HYBRID BAYESIAN NETWORKS FOR REASONING ABOUT COMPLEX SYSTEMS

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

Uri N. Lerner
October 2002

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

Daphne Koller
Department of Computer Science
Stanford University
(Principal Adviser)

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

Stephen Boyd
Department of Electrical Engineering
Stanford University

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

Ronald Parr
Department of Computer Science
Duke University

Approved for the University Committee on Graduate Studies:

To my parents and my brother.

Abstract

Many real-world systems are naturally modeled as hybrid stochastic processes, i.e., stochastic processes that contain both discrete and continuous variables. Examples include speech recognition, target tracking, and monitoring of physical systems. The task is usually to perform probabilistic inference, i.e., infer the hidden state of the system given some noisy observations. For example, we can ask what is the probability that a certain word was pronounced given the readings of our microphone, what is the probability that a submarine is trying to surface given our sonar data, and what is the probability of a valve being open given our pressure and flow readings.

Bayesian networks are a compact way to represent a probability distribution. They can be extended to dynamic Bayesian networks which represent stochastic processes. In this thesis we concentrate on hybrid (dynamic) Bayesian networks. Our contributions are three-fold: theoretical, algorithmic, and practical.

From a theoretical perspective, we provide a novel complexity analysis for inference in hybrid models and show that there is a fundamental difference between the complexity of inference in discrete models and in hybrid ones. In particular, we provide the first NP-hardness results for inference in very simple hybrid models. From an algorithmic perspective, we provide a suite of new inference algorithms, designed to deal with non-linearities present in many real-world systems, and to scale up to large hybrid models. We show that our algorithms often outperform the current state of the art inference algorithms for hybrid models. Finally, from a practical perspective, we apply our techniques to the task of fault diagnosis in a complex real-world physical system, designed to extract oxygen from the Martian atmosphere. We demonstrate the feasibility of our approach using data collected during actual runs of the system.

Acknowledgments

First and foremost I would like to thank Daphne Koller. If I had to describe Daphne in one word, excellence would be my choice. Her uncompromising standards make everybody, and mostly herself, work harder, but also bring her students to fulfill their true potential. Looking back I realize how little I knew when I came to Stanford and how much I learned from Daphne during these years: how to find interesting research problems, how to clearly present one's work both in papers and in talks, how to teach a class, and in short how to be a good researcher. I am most grateful to Daphne for bringing me to the point of being proud of my work.

Excluding Daphne, Ron Parr is the person to whom I owe my biggest debt of gratitude. Ron, who was a Research Associate in Daphne's research group, served as my unofficial second adviser during a crucial period of my Ph.D. career, and to a large extent he is responsible for putting me on the track which ultimately led to this thesis. Working closely with Ron was one of the most rewarding and enjoyable experiences of my Ph.D. career. Ron always had the time for a chat, and much of my work can be traced back to these chats. I feel extremely fortunate to have met Ron and to count him as a friend. Thank you Ron!

I would also like to thank Stephen Boyd, the third member of my reading committee, who pointed out to me many interesting directions and connections that turned out to be very useful. I thank Daphne, Ron and Stephen for taking the time to read this long thesis and come up with as many useful comments as they have had. I also thank Ross Shachter and Nils Nilsson who were in my Orals committee.

Working on the RWGS was truly a group effort, and I am privileged to have worked with such an amazing group of people. Among the people at Stanford I would like to thank Brooks Moses who was the only one of us who truly understood the system and was able to come up with a model for it, Sheila McIlraith who brought the problem into my attention and was instrumental in keeping the project on the right track, and Maricia Scott who shared much of the workload with me and was always invaluable during the crucial moments. Brooks, Maricia and I spent many hours on constructing and debugging the model. Their company transformed the experience from a potentially painful one to an enjoyable one.

On the NASA side I would first like to thank Charlie Goodrich whose help went above and beyond any reasonable expectation. Charlie was always there to patiently answer our questions, make sure we have access to the data that we needed, and was even kind enough to host us at his home during our visit to Kennedy Space Center. I could not have hoped for a more helpful or a nicer person than Charlie. I would also like to thank Dan Clancy, Bill Larson, Jon Whitlow, Clyde Parrish, Curtis Ihlfield and Dan Keenan for their help.

Zohar Manna was my adviser during my first year at Stanford, and made my transition from Israel to Stanford much smoother and easier. Zohar let me develop my research interests during my first year, was always there when I needed someone to talk with, and always believed in my abilities. For that I am truly grateful.

Shimon Ullman from the Weizmann Institute of Science in Israel helped me a great deal in my application process and encouraged me to come to Stanford. Since then he was always there to keep track of my progress and give me a good advice when I needed one. I may not have come to Stanford without Shimon's help, and I am in his debt for that.

The algorithms in this thesis were implemented using a code infrastructure called Frog or Phrog. Frog was initially created by Lise Getoor and myself, and I am convinced that had we known what we were getting ourselves into, we would not have done it. However, now that the beast walks among us, it is my hope that the various users of Frog actually benefited from the system, despite all the pain that it caused. I would like to thank Eric Bauer, Xavier Boyen, Maricia Scott, Ben Taskar and Drago Anguelov for their help in writing Frog.

Being a member of DAGS (Daphne's Approximate Group of Student) was a source of both pleasure and pride. Year after year, the research coming out of DAGS is of high quality and high impact, which is a testament to the extraordinary people that were and are members of DAGS, as well as to Daphne's vision and guidance. I would like to thank Drago Anguelov, Eric Bauer, Jenny Berglund, Xavier Boyen, Urszula Chajewska, Barbara Engelhardt, Raya Fratkina, Lise Getoor, Carlos Guestrin, Manfred Jaeger, Alex Kozlov, Brian Milch, Uri Nodelman, Dirk Ormoneit, Ron Parr, Avi Pfeffer, Mehran Sahami, Maricia Scott, Eran Segal, Christian Shelton, Ben Taskar, and Simon Tong.

I would like to thank some other friends and colleagues that I was fortunate enough to get to know along the years: Eyal Amir, Nikolaj Bjørner, Michael Colon, Arturo Crespo, Alon Efrat, Mattan Erez, Bernd Finkbeiner, Nir Friedman, Karl Pfleger, Henny Sipma, and Tomas Uribe.

I would also like to thank my different sources of funding: ONR Young Investigator (PECASE) under grant number N00014-99-1-0464, ARO under the MURI program, "Integrated Approach to Intelligent Systems", grant number DAAH04-96-1-0341, ONR under the MURI program "Decision Making under Uncertainty", grant number N00014-00-1-0637, and NASA under grant number NAG2-1337.

Finally, and above all, I would like to thank my parents, Tamar and Eliahu Lerner whose love and support were always something I could count on, and my brother Assaf who was always so proud of me — it is I who is proud to have you as a brother. This dissertation is dedicated to them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Bayesian Networks

Any intelligent agent must be able to perform reasoning under uncertainty. This is definitely the case for humans: One does not know whether the lottery ticket that one can buy will be a winner; one does not know what the weather will be like in two weeks; one does not know one's current blood pressure and so on.

Probability theory is the method of choice for dealing with uncertainty in many science and engineering disciplines such as control, physics and economics. Probability theory is well understood and seems to model extremely well many phenomena in the physical world. It provides us with a simple and clean semantics of how to maintain our beliefs in face of partial information and how to update them when presented with new evidence. It is therefore somewhat surprising that for many years the AI community did not use probabilistic models. Perhaps the most important reason for this phenomenon was the lack of compact ways to represent probability distributions and to reason with them.

The situation dramtically changed in the 1980s with the introduction of *Bayesian networks* [Pea88]. Bayesian networks represent a probability distribution using a graphical model of a directed, acyclic graph (DAG). Every node in the graph corresponds to a random variable in the domain and is annotated by a *Conditional Probability Distribution (CPD)*, defining the conditional distribution of the variable given its parents.

As we shall see in Chapter 2, Bayesian networks often allow for a compact and natural representation of probability distributions. Combined with a suite of inference and learning algorithms, Bayesian networks, or models based on Bayesian networks, have become a popular tool for reasoning about uncertainty. They have been successfully used as models for a wide variety of complex domains, such as medical diagnosis [MSH+91, PPMH94], determining the needs of software users [HBH+98], predicting the likelihood of meeting attendance [HKKJ02], visual figure tracking [PRCM99], troubleshooting for Microsoft Windows [HBR94], filtering junk email [SDHH98], and music parsing [Rap02].

Perhaps the most fundamental issue for Bayesian networks is the problem of inference. We are given a Bayesian network that represents some probability distribution and are required to answer some probabilistic query, often in the form of computing $P(\boldsymbol{X} \mid \boldsymbol{E} = \boldsymbol{e})$, i.e., the probability distribution over some random variables $\boldsymbol{X}$ given some evidence $\boldsymbol{E} = \boldsymbol{e}$. For example, in medical diagnosis, we can query the probability of a certain disease given the observed symptoms; in the meeting attendance domain, we can query the probability that Alice would attend a meeting given that she is not at her office now and the meeting is scheduled to last for 2 hours; and in the troubleshooting domain, we can query the probability that the printer cable is disconnected given that the printer is not printing.

As one might expect, much of the work on Bayesian networks has focused on the inference problem. The overwhelming majority of this work considers the case of discrete Bayesian networks, i.e., networks that contain only discrete random variables. Much progress has been made on the inference problem, and by now it is fairly well understood for discrete networks.

On a very high level, the work on the inference problem can be divided into two main classes: exact inference and approximate inference. Exact inference algorithms are designed to give an exact answer to the probabilistic query. By now we have a few exact inference algorithms and we understand the relations between them. Furthermore, we have a good understanding of the complexity of exact inference and how it relates to the structure of the network, i.e., the structure of the DAG. In particular we know that for "simple" network structures, exact inference can be very efficient.

Approximate inference is designed to give an approximate answer to the probabilistic query, with the understanding that often the exact probability is not crucial. For example, in the medical diagnosis domain, if the exact probability for cancer given the symptoms is 84.3%, then from a practical point of view, it probably would not matter if our approximate inference algorithm would estimate it as 83.6% or as 85% instead of the exact value. Once again, approximate inference is fairly well understood, although not as well as exact inference. Nonetheless, we have a large set of approximate inference algorithms which are often useful when exact inference is intractable, and a growing understanding of their computational complexity and their performance with respect to various properties of the network.

However, discrete networks are sometimes inadequate, since many important domains have continuous attributes as well as discrete ones. Examples include the visual figure tracking domain, where variables such as location and velocity are continuous, and the music parsing domain, where the features extracted from the audio signal are often continuous. In addition, in this thesis we shall often motivate our work using the domain of fault diagnosis for physical systems. In this domain, we have a model of a physical system and a set of measurements recorded by various sensors, and we are asked to diagnose the system. In probabilistic terms, we need to answer questions such as "what is the probability that sensor S_{17} is malfunctioning given its readings and readings from other sensors?" or "what is the probability that a certain valve is open given the sensor readings?" Obviously, physical systems contain discrete variables such as a valve being open or closed and a pump working or idle, as well as continuous variables such as pressures, flows and temperatures.

One can always discretize the continuous variables by partitioning their domain into some finite number of subsets, and by doing so transform the model to a discrete one. However, as we shall see in Chapter 3, this simple approach is often very problematic and might lead to unacceptable performance. Our approach is different: We treat the continuous variables as continuous without trying to discretize them.

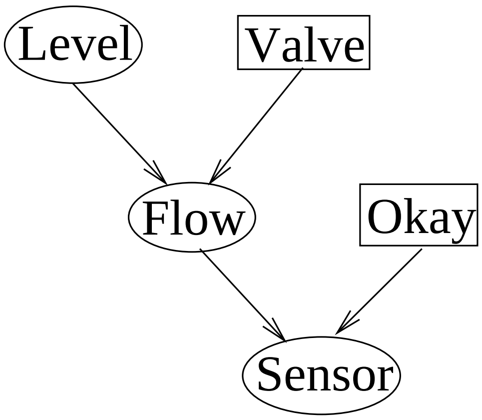


Figure 1.1: A simple example of a hybrid Bayesian network

1.2 A Hybrid Network

Figure 1.1 shows an example of a simple hybrid Bayesian network, i.e., a network that contains both discrete and continuous variables. The discrete variables in the network are represented by rectangles and the continuous variables are represented as ovals. This particular example models the flow of some liquid out of a container. It does so by representing the direct probabilistic dependencies in the domain (we shall define this formally in Chapter 2).

The flow of the liquid, represented by the random variable *Flow*, is influenced by two random variables: *Level*, representing the liquid level in the container, and the discrete variable *Valve*, representing whether the valve that lets the liquid flow out of the container is open or closed. In addition, we have the variable *Sensor* that represents a the value shown on a flow sensor measuring the liquid flow out of the container. Finally, we have the discrete variable *Okay* that represents whether the flow sensor is working properly or not.

As an example for a probabilistic query, assume that we observe that the flow sensor indicates that there is no flow, and we ask what is the probability that the valve is closed. Stated more formally, we are looking for $P(\mathit{Valve}{=}\mathit{Closed} \mid \mathit{Sensor} = 0)$. Note that the position of the valve does not directly influence the flow sensor, but these variables are not independent — if the sensor indicates that there is no flow, there is a higher likelihood that the valve is closed.

As another example, suppose that we are given the additional information that the flow sensor is faulty. Given this information, the position of the valve becomes irrelevant for the measurement — since the sensor is faulty it does not represent the actual flow and therefore we should not use it to make inferences on the position of the valve. Thus, we would expect that:

$$P(\mathit{Valve}{=}\mathit{Closed} \mid \mathit{Sensor} = 0, \mathit{Okay}{=}\mathit{False}) < P(\mathit{Valve}{=}\mathit{Closed} \mid \mathit{Sensor} = 0)$$

It is important to realize that the reasoning in these examples comes directly from the joint probability distribution over the domain variables. In other words, there is no need to create special reasoning rules — all the information that we need is encoded in the joint probability distribution. Thus, if we can represent the probability distribution and answer the probabilistic queries about it, we get this type of sophisticated reasoning "for free" using the probabilistic approach.

1.3 Classification of Hybrid Models

We can classify hybrid models along two axes: the expressive power of the CPDs, and static vs. dynamic models. Along the first axis there are two criteria for the classifications:

- **Does the model allow non-linear relations between the continuous variables?** A linear relation means that the variable X is some linear combination of its parents plus some Gaussian noise. For example, it is reasonable to model the flow sensor in our example of Figure 1.1 as $\mathit{Sensor} = \mathit{Flow} + V$. Here $V \sim \mathcal{N}(0, \sigma^2)$ represents some white Gaussian noise, corresponding to the noise of the sensor. On the other hand, it is obvious that in some cases the true dependencies are non-linear.
- **Does the model allow discrete nodes to have continuous parents?** In our example we do not have such a dependency, but consider a discrete sensor that indicates whether the liquid level is too low (much like the low gas level

warning light in a car). In this case it would be natural to have the discrete sensor variable be influenced by the continuous liquid level variable.

The most popular class of hybrid models are known as *Conditional Linear Gaussians (CLGs)*. These models answer the two questions with a negative answer: CLGs allow only linear relations between the continuous variables and do not allow discrete nodes to have continuous parents. The reason for the popularity of these models is their mathematical convenience. In CLGs, given any assignment to the discrete variables, the distribution over the continuous variables is a multivariate Gaussian. Thus, the joint probability distribution is a mixture of Gaussians, and can be handled using analytical tools. In this thesis we discuss CLGs in great detail, but also devote some chapters to hybrid models that violate the CLG restrictions.

The second axis for the classification is static vs. temporal models. Our example in Figure 1.1 is static, as there is no temporal aspect to the model. However, in many applications we are interested in stochastic processes, rather than static distributions. This is certainly the case for fault diagnosis as we would like to deal with the system over time. For example, we expect to get a sequence of measurements from each sensor and we would like to use past readings as well as current readings for our diagnosis. Bayesian networks can be extended to *Dynamic Bayesian Networks (DBNs)* that represent stochastic processes. Hybrid DBNs that impose the linearity restrictions of CLGs are called *Switching Linear Dynamic Systems (SLDS)*. The later chapters in this thesis discuss hybrid DBNs (both linear and non-linear) and their application for fault diagnosis.

1.4 Contributions

In this thesis we concentrate on the inference problem for hybrid networks from three different points of view: a theoretical point of view, an algorithmic point of view, and a practical point of view.

From a theoretical point of view, very little is known about hybrid Bayesian networks, beyond the obvious fact that by being a generalization of discrete Bayesian networks, they must be at least as hard. In this thesis, we provide a novel complexity analysis for CLGs. We show that there is a fundamental difference between inference in CLGs and inference in discrete networks. In particular we show that for simple network structures where exact inference in discrete models can be done in linear time, even approximate inference in hybrid models is NP-hard.

From an algorithmic point of view, the state of the art algorithm for exact inference in CLGs is Lauritzen's algorithm [Lau92, LJ01]. Lauritzen's algorithm is based on the *clique tree algorithm* [LS88, SS90, HD96], originally developed for discrete Bayesian networks. Unfortunately, although the generalization of clique trees to CLGs seems straightforward initially, this common perception is wrong, and in many cases Lauritzen's algorithm is intractable even for simple network structures (which is not surprising in view of our theoretical analysis). For approximate inference, the most popular framework is based on stochastic sampling. This framework applies to every class of hybrid models and not just CLGs. However, algorithms based on sampling might take a long time to converge to a reliable answer, and are not suited to real-time applications.

In this thesis we provide a large set of new inference algorithms for hybrid models. In most cases our main concern was to come up with tractable inference algorithms that can scale up to large hybrid models. In particular, our contributions are:

- We provide a clear and coherent presentation of Lauritzen's algorithm and discuss its computational complexity.
- We present a determinstic any-time algorithm for inference in CLGs. Our algorithm can be used even in cases where Lauritzen's algorithm is not tractable. We show empirically that our algorithm outperforms stochastic sampling based algorithms.
- We show how non-linear relations between continuous variables can be handled using numerical integration techniques by finding a Gaussian approximation to the resulting distribution. We provide a simple, general and efficient algorithm to do so.
- We provide the first systematic way to deal with networks that involve discrete nodes with continuous parents, again using numerical integration techniques.

In particular, we show how Lauritzen's algorithm can be generalized to these networks. We provide performance guarantees to our algorithm, namely that it finds the correct first two moments of the posterior distribution up to numerical integration errors.

- We provide techniques for inference in large hybrid DBNs. While the classical inference algorithms for SLDS do not scale up, our proposed algorithm can be used in models that contain tens or even hundreds of variables. Our algorithms can be easily combined with our numerical integration techniques to handle non-linear relations between continuous variables.

Finally, from a practical point of view, hybrid models used in practice are limited to relatively simple models with a very small number of discrete variables. In this thesis we show a real-world application of a hybrid model with a much richer and more complex model. In particular, we focus on the fault diagnosis domain where we tackle a complex real-world physical system, originally designed to extract oxygen from the Martian atmosphere. We show that our algorithms can perform fault diagnosis practically in real time, even when some useful sensor readings are missing. It is important to emphasize that much of our results is based on real data, collected during actual runs of the system prototype, and *not* only on synthetic data generated from our own model.

1.5 Outline

Chapters 2 and 3 provide an overview of discrete and hybrid Bayesian networks. In Chapter 2 we provide a formal definition of Bayesian Networks and then describe inference methods in discrete networks. We discuss in detail both exact and approximate inference, as these algorithms form the basis on which we build later in the thesis. In Chapter 3 we provide a quick overview of the normal distribution and then define and discuss CLG models. We give a detailed description of Lauritzen's algorithm for exact inference in CLGs and discuss its computational complexity. We conclude by discussing the popular approximate inference algorithms for CLGs.

In Chapter 4 we provide some new complexity results for CLGs and discuss their implications. We show that, unlike discrete models, approximate inference even in CLGs with a very simple network structure is NP-hard. However, in practice it is possible to use domain specific properties to our advantage. In Chapter 5 we present an efficient approximate inference algorithm that takes advantage of skewed distributions, which are common in many domains, including fault diagnosis (due to the low probability of faults). We compare our algorithm with other existing inference algorithms for CLGs.

In Chapters 6 and 7 we relax the CLG assumptions and show how we can perform approximate inference in non-linear models using numerical integration techniques. In Chapter 6 we show how to extend our algorithm from Chapter 5 to handle models with non-linear dependencies between the continuous variables. In Chapter 7 we show how to extend Lauritzen's algorithm to models with discrete nodes with continuous parents and discuss the issues that come up if we want to extend the algorithm from Chapter 5 to these models.

We next turn our attention to dynamic models. In Chapter 8 we provide an overview of both discrete and hybrid dynamic Bayesian networks and some current relevant inference algorithms. In Chapter 9 we discuss how to scale up these algorithms to much larger and more complex models. In particular, we discuss the issues that come up when the model contains a large number of variables and when there are non-linear dependencies between the continuous variables.

Finally, in Chapter 10 we present a large, complex, non-linear physical system, designed to extract oxygen from the Martian atmosphere. We discuss various issues that come up when trying to model such a system as a hybrid dynamic Bayesian network. We then apply many of the algorithms presented earlier in the thesis to track the state of the system and perform fault diagnosis on real data collected during actual runs of a prototype of the system. Chapter 11 contains discussion of some open questions and concluding remarks.

Some of the material in this thesis has appeared previously in conference papers. The material in Chapter 4 and parts of Chapter 5 appeared in [LP01]. Chapter 6 is mostly new, although a preliminary high-level version of these ideas appeared in [LMS+02]. The first two sections of Chapter 7 closely follow the presentation in [LSK01]. Chapter 9 is loosely based on [LPKB00] although the presentation was completely re-written and many ideas in the chapter go beyond the content of the paper. Finally, parts of Chapter 10 are based on [LMS+02], although most of the results in the chapter are new.

Chapter 2

Bayesian Networks

This chapter contains background material about Bayesian networks, which are a compact way of representing a probability distribution [Pea88]. We begin by defining Bayesian networks and then provide an overview of some of the inference algorithms. In this chapter we concentrate on discrete models, i.e., models in which all the random variables are discrete. In the rest of the thesis we shall concern ourselves with hybrid models, i.e., models that contain continuous variables, as well as discrete ones.

2.1 Notation

We denote random variables by upper case letters (e.g., A, B_i, X), and the actual value of these variables by the corresponding lower case letters (e.g., a, b_i, x). We denote sets of variables by bold-face upper case letters (e.g., $\boldsymbol{A}$, $\boldsymbol{B}_i$, $\boldsymbol{X}$) and their values by the corresponding lower case letters (e.g., $\boldsymbol{a}$, $\boldsymbol{b}_i$, $\boldsymbol{x}$). We use $\mathrm{Dom}(X)$ to denote the set of all possible values that X can take.

When we talk about hybrid models, containing both continuous and discrete variables, we often use the convention of assigning discrete variables letters from the beginning of the alphabet (e.g., A, B, C, D) and assigning continuous variables letters from the end of the alphabet (e.g., W, X, Y, Z). We use $\boldsymbol{\Delta}$ to refer to all the discrete variables in the model and $\boldsymbol{\Gamma}$ to refer to all the continuous ones.

We use the notation $P(X)$ to denote the probability distribution for X. The probability distribution is a mass function in the discrete case and a density function in the continuous case. We use the notation $P(X = x)$ (or $P(x)$ as a shorthand) the denote the probability that X takes the value x. As usual, $P(X \mid Y)$ refers to the conditional distribution of X given Y and $P(X \mid y)$ refers to the conditional distribution of X given $Y = y$.

We denote the expectation of X under the distribution P as $E_P[X]$. Similarly, we note the expectation of some function $f(X)$ as $E_P[f(X)]$. When the distribution under the which the expectation is taken is clear from the context we use the shorthand notation $E[X]$.

In Bayesian networks we use the notation $\text{Par}(X)$ the denote all the parents of X in the graph (this will become clear when we define Bayesian networks in Section 2.2).

