3 (b) also variable A and C are d-separated by variable B, given that B is observed.

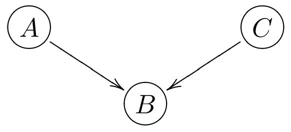


Figure 2.4: A converging connection

The third type of connection is the *converging connection* given in figure 2.4. A connection is not d-separated when variable B is observed. This is different from the other two connections where variable B has to be observed in order for a connection to be d-separated. In this case A and C are said to be *marginally independent* when B is not observed.

Definition 2.9 (Marginal Independence) *A variable A and a variable C are marginally independent if the following equation holds:*

$$P(A, C) = P(A)P(C) \tag{2.20}$$

When variable B is observed, in the BN of figure 2.4, the entered evidence in B will not block evidence between variables A and C. If evidence is instantiated in variable A this will affect the belief of the other parents of variable B (in this case only variable C). This is called the *explaining away* effect (see [6]).

In definition 2.10 a formal description is given of d-separation.

Definition 2.10 (D-separation) *Two distinct variables A and B are d-separated in a causal network if, for all paths between A and B there is a variable V such that the connection is either serial or diverging and variable V is instantiated or the connection is converging and neither V or any of its descendants are instantiated (have received evidence).*

In section A.3 in the appendix an example is presented of d-separation in a larger BN and in section B is explained why conditional independence occurs in given types of connections.

Why is d-separation important? With the knowledge of d-separation we can minimize the size of JPTs to do belief updating. Using the full JPT over all defined variables in a BN is, in general, unnecessary. Belief updating can be inefficient when a large potential is introduced through multiplication of smaller potentials. With d-separation we know which of the variables are dependent and which are not. This qualitative knowledge can be used to identify the smaller potentials that can be used for efficient belief updating.

3 Bayesian Inference

IN this chapter we review the basic principles of inference in Bayesian networks. In general, BNs explicitly encode conditional independence, which can be exploited for efficient inference.

There are several approaches to belief propagation in BNs. For singly connected Bayesian networks (poly trees) the approach by Pearl in [17] can be used. This approach is also known as $\lambda - \pi$ message passing. For multiply connected Bayesian networks *cutset conditioning* is used (described in [18]). With cutset conditioning the multiply connected networks are transformed to several singly connected networks and on these belief propagation is performed.

Another approach to inference in multiply connected BNs is based on Junction Trees (JTs) proposed by Lauritzen and Spiegelhalter.

Junction Trees (JTs) facilitate efficient Bayesian inference (belief propagation) in BNs. Every BN can be transformed into a JT in such a way that a significant portion of the conditional independence encoded in the BN is preserved. This qualitative knowledge entails d-separation that is used to form the JT. After the probabilistic knowledge of a BN is represented in an efficient JT, this graphical model can be used for efficient belief updating through *concise message passing*. Concise messages contain marginal probability distributions, obtained by marginalization, over a minimal set of stochastic variables, that are communicated between graphically separated clusters of stochastic variables. The order in which concise messages are allowed to be passed between sets of stochastic variables corresponds to an efficient marginalization order. In other words, JTs impose, on the belief propagation process, an efficient marginalization order that is based on the connections between the JT clusters.

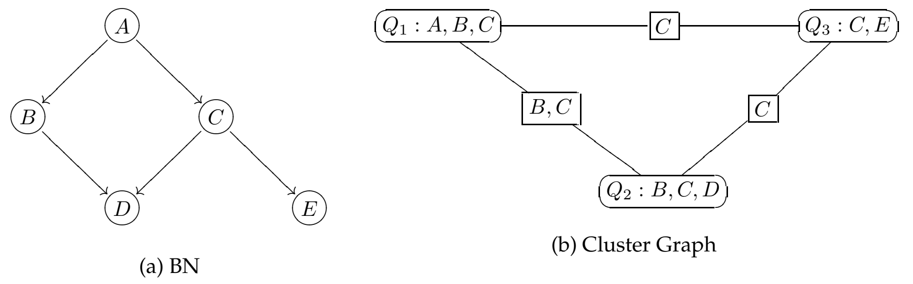
In this chapter we will discuss the graphical structure of JTs and how efficient belief propagation can be performed on JTs. In section 3.1 we will show that a JT is a special type of constraint cluster graph that enables consistent belief updating through concise message passing. In section 3.2 we discuss the belief propagation algorithms.

3.1 Cluster Graphs

In chapter 2 we have shown that conditional independence supports efficient Bayesian belief updating. Conditional independence in a BN is used to convert this BN to a *cluster graph*. A cluster graph encodes the conditional independence of the BN in a coarser way. When a marginal posterior probability is computed the cluster graph can be used to guide the marginalization order of variables. Every BN has, in general, several cluster graphs. However, not every cluster graph supports efficient and correct belief propagation. The cluster potentials that are defined for every cluster in a cluster graph should be minimal in the number of variables. In addition, most cycles in cluster should be prevented, because they do not support consistent belief updating.

In figure 3.1 a cluster graph of a BN is depicted. Note that the cluster graph has undirected edges instead of directed. The undirected edges encode probabilistic dependencies between two subsets of variables*, which are also called *clusters*. Between these two clusters messages with probabilistic knowledge can be passed. These messages should be of low complexity, which means that they should not be defined over a large set of variables. If the number of variables in a cluster is low *concise message passing* between clusters is possible.

Figure 3.1: A Cluster Graph



From figure 3.1 (b) we see that three clusters are defined over a domain of variables given in the BN.

Definition 3.1 (Cluster) *A cluster Q_i corresponds to a set of variables $\mathcal{V}_i$ where $\mathcal{V}_i$ can consist of an arbitrary subset of nodes from a BN.*

When two clusters have a non-empty intersection they can be connected where the non-empty set is called a *separator* of the two clusters. We can see in figure 3.1 (b) that Q_1 and Q_2 have domains $\{A, B, C\}$ and $\{B, C, D\}$, respectively, and share the separator set $Q_1 \cap Q_2 = \{B, C\}$. The variables in the clusters Q_1 and Q_2 are independent of each other given the variables defined in the separator. That means that variables $A \in Q_1$ and $D \in Q_2$ are separated from each other because they are conditionally independent given $\{B, C\}$.

It turns out that not every type of cluster graph is a suitable structure for belief updating through concise message passing. Only a special type of cluster graphs supports consistent belief updating. In this section different types of cluster graphs are discussed and finally the requirements on a cluster graph, which are necessary for consistent belief updating, will be presented.

3.1.1 Cluster Graphs with Cycles

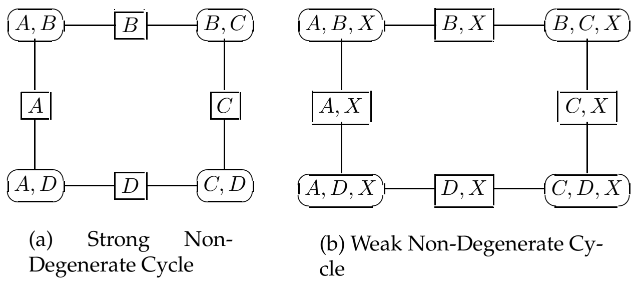
Two different types of cycles can be defined in a cluster graph, namely: non-degenerate cycles and degenerate cycles. A cycle ρ in a cluster graph is *strong non-degenerate* if every separator on ρ is different and a cycle ρ is *weak non-degenerate* if every separator S_i is different, but $\bigcap_i S_i \neq \emptyset$. In figure 3.2 the two different non-degenerate cycles are depicted.

It turns out that no matter how messages are passed in cluster graphs with non-degenerate cycles consistent belief updating is not possible (see [21]).

Another type of cluster graph is a cluster graph with a path ρ where a separator S on this path is contained in every other separator. Such a path is called a

* Directed edges in BNs often correspond to causal relations between variables

Figure 3.2: A Cluster Graph with a strong non-degenerate cycle (a) and a Cluster Graph with a weak non-degenerate cycle (b)

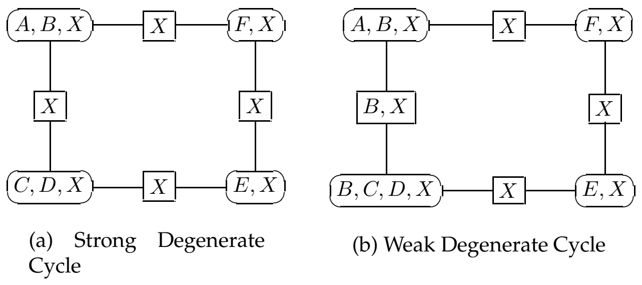


(a) Strong Non-Degenerate Cycle

(b) Weak Non-Degenerate Cycle

degenerate cycle. Figure 3.3 shows two different degenerate cycles, namely one with a strong degenerate cycle and one with a weak degenerate cycle.

Figure 3.3: A Cluster Graph with a strong degenerate cycle (a) and a Cluster Graph with a weak degenerate cycle (b)



(a) Strong Degenerate Cycle

(b) Weak Degenerate Cycle

Note that only cluster graphs that contain degenerate cycles support correct belief updating. In such cases the cycles can be cut to make a *cluster tree*. While it does not matter where a strong degenerate cycle is cut, a weak degenerate cycle can be cut only at certain separators. This brings us to a special type of cluster tree.

3.1.2 Junction Trees

It turns out that consistent belief updating is possible in a special type of cluster tree, namely the *junction tree* (JT)[†]. JTs have the *running intersection property* which is required to do consistent belief updating. Without this property consistent belief updating is not guaranteed.

Definition 3.2 (Junction Trees) *A cluster tree is a junction tree (JT) if for every pair of clusters Q_i and Q_j, $Q_i \cap Q_j$ is contained in every cluster on the path ρ between Q_i and Q_j. This is called the* **running intersection property**.

If we return to the cluster graph example given in figure 3.1 (b) we can cut the weak degenerate cycle. Cutting this cycle between the clusters Q_1 and Q_2 will result in the cluster tree given in figure 3.4 (a). This cluster tree is not a junction tree, because it does not have the running intersection property. The intersection between clusters Q_1 and Q_2 is $Q_1 \cap Q_2 = \{B, C\}$ where $\{B, C\} \nsubseteq Q_3$. Cutting the cluster graph between clusters Q_1 and Q_3 (given in figure 3.4 (b)) or between Q_3 and Q_2 will result in a junction tree.

In figure 3.5 another example of a junction tree is given. In this figure we can see that the intersection of cluster $\{A, B\}$ and cluster $\{C, E\}$ result in $\emptyset$. This means that this cluster tree is also a junction tree.

[†] In the Bayesian inference literature the definition of a JT differs. In [21] a JT is a collection of connected clusters while in [6] a JT is a *join tree* consisting of cliques with attached potentials. In this thesis the definition of a JT given in [21] is followed.

Figure 3.4: Two cluster trees

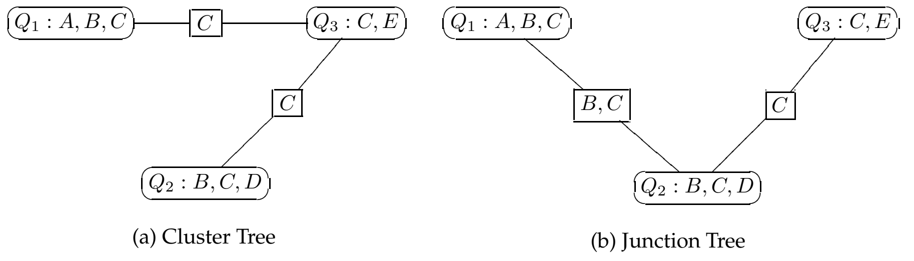
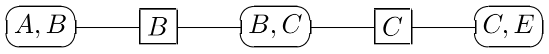


Figure 3.5: A Junction Tree



In appendix C the procedure of building a JT from a BN is explained.

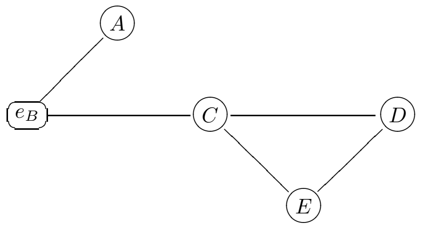
3.1.3 Independence Map

When a DAG is converted to a JT probabilistic independence should be preserved as much as possible in order to have an efficient computational probabilistic model. D-separation between variables in a DAG can be extended to the concept of *u-separation* for undirected graphs.

Definition 3.3 (U-Separation) *Let G be an undirected graph (also Markov Graph) and $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ disjoint sets of nodes in G. Evidence is blocked on a path ρ between nodes $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$ if $Z \in \mathcal{Z}$ is on ρ. Nodes X and Y are u-separated by $\mathcal{Z}$ if for every path between X and Y evidence is blocked. $\mathcal{X}$ and $\mathcal{Y}$ are u-separated by $\mathcal{Z}$ if for every $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$, X and Y are u-separated. This is denoted by $\langle \mathcal{X} | \mathcal{Z} | \mathcal{Y} \rangle_G$*

In figure 3.6 a Markov graph G is depicted where the variable B is instantiated to state b_1. According to definition 3.3 variable A is u-separated from variables $\{C, D, E\}$ given variable B, hence $\langle A | B | \{C, D, E\} \rangle_G$.

Figure 3.6: A Markov Graph



Conditional independence that is encoded in a BN (see section 2.2.3) needs to be preserved as much as possible during the transformation to the undirected graph. This can be achieved by keeping the *independence map*, defined in definition 3.4, minimal. A minimal I-map means that there is no subgraph contained in the I-map that is also an I-map.

Definition 3.4 (Independence Map (I-Map)) *A directed or undirected graph G is an independence map or I-map of a domain $\mathcal{V}$ if there is a one-to-one correspondence between nodes of G and variables in $\mathcal{V}$ and $\langle \mathcal{X} | \mathcal{Z} | \mathcal{Y} \rangle_G$ implies $I(\mathcal{X}, \mathcal{Z}, \mathcal{Y})$ for all disjoint subsets $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ of $\mathcal{V}$.*

The notation $I(\mathcal{X}, \mathcal{Z}, \mathcal{Y})$ is used to denote conditional independence between variable $\mathcal{X}$ and $\mathcal{Y}$ given $\mathcal{Z}$.

Adding links to an I-map does not change the I-mapness. However, conditional independence in this I-map will become invisible at the cost of less efficient belief updating. For example, when we add all possible links that can be added to some graph G every variable will be connected to each other variable. This means that all present conditional independence is invisible and will force us to do belief updating using the full JPD. In a JT we want to have concise messages of probabilistic knowledge between conditional independent sets of variables given the separator. When we preserve minimal I-mapness concise message passing in a graphical structure can be done efficiently.

3.1.4 Potential Assignment

The quantitative knowledge of a BN, that is the defined CPTs and priors, should be transferred to the corresponding JT. This quantitative knowledge is represented through cluster potentials. In figure 3.7 (a) we can see a BN with the corresponding JT with assigned potentials in figure 3.7 (b).

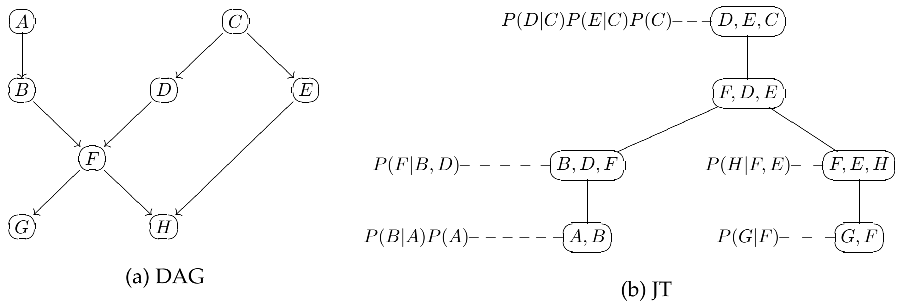


Figure 3.7: Potential assignment to cluster of a JT

These associated potentials are assigned in the following way: give all clusters and separators in the JT given in figure 3.7 (b) a uniform potential[‡] to avoid biased beliefs. Select a node v from figure 3.7 (a). Determine $\tau(v) = \{v, \pi(v)\}$ where $\pi(v)$ is the set of parent nodes and $\tau(v)$ are the parents of v and v itself. If $\tau(v) \subseteq Q_i$ attach the potential $P(v|\pi(v))$ to cluster Q_i and multiply $\phi(Q_i) \cdot P(v|\pi(v))$. Do the same for the remaining nodes. It can happen that some of the determined clusters in a JT do not get a CPT value assigned. In the example of figure 3.7 the cluster $\{F, D, E\}$ only has a uniform potential assigned. This is because the cluster $\{F, D, E\}$ doesn't cover a CPD. The parent C of variable D and E is not contained in the cluster and one of the parents of F is also not contained in the cluster. The uniform potential is assigned to these type of clusters to avoid duplicate specifications of the CPDs which will result in inconsistent reasoning.

How do the potentials of a JT relate to the JPD of all the variables defined in the problem domain? In theorem 3.1 the relation between JT potentials and the full JPD is given.

Theorem 3.1 *Let a JT T with a set of clusters $\mathcal{Q}$ and a set of separators $\mathcal{S}$ be an I-map over all domain variables $\mathcal{V}$. For each cluster Q, $\phi(Q) = \alpha_1 \cdot P(Q)$, and for each separator S, $\phi(S) = \alpha_2 \cdot P(S)$ where α_1 and α_2 are normalization constants. Then the joint probability distributions over $\mathcal{V}$ can be calculated with*

$$P(\mathcal{V}) = \alpha \cdot \frac{\prod_{Q \in \mathcal{Q}} \phi(Q)}{\prod_{S \in \mathcal{S}} \phi(S)} \tag{3.1}$$

A proof of this theorem can be found in [21] on page 95.

[‡] Every entry in a uniform potential has the value 1

3.2 Belief Updating in Junction Trees

Whenever we want to calculate a marginal probability of a variable in a cluster we have to make sure that the JT is globally consistent. Global consistency can be realized by concise message passing between clusters in a JT. A cluster can update its belief by considering the beliefs of neighboring clusters and update their own belief. If every cluster updates its belief in this way the JT will be globally consistent. Before message passing in JTs is discussed, first the definition of consistence between clusters should be defined.

Definition 3.5 (Consistence) *Given a cluster graph G that contains two adjacent cluster Q_i and Q_j with a separator $S\langle Q_i, Q_j\rangle$ with associated potentials $\phi(Q_i)$, $\phi(Q_j)$ and $\phi(S)$. Clusters Q_i and Q_j are consistent if:*

$$\sum_{Q_i \setminus S} \phi(Q_i) = \alpha_1 \cdot \phi(S) = \alpha_2 \cdot \sum_{Q_j \setminus S} \phi(Q_j) \tag{3.2}$$

where α_1 and α_2 are normalization constants. G is **locally consistent** *if all adjacent clusters are consistent. G is* **globally consistent** *if all pairs of clusters are consistent according to:*

$$\sum_{Q_i \setminus Q_j} \phi(Q_i) = \alpha \cdot \sum_{Q_j \setminus Q_i} \phi(Q_j) \tag{3.3}$$

Section 3.1.4 discussed how potentials should be assigned to clusters in a JT. Only assigning potentials to clusters is not enough to calculate correct marginal posteriors. This is because the JT is not globally consistent. We need to pass concise messages between clusters in a JT in order to make this JT globally consistent. The basic operation that is needed to get two clusters consistent with each other is called *absorption*[§].

Algorithm 3.1: Absorption

```
input  : two adjacent clusters $Q_i$ and $Q_j$ with separator $S$ in a JT with the
         following potentials $\phi(Q_i)$, $\phi(Q_j)$ and $\phi(S)$, respectively.
output: updated separator potential $\phi'(S)$ and cluster potential $\phi'(Q_i)$
1 if cluster $Q_i$ absorbs from cluster $Q_j$ then
2     Update $\phi(S)$ to:
          $$\phi'(S) = \sum_{Q_j \setminus S} \phi(Q_j) \tag{3.4}$$
      Update $\phi(Q_i)$ to:
          $$\phi'(Q_i) = \phi(Q_i) \cdot \frac{\phi'(S)}{\phi(S)} \tag{3.5}$$
3 end
```

Two connected clusters are consistent with each other if absorption, given in algorithm 3.1, is performed on both clusters. In section A.4 in the appendix an example of absorption is given by using a simple JT with two clusters. When a JT has more than two clusters all the clusters must be made consistent to each other. This can be done by performing first the algorithm `CollectEvidence` given in algorithm 3.2 and after that one `DistributeEvidence` given in algorithm 3.3. These two algorithms are combined in the algorithm `UnifyBelief` given in algorithm 3.4 (algorithms adopted from [21]).

Performing the `UnifyBelief` on a JT T will bring T into globally consistent state (see [21] for proof). If T is in globally consistent state correct marginal posterior probabilities can be calculated of every variable in any cluster.

[§] Absorption is based on the Hugin inference approach

Algorithm 3.2: Collecting evidence to a cluster

```
procedure: CollectEvidence
input    : $Q_i$ is a cluster in a JT $T$. A caller is either an adjacent cluster or $T$.
1 if the caller calls $Q_i$ then
2     Cluster $Q_i$ calls CollectEvidence in each adjacent agent except in
      caller;
3     After each called cluster has finished $Q_i$ absorbs from it;
4 end
```

Algorithm 3.3: Distributing evidence to clusters

```
procedure: DistributeEvidence
input    : $Q_i$ is a cluster in a JT $T$. A caller is either an adjacent cluster or $T$.
1 if the caller calls $Q_i$ then
2     If caller is a cluster, $Q_i$ absorbs from it;
3     Cluster $Q_i$ calls DistributeEvidence in each adjacent cluster except in
      caller ;
4 end
```

Algorithm 3.4: Bring JT into globally consistent state

```
procedure: UnifyBelief
input    : Junction tree $T$
1 Select cluster $Q_i$ in $T$;
2 Call CollectEvidence on $Q_i$;
3 if CollectEvidence is finished then
4     Call DistributeEvidence on $Q_i$;
5 end
```

The following algorithm can be used to update the beliefs after a set of observations $\mathcal{E}$ are entered into the JT T:

Algorithm 3.5: Instantiating evidence

```
procedure: EnterEvidence
input    : set of observations $e_{X_i} \in \mathcal{E}$
1 Find for each $X_i \in \mathcal{X}$ a cluster $Q_i$ in junction tree $T$ where $X_i \in Q_i$;
2 Replace $\phi(Q_i)$ with $\phi(Q_i) \cdot \phi(e_{X_i})$;
```

Algorithm3.5 is also applicable for soft evidence observations.

When a set of observations are processed and algorithm 3.5 will be used to update the potentials in the JT, it will bring the JT in a non globally consistent state. After performing `UnifyBelief` the JT will be in globally consistent state again and correct (marginal) posteriors can be calculated with all observations taken into account.

Part II

Distributed Probabilistic Inference

IN the foregoing part of this thesis Bayesian networks were discussed in a non-distributed environment. However, Bayesian networks used in the non-distributed environment might not be the most optimal setting. For example, when the problem domain is too complex to be modeled in a single agent because of computational limitations or the problem domain is geographically dispersed. In such cases it is more natural to model the problem domain in a distributed manner. However, using distributed algorithms to do belief updating in BNs can be very challenging: i) Belief propagation has to be performed in a consistent manner, ii) the belief propagation algorithm has to be robust against bad communication channels over which belief has to be propagated between agents, iii) the algorithm needs be robust against failing agents.

In this thesis three different approaches to probabilistic inference in a distributed environment are discussed:

- Distributed Perception Networks (DPNs) (see chapter 4);
- Multiply Sectioned Bayesian Networks (MSBNs) (see section 5.1);
- Prior/Likelihood Decomposable Models (PLDMs) (see section 5.2).

Before we talk about multiagent distributed inference using BNs we first describe the concepts *agent* and *multiagent system*. Unfortunately, there is no real agreement on what an agent exactly is (see [5] for a discussion), therefore we define an agent in the context of distributed inference.

An agent, that concerns itself with distributed probabilistic inference, is a computer program that is *situated* in the physical world, is able to receive information from sensors or other agents and can act *autonomously* upon receiving of relevant information. The task of a probabilistic inference agent is (i) to compute its local belief with respect to the received information originating from other cooperating agents and (ii) supply its partial belief to other *interested* agents. Given this task, agents have a clear goal-directed behavior and are able to collectively perform a global task. A multiagent system for probabilistic inference must be able to compute correct posterior probabilities over certain variables by using the evidence obtained from different cooperating agents. Of course, these agents have to be connected to each other through communication channels and should be aware of the agent(s) that are interested in their local belief.

4 Distributed Perception Networks

LATEST development in communication and sensor technology resulted in numerous easily accessible information sources. For example, buoys that are dispersed in oceans to observe weather phenomena, tsunami warning buoys, humans with PDA's or mobile phones, etc. The information that comes from these information sources can be used for situation awareness. Autonomous intelligent systems or decision makers can use this knowledge to respond to relevant aspects in their environments. However, this requires that information from different sources is mapped to relevant hypotheses about events that cannot be observed directly. This can be challenging because of:

- Sensor outputs are subject to very noisy or heterogeneous information;
- Relevant information sources must be found and integrated. These information sources are not known prior to operation;
- The relevant constellation of information sources must be captured. Since the information sources can become available or unavailable during runtime;
- The available information is often very uncertain and subjective;
- Huge amounts of information is provided and must be processed, and consequently, a central model might not be appropriate due to the processing and communication overload at the central processing unit.