2.2 Representation

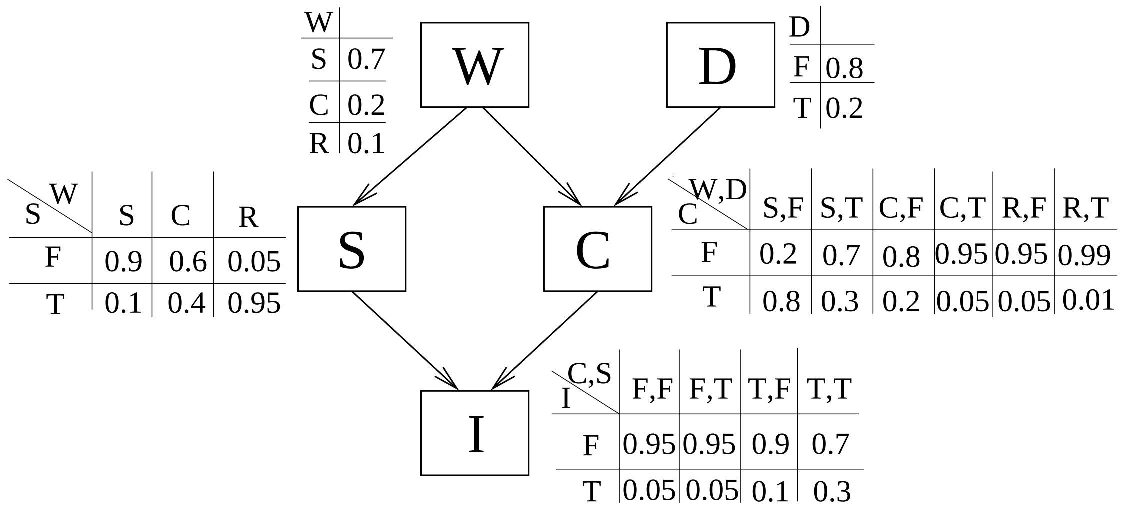
Bayesian networks (or *Bayes nets*) are a compact representation of a joint probability distribution over a set of variables $\boldsymbol{X}$. A Bayes net consists of two parts:

- A directed, acyclic graph $\mathcal{G}$, where each node corresponds to one random variable $X_i \in \boldsymbol{X}$.
- A set of *Conditional Probability Distributions (CPDs)*, one for each node in the graph $\mathcal{G}$.

The graph $\mathcal{G}$ encodes the structure of the joint probability distribution in terms of conditional independencies: *every node is independent of its non-descendants given its parents*. Each node X_i directly depends on its parents $\text{Par}(X_i)$, and the set of CPDs parameterizes this dependency. In the case of discrete Bayes nets, the CPD for node X_i encodes a probability distribution for X_i given every possible combination of its parents.

The joint probability induced by a Bayes net is defined via the *Chain Rule*. Assume the Bayes net has n nodes, representing n random variables $X_1, ..., X_n$. Then:

$$P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i \mid \text{Par}(X_i)) \tag{2.1}$$



W	
S	0.7
C	0.2
R	0.1

D	
F	0.8
T	0.2

S \ W	S	C	R
F	0.9	0.6	0.05
T	0.1	0.4	0.95

C \ W,D	S,F	S,T	C,F	C,T	R,F	R,T
F	0.2	0.7	0.8	0.95	0.95	0.99
T	0.8	0.3	0.2	0.05	0.05	0.01

I \ C,S	F,F	F,T	T,F	T,T
F	0.95	0.95	0.9	0.7
T	0.05	0.05	0.1	0.3

Figure 2.1: A simple Bayes net, modeling John's weekends. The nodes in the network are *Weather (W): Sunny, Cloudy, Rainy*; *Deadline (D): True, False*; *Climbing (C): True, False*; *Slippery(S): True, False* and *Injured (I): True, False.*

Before we continue, let us examine an example based on the way a certain Stanford student called John (the name was changed to protect the innocent) spends his weekends. Two factors influence John's plans for the weekend: the *Weather (W)* which can be *Sunny, Cloudy,* or *Rainy* and whether John has a Paper *Deadline (D)*. Based on these factors, John decides whether to go *Climbing (C)* or not. Furthermore, the *Weather* also influences the probability of the rocks to be *Slippery (S)*. Depending on whether John goes climbing and whether the rocks are slippery, John has a certain probability to get *Injured (I)*. Note that if John decides not to go *Climbing*, the probability of *Injured* does not depend on the rocks being *Slippery*.

John's weekend adventures are modeled in the Bayes net shown in Figure 2.1. The Bayes net structure encodes the conditional independencies, while the CPDs encode the actual parameters of the distribution. For example, although the probability of *Injured* is not independent of *Weather*, it *is* independent of *Weather* given its parents *Climbing* and *Slippery*. As another example, note that *Weather* and *Deadline* are independent (contrary to the conventional wisdom according to which paper deadlines are more likely to happen when the weather is nice). The CPDs encode the parameters of the probability distribution: for example $P(W = \mathit{Sunny}) = 0.7$ and $P(C = \mathit{True} \mid$

$W = Sunny, D = True) = 0.3$. Note also that the fact that *Injured* is independent of *Slippery* given that John decides not to go *Climbing* is not encoded in the structure of the Bayes net but rather in the CPD of the node *Injured*.

Let us examine a few simple examples for probabilistic inference using a Bayesian network. Suppose we want to know the probability of a certain atomic event, e.g., *Weather=Sunny, Deadline=False, Climbing=True, Slippery=False, Injured=False*. We can easily compute this using the chain rule from Equation 2.1

$$
\begin{aligned}
&P(W = Sunny, D = False, C = True, S = True, I = False) = \\
&\qquad P(W = Sunny) \cdot \\
&\qquad P(D = False) \cdot \\
&\qquad P(C = True \mid W = Sunny, D = False) \cdot \\
&\qquad P(S = True \mid W = Sunny) \cdot \\
&\qquad P(I = False \mid C = True, S = True) = \\
&\qquad 0.7 \cdot 0.8 \cdot 0.8 \cdot 0.1 \cdot 0.7 = 0.03136
\end{aligned}
$$

Obviously, if we can compute the probability of every possible atomic event we can answer every query. For example, if we are interested in the probability of John getting *Injured* while the *Weather* is *Sunny* then:

$$
\begin{aligned}
P(W = Sunny, I = True) &= \sum_{d,c,s} P(W = Sunny, D = d, C = c, S = s, I = True) \\
&= \sum_{d,c,s} P(W = Sunny) \cdot \\
&\qquad P(D = d) \cdot \\
&\qquad P(C = c \mid W = Sunny, D = d) \cdot \\
&\qquad P(S = s \mid W = Sunny) \cdot \\
&\qquad P(I = True \mid C = c, S = s)
\end{aligned}
$$

Where d, c and s can each be either *True* or *False*. Note that we sum 8 different events here, and in general the number of terms we have to consider might be exponential in the size of the network. We will return to this issue when we discuss inference in Bayes nets. Also note that by being able to compute the marginal we automatically know how to compute conditional distributions since any conditional distribution is the ratio of two marginals: $P(\boldsymbol{X} \mid \boldsymbol{Y}) = \frac{P(\boldsymbol{X},\boldsymbol{Y})}{P(\boldsymbol{Y})}$

We are now ready to formally define a Bayesian network.

Definition 2.1 *A Bayesian network $\mathcal{B}$ over the random variables $\boldsymbol{X} = \{X_1, ..., X_n\}$ is a pair $\langle \mathcal{G}, \boldsymbol{\theta} \rangle$. $\mathcal{G}$ is a directed acyclic graph with one node for every variable X_i. $\boldsymbol{\theta} = \{\theta_1, ..., \theta_n\}$ is a set of conditional probability distributions where θ_i is the conditional probability distribution $P(X_i \mid Par(X_i))$. The joint distribution represented by $\mathcal{B}$ is given by the chain rule in Equation 2.1.*

2.3 Exact Inference

Almost every system that uses Bayesian networks must be able to perform probabilistic inference, i.e., find the distribution over some variables under certain conditions. Given a Bayes net over the variables $\boldsymbol{X}$ we typically need to compute the distribution $P(\boldsymbol{Q} \mid \boldsymbol{e})$ where $\boldsymbol{Q}, \boldsymbol{E} \subseteq \boldsymbol{X}$. $\boldsymbol{Q}$ are called the *query variables* and $\boldsymbol{E} = \boldsymbol{e}$ is the *evidence* or *observations*. Since we can express the conditional distribution as ratio of two marginals, we start our discussion by assuming that $\boldsymbol{E} = \emptyset$. Unfortunately, even in this limited case inference in Bayesian networks is NP-hard.

Theorem 2.2 *Given a Bayesian network over $\boldsymbol{X}$ and some variables $\boldsymbol{Q} \subseteq \boldsymbol{X}$ then in general computing $P(\boldsymbol{Q})$ is NP-hard, even if $|\boldsymbol{Q}| = 1$.*

This theorem was proven by a relatively simple reduction from the SAT problem [GJ79] into the problem of Bayes net inference [Coo90]. However, in many cases we can take advantage of the structure of the Bayes net or its parameters to perform probabilistic inference or approximate probabilistic inference efficiently. In this section and the next one we discuss some of these methods.

2.3.1 Variable Elimination

As we have already seen, we can perform probabilistic inference by summing up the relevant terms, each of which we can compute using the chain rule. For example, assume that using the Bayes net in Figure 2.1 we would like to compute the marginal probability distribution of *Injured*. Then:

$$\begin{aligned} P(I) &= \sum_{w,d,s,c} P(d,w,s,c,I) \\ &= \sum_{w,d,s,c} P(d)P(w)P(c \mid w,d)P(s \mid w)P(I \mid c,s) \end{aligned} \tag{2.2}$$

Note that the expression involves two types of terms — constants and functions. For example the term $P(s \mid w)$ is a number while the term $P(I \mid c, s)$ is a function over the variable I. The functions, also called *factors*, can be represented as tables, where each row of the table corresponds to a specific instantiation of the function variables. For example, the function $P(I \mid c, s)$ for the specific values $c = True$ and $s = False$ can be represented as:

I	$P(I \mid c = \text{True}, s = \text{False})$
False	0.9
True	0.1

Alternatively we can think of all the terms in the Equation 2.2 as functions involving the variable w, d, s, c as well as the variable I. Under this view the term $P(I \mid c, s)$ is represented as the factor f_I:

$f_I(I, c, s) =$

I	c	s	
False	False	False	0.95
False	False	True	0.95
False	True	False	0.9
False	True	True	0.7
True	False	False	0.05
True	False	True	0.05
True	True	False	0.1
True	True	True	0.3

To evaluate Equation 2.2 we need to be able to perform two operations:

- Multiplying factors together
- Performing summation over some of the variables in the factor

To multiply the factors $f_1(\boldsymbol{Y}_1), ..., f_k(\boldsymbol{Y}_k)$ we need to create a factor f over the variables $\boldsymbol{Y}$ where $\boldsymbol{Y} = \cup_j \boldsymbol{Y}_j$, i.e., over the union of the variables of the k original factors. Consider some table entry in f, corresponding to some assignment $\boldsymbol{y}$. In each of the original k factors there is precisely one entry consistent with $\boldsymbol{y}$. The table entry in f is defined to be the product of these k tables entries in $f_1, ..., f_k$.

To sum some of the variables $\boldsymbol{Y}_1$ of the factor $f(\boldsymbol{Y})$ we need to create a factor f' of the remaining variables $\boldsymbol{Y}_2 = \boldsymbol{Y} - \boldsymbol{Y}_1$. Consider a table entry in $f(\boldsymbol{Y})$ corresponding to the assignment $\boldsymbol{y}$. $\boldsymbol{y}$ is consistent with one table entry in f', denoted as $\boldsymbol{y}_2$. We define the value of $f'(\boldsymbol{y}_2)$ as the sum of the matching table entries from f, i.e., $f'(\boldsymbol{y}_2) = \sum_{\boldsymbol{y}_1} f(\boldsymbol{y}_1, \boldsymbol{y}_2)$.

We shall soon consider some examples of multiplication and summation, but we can already make one crucial observation. If we want to evaluate Equation 2.2 as it is written we need to multiply five factors together resulting in f, a factor over all five variables d, w, c, s, I. In general, if we follow the pattern of Equation 2.2 in any Bayesian network, we will always end up with a factor over all the variables in our domain, whose size is exponential in the number of variables. Clearly, we will not be able to use this approach for large Bayes nets with hundreds of variables or even medium Bayes nets with tens of variables.

The key observation to making Bayes net inference tractable is that we can re-write Equation 2.2 by re-arranging the terms and pushing the summations inside:

$$P(I) = \sum_{c,s} P(I \mid c, s) \sum_{w} P(w)P(s \mid w) \sum_{d} P(d)P(c \mid w, d) \tag{2.3}$$

Let us examine the evaluation of Equation 2.3. We start by evaluating the product $P(d)P(c \mid w, d)$. We first represent $P(d)$ and $P(c \mid w, d)$ as the factors f_d and f_c:

$f_d(d) =$

D	
False	0.8
True	0.2

$f_c(c, w, d) =$

c	w	d	
False	Sunny	False	0.2
False	Sunny	True	0.7
False	Cloudy	False	0.8
False	Cloudy	True	0.95
False	Rainy	False	0.95
False	Rainy	True	0.99
True	Sunny	False	0.8
True	Sunny	True	0.3
True	Cloudy	False	0.2
True	Cloudy	True	0.05
True	Rainy	False	0.05
True	Rainy	True	0.01

The result of multiplying these factors together is:

$f_1(c, w, d) =$

c	w	d	
False	Sunny	False	0.16
False	Sunny	True	0.14
False	Cloudy	False	0.64
False	Cloudy	True	0.19
False	Rainy	False	0.76
False	Rainy	True	0.198
True	Sunny	False	0.64
True	Sunny	True	0.06
True	Cloudy	False	0.16
True	Cloudy	True	0.01
True	Rainy	False	0.04
True	Rainy	True	0.002

As an example, the first line corresponds to the first line of each of the factors f_d and f_c and accordingly $0.8 \cdot 0.2 = 0.16$. We next evaluate $\sum_d P(d)P(c \mid w, d) = \sum_d f_1$. The result is the factor f_2:

$f_2(c, w) =$

c	w	
False	Sunny	0.3
False	Cloudy	0.83
False	Rainy	0.958
True	Sunny	0.7
True	Cloudy	0.17
True	Rainy	0.042

For example, the second line in f_2 corresponds to the third and fourth lines from f_1 and accordingly $0.64 + 0.19 = 0.83$. We are left with the following expression:

$$P(I) = \sum_{c,s} P(I \mid c, s) \sum_{w} P(w)P(s \mid w) f_2(c, w)$$

Note that we have *eliminated* the variable d, which will not appear in any of our computations from now on. The early elimination of variables is the key to avoiding the creation of exponentially large factors.

We can continue in the same fashion. We convert $P(w)$ and $P(s \mid w)$ into factors and multiply them together with $f_2(c, w)$, resulting in the factor $f_3(c, s, w)$. We then eliminate w, resulting in the factor $f_4(c, s)$. The next step is to convert $P(I \mid c, s)$ and multiply it with $f_4(c, s)$, resulting in $f_5(c, s, I)$. Finally we sum out c and s from f_5 leading to the answer $f_6(I)$ (note that the entries sum to 1, as expected from a marginal distribution):

$f_6(I) =$

I	
False	0.910272
True	0.089728

To sum up, we performed the following computation:

$$\begin{aligned}
f_1(d, w, c) &= P(d)P(c \mid w, d) && (2.4)\\
f_2(c, w) &= \sum_d f_1(c, d, w) && (2.5)\\
f_3(w, c, s) &= f_2(c, w)P(w)P(s \mid w) && (2.6)\\
f_4(c, s) &= \sum_w f_3(w, c, s) && (2.7)\\
f_5(c, s, i) &= f_4(c, s)P(i \mid c, s) && (2.8)\\
f_6(i) &= \sum_{c,s} f_5(c, s, i) && (2.9)
\end{aligned}$$

The largest factor we had to deal with was f_3, having 3 variables and 12 entries. This is much better than directly using Equation 2.2 which led to a factor with 5 variables and 48 entries. In larger networks the savings are usually even more significant.

We are now ready to state the variable elimination algorithm [BB72, ZP94]. We first convert all our CPDs into a set of factors $\mathcal{F}$. As long as $\mathcal{F}$ contains variables which do not appear in the query we choose such variable Y and eliminate it. The elimination is done by extracting all the factors to which Y belongs, multiplying them together, summing out Y and returning the result to $\mathcal{F}$. When the only variables left

```
Variable Elimination (𝒫 = {P_1,...,P_n} a set of CPDs, Q query variables)
   1.  Let f_i be the factor representation of P_i
   2.  Let ℱ = {f_1,...,f_n} (a set of factors)
   3.  Let X be the set of random variables used in ℱ
   4.  Let Y = X − Q
   5.  While (Y ≠ ∅)
     6.  Choose some variable Y ∈ Y and set Y ← Y − {Y}
     7.  Extract from ℱ all factors f_1,...,f_k mentioning Y
     8.  Let f = ∏_{i=1}^k f_i
     9.  Let f' be the result of summing out y from f
     10. Add f' to ℱ
   11. End-while
   12. return ∏_{f∈ℱ} f
```

Figure 2.2: The Variable Elimination Algorithm

in $\mathcal{F}$ are query variables, we multiply all the remaining factors together and return the resulting factor as our query result. Figure 2.2 shows the pseudo-code of the variable elimination algorithm.

The complexity of the variable elimination algorithm is determined by the size of the factors that are generated along the way. This in turn depends on the elimination order that we choose. For example, if we first eliminate the variable C then we get a factor over W, D, C, I, which is larger than any factor we actually had. Unfortunately, although one can use some simple greedy heuristics in choosing the elimination order [Kja90, HD96], finding the optimal elimination order, i.e., the one resulting in the smallest factors, is NP-hard [ACP87]. Furthermore, in some networks, even the optimal elimination order leads to factors which are exponential in the size of the Bayes net. In these cases we cannot use exact inference methods and must resort to approximations. Still, the variable elimination algorithm, and especially the clique tree algorithm which is based on it, are in general the best known methods for exact inference in Bayesian networks and can be used for inference in many non-trivial networks used in practice.

Handling Evidence

It is easy to generalize the variable elimination algorithm to deal with evidence. An observation such as $X = x$ can be expressed as adding an indicator function $1_x(X)$ when using the chain rule. For example, assume we want to compute $P(I)$ with the observation $S{=}\mathit{True}$. We can then write:

$$\begin{aligned} P(I, S = \mathit{True}) &= \sum_{w,d,s,c} P(d, w, s, c, I) \cdot 1_{\text{True}}(s) \\ &= \sum_{w,d,s,c} P(d)P(w)P(c \mid w, d)P(s \mid w)P(I \mid c, s) \cdot 1_{\text{True}}(s) \end{aligned}$$

$1_{\text{True}}(s)$ is 1 if $s{=}\mathit{True}$ and is 0 otherwise, effectively removing all the terms which are not consistent with the evidence. To add the evidence into the variable elimination algorithm, we simply translate the indicator function into a factor over the observed variable with 1 for the entry consistent with the evidence and 0 elsewhere, and add it to $\mathcal{F}$. In our example, we convert $1_{\text{True}}(s)$ into the factor f_s^E:

$f_s^E =$

s	
False	0
True	1

The result now is the factor

I	
False	0.2161025
True	0.0288975

Note that the entries do not sum up to 1, but rather to 0.245. The reason is that the result represents $P(I, S = \text{True})$ and 0.245 is the probability $P(S = \text{True})$. If instead we are interested in the conditional probability $P(I \mid S = \text{True})$ we simply need to divide the result by $P(S = \text{True}) = 0.245$, i.e., we need to *normalize* the

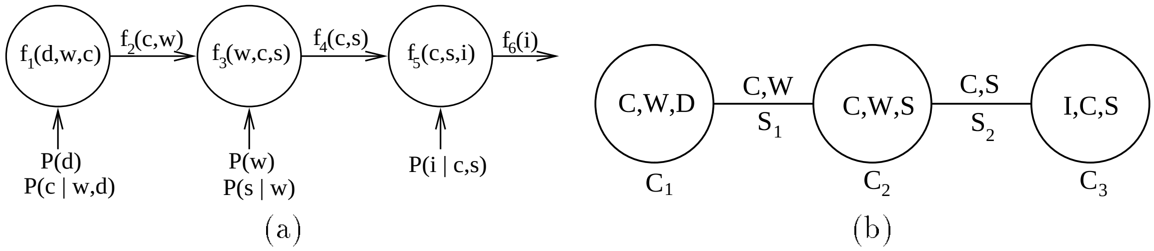


Figure 2.3: (a) Graphical illustration of the variable elimination computation of Equation 2.4 through Equation 2.9 (b) The clique tree

factor to sum up to 1. The normalized result is:

I	
False	0.882051
True	0.117949

We note that there are more efficient ways to handle evidence when doing variable elimination. Instead of adding an extra factor we can modify the factors coming from the CPDs to account for the evidence, often leading to computations which are more efficient in time in space. We took the approach of adding an extra factor since it is the approach used for clique tree inference, which we discuss next.

2.3.2 Clique trees

The *clique tree* algorithm [LS88, SS90, HD96], also known as the *join tree*, the *cluster tree* and the *junction tree* algorithm, is an inference algorithm which is based on the variable elimination algorithm. It has two main advantages:

- If we want to compute more than one marginal we need to run the variable elimination algorithm multiple times. By using clique trees we can compute multiple marginals while performing at most twice as many computations as are needed for one marginal using variable elimination.
- The data structures used by clique trees usually lead to more efficient implementations of the algorithm than direct implementations of variable elimination.

Let us take another look at the way we evaluated Equation 2.3 using variable elimination, as is summarized in Equation 2.4 through Equation 2.9. The computation had two interleaving steps. In the first step (Equation 2.4, Equation 2.6 and Equation 2.8) we multiplied some factors together. These factors were either CPDs or factors generated by earlier computations. In the second step (Equation 2.5, Equation 2.7, and Equation 2.9) we marginalized the results of the first step to smaller factors, which we later used. We can think of the two steps as one operation, taking a few factors as output and producing some other factor as input. It is convenient to visualize this using a graphical illustration — Figure 2.3(a) shows the graphical illustration for Equation 2.4 through Equation 2.9.

A *clique tree* tree can be viewed as a graph induced by the variable elimination computation. Each node in the graph corresponds to an operation of multiplying factors together, and is associated with the variables used in this factor. Each edge in the graph corresponds to a factor generated by summing out some variable — it connects the node in which the factor originated and the node in which the factor was later used. Each edge is associated with the variables used in the summed out factor, which in turn are the intersection between the variables of the two nodes it connects. As an example, the clique tree in Figure 2.3(b) is induced by the computation illustrated in Figure 2.3(a).

Definition 2.3 *A* Clique Tree *over a Bayesian network $\mathcal{B}$ with the random variables $\boldsymbol{X}$ is an undirected acyclic graph $\mathcal{T} = (V, E)$. Each node $C_i \in V$ in the graph, also called a* clique *or a* cluster*, is associated with some variables $\boldsymbol{X}_i \subseteq \boldsymbol{X}$. Each edge in the graph is associated with the non-empty intersection of the variables of the cliques it connects. Each clique C is associated with a factor $\Phi(C)$ called a* potential *and each edge S is associated with a factor $\Phi(S)$ called a* sepset *over their variables. The clique tree obeys two properties:*

- *For every CPD in $\mathcal{B}$ over the variables $\boldsymbol{Y} \subseteq \boldsymbol{X}$ there exists a clique C_i such that $\boldsymbol{Y} \subseteq \boldsymbol{X}_i$*
- *Let C_i and C_j be two cliques such that $X \in \boldsymbol{X}_i \cap \boldsymbol{X}_j$. Then every node along the path connecting C_i and C_j has X in its domain. This property is known as*

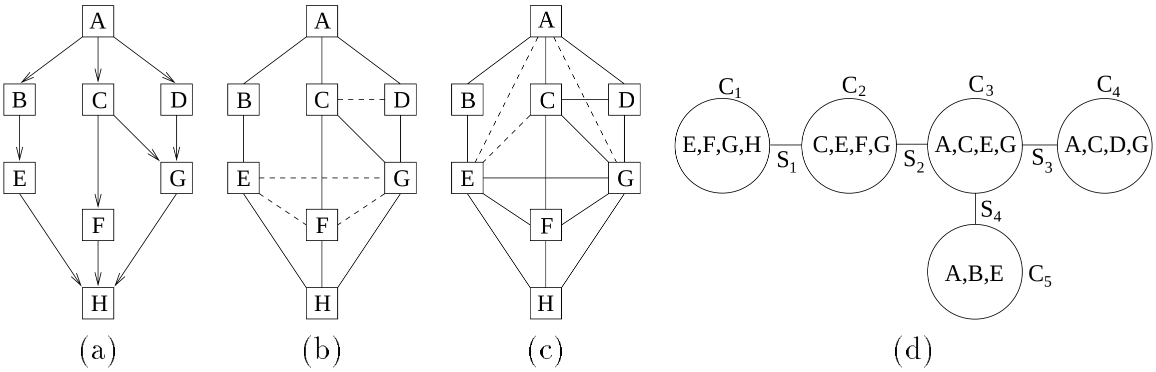


Figure 2.4: Building a clique tree from a Bayes net (a) The Bayes net (b) The moralized graph (c) A triangulated graph (d) A clique tree

the running intersection property.

Using a Clique Tree

Performing probabilistic inference using a clique tree can be viewed as a message passing algorithm. We start by discussing how to compute the marginal distribution of a certain clique, called the *root* node, and later extend the message passing algorithm to compute the probability distribution of all the cliques.

The initialization step is incorporating the CPDs into the clique tree. We first convert the CPDs into factors and for each such factor we choose a clique C that contains all the variables in the factor. We then initialize the potential of each clique to be a product of the CPDs associated with it. If no CPD belongs to some clique we set its potential to have the value 1 at every entry. Now all the relevant information is in the tree, and we can use it to compute the distribution of the root node.

The computation associated with the variable elimination algorithm can be performed as a message passing algorithm, where messages are passed along the tree starting with the leaves and going towards the root as is illustrated in Figure 2.3(b) (where C_3 is the root). In this scheme, each node C needs to send a message to its neighbor C' along the path to the root. To do so, it first needs to receive messages from all its other neighbors, multiply them together with its own potential and sum out all the variables that are not in C'.

For example, assume we want to compute the distribution of C_5 in Figure 2.4(d). We can use the following sequence of messages:

Step	From Clique	To Clique	Operation
0			Initialize potentials from CPDs
1	C_1	C_2	$f_{12} \leftarrow \sum_H \Phi(C_1)$; $\Phi(C_2) \leftarrow f_{12} \cdot \Phi(C_2)$
2	C_4	C_3	$f_{43} \leftarrow \sum_D \Phi(C_4)$; $\Phi(C_3) \leftarrow f_{43} \cdot \Phi(C_3)$
3	C_2	C_3	$f_{23} \leftarrow \sum_F \Phi(C_2)$; $\Phi(C_3) \leftarrow f_{23} \cdot \Phi(C_3)$
4	C_3	C_5	$f_{35} \leftarrow \sum_{C,G} \Phi(C_3)$; $\Phi(C_5) \leftarrow f_{35} \cdot \Phi(C_5)$

Note that this is not the only possible order. For example we can reverse steps 1 and 2, but we cannot reverse steps 3 and 4 (C_3 can send the message to C_5 only after receiving messages from both C_2 and C_4).

Assume now that we want to compute the marginal distribution for every node in the clique tree, not just the root. We can repeat the process we just outlined for every node in the tree, but we can do better. Continuing our last example of Figure 2.4(d) let us assume that, having computed to marginal probability of C_5, we are also interested in computing the marginal probability of C_3. We have already done a big part of the work — steps 1,2,3 of the computation are also useful for computing the distribution of C_3. Our goal is to reuse this work rather than repeat it for C_3.

Naively, one might think that all we need to do is to create the message $f_{53} = \sum_B \Phi(C_5)$ and multiply $\Phi(C_3)$ by it. Unfortunately this is incorrect. The reason is that $\Phi(C_5)$ was already multiplied by the message f_{35}. Let us note the original factor in C_5 (before multiplying it by f_{35}) by $\Phi_{\text{orig}}(C_5)$, and let $\Phi_{\text{new}}(C_5)$ be the factor in C_5 after multiplying by f_{35}. What we need is to send from C_5 to C_3 the message $\sum_B \Phi_{\text{orig}}(C_5)$, but by using the current potential of C_5 we would get the message $\sum_B \Phi_{\text{new}}(C_5)$.

Fortunately there is a very simple solution to our problem:

$$\sum_B \Phi_{\text{orig}}(C_5) = \frac{\left(\sum_B \Phi_{\text{orig}}(C_5)\right) \cdot f_{35}}{f_{35}} = \frac{\sum_B \left(\Phi_{\text{orig}}(C_5) \cdot f_{35}\right)}{f_{35}} = \frac{\sum_B \Phi_{\text{new}}(C_5)}{f_{35}} \tag{2.10}$$

The division of factors $\frac{f_1}{f_2}$ is defined just like a product of factors — we create a

```
Message Passing in a Clique Tree from C_1 to C_2 along the sepset S
    1. Let f_1 = Σ_{C_1-S} Φ(C_1)   /* Sum out C_1 */
    2. Let f_2 = f_1 / Φ(S)         /* Divide by the last message sent */
    3. Let Φ(S) = f_1               /* Store the message */
    4. Let Φ(C_2) = Φ(C_2) · f_2    /* Multiply C_2 by the message*/
```

Figure 2.5: Sending a message in the clique tree

```
Calibration of a Clique Tree
   1. Initialize each potential as the product of the relevant CPDs
   2. Initialize each sepset by a factor with all entries set to 1
   3. Let Q be all directed pairs of adjacent nodes ⟨C_i, C_j⟩
   4. While Q ≠ ∅
     5. Let ⟨C_i, C_j⟩ ∈ Q s.t. C_i got messages from all its
        neighbors except perhaps C_j
     6. Send a message from C_i to C_j using the protocol of Figure 2.5
     7. Let Q = Q − {⟨C_i, C_j⟩}
   8. end-while
```

Figure 2.6: The Calibration Algorithm

factor over the union of the variables but instead of multiplying the relevant entries we divide them. Furthermore, for the purposes of dividing factors we define $\frac{0}{0} = 0$.

From Equation 2.10, we see that all we have to do in order to send a message backwards in the tree is to save the message we sent towards the root. Recall that we associated each edge with a sepset — a factor of the variables of the edge. We now use the sepsets to store the messages sent across it in the tree. Figure 2.5 summarizes the operation of sending a message from one node to another node in a clique tree.

We are now ready to present the *Calibration* algorithm, shown in Figure 2.6. Using the message passing protocol of Figure 2.5 each node sends a message to each one of its neighbors after receiving a message from all its other neighbors, for a total of $2n-2$ messages if we have n nodes connected by $n-1$ sepsets. Note that it is always possible to send all $2n-2$ messages — for example we can choose a node as a root, send all $n-1$ messages towards the root and then send $n-1$ messages from the root towards the leaves. Sending the messages towards the root is called an *upward* pass or an *upstream* pass. Sending the messages to the leaves is called a *downward* pass or a *downstream* pass.

Theorem 2.4 *After running the calibration algorithm each potential in the tree contains the correct marginal distribution over its variables. Furthermore, the sepsets contain the correct marginal distribution over their variables.*

Proof: We first show that the potentials have the correct marginal distribution. Recall that we have already seen that by performing an upward pass we get the correct marginal distribution in the root. It is therefore enough to show that the calibration algorithm of Figure 2.6 multiplies each potential by the same factors it would have been multiplied by if it were the root in an upward pass.

The proof goes by induction on the number of the message. For the first message the claim is trivial since dividing by a sepset whose entries are all set to 1 does not change the message. For the k'th message going from C_i to C_j there are two cases. In the first case C_j did not yet send a message to C_i and the sepset is still set to 1's. In this case the division does not change the message and using the induction hypothesis we have the correct message. In the second case, C_j already sent a message to C_i. Using the argument of Equation 2.10 we get that the message being sent is the same message we would have sent in an upward pass.

It is left to show that the sepsets have the correct marginal distributions. Let S be a sepset between C_i and C_j and WLOG assume that C_i sent a message to C_j after C_j sent a message to C_i. S will be set to the marginal distribution of C_i when it sent the message. By this point C_i must have received messages from all its other neighbors and we assumed it also received the message from C_j, therefore C_i already received all its message and contained the correct marginal distribution. It follows that the message from C_i to C_j, which is stored in S, has also the correct marginal distribution. ∎

Clique trees as a probability distribution

Clique trees are more than just a more efficient way of implementing the variable elimination algorithm. Just like Bayes nets, clique trees are also a graphical model that represents a joint probability distribution [Pea88] which is the product of the potentials divided by the product of the sepsets.

Definition 2.5 *A clique tree $\mathcal{T} = (V, E)$ over the variables $\boldsymbol{X}$ represents a joint probability distribution $P(\boldsymbol{X})$ given by:*

$$P(\boldsymbol{X}) = \frac{\prod_{C_i \in V} \Phi(C_i)}{\prod_{S_i \in E} \Phi(S_i)} \tag{2.11}$$

Note how this view leads to an alternative way to justify the clique tree algorithm in Figure 2.6. Initially we mutiply all the CPDs into the cliques and all the sepsets are set to be 1. Thus, after the initialization step, the distribution represented by the clique tree is the product of the CPDs, which is indeed the distribution represented by the Bayes net. It is easy to verify that Equation 2.11 is invariant to message passing as described in Figure 2.5, i.e., whenever we pass messages in the tree we maintain the correct joint distribution represented by $\mathcal{T}$. Therefore at the end of the message passing algorithm $\mathcal{T}$ represents the probability distribution represented by the Bayes net.

Handling evidence

We introduce evidence into a clique tree by the same technique of indicator functions that we used for variable elimination. If we have some evidence $\boldsymbol{e} = e_1, ..., e_k$ we can treat it as multiplying the tree by the indicator functions $1_{e_1}(X_1), ..., 1_{e_k}(X_k)$. We convert the function $1_{e_i}(X_i)$ into a factor $f_i^E(X_i)$, find a clique that contains X_i and then multiply it by the factor $f_i^E(X_i)$. If we calibrate the tree, each clique will now contain the marginal distribution of its variable after observing the evidence. To get the conditional distribution given the evidence we simply need to re-normalize the cliques (and the sepsets).

From a Bayes net to a clique tree

We conclude by examining the process of building a clique tree from a given Bayes net $\mathcal{B}$ more closely. To build a clique tree from $\mathcal{B}$ we can simply run the variable elimination algorithm, without actually computing the factors, as it is enough to just determine which factors involve which variables. However, it is convenient to view this process in a different way, leading to some insights about how to choose a good elimination ordering. The idea is to transform $\mathcal{B}$ into an undirected graph $\mathcal{G}$ over the same random variable. Each edge in $\mathcal{G}$ corresponds to two variables that appear together in at least one factor. The factors generated during the variable elimination algorithm correspond to the maximal cliques in $\mathcal{G}$ (by definition of $\mathcal{G}$, a set of variables is in one factor iff it is in a clique in $\mathcal{G}$).[1]

The Graph $\mathcal{G}$ is constructed in two steps: *moralization* and *triangulation*. In the moralization step we build the graph $\mathcal{G}_M$ by connecting every two variables that appear in the same CPD in $\mathcal{B}$ (and thus appear together in some factor). Obviously every edge in $\mathcal{B}$ becomes an undirected edge in $\mathcal{G}_M$. In addition, if X_i and X_j are both parents of some node X_k in $\mathcal{B}$ then we also add an edge between them (since they appear together in the CPD for X_k).[2] As an example, Figure 2.4(a) is some Bayes net $\mathcal{B}$, and Figure 2.4(b) shows the moralized graph. Note for example that the nodes E, F, G were married.

Although $\mathcal{G}_M$ contains a clique for every CPD in $\mathcal{B}$, it is not necessarily enough, since we have to generate intermediate factors as we eliminate the variables. Thus, when building the graph $\mathcal{G}$, we must add enough edges to $\mathcal{G}_M$ such that, for some chosen elimination order, variables appearing together in some intermediate factors will also be connected in $\mathcal{G}$.

The simplest way to construct $\mathcal{G}$ is to initially set $\mathcal{G} \leftarrow \mathcal{G}_M$. We then pick some elimination order. When we eliminate some variable X_i we add edges between all its neighbors in $\mathcal{G}$ that were not already eliminated, i.e., we create a clique over X_i and its non-eliminated neighbors. The reason is simple: when we eliminate X_i we multiply all the factors in which it appears. The variables in these factors are exactly the neighbors of X_i that were not yet eliminated. When multiplying them together, we force them to be in the same factor, and therefore we need to put them in the same clique.

[1]This construction is the source of the name "clique tree".
[2]Connecting, or marrying, the parents leads to the name moralization.

The process we described will result in a graph $\mathcal{G}$ that is *chordal*. Intuitively, a chordal graph is a graph in which every cycle of length four or more contains a "shortcut" — an edge between two non-adjacent nodes. Thus, all the cycles in $\mathcal{G}$ are broken up into cycles of length three, and therefore $\mathcal{G}$ is also called *triangulated*.

Definition 2.6 *An undirected graph $\mathcal{G}$ is* chordal *or* triangulated *if all cycles of length bigger than three have some chord in them, i.e., an edge that is not a part of the cycle, connecting two nodes of the cycle. In other words, $\mathcal{G}$ does not have a subgraph that is isomorphic to a cycle of length $k > 3$.*

The reason we end up with a chordal graph is simple: Suppose that at some point $\mathcal{G}$ contains a cycle of length four or more, and let X_i be the first eliminated variable in this cycle. When eliminating X_i we add an edge between its neighbors, which were non-adjacent nodes on the cycle. As the following theorem states, the converse is also true. For details, and an algorithm to extract an elimination order from a chordal graph, see [TY84].

Theorem 2.7 *Let $\mathcal{G} = (V, E)$ be some undirected graph which contains some moralized graph $\mathcal{G}_M = (V, E_M)$ as a subgraph, i.e., $E_M \subseteq E$. Then the maximal cliques in $\mathcal{G}$ correspond to a set of factors that can be used for variable elimination for $\mathcal{G}_M$ iff $\mathcal{G}$ is chordal.*

The process of taking an undirected graph and making it chordal by adding edges to it, is called *triangulation*. Thus, the second phase of the algorithm is triangulating $\mathcal{G}$. Figure 2.4 shows an example where the moral graph shown in Figure 2.4(b) was triangulated, resulting in the graph in Figure 2.4(c).

Finding the optimal triangulation, i.e., one that induces small cliques is NP-hard — this is not a surprise since finding the optimal elimination order is NP-hard. However, one can use some simple heuristics which tend to produce good triangulation on many real-life Bayes net. One such simple heuristic is triangulating the graph using an elimination order where at each time we greedily eliminate the variable that adds the smallest number of new edges to $\mathcal{G}$ [Kja90]

To summarize, transforming a Bayes net into a clique tree can be done in following steps, illustrated in Figure 2.4:

- Building the moralized graph $\mathcal{G}_M$.
- Triangulating the moralized graph by using either variable elimination or some different algorithm.
- Finding the maximal cliques in the triangulated graph. Since the graph is triangulated this step is easy; there is an efficient algorithm to find the maximal cliques in a triangulated graph [TY84]. In our case, if we triangulate the graph using variable elimination we can simply record the cliques created during the elimination process, while discarding cliques which are subsets of other cliques.
- Connecting the cliques in a way that preserves the running intersection property. To do so, we need to connect cliques that have the most variables in common. This can be done by using the maximal spanning tree algorithm [CLR90], where the weight of each edge is the number of random variables in its (non-empty) domain, i.e, an edge that connects cliques with the variables $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$ has a weight of $|\boldsymbol{X}_i \cap \boldsymbol{X}_j|$.

2.4 Sampling Techniques

As already mentioned, exact inference in Bayesian networks is often intractable since the problem is NP-hard [Coo90]. When the factors involved in the variable elimination algorithm or clique tree algorithm become too large to handle efficiently, we must look for alternative approaches. The obvious alternative for exact inference is approximate inference, where we settle for an answer that is only approximate. Although in general even approximate inference in Bayesian networks is NP-hard [DL93], there are many important cases in which exact inference is not tractable but there exists an efficient approximate inference algorithm that leads to provably good approximations.

In this section we concentrate on *sampling* based approximate inference algorithms. The idea behind sampling methods is to randomly pick assignments of the random variables (these assignments are called *samples*) and then estimate various properties of the joint distribution using these samples.

More formally, assume we have some joint distribution $P(\boldsymbol{X})$ and we want to evaluate the expectation of some function f under this distribution, i.e., we are interested in evaluating $E_P[f(\boldsymbol{X})]$. It is important to note that the formulation is very general. In particular given some evidence $\boldsymbol{e} = e_1, \ldots, e_n$, we can reduce the problem of computing the probability $P(\boldsymbol{e})$ to this form. We define the indicator function $1_e(\boldsymbol{X})$ as

$$1_e(\boldsymbol{X}) = \prod_i 1_{e_i}(\boldsymbol{X})$$

(Recall from Section 2.3 that the indicator function $1_{e_i}(\boldsymbol{X})$ is 1 if $\boldsymbol{X}$ agrees with e_i and 0 otherwise). We can now write:

$$P(\boldsymbol{e}) = \sum_{\boldsymbol{x}} 1_e(\boldsymbol{x}) \cdot P(\boldsymbol{x}) = E_P[1_e(\boldsymbol{X})]$$

We now return to the problem of estimating $E[f(\boldsymbol{X})]$. Define the random variable $Y = f(\boldsymbol{X})$, and then $E[Y] = E[f(\boldsymbol{X})]$. We will assume that $\sigma_Y^2 < \infty$ where $\sigma_Y^2 \stackrel{def}{=} \mathrm{Var}(Y)$.

Assume now that we have a sequence of N i.i.d. (independent, identically distributed) samples $\boldsymbol{x}[1], \boldsymbol{x}[2], ..., \boldsymbol{x}[N]$ drawn from the distribution $P(\boldsymbol{X})$ and let $y[i] = f(\boldsymbol{x}[i])$. Consider the sum:

$$\frac{1}{N}\sum_{i=1}^{N} y[i] = \frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{x}[i])$$

Using the Central Limit Theorem, for large N the sum has a normal distribution with mean $E[f(\boldsymbol{X})]$ and variance $\frac{\sigma_Y^2}{N}$. Therefore the sum converges to $E[f(\boldsymbol{X})]$ and the rate of convergence is proportional to $1/\sqrt{N}$. In other words, we have a way to answer general probabilistic queries by drawing i.i.d. samples from the distribution:

$$E[f(\boldsymbol{X})] \approx \frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{x}[i])$$

2.4.1 Importance Sampling

In the previous section we discussed the problem of estimating $E[f(\boldsymbol{X})]$, where the expectation is over the joint distribution $P(\boldsymbol{X})$. However most problems also involve some observations $\boldsymbol{e}$ and we are interested in the expectation under these observations, i.e., we want to estimate $E[f(\boldsymbol{X}) \mid \boldsymbol{e}]$. To make the problem well defined, we assume that $P(\boldsymbol{e}) > 0$.

The most straightforward way to solve this problem using sampling methods is called *rejection sampling*. We can write:

$$E[f(\boldsymbol{X}) \mid \boldsymbol{e}] = \frac{E[f(\boldsymbol{X})1_e(\boldsymbol{X})]}{E[1_e(\boldsymbol{X})]}$$

We now have two expectations that we can compute using our standard sampling method. For example, in order to compute $E[f(\boldsymbol{X})1_e(\boldsymbol{X})]$, we generate samples, discard all the samples that do not agree with $\boldsymbol{e}$, and use the remaining samples to estimate the expectation. The problem with this approach is that the convergence rate may be very low. Out of N samples we expect only $P(\boldsymbol{e})$ to be consistent with the evidence; thus, effectively we only use $NP(\boldsymbol{e})$ samples to compute our expectation. Often $P(\boldsymbol{e})$ is very low (especially if the dimension of $\boldsymbol{e}$ is high) in which case rejection sampling is very inefficient. Furthermore, if $\boldsymbol{e}$ contains at least one continuous variable, then the probability of sampling exactly its value is zero, and we will discard all our samples.

To overcome this difficulty it is desirable to draw samples directly from $P(\boldsymbol{X} \mid \boldsymbol{e})$, but this is not always possible. More generally, it is often the case that we have some distribution $P(\boldsymbol{X})$ that we cannot sample from. *Importance Sampling* is a general way of dealing with this problem, based on a very simple observation. Let $Q(\boldsymbol{X})$ be some arbitrary distribution where $Q(\boldsymbol{x}) > 0$ whenever $P(\boldsymbol{x}) > 0$. Then:

$$E_P[f(X)] = \sum_{\boldsymbol{x}} f(\boldsymbol{x})P(\boldsymbol{x}) = \sum_{\boldsymbol{x}} f(\boldsymbol{x})\frac{P(\boldsymbol{x})}{Q(\boldsymbol{x})}Q(\boldsymbol{x}) = E_Q\left[f(\boldsymbol{X})\frac{P(\boldsymbol{X})}{Q(\boldsymbol{X})}\right] \tag{2.12}$$

Let $\boldsymbol{x}[1], ..., \boldsymbol{x}[N]$ be N i.i.d. samples *drawn from* Q. Define $w[i] = \frac{P(\boldsymbol{x}[i])}{Q(\boldsymbol{x}[i])}$ and then:

$$E_P[f(X)] \approx \frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{x}[i])\frac{P(\boldsymbol{x}[i])}{Q(\boldsymbol{x}[i])} = \frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{x}[i])w[i] \tag{2.13}$$

Sometimes we cannot even compute $P(\boldsymbol{x})$ and instead we can only evaluate $P'(\boldsymbol{x})$

where $P'(\boldsymbol{x}) = cP(\boldsymbol{x})$ and c is some unknown constant, i.e., we can only evaluate $P(\boldsymbol{x})$ up to some unknown constant. In this case we cannot use the estimator of Equation 2.13, but we can use a slightly different estimator. We define the weights $w(\boldsymbol{x}) = \frac{P'(\boldsymbol{x})}{Q(\boldsymbol{x})}$. Notice that each weight is an unbiased estimator for c:

$$E_Q[w] = \sum_{\boldsymbol{x}} w(\boldsymbol{x})Q(\boldsymbol{x}) = \sum_{\boldsymbol{x}} \frac{P'(\boldsymbol{x})}{Q(\boldsymbol{x})} Q(\boldsymbol{x}) = \sum_{\boldsymbol{x}} P'(\boldsymbol{x}) = c$$

Rewriting Equation 2.12 we get:

$$E_P[f(X)] = \sum_{\boldsymbol{x}} f(\boldsymbol{x}) \frac{P(\boldsymbol{x})}{Q(\boldsymbol{x})} Q(\boldsymbol{x}) = \sum_{\boldsymbol{x}} \frac{f(\boldsymbol{x}) \frac{P'(\boldsymbol{x})}{Q(\boldsymbol{x})} Q(\boldsymbol{x})}{c} = \frac{E_Q[wf(\boldsymbol{x})]}{E_Q[w]}$$

Therefore, given N i.i.d. samples $\boldsymbol{x}[1], ..., \boldsymbol{x}[N]$ from Q, we get the following estimator for $E_P[f(X)]$:

$$E_P[f(X)] \approx \frac{\frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{x}[i])w[i]}{\frac{1}{N}\sum_{i=1}^{N} w[i]} = \frac{\sum_{i=1}^{N} f(\boldsymbol{x}[i])w[i]}{\sum_{i=1}^{N} w[i]} \tag{2.14}$$

Note that if c is known to be 1, we can replace $\frac{1}{N}\sum w[i]$ by 1 and return to Equation 2.13. The estimator of Equation 2.14 is a biased estimator because of the division. On the other hand, it usually has a smaller variance than the estimator of Equation 2.13 and thus is often preferred in practice even when c is known.