Distributed Perception Networks (DPNs) [15, 3, 2] are a probabilistic inference approach that can deal with these challenges. It provides an inference structure that is able to collect information from relevant information sources and map them to distributions over hypotheses of interest in a reliable and efficient manner.

This chapter is organized as follows: in section 4.1 we use an example in order to illustrate the domains were DPNs are applicable. We argue that many domains can be described through causal models, which in turn facilitates distributed modeling and inference. In section 4.2 we describe how probabilistic models can efficiently be distributed throughout a system of independent processing units. In section 4.3 we describe algorithms that support automated assembly of distributed probabilistic models, which support efficient and robust distributed belief propagation. In section 4.4 we describe inference in distributed BNs.

4.1 Causal Models

Causal processes can be viewed as sequences of events*, where some events cause other events. That means that these events are causally related to each other. Such processes can easily be described through causal models. Causal models are defined through directed graphs where vertices denote events and directed edges denote cause-effect relations. By using causal models we can represent different observations and their causes in a systematic way.

Consider a situation were we want to estimate the presence of a lethal concentration of toxic gas `GasX` in a populated area A. We assume that two types of sensors are installed in area A. One sensor type detects a lethal concentration of `GasX` by measuring the conductivity of the gas mixture and the other sensor type measures the ionization level. Next to the sensor measurements, humans that come in contact with `GasX` will develop specific symptoms such as coughing, headache, nausea, cyanosis, etc. Such symptoms are observable and can be used, next to the two sensor types, as possible *information sources* to detect `GasX`. This is possible because these information sources are causally related to `GasX`. If `GasX` is present in area A, the sensors start measuring certain conductivity and ionization levels, humans will smell the gas, etc. In other words, these observable events were caused by the presence of `GasX`. Thus, we can use these observable *effects* to reason back to the *cause*. Figure 4.1 shows a causal model which explicitly describes causal relations between the relevant events in this example.

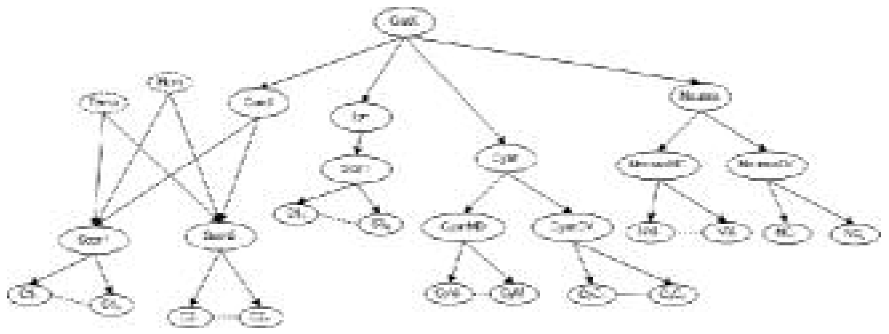


Figure 4.1: A causal model that captures causal relations between information sources (like sensors and humans) and the existence of a lethal concentration of toxic gas

For the given concentration of `GasX`, the materials used in gas sensors could have a certain conductivity that would result in correct detection of `GasX` with a high probability if the sensor hardware would work properly. Such a situation is captured in the model by node labeled `Cond`. Moreover, a situation represented by node `Cond` in combination with air temperature `Temp` and humidity `Hum` will result, with a certain probability, in situations, in which the majority of the measurements obtained with the sensor will indicate the presence of `GasX`. We call such a situation *sensor status*. The model in figure 4.1 contains nodes `Scond1` and `Scond2`, each denoting a sensor status of a sensor that measures conductivity.

Moreover, the sensor status of a particular sensor results, with a certain probability, in a measurement that will indicate the presence of `GasX`. For example, sensor status `Scond1` will produce a sequence of measurements denoted by variables $\mathtt{C1}_i$. Because the sensor corresponding to sensor status `Scond1` is inherently noisy, there is a chance that a measurement can still indicate absence of `GasX`, despite the fact that the gas concentration exceeded the critical threshold and the sensor components were working properly.

In general, we assume that the observations resulting from a causal process are captured through the leaf nodes as for example $\mathtt{S1}_i$, $\mathtt{C1}_i$, $\mathtt{C2}_i$, etc. (see figure 4.1). Every measurement must be represented by a node in the model.

Node `Sion` represents status of a sensor that measures the ionization level.

* In this thesis an event is synonymous to a realization of a certain situation, i.e. a state of affairs defined through a set of states.

Also in the figure we can see node Cyan that corresponds to cyanosis. For the Cyan node two states are possible: one state represents the presence of cyanosis and the other the absence of cyanosis. Humans can determine, with certain probability, when cyanosis is present. The nodes CyanMD and CyanCV correspond to physicians and civilians that can observe the presence of cyanosis, respectively. A physician will determine whether cyanosis is the case with higher certainty than civilians, because physicians are experts. In the same way the nodes Nausea, NauseaMD, NauseaCV can be explained.

In addition, in the causal model in figure 4.1 there are two types of variables, namely *process variables* and *environment variables*. Process variables are influenced through a stochastic process, for example the variables Scon1, Cyan, etc. While environment variables influence the causal process where the causal process itself does not influence the states of the environment variables. Examples of environment variables in figure 4.1 are Temp and Hum which corresponds to the temperature and humidity, respectively. Temperature and humidity can influence the sensitivity of a sensor, but the sensitivity of the sensor does not influence the temperature or humidity.

4.1.1 Temporal Aspects

Each sequence of hidden states resulting in a particular observation takes place in a finite time interval. In other words, there exist a finite time interval between the materialization of the hidden state of interest and an observation of a symptom. Obviously, estimation makes sense in domains where hidden events are quasi static; i.e. after unobservable states materialized they do not change before the resulting observations are interpreted and used in a decision making process. In other words, relations between the hidden variables capture the so called statistical time (i.e. certain events preceded other events), however, the sets of the hidden states do not evolve for a certain period of time after they have materialized. In this context, we introduce a fusion time slice, a period of time during which we gather and process observations whose hidden causes, i.e. sets of hidden states, do not change after they have materialized.

For example, the causal model in figure 4.1 shows primarily static events (except for the leaf nodes), while we could argue that the presence of GasX is influenced by time. However, we can still model presence of GasX in this way, because most events have a quasi-static behavior. If we assume that GasX is present in area A, GasX will cause, with a certain probability, that several humans suffer from cyanosis. The gas and the presence of cyanosis will *exist* for a period of time in area A. Changes in the concentration of the gas will not influence the existence of cyanosis anymore. Therefore we can model events like presence of gas and the presence of cyanosis as static events for a finite period of time. Of course, eventually the gas and the cyanosis will vanish, hence the name *quasi-static events*.

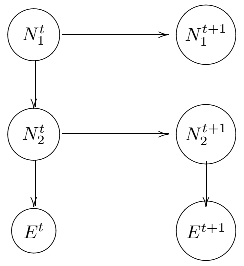
Observation sequences introduce dynamics to the model, which can be described through a class of dynamic BNs (DBNs) with distinctive topological features (see figure 4.2).

In order to be able to model quasi-static events and the dynamics of observation sequences we introduce two types of nodes:

- *Quasi-static nodes* capture events that do not change during the fusion process;
- *Dynamic nodes* capture different types of sensor information, which is due to the processing noise not 100% reliable.

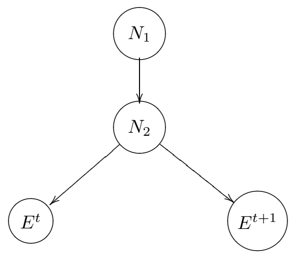
Figure 4.2 shows a BN modeling two time slices corresponding to t and $t+1$. In the first time slice a quasi-static event corresponding to node N_1^t causes another quasi-static event corresponding to node N_2^t. Finally, the event corresponding to node N_2^t will cause an observation represented by node E^t. Note that this sequence of events within a single time slice reflects statistical time. At the next time slice, there is no connection between the quasi static nodes N_1^{t+1} and N_2^{t+1}, because the event corresponding to N_1^{t+1} cannot influence the existence of the event corresponding to N_2^{t+1} anymore, e.g. after gas caused cyanosis at time slice t, the cyanosis will continue to *exist* at later time slices, independently of the concentration of the gas. But the presence of cyanosis will cause a new sequence of observations[†].

Figure 4.2: Dynamic network


$$\begin{tikzcd}
N_1^t \arrow[r] \arrow[d] & N_1^{t+1} \\
N_2^t \arrow[r] \arrow[d] & N_2^{t+1} \arrow[d] \\
E^t & E^{t+1}
\end{tikzcd}$$

The DBN given in figure 4.2 can be simplified because of the quasi-static events N_1^t and N_2^t. In figure 4.3 the quasi-static nodes N_1^{t+1} and N_2^{t+1} are collapsed onto the quasi-static nodes N_1 and N_2 respectively. This is possible because the causal relation between the nodes N_1^t and N_1^{t+1} and the causal relation between the nodes N_2^t and N_2^{t+1} are identities.

Figure 4.3: Static network equivalent to the dynamic model depicted in figure 4.2


$$\begin{tikzcd}
 & N_1 \arrow[d] & \\
 & N_2 \arrow[dl] \arrow[dr] & \\
E^t & & E^{t+1}
\end{tikzcd}$$

This example illustrates the following property:

Proposition 4.1 (DBN Simplification) *When in a DBN the non terminal nodes represent quasi-static events, captured by the nodes $\{N_i^t | 1 \leq t \leq n, 1 \leq i \leq m\}$, and the leaf nodes represent dynamic nodes then all quasi-static nodes $\{N_i^t | t \geq 2\}$ can be collapsed onto node $N_i^{t=1}$ for every i. In other words, every node N_i is time-invariant.*

A proof of this model simplification can be found in [3].

Note that this property is very important, because such a simplification allows very efficient and robust distribution of probabilistic models and inference processes through relatively simple modeling fragments (see section 4.4).

4.1.2 Problem Decomposition

Problems that are described through causal models can often be decomposed in several smaller network fragments because of d-separation (see section 2.2.3). The causal model in figure 4.1 can be seen as a BN where every causal relation has an attached CPT. Assume that we try to determine the JPD $P(\texttt{GasX}, \mathcal{E})$ where $\mathcal{E}$ are all observations collected during one time slice. The DAG structure in figure 4.1 implies a certain factorization for the JPD $P(\texttt{GasX}, \mathcal{E})$ where each factor is conditionally independent of other factors given the hypothesis node `GasX`:

[†] Note that in this example the variables have only two states representing existence of certain facts, such as gas concentration is above or below a critical threshold. This model would be inappropriate if we would reason about the gas concentration, which changes over time.

$$
\begin{aligned}
P(\mathtt{GasX}, \mathcal{E}) \quad &= \quad P(\mathtt{GasX})P(\mathcal{E}_{\mathtt{Cond}}|\mathtt{GasX})P(\mathcal{E}_{\mathtt{Cyan}}|\mathtt{GasX}) \\
&\quad \cdot P(\mathcal{E}_{\mathtt{Nausea}}|\mathtt{GasX})P(\mathcal{E}_{\mathtt{Ion}}|\mathtt{GasX}) \qquad (4.1)
\end{aligned}
$$

where $\mathcal{E}_{\mathtt{Cond}}$, $\mathcal{E}_{\mathtt{Cyan}}$, $\mathcal{E}_{\mathtt{Nausea}}$ and $\mathcal{E}_{\mathtt{Ion}}$ correspond to the instantiated states of the evidence nodes in conditionally independent subgraphs connected through nodes `Cond`, `Cyan`, `Nausea` and `Ion` to the hypothesis node `GasX`, respectively. Each factor in equation 4.1 can be computed independently, as for example $P(\mathcal{E}_{\mathtt{Cond}}|\mathtt{GasX})$:

$$
\begin{aligned}
P(\mathcal{E}_{\mathtt{Cond}}|\mathtt{GasX}) \quad &= \quad \sum_{\mathtt{Cond}} P(\mathcal{E}_{\mathtt{Scon2}}|\mathtt{Cond}, e_{\mathtt{Temp}}, e_{\mathtt{Hum}}) \\
&\quad \cdot P(\mathcal{E}_{\mathtt{Scon1}}|\mathtt{Cond}, e_{\mathtt{Temp}}, e_{\mathtt{Hum}}) \qquad (4.2)
\end{aligned}
$$

The priors `Temp` and `Hum` are observed and instantiated with hard evidence. This means that we can again identify conditionally independent factors $P(\mathcal{E}_{\mathtt{Scon1}}|\mathtt{Cond}, e_{\mathtt{Temp}}, e_{\mathtt{Hum}})$ and $P(\mathcal{E}_{\mathtt{Scon2}}|\mathtt{Cond}, e_{\mathtt{Temp}}, e_{\mathtt{Hum}})$ given the hypothesis node `Cond`. Each factor corresponds to a single sensor and can again be computed independently:

$$
\begin{aligned}
&P(\mathcal{E}_{\mathtt{Scon1}}|\mathtt{Cond}, e_{\mathtt{Temp}}, e_{\mathtt{Hum}}) = \\
\sum_{\mathtt{Scon1}} P(\mathtt{Scon1}|\mathtt{Cond}, e_{\mathtt{Temp}}, e_{\mathtt{Hum}}) &\prod_i P(\mathtt{C1}_i|Scon1)\phi(e_{\mathtt{Scon1_i}}) \qquad (4.3)
\end{aligned}
$$

and

$$
\begin{aligned}
&P(\mathcal{E}_{\mathtt{Scon2}}|\mathtt{Cond}, e_{\mathtt{Temp}}, e_{\mathtt{Hum}}) = \\
\sum_{\mathtt{Scon2}} P(\mathtt{Scon2}|\mathtt{Cond}, e_{\mathtt{Temp}}, e_{\mathtt{Hum}}) &\prod_i P(\mathtt{C2}_i|Scon2)\phi(e_{\mathtt{Scon2_i}}) \qquad (4.4)
\end{aligned}
$$

If the environment variables `Temp` and `Hum` were not observed then the factors $P(\mathcal{E}_{\mathtt{Scon1}}|\mathtt{Cond}, e_{\mathtt{Temp}}, e_{\mathtt{Hum}})$ and $P(\mathcal{E}_{\mathtt{Scon2}}|\mathtt{Cond}, e_{\mathtt{Temp}}, e_{\mathtt{Hum}})$ would not be conditionally independent and therefore independent computation is not possible.

The posterior probability distribution $P(\mathtt{GasX}|\mathcal{E})$ is computed by using the JPD $P(\mathtt{GasX}, \mathcal{E})$ and normalize with $P(\mathcal{E})$:

$$
P(\mathtt{GasX}|\mathcal{E}) = \frac{P(\mathtt{GasX}, \mathcal{E})}{P(\mathcal{E})} \qquad (4.5)
$$

4.2 Distributed Models

The factorization properties illustrated in section 4.1.2 suggest that conditional independence in causal models can be used to identify independent network fragments which correspond to independent factors. In other words, each factor can be computed independently of other factors, which implies that we can easily distribute probabilistic models and belief propagation among different processing units. These processing units can be represented through DPN agents.

Definition 4.1 (DPN Agent) *A DPN agent A_i is defined through a local BN ψ_i, which has a set of variables $\mathcal{V}_i$. Where the root variable $R \in \mathcal{V}_i$ is the service concept and the leaf variables $\mathcal{L}_i \subset \mathcal{V}_i$ are the input concepts.*

4.2.1 Input Concepts and Service Concepts

Root and leaf concepts of the local DAGs have a very important role in the DPN organization. These concepts determine how the independent network fragments can be connected to each other to form the global structure (like the causal model in figure 4.1). Network fragments can connect to each other if the root concept or *service concept* of one fragment is semantically identical (describing the same event) to leaf concept or *input concept* of another fragment. This is called the *sharing condition* between two network fragments defined in definition 4.2.

Definition 4.2 (Sharing Condition) *Agents A_i and A_j can integrate their network fragments if the* **service concept** *R_i of A_i is identical to an input concept of A_j:*

$$\{R_i\} \cap \mathcal{L}_j \neq \emptyset \tag{4.6}$$

where $\mathcal{L}_j$ is the set of **input concepts** *of agent A_j.*

In figure 4.4 we can see an example of a DPN that is formed by connecting local DAGs. The connections between local fragments must result in a separator defined in definition 4.3.

Definition 4.3 (DPN separator) *Given clusters Q_i and Q_j the separator $S\langle Q_i, Q_j\rangle$ is defined by*

$$S\langle Q_i, Q_j\rangle = Q_i \cap Q_j \quad \text{where } |S\langle Q_i, Q_j\rangle| = 1 \tag{4.7}$$

That means we can only have a connection between two network fragments if there is overlap of only one variable according to the sharing condition in definition 4.2. For example, there exists a connection between cluster $Q_i = \{\texttt{GasX}, \texttt{Ion}, \texttt{Cyan}, \texttt{Nausea}\}$ and cluster $Q_j = \{\texttt{Ion}, \texttt{Sion}\}$ because $|Q_i \cap Q_j| = |\{\texttt{Ion}\}| = 1$.

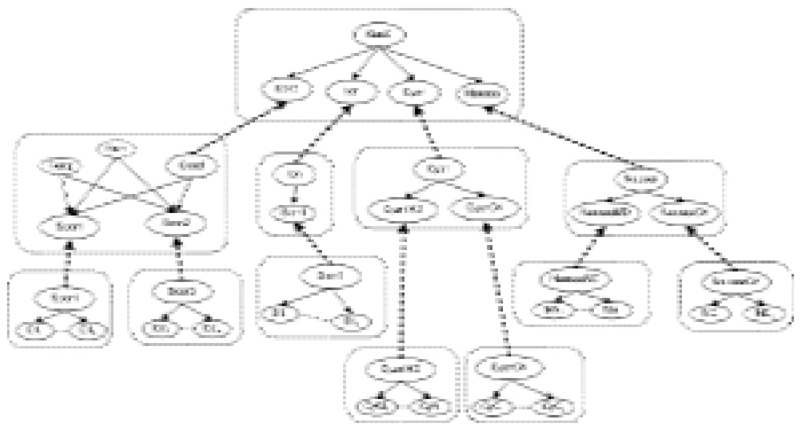


Figure 4.4: An example of the distributed `GasX` *and organization of local network fragments forming a DPN*

Because the network fragments are DAGs and can be described through BNs we have to assign a potential to every network fragment. This potential is defined over the variables (events) defined in the network fragment according to definition 4.4.

Definition 4.4 (DPN Local BN) *A local BN ψ_i is a tuple $\psi_i = (\mathcal{G}_i, \psi(\mathcal{V}_i))$ in agent A_i. A DPN local DAG G_i contains a single service root node $R \in \mathcal{V}_i$ which is an ancestor of the input concept nodes $\mathcal{L}_i \subset \mathcal{V}_i$. The potential $\phi(\mathcal{V}_i)$ of agent A_i is defined according to:*

$$\phi(\mathcal{V}_i) = \prod_{j}^{m} P(V_j|\pi(V_j)) \tag{4.8}$$

where $V_j \in \mathcal{V}_i$, m is the number of variables defined in the BN ψ_i and where the probability distribution of the service concept $R \in \mathcal{V}_i$ is uniform.

According to definition 4.4 only the service concept gets a uniform potential. Using a uniform potential for the roots will not influence the computation of the marginal posterior.

Furthermore we define the Distributed Perception Network as follows:

Definition 4.5 (Distributed Perception Network (DPN)) *A DPN Domain graph Ψ is defined as a quadruplet $D = (\mathcal{G}, \mathcal{V}, \mathcal{S}, \mathcal{P})$ where $G_i \in \mathcal{G}$ is a local DAG. $\mathcal{G}$ is the set of local DAGs. $\mathcal{V}$ are all the variables defined in a DPN domain graph and $Q_i \subseteq \mathcal{V}$ is called a cluster and defines a subset of variables. $Q_h \in \mathcal{V}$ is the cluster that contains the hypothesis where we want to reason about. $\mathcal{S}$ is the full separator set where every separator $S_i \in \mathcal{S}$ is defined according to definition 4.3. For every pair of separators $S_i \in \mathcal{S}$ and $S_j \in \mathcal{S}$ the intersection $S_i \cap S_j = \emptyset$ where $i \neq j$. In other words every separator is uniquely defined. $\mathcal{P}$ is the set of potentials defined over the full DAG and $P_i \in \mathcal{P}$ are the potentials defined according to definition 4.4 for a cluster Q_i.*

4.2.2 Organization Constraints

The assembled DPN structure must support correct belief propagation. Consequently, during self-organization the DPN agents must consider certain organization constraints:

Definition 4.6 (DPN Organization Constraints) *Given a DPN with a set of clusters $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$ that are directly or indirectly connected, a set of separators $\mathcal{S} = \{S_1, \ldots, S_n\}$ and a new cluster Q_i. Cluster Q_i can be connected to $Q_j \in \mathcal{Q}$ with separator $S\langle Q_i, Q_j\rangle$ if the following two rules are satisfied:*

i) $\exists! Q_j : (Q_i \cap Q_j) \neq \emptyset$

ii) $|Q_i \cap Q_j| = |S\langle Q_i, Q_j\rangle| = 1$

The first rule i) is the *intersection constraint* and states that there exist only one cluster $Q_j \in \mathcal{Q}$ where the newly added cluster Q_i has an intersection not equal to the empty set. This constraint implies that every separator $S\langle Q_i, Q_j\rangle \in \mathcal{S}$ is uniquely defined. The second rule ii) is the *separator constraint* and simply states that all separators should have size 1. Any cluster that is a candidate for assembling should satisfy both constraints otherwise it is rejected and not connected to the DPN.

These design constraints imply the following proposition:

Proposition 4.2 (Preserving Junction Tree) *If the assembly rules are such that a newly added cluster satisfies constraints in definition 4.6, the resulting distributed model will automatically correspond to a junction tree.*

Proof Given that a new cluster can only be connected to one other cluster (consequence of rule i) it can never introduce a non-degenerate cycle. Moreover, the new separator will be unique given the existing separators $\mathcal{S}$. This means that non-adjacent clusters always have an empty separator which would mean that we still have a JT, because the running intersection property is satisfied. In other words the JT property of the DPN domain graph is preserved. □

4.3 Self Organization Principles

When a query is issued by a DPN user, the DPN should be able to self organize the DPN agents, that is, establish connections between the relevant processing units in order to answer the query. In this section we will describe the procedure of self organization with two algorithms and show through an example how these algorithms work. We will start with an explanation of the CNET-protocol which is used in the organization algorithms.

4.3.1 CNET Protocol

Agent organization is facilitated by *contract net (CNET) protocol* procedure (see [19]). The CNET protocol procedure uses three types of messages:

- Call for proposal - a call for proposal message is sent by an agent A_i to all agents $\{A_j | 0 \leq j \leq n\}$ that can supply a service that is relevant for A_i.
- Bid - a bid message is sent from an agent A_j that received a call for proposal. For now, a bid is only used as a confirmation message that agent A_j is ready to supply information.
- Fusion contract - when agent A_i received a bid from an agent A_j the organization constraints in definition 4.6 are verified. When the constraints are satisfied a fusion contract is issued by agent A_i to agent A_j. From this point on agents A_i and A_j are connected and communication is possible.

To illustrate this procedure see figure 4.5 where two agents A_1 and A_2 are depicted. Agent A_1 needs a service from A_2 and issues a call for proposal to agent A_2 (message 1). When agent A_2 receives the call for proposal from agent A_1 it sends a bid (message 2) to A_1. When agent A_1 receives the bid it sends back a fusion contract (message 3) to agent A_2 if the organization rules given in definition 4.6 are satisfied.

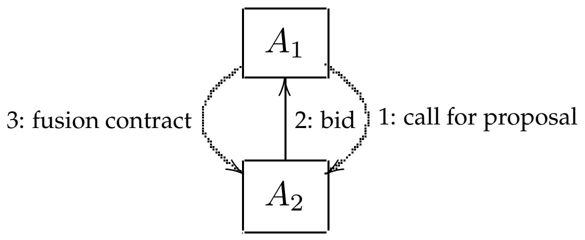


Figure 4.5: CNET protocol

4.3.2 Organization Algorithms

A DPN system can organize the local DAGs in a *top-down* or *bottom-up* manner such that we obtain valid DPN systems satisfying constraints from definition 4.6. Both ways of organization are *concept driven* which means that the input and output concepts of the local DAGs determine how the DPN can organize itself.

Algorithm 4.1: Top Down Network Configuration

```
procedure: TopDownConfiguration($X$)
input    : $X$ is a query concept
1 Find a set of agents $\mathcal{A}_q \subseteq \mathcal{A}$ such that $\forall A_i \in \mathcal{A}_q : R_i = X$;
2 foreach agent $A_i \in \mathcal{A}_q$ do
3     Use CNET Protocol to establish a fusion contract with $A_i$ by satisfying the
      organization constraints in definition 4.6;
4     if fusion contract with $A_i$ is established successfully then
5         foreach input concept $L_{i,j} \in \mathcal{L}_i$ from agent $A_i$ do
6             in agent $A_i$ call TopDownConfiguration($L_{i,j}$);
7         end
8     end
9 end
```

The formal description of the illustrated DPN top-down organization is given in algorithm 4.1.

The bottom-up organization procedure given in algorithm 4.2 is useful when a DPN is used as an alarming system. A single sensor observation can spawn several DPNs resulting in different hypothesis estimations. The spawned DPNs try to find as many relevant information sources as possible. This is to avoid wrong estimations when the DPN is spawned by a faulty sensor.

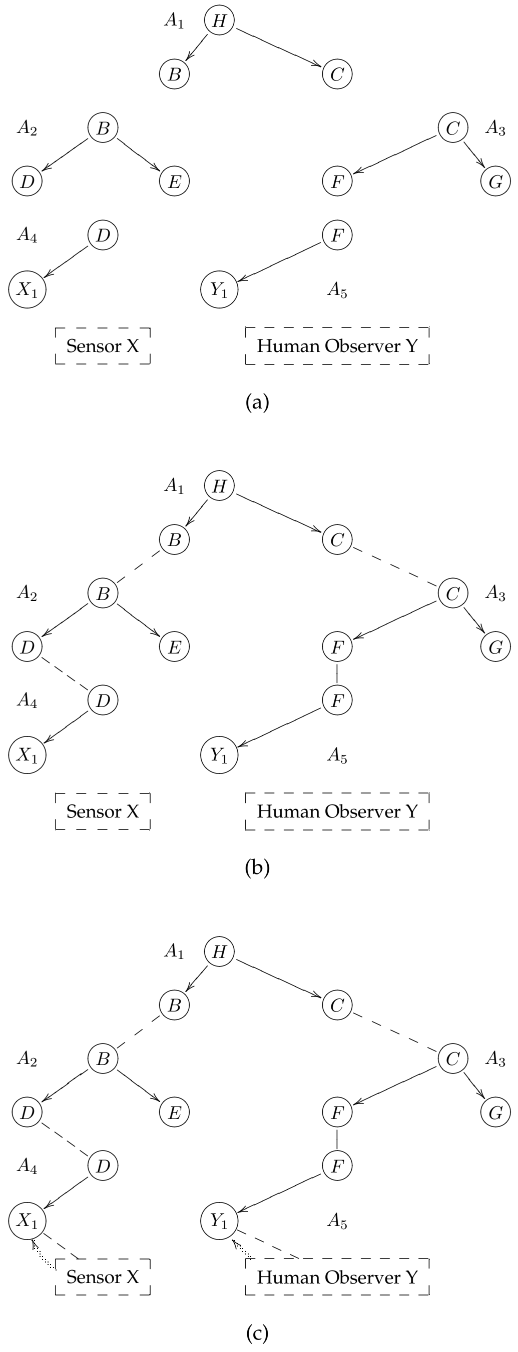
Algorithm 4.2: Bottom-Up Network Configuration

```
procedure: BottomUpConfiguration($R_k$)
input    : $R_k$ is the service concept of the caller agent $A_k$
1  Find a set of agents $\mathcal{A}_s \subseteq \mathcal{A}$ such that $\forall A_i \in \mathcal{A}_s : R_k \in \mathcal{L}_i$;
2  foreach agent $A_i \in \mathcal{A}_s$ do
3      Use CNET Protocol to establish a fusion contract between the caller agent
       and $A_i$ by satisfying the organization constraints in definition 4.6;
4      if Fusion contract with $A_i$ is established successfully then
5          foreach input concept $L_{i,j} \in \mathcal{L}_i$ from agent $A_i$ do
6              in agent $A_i$ call TopDownConfiguration($L_{i,j}$);
7          end
8          in agent $A_i$ call BottomUpConfiguration($R_i$);
9      end
10 end
```

In figure 4.6 an example is given of a top-down self organizing DPN following algorithm 4.1. In figure 4.6 (a) all agents, each having a local DAG, are initially disconnected and do not know the whereabouts of the other agents with relevant local DAGs. In this example we assume that all the agents have access to an information source with all the addresses of the agents available in the DPN, the services they need and the services they can supply.

A user might be interested in the belief of a concept H and a caller agent A_c calls `TopDownConfiguration(`H`)`. A_c will find the agent A_1. At this point, the system wants to know the values (soft evidence) for the concepts B and C and `TopDownConfiguration(`B`)` and `TopDownConfiguration(`C`)` are called on agent A_1. In figure 4.6 (b) we can see that a connection is established between agents A_1 with A_2 and A_3. The same process is repeated for agents A_2 and A_3 and they find agents A_4 and A_5, respectively. In agent A_2 there is a concept E for which no agent could be found that can deliver the soft evidence for concept E. A similar situation applies for the agent A_3 with concept G. These concepts do not influence the estimation of $P(H)$. It might happen that these information sources become available later in the information fusion process.

Figure 4.6: An example of self organization of a DPN



4.4 Distributed Inference through DPN Network Fragments

Relevant information sources for a DPN fusion task can become available during runtime. These new information sources can be found by the DPN system and integrated into the fusion structure. This means that the network topology is dynamic in the sense that network fragments can be added, removed or changed during the inference process. There are three types of modeling fragments: *static*, *dynamic* and *appendable*. These modeling fragments are the three basic building blocks for the DPN global inference structure.

In this section we will explain how inference is performed in the different types of modeling fragments.

4.4.1 Static fusion algorithm

Static modeling fragments are specified through an arbitrarily complex BN. The topology of a static modeling fragment remains unchanged during their lifetime in the inference process.

Definition 4.7 (Model of Static Fragment) *The model of a static modeling fragment in agent A_i is specified through a BN with DAG G_i that has a set of quasi-static variables $\mathcal{V}_i$. The model potential over variables $\mathcal{V}_i$ is defined according to definition 4.4. Only one of the root concepts $\mathcal{R}_i \subseteq \mathcal{V}_i$ of G_i can represent the service concept of agent A_i and its prior probability is uniform. The leaf concepts $\mathcal{L}_i \subseteq \mathcal{V}_i$ of agent A_i correspond to the input concepts.*

In figure 4.7 an example is given of a static network fragment G_i.

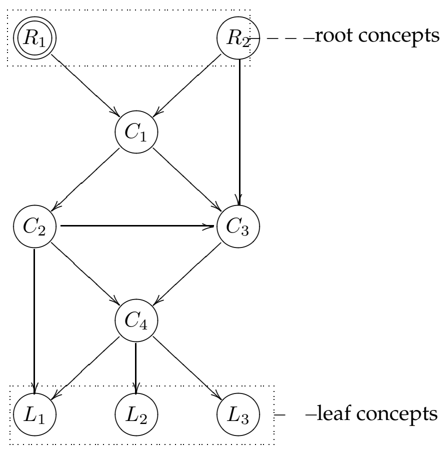


Figure 4.7: A DPN static network fragment G_i

This network fragment has the root concepts $\mathcal{R}_i = \{R_1, R_2\}$. However, only one root concept can be used as a service concept (denoted with a double circle) and may connect to another network fragment with the same input concept. The network fragment G_i of a static network fragment has also three input concepts $\mathcal{L}_i = \{L_1, L_2, L_3\}$. Leaf concepts, also called input concepts, may all connect to other network fragments with the same corresponding root concept (see the sharing condition in definition 4.2).

Belief updating in static modeling fragments is based on multiplication and division with separator potentials (see algorithm 4.3).

4.4.2 Dynamic fusion algorithm

A dynamic modeling fragment models sequences of observations, originating from one information source. In general, information sources are inherently noisy and by processing sequences of observations we can reduce the impact of observation noise. Given a quasi-static event, the sequences of resulting observations can be captured by a naive BN. However, this is not practical,

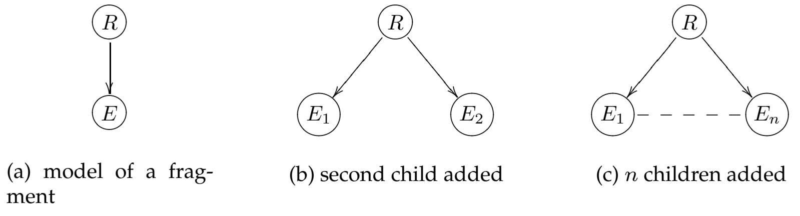
Algorithm 4.3: Static fusion algorithm

```
procedure: StaticFusion($\phi'(S\langle\mathcal{V}_i,\mathcal{V}_l\rangle)$)
input     : Separator potential $\phi'(S\langle\mathcal{V}_i,\mathcal{V}_l\rangle)$ over a leaf node (corresponding
            to soft evidence)
output    : Separator potential $\phi(S\langle\mathcal{V}_i,\mathcal{V}_j\rangle)$ or prior distribution $P(R)$ over
            the service root node
1 Update potential $\phi(\mathcal{V}_i)$ of graph $G_i$ with the received separator potential
  $\phi(S'\langle\mathcal{V}_i,\mathcal{V}_l\rangle)$ from agent $A_l$ with variables $\mathcal{V}_l$ with:

      $$\phi(\mathcal{V}_i) = \frac{\phi(\mathcal{V}_i)\phi'(S\langle\mathcal{V}_i,\mathcal{V}_l\rangle)}{\phi(S\langle\mathcal{V}_i,\mathcal{V}_l\rangle)} \quad (4.9)$$

  where $\phi(S\langle\mathcal{V}_i,\mathcal{V}_l\rangle)$ is the old separator potential;
2 Set $\phi(S\langle\mathcal{V}_i,\mathcal{V}_l\rangle) \leftarrow \phi'(S\langle\mathcal{V}_i,\mathcal{V}_l\rangle)$;
3 compute $\phi(R) = \phi(\mathcal{V}_i)^{\downarrow R}$ where $R \in \mathcal{V}_i$ is the root node;
4 if some agent $A_j$ has a local DAG $G_j$ with a child node $L \in \mathcal{V}_j$, such that $L = R$[‡]
  then
5     send $\phi(S\langle\mathcal{V}_i,\mathcal{V}_j\rangle) \leftarrow \phi(R)$ to $A_j$;
6 else
7     compute the hypothesis belief $P(R) = \alpha \cdot \phi(R)$ ;
8 end
```

since for each new observation we have to change the DAG of the local BN. Instead, we can obtain identical results by using a simple belief updating algorithm that makes use of a BN consisting of two nodes.

Definition 4.8 (Model of Dynamic Fragment) *The model of a dynamic modeling fragment is specified as a local BN with a DAG G_i that contains variables $\mathcal{V}_i$ in agent A_i. The local BN has a potential $\phi(\mathcal{V}_i)$ according to definition 4.4. G_i has one quasi-static root node $R \in \mathcal{V}_i$ corresponding to the service concept of A_i where the prior probability is initially set uniform, and one dynamic child node $E \in \mathcal{V}_i$ corresponding to the input concept, which is instantiated with hard evidence. The CPT connecting the two nodes captures observation noise.*

Figure 4.8:



(a) model of a fragment (b) second child added (c) n children added

We first explain principles of the dynamic fusion algorithm with the help of an example. Let's assume that according to definition 4.8 the local model is specified through a BN with a simple topology consisting of a single root node R and a single child node E (see figure 4.8 (a)). These nodes are related through a CPT $P(E|R)$ and the prior probability of the root node is initially uniform. When a new observation, encoded through the potential $\phi(e_E)$ corresponding to hard evidence (see definition 2.6), is received from a sensor or a human observer we compute posterior probability $P(R|e_E)$

$$P(R|e_E) \quad = \quad \alpha \cdot P(R)P(E|R)\phi(e_E), \tag{4.10}$$

‡ i.e. agent A_i and agent A_j have the DPN separator $S\langle\mathcal{V}_i,\mathcal{V}_j\rangle$

where α is a normalization constant. The computed posterior is used as a prior in the next computation step.

$$P(R) \quad \leftarrow \quad P(R|e_E) \tag{4.11}$$

When a new observation is received from the same sensor or human observer we can use the updated prior $P(R)$ and the same CPT $P(E|R)$ to compute the new posterior $P(R|e_{E_1}, e_{E_2})$ and update the prior $P(R)$ again ($E_1 = E$). Now the prior $P(R)$ corresponds to $P(R|e_{E_1}, e_{E_2})$ which is identical to:

$$P(R|e_{E_1}, e_{E_2}) = \alpha \cdot P(R)P(E|R)P(E|R)\phi(e_{E_1})\phi(e_{E_2}) \tag{4.12}$$

corresponding to the network in figure 4.8 (b). This process can be repeated for n updates where we will get an equation that corresponds to the network in figure 4.8 (c). In other words, by updating the prior we do not have to specify the actual network topology, but we can simulate it. This means that the dynamic modeling fragment is using an algorithm that manipulates the simple BN in figure 4.8 (a) in such a way that the fusion results are equivalent to inference with the BN in figure 4.8 (c).

Algorithm 4.4: Dynamic fusion algorithm

```
procedure: DynamicFusion($\phi(e_{E_k})$)
input    : Hard evidence encoded in $\phi(e_{E_k})$
output   : Separator potential $\phi_{S\langle\mathcal{V}_i,\mathcal{V}_j\rangle}$
1 compute potential $\phi(\mathcal{V}_i)$ of local DAG $G_i$ with:
      $\phi(\mathcal{V}_i) = P(R)P(E_k|R)\phi(e_{E_k})$   (4.13)
2 compute the service concept potential $\phi(R)$ with:
      $\phi(R) = \phi(\mathcal{V}_i)^{\downarrow R}$   (4.14)
3 update prior probability $P(R)$ of local DAG $G_i$ with:
      $P(R) \leftarrow \alpha \cdot \phi(R)$   (4.15)
4 if some agent $A_j$ has a local DAG $G_j$ with a child node $L \in \mathcal{V}_j$, such that $L = R$[§]
  then
5     send $\phi(S\langle\mathcal{V}_i, \mathcal{V}_j\rangle) \leftarrow \phi(R)$ to $A_j$;
6     compute the hypothesis belief $P(R) = \alpha \cdot \phi(R)$ ;
7 end
```

4.4.3 Appendable fusion algorithm

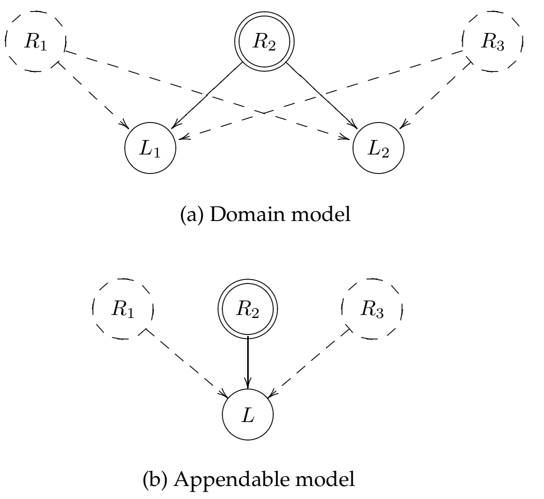
Appendable fragments support incorporation of information sources that provide identical services at runtime.

Definition 4.9 (Model of Appendable Fragment) *The model of an appendable modeling fragment in agent A_i is specified as a local BN with a DAG G_i that has a set of quasi-static variables $\mathcal{V}_i$. The model potential over variables $\mathcal{V}_i$ is defined according to definition 4.4. One root node $R_i \in \mathcal{V}_i$ corresponds to the service concept of agent A_i where the prior probability is uniform. There can exist an arbitrary set of additional root nodes $(\mathcal{R}\backslash R_i) \subset \mathcal{V}_i$ which are all instantiated (which we call environment variables). Graph G_i can only have one leaf node $L \subset \mathcal{V}_i$ which corresponds to an information source.*

[§] i.e. agent A_i and agent A_j have the DPN separator $S\langle\mathcal{V}_i, \mathcal{V}_j\rangle$)

We first explain principles of the appendable fusion algorithm with the help of an example. Let's assume the network in figure 4.9 (a) that has a service concept R_2 (denoted by a double circle), optional environment variables R_1 and R_3 (denoted as dashed circles) and the input concepts L_1 and L_2 corresponding to two information sources providing the same type of information.

Figure 4.9:



(a) Domain model

(b) Appendable model

We assume that the environment variables R_1 and R_3 are instantiated with hard evidence e_{R_1} and e_{R_3}, respectively. The input variables L_1 and L_2 are associated with soft evidence $\overline{e}_{L_1}$ and $\overline{e}_{L_2}$, respectively. Given the evidence and the BN from figure 4.9 (a) we obtain the following factorization of posterior $P(R_2|e_{R_1}, e_{R_3}, \overline{e}_{L_1}, \overline{e}_{L_2})$:

$$\begin{aligned} P(R_2|e_{R_1}, e_{R_3}, \overline{e}_{L_1}, \overline{e}_{L_2}) &= \alpha \cdot P(R_2)\phi(e_{R_1})\phi(e_{R_3}) \sum_{L_1} P(L_1|R_1, R_2, R_3)\phi(\overline{e}_{L_1}) \\ &\cdot \sum_{L_2} P(L_2|R_1, R_2, R_3)\phi(\overline{e}_{L_2}), \end{aligned} \tag{4.16}$$

where α denotes a normalization constant. In this factorization we see that the input variables L_1 and L_2 are conditionally independent given the variable R_2 which is due to the fact environment variables R_1 and R_3 are instantiated with hard evidence. Because of conditional independence we can compute the potential over the service concept by using a simpler network depicted in figure 4.9 (b). That is, for each soft evidence $\overline{e}_{L_i}$ we can use this network to compute the joint probability $P(R_2, e_{R_1}, e_{R_3}, \overline{e}_{L_i}) = P(R_2)\phi(e_{R_1})\phi(e_{R_3}) \sum_{L_1} P(L_1|R_1, R_2, R_3)\phi(\overline{e}_{L_i})$ over the root variable R_2.

Because $\phi(e_{R_1})$ and $\phi(e_{R_3})$ are hard evidence and $P(R_2)$ is a uniform prior distribution we can easily verify that the following equation holds:

$$P(R_2|e_{R_1}, e_{R_3}, \overline{e}_{L_1}, \overline{e}_{L_2}) = \alpha' \cdot P(R_2, e_{R_1}, e_{R_3}, \overline{e}_{L_1})P(R_2, e_{R_1}, e_{R_3}, \overline{e}_{L_2}), \tag{4.17}$$

where α' denotes a normalization constant. We can extend this approach to arbitrarily many leaf nodes and instantiated roots (environment nodes):

$$\begin{gathered} P(R_s|e_{R_1}, \ldots, e_{R_m}, \overline{e}_{L_i}, \ldots, \overline{e}_{L_n}) = \\ \alpha' \cdot \prod_i^n P(R_s) \prod_k^m \phi(e_{R_k}) \sum_{L_i} P(L_i|R_s, R_1, \ldots, R_m)\phi(\overline{e}_{L_i}) \end{gathered} \tag{4.18}$$

Note that repeated multiplications with the service root probability $P(R_s)$ and hard evidence potentials over environment variables $\phi(e_{R_k})$ in this expression will not result in an incorrect computation of the posterior distribution

$P(R_s|e_{R_1},\ldots,e_{R_m},\overline{e}_{L_i},\ldots,\overline{e}_{L_n})$. Namely, through normalization the uniform prior $P(R_s)$ and instantiated environment variables do not affect the posterior distribution $P(R_s|e_{R_1},\ldots,e_{R_m},\overline{e}_{L_i},\ldots,\overline{e}_{L_n})$.

Algorithm 4.5: Appendable fusion algorithm

```
procedure: AppendableFusion($\phi'(S\langle\mathcal{V}_i,\mathcal{V}_k\rangle)$)
input     : Separator potential $\phi'(S\langle\mathcal{V}_i,\mathcal{V}_k\rangle)$ over a leaf node (corresponding
            to soft evidence)
output    : Separator potential $\phi(S\langle\mathcal{V}_i,\mathcal{V}_j\rangle)$ or prior distribution $P(R)$ over
            the service root node
1  Compute:
```

$$\phi'_k(R_s) \leftarrow \left(P(R_s)\prod_p^n \phi(e_{R_p})\sum_{L_k} P(L_k|R_s,R_1,\ldots,R_n)\phi'(S\langle\mathcal{V}_i,\mathcal{V}_k\rangle)\right)^{\downarrow R_s} \tag{4.19}$$

```
   if $S\langle\mathcal{V}_i,\mathcal{V}_k\rangle)$ is from a new information source then
2      $\phi_\Psi(R_s) \leftarrow \phi_\Psi(R_s)\phi'_k(R_s)$;
3      Append uniform potential $\phi_k(R_s)$ to a list of separator potentials of all
       leaf nodes;
4  else
5      $\phi_\Psi(R_s) \leftarrow \frac{\phi_\Psi(R_s)\phi'_k(R_s)}{\phi_k(R_s)}$;
6  end
7  Set $\phi_k(R_s) \leftarrow \phi'_k(R_s)$;
8  if some agent $A_j$ has a local DAG $G_j$ with a child node $L \in \mathcal{V}_j$, such that $L = R_s$¶
   then
9      send $\phi(S\langle\mathcal{V}_i,\mathcal{V}_j\rangle) \leftarrow \phi_\Psi(R_s)$ to $A_j$;
10 else
11     compute the hypothesis belief $P(R_s) = \alpha \cdot \phi_\Psi(R_s)$ ;
12 end
```

4.4.4 Distributed Inference Process

The algorithms 4.3, 4.4 and 4.5 can be used to compute the posterior $P(H|\mathcal{E})$ in a bottom-up manner. This means that the distribution over the service concept of agent A_i is sent to agent A_j if this agent has an input concept identical to the the service concept of agent A_i. Due to the definition of a local BN (see definition 4.4) the service concept of A_j is ancestor of the service concept of agent A_i.

In a DPN we are only interested in one hypothesis contained in some agent A_i which means that there is no need to compute the full joint system potential (JSP) over all variables defined in a DPN. Computation of the JSP would require global consistency of all clusters. Avoiding global consistency considerably simplifies the DPN inference algorithm. By only collecting evidence to agent A_i from adjacent modeling fragments it is possible to calculate the correct hypothesis posterior based on the instantiated evidence into the DPN. Note, in DPN only the agents with direct access to information sources, such as sensors, can have dynamic modeling fragments.

For example, in figure 4.4 the sensor and human observations are directly processed by the agents containing dynamic modeling fragments (see, for example, the agents on the bottom of figure 4.4). These agents send partial beliefs to agents with static and appendable modeling fragments. For example agents with service concepts `Scon1` and `Scon2` send their partial fusion results to an agent with an appendable modeling fragment. This modeling fragment considers observed environment variables `Temp` and `Hum` and sends a partial

¶ i.e. agent A_i and agent A_j have the DPN separator $S\langle\mathcal{V}_i,\mathcal{V}_j\rangle$

belief over variable Cond to an agent with a static modeling fragment, which computes posterior probability distribution $P(\texttt{GasX}|\mathcal{E})$.

After the full evidence set $\mathcal{E}$ is propagated through a system of fragments in a DPN the computed marginal posterior $P(\texttt{GasX}|\mathcal{E})$ will reflect entire evidence set $\mathcal{E}$ correctly.

Proposition 4.3 (Correct Posterior Calculation) *Let us assume a DPN Ψ that computes belief over hypothesis variable H and set of evidence $\mathcal{E}$ that is used for the instantiation of input variables in different agents of DPN Ψ. If the dynamic modeling fragments are used only in agents that have direct access to subsets of evidence in $\mathcal{E}$, then after execution of algorithms 4.3, 4.4 and 4.5 on different agents will result in a posterior $P(H|\mathcal{E})$ which will correctly reflect the entire evidence set $\mathcal{E}$. This distributed process is equivalent to exact belief propagation in the monolithic BN that captures the complete causal process.*

Proof (**sketch**) If we investigate the relations between the BN graphical models and the corresponding factorizations we can observe that the graphical model, modeling a sequence of events, corresponds to a nested factorization. Every factor is defined by a summation over factors. For every factor in this summation we can again identify another nested factorization, etc. Therefore, the innermost factors will correspond to the leaf nodes of the sequence of modeled events where evidence is instantiated. The inference is based on the reverse causal direction between local BN and therefore the agents automatically combine different factors, that will correspond to partial fusion results, in such an order that the nested factorization is correct. In other words the encoded conditional independence given the service concept in the graphical model is directly related to the factorization order. □

In the following example we show that by combining different modeling fragments described in sections 4.4.1, 4.4.2 and 4.4.3, we obtain fusion systems that guarantee:

- Posterior distribution over a hypothesis variable, which correctly reflects the entire evidence set that has been injected through different parts (i.e. agents) of a DPN system.
- Exact belief propagation without compilation of global fusion structures. Only local models are precompiled independently.
- Fusion results are correct without using any synchronization of partial fusion processes.

We show that by running an implemented DPN system in a simulation environment the designed DPN system can detect presence of a toxic gas GasX (see section 4.1 for more details about this example). In order to get data for the model we simulated the sensor and human reports by following the procedure described in section D in the appendix. We assumed that toxic gas GasX was released and that the DPN, consisted of a system of agents shown in figure 4.4, could detect this gas.

The outputs of the data generation system were used as inputs to the DPN system. In figure 4.10 we can see the results of the detection of GasX.