No matter whether we choose to use Equation 2.13 or Equation 2.14, the obvious question one has to answer in order to use importance sampling is how to choose the distribution Q. Since in typical cases the estimator is either unbiased or has a relatively small bias compared to its variance, a reasonable criterion to choose Q is to minimize the variance of the estimator. Analysis of the estimator's variance (see for example [Gew89, Liu96]) shows that the variance of the estimator is related to the variance of the weights $w[i]$ — in general the lower the variance of the weights the lower the variance of the estimator is.

In one extreme case, $Q = P$ and the variance of the weights is 0. The analysis therefore implies that the actual distribution P would be a good choice as the importance distribution, and in general we would like Q to be as "close" as possible to P. If

Q is very different from P the variance of the weights will be very high, and intuitively a very small number of samples will account for almost the entire probability mass. All the other samples will contribute very little to our estimation and will be almost irrelevant. Thus, in the worst case, only a small fraction of our N samples would be useful, making importance sampling very inefficient compared to sampling from the correct distribution.

Likelihood Weighting

Likelihood Weighting (LW) [SP89] is the most popular way to apply importance sampling for inference in Bayesian networks. Assume we have a Bayes net $\mathcal{B}$ over the variables $\boldsymbol{X}$ and some evidence $\boldsymbol{e}$ where $\boldsymbol{E} \subseteq \boldsymbol{X}$. We would like to generate samples from the posterior distribution:

$$P(\boldsymbol{x} \mid \boldsymbol{e}) = \frac{P(\boldsymbol{x}, \boldsymbol{e})}{P(\boldsymbol{e})}$$

It is not easy to evaluate the conditional distribution P since we do not usually know the normalization factor $P(\boldsymbol{e})$. However, evaluating $P'(\boldsymbol{x}) = P(\boldsymbol{x}, \boldsymbol{e})$ is easy: It is 0 if $\boldsymbol{x}$ and $\boldsymbol{e}$ are inconsistent and otherwise we can compute it directly from $\mathcal{B}$ using the chain rule. We therefore have a function P' which is equal to P up to some unknown constant, and we can use the estimator of Equation 2.14.

It remains to define the sampling distribution Q. Given the Bayes net $\mathcal{B}$ we construct a new Bayes net $\mathcal{B}'$ by disconnecting all the evidence nodes from their parents, and set their CPD to be a deterministic CPD, with the value from $\boldsymbol{e}$. For example, if the node X was observed with the value "True" then X will have no parents and its CPD will be $P(X = \text{True}) = 1$. $\mathcal{B}'$ represents the sampling distribution Q. Note that when sampling from $\mathcal{B}'$, we let the evidence nodes influence the distribution of their descendents in the original Bayes net but they do not influence the distribution of their ancestors. It is obvious that, in general, $\mathcal{B}'$ does not represent the true posterior distribution given the evidence $\boldsymbol{e}$, but intuitively it should be a better approximation to the posterior than the prior distribution, because it accounts for at least some of the changes induced by the evidence.

Consider some sample $\boldsymbol{x}$ consistent with $\boldsymbol{e}$. Let $P_{\mathcal{B}}(X \mid \mathrm{Par}(X))$ be the distribution of X given its parents in $\mathcal{B}$ and let $P_{\mathcal{B}'}(X \mid \mathrm{Par}(X))$ be the distribution of X given its parents in $\mathcal{B}'$. For every $\boldsymbol{x}$ consistent with the evidence $\boldsymbol{e}$ we can use the chain rule and get:

$$\frac{P'(\boldsymbol{x})}{Q(\boldsymbol{x})} = \frac{\prod_{x \in \boldsymbol{x}} P_{\mathcal{B}}(x \mid \mathrm{Par}(X))}{\prod_{x \in \boldsymbol{x}} P_{\mathcal{B}'}(x \mid \mathrm{Par}(X))} = \frac{\prod_{x \in e} P_{\mathcal{B}}(x \mid \mathrm{Par}(X)) \prod_{x \notin e} P_{\mathcal{B}}(x \mid \mathrm{Par}(X))}{\prod_{x \in e} P_{\mathcal{B}'}(x \mid \mathrm{Par}(X)) \prod_{x \notin e} P_{\mathcal{B}'}(x \mid \mathrm{Par}(X))}$$

For the observed nodes $X \in \boldsymbol{E}$ the CPD in $\mathcal{B}$ is deterministic, and therefore $P_{\mathcal{B}'}(x \mid \mathrm{Par}(X)) = 1$. For the hidden nodes $P_{\mathcal{B}'}(x \mid \mathrm{Par}(X)) = P_{\mathcal{B}}(x \mid \mathrm{Par}(X))$. Therefore:

$$\frac{P'(\boldsymbol{x})}{Q(\boldsymbol{x})} = \prod_{x \in e} P_{\mathcal{B}}(x \mid \mathrm{Par}(X))$$

In practice there is no need to construct $\mathcal{B}'$ and we can sample directly from $\mathcal{B}$. We process the nodes in topological order. If the node is unobserved, we sample a value for it based on its CPD and the value already sampled for its parents. If the node is observed, we set it to its observed value and multiply the sample weight w by $P_{\mathcal{B}}(x \mid \mathrm{Par}(x))$. It is easy to verify that, at the end of this process, we indeed have $w = \prod_{x \in e} P_{\mathcal{B}}(x \mid \mathrm{Par}(x))$.

Recall that, in general, the efficiency of importance sampling depends on how close the sampling distribution (represented by $\mathcal{B}'$ in likelihood weighting) is to the true posterior distribution. In the best case all the evidence is in root nodes, in which case $\mathcal{B}'$ represents exactly the posterior distribution. In this case we sample from the correct distribution and get the best rate of convergence. On the other hand, if all the evidence is in leaf nodes we sample from the prior distribution, and the effectiveness of the sampling process depends on how well the prior distribution approximates the posterior distribution. In this case, we can get a very low convergence rate, especially if the evidence is unlikely, making the posterior distribution very different from the prior distribution.

2.4.2 Markov Chain Monte Carlo and Gibbs Sampling

The main competitor of LW is *Markov Chain Monte Carlo (MCMC)*. MCMC is a sampling method that is much less sensitive to the probability of the evidence, and can still work well even if the evidence is unlikely. MCMC is based on the theory of *Markov Chains*, and therefore we provide a short introduction to Markov chains.

A Markov chain is a stochastic process, evolving in discrete time steps. We use $\boldsymbol{X}^0, \boldsymbol{X}^1, \ldots$ to denote the state of the process at the various time steps. For our purposes, we limit ourselves to finite chains, i.e., chains where the number possible states is finite. The distribution for $\boldsymbol{X}^0$ is the *initial distribution*. The distribution of $\boldsymbol{X}^{n+1}$ given the past depends only on $\boldsymbol{X}^n$. The conditional distribution $P(\boldsymbol{X}^{n+1} \mid \boldsymbol{X}^n)$ is called the *transition probability*. If the transition probability does not depend on n then the process is called a *stationary process*. For our purposes it is enough to consider only stationary processes.

Definition 2.8 *An* invariant *or a* stationary *distribution of a Markov Chain is a distribution that persists forever, once it is reached, i.e., π is an stationary distribution iff for every state $\boldsymbol{x}$ we have:*

$$\pi(\boldsymbol{x}) = \sum_{\tilde{\boldsymbol{x}}} \pi(\tilde{\boldsymbol{x}}) P(\boldsymbol{X}^{t+1} = \boldsymbol{x} \mid \boldsymbol{X}^t = \tilde{\boldsymbol{x}}) \tag{2.15}$$

As an example, consider the Markov Chain from Figure 2.7. It has just one variable with three states $\{1, 2, 3\}$. For example $P(X^{t+1} = 1 \mid X^t = 1) = 0.25$, $P(X^{t+1} = 1 \mid X^t = 2) = 0$, and so on. As the reader can verify, the Markov Chain has a stationary distribution π where: $\pi(X = 1) = 0.2$, $\pi(X = 2) = 0.5$ and $\pi(X = 3) = 0.3$.

Often we are interested in Markov Chains that have a unique stationary distribution π. Furthermore, we want the process to converge to π, regardless of the initial distribution. To characterize such Markov Chains, we define the concept of regular chains.

Definition 2.9 *A finite Markov Chain is* regular *if there exists some k such that for every two states $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ the probability of being in state $\boldsymbol{x}_1$ and getting to state*

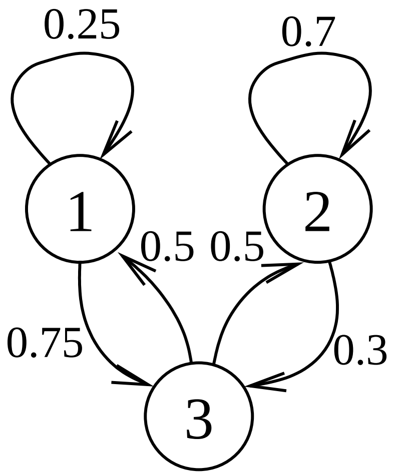


Figure 2.7: A 3-state Markov Chain

$\boldsymbol{x}_2$ *in exactly k steps is larger than zero, i.e., it is possible to get from every state to every state in exactly k steps. For finite chains, the regularity condition is equivalent to a property called* ergodicity*, which we do not define for the general case.*[3]

We are now ready to state the fundamental theorem of Markov Chains:

Theorem 2.10 *An ergodic Markov Chain converges to a unique stationary distribution π regardless of the initial distribution.*

For the proof of Theorem 2.10 and more details, see for example [TK98, Nea93]. As an example, the Markov Chain in Figure 2.7 is ergodic, since it is possible to get from every state to every state in exactly two steps, and therefore the stationary distribution π given above is unique.

Markov Chain Monte Carlo is an algorithm that uses Markov Chains in order to generate the samples for approximate inference. The idea is to construct an ergodic chain such that the stationary distribution is the distribution from which we want to sample. We then run the chain and use its sequence of states as our samples. There are two issues that need to be addressed:

[3]The formal definition of ergodic chains is more complicated as it also applies to infinite state spaces, but since we are only concerned with finite state spaces, we can identify regular chains with ergodic chains.

- How to construct an ergodic chain that has the desired stationary distribution?
- How to use the samples generated from the chain

We answer the second question first. We start by choosing some random assignment to $\boldsymbol{X}^0$ from the initial distribution. We then let the chain run for N steps after which the distribution over the state of the chain should be close to the stationary distribution. This initial phase is called the *burn in* phase. We now pick the state of the chain as our first sample. We can let the chain perform one more transition and use it as our next sample, but in general these two samples will be correlated, and not i.i.d. as we desire. Alternatively, we can let the chain run for M more steps before picking the next sample, in order to reduce the correlation between the samples. The samples we collect over the run of the chain are then used to estimate the desired quantity (just like the samples from LW). Note that the larger M is the weaker the correlation between our samples is, but we pay a penalty in terms of computational cost. In practice M is often chosen to be 1 unless the cost of generating the samples is low compared to the cost of actually using the samples.

We now turn our attention to the task of constructing a Markov Chain with some desired stationary distribution. This task may sound difficult, but in many practical cases it is possible to do so quite easily. The general technique of constructing these Markov Chains is called the Metropolis algorithm [MRR+53]. Here we concentrate on a more specific technique, called Gibbs sampling [GG84] which is often convenient to use with Bayesian networks.

Gibbs sampling starts by generating some random sample, consistent with the evidence (e.g., with likelihood weighting). We now run the Markov Chain — in step k we randomly choose some unobserved variable X_i and sample a value for it, given the current assignment to the other variables (in fact it is enough to consider only CPDs that involve X_i). Formally, we sample from

$$P(X_i \mid x_1[k], \ldots, x_{i-1}[k], x_{i+1}[k], \ldots, x_n[k])$$

Sampling X_i can be done very efficiently: Since all the Markov blanket is known we only need to create a factor over X_i and multiply the relevant values from the

CPDs into it.

A sufficient (although not necessary) condition to ensure that Gibbs sampling results in an ergodic Markov Chain is that whenever we sample X_i there is a non-zero probability for it to be assigned to any one of its possible values (this ensures that we have a positive probability to get from any state to any state in n moves). In particular, if there are no zero entries in the CPDs the chain is ergodic. Furthermore, if the chain is ergodic it is easy to verify using Equation 2.15 that the stationary distribution of the chain is the posterior distribution $P(\boldsymbol{x} \mid \boldsymbol{e})$.

The crucial factor in analyzing Markov Chains is the *mixing rate*, i.e., the rate with which a chain converges to the stationary distribution. Recall that we start the chain with the burn in phase — if the mixing rate is very low the burn in phase must be very long to avoid generating samples from the wrong distribution. Thus, we might need to generate a very large number of samples without being able to use them. Furthermore, when the mixing rate is low, the samples we generate after the burn in phase are very correlated and we might need a very large number of them in order to get a reliable estimate. As an example, consider again the chain in Figure 2.7 but now assume that the probabilities of staying in states 1 and 2 are 0.9999 and the probabilities of leaving them are just 0.0001. If we start in state 1 we will need on expectation 5000 samples in order to move into any other state. Intuitively, it is clear that we need many samples in order to have approximately the right proportion of the samples drawn from states 1 and 2.

Unfortunately, the task of determining the mixing rate of a Gibbs sampler on some given Bayesian network is quite difficult. It is not even simple to decide whether the chain has mixed given a certain burn in trajectory. Thus, the questions of when we can start using the samples and how many samples are needed for a reliable estimate are hard and make the use of Gibbs sampling a non-trivial task. Nonetheless, note that the performance of Gibbs sampling does not degrade when the evidence is unlikely. Thus, Gibbs sampling often works well in situations were exact inference is not possible and likelihood weighting performs poorly.

2.5 Most Probable Explanations

So far we discussed the problem of finding the marginal probability distribution over some variables given some evidence. We now consider a different problem, called the *Most Probable Explanation (MPE)* problem. In the MPE problem, we are given a Bayes net $\mathcal{B}$ and some evidence $\boldsymbol{E} = \boldsymbol{e}$ and are asked to find the most likely configuration of the remaining variables given the evidence, i.e., the best explanation for the evidence. An important generalization of the MPE problem is to find the K most probable explanations for some $K \geq 1$.

2.5.1 Finding the MPE

We first note that, even without evidence, the most likely value of a certain variable A, i.e., the value a_i that maximizes $P(A = a_i)$, is not necessarily a part of the MPE assignment (and thus we cannot use simple greedy algorithms that pick the most likely value one variable at a time). For example, consider a very simple Bayesian network $A \rightarrow B$ where both A and B are binary and their CPDs are:

A	
$\bar{a}$	0.6
a	0.4

;

B	$A = \bar{a}$	$A = a$
$\bar{b}$	0.5	0.2
b	0.5	0.8

The most likely value for A is $\bar{a}$ but the MPE in this case is $A = a, B = b$ with probability $0.4 \times 0.8 = 0.32$ (the most likely assignment involving $A = \bar{a}$ has the probability $0.6 \times 0.5 = 0.3 < 0.32$).

It is easy to adapt the variable elimination and clique tree algorithms to the MPE problem. We first consider the case of $\boldsymbol{E} = \emptyset$. Given a Bayes net $\mathcal{B}$ over the variables $\boldsymbol{X} = X_1, \ldots, X_n$ we can write:

$$\max_{X_1,\ldots,X_n} P(X_1, \ldots, X_n) = \max_{X_1,\ldots,X_n} \prod_{i=1}^{n} P(X_i \mid \mathrm{Par}(X_i)). \tag{2.16}$$

Note the similarity between Equation 2.16 and Equation 2.2, where the summation operation is replaced by a maximization operation. Indeed, we can use the same technique we used for variable elimination and push the maximizations inside. For example, if we want to find the MPE for the Bayes net from Figure 2.1 we can write:

$$\max_{W,D,S,C,I} P(W,D,S,C,I) = \max_{S,C,I} P(I \mid C,S) \max_{W} P(W)P(S \mid W) \max_{D} P(D)P(C \mid W,D). \tag{2.17}$$

Note again the similarity between Equation 2.17 and Equation 2.3. Thus we can use the variable elimination algorithm to solve the MPE problem where the summation operations are replaced by maximization operations. Since the clique tree algorithm is based on the variable elimination algorithm, we can use the same idea of replacing summations by maximizations in Figure 2.5 and get a version of the clique tree algorithm that solves the MPE problem. Evidence can be handled in the usual way of multiplying the potentials by indicator functions.

The calibration algorithm for MPE is called *max-calibration*, and the potentials in the clique tree are called *max-potentials*. Each entry in the max-potential represents the probability of the most likely assignment for the network variables that is consistent with the particular entry. A sepset S between the cliques C_1 and C_2 is the result of maximizing out the variables which are not in the sepset, i.e., $\max_{C_1-C_2} C_1 = \max_{C_2-C_1} C_2$. The proofs are very similar to the proofs of the standard clique tree algorithm and can be found in [Daw92] or [Jen96].

After performing max-calibration it is an easy task to extract the MPE assignment from the clique tree. We arbitrarily choose one of the cliques as root and find the MPE using a downstream pass. Suppose we order our cliques as $C_1, C_2, \ldots, C_n$ where C_1 is the root. We first choose the most likely assignment for the variables in clique C_1. When we get to clique C_i we choose the most likely assignment that is consistent with our partial assignment over $C_1, \ldots, C_{i-1}$ (ties are broken arbitrarily). It is easy to verify that this algorithm will result in an assignment which is the MPE. Assume first that $\boldsymbol{x}$ is the unique MPE with probability p. Then the projection of $\boldsymbol{x}$ to the max-potential of C_1 must have probability p that must be larger than any other entry in the max-potential, and therefore will be chosen by our algorithm. The situation in the other cliques is similar and thus the result returned by the algorithm must be $\boldsymbol{x}$. This argument generalizes to the case of non-unique MPE, showing that the algorithm must choose one of the MPE assignments — the key insight is that because

```
Finding K most likely assignments
1.  Build tree and perform max-calibration
2.  Find MPE x_0
3.  Initialize heap with ⟨P(x_0), x_0, X⟩
4.  For i = 1..K do
   5.  Extract top heap element ⟨P(x), x, Y⟩
   6.  Report x as i-th instantiation
   7.  Partition Y − {x} into subsets Y_1, ..., Y_l
   8.  For each non-empty Y_j  j = 1..l
      9.  Find most likely assignment x_j ∈ Y_j and P(x_j)
      10. Insert ⟨P(x_j), x_j, Y_j⟩ into heap
```

Figure 2.8: Finding the K most likely instantiations. $\mathcal{X}$ represents all the possible instantiations.

of the way we order the cliques the algorithm will end up with a consistent assignment and it is easy to verify that no other assignment can be more likely.

2.5.2 Finding the K Most Likely Explanations

It is possible to generalize the max-calibration algorithm to generate the K most likely assignments rather than just the single most likely assignment [Nil98]. Assume we have a max-calibrated tree with the cliques $C_1, \ldots, C_n$ sorted in a way such that every clique on the path between C_i and C_j appears between C_i and C_j in our order (we can easily find such an order, for example using DFS traversal of the tree).

Let $\mathcal{X}$ be the set of all possible instantiations, and let $\mathcal{A}$ be the set of instantiations that were already generated by the algorithm (initially $\mathcal{A} = \emptyset$). The idea of the algorithm shown in Figure 2.8 is to partition $\mathcal{X} - \mathcal{A}$ into subsets $\mathcal{Y}_1, \ldots, \mathcal{Y}_m$ where for each subset $\mathcal{Y}_j$ we also keep the the most likely instantiation within it $\boldsymbol{x}_j \in Y_j$. Note that the most likely instantiation in $\mathcal{X} - \mathcal{A}$ must be the most likely instantiation among the candidates $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$. Thus, we keep all the candidates in a heap sorted by the likelihood of the instantiations, such that at each step the top element in the heap represents the next most likely instantiation. In each step we extract the top element $\boldsymbol{x}_j$ from the heap and report $\boldsymbol{x}_j$ as the next most likely instantiation. We then refine our partition scheme by partitioning $\mathcal{Y}_j - \{\boldsymbol{x}_j\}$. For each new subset we find the most likely assignment and insert it to the heap.

The obvious question is how to define the partition scheme. A good partition scheme must have two properties:

- The number of subsets m is not too large
- It is easy to find the most likely instantiation in every subset $\mathcal{Y}_j$

Note that these properties tend to be conflicting. At one extreme, if we keep all the instantiations in one set, we get the first property but not the second; on the other if we keep small subsets such that each contains just one instantiation we have the second property but not the first. Therefore the heart of the algorithm is identifying a good partition scheme and a way to maintain it. Nilsson [Nil98] suggests to associate each subset $\mathcal{Y}$ with some clique C_i. $\mathcal{Y}$ is defined by an assignment $\boldsymbol{y}$ to all the variables in the cliques $C_1, \ldots, C_{i-1}$ and by a set $\mathcal{F}$ of forbidden assignments to the variables in $C_i - C_{i-1}$. We denote such a subset $\mathcal{Y} = \text{Assign}(i, \boldsymbol{y}, \neg\mathcal{F})$. It is important to make sure that the subsets have an empty intersection. If we have another subset $\mathcal{Y}' = \text{Assign}(j, \boldsymbol{y}', \neg\mathcal{F}')$ for some $j < i$ then either $\boldsymbol{y}$ is not consistent with $\boldsymbol{y}'$ or $\boldsymbol{y}$ belongs to one of the forbidden assignments in $\mathcal{F}'$.

As an example consider the tree in Figure 2.4(d) and assume all the variables a binary. A possible subset $\mathcal{Y}$ that corresponds to C_3 can be defined as $\mathcal{Y} = \text{Assign}(3, \langle c, \bar{e}, \bar{f}, g, h\rangle, \neg\{a\})$, i.e., the instantiations $\mathcal{Y} = \{\langle A, B, c, D, \bar{e}, \bar{f}, g, h\rangle \mid A \notin \{a\}\}$. A subset $\mathcal{Y}'$ that corresponds to C_2 can be $\mathcal{Y}' = \text{Assign}(2, \langle \bar{e}, f, g, h\rangle, \neg\{\bar{c}\})$. It is easy to verify that $\mathcal{Y}$ and $\mathcal{Y}'$ have an empty intersection since every instantiation in $\mathcal{Y}$ has $F = \bar{f}$ and every instantiation in $\mathcal{Y}'$ has $F = f$. As another example, consider the subset $\mathcal{Y}'' = \text{Assign}(2, \langle \bar{e}, \bar{f}, g, h\rangle, \neg\{c\})$. $\mathcal{Y}''$ has an empty intersection with $\mathcal{Y}'$ since every assignment in $\mathcal{Y}''$ has $F = \bar{f}$ while every assignment in $\mathcal{Y}'$ has $F = f$. $\mathcal{Y}''$ also has an empty intersection with $\mathcal{Y}$ since every assignment in $\mathcal{Y}$ has $C = c$, but $C = c$ is a forbidden assignment in $\mathcal{Y}''$.

The question of course is how to come up with a partition. Recall that at each iteration of the algorithm we have the most likely assignments in each one of our subsets. Among these candidates we find the most likely assignment $\boldsymbol{x}_j \in \mathcal{Y}_j$ and then partition $\mathcal{Y}_j - \{\boldsymbol{x}_j\}$. Thus, for some subset $\mathcal{Y}$ there are two operations that need to be explained: how to find the most likely instantiation $\boldsymbol{x} \in \mathcal{Y}$ and how to partition $\mathcal{Y} - \{\boldsymbol{x}\}$ in a way that makes sure that the new subsets have an empty intersection.

We first explain how to find the most likely instantiation for $\mathcal{Y} = \text{Assign}(i, \boldsymbol{y}, \neg\mathcal{F})$. The assignment to all the variables in $C_1, \ldots, C_{i-1}$ is given by $\boldsymbol{y}$. In C_i we greedily choose the most likely assignment that is consistent with the C_{i-1} assignment and does not appear in the set of forbidden assignments $\mathcal{F}$. We then continue with the cliques $C_{i+1}, \ldots, C_n$ at each time choosing the most likely assignment consistent with our current partial assignment, resulting in the answer $\boldsymbol{x}$ (the proof is similar to the argument at the end of Section 2.5.1). To compute the likelihood of $\boldsymbol{x}$ we note that Equation 2.11 also holds for max-calibration, since it is invariant to the message passing protocol (Figure 2.5) just like regular calibration. Thus we can compute the probability of $\boldsymbol{x}$ using Equation 2.11 in time linear in the number of cliques.