The experiments show that, given a certain evidence set $\mathcal{E}$, the fusion result $P(\texttt{GasX}|\mathcal{E})$ supplied by this system of agents was identical to the fusion result of the equivalent monolithic BN, shown in figure 4.1. In addition, we ran several experiments with different sequences of the same observation set in order to show that the fusion results are correct, despite the fact that no synchronization of partial fusion processes is used. By using the same set of observations, we generated different sequences of evidence node instantiations in different agents. Independently of the sequence, the distributed inference always converged to the correct $P(\texttt{GasX}|\mathcal{E})$, i.e. distribution which was identical to $P(\texttt{GasX}|\mathcal{E})$ obtained with the monolithic model. This is a convenient property of a DPN, because we never know in which order the observations will be processed.

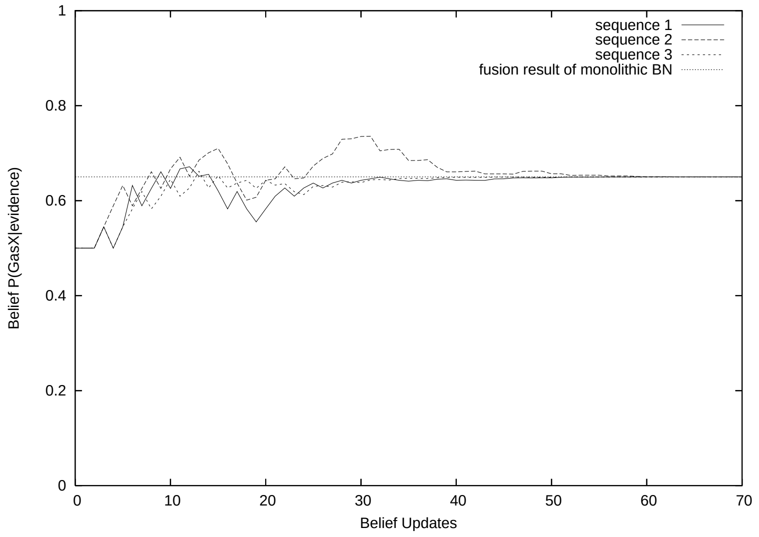


Figure 4.10: The belief $P(R|\mathcal{E})$ computed for three evidence sequences in different orders

Figure 4.11 shows how the DPN agents are organized using self organization principles described in section 4.3. The arrows indicate the organization direction, which can be top-down or bottom-up. Because the data was generated for the observation nodes the associated DPN agents initiated `BottomUp-Configuration` described in algorithm 4.2. Therefore, the DPN agent organization was bottom-up and the arrows point away from the nodes that correspond to information sources (like `NH-nausea-CV`, `NH-cyanosis-CV`, etc).

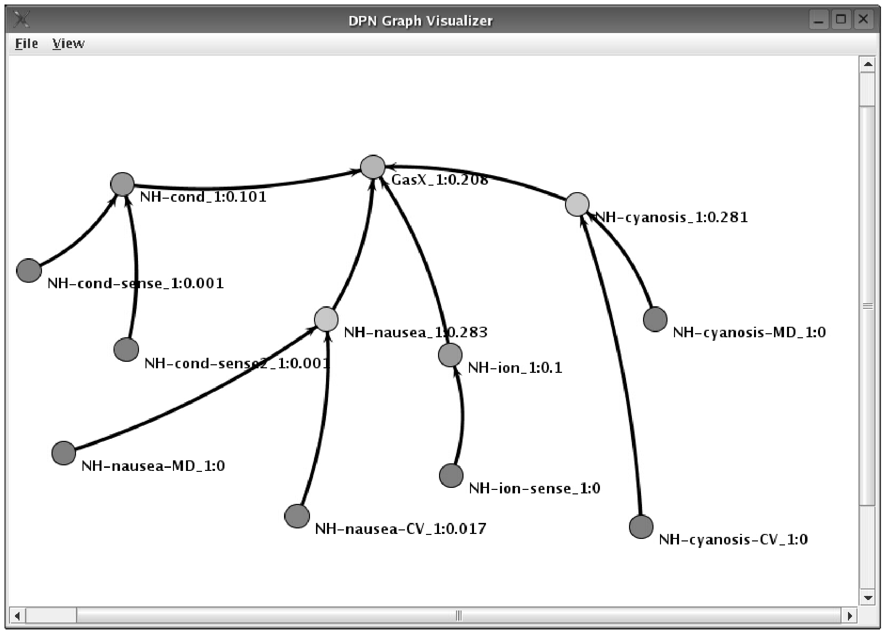


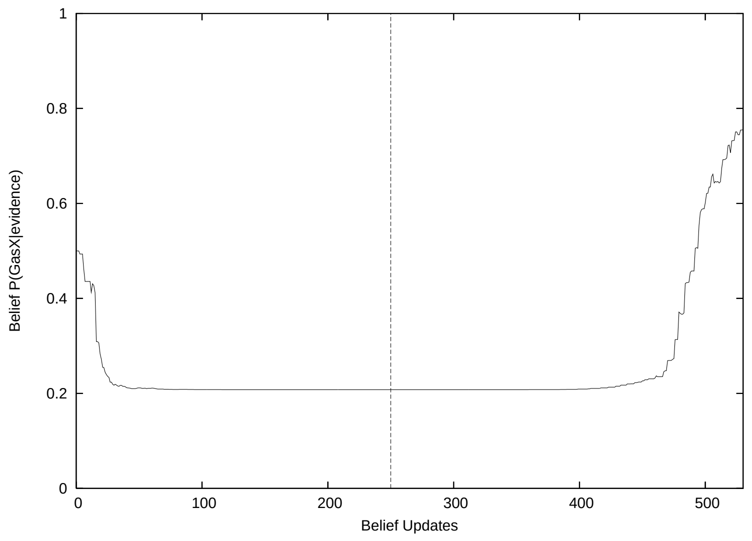
Figure 4.11: Organization of DPN agents

4.4.5 Resetting

A continuous running DPN system has the unfortunate property of becoming irresponsive when the computed posterior beliefs are converged. In this situation it is difficult to detect unpremeditated changes in the environment efficiently. To illustrate the problem we are going to use the example described in section 4.1 again. Assume that we want to measure the presence of `GasX` in a populated area. In this example we first set the ground truth of hypothesis `GasX` to false and generate 250 observations for the sensor and human observation nodes (like `C1`, `C2`, `S1`, etc.) in total (following the procedure in appendix D). When all simulated observations are processed by the DPN system we change the ground truth of hypothesis `GasX` to true (in other words, there is a toxic gas present in the populated area) and we generate 280 observations for the sensor and human observation nodes in total. By changing the ground truth of the presence of `GasX` we can simulate a changing environment, during execution of the DPN system.

In figure 4.12 we can see that after 250 updates the converged belief of $P(\mathtt{GasX}|\mathcal{E})$ is irresponsive to changes in the environment. Eventually, the presence of `GasX` is noticed by the system, but requires an unacceptable amount of time.

Figure 4.12: The computed posterior belief $P(\mathtt{GasX}|\mathcal{E})$ where the ground truth of `GasX` was changed after approximately 250 belief updates



In order to reduce the detection time we can use a straightforward solution of resetting the values associated with the local network fragment (for example, by setting the prior probability $P(R)$ uniform in algorithm 4.4) of an agent every Δt seconds. With resetting we can go back to the initial situation where the DPN system did not perform any fusion yet.

The DPN resetting algorithm can be initiated with `IterativeReset` described in algorithm 4.6.

Algorithm 4.6: Iterative resetting of a DPN on agent A_i

```
procedure: IterativeReset
1 if IterativeResetMessage is received from caller agent $A_k$ then
2     Get list of agents $\mathcal{A}_c$ where the intersection of the input concepts $\mathcal{L}_i$ of
      agent $A_i$ with service concept $R_j$ of agent $A_j$ is non empty;
3     foreach agent $A_j \in \mathcal{A}_c$ do
4         Set BlockFusion($A_j$) = true ;
5         Send IterativeResetMessage to $A_j$;
6     end
7     Reset values associated with local BN;
8     Send ResetConfirmMessage to $A_k$;
9 end
```

Let's assume that agent A_k has sent an `IterativeResetMessage` to agent A_i (see message 1 in figure 4.13) after blocking partial fusion result messages from agent A_i. Consequently, agent A_i will call `IterativeReset` on itself. The `IterativeReset` algorithm will first get a list of connected agents $\mathcal{A}_c$ for which the following condition is true: $\mathcal{L}_i \cap R_j \neq \emptyset$ (see also sharing condition in definition 4.2). For each agent $A_j \in \mathcal{A}_c$ the fusion messages containing partial fusion results will be blocked and an `IterativeResetMessage` is sent (see message 2 and 3 in figure 4.13). Next, the algorithm will reset the values that are associated with the local BN in agent A_i. After all values are reset a `ResetConfirmMessage` will be sent back to the calling agent A_k (see message 4 in figure 4.13).

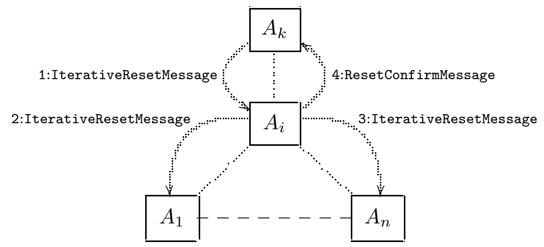


Figure 4.13: Example of the message sequence of the resetting algorithm

When agent A_k received a `ResetConfirmMessage` it will call `ResetConfirm` (see algorithm 4.7) on itself. This will unblock partial fusion result messages from agent A_i. Consistent reasoning is ensured in agent A_k, because partial fusion result messages from agent A_i are based on values of the reseted BN and not on old fusion values.

Algorithm 4.7: Reset confirmation on agent A_k

```
procedure: ResetConfirm
1 if ResetConfirmMessage is received from caller agent $A_i$ then
2     Set BlockFusion($A_i$) = false;
3 end
```

To see the effect on the detection time we are going to perform the same experiment on the `GasX` example by using the resetting algorithm (see algorithm 4.6). The resetting algorithm will be executed every 60 seconds on the DPN network. In figure 4.14 the results are presented. In this figure we can see that the change in environment is detected much faster than in the case where no resetting was used. This illustrates that resetting is an indispensable tool when changes in the environment need to be detected efficiently.

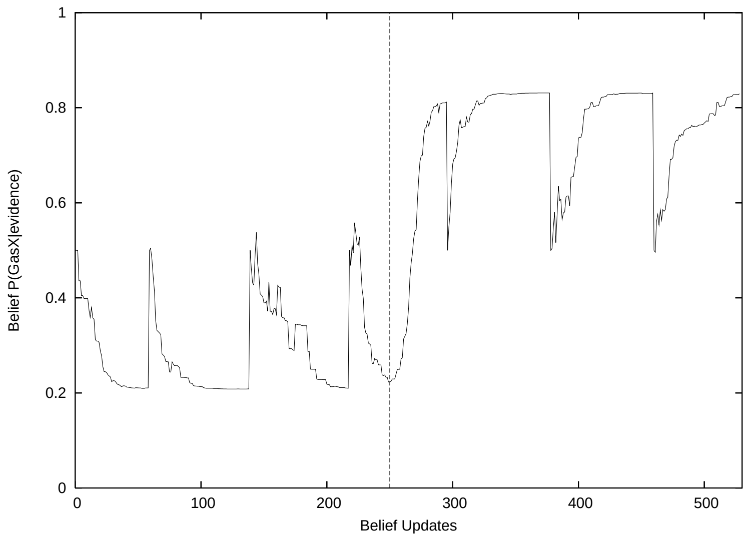


Figure 4.14: The computed posterior belief $P(\texttt{GasX}|\mathcal{E})$ using resetting. The ground truth of the environment is changed after approximately 250 belief updates

5 Alternative Approaches to Distributed Probabilistic Inference

BESIDE the DPN, there exist other approaches to probabilistic inference in distributed systems. In this thesis we compare DPNs with two well known approaches, namely:

- Multiply Sectioned Bayesian Networks (MSBNs) ([21, 20]);
- Prior/Likelihood Decomposable Models (PLDMs) (see [10, 11, 14]).

In this chapter, we briefly describe the basic principles of the two approaches.

5.1 Multiply Sectioned Bayesian Networks (MSBNs)

In this section MSBNs are presented. MSBNs are an extension of BNs for flexible modeling of complex domains. By using MSBNs complex domains can be modeled in a distributed way, such that domain knowledge and inference is distributed and reliable inference is ensured.

5.1.1 Architecture

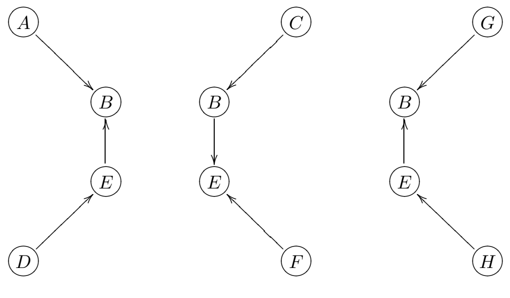
The MSBN architecture consists of local BNs that are connected through interfaces*. An interface exists if two local BNs describe the same variables. These variables describe the domain of the interface. In other words, a subgraph G_i and another subgraph G_j with variables $\mathcal{V}_i$ and $\mathcal{V}_j$, respectively, have an interface if the following holds: $\mathcal{V}_i \cap \mathcal{V}_j \neq \emptyset$. However, next to the interface requirement two graphs also have to be *graph consistent*.

Definition 5.1 (Graph Consistent) *Two subgraphs G_i and G_j are* **graph consistent** *if the subgraphs of G_i and G_j spanned by $\mathcal{V}_i \cap \mathcal{V}_j$ are identical.*

In figure 5.1 we can see that graph G_i and graph G_j are not graph consistent, because in graph G_i we have the subgraph $B \leftarrow E$ and in G_j the subgraph $B \rightarrow E$ spanned by $\mathcal{V}_i \cap \mathcal{V}_j$. Clearly, these two subgraph are not identical. Subgraphs G_i and G_k are graph consistent. Definition 5.1 also applies to undirected graphs.

* Interfaces are comparable with separators

Figure 5.1: Graph consistency



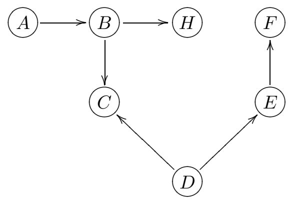
(a) G_i (b) G_j (c) G_k

If two graphs are graph consistent we can take the *union* of these graphs.

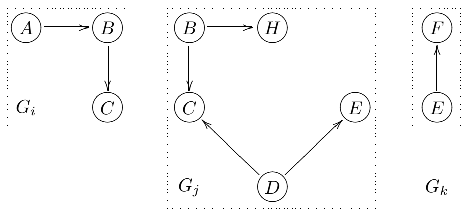
Definition 5.2 (Union) *Given two consistent graphs $G_i = (\mathcal{V}_i, \mathcal{E}_i)$ and $G_j = (\mathcal{V}_j, \mathcal{E}_j)$, the* **union** *is defined by $G = (\mathcal{V}_i \cup \mathcal{V}_j, \mathcal{E}_i \cup \mathcal{E}_j)$ and denoted by $G = G_i \sqcup G_j$.*

In figure 5.2 (b) we can see the subgraphs G_i, G_j and G_k of the graph G in figure 5.2 (a). Notice that the subgraphs are all graph consistent and the union will result in graph G. We can also say that graph G is *sectioned* into the subgraphs G_i, G_j and G_k.

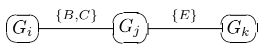
Figure 5.2: The graph G in (a) can be sectioned into the subgraphs given in (b) where the hypertree of G is given in (c)



(a) Graph G



(b) Subgraphs of G



(c) Hypertree of G

When two subgraphs are connected we not only have to make sure that they are graph consistent but also that the distributed model is an I-map (see definition 3.4). The interfaces should induce conditional independence, that is, $I(\mathcal{X}, \mathcal{Z}, \mathcal{Y})$, the interface variables should graphically separate $\mathcal{X}$ and $\mathcal{Y}$ in the DAG union. Which means that we will have $\langle \mathcal{X}, \mathcal{Z}, \mathcal{Y} \rangle$ and that the variables in the interface must be a *d-sepset* (where every variable is a *d-sepnode*).

Definition 5.3 (D-Sepset) *Let $G_i = (\mathcal{V}_i, \mathcal{E}_i)$ and $G_j = (\mathcal{V}_j, \mathcal{E}_j)$ be two DAGs such that $G = G_i \sqcup G_j$ is a DAG. A node X in the interface $I\langle \mathcal{V}_i, \mathcal{V}_j \rangle = \mathcal{V}_i \cap \mathcal{V}_j$ with parents $\pi(X)$ in G is a* **d-sepnode** *between G_i and G_j if either $\pi(X) \subseteq \mathcal{V}_i$ or $\pi(X) \subseteq V_j$. If every node in $I\langle \mathcal{V}_i, \mathcal{V}_j \rangle$ is a d-sepnode, then $I\langle \mathcal{V}_i, \mathcal{V}_j \rangle$ is a* **d-sepset**

If we return to the example in figure 5.2 (b) we see that there are two interfaces $I\langle \mathcal{V}_i, \mathcal{V}_j \rangle = \{B, C\}$ and $I\langle \mathcal{V}_j, \mathcal{V}_k \rangle = \{E\}$, where $\mathcal{V}_i$, $\mathcal{V}_j$ and $\mathcal{V}_k$ are the variables defined in the subgraphs G_i, G_j and G_k, respectively. To ensure that graph G is an I-map the interfaces $I\langle \mathcal{V}_i, \mathcal{V}_j \rangle$ and $I\langle \mathcal{V}_j, \mathcal{V}_k \rangle$ have to be d-sepsets. Variable $B \in I\langle \mathcal{V}_i, \mathcal{V}_j \rangle$ has the parent set $\pi(B) \subset \mathcal{V}_i$ and is completely contained in graph G_i. Variable $C \in I\langle \mathcal{V}_i, \mathcal{V}_j \rangle$ has its parent set $\pi(C) \subset \mathcal{V}_j$ fully contained in subgraph G_j. This means that all the variables in $I\langle \mathcal{V}_i, \mathcal{V}_j \rangle$ are d-sepnodes and the interface $I\langle \mathcal{V}_i, \mathcal{V}_j \rangle$ is a d-sepset. Also the interface $I\langle \mathcal{V}_j, \mathcal{V}_k \rangle$ is a d-sepset, because the only variable defined in this interface has its parent set $\pi(E) \subset V_j$ completely contained in subgraph G_j.

Given definition 5.3 we can state the following proposition:

Proposition 5.1 *Let $G_i = (\mathcal{V}_i, \mathcal{E}_i)$ and $G_j = (\mathcal{V}_j, \mathcal{E}_j)$ be two DAGs such that $G = G_i \sqcup G_j$ is a DAG, then $\mathcal{V}_i \backslash \mathcal{V}_j$ and $\mathcal{V}_j \backslash \mathcal{V}_i$ are d-separated by the interface $I\langle \mathcal{V}_i, \mathcal{V}_j \rangle = \mathcal{V}_i \cap \mathcal{V}_j$ if and only if $I\langle \mathcal{V}_i, \mathcal{V}_j \rangle$ is a d-sepset.*

For a proof consult [21] pag. 127.

The requirement on the variables of the interfaces dictates how the multi-agent dependence structure is formed. The *hypertree* is another requirement that is necessary to allow consistent probabilistic reasoning.

Definition 5.4 (Hypertree) *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}) = \bigsqcup_i G_i$ be a graph sectioned into subgraphs $G_i = (\mathcal{V}_i, \mathcal{E}_i)$. If the subgraphs can be organized as a connected tree Ψ, where each node is labeled by a G_i and each edge between G_k and G_m is labeled by $V_k \cap V_m$, such that for each i and j, $\mathcal{V}_i \cap \mathcal{V}_j$ is contained in each subgraph on the path between G_i and G_j in Ψ (running intersection property), then Ψ is a* **hypertree** *over $\mathcal{G}$. Each G_i is a* **hypernode** *and each interface is a* **hyperlink**.

The graph in figure 5.2 (c) is a hypertree Ψ over the graph G defined in figure 5.2 (a). Note that the intersection between $\mathcal{V}_i$ from G_i and $\mathcal{V}_k$ from G_k is empty and that means the running intersection property is satisfied, hence Ψ is a hypertree. The hypernodes in the hypertree G can also be seen as clusters of a JT only these clusters do not have to contain cliques.

Given the hypertree definition we can define the *hypertree multiply sectioned DAG* or simply *hypertree MSDAG*.

Definition 5.5 (Hypertree MSDAG) *$\mathcal{G} = \bigsqcup_i G_i$, where each $G_i = (V_i, E_i)$ is a DAG, is a* **hypertree MSDAG** *if and only if (1) there exists a hypertree Ψ over $\mathcal{G}$, and (2) each link in Ψ is a d-sepset.*

The hypertree MSDAG is the basic building block for a *Multiply Sectioned Bayesian Network (MSBN)*.

Definition 5.6 (MSBN) *A MSBN is defined over a set of variables $\mathcal{V}$ called the domain, and consists of a set of agents $\mathcal{A}$. Each agent $A_i \in \mathcal{A}$ has knowledge of a subdomain $\mathcal{V}_i$ such that $\mathcal{V} = \bigcup_i \mathcal{V}_i$, and is structured by a DAG $\mathcal{G}$. The complete MSBN is structured as a hypertree MSDAG $\Lambda = \bigsqcup_i G_i$, where each node is labeled by G_i. Furthermore, each agent defines a joint probability table (JPT) P_i over $\mathcal{V}_i$, such that $P_i = \prod_{X \in \mathcal{V}_i} P(X|\pi(X))$, where $\pi(X)$ are the parents of X in A_i. For each shared variable X, only one agent which contains $\pi(X)$ specifies the correct CPD $P(X|\pi(X))$, and all other agents specify a uniform CPD. The JPD over $\mathcal{V}$ is defined as $\mathcal{P} = \prod_i P_i$.*

5.1.2 Compilation

Before inference in a hypertree MSDAG is possible the structure has to be compiled into a *Linked Junction Forest (LJF)* with *Linkage Trees (LT)* as communication channels before belief updating through concise message passing is possible (see [21]).

Definition 5.7 (Linkage Tree (LT)) *Let G be a subgraph in a hypertree MSDAG, I be the d-sepset between G and an adjacent subgraph and T be a JT converted from G. Repeat the following procedure in T until no removal is possible:*

1. *Remove $X \in I$ if X is contained in a unique cluster Q_i;*
2. *After removal, if Q_i becomes a subset of an adjacent cluster Q_j merge Q_i into Q_j.*

Let L be the resultant cluster graph then L is a **linkage tree** *of T with respect to I if*

$$\bigcup_{Q \in L} Q = I \tag{5.1}$$

where each cluster Q in L is called a **linkage**. *A cluster in T that contains Q is called the* **linkage host** *of Q.*

Definition 5.8 (Linked Junction Forest (LJF)) *A LJF Φ is a tuple $(\mathcal{V}, \mathcal{G}, \mathcal{T}, \mathcal{L})$. $\mathcal{V} = \bigcup_i \mathcal{V}_i$ is the total universe where each $\mathcal{V}_i$ is a set of variables. $\mathcal{G} = \bigcup_i G_i$ where each $G_i = (\mathcal{V}_i, \mathcal{E}_i)$ is a chordal graph*† *such that there exists a hypertree Ψ over $\mathcal{G}$. $\mathcal{T} = \bigcup_i T_i$ is a set of JTs each of which is a corresponding JT of G_i. $\mathcal{L} = \bigcup_i \mathcal{L}_i$ is the set of LTs. Each $\mathcal{L}_i$ is a set of linkage trees one for each hyperlink incident to G_i in Ψ. Each $L\langle \mathcal{V}_i, \mathcal{V}_j \rangle$ is a linkage tree of T_i with respect to a hyperlink $V_i \cap V_j$.*

The compilation process contains tree steps, namely

- moralization of hypertree MSDAG
- triangulation of moral graph
- compiling to LJF

Moralization The compilation procedure is somewhat similar to the traditional way of organizing a DAG into a JT. First, all the agents have to be moralized by *cooperative distributive moralization* after doing local moralization within the agent itself. Cooperative distributive moralization between agents is necessary to do correct moralization over the whole hypertree MSDAG and means that newly created undirected edges are communicated between adjacent agents if both connected variables are in the agent's interface. The main

† The definition of a chordal graph can be found in section C.2 in the appendix

algorithm is `CoMoralize` (see [21] pag. 150) that calls first `CollectMlink` and after that `DistributeMlink`. `CollectMlink` is a recursive algorithm that collect links from adjacent agents. If all the links are collected then the recursive `DistributeMlink` is called. In this recursive algorithm all the adjacent agents get the collected links communicated. Now all the agents are correctly moralized.