We next turn our attention to the task of partitioning $\mathcal{Y} - \{\boldsymbol{x}\}$, where again $\mathcal{Y} = \text{Assign}(i, \boldsymbol{y}, \neg\mathcal{F})$. We will use the notation $\boldsymbol{x}^{C_i}$ to denote the assignment $\boldsymbol{x}$ restricted to the variables in C_i. In particular, note that when $\boldsymbol{x} \in \text{Assign}(i, \boldsymbol{y}, \neg\mathcal{F})$ we have $\boldsymbol{x}^{C_1,\ldots,C_{i-1}} = \boldsymbol{y}$. Partitioning $\mathcal{Y} - \{\boldsymbol{x}\}$ involves the creation of $n - i + 1$ new subsets. The first is $\text{Assign}(i, \boldsymbol{x}^{C_1,\ldots,C_{i-1}}, \neg\mathcal{F}')$ where $F' = \mathcal{F} \cup \{\boldsymbol{x}^{C_i - C_{i-1}}\}$. In other words, we add the assignment we used in $\boldsymbol{x}$ to the set of forbidden assignments in C_i, creating a subset that agrees with $\boldsymbol{y}$ but is different from $\boldsymbol{x}$ on the assignment in C_i. Obviously, this is not enough: We may have some instantiations that agree with $\boldsymbol{x}$ in $C_1, \ldots, C_i$ but are different in some other C_j for some $j > i$. Thus, for every $j > i$ we add the subset $\text{Assign}(j, \boldsymbol{x}^{C_1,\ldots,C_{j-1}}, \neg\boldsymbol{x}^{C_j - C_{j-1}})$, i.e., all the instantiations that agree with $\boldsymbol{x}$ on $C_1, \ldots, C_{j-1}$ but are different in C_j.

As an example, consider again the tree in Figure 2.4(d) and let $\mathcal{Y}$ be the set $\mathcal{Y} = \text{Assign}(3, \langle c, \bar{e}, \bar{f}, g, h\rangle, \neg\{a\})$. Assume that the most likely assignment in $\mathcal{Y}$ turned out to be $\boldsymbol{x} = \langle \bar{a}, b, c, \bar{d}, \bar{e}, \bar{f}, g, h\rangle$. We partition the set $\mathcal{Y} - \{\boldsymbol{x}\}$ into three subsets:

- $\text{Assign}(3, \langle c, \bar{e}, \bar{f}, g, h\rangle, \neg\{a, \bar{a}\})$
- $\text{Assign}(4, \langle \bar{a}, c, \bar{e}, \bar{f}, g, h\rangle, \neg\{b\})$

- Assign$(5, \langle \bar{a}, b, c, \bar{e}, \bar{f}, g, h\rangle, \neg\{\bar{d}\})$

Since our variables are binary, the first set is empty and is discarded. The second set contains the assignments $\langle \bar{a}, \bar{b}, c, d, \bar{e}, \bar{f}, g, h\rangle$ and $\langle \bar{a}, \bar{b}, c, \bar{d}, \bar{e}, \bar{f}, g, h\rangle$. The third set contains just the assignment $\langle \bar{a}, b, c, d, \bar{e}, \bar{f}, g, h\rangle$. Thus, we created a partition for $\mathcal{Y} - \{\boldsymbol{x}\}$.

At every step of the algorithm we create at most $n = |C|$ new subsets. Thus, in order to generate the K most likely assignments, we create no more than nK assignments. If we are careful about the way we find the most likely assignment in some $\mathcal{Y}$ and compute its probability, we can perform these operations in $O(|\boldsymbol{X}|)$ where $|\boldsymbol{X}|$ is the number of variables in the Bayes net. Thus, the operations in lines 7-9 in Figure 2.8 can be done in $O(nK|\boldsymbol{X}|)$. Since inserting an element to the heap and extracting an element from the heap can be done in time $O(N \log N)$ where N is the number of heap elements, the operations in lines 5-6,10 can be done in $O(nK \log(nK))$. In addition, we need to create and max-calibrate the tree which can be done in $O(|\mathcal{T}|)$ where $|\mathcal{T}|$ is the total size of the tree (including the potentials). Thus, the total complexity of the algorithm is:

$$O\left(|\mathcal{T}| + nK|\boldsymbol{X}| + nK\log(nK)\right)$$

Chapter 3

Hybrid Bayesian Networks

So far, we considered models that include only discrete variables. However, many real-world domains also include continuous variables as well as discrete ones. These domains are known as *hybrid domains*. Examples for such domains include: target tracking [BSLK01], where the continuous variables represent the state of one or more targets and the discrete variables might model the maneuver type; visual tracking (e.g., [PRCM99]), where the continuous variables represent the positions of various body parts of a person and the discrete variables the type of movement; and speech recognition [Jel97, ch.9] where a discrete phoneme determines a distribution over the acoustic signal. In this thesis, we later focus on another such domain, namely fault diagnosis where a physical system contains continuous variables such as flows and pressures and discrete variables such as failure events.

In this chapter we discuss models that represent a probability distribution over both the discrete and the continuous variables. We begin with a short review of the normal distribution which forms the basis for almost all the models used throughout this thesis. We then discuss Bayesian networks that include continuous variables and focus on Bayes nets that represent a probability distribution which is a mixture of Gaussians.

3.1 The Normal Distribution

The family of *normal* distributions (also called *Gaussian* distribution or just *Gaussians*) is by far the most important family of distributions in probability and statistics as well as many other disciplines. There are three main reasons that account for the popularity of the normal distribution:

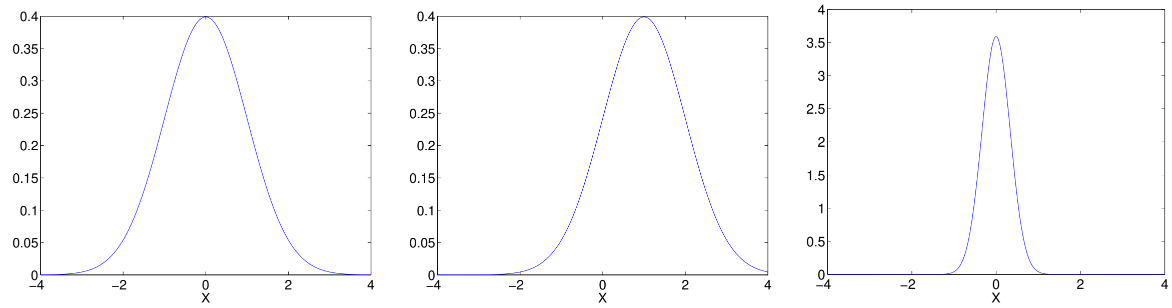
- The normal distribution comes up in the Central Limit Theorem. If $X_1, ..., X_n$ are i.i.d. random variables and we define $Y = \sum_i X_i$ then, under some weak technical conditions, the distribution of Y converges to a normal distribution when $n \to \infty$.
- Normal distributions or distributions that are well approximated by the normal distribution arise naturally in many real-world situations and play a key role in fields such as physics, biology and social sciences.
- The mathematical theory of this family is simple and tractable. The family is closed under operations such as summation, multiplication and conditioning that also have an analytical closed form.

In the univariate case, the normal distribution is characterized by two parameters — the mean μ and the variance σ^2. For mathematical convenience we parameterize the distribution using the variance, which is the square of the standard deviation. The density function has the form:

$$P(X) = \mathcal{N}\left(X; \mu, \sigma^2\right) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-u)^2}{2\sigma^2}\right) \tag{3.1}$$

Note that we use the notation $\mathcal{N}(X; \mu, \sigma^2)$ to denote that the variable X has the normal distribution with mean μ and variance σ^2. Sometimes we will also use the notation $\mathcal{N}(\mu, \sigma^2)$ without explicitly noting the random variable.

Intuitively the mean parameter μ controls the location of the Gaussian, i.e., the value for which the Gaussian gets its maximum value. The variance parameter σ^2 controls how peaked the Gaussian is: the smaller the variance, the more peaked the Gaussian. More formally, the mean and variance correspond to the first two moments



(a) (b) (c)

Figure 3.1: Univariate Gaussians: (a) μ=0, σ^2=1 (b) μ=1, σ^2=1 (c) μ=0, σ^2=$\frac{1}{9}$ (note the different scale on the Y axis)

of the normal distribution, i.e., $\mu = E[X]$ and $\sigma^2 = E[X^2] - E[X]^2$. Figure 3.1 shows some examples of Gaussians with different values for the mean and variance. Note that the density can be larger than 1 (unlike the probability distribution of discrete variables); the constraint is that the integral over the density function is 1.

In the multivariate case, the normal distribution is characterized by two parameters — the mean vector and the covariance matrix. More precisely, the random variables $\boldsymbol{X} = X_1, ..., X_n$ are said to have a normal distribution $\mathcal{N}(\boldsymbol{X}; \boldsymbol{\mu}, \Sigma)$ (where $\boldsymbol{\mu}$ is a vector of size n and Σ is a symmetric positive-definite matrix of size $n \times n$) if:

$$P(\boldsymbol{X}) = \mathcal{N}(\boldsymbol{X}; \boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right) \tag{3.2}$$

Figure 3.2 shows two examples of two-dimensional Gaussians. Intuitively, the contours of multivariate Gaussians look like ellipsoids around the mean μ where every ellipsoid has some constant density. The mean vector determines the center of the ellipsoids and the covariance matrix determines their shape. In general, $\Sigma_{i,i}$ represents the variance of X_i and $\Sigma_{i,j} = \Sigma_{j,i}$ (for $i \neq j$) represents the covariance between X_i and X_j. Once again the mean vector and covariance matrix correspond to the first two moments of the normal distribution, i.e., $\boldsymbol{\mu} = E[\boldsymbol{X}]$ and $\Sigma = E[\boldsymbol{X}\boldsymbol{X}^T] - E[\boldsymbol{X}]E[\boldsymbol{X}]^T$.

We now give a short overview of some of the important properties of the normal distribution. For more details and proofs for the theorems, see almost any introductory textbook for statistics and probability. In particular, see sections 5.7 and 6.7 in the textbook by Stone [Sto96].

Theorem 3.1 *Let* $\boldsymbol{X} = X_1, ..., X_n$ *have a joint normal distribution* $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma)$. *Then* X_i *and* X_j *are independent iff* $\Sigma_{i,j} = 0$.

If all the non-diagonal elements in Σ are zero then all the variables are independent from each other and we say that the Gaussian is *spherical*. Figure 3.2(a) and (c) show a spherical Gaussian where X and Y are independent while Figure 3.2(b) and (d) show an example where X and Y are correlated (note the different scale of the Z-axis).

We often consider joint normal distributions over $\{\boldsymbol{X}, \boldsymbol{Y}\}$ where $\boldsymbol{X} \in \mathbb{R}^n$ and $\boldsymbol{Y} \in \mathbb{R}^m$. We shall use the following notation:

$$P(\boldsymbol{X}, \boldsymbol{Y}) = \mathcal{N}\left(\left(\begin{array}{c} \boldsymbol{\mu}_{\boldsymbol{X}} \\ \boldsymbol{\mu}_{\boldsymbol{Y}} \end{array}\right), \left[\begin{array}{cc} \Sigma_{\boldsymbol{XX}} & \Sigma_{\boldsymbol{XY}} \\ \Sigma_{\boldsymbol{YX}} & \Sigma_{\boldsymbol{YY}} \end{array}\right]\right) \tag{3.3}$$

Here $\boldsymbol{\mu}_{\boldsymbol{X}} \in \mathbb{R}^n$, $\boldsymbol{\mu}_{\boldsymbol{Y}} \in \mathbb{R}^m$, $\Sigma_{\boldsymbol{XX}}$ is a matrix of size $n \times n$, $\Sigma_{\boldsymbol{XY}}$ is a matrix of size $n \times m$, $\Sigma_{\boldsymbol{XY}} = \Sigma_{\boldsymbol{YX}}^T$ is a matrix of size $n \times m$ and $\Sigma_{\boldsymbol{YY}}$ is a matrix of size $m \times m$.

Theorem 3.2 *Let* $\{\boldsymbol{X}, \boldsymbol{Y}\}$ *have a joint normal distribution defined in Equation 3.3. The marginal distribution over* $\boldsymbol{Y}$ *is a normal distribution* $\mathcal{N}(\boldsymbol{Y}; \boldsymbol{\mu}_Y, \Sigma_{YY})$.

One of the most important properties of the family of normal distributions is that it is closed under linear combinations.

Theorem 3.3 *Let* $\boldsymbol{X} = X_1, ..., X_n$ *have a joint normal distribution* $\mathcal{N}(\boldsymbol{X}; \boldsymbol{\mu}, \Sigma)$. *Let* $\beta_0 \in \mathbb{R}$, $\boldsymbol{\beta} = (\beta_1, ..., \beta_n) \in \mathbb{R}^n$ *where* $\boldsymbol{\beta} \neq \boldsymbol{0}$ *and let* $\sigma_W^2 > 0$. *Define* Y *to be the random variable* $Y = \beta_0 + \boldsymbol{\beta}^T \boldsymbol{X} + W$ *where* $P(W) = \mathcal{N}(\boldsymbol{0}, \sigma_W^2)$. *Then:*

- *The conditional distribution* $P(Y \mid \boldsymbol{x})$ *is normal:*

$$P(Y \mid \boldsymbol{x}) = \mathcal{N}\left(Y; \beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}, \sigma_W^2\right)$$

- *The distribution of* Y *is a normal distribution* $P(Y) = \mathcal{N}(Y; \mu_Y, \sigma_Y^2)$ *where:*

$$\begin{aligned} \mu_Y &= \beta_0 + \boldsymbol{\beta}^T \boldsymbol{\mu} \\ \sigma_Y^2 &= \sigma_W^2 + \boldsymbol{\beta}^T \Sigma \boldsymbol{\beta} \end{aligned}$$

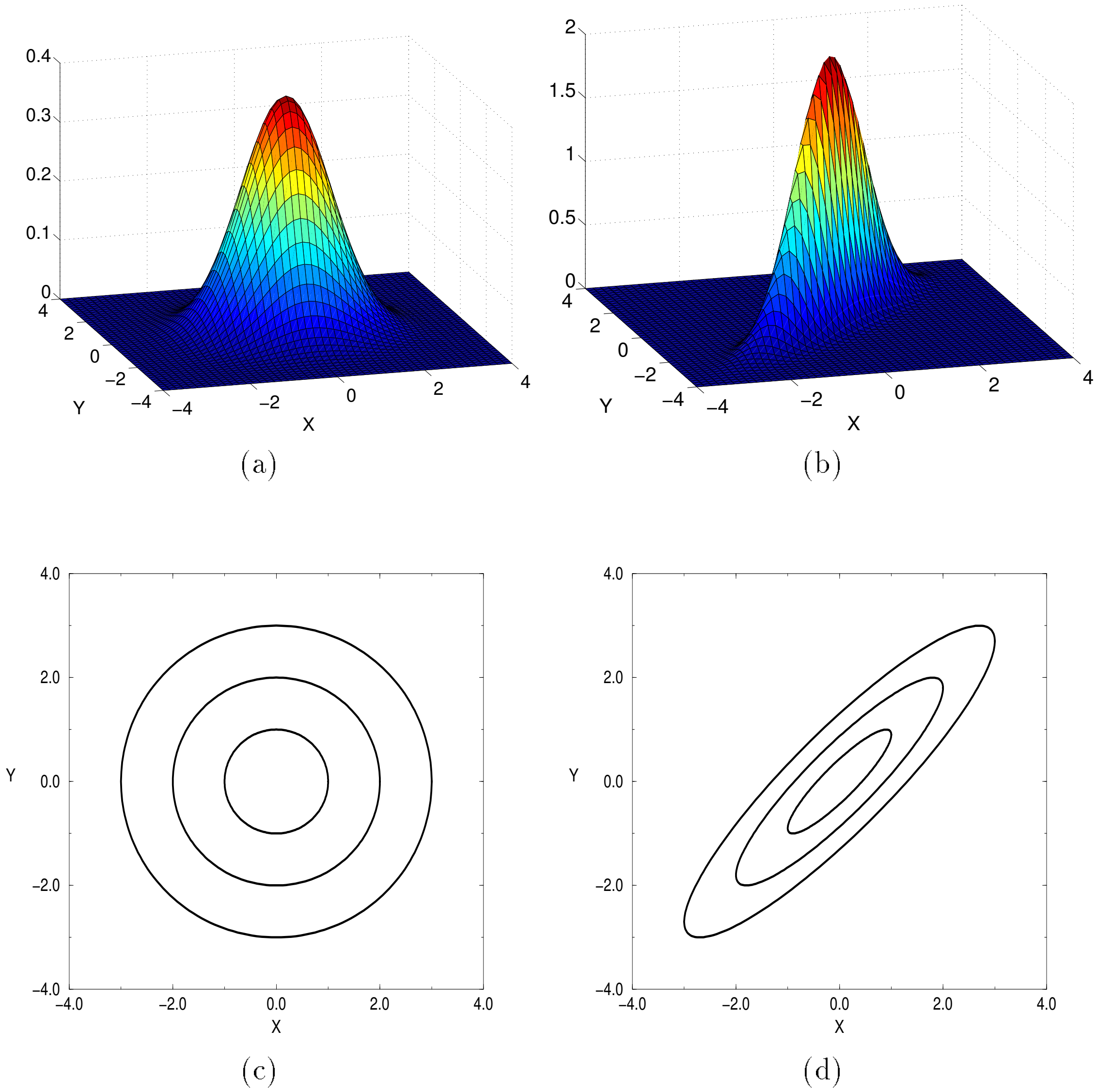


Figure 3.2: Two-dimensional Gaussians: (a), (c) Independent variables (b), (d) Correlated variables (with the correlation coefficient $\rho = 0.9$). In all cases the variances of both X and Y are 1. The contours in (c) and (d) correspond to 1, 2, and 3 standard deviations.

- *Furthermore, the joint distribution over* $\{\boldsymbol{X}, Y\}$ *is a normal distribution where:*

$$\mathrm{Cov}(X_i, Y) = \sum_j \boldsymbol{\beta}_j \Sigma_{i,j}$$

The converse to the last theorem is also true: the result of conditioning is a normal distribution where there is a linear dependency on the conditioning variables.

Theorem 3.4 *Let* $\{\boldsymbol{X}, \boldsymbol{Y}\}$ *have a joint normal distribution defined in Equation 3.3. The conditional distribution* $P(\boldsymbol{Y} \mid \boldsymbol{X})$ *is a normal distribution* $\mathcal{N}(\boldsymbol{Y}; \boldsymbol{\mu}'_{\boldsymbol{Y}}, \Sigma'_{\boldsymbol{YY}})$ *where*

$$\begin{aligned}
\boldsymbol{\mu}'_{\boldsymbol{Y}} &= \boldsymbol{\mu}_{\boldsymbol{Y}} + \Sigma_{\boldsymbol{YX}}\Sigma^{-1}_{\boldsymbol{XX}}(\boldsymbol{x} - \boldsymbol{\mu}_{\boldsymbol{X}}) \\
\Sigma'_{\boldsymbol{YY}} &= \Sigma_{\boldsymbol{YY}} - \Sigma_{\boldsymbol{YX}}\Sigma^{-1}_{\boldsymbol{XX}}\Sigma_{\boldsymbol{XY}}
\end{aligned}$$

In particular, if we use Theorem 3.4 when $|\boldsymbol{Y}| = 1$ we get the following corollary:

Corollary 3.5 *Let* $\{\boldsymbol{X}, Y\}$ *have a joint normal distribution defined in Equation 3.3. The conditional distribution* $P(Y \mid \boldsymbol{X})$ *is a normal distribution* $\mathcal{N}\left(Y; \beta_0 + \boldsymbol{\beta}^T\boldsymbol{X}, \sigma^2\right)$*:*

$$\begin{aligned}
\beta_0 &= \mu_Y - \Sigma_{Y\boldsymbol{X}}\Sigma^{-1}_{\boldsymbol{XX}}\boldsymbol{\mu}_{\boldsymbol{X}} \\
\boldsymbol{\beta} &= \Sigma_{Y\boldsymbol{X}}\Sigma^{-1}_{\boldsymbol{XX}} \\
\sigma^2 &= \Sigma_{YY} - \Sigma_{Y\boldsymbol{X}}\Sigma^{-1}_{\boldsymbol{XX}}\Sigma_{\boldsymbol{X}Y}
\end{aligned}$$

3.2 Linear Gaussians

3.2.1 Definition of Linear Gaussians

We can view Corollary 3.5 as a way to convert a multivariate Gaussian into a Bayesian network. We order the variables in some order $X_1, ..., X_n$. We then use Corollary 3.5 to find the conditional distribution

$$P(X_i \mid X_1, ..., X_{i-1}) = \mathcal{N}\left(X_i; \beta_{i,0} + \sum_{j=1}^{i-1} \beta_{i,j}X_j, \sigma_i^2\right). \tag{3.4}$$

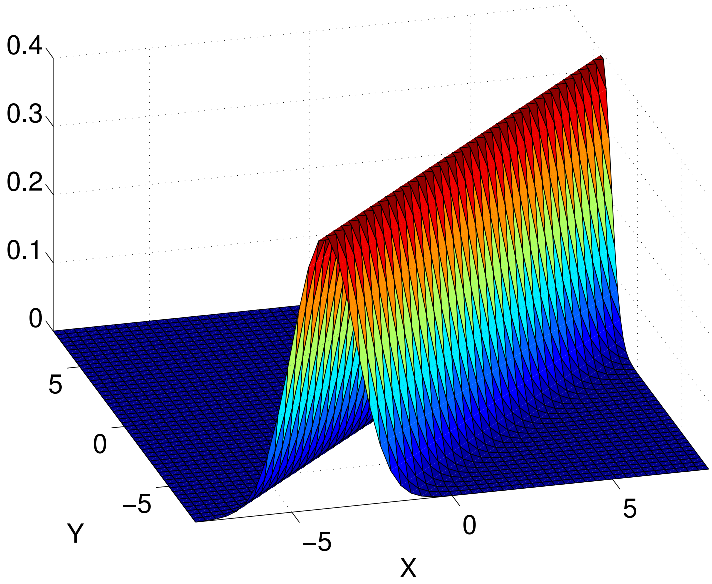


Figure 3.3: A linear CPD $P(Y \mid X) = \mathcal{N}(Y; X - 4, 1)$

We create an edge from X_j to X_i $(1 \leq j < i)$ iff $\beta_{i,j} \neq 0$. The CPD of X_i is called a *linear CPD* and has the form of Equation 3.4 (after dropping all the zero $\beta_{i,j}$'s). Note that a linear CPD for root nodes is simply a univariate Gaussian. A Bayes net in which all the CPDs are linear is called a *linear Gaussian (LG)*. Thus, every multivariate Gaussian can be represented as a linear Gaussian. The converse is also true. Every Bayes net with linear CPDs represents a normal joint distribution which can be recovered using Theorem 3.3.

3.2.2 Canonical Forms

We can perform inference in LGs in many ways. One is to simply convert the LG into a multivariate Gaussian and do the operations on it. The other is to adapt the variable elimination algorithm or the clique tree algorithm from Section 2.3.1. The main difference is that the factors cannot be represented as tables anymore. Naively we might think that we can represent factors as Gaussians, but this is not the case. The reason is that linear CPDs are not Gaussians, but are rather a conditional distribution (unless the number of parents is zero). Figure 3.3 shows an example of the CPD $P(Y \mid X) = \mathcal{N}(Y; X-4, 1)$. The shape of the function is a ridge along the line $X-4$ and the width of the ridge is determined by the variance of $P(Y \mid X)$, which is 1 in this case.