Triangulation Second, the moralized hypertree MSDAG now needs to be triangulated. Triangulation is performed by elimination where a constraint on the elimination sequence of variables is used. It will eliminate all the variables $V_i \backslash I_i$ where V_i are variables in agent A_i and I_i is the agent A_i's interface to another adjacent agent and then it will eliminate variables in I_i. The reason for this constraint is to be able to compute a LT when compiling to a LJF. The triangulation procedure uses an algorithm called `CoTriangulate` (see [21] pag. 167) that calls the algorithms `DepthFirstEliminate` and `DistributeDlink` (see [21]). `DepthFirstEliminate` goes through all the agents in a depth first manner. If `DepthFirstEliminate` finishes the recursive algorithm `DistributeDlink` will be called and communicates the added edges to the adjacent agents. Sometimes it is required to rerun `CoTriangulate` because not every local moral graph is fully triangulated, which mean that new edges (also fill-ins) can be introduced by using the elimination sequence $(V_i \backslash I_i, I_i)$.

Linked Junction Forrest Last, the triangulated graphs have to be organized into a LJF that supports effective inference with concise message passing. This is done by first organizing all the local triangulated graphs into their JT representation. This transformation can be done locally without any knowledge from other adjacent agents. Because local JTs in a LJF are connected through LTs they must be computed from an agent's JT. This can be done for all agents and the final structure will be a LJF.

5.1.3 Inference

In this section the inference algorithm for MSBNs performed on a LJF is described. The inference algorithm tries to update the full JSP, which means that *all* agents should be able to compute their beliefs based on evidence entered in the variables of one or more subgraphs.

Potentials Theorem 3.1 shows how to compute the *joint system potential* (JSP) for JTs. In LJFs we do not have a separator defined between two hypernodes, but a LT. The potential of a LT $L\langle \mathcal{V}_i, \mathcal{V}_j \rangle$ between hypernodes Q_i and Q_j with variables $\mathcal{V}_i$ and $\mathcal{V}_j$, respectively, is defined in the following way:

$$\phi(L\langle \mathcal{V}_i, \mathcal{V}_j \rangle) = \frac{\prod_m \phi(Q_m)}{\prod_k \phi(S_k)} \tag{5.2}$$

where Q_m represents the cluster of the linkage and k is defined over all indices of all separators S_k. With the definition of $L\langle \mathcal{V}_i, \mathcal{V}_j \rangle$ the JSP $\phi(\mathcal{V})$ over LJF Φ with total universe $\mathcal{V}$ can be computed as follows:

$$\phi(\mathcal{V}) = \frac{\prod_i \phi(\mathcal{V}_i)}{\prod_k \phi(I_k)} \tag{5.3}$$

where i goes over all JTs T_i with variables $\mathcal{V}_i$ in the LJF and k is defined over all LTs L_k with interface I_k (one for each hyperlink in the hypertree).

The potential assignment for the local JTs in the hypernodes is the same as explained in section 3.1.4 for the non distributed environment. For the LT each linkage gets a uniform potential assigned. In doing that the JSP $\phi_F(\mathcal{V})$ will be equal to the JPD $P(\mathcal{V})$ of the MSBN:

$$\phi(\mathcal{V}) = \prod_i \phi(\mathcal{V}_i) = \prod_i P_i(\mathcal{V}_i) = P(\mathcal{V}) \tag{5.4}$$

Extended Linkage Potential Before we can talk about the message passing algorithm for belief propagation in LJFs we introduce the *extended linkage potential*. The messages that are passed between agents contain extended linkage potentials. Extended linkage potentials are computed by dividing the potential of a linkage by the *peer separator*.

Definition 5.9 (Peer Separator) *Let $L\langle\mathcal{V}_i, \mathcal{V}_j\rangle$ be a LT of a local JT defined between the variables V_i and V_j. Convert $L\langle\mathcal{V}_i, \mathcal{V}_j\rangle$ into a rooted tree by selecting a linkage cluster Q_i arbitrarily as the root and direct links away from it. For every linkage cluster Q_j where $j \neq i$ assign the separator with its parent linkage as the* **peer separator***.*

The extended linkage potential can be computed according to the following definition:

Definition 5.10 (Extended Linkage Potential) *Let $L\langle\mathcal{V}_i, \mathcal{V}_j\rangle$ be a LT defined between the variables V_i and V_j with linkage potentials, separator potentials and linkage peers defined. For each linkage η_i in L with peer separator R_i its* **extended linkage potential** *is*

$$\phi^*(\eta_i) = \frac{\phi(\eta_i)}{\phi(R_i)} \tag{5.5}$$

For the cluster η_i without peer the extended linkage potential is

$$\phi^*(\eta_i) = \phi(\eta_i) \tag{5.6}$$

The linkage potential $\phi(L\langle\mathcal{V}_i, \mathcal{V}_j\rangle)$ of the linkage tree $L\langle\mathcal{V}_i, \mathcal{V}_j\rangle$ can alternatively be computed by

$$\phi(L\langle\mathcal{V}_i, \mathcal{V}_j\rangle) = \prod_i \phi^*(\eta_i) \tag{5.7}$$

Messages Passing between Agents The message passing procedure can be described by two algorithm: `AbsorbThroughLinkage` and `UpdateBelief`.

Algorithm 5.1: Absorbtion through linkage

```
procedure: AbsorbThroughLinkage
/* Let $A_i$ and $A_j$ be two adjacent agents.  Agent $A_i$ is
   associated with the local JT $T_i$ and LT $L_i$ and $A_j$ with
   $T_j$ and $L_j$.  Let $\eta_i$ be a linkage in $L_i$, $C_i$ be the
   linkage host of $\eta_i$ in $T_i$ and $\eta_j$ be the corresponding
   linkage in $L_j$.  When AbsorbThroughLinkage is called on
   $A_i$ for $C_i$ to absorb through $\eta_i$ the following steps are
   performed:                                                    */
1 Agent $A_i$ request transmission of $\phi^*(\eta_j)$ from $A_j$;
2 Upon receipt $A_i$ updates its host potential $\phi'(\eta_i) = \phi(\eta_i) \cdot \frac{\phi^*(\eta_j)}{\phi^*(\eta_i)}$;
3 Agent $A_i$ updates its linkage potential $\phi^{*'}(\eta_i) = \phi^*(\eta_j)$;
```

Algorithm `UpdateBelief` is analogue to the absorbtion algorithm defined in algorithm 3.1 but only it is defined for the a distributed environment where belief is propagated through multiple linkages (hyperlink).

Multiagent Communication Initially a LJF is not in globally consistent state. In order to bring it into globally consistent state message passing has to be performed. This can be done by organizing the belief communication in the following way: `CommunicateBelief` (see definition 5.3) is the main algorithm for belief propagation and calls first the algorithm `CollectBelief` and after

Algorithm 5.2: Updating belief

```
procedure: UpdateBelief
/* Let $A_i$ and $A_j$ be two adjacent agents. Agent $A_i$ is
   associated with the local JT $T_i$ and LT $L_i$ and $A_j$ with
   $T_j$ and $L_j$. When UpdateBelief is called on $A_i$ relative
   to $A_j$ the following steps are performed:                    */
1 Upon request from $A_i$ for each linkage $Q_j$ with host $C_j$ in $L_j$, $A_j$ assigns
  $\phi_{Q_j}(Q_j) = \sum_{C_j \setminus Q_j} \phi_{C_j}(C_j)$;
2 For each linkage $Q_i$ with host $C_i$ in $L_i$, $A_i$ call AbsorbThroughLinkage on
  itself for $C_i$ to absorb through $Q_i$;
3 Agent $A_i$ performs UnifyBelief given in algorithm 3.4;
```

that the algorithm `DistributeBelief`. `CollectBelief` recursively propagates belief inwards from terminal agents towards an initiating agent, where `CollectBelief` is analogous to `CollectEvidence` for the non distributed environment proposed by [6]. `DistributeBelief` recursively propagates belief outwards from the initiating agent to the terminal agents and is analogous to `DistributeEvidence` also proposed by [6].

In order to bring a LJF into globally consistent state algorithm 5.3 needs to be performed.

Algorithm 5.3: Communicating belief

```
procedure: CommunicateBelief
1 Choose an agent $A_i$ arbitrarily;
2 Call CollectBelief on $A_i$;
3 Call DistributeBelief on $A_i$;
```

Algorithm 5.4: Collecting belief

```
procedure: CollectBelief
/* Let $A_i$ be an agent with a local JT $T_i$. A caller is
   either an adjacent agent $A_c$ or a system coordinator.
   Additional adjacent agents to $A_i$ are denoted by
   $A_1, \ldots, A_n$. When the caller calls on $A_i$ the following
   steps are performed:                                           */
1 if $A_i$ has no adjacent agents except caller then
2     perform UnifyBelief at $T_i$ and returns;
3 else
4     for each agent $A_j$ $(j = 1, \ldots, m)$ $A_i$ calls CollectBelief on $A_j$. After $A_i$
      finishes, $A_i$ calls on itself to UpdateBelief relative to $A_i$;
5 end
```

Running `CommunicateBelief` will bring the LJF into globally consistent state. Whenever there are new observations, which are injected into the LJF with `EnterEvidence` (see algorithm 3.5), the LJF becomes globally inconsistent. By running `CommunicateBelief` on the LJF will bring it back into a globally consistent state. This brings us to a very important theorem.

Theorem 5.1 *After* **CommunicateBelief** *is applied to a LJF representation, the LJF will be globally consistent.*

For a proof of theorem 5.1 see [21] page 198.

When we know that a LJF is globally consistent the potential of every cluster $\phi_{Q_i}(Q_i)$ can be calculated by:

Algorithm 5.5: Distribute belief

```
procedure: DistributeBelief
/* Let $A_i$ be an agent with a local JT $T_i$.  A caller is
   either an adjacent agent $A_c$ or a system coordinator.
   Additional adjacent agents to $A_i$ are denoted by
   $A_1, \ldots, A_n$.  When the caller calls on $A_i$ the following
   steps are performed:                                    */
1 if caller is an adjacent agent then
2     $A_i$ calls UpdateBelief on itself relative to the caller;
3 end
4 foreach agent $A_j$ $(j = 1, \ldots, m)$ do
5     $A_i$ calls DistributeBelief on $A_j$;
6 end
```

$$\phi(Q_i) = \alpha \cdot \sum_{\mathcal{V} \backslash Q_i} P(\mathcal{V}|\mathcal{E}) \tag{5.8}$$

where $\mathcal{V}$ denotes all the variables in the hypertree MSDAG, $\mathcal{E}$ denotes the set of observations and α is a normalization constant.

5.2 Prior/Likelihood Decomposable Models

Large-scale sensor network needs a special approach to distributed probabilistic reasoning that can handle bad communication links between different nodes in the network, sensor failures and dynamic agent systems. Prior/Likelihood Decomposable Models (PLDMs) (see [10, 14]) provide an inference framework that can deal with these difficulties in a robust manner.

A wide range of problems can be solved by message passing on junction trees. In this section an architecture is presented where the nodes of the sensor network assemble themselves into a *network junction tree*. In this junction tree every node has a clique and set of factors. This architecture adapts to the current situation of the environment. In other words, the architecture updates the network junction robustly with respect to unreliable communication channels and failing sensor nodes. Using an asynchronous message passing algorithm on this junction tree the nodes can solve the inference problem efficiently.

Each node in the network junction tree has a set of associated *query variables*. After the message passing algorithm converges every node can compute correct posterior over these query variables. Because convergence of the message passing algorithm is difficult in dynamic agent systems *partial beliefs* will be considered. Partial beliefs combine local information with information that was received through messages.

In the ideal case the inference algorithm should have the following properties:

- Local correctness - before any communication has occurred, each node can compute correct posterior of its query variables given its measurements;
- Global correctness - after convergence, each node can compute correct global posteriors of its query variables given its measurements;
- Partial correctness - before convergence, a node can compute correct partial posteriors of its query variables given the measurements that have been incorporated in the messages it has received.

In this section an efficient algorithm, called *robust message passing*, is presented that satisfies local and global correctness and a relaxed from of partial correctness.

5.2.1 Inference Problem

In the PLDMs approach a network model is assumed where each node can perform local computations and is able to communicate with other sensor nodes through a broadcast channel. The nodes of the network can fail and new nodes may be introduced. The communication between two nodes can only be performed when there exist a good *link quality* between these two nodes. Link quality is defined through the probability of a successful transmission of a message. These communication links between nodes are wireless and therefore the link quality can easily change over time. In addition, the link qualities between node-pairs can be correlated.

The associated random variables of the inference problem can be divided into two classes of variables: observable and hidden. The observable variables are denoted with $\mathcal{M} = \{M_1, \ldots, M_n\}$ and are called *measurement variables*. Each measurement variable corresponds to one of the sensor network nodes. The hidden variables are denoted with $\mathcal{X} = \{X_1, \ldots, X_n\}$ and are called *environment variables*. These stochastic variables describes the state of the sensor network's environment. Measurement and environment variables represent the variables of the problem domain, so $\mathcal{M} \cup \mathcal{X} = \mathcal{V}$, where $\mathcal{V}$ is the set of all variables defined in the problem domain. For each measurement variable M_k we have a measurement model that is defined through a conditional probability $P(M_k|\mathcal{B}_k)$ where $\mathcal{B}_k \subseteq \mathcal{X}$. Given these variables the full joint can be calculated by:

$$P(\mathcal{V}) = \overbrace{\alpha \cdot \prod_i \phi(Q_i)}^{\text{factorized prior } P(\mathcal{X})} \cdot \prod_k^n \overbrace{P(M_k|\mathcal{B}_k)}^{\text{measurement model}} , \tag{5.9}$$

where each $Q \in \mathcal{Q}$ is a subset of the environment variables and α is a normalization constant.

Every measurement model is stored at the sensor node where the corresponding measurement is retrieved and the factors of the prior $P(\mathcal{X})$ are partitioned across the nodes of the network. Every node has a subset of environment variables $\mathcal{Y} \subseteq \mathcal{X}$ that are called *query variables*. The distributed inference problem is defined by the calculation of $P(\mathcal{Y}|e_{M_1}, \ldots, e_{M_i})$ after every node has received its observations. This means that the nodes have to communicate in order to compute the correct local posteriors of $\mathcal{Y}$ for every node.

5.2.2 Distributed Sensor Calibration

An example of a sensor network is given in figure 5.3 where every sensor can measure its surrounding temperature. A Markov graph depicts the probabilistic dependencies between sensor nodes according to their location in the map. The temperature measurements of the sensor nodes are correlated, because the measured temperature at node 54 will say something about the temperature measurement given at node 53. (This correlation between node 53 and 54 is also depicted in the Markov graph in figure 5.3). The task of sensor calibration is to automatically detect biases of sensors and correct for these biases. This is possible because the sensors are correlated and the biases are marginally independent.

For the sensor network in figure 5.3 we can set up the graphical model given in figure 5.4 (a). Each node has three associated variables: the observable temperature M_i, the true temperature at its location T_i and the sensor bias B_i. Here the true temperature variable T_i and the bias variable B_i are the environment variables. Given the temperature measurements we can compute the posterior distributions over bias variables to automatically calibrate the sensors. The joint probability distribution can be calculated through:

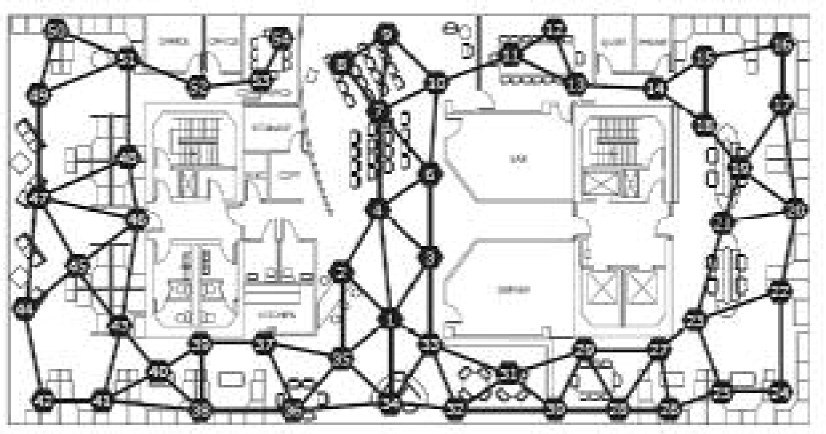


Figure 5.3: A Markov graph of the nodes' temperature in the Intel Berkeley Lab. (Image taken from [10])

$$P(\mathcal{V}) = \overbrace{\alpha \cdot \prod_{(i,j)\in\mathcal{E}} \phi(T_i, T_j)}^{\text{temperature prior}} \cdot \prod_{i\in\mathcal{N}} \overbrace{P(B_i)}^{\text{bias prior}} \; \overbrace{P(M_i|B_i, T_i)}^{\text{measurement model}} , \tag{5.10}$$

where $\mathcal{N}$ and $\mathcal{E}$ are the nodes and edges of the Markov network in figure 5.3, respectively. Every bias prior $P(B_i)$ is distributed to node $i \in \mathcal{N}$ and the temperature prior $\phi(T_i, T_j)$ is distributed to node i or node j.

In this example the query variables are T_i and B_i. Therefore, we want to calculate the posterior $P(T_i, B_i|e_{M_1}, \ldots, e_{M_n})$ of all observations injected into the network.

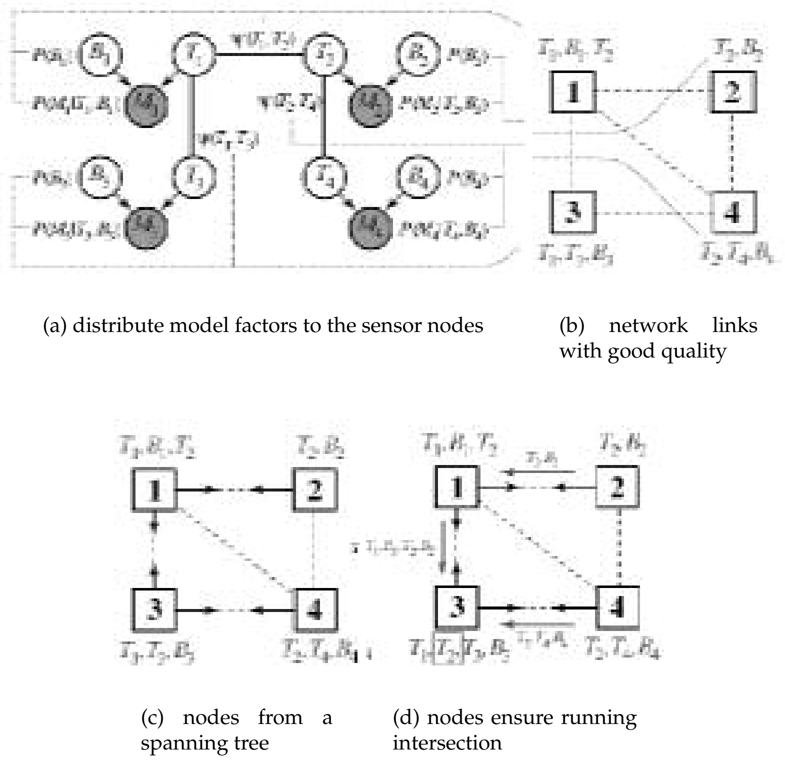


Figure 5.4: A graphical model for the sensor network given in figure 5.3 for four sensor nodes. (Images taken from [10])

(a) distribute model factors to the sensor nodes

(b) network links with good quality

(c) nodes from a spanning tree

(d) nodes ensure running intersection

5.2.3 Architecture

The nodes in the sensor network organizes themselves into a JT by running four algorithms concurrently on each sensor node:

- spanning tree formation
- junction tree formation
- tree optimization
- belief propagation

Let's discuss each algorithm in turn.

Spanning Tree Formation Every sensor node in the network chooses a set of neighbor nodes to form a spanning tree. The set of neighbor nodes is determined by the quality of the links between sensor nodes. For example in figure 5.4 (b) the sensor nodes discover that there is a bad communication link between node 3 and 4. The spanning tree algorithm is constantly running in order to anticipate on changing link qualities. It can happen that a link quality deteriorates and that the spanning tree must change. The used spanning tree algorithm will find a stable spanning tree if there exists one.

Junction Tree Formation When the spanning tree is constructed the concurrently running junction tree formation algorithm can start constructing a valid JT. The constructed spanning tree in figure 5.3 (c) is not a valid junction tree because it does not have the running intersection property (see definition 3.2). Therefore, it is necessary to run the junction tree formation algorithm.

In the junction tree formation algorithm the nodes collaborate to learn which variables they should have in order to ensure the running intersection property. This algorithm uses message passing along the spanning tree, where the messages contain information about the neighboring nodes.

For each edge $i \to j$ we define the *variables reachable to j from* i by

$$\mathcal{R}_{ij} = \mathcal{V}_i \cup \bigcup_{k \in n(i):k \neq j} \mathcal{R}_{ki} \tag{5.11}$$

where $n(i)$ are the neighbors of node i in the spanning tree. Node i computes $\mathcal{R}_{ij}$ by taking the union of all collected messages $\mathcal{R}_{ki}$ from its neighbors k and the variables $\mathcal{V}_i$ of node i. A message with reachable variable $\mathcal{R}_{ij}$ is sent to node j.

When a node receives two reachable variables with some variable X, then it should also contain this variable. The clique at node i is computed by:

$$Q_i = \mathcal{V}_i \cup \bigcup_{j,k \in \tau(i):j \neq k} \mathcal{R}_{ij} \cap \mathcal{R}_{ki} \tag{5.12}$$

In figure 5.4 (d) we can see that node 3 received two reachable variable messages containing variable T_2 which is not in $V_3 = \{T_1, T_3, B_3\}$. T_3 is added to V_3 to ensure the running intersection property.

Tree Optimization The found JT in the junction tree formation algorithm is not always the most efficient JT. When the size of cliques and separators is small the computation and communication cost in a JT is cheaper. By considering the size of cliques and separators the tree optimization algorithm tries to find the most efficient junction tree by swapping edges between nodes.

Belief Propagation In the final algorithm, called *robust message passing*, the inference problem is solved. The used algorithm for belief propagation differs from the standard sum-product algorithm. The proposed algorithm for PLDMs uses asynchronous message passing and is able to compute *partial beliefs*, that is, at any point of the inference process the calculated beliefs are an approximation to the correct posteriors. The reason that the standard sum-product algorithm is not suitable for sensor networks is because it cannot handle node loss. When a node dies it can take out very important priors that are crucial for the computation of the correct posterior. Also, the sum-product algorithm scales with the size of the cliques and the separators which in turn means that the computational cost depends on the network topology. In section 5.2.6 robust message passing algorithm is discussed.

5.2.4 Model Decomposition

The global joint distribution can be decomposed into smaller local priors. For example, in figure 5.4 (a) we can calculate the prior $P(T_1, T_2, T_3, T_4)$ over the temperature in the following way:

$$P(T_1, T_2, T_3, T_4) = \frac{P(T_1, T_3)P(T_1, T_2)P(T_2, T_4)}{P(T_1)P(T_2)}$$

In this example we see that global joint $P(T_1, T_2, T_3, T_4)$ can be decomposed into the smaller priors $P(T_1, T_3)$, $P(T_1, T_2)$ and $P(T_2, T_4)$. Before we can identify the local priors we first have to compute the *external junction tree*. By using message passing on the external JT we can compute the clique and separator marginals. The global prior is then represented as:

$$P(\mathcal{X}) = \frac{\prod_i P(Q_i)}{\prod_j P(S_i)}, \tag{5.13}$$

where Q_i ranges over the cliques of the external JT and S_i ranges over the separators. For the algorithm only the clique priors are of interest and form the *implicit* representation of the global prior. The separator marginals do not have to be specified because they can be recomputed by using the external JT.

5.2.5 Distribution of the Model

After the local priors are identified they must be distributed over the sensor nodes. One important property of a sensor node is that the local posterior can be computed (local correctness). This is possible if the sensor node can access the local prior over the query variables $\mathcal{Y}_i$ and the parents of $\mathcal{Y}_i$: $\pi(\mathcal{Y}_i)$. Consequently, the variables $\mathcal{Y}_i$ should be located at the sensor node next to the measurement models of that particular sensor node.

In figure 5.5 (a) the graphical model is given of the sensor node with their dependencies and good quality links. Figure 5.5 (b) shows the external JT for this graphical model. In order to determine the cliques that have to be distributed over the model the external JT computation must be prior to the actual belief propagation algorithm. Figure 5.5 (c) shows how these cliques are distributed over the sensor nodes. Sensor node 2 gets the prior $P(T_2, T_3, T_5)$ with the same variables defined in one of the cliques in the external JT. Besides that, sensor node 2 also gets the measurement model $P(M_2|T_2)$ because M_2 is located at that sensor node. Sensor node 2 can now compute its local posterior correctly.