To deal with the problem of representing linear CPDs, we use a different representation called *canonical form* or *canonical characteristics* [Lau92, Mur98]. A canonical form represents a function of the form $e^{Q(\boldsymbol{x})}$ where Q is some quadratic function. More precisely, we define $\mathcal{C}(\boldsymbol{X}; K, \boldsymbol{h}, g)$ (or $\mathcal{C}(K, \boldsymbol{h}, g)$ where we omit the random variables) as:

$$\mathcal{C}(\boldsymbol{X}; K, \boldsymbol{h}, g) \stackrel{def}{=} \exp\left(-\frac{1}{2}\boldsymbol{X}^T K \boldsymbol{X} + \boldsymbol{h}^T \boldsymbol{X} + g\right) \tag{3.5}$$

We can represent every Gaussian as a canonical form. Rewriting Equation 3.2 we get:

$$\begin{aligned} & \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \\ = \quad & \exp\left(-\frac{1}{2}\boldsymbol{x}^T\Sigma^{-1}\boldsymbol{x} + \boldsymbol{\mu}^T\Sigma^{-1}\boldsymbol{x} - \frac{1}{2}\boldsymbol{\mu}^T\Sigma^{-1}\boldsymbol{\mu} - \log\left((2\pi)^{n/2}|\Sigma|^{1/2}\right)\right) \end{aligned}$$

Thus, $\mathcal{N}(\boldsymbol{\mu}, \Sigma) = \mathcal{C}(K, \boldsymbol{h}, g)$ where:

$$\begin{aligned} K &= \Sigma^{-1} \\ \boldsymbol{h} &= \Sigma^{-1}\boldsymbol{\mu} \\ g &= -\frac{1}{2}\boldsymbol{\mu}^T\Sigma^{-1}\boldsymbol{\mu} - \log\left((2\pi)^{n/2}|\Sigma|^{1/2}\right) \end{aligned}$$

However, canonical forms are more general than Gaussians: If K is not invertible, the canonical form is well-defined, but it is not the inverse of a legal covariance matrix. In particular we can represent CPDs such as the one shown in Figure 3.3 as canonical forms (recall that $\mathcal{N}(Y; X-4, 1) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}(Y-(X-4))^2\right]$).

It is possible to perform various operations on canonical forms:

- A canonical form $\mathcal{C}(K, \boldsymbol{h}, g)$ can be initialized by setting $K = 0$, $\boldsymbol{h} = \boldsymbol{0}$, $g = 0$.

- Multiplication is easy. Using Equation 3.5 we get:

$$\mathcal{C}(K_1, \boldsymbol{h}_1, g_1) \cdot \mathcal{C}(K_2, \boldsymbol{h}_2, g_2) = \mathcal{C}(K_1 + K_2, \boldsymbol{h}_1 + \boldsymbol{h}_2, g_1 + g_2) \tag{3.6}$$

- Division is defined in a similar way to multiplication:

$$\frac{\mathcal{C}(K_1, \boldsymbol{h}_1, g_1)}{\mathcal{C}(K_2, \boldsymbol{h}_2, g_2)} = \mathcal{C}(K_1 - K_2, \boldsymbol{h}_1 - \boldsymbol{h}_2, g_1 - g_2) \tag{3.7}$$

- Canonical forms can be extended to be defined over a superset of their variables by increasing the dimensions of K and $\boldsymbol{h}$ and setting the extra entries to zeros.
- Let $\mathcal{C}(\boldsymbol{X}, \boldsymbol{Y}; K, \boldsymbol{h}, g)$ be some canonical form over $\{\boldsymbol{X}, \boldsymbol{Y}\}$ where

$$K = \begin{bmatrix} K_{\boldsymbol{XX}} & K_{\boldsymbol{XY}} \\ K_{\boldsymbol{YX}} & K_{\boldsymbol{YY}} \end{bmatrix} \quad ; \quad h = \begin{pmatrix} h_{\boldsymbol{X}} \\ h_{\boldsymbol{Y}} \end{pmatrix} \tag{3.8}$$

  We define the marginalization of the variables $\boldsymbol{Y}$ as $\int \mathcal{C}(\boldsymbol{X}, \boldsymbol{Y}; K, \boldsymbol{h}, g)\, d\boldsymbol{Y}$ resulting in a function over $\boldsymbol{X}$. The integral is finite iff $K_{\boldsymbol{YY}}$ is positive definite, i.e., it is the inverse of a legal covariance matrix, in which case the result is a canonical form $\mathcal{C}(\boldsymbol{X}; K', \boldsymbol{h}', g')$ given by [Lau92]:

$$\begin{aligned} K' &= K_{\boldsymbol{XX}} - K_{\boldsymbol{XY}} K_{\boldsymbol{YY}}^{-1} K_{\boldsymbol{YX}} \\ h' &= \boldsymbol{h}_{\boldsymbol{X}} - K_{\boldsymbol{XY}} K_{\boldsymbol{YY}}^{-1} \boldsymbol{h}_{\boldsymbol{Y}} \\ g' &= g + \frac{1}{2}\left(|\boldsymbol{Y}| \log(2\pi) - \log|K_{\boldsymbol{YY}}| + \boldsymbol{h}_{\boldsymbol{Y}}^T K_{\boldsymbol{YY}} \boldsymbol{h}_{\boldsymbol{Y}}\right) \end{aligned}$$

- It is possible to enter evidence into a canonical form, i.e., set the values of some of its variables. The result is a canonical form over the unobserved variables. Assume the canonical form $\mathcal{C}(K, \boldsymbol{h}, g)$ is given by Equation 3.8, then setting $\boldsymbol{Y} = \boldsymbol{y}$ results in the canonical form $\mathcal{C}(\boldsymbol{X}; K', \boldsymbol{h}', g')$ given by [Mur98]:

$$\begin{aligned} K' &= K_{\boldsymbol{XX}} \\ h' &= \boldsymbol{h}_{\boldsymbol{X}} - K_{\boldsymbol{XY}} \boldsymbol{y} \end{aligned}$$

$$g' = g + \boldsymbol{h}_{\boldsymbol{Y}}^T \boldsymbol{y} - \frac{1}{2}\boldsymbol{y}^T K_{\boldsymbol{YY}} \boldsymbol{y} \tag{3.9}$$

Since we can perform all these operations with canonical forms, we can adapt the clique tree algorithm from Section 2.3.1 to linear Gaussians, representing the factors as canonical forms. There are two issues that must be addressed. The first issue is that it is not always possible to marginalize canonical forms. Fortunately, this is not a problem. Recall that the need to marginalize comes up when sending a message from clique C_i (over the variables $\boldsymbol{X}_i$) to clique C_j (over the variables $\boldsymbol{X}_j$). We need to compute the marginal over the variables $\boldsymbol{X}_i \cap \boldsymbol{X}_j$, i.e., we need to sum out $\boldsymbol{Y} = \boldsymbol{X}_i - \boldsymbol{X}_j$. Let $\mathcal{C}(K, \boldsymbol{h}, g)$ be the canonical form of the clique of C_i just before sending the message to C_j and let $\mathcal{C}(K', \boldsymbol{h}', g')$ be the canonical form of the correct marginal distribution over C_i. When sending the message to C_j, C_i has already received messages from all its neighbors (except perhaps for C_j), thus all the factors which contain $\boldsymbol{Y}$ in their domain were already multiplied into C_i. It is quite easy to show that this implies that $K_{\boldsymbol{YY}} = K'_{\boldsymbol{YY}}$, i.e., the submatrices of K and K' for $\boldsymbol{Y}$ are the same. Since the marginal distribution of C_i is a Gaussian, K' is positive definite and therefore $K'_{\boldsymbol{YY}}$ is also positive definite. It follows that $K_{\boldsymbol{YY}}$ is positive definite and therefore the marginalization operation is possible.

The other issue is that, unlike the discrete factors, when we instantiate evidence, the evidence variables are no longer a part of the canonical form. It is easy to adapt the clique tree algorithm to this case: We can directly instantiate the evidence in every clique and every sepset that contains any observed variables, and then reduce the dimension of these cliques and sepsets so that they do not include the observed variables.

3.2.3 Conditional Forms

Although canonical forms can be used as the representation in the clique tree algorithm, they have two problems:

- They cannot represent deterministic linear relations, such as $Z = X - 2Y$, since the covariance matrix Σ is not invertible.

- They are numerically unstable, and often lead to serious numerical errors.

The second problem is a serious concern that led to an alternative representation to canonical forms introduced in [LJ01]. We call this representation *conditional forms.*

Definition 3.6 *Let $\boldsymbol{X} = \boldsymbol{H} \cup \boldsymbol{T}$ be a set of continuous variables where $\boldsymbol{H} \cap \boldsymbol{T} = \emptyset$, $|\boldsymbol{H}| = r$ and $|\boldsymbol{T}| = s$. A* conditional form *$[w, \boldsymbol{a}, B, C](\boldsymbol{H} \mid \boldsymbol{T})$ represents a conditional distribution $P(\boldsymbol{H} \mid \boldsymbol{T})$ of the following form:*

$$P(\boldsymbol{H} \mid \boldsymbol{T}) = w \cdot \mathcal{N}\left(\boldsymbol{H}; \boldsymbol{a} + B\boldsymbol{T}, C\right)$$

where $\boldsymbol{a}$ is a vector with dimension r, B is a matrix with dimensions $r \times s$ and C is a matrix with dimensions $r \times r$.

If $C = 0$ then the conditional form represents a deterministic relation between $\boldsymbol{H}$ and $\boldsymbol{T}$. Furthermore, we can easily represent a linear CPD $P(X \mid \mathrm{Par}(X))$ using conditional forms (where $\boldsymbol{H} = X$ and $\boldsymbol{T} = \mathrm{Par}(X)$). Finally, if $\boldsymbol{T} = \emptyset$ then B is redundant and the conditional form represents a multivariate Gaussian. All the operations involved in the clique tree algorithm can be performed using conditional forms [LJ01], leading to a variant of the algorithm that uses conditional forms as its main representation.

Conditional forms do not have the numerical problems that canonical forms suffer from, and thus should be preferred when implementing the clique tree algorithms for linear Gaussian networks or conditional linear Gaussian networks (discussed in the next section). However, the operations on conditional forms are more complex and add an extra layer of complication to the algorithms. We shall therefore explain the algorithms in the context of canonical forms, keeping in mind that the discussion carries over to conditional forms as well.

3.3 Conditional Linear Gaussians

LGs form the basis for more sophiticated models which we shall explore throughout this thesis. In this section, we discuss one such model, namely *Conditional Linear*

Gaussians (CLGs). CLGs are Bayesian networks that combine both discrete and continuous variables, and thus let us represent a richer class of distributions than either LGs or discrete Bayes nets.

3.3.1 Definition of CLGs

Definition 3.7 *A conditional linear Gaussian (CLG) is a Bayesian network containing both continuous variables (denoted as $\mathbf{\Gamma}$) and discrete variables (denoted as $\mathbf{\Delta}$), with the following restrictions:*

- *A discrete node cannot have a continuous parent; thus, all the CPDs for discrete nodes can be represented as in discrete BNs.*
- *The CPD of any continuous variable is a linear CPD given any combination of the discrete parents. More formally, if a node Y has the parents $\{X_1, \ldots, X_k\} \subseteq \mathbf{\Gamma}$ and $\boldsymbol{D} = \{D_1, \ldots, D_l\} \subseteq \mathbf{\Delta}$, we define its CPD using the following parameters: for every $\boldsymbol{d} \in \mathrm{Dom}(\boldsymbol{D})$, we have $\beta_{\boldsymbol{d},0}, \ldots, \beta_{\boldsymbol{d},k}$ and $\sigma^2_{\boldsymbol{d}}$. The CPD is then defined as:*
$$P(Y \mid \boldsymbol{x}, \boldsymbol{d}) = \mathcal{N}\left(Y; \beta_{\boldsymbol{d},0} + \sum_{i=1}^{k} \beta_{\boldsymbol{d},i} x_i, \sigma^2_{\boldsymbol{d}}\right).$$

Note that, if all the discrete variables are given, then the CPDs of the continuous variables are all linear CPDs. Thus, given any assignment of the discrete variables, a CLG is reduced to a LG and therefore represents a normal distribution. It follows that the joint distribution represented by a CLG is a mixture of Gaussians where every mixture component corresponds to an instantiation of the discrete variables.

As an example of a simple CLG consider the network in Figure 3.4(a). Throughout the thesis we will use the convention that, in hybrid Bayesian networks, discrete variables are depicted as squares or rectangles while continuous variables are depicted as circles or ellipses. This is a very simplified model of a room temperature sensor. The variable T is the room temperature with the prior distribution $\mathcal{N}(T; 20, 25)$, i.e., T has a mean of $20^o C$ with a standard deviation of $5^o C$. The discrete variable *OK* indicates whether the sensor is working properly or is faulty. With probability 0.9, *OK* is *True*. The actual reading of the sensor R depends on the temperature and on

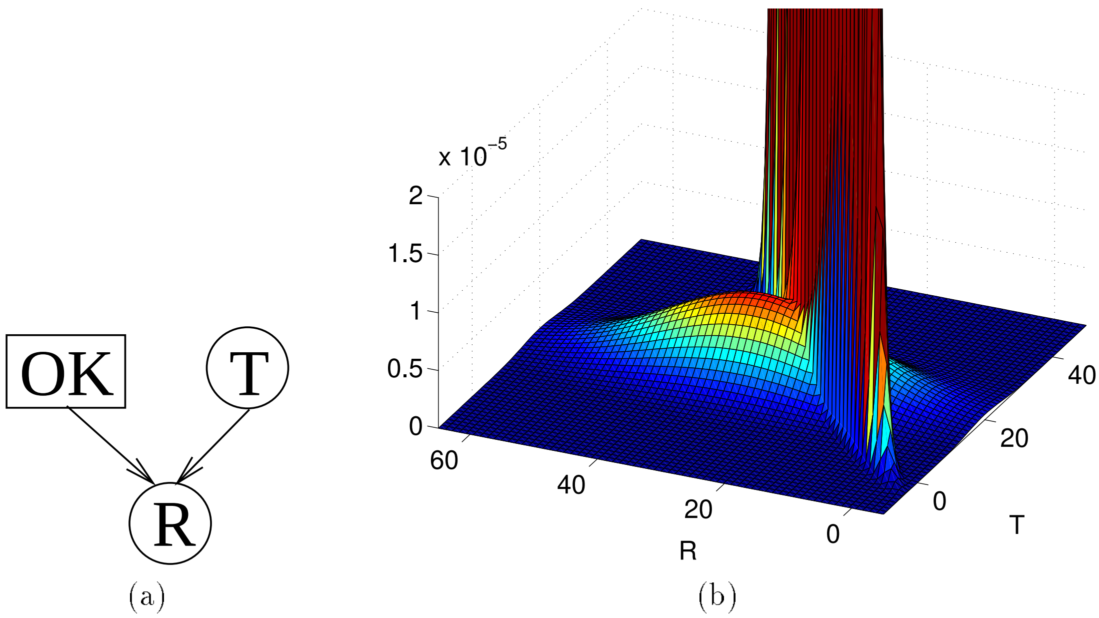


Figure 3.4: An example of a CLG: (a) The model — the sensor reading R depends on the temperature T and on the sensor's state (b) The joint distribution of T, R

the sensor's state. If the sensor is working properly, R is approximately equal to the actual temperature — it adds some Gaussian noise with standard deviation of $1^o C$. However if the sensor is faulty, its reading does not depend on the actual temperature and instead typically shows values around $30^o C$ with a large uncertainty (standard deviation of $15^o C$). The behavior of the sensor can be modeled using the following CPD:

$$P(R \mid OK, T) = \begin{cases} \mathcal{N}(R; T, 1) & OK = \mathit{True} \\ \mathcal{N}(R; 30, 225) & OK = \mathit{False} \end{cases}$$

The resulting distribution of T, R, after marginalizing out OK, is shown in Figure 3.4(b). The distribution is a mixture of two Gaussians. The peaked Gaussian represents the case of $OK = \mathit{True}$ (note that the density of this Gaussian goes beyond the scale of the Z-axis) and the other Gaussian corresponds to $OK = \mathit{False}$.

3.3.2 Representation of Factors

As usual, we are interested in the problem of probabilistic inference, and once again we take the approach offered by the clique tree algorithm. We shall develop a version of the clique tree algorithm which is suited for CLGs, proposed in [Lau92] and often called *Lauritzen's algorithm*. The key decision in adapting the clique tree algorithm into CLGs is how to represent the clique potentials and the sepsets, and in particular how to represent the functions over continuous variables.

Figure 3.4(b) shows an example of a distribution over a continuous variable, and in this case the marginal distribution is a mixture of Gaussians. In general, the marginal distribution over a subset of variables $\{\boldsymbol{D}, \boldsymbol{X}\}$ (where $\boldsymbol{D} \subseteq \boldsymbol{\Delta}$ and $\boldsymbol{X} \subseteq \Gamma$) can be expressed as a mixture of Gaussians over $\boldsymbol{X}$ for every assignment to $\boldsymbol{D}$. The sum of the probabilities of the mixture components for a certain $\boldsymbol{D} = \boldsymbol{d}$ represents the probability $P(\boldsymbol{D} = \boldsymbol{d})$. Note that if $\boldsymbol{D}$ are all the discrete variables in the network then for every $\boldsymbol{D} = \boldsymbol{d}$ we get a single Gaussian rather than a mixture. This leads to a natural way to represent an intermediate function over $\{\boldsymbol{D}, \boldsymbol{X}\}$ as a table with one entry for every possible assignment to $\boldsymbol{D}$. Each entry contains a mixture of Gaussians over the continuous variables.

A mixture of Gaussians over the variables $\boldsymbol{X}$ can be represented as a set of pairs $\langle w_i, \mathcal{N}(\boldsymbol{X}; \boldsymbol{\mu}_i, \Sigma_i)\rangle$ where w_i is the weight of the i-th mixture component. If $\sum_i w_i = 1$ we say that the mixture is *normalized* in which case it represents a probability density function: with probability w_i, $\boldsymbol{X}$ has the normal distribution $\mathcal{N}(\boldsymbol{X}; \boldsymbol{\mu}_i, \Sigma_i)$. Alternatively, we can also represent a mixture using canonical forms as defined in Equation 3.5. In this case the weight w_i becomes a part of the canonical form by incorporating it into the parameter g_i, i.e., we add $\log w_i$ to g_i.

Unfortunately this natural representation has two serious problems:

- The size of the representation is not fixed, and might vary during the actual run of the algorithm. For example, a function over one continuous variable can be as simple as one weight and univariate Gaussian, and as complex as a mixture with as many as $|\mathrm{Dom}(\Delta)|$ components.
- Some of the operations used in the clique tree algorithm are not defined. In

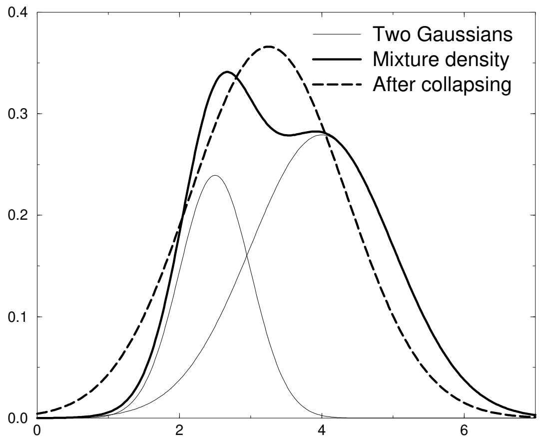


Figure 3.5: Collapsing a mixture of two Gaussians

particular, dividing two mixtures of Gaussians does not result in a mixture of Gaussians and does not have a closed form. Furthermore, marginalization is not defined for mixtures of canonical forms.

We now consider an alternative approach to the mixture representation. We still represent a function over $\{\boldsymbol{D}, \boldsymbol{X}\}$ as a table with one entry for every assignment to $\boldsymbol{D}$, but now every entry contains just one canonical form rather than a mixture. We refer to this data structure as *canonical factors*. Canonical factors have an obvious problem: We just stated that the marginal distribution over continuous variables can be a mixture of Gaussians, but canonical factors do not allow us to represent mixtures (unless all the relevant discrete variables are included in their domain). How will we then represent mixtures if the discrete variables are not in the domain of the function? The answer is that we will not. We shall approximate the mixtures that we cannot represent with just one Gaussian. An example is shown in Figure 3.5, showing a mixture of two Gaussians. Instead of representing the actual density, we shall keep just one Gaussian which will be the result of *collapsing* the two original Gaussians.

How should the collapsing operation be defined? Recall that Gaussians are parameterized by their first two moments: the mean vector and the covariance matrix.

The best we can do to approximate a mixture is to come up with a Gaussian which has the same mean vector and covariance matrix as the entire mixture. The following theorem [Lau96] tells us how to collapse a mixture of Gaussians.

Theorem 3.8 *Let Q be the density function of a normalized mixture of n Gaussians $\langle w_i, \mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)\rangle$. Let $P = \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ be a normal distribution defined as:*

$$\boldsymbol{\mu} = \sum_i w_i \boldsymbol{\mu}_i \qquad (3.10)$$

$$\Sigma = \sum_i w_i \Sigma_i + \sum_i w_i (\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T \qquad (3.11)$$

Then P has the same first two moments (means and covariances) as Q. Furthermore, P minimizes the KL-divergence between Q and any normal distribution, i.e., for any other normal distribution $P' = \mathcal{N}(\boldsymbol{\mu}', \Sigma')$ we have $D(Q \parallel P) \leq D(Q \parallel P')$ with equality iff $P' \equiv P$.

Proof: It is easy to verify that P and Q have the same first two moments by directly computing the first two moments of Q. To prove the second part of the theorem we write

$$\begin{aligned} D(Q \parallel P') &= \int Q(\boldsymbol{x}) \log \frac{Q(\boldsymbol{x})}{P'(\boldsymbol{x})} d\boldsymbol{x} \\ &= \int Q(\boldsymbol{x}) \log \frac{Q(\boldsymbol{x})}{P(\boldsymbol{x})} \frac{P(\boldsymbol{x})}{P'(\boldsymbol{x})} d\boldsymbol{x} \\ &= D(Q \parallel P) + \int Q(\boldsymbol{x})(\log P(\boldsymbol{x}) - \log P'(\boldsymbol{x})) d\boldsymbol{x} \end{aligned}$$

Since both P and P' are Gaussian distributions, both $\log P$ and $\log P'$ are quadratic polynomials. We can therefore write $\log P(\boldsymbol{x}) - \log P'(\boldsymbol{x}) = \boldsymbol{x}^T A \boldsymbol{x} + \boldsymbol{b}^T \boldsymbol{x} + c$. However, Q and P have the same first two moments, and therefore the expectation for any quadratic polynomial is the same with respect to Q and P. Therefore

$$\int Q(\boldsymbol{x})(\log P(\boldsymbol{x}) - \log P'(\boldsymbol{x})) d\boldsymbol{x} = \int P(\boldsymbol{x})(\log P(\boldsymbol{x}) - \log P'(\boldsymbol{x})) d\boldsymbol{x} = D(P \parallel P')$$

Thus, we have

$$D(Q \parallel P') = D(Q \parallel P) + D(P \parallel P') \geq D(Q \parallel P)$$

with equality iff $P \equiv P'$. ▮

Note that the covariance matrix, as defined by the collapsing operation, has two terms: the weighted average of the covariance matrices of the mixture components and a second term which corresponds to the distances between the means of the mixture components — the larger these distances are, the larger the "space" between the mixture components, and thus the larger the variances in the new covariance matrix.

When approximating a mixture of Gaussians by one Gaussian, the quality of the approximation depends on how close the mixture density was to a single multivariate Gaussian and in general can be quite bad. However, there are two reasons why this can be considered a reasonable representation when using the clique tree algorithm:

- As we shall see, the results of using the clique tree algorithm will have the correct first two moments for the continuous variables and the correct distribution over the discrete variables. For many applications this is enough to answer many questions of interest.
- If we insist on finding the exact marginal distributions without any collapsing, we can always add the relevant discrete variables to the cliques and sepsets (with the cost of extra space and running time). Thus we can view the canonical factors representation as a flexible representation, offering a spectrum of approximations trading off efficiency and accuracy.[1]

Having chosen a representation for the potentials and sepsets, we need to verify that we can perform the various operations required by the algorithms. With the important exception of marginalization, which we shall soon discuss, all these operations are a natural generalization of the operations defined on discrete factors and the operations defined for canonical form:

[1]Unfortunately, as we shall see, even the most efficient approximation is often still impractical.