5.2.6 Robust Message Passing

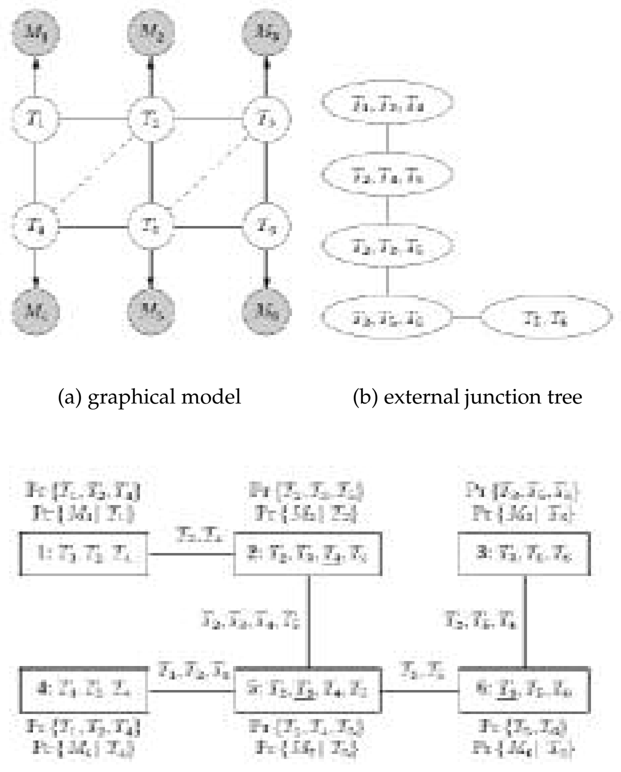
Every sensor node that participates in the network JT will have a prior and a measurement model. These two factors can be reorganized into the *prior/likelihood (PL) factor*, where the measurement variable is instantiated in the measurement model.

Definition 5.11 (Prior/Likelihood (PL) factor) *A prior/likelihood (PL) factor for a set of environment variables C is a pair $\langle \pi_C, \lambda_C \rangle$ where*

- *π_C is a (possibly approximate) prior distribution for C*
- *λ_C is a (possibly approximate) likelihood function $P(e_{M_C}|C)$*

$\langle \pi_C, \lambda_C \rangle$ is **exact** *if π_C and λ_C are exact.*

Figure 5.5: Robust message passing. (Images taken from [10])



(a) graphical model (b) external junction tree

(c) network junction tree and distribution of the model

Their are two basic operations involved in robust message passing, namely combination and summarization.

Definition 5.12 (Combination) *Let $\langle \pi_C, \lambda_C \rangle$ and $\langle \pi_D, \lambda_D \rangle$ be two PL factors. The* **combination** *of $\langle \pi_C, \lambda_C \rangle$ and $\langle \pi_D, \lambda_D \rangle$ is:*

$$\langle \pi_C, \lambda_C \rangle \otimes \langle \pi_D, \lambda_D \rangle = \left\langle \frac{\pi_C \times \pi_D}{\sum_{C \setminus D} \pi_C}, \lambda_C \times \lambda_D \right\rangle \tag{5.14}$$

Definition 5.13 (Summarization) *Let $\langle \pi_D, \lambda_D \rangle$ be a PL factor and S be a set of random variables. The* **summary** *of $\langle \pi_D, \lambda_D \rangle$ to S is:*

$$\bigoplus_S \langle \pi_D, \lambda_D \rangle = \left\langle \sum_{D \setminus S} \pi_D, \frac{\sum_{D \setminus S} \pi_D \times \lambda_D}{\sum_{D \setminus S} \pi_D} \right\rangle \tag{5.15}$$

With the combination rule we can multiply prior probabilities and the likelihoods. The summarization, on the other hand, is used to compute a marginal prior and computes the marginal likelihood by forming the joint, marginalizing it down, and dividing out the marginal prior.

The robust message passing algorithm uses combinations and summarizations on PL-factors. A set of PL-factors is called a model fragment factor.

Definition 5.14 *A* **model fragment factor** Φ *is a collection of PL-factors* $\{\langle \pi_C, \lambda_C \rangle : C \in \mathcal{C}\}$.

Model fragment factors represent the factorization of the model where the priors are used implicitly to represent the global prior. This global prior can be reconstructed by forming a JT and compute the global prior according to formula 5.13. If we want to compute the posterior distribution represented by the model fragment factor we need to build a canonical clique tree.

Definition 5.15 (Canonical Clique Tree) *Let Φ be a model fragment factor. A* **canonical clique tree** *for Φ is a tree over the PL factors of Φ which has maximum cardinality variable intersections of neighboring cliques. Φ is consistent if it has a canonical clique tree T such that (1) T satisfies the running intersection property and (2) the conditional independencies encoded by T are also encoded by the external junction tree.*

When we know the canonical clique tree we can *flatten* the model fragment factor to a single PL-factor.

Definition 5.16 *Let $\Phi = \{\langle \pi_C, \lambda_C \rangle : C \in \mathcal{C}\}$ be a model fragment factor. A PL-factor $\langle \pi_V, \lambda_V \rangle$ is a* **flattening** *of Φ if it can be obtained in the following manner:*

- *Compute a canonical clique tree T for Φ.*
- *Iterate: let C be a leaf clique of T with neighbor D. Replace C and D with a new clique $C \cup D$ whose associated PL-factor is $\langle \pi_C, \lambda_C \rangle \otimes \langle \pi_D, \lambda_D \rangle$.*

Then, the posterior is computed by simply multiplying $\pi_V \times \lambda_V$, the product of the flat prior and likelihood.

The main operations of robust message passing are combinations and summarizations on model fragment factors. The actual inference is done by using the summarization operation.

Definition 5.17 (Summary on Model Fragment Factors) *Let $\Phi = \{\langle \pi_C, \lambda_C \rangle : C \in \mathcal{C}\}$ be a model fragment factor and let S be a set of random variables. Another model fragment factor Ψ is a summary of Φ to S iff it can be obtained in the following way:*

- *Compute a canonical clique tree T for Φ.*
- *Iterate: let C be a leaf clique of T with neighbor D such that $C \cap S \subseteq D$. If there is no such clique terminate. Update the PL-factor of D as follows:*

$$\langle \pi'_D, \lambda'_D \rangle = \langle \pi_D, \lambda_D \rangle \otimes \left(\bigoplus_{C \cap D} \langle \pi_C, \lambda_C \rangle \right) \tag{5.16}$$

  Then remove C from T and $\langle \pi_C, \lambda_C \rangle$ from Φ.

PL-factors are repeatedly pruned by transferring its likelihood information onto another PL-factor by eliminating the prior information that is not necessary anymore. The proposed algorithm supports exact inference.

6 Comparison

In chapter 4, sections 5.1 and 5.2 three different approaches to distributed probabilistic reasoning were discussed, namely Distributed Perception Networks (DPNs), Multiply Sectioned Bayesian Networks (MSBNs) and Prior/-Likelihood Decomposable Models (PLDMs), respectively. In this chapter we will compare these three approaches theoretically. Most existing comparisons between inference algorithms focus on the architectures and algorithms in non-distributed environments, for example in [9] and [8]. Another comparison in [22] illustrates different inference algorithms within the MSBN inference structure. The comparison presented in this chapter is not only based on the inference architectures and algorithms, but will also focus on challenging aspects of distributed systems such as self-configuration and compilation, dynamic agent systems and sequential evidence processing.

In section 6.2 the distributed inference architectures are discussed with respect to the modeling complexity. Exact inference in a distributed system can only be achieved if certain constraints are respected when connecting local DAGs. These constraints limit the modeling complexity, i.e. the repertoire of the domains that can be adequately described with the distributed models. We show that each of the compared approaches imposes different modeling constraints, which correspond to different modeling flexibility.

In distributed systems the agents, each supporting only partial fusion capabilities, must discover other agents that can provide the inputs which are required to perform a global task. Such configuration must result in meaningful distributed (global) DAG-structures. In addition, a compilation concerns itself with the transformation of the DAG-structure to a probabilistic structure that supports efficient propagation. Section 6.3 discusses the differences between the three distributed approaches with respect to the configuration and compilation.

Section 6.4 discusses the different distributed reasoning approaches with respect to the capability to cope with dynamic agent systems. A dynamic agent system is an environment where agents can join and/or leave a fusion system at runtime. Beside dynamic agent systems self-configuration of the probabilistic DAG-structure will also be discussed.

Section 6.5 discusses the differences of belief propagation in the three distributed reasoning approaches. Evidence can be processed simultaneously or in sequence depending on the problem domain. We show that sequential processing of evidence with high frequency can cause problems for inference algorithms. Frequent evidence processing will be discussed in section 6.6.

6.1 Problem Domains

Before we compare the three distributed inference approaches with respect to the features described above, we first give an overview of the type of problem domains that can be solved by using these approaches.

In some problem domains it is not always obvious which of the information sources will participate in the inference process prior to the operation. For example, using a large sensor network covering a large geographical area to do weather predictions. We never know which of the sensors will be available during the inference process. This requires a probabilistic inference approach that can incorporate such sensors on the fly and also exclude a sensor when it becomes unavailable. It turns out that MSBNs or PLDM might not be suitable for such domains. However, if we are interested in a distribution over a single variable, then we can show that DPNs support theoretically rigorous exact inference even if the information sources and information demand are unknown prior to the operation and large amounts of heterogeneous information has to be processed.

Complex problem domains, as for example monitoring of complex systems, often require complex and highly connected BNs which encode probabilistic relations between the states of the different system components. In such applications we must compute globally consistent distributions (see definition 3.5) over all variables in the BN. In other words, the posteriors of every variable describing a component in the system must be based on the entire set of observations about the system (i.e. evidence) .

In many large (wireless) sensor networks failing communication channels and sensors are inevitable. This requires an inference approach that is able to adapt to the environment and is robust for losing important information. PLDMs are focusing on applications where all potentially useful information sources and dependencies between their reports are known a priori and certain communication channels and sensor information sources can become unavailable.

6.2 Modeling Complexity

In general, the model complexity corresponds to the degree of the probabilistic dependencies between the modeled variables. In other words, complex models correspond to multiply connected graphs. In this section we show that the three compared approaches to distributed inference impose different modeling constraints which can reduce the modeling flexibility to a certain degree.

It turns out that MSBNs and PLDMs can model more general distributed probabilistic models than DPNs.

6.2.1 Distributed Perception Networks

In DPNs, according to definition 4.6, the separator sets must contain only one variable. In other words, any two network fragments can be connected through a single directed link which limits the modeling flexibility. Consequently, highly connected Bayesian models are difficult to distribute in a DPN. In other words, we assume that DPNs are suitable for a significant class of problem domains which can be described by DAG structures with a significant amount of conditionally independent network fragments.

A convenient consequence of definition 4.6 is that the d-sepset, hypertree and graph consistency requirements between connected DAGs are guaranteed without further verification*.

* For a definition of d-sepset, hypertree and graph consistency refer to section 5.1

6.2.2 Multiply Sectioned Bayesian Networks

MSBNs can deal with highly connected DAG structures, since the separator sets can be arbitrarily large. In this way local DAGs from different agents can be connected through multiple links, i.e. each connected pair of DAGs can share several variables. However, three constraints apply to the separator set and the distributed DAG structure: (i) every variable in the separator set must be a d-sepnode (see definition 5.3), (ii) the graph spanned by the separator set must be identical (see definition 5.1) and (iii) the resulting connected DAG should be a hypertree (see definition 5.4). The first two properties are required otherwise the conditional distribution is difficult to define. The third property ensures that consistent reasoning can be performed.

6.2.3 Prior/Likelihood Decomposable Models

Like MSBNs, PLDMs also allow arbitrarily many d-sepnodes in the separators. This means that this approach supports complex BNs. Also, the same restrictions as in MSBNs apply to the separator sets, global DAG structure and the DAG structure that is spanned by the separator set.

6.3 Configuration and Compilation

In this section we compare the configuration and compilation processes of the three approaches. Configuration is the process where agents are combined into agent systems in such a way, that they assemble meaningful distributed probabilistic models. This process corresponds to the specification of the communication interfaces between the cooperating agents. Compilation of probabilistic structure, on the other hand, is the process where the BN is transformed to another probabilistic structure that exploits d-separations (like JTs or LJFs) to support efficient inference.

6.3.1 Distributed Perception Networks

In DPNs local BNs are compiled independently of each other prior to the network configuration. The local BNs can be precompiled before the configuration because of the configuration rules given in definition 4.6. These rules make sure that local DAGs are connected in such a way that the resulting probabilistic structure automatically corresponds to a JT.

6.3.2 Multiply Sectioned Bayesian Networks

In the MSBNs approach the configuration precedes the compilation process. Namely, the compilation consists of cooperative distributed moralization and triangulation (see section 5.1.2), which require knowledge from adjacent local BNs. This, however, requires that the local BNs are assembled into a global BN prior to the compilation. Compilation for MSBNs requires the following steps one after the other: moralization, triangulation and finally compilation to a LJF. The used algorithms for moralization and triangulation will fully traverse through the distributed network. In MSBN the BNs need to be compiled before consistent inference can be done. In other words, inference in MSBNs requires a sequence of different synchronized processes, which can be problematic in domains where complex MSBN systems are distributed throughout dynamic agent systems, where agents join and leave a fusion system frequently.

6.3.3 Prior/Likelihood Decomposable Models

In PLDMs the configuration is driven through the quality of available communication links. The configuration is not concept driven, like in DPNs, but purely based on the good quality of the communication links. This configuration process is captured in the spanning tree algorithm (see section 5.2.3). Because the quality of the links determines how the local BNs (in sensor nodes) are connected, the assembled BN often does not correspond to a JT. Therefore, the subsequent compilation consists of the junction tree formation and tree optimization algorithms (see also section 5.2.3). The advantage of PLDM approach is that configuration and compilation algorithms can be performed simultaneously.

6.4 Self Configuration and Dynamic Agent Systems

Self configuration is the process where agents discover each others services and autonomously form meaningful distributed probabilistic models. This is important in the systems where agents relevant for a particular inference process are not known prior to the operation. In addition, agent systems are often dynamic, since new agents can join a system or agents in a fusion system can become unavailable due to different processing and communication failures. Consequently, robust inference approaches should support self-configuration in order to incorporate new agents joining a fusion system on the fly while the inference quality should not be significantly reduced if agents become unavailable. In this section we compare the three architectures and inference algorithms with respect to the *self-configuration* and *dynamic agent systems*.

6.4.1 Distributed Perception Networks

DPNs can deal with dynamic agent systems that change rapidly. The DPN organization constraints given in definition 4.6 ensure that the constructed model will have separators of size one. Because of that, no significant coordination is required between the agent designers. Moreover, the rules in definition 4.6 make sure that the resulting structure will always correspond to a JT. This means that no precompilation of a distributed BN is required. Also, DPNs are used in applications where distribution over a single hypothesis variable must be computed; i.e. global consistency is not required. Because of that the inference process does not require a full traversal through the network. In other words, the computation of the posterior probability can be efficient and is, in general, not influenced by changing agent systems, which is the case with MSBNs.

Contrary to the MSBNs and PLDMs, DPN agents can autonomously configure meaningful fusion systems without any prior knowledge of the available agents and adapt to dynamic agent systems efficiently by using a self configuration algorithm (see section 4.3); i.e. DPNs can autonomously find the relevant local BNs and adapt to the current environment at runtime.

However, the drawback of the constraints in definition 4.6 is that the assembled models have more limited topologies than models supported by the other two approaches. In addition, DPNs support only diagnostic inference between agents.

6.4.2 Multiply Sectioned Bayesian Networks

MSBNs might not be very suitable for dynamic agent systems. In large MSBNs adding or removing a DAG structure can be a problem because of several reasons:

(i) The d-sepset of agents can be arbitrarily large which makes them difficult to connect to other d-sepsets without a significant coordination between agent designers. Without such coordination, the chance that local BNs will support valid d-sepsets, a precondition for successful self-organization, is very low.

(ii) The compilation algorithms `CoMoralize` and `CoTriangulate` (discussed in section 5.1.2) must be executed if the network structure changes. These algorithms need to be terminated successfully before inference can be done. In an environment where agents join and leave the system very frequently this can be a problem, since the agent system could change faster than the sequence of configuration and compilation processes
would be completed.

(iii) The inference algorithm `CommunicateBelief` in section 5.1.3 calls two algorithms that traverse the full hypertree (see definition 5.4 for description of a hypertree). When agents leave and/or join the agent system with high frequency the algorithm cannot bring the hypertree into globally consistent state. Global consistency is a requirement to compute correct posteriors from every local BN/agent.

In MSBNs self configuration is not trivial because of the problems (i) and (ii).

6.4.3 Prior/Likelihood Decomposable Models

PLDMs support dynamic agent systems in a limited way. Before reasoning takes place in PLDMs the Markov graph, showing the probabilistic dependencies between sensor nodes, should be known a priori. From this graph the external JT can be computed. The external JT gives insight in the kind of priors that should be distributed over the sensor nodes. When the robust belief propagation algorithm is running the sensor network can change over time dependent on the link qualities, but the network cannot add new sensor nodes that were not there when the external JT was computed. This is different from DPNs where agents that are not known a priori can be added during runtime.

6.5 Exact Propagation

The compared distributed inference algorithms are focusing on exact belief propagation and can be classified into two types:

(i) Inference algorithms that allow computation of the joint system potential (JSP), which means that the correct posterior is computed for every random variable in every local DAG;

(ii) Inference algorithms that support the computation of the posterior distributions reflecting all pieces of evidence in only one agent.

The inference algorithms of type (i) support global belief consistency (definition 3.5). Global belief consistency is required for the computation of the JSP. However, there are problem domains that do not require the computation of the JSP. Instead, we are interested in the posterior distribution over a single variable within a distributed system, which means that we can use inference algorithms of type (ii). Inference algorithms of type (ii) are computationally cheaper compared to the inference algorithms of type (i).

6.5.1 Distributed Perception Networks

DPNs do not support computation of globally consistent posteriors in all agents. They belong to the inference class (ii). DPNs can compute the globally correct posterior of variables within a single BN at the top of the hierarchy of distributed BNs (see section 4.4); i.e. only posterior distributions within a single BN (i.e. one agent) are based on all observations injected into a DPN system. For these variables the evidence must be collected to the correct agent, which is described in the algorithms 4.3, 4.4 and 4.5. These algorithms can be compared to the algorithm for MSBNs in algorithm 5.4 in that they collect evidence to a specified agent. In DPNs collecting evidence to an agent corresponds to diagnostic reasoning between agents.

The inference in DPNs is robust for two main reasons: (i) a DPN domain often has a significant amount of independent branches and only diagnostic reasoning is performed. These properties make inference robust in a DPN (see [13]); (ii) failing agents (with important priors) can be substituted by other agents modeling identical events.

6.5.2 Multiply Sectioned Bayesian Networks

As already mentioned in the previous section, MSBNs try to bring the LJF (see definition 5.8) into globally consistent state. In other words, MSBNs support inference algorithms of type (i). To bring the LJF into globally consistent state the algorithm `CommunicateBelief` defined in algorithm 5.3 has to be executed. Algorithm `CommunicateBelief` requires that all messages are passed in the LJF. If this is not the case important priors, defined in other hypernodes (definition 5.4) can be lost. Losing important prior can give a skewed view on the probability of different events. This means that MSBNs are not suitable in problem domains where we cannot guarantee reliable message passing and reliable services provided by agents.

6.5.3 Prior/Likelihood Decomposable Models

Belief propagation algorithm in PLDMs is focusing on three properties, namely local correctness, global correctness and partial correctness (see section 5.2). For local correctness this means that every agent can compute the correct local posterior. Global correctness means that every agent can compute the correct global posterior after the belief propagation algorithm is finished (the same as for MSBNs). Finally, the most important property is partial correctness, which means that at any point during the inference process correct partial posteriors can be computed. The PLDM's inference algorithm **robust message passing** can handle a relaxed form of the partial correctness property. Because most priors in a PLDM are stored redundantly and the algorithm passes likelihoods between agents instead of separator marginals this partial correctness property makes the PLDM inference algorithm very robust for sensor failures and bad communication channels.

6.6 Frequent Evidence Processing

In general, the main goal of any inference process is to compute posterior probability distributions over variables of interest by considering entire evidence. However, in large systems the sensors can provide observations with high frequency. If for example such a system has 1000 sensors that would sense something every 0.1 seconds then there will be 18.000.000 observations every half an hour. Therefore, the computation of posterior distributions that reflect the complete evidence can be challenging in distributed BNs because of the following problems:

(i) Communication channels have a limited throughput capacity and can introduce messaging delays;

(ii) Computational resources are limited and might not be able to cope with large number of received messages.

In other words, the updating of the posteriors of interest requires a certain time interval Δt_p for each piece of evidence. Assume that with every Δt_n a new evidence is obtained and the corresponding variable somewhere in the distributed BN is instantiated. Whenever we have the situation where $\Delta t_n < \Delta t_p$ messages with partial beliefs will be either buffered or they will be lost. In large decentralized systems this often cannot be avoided. We can obtain posterior distributions that are based only on a subset of the full evidence set if either the messages are lost or we cannot wait until all observations are communicated and processed. Clearly, with the growing numbers of observations these problems become significant if for the majority of the observations $\Delta t_n < \Delta t_p$.

In addition, algorithms that require global consistency will make the interval Δt_p larger, which will result in a situation where $\Delta t_n < \Delta t_p$ is more likely. In situations where $\Delta t_n < \Delta t_p$ global consistency will never be obtained without using synchronization. However, using synchronization to get a globally consistent model will make the difference between the computable posterior probability from the globally consistent model and the correct posterior reflecting the current situation larger.

6.6.1 Distributed Perception Networks

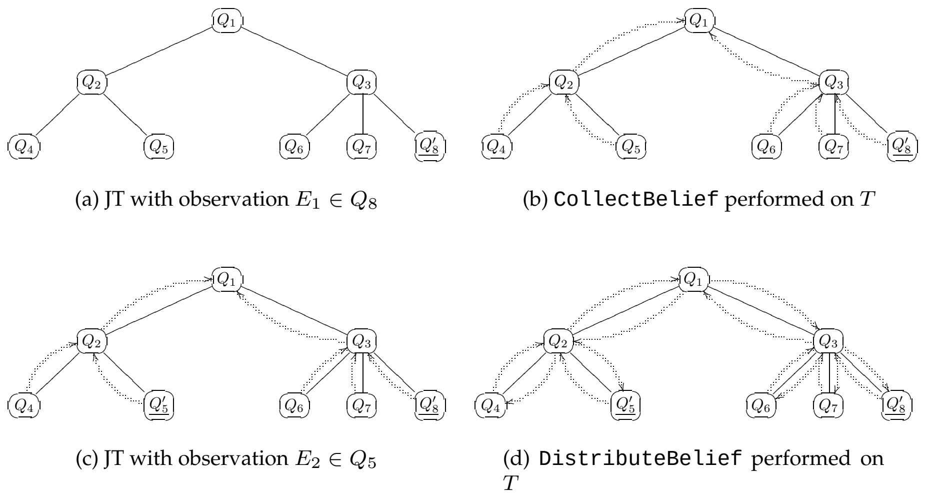
DPNs are used in application that do not require globally consistent distributions over all variables in a distributed model. Therefore, only a collect evidence (see section 4.4.4) has to be executed. Consequently, this will make the situation $\Delta t_n < \Delta t_p$ less likely, because evidence can be processed faster when no global consistency is required. Also, only the last message that is sent from a sensor is of importance and because of that many previously sent messages can be dropped. This will also reduce Δt_p.

6.6.2 Multiply Sectioned Bayesian Networks

MSBNs use an inference algorithm that is based on global consistency. Inference algorithms that use global consistency have difficulties of dealing with large amount of observations for the reasons indicated before in this section.

With an example we illustrate that instantiating evidence sequentially will require to rerun the MSBN algorithm `CommunicateBelief` which again requires a larger Δt_p. The junction tree T in figure 6.1 is used as an inference structure. The example demonstrates a situation where only two observations have to be processed sequentially, but when many observations have to be processed in a sequential order this will inescapably lead to inconsistent beliefs.

Figure 6.1: Junction Tree with frequent evidence updating



(a) JT with observation $E_1 \in Q_8$

(b) `CollectBelief` performed on T

(c) JT with observation $E_2 \in Q_5$

(d) `DistributeBelief` performed on T

In figure 6.1 (a) we can see the clusters $\{Q_1, \ldots, Q_8\}$ that are distributed over a set of agents. In figure 6.1 (a) an observation is received for a variable $E_1 \in Q_8$ which updates the cluster potential $\phi(Q_8)$ denoted by $\underline{Q'_8}$ in the figure. Because of this Q_1 can initiate an algorithm like `CommunicateBelief` described in algorithm 5.3 for MSBNs to bring T into globally consistent state.

CommunicateBelief will first call CollectBelief described in algorithm 5.4 that will result in figure 6.1 (b) where the curved arrows indicate that belief is propagated over the associated communication link. Let us assume that at this point evidence is instantiated at cluster Q_5 for variable $E_2 \in Q_5$ which will update the cluster potential $\phi(Q_5)$ denoted as $\underline{Q_5'}$ in figure 6.1 (c). At this point, the initiated algorithm by Q_1 is still not finished and will run DistributeBelief described in algorithm 5.5 shown in figure 6.1 (d) after finishing. Calculating the marginal probability of $P(A)$ where $A \in Q_1$ will only be based on the evidence set $\mathcal{E} = \{E_1\}$ and that is clearly incorrect because we had the actual evidence set $\mathcal{E} = \{E_1, E_2\}$. To solve this problem CommunicateBelief can be rerun to bring T again in a globally consistent state, but in an environment where frequent evidence observations are processed sequentially there is a high probability that again new evidence is instantiated during the execution of CommunicateBelief. This makes it difficult to have a globally consistent JT since we have to rerun Communicate-Belief frequently.

The inference algorithm for MSBNs are suitable for problem domains where all the variables in the network are instantiated with evidence simultaneously or the sequence of observations are sufficiently slow that $\Delta t_n > \Delta t_p$

6.6.3 Prior/Likelihood Decomposable Models

In PLDMs the inference algorithm brings, after convergence, the inference structure into globally consistent state. However, PLDMs is not fully dependent on global consistency, because the robust message passing algorithm allows computation of partial beliefs (partial correctness). Partial beliefs are an approximation to the posterior probability after the algorithm is finished and processed all observations. Although, when correct global posteriors are desired the system needs to be in globally consistent state and the same problems apply when using MSBNs. Because of partial correctness the situation where $\Delta t_n < \Delta t_p$ is not as problematic as in the MSBN case.

6.7 Concise Overview

This section presents a concise overview of the pros and cons of the three presented belief propagation approaches used in a distributed environment. The tables 6.1, 6.2 and 6.3 show the advantages and disadvantages of the discussed inference approaches DPNs, MSBN and PLDM, respectively, with respect to different application aspects.

Table 6.1: Pros and cons of the DPN approach

Feature	Pros	Cons
Modeling complexity	-	Limited
Configuration & Compilation	Precompilation possible; configuration can be fast	-
Dynamic Agent Systems & Self Configuration	Supported	-
Exact Propagation	Uses asynchronous message passing	Only computation of correct posterior of variables in one agent; only diagnostic reasoning supported
Frequent Evidence Processing	Supported	-

Table 6.2: Pros and cons of MSBNs

Feature	Pros	Cons
Modeling complexity	Can be complex	-
Configuration & Compilation	-	Configuration needed before compilation; Expensive communication and compilation becomes intractable in large MSBNs
Dynamic Agent Systems & Self Configuration	-	Difficult, d-sepsets can be arbitrarily large which make them difficult to connect without high coordination between agent suppliers
Exact Propagation	Brings model into globally consistent state which allows computation of correct posterior of every variable in each agent	Can be problematic in large MSBNs with failing agents, bad communication links and frequent evidence processing
Frequent Evidence Processing	-	Difficult, especially when computational resources are limited and globally consistent beliefs for every variable are required

Table 6.3: Pros and cons of PLDMs

Feature	Pros	Cons
Modeling complexity	Can be complex	-
Configuration & Compilation	algorithms run concurrently; adapts JT to possible communication links	can be expensive operation when communication links do not have stable quality
Dynamic Agent Systems & Self Configuration	Supported	Only possible between a predefined set of agents
Exact Propagation	Uses asynchronous message passing; calculation of partial belief (partial correctness) possible; propagation robust against node failures and bad communication links	-
Frequent evidence processing	Supported	Can be intensive in the case of many unstable communication links

7 Conclusions

THE main purpose of this thesis was twofold. Firstly, it introduced a distributed inference approach called Distributed Perception Networks (DPNs). Secondly, DPNs and two other well known approaches to Bayesian inference in a distributed environment were compared in the context of different application aspects. Conclusions with respect to DPNs are given in section 7.1 and conclusions with respect to the comparison are given in section 7.2.

7.1 Distributed Perception Networks

We introduced Distributed Perception networks (in chapter 4), a MAS-approach to information fusion using distributed Bayesian networks. We have shown that DPNs can support (i) fusion of very heterogeneous and noisy information; (ii) decentralized self organization of local network fragments to form a global inference structure; (iii) discovery and incorporation of relevant information sources, that are not known prior to operation, during runtime; (iv) correct computation of posterior distributions without synchronization of partial fusion processes and compilation of global inference structures.

7.2 Comparison of Distributed Inference Approaches

Next to DPNs, we have briefly described two well known approaches to distributed inference in Bayesian networks, namely Multiply Sectioned Bayesian Networks and Prior/Likelihood Decomposable Models. These three different approaches were compared by focusing on different application aspects. It turns out that each of the compared approaches has advantages and disadvantages with respect to these different application aspects. DPNs support a limited class of models, while they have self organization capabilities. Consequently, they can easily adapt to changing constellations of information sources and dynamic agent systems at runtime. MSBNs and PLDMs, on the other hand, can model very complex domains. However, the modeling flexibility requires computationally expensive preprocessing prior to the operation. Therefore, these approaches are not suitable for domains where constellations of information sources and processing nodes change frequently at runtime.

These conclusions were obtained through analysis with respect to the following relevant application aspects:

Problem domains The investigated approaches to distributed inference are typically designed for specific problem domains. MSNBs are designed to deal with very complex problem domains, while DPNs and PLDMs are more focusing on problem domains where the agent constellations must change to capture relevant or available information sources. DPNs support problem domains where evidence needs to be processed sequentially with high frequency and where no globally consistent models are required. The problem domains associated with MSBNs and PLDMs require computation of the joint system potential which in turn requires globally consistent models and prior knowledge of information sources. However, in PLDMs the correct posteriors can be approximated by computing partial beliefs.

Modeling complexity The modeling complexity is directly related to the number of variables in separator sets. The more variables are allowed in such sets the higher the complexity and connectedness of distributed BNs can be. DPNs allow only one variable in its separator set and therefore it cannot deal with highly connected BNs. MSBNs and PLDMs allow more variables in their separator sets and therefore these approaches can deal with highly complex and connected models.

Configuration and compilation The three discussed approaches to distributed inference all use a compiled version of the underlying probabilistic DAG structure. Compilation in MSBNs and PLDMs first require the local BNs to be configured, that is, all the local BNs must be connected to each other to form the global inference structure. After configuration, a distributed cooperative compilation algorithm has to be executed on the global BN structure. The compilation algorithm for MSBNs is quite expensive with respect to communication and sensitive to communication failures. Also, all compilation steps have to be executed in sequence. For PLDMs the compilation algorithm is robust against communication failures. In addition, the configuration and compilation algorithms can be executed simultaneously. PLDMs require a priori knowledge about the dependencies between agents in the form of a Markov network. DPNs supports precompilation of the local BNs before configuration and no distributed cooperative compilation algorithm is required.

Self configuration and dynamic agent systems DPNs and PLDMs facilitate dynamic agent systems (see section 6.4) and self configuration. The restrictions on the topology of DPNs allow to connect unknown information sources to the global inference structure while in PLDMs all information sources must be known a priori. In MSBNs dynamic agent systems and self configuration are difficult because of the expensive and sensitive compilation algorithm. Beside that, the separator sets between local BNs can be arbitrarily large which make them difficult to connect without high coordination between designers.

Exact propagation The inference algorithms of the three approaches can be classified into (i) algorithms that focus on the computation of the joint system potential and (ii) algorithms that focus on the computation of correct posteriors over a subset of variables modeled in one agent. MSBNs and PLDMs use an inference algorithm of the first class and DPNs of the second class. Inference algorithms of class (i) are computationally more expensive than algorithms from class (ii), but support computation of correct posteriors of every variable.

Frequent evidence processing Large MSBNs are not suited to process high frequency of observations sequentially, because global consistency is required to compute correct posteriors (see section 6.6.2). DPNs and PLDMs this problem is relaxed. DPNs do not compute globally consistent distributions over all variables (note, they compute globally consistent distributions over a few variables of interest), which in turn means less communication and no synchronization. PLDMs use an inference algorithm that has the partial correctness property where the computed posteriors are, after the robust message passing algorithm converges, an approximation to the actual posteriors.

A Examples

In this appendix we describe a few examples that clarifies the concepts discussed in previous chapters.

A.1 Visit to Asia

In order to clarify BNs let us take a look at an example called the 'Visit to Asia' example. The following text (from [7]) describes the medical problem domain for this example:

> Tuberculosis and lung cancer can cause shortness of breadth (dyspnoea) with equal likelihood. The same is true for a positive chest X-Ray (i.e. positive chest X-Ray is also equally likely given either tuberculosis or lung cancer). Bronchitis is another cause of dyspnoea. A recent visit to Asia increases the likelihood of tuberculosis, while smoking is a possible cause of both lung cancer and bronchitis.

Given the description of the problem domain we can set up the Bayesian network given in figure A.1.

Every variable in the 'Visit to Asia' example has only two states. For example 'Does the person smokes?' can be answered with a 'yes' or a 'no'. This means that there are $2^8 = 256$ values to specify for the full JPD in this example.

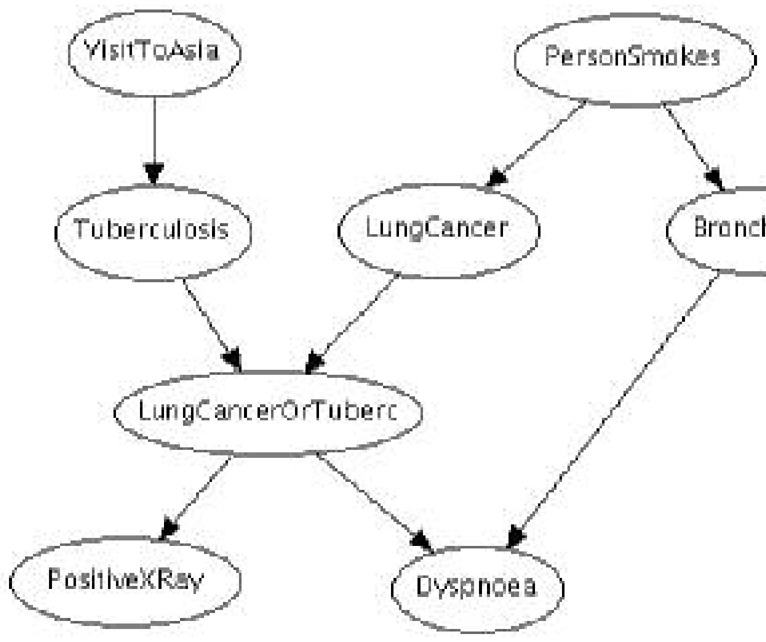
All the arrows shown in this example have attached CPTs. These CPTs have to be specified in advance by a *domain expert* that is familiar with the medical domain, for example a doctor*. Next to this the variables `VisitToAsia` and `PersonSmokes` have prior probabilities that also have to be specified by the domain expert. If we know that a person has visited Asia in the last month and has dyspnoea we can infer the likelihood that this person has tuberculosis given the causal model in figure A.1 through belief updating. In other words, we are calculating the posterior probability of a person having tuberculosis given the observations that the person visited Asia and has dyspnoea.

A.2 Evidence Updating

Let us assume that we have the simple BN given in figure A.2 with binary

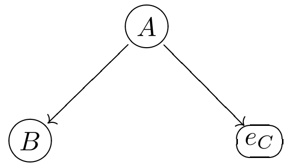
* Acquisition of CPT values can also done through batch learning, tuning and adaptation of the BN. For more details about learning algorithms for BN consider [12, 6]

Figure A.1: Visit to Asia Bayesian Network



variables and with the observation e_C where the probability of the state c_1 is one, hence $P(C = c_1) = 1$.

Figure A.2: A Bayesian Network with observation e_C



Probabilities are often converted to potentials to avoid normalization after each computation. Table A.1 shows the associated potentials $\phi_1(A)$, $\phi_2(A, B)$ and $\phi_3(A, C)$ for the probabilities $P(A)$, $P(B|A)$ and $P(C|A)$. According to definition 2.5 we can calculate $P(A)$ by[†]

$$P(A) = P(A) \sum_B P(B|A) \sum_C P(C|A) \tag{A.1}$$

This is identical to the potential multiplication:

$$\phi_1(A) = \phi_1(A) \sum_B \phi_2(A, B) \sum_C \phi_3(A, C)$$

As already stated earlier using potentials instead of probabilities avoids normalization after multiplication.

Table A.1: Potentials of the BN in figure A.2

A	$\phi_1(A)$	(A, B)	$\phi_2(A, B)$	(A, C)	$\phi_3(A, C)$
a_1	0.4	(a_1,b_1)	0.2	(a_1,c_1)	0.1
a_2	0.6	(a_1,b_2)	0.8	(a_1,c_2)	0.9
		(a_2,b_1)	0.3	(a_2,c_1)	0.3
		(a_2,b_2)	0.7	(a_2,c_2)	0.7

If we want to calculate the posterior belief $P(A|\mathcal{E})$ where $\mathcal{E} = \{e_C\}$ is the evidence set, we can multiply, for example, the potential $\phi_3(A, C)$ with the

[†] Keep in mind that multiplication of probabilities and potentials is not identical to vector/matrix multiplication. For more details about mathematical operations on potentials consult section 2.1.4, definition 2.2

evidence vector $e_C = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ which will result in the following potential given in table A.2:

Table A.2: Potential of $\phi'_3(A,C) = \phi_3(A,C) \cdot P(E_C)$

(A,C)	$\phi'_3(A,C)$
(a_1,c_1)	0.1
(a_1,c_2)	0
(a_2,c_1)	0.3
(a_2,c_2)	0

$\phi'_3(A,C)$ can be multiplied with either potential $\phi_1(A)$ or $\phi_2(A,B)$. In table A.3 $\phi'_3(A,C)$ is multiplied with $\phi_2(A,B)$. Notice that this multiplication results in a new potential $\phi_4(A,B,C)$ defined over the domain $\mathcal{D}_{\phi_4} = \{(a_1,b_1,c_1), (a_2,b_1,c_1), \ldots\}$.

Table A.3: Potential of $\phi_4(A,B,C) = \phi'_3(A,C) \cdot \phi_2(A,B)$

(A,B,C)	$\phi_4(A,B,C)$	(A,B,C)	$\phi_4(A,B,C)$
(a_1,b_1,c_1)	0.02	(a_2,b_1,c_1)	0.09
(a_1,b_1,c_2)	0	(a_2,b_1,c_2)	0
(a_1,b_2,c_1)	0.08	(a_2,b_2,c_1)	0.21
(a_1,b_2,c_2)	0	(a_2,b_2,c_2)	0

The remaining computation to calculate $P(A,e_C) = \alpha \cdot (\phi_4(A,B,C) \cdot \phi_1(A))^{\downarrow\{A\}}$. First we multiply $\phi_4(A,B,C)$ with $\phi_1(A)$ which result in the potential $\phi'_4(A,B,C)$ given in table A.4

Table A.4: Potential of $\phi'_4(A,B,C) = \phi_4(A,B,C) \cdot \phi_1(A)$

(A,B,C)	$\phi'_4(A,B,C)$	(A,B,C)	$\phi'_4(A,B,C)$
(a_1,b_1,c_1)	0.008	(a_2,b_1,c_1)	0.054
(a_1,b_1,c_2)	0	(a_2,b_1,c_2)	0
(a_1,b_2,c_1)	0.0320	(a_2,b_2,c_1)	0.210
(a_1,b_2,c_2)	0	(a_2,b_2,c_2)	0

where the projection of $\phi'_4(A,B,C)^{\downarrow\{A\}}$ yields the updated potential $\phi'_1(A) = \begin{pmatrix} 0.04 \\ 0.18 \end{pmatrix}$ that can be normalized to calculate $P(A|e_C) = \begin{pmatrix} 0.1818 \\ 0.8182 \end{pmatrix}$.

According to commutative law of theorem 2.2 it does not matter in which order $P(A|e_C)$ will be calculated. However, some multiplication orders will be more efficient than others. In the used example we saw that a new potential $\phi'_4(A,B,C)$ with 8 entries had to be introduced to calculate the posterior $P(A|e_C)$. This computational inefficient potential was unnecessary and can be avoided by eliminating variables through an appropriate sequence of projections.

If we investigate the BN given in figure A.2 again we can see that only variable C has received hard evidence. This means that the formula given in equation A.1 can be simplified, because B is a *barren node* where $\sum_B P(B|A) = 1$.

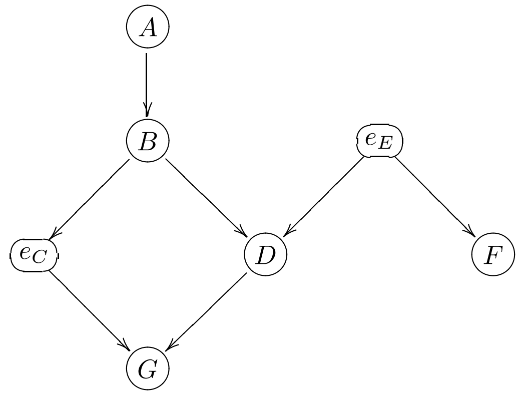
$$P(A) = P(A) \sum_C P(C|A) \tag{A.2}$$

This shows us that the potential $\phi_2(A,B)$ is not needed in the computation of $P(A)$ which means that only 4 entries instead of 8 were required to do the computation.

A.3 D-Separation

To clarify how d-separation works in a larger BN let's consider the BN depicted in figure A.3. In this example we have the following hard evidence observations e_C and e_E for variables C and E, respectively.

Figure A.3: Example Bayesian network with observations

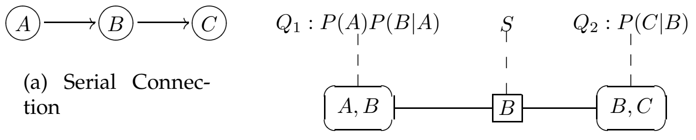


From definition 2.10 we know that the diverging connection $D \leftarrow E \rightarrow F$ d-separates variable D from variable F. Also in the serial connection $B \rightarrow C \rightarrow G$ we can see that variable B and G are d-separated, but because we have also the serial connection $B \leftarrow D \leftarrow G$ where variable D is not instantiated B and G are *not* d-separated. So, in this example only variables $\{E, F\}$ are d-separated from the rest of the network through variable D. This can also be written as $\langle\{A, B, G, D\}|\{C, E\}|\{F\}\rangle$ which means that the set of variables $\{A, B, G, D\}$ and $\{F\}$ is d-separated given the set $\{C, E\}$. Notice that variables C and D are not d-separated by variable G, because there is also the diverging connection $C \leftarrow B \rightarrow D$.

A.4 Absorption

In this example, given in figure A.4, absorption is explained and how a JT can be made globally consistent through concise message passing.

Figure A.4: Junction Tree of a serial connection



(b) JT

In this example all the variables have binary states, namely 'y' or 'n'. Table A.5 lists the values for all the potentials that are defined in figure A.4.

Table A.5: Potentials of JT in figure A.4 (b)

(A,B)	$\phi_{Q_1}(A,B)$	B	$\phi_S(B)$	(B,C)	$\phi_{Q_2}(B,C)$
(y,y)	0.03	y	1	(y,y)	0.05
(y,n)	0.44	n	1	(y,n)	0.33
(n,y)	0.38			(n,y)	0.23
(n,n)	0.76			(n,n)	0.45

From table A.5 we can verify if the two clusters Q_1 and Q_2 are consistent by calculating:

$$\begin{aligned} \sum_{Q_1\backslash Q_2} \phi_{Q_1}(A,B) &\neq \sum_{Q_2\backslash Q_1} \phi_{Q_2}(B,C) \\ \begin{pmatrix} 0.03+0.38 \\ 0.44+0.76 \end{pmatrix} &\neq \begin{pmatrix} 0.05+0.33 \\ 0.23+0.45 \end{pmatrix} \\ \begin{pmatrix} 0.41 \\ 1.20 \end{pmatrix} &\neq \begin{pmatrix} 0.38 \\ 0.68 \end{pmatrix} \end{aligned}$$

Clearly clusters Q_1 and Q_2 are not consistent with each other, which means that the JT is not locally consistent and this in turn implies that the JT is not globally consistent. In order to make the two clusters consistent the algorithm `Absorption` given in algorithm 3.1 has to be performed twice. First, we are going to perform absorption from Q_2 to Q_1 or in other words Q_1 absorbs Q_2. This means that the updated separator potentials have to be calculated by using the first step of algorithm 3.1.

$$\begin{aligned}\phi'_{S_{1\leftarrow 2}}(B) &= \sum_{Q_2\setminus S}\phi_{Q_2}(B,C)\\ &= \begin{pmatrix}0.05+0.33\\0.23+0.45\end{pmatrix}\\ &= \begin{pmatrix}0.38\\0.68\end{pmatrix}\end{aligned}$$

In the second step of algorithm 3.1 the potential $\phi_{Q_1}(A,B)$ is updated.

$$\begin{aligned}\phi'_{Q_1}(A,B) &= \phi_{Q_1}(A,B)\cdot\frac{\phi'_{S_{1\leftarrow 2}}(B)}{\phi_S(B)}\\ &= \begin{pmatrix}0.03 & 0.44\\0.38 & 0.76\end{pmatrix}\cdot\frac{\begin{pmatrix}0.38\\0.68\end{pmatrix}}{\begin{pmatrix}1\\1\end{pmatrix}}\\ &= \begin{pmatrix}0.0114 & 0.2992\\0.1444 & 0.5168\end{pmatrix}\end{aligned}$$

All the current values are given in table A.6.

Table A.6: Potentials of JT in figure A.4 (b) after Q_1 absorbs Q_2

(A,B)	$\phi_{Q_1}(A,B)$	B	$\phi_S(B)$	(B,C)	$\phi_{Q_2}(B,C)$
(y,y)	0.0114	y	0.38	(y,y)	0.05
(y,n)	0.2992	n	0.68	(y,n)	0.33
(n,y)	0.1444			(n,y)	0.23
(n,n)	0.5168			(n,n)	0.45

The two clusters are still not consistent with each other.

$$\begin{aligned}\sum_{Q_1\setminus Q_2}\phi_{Q_1}(A,B) &\neq \sum_{Q_2\setminus Q_1}\phi_{Q_2}(B,C)\\ \begin{pmatrix}0.0114+0.1444\\0.2992+0.5168\end{pmatrix} &\neq \begin{pmatrix}0.05+0.33\\0.23+0.45\end{pmatrix}\\ \begin{pmatrix}0.1558\\0.8160\end{pmatrix} &\neq \begin{pmatrix}0.38\\0.68\end{pmatrix}\end{aligned}$$

In order to get the clusters consistent the absorption algorithm has to be performed in both directions. That means the adsorption from Q_1 to Q_2 also has to be performed. For the separator $\phi''_{S_{2\leftarrow 1}}(B)$ we get:

$$\begin{aligned}\phi''_{S_{2\leftarrow 1}}(B) &= \sum_{Q_1\setminus S}\phi_{Q_1}(A,B)\\ &= \begin{pmatrix}0.0114+0.1444\\0.2992+0.5168\end{pmatrix}\\ &= \begin{pmatrix}0.1558\\0.8160\end{pmatrix}\end{aligned}$$

and for the potential $\phi'_{Q_2}(B,C)$:

$$\begin{aligned}
\phi'_{Q_2}(B,C) &= \phi_{Q_2}(B,C)\cdot\frac{\phi'_{S_{2\leftarrow 1}}(B)}{\phi'_{S_{1\leftarrow 2}}(B)} \\
&= \begin{pmatrix} 0.05 & 0.33 \\ 0.23 & 0.45 \end{pmatrix}\cdot\frac{\begin{pmatrix} 0.1558 \\ 0.8160 \end{pmatrix}}{\begin{pmatrix} 0.38 \\ 0.68 \end{pmatrix}} \\
&= \begin{pmatrix} 0.0205 & 0.1353 \\ 0.2760 & 0.5400 \end{pmatrix}
\end{aligned}$$

For the calculation of $\phi'_{Q_2}(B,C)$ we can see that we must divide through $\phi'_{S_{1\leftarrow 2}}(B)$. If we would ignore this division we are actually propagating the information gathered from cluster Q_2 back to itself. Dividing by $\phi'_{S_{1\leftarrow 2}}(B)$ removes the gathered information from cluster Q_2 and the multiplication with $\phi'_{S_{2\leftarrow 1}}(B)$ only incorporates the information from cluster Q_1.

After absorption is performed in both directions the clusters are consistent with each other.

$$\begin{aligned}
\sum_{Q_1\setminus Q_2}\phi_{Q_1}(A,B) &= \sum_{Q_2\setminus Q_1}\phi_{Q_2}(B,C) \\
\begin{pmatrix} 0.0114+0.1444 \\ 0.2992+0.5168 \end{pmatrix} &= \begin{pmatrix} 0.0205+0.1353 \\ 0.2760+0.5400 \end{pmatrix} \\
\begin{pmatrix} 0.1558 \\ 0.8160 \end{pmatrix} &= \begin{pmatrix} 0.1558 \\ 0.8160 \end{pmatrix}
\end{aligned}$$

In table A.7 all the current values of the JT are given.