- We can extend the domain of the canonical factor by adding discrete variables (just like in discrete factors) and by adding continuous variables to every canonical form (as defined in Section 3.2.2).
- We multiply (divide) canonical factors by multiplying (dividing) the relevant entries (just like in discrete factors) as defined in Equation 3.6 and Equation 3.7.
- Let $\{\boldsymbol{d}, \boldsymbol{x}\}$ be a set of observations (where $\boldsymbol{d}$ is discrete and $\boldsymbol{x}$ is continuous). We instantiate $\boldsymbol{d}$ by setting the entries which are not consistent with $\boldsymbol{d}$ to zero. We instantiate $\boldsymbol{x}$ by instantiating every canonical form with this evidence.

It remains to discuss marginalization. We can marginalize continuous variables by marginalizing each of the canonical forms. Marginalizing discrete variables is more subtle, and involves combining a few canonical forms into one. More precisely, assume we have a canonical factor defined over $\{\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{X}\}$ (where $\boldsymbol{A}, \boldsymbol{B} \subseteq \boldsymbol{\Delta}$ and $\boldsymbol{X} \subseteq \boldsymbol{\Gamma}$) and we want to marginalize it to a canonical factor over $\boldsymbol{A}, \boldsymbol{X}$. For every value $\boldsymbol{A} = \boldsymbol{a}$, we need to find the entries consistent with $\boldsymbol{a}$ and combine them into one. The operation of combining a few canonical forms is equivalent to the collapsing operation. However, the collapsing operation was defined for a mixture of Gaussians and not for a mixture of general canonical forms. Indeed, combining canonical forms into one is well defined iff the canonical forms have finite first two moments, i.e., can be represented as Gaussians. Note that, even when the collapsing operation is well defined, we only find an approximation to the marginal distribution, as we collapse a mixture of Gaussians into just one Gaussian. Marginalization that involves collapsing Gaussians is called *weak marginalization* (as opposed to *strong marginalization* that does not involve any approximations). The following definition and theorem summarize this discussion.

Definition 3.9 *Marginalization of canonical factors can be either* strong *or* weak. *Strong marginalization does not involve any approximations. It can be performed in one of the following cases:*

- *Only continuous variables are marginalized*
- *The factor is defined over discrete variables only*
- *All the canonical forms are identical*

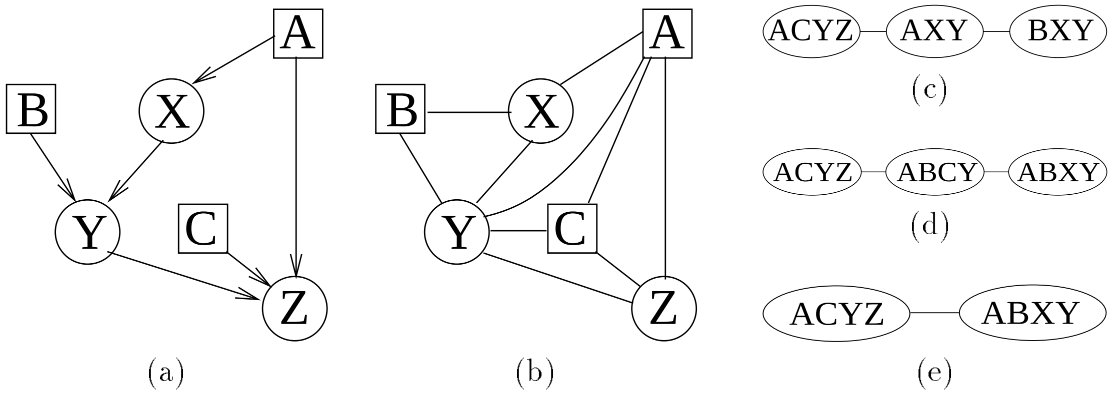


Figure 3.6: (a) A CLG (b) The moralized (and triangulated) graph (c) Resulting clique tree (d) A clique tree with a strong root (e) Yet another clique tree

If none of these conditions hold then the marginalization is weak.

Theorem 3.10 *The weak marginalization operation is well defined iff all the canonical forms can be represented as Gaussians. In this case, the weak marginalization operation can be carried out using Theorem 3.8. Weak marginalization involves collapsing and results in an approximation to the true marginal distribution, preserving the first two moments.*

We end this section by noting that we can replace the canonical forms by conditional forms (defined in Section 3.2.3), leading to *conditional factors*. The discussion regarding canonical factors applies to conditional factors as well. In particular, Definition 3.9 applies to conditional factors. The analog of Theorem 3.10 is that weak marginalization in conditional factors is well defined only when $\boldsymbol{T} = \emptyset$, in which case every entry in the conditional factor represents a multivariate Gaussian. The resulting clique tree algorithm will be very similar to the one described next, but will be much more numerically stable [LJ01]. Again, in order to simplify our discussion, we shall present the algorithm in the context of canonical factors.

3.3.3 Message Passing in Strongly Rooted Trees

Not being able to combine general canonical forms forces us to restrict the elimination order in the variable elimination algorithm or the clique tree algorithm. As an example, consider the network in Figure 3.6(a). After moralization the graph is already triangulated as shown in Figure 3.6(b). If we now extract the maximum cliques and build the clique tree, we get the graph shown in Figure 3.6(c). Unfortunately, at this point we are stuck and cannot send any message in the tree. For example, the clique over $\{B, X, Y\}$ contains the CPDs for $P(B)$ and $P(Y \mid B, X)$. It does not contain the CPD for X and therefore the canonical forms over $\{X, Y\}$ represent a linear CPD and not a Gaussian. It follows that we cannot marginalize out B and thus we cannot send the message. For similar reasons we cannot send the message from the clique over $\{A, C, Y, Z\}$.

One way to make sure that message passing is possible is to create clique trees which are *strongly rooted*. In a strongly rooted tree it is possible to choose a clique R (called a *strong root*) such that there will be no weak marginalization when sending a message towards R. As an example, the clique tree in Figure 3.6(c) is not strongly rooted while the clique tree in Figure 3.6(d) has a strong root (the clique $\{A, B, C, Y\}$). Not having to perform weak marginalization on the upward pass means that, when sending a message towards the root, either there are no continuous variables on the sepset or there was no need to sum out any discrete variables. The following definition formalizes this idea:

Definition 3.11 *A clique R in a clique tree $\mathcal{T}$ is a strong root if every two adjacent cliques C_1 and C_2 such that C_1 is closer to R than C_2 satisfy*

$$(C_1 \cap C_2) \subseteq \mathbf{\Delta} \ \vee \ (C_2 - C_1) \subseteq \mathbf{\Gamma}$$

We defer the discussion of how to create a strongly rooted tree to the next section and continue our discussion of message passing. When a tree is strongly rooted, it is possible to calibrate it using a message passing algorithm. We first run an upstream pass towards R. This pass is possible since there is no weak marginalization involved. We then run the downward pass. This time we will have to perform weak

```
Calibration of a Strongly Rooted Tree
   1. Pick a strong root R
   2. Initialize canonical factors in potentials and sepsets
   3. Multiply potentials by the CPDs
   4. Enter evidence
     5. Discrete: in any potential
     6. Continuous: in all potentials and sepsets;
        remove observed variables from tree
   7. Send upstream messages towards R using Fig. 2.5 protocol
   8. Send downstream messages towards leaves using Fig. 2.5 protocol
```

Figure 3.7: Calibration of a strongly rooted tree

marginalization but it will always be possible. The reason is that in the downward pass we always send messages from cliques that already received all their messages, and therefore represent a probability function. Thus, the canonical forms must represent Gaussians and can be collapsed.[2]

As in discrete clique trees, it is possible to introduce evidence into the tree. Discrete evidence can be introduced in the tree by multiplying one of the potentials by the relevant indicator function (as in the case of discrete clique trees). Continuous evidence can be introduced by instantiating every relevant potential (as in the clique tree for linear Gaussian). The calibration algorithm, including evidence instantiation is shown in Figure 3.7. Note that evidence can also be introduced after calibrating the tree, in which case the continuous evidence also needs to be instantiated in the sepsets.

So far, we have motivated strongly rooted trees as a way to make sure that all the operations on the canonical factors are well defined, and in particular that no collapsing is performed unless the canonical forms represent Gaussians. However, it is possible to find clique trees which are not strongly rooted but in which message passing is still possible. For example, consider the clique tree in Figure 3.6(e). The tree is not strongly rooted, but after multiplying in the CPDs, the clique over

[2]When using conditional factors, $\boldsymbol{T} = \emptyset$ for the strong root R and for any other clique C, $\boldsymbol{T}$ contains the continuous variables that appear on the sepset between C and its neighbor on the path to R. All the other continuous variables in the clique are in $\boldsymbol{H}$.

$\{A, B, X, Y\}$ represents Gaussians over $\{X, Y\}$ (since the CPDs of $P(X \mid A)$ and $P(Y \mid A, X)$ were already multiplied in). Thus, we can weakly marginalize this clique and send the message over $\{A, Y\}$ to the clique $\{A, C, Y, Z\}$. Now it is possible to weakly marginalize the clique $\{A, C, Y, Z\}$ and send a message back to the clique $\{A, B, X, Y\}$. Thus, we have the following theorem:

Theorem 3.12 *Having a strong root is a sufficient, but not a necessary condition for the operations of the message passing algorithm to be well defined in hybrid clique trees.*

The reason we use strongly rooted trees is that it ensures that message passing is not only well defined, but also leads to the exact results. In general, this is not the case when using trees that are not strongly rooted.

Theorem 3.13 *Let $\mathcal{T}$ be a clique tree and R be a strong root in $\mathcal{T}$. After instantiating the evidence and running an upstream pass followed by a downstream pass the tree is calibrated and every potential contains the correct (weak) marginal. In particular, every clique contains the correct probability distribution over the discrete variables and the correct mean and variance of any continuous variable.*

Before proving the theorem we present a useful lemma.

Lemma 3.14 *Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be two sets of random variables with some joint distribution $P(\boldsymbol{X}, \boldsymbol{Y})$ with $|\boldsymbol{X}| = n$ and $|\boldsymbol{Y}| = m$. Let $P(\boldsymbol{Y} \mid \boldsymbol{X}) = \mathcal{N}(\boldsymbol{Y}; \boldsymbol{a} + B\boldsymbol{X}, C)$ where $\boldsymbol{a}$ is a vector of size m, B is a matrix of size $m \times n$ and C is a matrix of size $m \times m$. The first two moments of $P(\boldsymbol{X}, \boldsymbol{Y})$ depend only on the first two moments of $P(\boldsymbol{X})$ and not on the distribution $P(\boldsymbol{X})$ itself.*

The proof for the lemma is in Appendix A.1. We are now ready to prove Theorem 3.13.

Proof: Consider the algorithm in Figure 3.7. Since every CPD has a potential that contains all its variables, then all the multiplications in line 3 are exact, and after performing them the product of the potentials is indeed the correct joint distribution. Similarly it is easy to show that the evidence instantiation in lines 4-6 is also exact, using similar arguments to purely discrete and purely continuous clique trees. Note that, since all the sepsets are initialized to $\mathcal{C}(0, \mathbf{0}, 0)$, there is no need to perform any operations on the sepsets other than removing the observed variables from their domains. It is therefore left to show that the message passing leads to the correct weak marginals in every clique.

The upward pass is simple — all the marginalization operations involved are *strong* marginalizations, and therefore all the operations are exact. Thus, the upward pass is equivalent to running the variable elimination algorithm for the variables in the strong root, and its correctness follows from the correctness of the variable elimination algorithm. The result of the upward pass is the correct (strong) marginal in the strong root R.

The downward pass involves weak marginalization, and therefore its correctness is not immediate. However, we can still show that it results in the correct (weak) marginals in the cliques. The proof is by induction on the distance of the clique from the strong root R. For R itself we already know that we have the correct marginal. Assume now that we have two cliques V and W such that W is the first node on the path from V to R. Let S be the sepset between V and W. We need to show that after sending the message from W to V, V has the correct weak marginal.

We denote the variables in V and W as $\boldsymbol{V}$ and $\boldsymbol{W}$ and the sepset variables as $\boldsymbol{S} = \boldsymbol{V} \cap \boldsymbol{W}$. Also, let $\boldsymbol{W}_1 = \boldsymbol{W} - \boldsymbol{S}$ and $\boldsymbol{V}_1 = \boldsymbol{V} - \boldsymbol{S}$. Next we denote the potentials of W and V (before sending the message) as $\Phi(W)$ and $\Phi(V)$ and we denote the factor on the sepset S (before sending the message) as $\Phi(S)$. Finally, we denote by $\Phi(V)'$ the potential in V after sending the message from W to V, and by $\Phi(S)'$ the message itself, i.e., $\Phi(S)' = \sum_{\boldsymbol{W}_1} \Phi(W)$ where the summation may also involve some integrals.

We first show that after sending the message from W to V, V agrees with W on the marginal distribution of $\boldsymbol{S}$. Since, by induction, W has the correct weak marginal, this will prove that $\Phi(V)'$ represents the correct weak marginal for $\boldsymbol{S}$.

$$\sum_{\boldsymbol{V}_1} \Phi(V)' = \sum_{\boldsymbol{V}_1} \Phi(V) \cdot \frac{\Phi(S)'}{\Phi(S)} = \frac{\Phi(S)'}{\Phi(S)} \cdot \sum_{\boldsymbol{V}_1} \Phi(V) = \frac{\Phi(S)'}{\Phi(S)} \cdot \Phi(S) = \Phi(S)' \quad (3.12)$$

Note that $\sum_{\boldsymbol{V}_1} \Phi(V) = \Phi(S)$ holds because the sepset S has the strong marginal of V computed on the upward pass. If S is not the strong marginal of V then the two cliques will *not* agree on the distribution of $\boldsymbol{S}$ after sending the message.

It is left to show that $\Phi(V)'$ is the correct weak marginal for all the variables in V, i.e., that the first two moments for $\boldsymbol{V}$ are correct given that the first two moments for $\boldsymbol{S}$ are correct. We first observe that every entry in the canonical form in $\Phi(V)'$ must represent the correct conditional Gaussian distribution $P(\boldsymbol{V}_1 \mid \boldsymbol{S})$. The reason is that during the upstream pass all the information about the variables in $\boldsymbol{V}_1$ was already sent to V without any weak marginalization (the rest of the tree cannot have any information about the variables $\boldsymbol{V}_1$ since they are not present in the sepset S). It follows that every entry in $\Phi(V)'$ represents a Gaussian over $\{\boldsymbol{S}, \boldsymbol{V}_1\}$ with the correct first two moments for $\boldsymbol{S}$ and the correct $P(\boldsymbol{V}_1 \mid \boldsymbol{S})$. Using Theorem 3.4 we can write $P(\boldsymbol{V}_1 \mid \boldsymbol{S})$ as a linear function plus some Gaussian noise, and then directly use Lemma 3.14 to get the desired result. ∎

Note that, if the tree is not strongly rooted, the proof breaks down in two places: The upward pass is not exact and Equation 3.12 does not hold. Indeed, in general, if a tree is not strongly rooted, running the message passing algorithm will not lead to the correct first two moments and the tree will not be calibrated, even if all the operations are well defined. In fact, message passing in trees that are not strongly rooted may lead to illegal covariance matrices as shown in the next section.

3.3.4 Ill Defined Message Passing

In this section, we give an example of message passing in a clique tree without a strong root that leads to illegal covariance matrices, and in particular to a negative variance. Consider the CLG in Figure 3.8(a). The CPDs of the discrete variables are uniform and the CPDs of the continuous nodes are defined as follows:

$$P(X \mid A) = \begin{cases} \mathcal{N}(0,2) & A = 0 \\ \mathcal{N}(0,6) & A = 1 \end{cases}$$

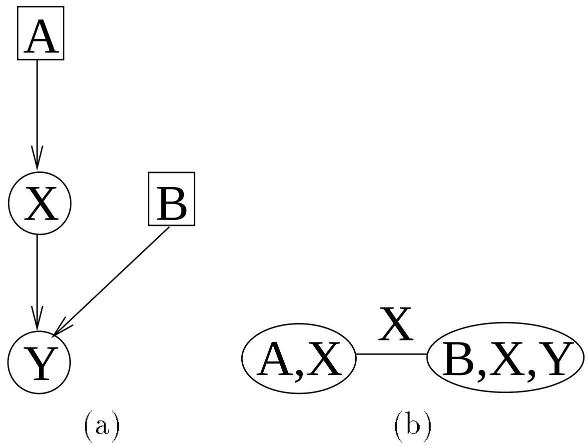


Figure 3.8: (a) A simple CLG (b) A clique tree without a strong root

$$P(Y \mid B, X) = \begin{cases} \mathcal{N}(X, 0.01) & B = 0 \\ \mathcal{N}(-X, 0.01) & B = 1 \end{cases}$$

Assume we use the clique tree in Figure 3.8(b), which is *not* strongly rooted, and assume we have the evidence $Y = 4$. Let us follow the results of the message passing algorithm. To make the presentation easier to follow, we present some of the intermediate results in moments form rather than canonical form.

Let us assume that the message passing algorithm first sends a message from the clique $\{A, X\}$ to the clique $\{B, X, Y\}$. Since the cliques share only the variable X, we collapse the prior distribution of X using Theorem 3.8 and get the message $M_1 = \mathcal{N}(X; 0, 4)$. When we multiply the potential in $\{B, X, Y\}$ by this message we get a mixture of two Gaussians with equal weights:

$$\mathcal{N}\left(X, Y; \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{bmatrix} 4 & 4 \\ 4 & 4.01 \end{bmatrix}\right) \quad \text{when } B = 0$$
$$\mathcal{N}\left(X, Y; \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{bmatrix} 4 & -4 \\ -4 & 4.01 \end{bmatrix}\right) \quad \text{when } B = 1$$

After instantiating the evidence $Y = 4$ we get a new mixture of two Gaussians:

$$\begin{array}{ll} \mathcal{N}(X; 3.99, 0.00998) & \text{when } B = 0 \\ \mathcal{N}(X; -3.99, 0.00998) & \text{when } B = 1 \end{array}$$

The weights of the Gaussians in the new mixture are still the same. The reason is that Y had the same marginal in the original mixture $\mathcal{N}(Y; 0, 4.01)$ for both $B = 0$ and $B = 1$, and therefore the evidence had the same likelihood for the two Gaussians.

We now need to send a message back to the clique $\{A, X\}$. To do so, we need to collapse the two Gaussians. The result of the collapsing is the message $M_2 = \mathcal{N}(X; 0, 15.93)$. Note that, in this example, the evidence caused the variance of X to increase.

To incorporate the message M_2 into the clique $\{A, X\}$ we need to divide it by the message M_1. We then multiply each entry in the potential of $\{A, X\}$ by the result of the division, i.e., by $\frac{M_2}{M_1}$. In particular, for $A = 1$ we need to perform the operation:

$$\frac{\mathcal{N}(0, 6) \cdot \mathcal{N}(0, 15.93)}{\mathcal{N}(0, 4)}$$

This operation can be carried out in canonical forms $\mathcal{C}(K, \boldsymbol{h}, g)$, and for our purposes it is enough to consider only the second order coefficient K. Recall that in general $K = \Sigma^{-1}$ and therefore, for $\sigma^2 = 6$ we get $K = \frac{1}{6}$, for $\sigma^2 = 15.93$ we get $K = \frac{1}{15.93}$ and for $\sigma^2 = 4$ we get $K = \frac{1}{4}$. Thus the new K is equal to

$$\frac{1}{6} + \frac{1}{15.93} - \frac{1}{4} = -0.0206$$

However, $K < 0$ does not represent a legal Gaussian — if we use the inverse transformation we get $\sigma^2 = \frac{1}{-0.0206}$, which is not a legal variance. Thus, doing message passing in a clique tree that is not strongly rooted is not safe.

3.3.5 Strong Triangulation and its implications

Since strongly rooted trees prove to be useful for inference in CLGs, the next natural question is how to construct strongly rooted trees from a given CLG. To address this question we first define *strongly triangulated* graphs.

Definition 3.15 *We say that a chordal graph is strongly triangulated if it corresponds to an elimination order where all the continuous variables are eliminated before the discrete ones.*

Recall from Section 2.3.2 that the nodes in a clique tree correspond to the maximal cliques in a triangulated graph. Theorem 3.16 ([Lei89] theorem $2'$ statement $(iii)'$) tells us that the same connection holds for strongly rooted trees.

Theorem 3.16 *A clique tree has a strong root iff it represents the maximal cliques of a strongly triangulated graph.*

In other words, in order to create a strongly rooted tree, we simply use an elimination order where the continuous variables appear before the discrete ones. We now turn our attention to the computational implications of strong triangulation. Using Theorem 3.16 we can prove an important property of strongly rooted trees.

Definition 3.17 *A* continuous connected component *is a set of continuous nodes* $\boldsymbol{X}$ *such that if* $X_1, X_2 \in \boldsymbol{X}$ *then there is a path between* X_1 *and* X_2 *in the moralized graph where all the nodes on the path are continuous. A* maximal continuous connected component *is a continuous connected component which is not a proper subset of any larger continuous connected component. The* discrete neighbors *of a continuous connected component* $\boldsymbol{X}$*, denoted as* $DN(\boldsymbol{X})$*, are all the discrete nodes which are adjacent to some node* $X \in \boldsymbol{X}$ *in the moralized graph.*

For example, all the continuous nodes $\{X, Y, Z\}$ in Figure 3.6(b) are in one continuous connected component, and all the discrete nodes are its neighbors. The following theorem states the relation between continuous connected components and strongly rooted trees.

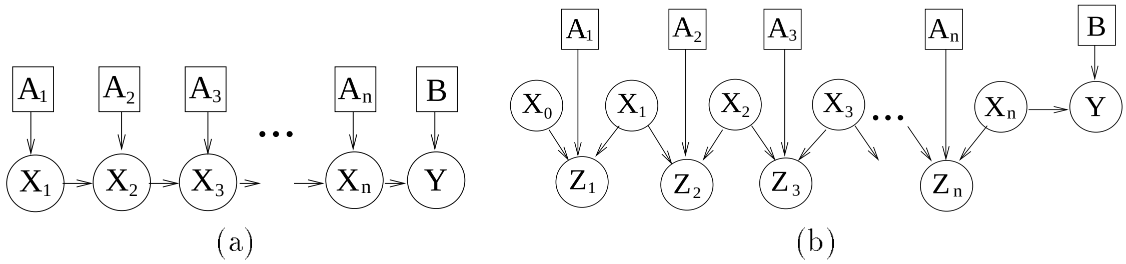


Figure 3.9: CLGs on which strong triangulation leads to exponential trees

Theorem 3.18 *Let $\boldsymbol{X}$ be a maximal continuous connected component and let $\boldsymbol{D} = DN(\boldsymbol{X})$. Then when using strong triangulation and building a clique tree $\mathcal{T}$, there exists a clique in $\mathcal{T}$ that contains all the nodes in $\boldsymbol{D}$ in its domain and at least one node $X \in \boldsymbol{X}$.*

Proof: The proof is by induction on $n = |\boldsymbol{X}|$. If $n = 1$ then all the discrete neighbors are adjacent to one continuous node X. Due to strong triangulation we must eliminate X before any one of the nodes in $\boldsymbol{D}$, and when doing so we create a clique that contains $\{X, \boldsymbol{D}\}$.

For some $n > 1$, let $Y \in \boldsymbol{X}$ be the first node to be eliminated from $\{\boldsymbol{X}, \boldsymbol{D}\}$ (again, due to strong triangulation, no node from $\boldsymbol{D}$ can be eliminated before). When eliminating Y, we add edges from all its discrete neighbors to all its continuous neighbors, thus if $D \in \boldsymbol{D}$ was adjacent to Y it will be adjacent to $\boldsymbol{X} - \{Y\}$ after the elimination of Y. In addition we connect all the continuous neighbors of Y; thus, $\boldsymbol{X} - \{Y\}$ is still a continuous connected component. We are left with a continuous connected component with size $n - 1$ which still has $\boldsymbol{D}$ as discrete neighbors, and we can use the induction hypothesis. ∎

As an example, consider again the CLG in Figure 3.6(a). Since all the continuous variables are in one connected component, all the discrete variables must end up in the same clique in a strongly rooted clique tree together with one of the continuous nodes, as indeed is the case in Figure 3.6(d).