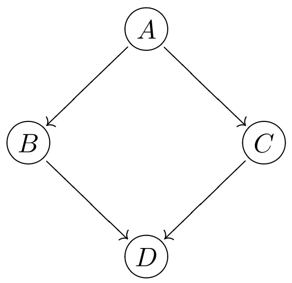
Table A.7: Potentials of JT in figure A.4 (b) after Q_1 absorbs Q_2 and Q_2 absorbs Q_1

(A,B)	$\phi_{Q_1}(A,B)$	B	$\phi_S(B)$	(B,C)	$\phi_{Q_2}(B,C)$
(y,y)	0.0114	y	0.1558	(y,y)	0.0205
(y,n)	0.2992	n	0.8160	(y,n)	0.1353
(n,y)	0.1444			(n,y)	0.2760
(n,n)	0.5168			(n,n)	0.5400

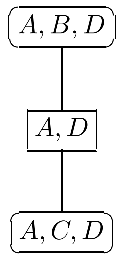
A.5 Independence Map

In figure A.5 (a) a BN G is depicted. According to definition 2.10 we can say that $\langle B|A|C\rangle_G$ is true, because when variable A is observed B and C are d-separated, but notice that $\langle B|\{A,D\}|C\rangle_G$ is not true. This is because variable D is observed and makes B and C dependent on each other. Given the d-separation of G we want to know if the graph in figure A.5 (b) is an I-map. We can see that $\langle B|A|C\rangle_G$ is true and that should imply $I(B,A,C)$. This is not the case in figure A.5 (b) because in this figure variable B is only d-separated from C if variables A and D are instantiated, hence $I(B,\{A,D\},C)$. $\langle B|\{A,D\}|C\rangle_G$ was not true in G and therefore figure A.5 (b) is not an I-map for graph G. The graph figure A.5 (c) however is an I-map for graph G.

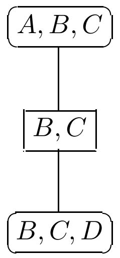
Figure A.5: Example I-map



(a) BN



(b) Not an I-map



(c) I-map

B Why BNs Encode Conditional Independence?

Why does d-separation occur in the three possible connections of a BN in section 2.2.3? It can be shown mathematically by using the examples given in figure 2.3 and figure 2.4 for the serial, diverging and converging connection. First we show it for the serial connection.

If we want to calculate the joint probability $P(A, B, C)$ then the chain rule can be used.

$$P(A, B, C) = P(A)P(B|A)P(C|B) = P(A, B)P(C|B)$$

We want to show that $P(C|A, B) = P(C|B)$. This can be shown by rewriting the above equation:

$$P(C|B) = \frac{P(A, B, C)}{P(A, B)} = \frac{P(A, B)P(C|A, B)}{P(A, B)} = P(C|A, B)$$

For the diverging connection a similar way of rewriting can be used. What about the converging connection? The joint probability of that example can be written as

$$P(A, B, C) = P(A)P(B)P(C|A, B)$$

We want to show is that $P(A|C) = P(A)$ which is similar to proving that $P(A, B) = P(A)P(B)$.

The chain rule of example in figure 2.4 looks like:

$$P(A, B, C) = P(A)P(C)P(B|A, C)$$

Because the distributive law holds in probability multiplications (see [6]), we have

$$\begin{aligned} P(A,C) &= \sum_B P(A)P(C)P(B|A,C) \\ &= P(A)P(C)\sum_B P(B|A,C) \\ &= P(A)P(C) \end{aligned}$$

Variable A and C are independent if we do not know variable B. In other words the variable B is a barren node, because $\sum_B P(B|A,C) = 1$ and makes variables A and C marginally independent.

The above mathematical equations show that d-separation is encoded into the chain rule.

C Building a Junction Tree

Compiling a BN into a JT requires three steps,namely:

- moralization
- triangulation
- organizing cliques into a JT

Let's discuss each step in turn.

C.1 Moralization of a BN

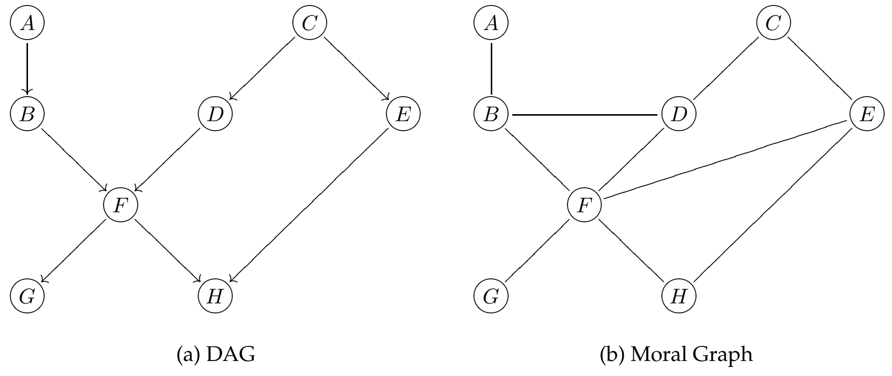
The first step in organizing a BN into a JT is moralization. Let us explain the process of moralization through an example given in figure C.1 which is the example 'Visit to Asia' from appendix A.1. All the variables in this example are replaced by stochastic variables $\{A, B, C, D, E, F, G, H\}$. From figure C.1 (a) we can say that $\langle D|C|E\rangle$ implies $I(D|C|E)$ is valid. Also $\langle C|\{D, F\}|G\rangle$ implies $I(C, \{D, F\}, G)$ holds. However we can not say that $\langle B|\emptyset|D\rangle$ implies $I(B, \emptyset, D)$, because when we know F, variables B and D are dependent on each other. This is also called *induced dependence* and needs to be encoded into the undirected representation of the BN. We know that $\langle B|\emptyset|D\rangle$ and $\neg\langle B|F|D\rangle$ can not be shown simultaneously in an undirected graph. Because we can have an induced dependence between B and D we must connect them by adding an undirected link. This can also be seen as negating $\langle B|F|D\rangle$. If we do the same with F and E and drop the direction of all the arrows the resulting graph is called a *moral graph* given in figure C.1 (b). Given definition 3.3 of u-separation a new probabilistic dependency structure is created.

It is proven that a moral graph of a DAG is a minimal I-map (see [21]).

C.2 Triangulation

We know from definition 3.3 that pairwise connected nodes do not allow graphical separation. A set of nodes is called *complete* if they are all pairwise connected. A maximal set of nodes that is complete is called a *clique*. Every clique in an undirected graph has to be organized into a cluster of a JT, because the set of variables in a clique do not allow graphical separation. It turns out that whenever a JT can be computed from an undirected graph, this graph has to be *triangulated* in order to identify cliques that can be organized into a JT.

Figure C.1: The moralized graph of the visit to Asia DAG



Definition C.1 (Clique) *A clique is a set of variables $\mathcal{V}$ that form a maximal complete set based on a given undirected graph G. The set of variables $\mathcal{V}$ do not allow any graphical separation.*

A graph that is triangulated is called a *chordal graph* or just simply *triangulated graph*. A graph G is chordal or triangulated if every cycle ρ of length bigger than or equal to 4 has a *chord*. This means that two non adjacent nodes on ρ have a link between them. If we look at the moral graph of figure C.1 (b) we can see that the cycle $F-E-C-D-F$ is a chordless cycle which means that the moral graph is not a chordal graph. To make this graph chordal we need to add a *fill-in* to make the cycle $F-E-C-D-F$ a cycle with a chord. In this case only one chord will suffice to make the cycle chordal. However, it can happen that multiple fill-ins have to be added before all chordless cycles are removed. In order to find these fill-ins we can use the technique *triangulation by elimination* to make a graph triangulated.

Before we are going to use the technique 'triangulation by elimination' we first want to know if some graph is triangulated. There is a simple procedure in order to see if a graph has that property. If we have a graph G with nodes $\mathcal{V}$ we can search the nodes for a *simplicial* node, that is a node where its adjacent nodes are complete. If found, then delete this node and try to find another simplicial node. If all nodes can be eliminated graph G is called *eliminatable* and the graph is chordal. If no simplicial node can be found in V then G is not a chordal graph. In algorithm C.1 this procedure is summarized.

If graph G is not chordal it can be triangulated by node elimination. Let us consider the example in figure C.1 (b) again. We start by eliminating all simplicial nodes until no simplicial nodes can be found. In this case that means that we first eliminate node G[1]. This is followed by A. Next, B can be eliminated because F and D are in the same complete set. Finally we can delete G. We end up with the chordless cycle $F-E-C-D-F$. Say, we want to eliminate F. This means that the nodes D and E need to be connected by introducing a fill-in. F can be eliminated and finally the rest of the nodes can be eliminated as well. By introducing a fill-in between D and E the moral graph given in figure C.1 is a chordal graph. The resulting chordal graph is given in figure C.2.

The example used in figure C.1 is an example of an unlucky moral graph which is not a chordal graph. However, it can be the case that a moralized BN is already a chordal graph. In such cases no triangulation is necessary in order to find its JT. In the case were a moral graph is not a chordal graph the number of fill-ins that are needed in order to make the graph eliminatable should be minimal. When fill-ins are added to a graph some of the present

[1] Note that the procedure **triangulation by elimination** is an ambiguous process

Algorithm C.1: Checking if graph is chordal

```
procedure: IsChordal($G$)
input     : Graph $G = (V, E)$
output    : True or false
/* Given a graph $G$ with a set of nodes $V$.  Repeat the
   following steps until no nodes from $V$ can be
   eliminated.                                            */
for $i = 1$ to $|V|$ where $N_i \in V$ do
    Search for $N_i$ that is simplicial;
    if simplicial node found then
        eliminate simplicial node;
    else
        $G$ is not chordal;
    end
end
if all nodes are eliminated then
    $G$ is chordal;
else
    Go to step 1;
end
```

Algorithm C.2: Converting graph to chordal graph

```
procedure: GetChordalGraph($G$)
input     : Graph $G = (V, E)$
output    : Chordal graph $G' = (V, E \cup F)$
/* Given a graph $G$ with a set of nodes $\mathcal{V}$ and a set of
   edges $\mathcal{E}$.                                        */
Set $F = \emptyset$;
for $i = 1$ to $|V|$ where $N_i \in V$ do
    Search for $N_i$ that is simplicial;
    if found simplicial node then
        eliminate simplicial node $N_i$;
    else
        Select a node $N_j$ to eliminate and add the necessary fill-ins to $F$;
    end
end
Return $G' = (V, E \cup F)$;
```

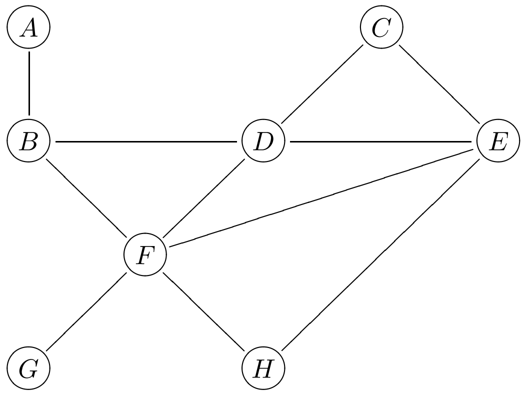


Figure C.2: A Chordal graph

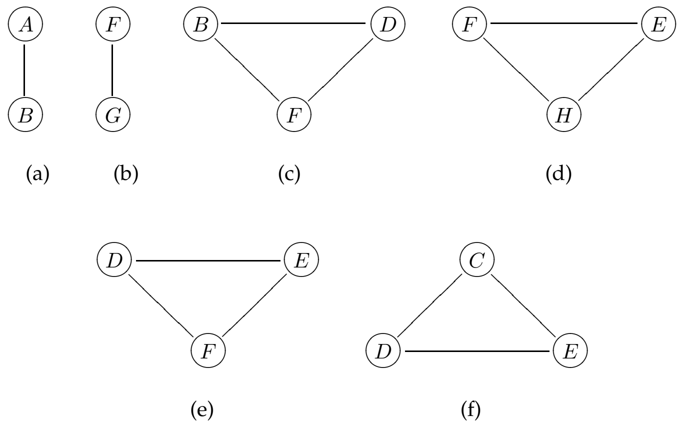
conditional independence is hidden which in turn makes belief updating less efficient.

C.3 Organizing cliques into a JT

In the last step of the compilation process of organizing a BN into a JT is finding the cliques that form clusters in the JT. There are different algorithms to organize the found cliques into a JT. In this thesis the procedure from [21] is followed.

In figure C.2 we can identify the following cliques: $\{A, B\}$, $\{G, F\}$, $\{B, D, F\}$, $\{F, E, H\}$, $\{F, D, E\}$ and $\{D, E, C\}$ given in figure C.3. These cliques are the maximum complete sets in figure C.2

Figure C.3: Cliques of 'Visit to Asia' DAG

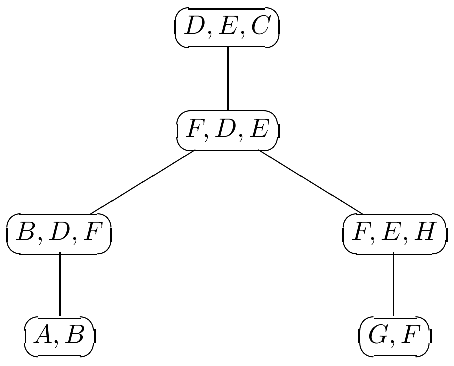


How should the cliques be connected to form a JT? One clique is selected, say $\{B, D, F\}$ then we can determine the separators with other cliques. The clique with the highest separator weight will be connected, where the separator weight $w(S)$ is defined as follows:

$$w(S) = |S| \tag{C.1}$$

The intersection of $\{B, D, F\}$ with $\{A, B\}$, $\{G, F\}$, $\{F, E, H\}$, $\{F, D, E\}$ and $\{D, E, C\}$ results in the following separators: $\{B\}$, $\{F\}$, $\{F\}$, $\{F, D\}$ and $\{D\}$ respectively with the corresponding weights: 1, 1, 1, 2 and 1 respectively. This means that $\{B, D, F\}$ is connected with $\{F, D, E\}$. Another clique can be connected to $\{B, D, F\}$ or $\{F, D, E\}$ with also a maximum separator weight. The final structure is called a *maximum cluster tree* and corresponds to a JT (See figure C.4 for a possible JT). The reason why it is called a maximum cluster tree is because $w(T) = \sum_i w(S_i)$ is maximum, where T is the JT and S_i are the separators of this JT. In [21] it is proven that the found maximum cluster tree is always a JT.

Figure C.4: A Junction Tree of 'Visit to Asia' DAG



In figure C.4 we can see that any intersection of two cliques will result in a set that is contained in every cluster on the path between these two clusters. For example, the intersection between $\{B, D, F\}$ and $\{F, E, H\}$ results in the set $\{F\}$ which is contained in the cluster $\{F, D, E\}$. The cluster tree in figure C.4 has the running intersection property and therefore it is a JT.

D Sensor Data Simulation

In this appendix we show how sensor outputs can be simulated for the observation nodes in figure D.1 (like $\mathtt{S1}_i$, $\mathtt{C1}_i$, $\mathtt{C2}_i$).

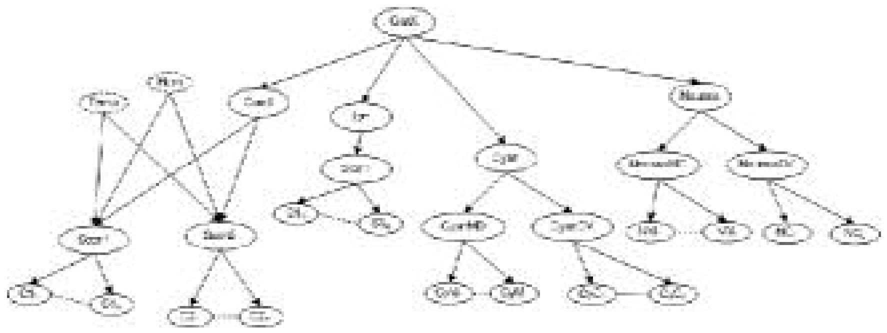


Figure D.1: The used network for experiments

In order to generate sensor observations in the causal model of `GasX` a causal process can be simulated by instantiating the root `GasX` concept. In other words, a lethal concentration of toxic gas is present or absent. The children of `GasX`, which are variables `Cond`, `Ion`, `Cyan` and `Nausea`, can also be instantiated by considering the corresponding CPT $P(\mathtt{Cond}|\mathtt{GasX})$. For example, if we want to instantiate node `Cond` we use the CPT shown in table D.1. If `GasX` is instantiated to true, (i.e. the ground truth of the presence of `GasX` is true) then we can select the column in which `GasX` is true and generate a value for `Cond` according to the distribution $(0.7, 0, 3)$. This means that in 70% of the cases node `Cond` is instantiated to true, which corresponds to a conductivity that indicated the presence of `GasX`, and in 30% of the cases to false, which corresponds to a conductivity that indicated the absence of `GasX`. After `Cond` is instantiated we can do the same for the children of `Cond`, which are `Scon1` and `Scon2`. This process of instantiating nodes is repeated all the way down to the leaf nodes $\mathtt{S1}_i$, $\mathtt{C1}_i$, $\mathtt{C2}_i$, etc. The generated values for the leaf nodes can be used as simulated outputs for the information sources in figure 4.4.

Table D.1: The CPT of $P(\mathtt{Cond}|\mathtt{GasX})$

Cond\GasX	GasX = true	GasX = false
Cond = true	0.7	0.1
Cond = false	0.3	0.9

In section 4.1.1 two type of nodes were discussed: quasi-static and dynamic nodes. In figure D.1 the leaf nodes are all dynamic while the other variables are quasi-static. Therefore, generating a value for a dynamic node must be regenerated every time a new value is required. However, for quasi-static nodes a value is only generated once, because the events, corresponding to these concepts, do not change for a period of time. In other words, quasi-static nodes are all instantiated once, while values for dynamic nodes are re-instantiated, by considering the associated CPT, every time a new value is needed.

E List of Notations

Below a list of frequently used terms and notations throughout this thesis is given. In addition, a separate list of notations is given for every distributed inference approach. Some terms and notations are ambiguous, but its meaning is always clear from the context.

V	An event
v	A state of event V
$P(A)$	Probability distribution over stochastic variable A
$P(a)$	Probability of the state a
$P(A\|B)$	Conditional probability of event A given event B
$P(A, B)$	The joint probability of event A and B
$\phi(A, B)$	A potential defined over variables A and B
$\mathcal{G}$	A set of graphs $\{G_1, \ldots, G_n\}$
G_i	A DAG i
$\mathcal{V}$	A set of variable sets $\{\mathcal{V}_1, \ldots, \mathcal{V}_n\}$
$\mathcal{V}_i$	A set of variables $\{V_1, \ldots, V_n\}$
$\mathcal{A}$	A set of agents $\{A_1, \ldots, A_n\}$
A_i	An agent i
$\mathcal{Q}$	A set of clusters $\{Q_1, \ldots, Q_n\}$
Q_i	A cluster i with stochastic variables $\{V_1, \ldots, V_n\}$
$\mathcal{E}$	A set of evidence $\{e_{V_1}, \ldots, e_{V_n}\}$
e_V	Hard evidence for variable V
$\overline{e}_V$	Soft evidence for variable V
$\sum_B P(A, B)$	Summation (marginalization) over variable B
$\prod_i \phi(Q_i)$	A factorization over all potential Q_i
$P(A, B)^{\downarrow A}$	Projection down to variable A
$\mathcal{D}_X$	The domain $\{x_1, \ldots, x_n\}$ of variable X
$\mathcal{V}_i \cup \mathcal{V}_j$	The union of variable sets $\mathcal{V}_i$ and $\mathcal{V}_i$
$\mathcal{V}_i \cap \mathcal{V}_j$	The intersection of variable sets $\mathcal{V}_i$ and $\mathcal{V}_i$
$\mathcal{V}_i \backslash \mathcal{V}_j$	The variables in $\mathcal{V}_i$ that are not in $\mathcal{V}_j$
$V \in \mathcal{V}$	V is an element of $\mathcal{V}$
$\|\mathcal{V}\|$	The number of elements in $\mathcal{V}$
$\langle \mathcal{X} \| \mathcal{Z} \| \mathcal{Y} \rangle$	Variables in $\mathcal{X}$ are d-separated from variables in $\mathcal{Y}$ given $\mathcal{Z}$
$I(\mathcal{X}, \mathcal{Z}, \mathcal{Y})$	Variables in $\mathcal{X}$ are conditionally independent from variables in $\mathcal{Y}$ given $\mathcal{Z}$
$\pi(V)$	The parents of variable V
$\tau(V)$	The parents of variable V and V itself
α	A normalization constant

DPN

R	root concept or service concept
$\mathcal{L}$	a set of leaf concept or input concepts
ψ	A Bayesian network
$S\langle\mathcal{V}_i, \mathcal{V}_j\rangle$	Separator defined by $\mathcal{V}_i \cap \mathcal{V}_j$

MSBN

$\mathcal{E}$	A set of edges
T	A junction tree
$I\langle\mathcal{V}_i, \mathcal{V}_j\rangle$	Interface I defined by $\mathcal{V}_i \cap \mathcal{V}_j$
$L\langle\mathcal{V}_i, \mathcal{V}_j\rangle$	Linkage L defined by $\mathcal{V}_i \cap \mathcal{V}_j$
$\phi * (\eta)$	Extended linkage potential

PLDM

$\mathcal{M}$	Set of measurement variables
M_i	A measurement variable
$\mathcal{X}$	Set of environment variables
$\mathcal{B}$	Set of bias variables
B	Bias variable
T	True temperature variable
$\mathcal{E}$	A set of edges
$\mathcal{N}$	A set of nodes
π_C	Prior distribution for C
λ_C	Likelihood function
$\langle\pi_C, \lambda_C\rangle$	PL-factor
$\langle\pi_C, \lambda_C\rangle \otimes \langle\pi_D, \lambda_D\rangle$	Combination of two PL-factors
$\bigoplus_S\langle\pi_C, \lambda_C\rangle$	Summary of PL-factor to S

F List of Abbreviations

BN	Bayesian Network
CPD	Conditional Probability Distribution
CPT	Conditional Probability Table
DAG	Directed Acyclic Graph
DBN	Dynamic Bayesian Network
DPN	Distributed Perception Network
JPD	Joint Probability Distribution
JPT	Joint Probability Table
JSP	Joint System Potential
JT	Junction Tree
LJF	Linked Junction Forest
LT	Linkage Tree
MSBN	Mutiply Sectioned Bayesian Network
PLDM	Prior/Likelihood Decomposable Model

Bibliography

[1] G. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. volume 42(2-3), pages 393–405, 1990.

[2] Patrick de Oude, Jan Nunnink, and Gregor Pavlin. Distributed bayesian networks in highly dynamic agent organizations. In *The 17th Belgian-Dutch Conference on Artificial Intelligence*, pages 195–201, Brussels, Belgium, 2005.

[3] Patrick de Oude, Brammert Ottens, and Gregor Pavlin. Information fusion with distributed probabilistic networks. In *Artificial Intelligence and Applications*, pages 195–201, Innsbruck, Austria, 2005.

[4] Morris H. DeGroot and Mark J. Schervish. *Probability and Statistics*. Addison-Wesley, third edition edition, 2002.

[5] Stan Franklin and Art Graesser. Is it an agent, or just a program?:a taxonomy for autonomous agents. In *Proceedings of the Third InternationalWorkshop on Agent Theories, Architectures, and Languages*, pages 21–36. Springer-Verlag, 1996.

[6] Finn V. Jensen. *Bayesian Networks and Decision Graphs*. Springer-Verslag, 2001.

[7] S.L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of Royal Statistical Society*, 50(2), 1988.

[8] V. Lepar and P. P. Shenoy. A comparison of lauritzen-spiegelhalter, hugin and shenoy-shafer architectures for computing marginals of probability distributions. In G. Cooper and S. Moral, editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 328–337. Morgan Kaufmann, 1998.

[9] Anders L. Madsen and Finn V. Jensen. Lazy propagation in junction trees. In *In Proc. of the 14th Conference on UAI*, pages 362–369, San Francisco, CA, 1998.

[10] Paskin Mark and Guestrin Carlos. Robust probabilistic inference in distributed systems. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI-04)*, pages 436–445, Banff, Canada, 2004. AUAI Press.

[11] Paskin Mark, Guestrin Carlos, and Jim McFadden. A robust architecture for inference in sensor networks. In *Fourth International Conference on Information Processing in Sensor Networks (IPSN'05)*, Los Angelos, California, 2005.

[12] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Companies, Inc, 1997.

[13] J. Nunnink and G. Pavlin. Towards robust state estimation with bayesian networks: A new perspective on belief propagation. In *The 9th International Conference on Intelligent Autonomous Systems (IAS-9)*, Tokyo, 2006. to appear.

[14] Mark A. Paskin. *Exploiting Locality in Probabilistic Inference*. PhD thesis, University of California, Berkeley, 2004.

[15] G. Pavlin, M. Maris, and J. Nunnink. An agent-based approach to distributed data and information fusion. In *IEEE/WIC/ACM Joint Conference on Intelligent Agent Technology*, pages 466–470, 2004.

[16] Gregor Pavlin and Patrick de Oude. A mas approach to fusion of heterogeneous information. In *IEEE/WIC/ACM Joint Conference on Intelligent Agent Technology*, Campiegne, France, 2005.

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

[18] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan-Kaufmann, 1988.

[19] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley and Sons, LTD, 2002.

[20] Y. Xiang and V. Lesser. Justifying multiply sectioned bayesian networks. In *Proc. 6th Inter. Conf. on Multi-agent Systems*, pages 349–356, Boston, 2000.

[21] Yang Xiang. *Probabilistic Reasoning in Multiagent Systems: A Graphical Models Approach*. Cambridge University Press, 2002.

[22] Yang Xiang. Comparison of multiagent inference methods in multiply sectioned bayesian networks. volume 33, pages 235–254, 2003.