Unfortunately, the size of canonical factors is exponential in the number of discrete variables, and therefore Theorem 3.18 implies that, in many cases, strong triangulation leads to clique trees which are intractable, even when the structure of the CLG is very simple. For example, in the networks shown in Figure 3.9(a) and Figure 3.9(b) all the discrete nodes will end up in one clique, making the respective clique trees exponentially large.

At this point the reader may wonder whether strong triangulation is necessary after all, given the heavy computational penalty we have to pay for it. Perhaps we should be looking for a data structure other than canonical or conditional factors that avoids the problems caused by weak marginalization, and perhaps we can use it in trees which are not strongly triangulated but on which we can still prove a theorem similar to Theorem 3.13. Alternatively, perhaps we should give up on our goal of exact inference in CLGs and develop an approximate inference algorithm on which we can prove an approximation version of Theorem 3.13. Answering these important questions (unfortunately with a negative answer) is one of the main contributions of this thesis, discussed in Chapter 4.

3.4 Approximate Inference in CLGs

As we have seen, exact inference in CLGs is harder than exact inference in discrete Bayesian networks. It is therefore not surprising that in many cases people resort to approximate inference algorithms in CLGs, which we discuss in this section.

3.4.1 Discretization

Discretization is an approach that aims to transform hybrid models into discrete ones. The idea is to partition the domain of every continuous variable into some finite number of sets. For example, given a continuous variable such as age, we can partition its domain to 6 sets 0-10, 10-20, 20-35, 35-50, 50-70 and 70+. We represent the continuous variable for age by a discrete variable that takes one of 6 possible values. We then discretize the CPDs accordingly and end up with a discrete model which we can solve using techniques from Chapter 2.

Unfortunately, discretization has some problems. The first problem is how to partition the continuous domain. A good discretization scheme should have a fine

```
Expectation Propagation
   1. Initialize the approximations t̃_i
   2. Compute the (unnormalized) joint q = ∏_i t̃_i
   3. Do until convergence of t̃_i
      4. Choose some t̃_i
      5. Set q^{-i} = q / t̃_i
      6. Multiply q^{-i} and t_i and minimize KL-divergence to get q' within
         family
      7. Set t̃_i = q' / q^{-i}
      8. Set q = q'
```

Figure 3.10: The Expectation Propagation algoritm

resolution where the posterior probability mass lies, but before we actually perform the inference we do not know the posterior distribution. In order to deal with this problem, [KK97] suggest an approach in which we start with some rough discretization and then iteratively refine it by finer partitions of areas where most of the probability distribution lies.

A more serious problem is that discretization is very problematic when we need to approximate distributions over more than a handful of discretized variables, since the representation of the resulting grid is exponential in the number of variables. For example, the number of parameters to represent a Gaussian over d variables is $O(d^2)$. If we discretize each one of these variables into m values, then we need $O(m^d)$ parameters to describe their joint probability distribution. Thus, not only did we introduce extra approximations into our joint probability distribution, we also ended up with an exponential representation rather than a polynomial one. The situation only becomes worse if we need to approximate a mixture of Gaussians rather than a single Gaussian, since deciding on a good discretization scheme becomes even more challenging.

3.4.2 Expectation Propagation

One can view the clique tree approach as first deciding on some representation of a family of probability distributions and then finding the best approximation of the

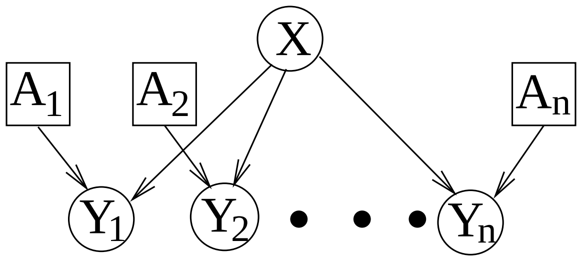


Figure 3.11: A CLG used as an EP example

joint distribution within the chosen family. For example, consider the distribution of Z in the Bayes net from Figure 3.6(a). It is easy to see that the distribution of Z is a mixture of Gaussians, with one mixture component for every combination of A, B, and C (since Y has a different distribution for every value of B the same is also true for Z). However, the structure of the clique tree in Figure 3.6(d) dictates that the distribution for Z would be expressed as a mixture of Gaussians with one mixture component for every combination of just A and C. Thus, the structure of the tree does not allow us to represent the correct joint distribution, and instead we try to find the best approximation which can be expressed using the tree (this is done using the collapsing operation which finds the best single Gaussian approximation to a mixture of Gaussians).

In general we have a set of functions $t_1, t_2, \ldots t_n$ whose product $\prod_i t_i$ is the joint probability distribution (in our case these could be the CPDs and perhaps some indicator functions for the evidence). Our problem is that $\prod_i t_i$ is too complex to represent. Instead, we choose some family of representations and try to find the family member that best approximates the true joint $\prod_i t_i$, e.g., by minimizing the KL distance. The key issue is how to find the best approximation within our family of distributions. Lauritzen's algorithm is a way to do so, but it puts some severe restrictions on the family of distributions. Namely, the family must correspond to some strongly triangulated tree, which is often exponential in the size of the problem. Thus, the family of representations necessary for Lauritzen's algorithm is often too complex by itself, and there is a need for a different approach.

Expectation propagation (EP) [Min01] is an alternative approach for finding a family member that is a good approximation to the joint probability distribution. Instead of treating each term t_i exactly and then approximating the joint that includes the t_i's, EP first approximates t_i with some $\tilde{t}_i$ and then uses the product $\prod_i \tilde{t}_i$ as the approximation for the joint distribution. The algorithm then iteratively improves the approximations for $\tilde{t}_i$. Assume $q = \prod_i \tilde{t}_i$ is our current approximation. To improve our approximation for $\tilde{t}_i$, we first remove $\tilde{t}_i$ from q and define $q^{-i} = \prod_{j \neq i} \tilde{t}_j = \frac{q}{\tilde{t}_i}$. We now combine q^{-i} and the exact t_i and project the result to our family of distributions, leading to a new approximation q'. We can now update our approximation $\tilde{t}_i$ to be the contribution leading from q^{-i} to q', i.e., $\tilde{t}_i \leftarrow \frac{q'}{q^{-1}}$. An important property of the algorithm is that, if our family of distributions is in the exponential family, then the term approximations are also in the same family. The algorithm is given in Figure 3.10.

As an example, consider the CLG in Figure 3.11. We assume that all the discrete variables are binary and that $Y_1, \ldots, Y_n$ are given. We are interested in the distribution of X. It is easy to see that this distribution is a mixture of 2^n Gaussians, since every combination of the discrete variables induces some different multivariate Gaussian over $\{X, Y_1, \ldots, Y_n\}$. When n is large we cannot solve this problem using Lauritzen's algorithm, since according to Theorem 3.18 every strongly triangulated tree must have a clique that contains all the discrete variables.

Instead, we can use the EP approach. We set the terms t_i to correspond to the product of the CPDs $P(A_i)$ and $P(Y_i \mid A_i, X)$. In this example, suppose that we decide to approximate the distribution of X as a single Gaussian, in which case we can represent $q(X)$ as well as $\tilde{t}_i(X)$ using a canonical form. Line 1 of Figure 3.10 involves setting $\tilde{t}_i(X) = \mathcal{C}(X; 0, \mathbf{0}, 0)$ as well as the prior $\tilde{t}_0(X) = t_0(X) = P(X)$ which there is no need to approximate. In line 2, we simply let $q = P(X)$ be the prior (in canonical form). The division in line 5 is easy using canonical form. In line 6 we multiply t_i and q^{-i}, resulting in a mixture of two Gaussians over X, after Y_i is instantiated to its observed value. We then let q' be the result of collapsing the mixture into one Gaussian. The division in line 7 is again simple using canonical forms. Note that, after convergence, we can easily compute other quantities of interest, such as the probability of the evidence or the distribution of the discrete variables A_i using our approximation to the joint distribution.

It can be shown [Min01] that the EP iterations are guaranteed to have a fixed point when the approximations are in an exponential family. EP often leads to good approximations, especially when the joint distribution can be well approximated by a member of the chosen family. Unfortunately, the algorithm can also behave arbitrarily badly, which is not surprising given the analysis in Chapter 4.

3.4.3 Sampling Approach

As in discrete Bayes nets, we can use the sampling approach as an approximate inference algorithm. In particular, we can take the LW algorithm from Section 2.4.1 and apply it on hybrid BNs without any modifications. In fact, the resulting algorithm has a big advantage, because it does not rely on the linearity of the CPDs, and can be applied to non-linear hybrid BNs as well as to CLGs.

In the case of CLGs, we can improve on the LW algorithm by using a technique called *Rao-Blackwellization (RB)*. Recall that the reason we had to resort to approximate inference was that we could not compute analytically the distribution over the random variables in our domain (in the case of CLGs the mixtures are too large). Sampling methods approximate this distribution by generating a set of assignments to the random variables and use them to estimate various properties of the posterior distribution.

The key observation for Rao-Blackwellized sampling is that it not always necessary to sample values for *all* the random variables in our domain: Sometimes, given an assignment to some subset of the variables, the distribution over the other variables becomes tractable and we can deal with it analytically.

Formally, RB suggests to partition our domain into two mutually exclusive subsets $\boldsymbol{X} = \boldsymbol{X}_1 \cup \boldsymbol{X}_2$. For our purposes, we shall assume that the Bayes net does not contain any edges from variables in $\boldsymbol{X}_2$ to variables in $\boldsymbol{X}_1$. Each sample, called a *RB-sample*, consists of an assignment $\boldsymbol{x}_1$ to the variables in $\boldsymbol{X}_1$ and a distribution $P(\boldsymbol{X}_2 \mid \boldsymbol{x}_1)$. In some cases we use weighted sample, in which case we also associate each sample with a weight.

Rao-Blackwellized sampling is based on the following observation. Assume that we want to estimate the expectation of some function $f(\boldsymbol{X})$ given some evidence $\boldsymbol{E} = \boldsymbol{e}$. Then:

$$\begin{aligned} E_{P(\boldsymbol{X}|e)}[f(\boldsymbol{X})] &= \sum_{\boldsymbol{x}_1,\boldsymbol{x}_2} P(\boldsymbol{x}_1, \boldsymbol{x}_2 \mid \boldsymbol{e}) f(\boldsymbol{x}_1, \boldsymbol{x}_2) \\ &= \sum_{\boldsymbol{x}_1} P(\boldsymbol{x}_1 \mid \boldsymbol{e}) \sum_{\boldsymbol{x}_2} P(\boldsymbol{x}_2 \mid \boldsymbol{x}_1, \boldsymbol{e}) f(\boldsymbol{x}_1, \boldsymbol{x}_2) \\ &= \sum_{\boldsymbol{x}_1} P(\boldsymbol{x}_1 \mid \boldsymbol{e}) E_{P(\boldsymbol{X}_2|\boldsymbol{x}_1,e)}[f(\boldsymbol{x}_1, \boldsymbol{X}_2)] \end{aligned}$$

where some of the sums can be replaced by integrals. Thus, if we can efficiently compute the expectation term, there is no need need to sample $\boldsymbol{X}_2$. Instead if we have a set of RB-samples drawn from $P(\boldsymbol{X}_1 \mid \boldsymbol{e})$ with the assignments $\{\boldsymbol{x}_1[1], \ldots, \boldsymbol{x}_1[N]\}$ we can use the following estimate:

$$E_{P(\boldsymbol{X}|e)}[f(\boldsymbol{X})] \approx \frac{1}{N} \sum_{i=1}^{N} E_{P(\boldsymbol{X}_2|\boldsymbol{x}_1[i],e)}[f(\boldsymbol{x}_1[i], \boldsymbol{X}_2)]$$

As usual, it is often hard to sample $\boldsymbol{X}_1$ from the posterior distribution $P(\boldsymbol{X}_1 \mid \boldsymbol{e})$. However, we can view the term $E_{P(\boldsymbol{X}_2|\boldsymbol{x}_1,e)}[f(\boldsymbol{x}_1, \boldsymbol{X}_2)]$ as a function of $\boldsymbol{x}_1$, and this view automatically justifies the use of LW or MCMC as a way to generate samples for $\boldsymbol{X}_1$. For notation convenience, for the remainder of this section we partition the evidence $\boldsymbol{E}$ as $\boldsymbol{E} = \boldsymbol{E}_1 \cup \boldsymbol{E}_2$ where $\boldsymbol{E}_1 \subseteq \boldsymbol{X}_1$ and $\boldsymbol{E}_2 \subseteq \boldsymbol{X}_2$.

When using LW, recall that the weight of the particle corresponds to the likelihood of the evidence given the rest of the sample. The variables in $\boldsymbol{E}_1$ are handled in the usual way. We sample the nodes in topological order and whenever we get to a node $E_i \in \boldsymbol{E}_1$, we multiply the sample weight by $P(e_i \mid \mathrm{Par}(E_i))$. For $\boldsymbol{E}_2$ we first compute the probability distribution $P(\boldsymbol{X}_2 \mid \boldsymbol{x}_1)$ and then use it to compute $P(\boldsymbol{e}_2 \mid \boldsymbol{x}_1)$. Thus, the weight of the sample is set to be $\prod_{E_i \in \boldsymbol{E}_1} P(e_i \mid \mathrm{Par}(E_i)) P(\boldsymbol{e}_2 \mid \boldsymbol{x}_1)$. In addition we also condition the sample's distribution on the evidence $\boldsymbol{e}_2$, i.e., the distribution that we keep with the sample is over $\boldsymbol{X}_2 - \boldsymbol{E}_2$ and can be written as $P(\boldsymbol{X}_2 - \boldsymbol{E}_2 \mid \boldsymbol{x}_1, \boldsymbol{e}_2)$.

For MCMC, recall that the transition distribution is determined by the posterior distribution, which is influenced by the likelihood of the evidence. We create an RB-sample for each possible successor state $\boldsymbol{x}_1^k$, i.e., successor state k is defined by an assignment $\boldsymbol{x}_1^k$, a weight defined as $w^k = \prod_{E_i \in \boldsymbol{E}_1} P(e_i \mid \mathrm{Par}(E_i)) P(\boldsymbol{e}_2 \mid \boldsymbol{x}_1^k)$, and a probability distribution over the remaining variables $P(\boldsymbol{X}_2 - \boldsymbol{E}_2 \mid \boldsymbol{x}_1^k, \boldsymbol{e}_2)$. We then re-normalize the weights of the possible successor states to get a probability distribution over the states, and choose one of them as the next state according to this probability distribution. If we use Gibbs sampling, the successor states are simply all the possible values of some $X \in \boldsymbol{X}_1$, i.e., all the assignments that agree with $\boldsymbol{x}_1$ on the values of the variables in $\boldsymbol{X}_1 - \{X\}$.

In the case of CLGs, there is a very natural choice for the partition of the random variables: Sample the discrete variables and handle the continuous variables analytically. The reason is that, given an assignment to the discrete variables, the distribution over the continuous variables is a Gaussian which we can handle analytically in polynomial time and space. Although it is more expensive to generate an RB sample than it is to generate regular samples, each RB sample contains much more information — it takes many regular samples with the same discrete assignment to be a good approximation of one RB sample.

As an example, assume we want to compute the mean of some continuous variable Z given some evidence $\boldsymbol{E} = \boldsymbol{e}$. We first partition the evidence into $\boldsymbol{e} = \boldsymbol{e}_1 \cup \boldsymbol{e}_2$ where $\boldsymbol{e}_1$ is over the discrete variables and $\boldsymbol{e}_2$ is over the continuous ones. We now generate N samples corresponding to N assignments over the discrete variables $\boldsymbol{X}_1$ $\{\boldsymbol{x}_1[1], \ldots, \boldsymbol{x}_1[N]\}$ with the weights $\{w[1], \ldots, w[n]\}$ (if we use MCMC all the weights are set to 1). We then compute the posterior mean of Z given the evidence $\boldsymbol{e}_2$ as $\mu_Z[i] = E_{P(\boldsymbol{X}_2 \mid \boldsymbol{x}_1[i], e_2)}[Z]$. Note that this operation can be done efficiently in closed form — given the assignment $\boldsymbol{x}_1[i]$ the distribution over $\boldsymbol{X}_2$ is a Gaussian and therefore it is easy to condition it on the evidence $\boldsymbol{e}_2$ and compute the mean $\mu_Z[i]$. Our final estimate is:

$$\mu_Z \approx \frac{\sum_i w[i] \mu_Z[i]}{\sum_i w[i]}.$$

The Rao-Blackwellization technique is not limited to CLGs (although CLGs are one of the most natural models in which it can be applied). Also, even in CLGs, we can use different partitions from the discrete and the continuous variables. For example, we can decide not to sample a certain discrete variable and represent all the Gaussians associated with its different values analytically as a mixture of Gaussians. In general, we can think of a spectrum whose two extremes are exact inference on one side and full sampling on the other. RB gives us points along the spectrum by trading off the number of variables we sample and the complexity of exact inference on the remaining variables.

Chapter 4

Theoretical Analysis

Although CLG models are commonly used, surprisingly little formal work has been done on analyzing their complexity. Obviously CLGs are a generalization of discrete Bayesian networks, and therefore are at least as difficult. However, it is not obvious whether network structures which are easy in the discrete case remain easy for CLGs. In fact, we saw in Theorem 3.18 that the complexity of Lauritzen's algorithm is often exponential in the size of the network even for network structures that are simple in the discrete case. In particular, in the discrete case inference can be done in linear time if the network is a polytree, but Lauritzen's algorithm might still lead to exponentially large cliques. Therefore, it is natural to ask whether we can perform inference efficiently in polytree CLGs.

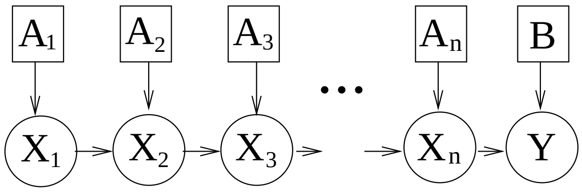


Figure 4.1: Network used in proof of Theorem 4.1

4.1 Hardness Results

For the purposes of our analysis, we assume we are working with finite precision continuous variables. We will restrict ourselves to queries involving only a single discrete variable. These are the simplest queries one can ask, and therefore, only if we can answer these queries efficiently would it make sense to consider more general queries. Thus, we are interested in solving questions of the form, "given some evidence $\boldsymbol{E}$, what is the probability distribution of some discrete variable A?", or phrased as a decision problem, "given some evidence $\boldsymbol{E}$, is the probability that a discrete variable A takes on a value d greater (smaller) than α?"

Theorem 4.1 *The following problem is NP-hard:*

- **Instance:** *A polytree CLG $\mathcal{T}$ with binary discrete variables and some evidence $\boldsymbol{E} = \boldsymbol{e}$.*
- **Question:** *What is the posterior probability $P(A \mid \boldsymbol{e})$ for some discrete variable A?*

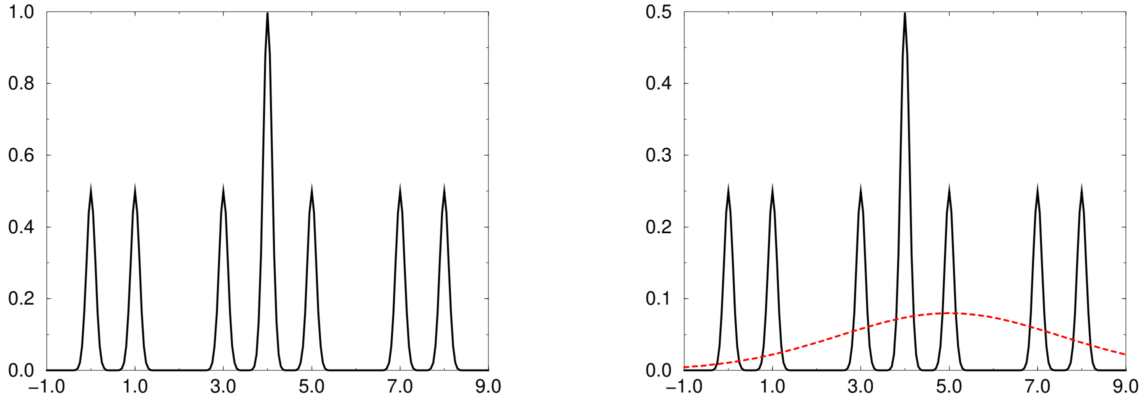
Furthermore, unless P=NP there does not exist any polynomial approximate inference algorithm with absolute error smaller than 0.5.

Proof: Consider the NP-complete *Subset Sum* problem [GJ79]. We are given a multiset $S = \{s_1, s_2, ..., s_n\}$, where each element $s_i \in S$ is a non-negative integer, and a positive integer L. The question is whether there exists a subset $S' \subseteq S$ such that the sum of elements in S' is exactly L.

We reduce this problem to a polytree CLG model, shown in Figure 4.1. The discrete variables (shown as squares) are binary over $\{0, 1\}$. The decision problem in the CLG is whether $P(B = 1 \mid Y = L) > 0.5$ or not. All the discrete variables have a uniform prior and the CPDs of the continuous variables are:

$$P(X_1) = \begin{cases} \mathcal{N}(0, \sigma^2) & A_1 = 0 \\ \mathcal{N}(s_1, \sigma^2) & A_1 = 1 \end{cases}$$

$$\text{For } i = 2, 3, ..., n: \quad P(X_i) = \begin{cases} \mathcal{N}(X_{i-1}, \sigma^2) & A_i = 0 \\ \mathcal{N}(X_{i-1} + s_i, \sigma^2) & A_i = 1 \end{cases}$$



(a) (b)

Figure 4.2: An example of the reduction for $S = \{1, 3, 4\}$ (a) $P(X_3)$ (b) $P(Y)$ as a mixture of Gaussians

$$P(Y) \;=\; \begin{cases} \mathcal{N}(L - \sqrt{2n}, 1) & B = 0 \\ \mathcal{N}(X_n, \sigma^2) & B = 1 \end{cases}$$

We choose $\sigma = \frac{1}{\sqrt{2Cn(n+1)}}$. C is a constant that will be discussed later, but for now we can simply assume $C = 2$.

Before showing the details of the proof, it is helpful to consider a simple example. Assume that $S = \{1, 3, 4\}$ and $L = 5$. We build a network with the four continuous nodes X_1, X_2, X_3, Y and the four discrete nodes A_1, A_2, A_3, B. The marginal distribution of X_3 is shown in Figure 4.2(a). Each mode of X_3 corresponds to some combination of A_1, A_2 and A_3, which in turn corresponds to some subset S'. For example, the mode at $X_3 = 5$ corresponds to the combination $A_1 = 1, A_2 = 0, A_3 = 1$ which in turn corresponds to the subset $S' = \{1, 4\}$. Note that there are two subsets that sum up to 4 ($\{1, 3\}$ and $\{4\}$) which is the reason for the bigger mode at $X_3 = 4$.

The distribution of Y is a mixture corresponding to the possible values of B as shown in Figure 4.2(b). For $B = 1$ we get a contribution from the marginal distribution of X_3 (represented by the solid line) and for $B = 0$ we get a Gaussian with a mean at L and a much larger variance (represented by the dashed line). Intuitively, if we observe $Y = L$, the posterior distribution of B depends on whether this observation was more likely to have come from the solid line or the dotted line. If the solid line is above the dotted line at $Y = L$ then the observation was more likely to have come from one of the mixture components of X_3, and $P(B = 1 \mid Y = L) > 0.5$. Similarly, if the solid line is below the dotted line at $Y = L$ then $P(B = 1 \mid Y = L) < 0.5$. In our example $L = 5$ and the solid line is above the dotted line, since there exists a subset whose elements sum up to 5, thus $P(B = 1 \mid Y = 5) > 0.5$.

We now turn to the task of formalizing the last intuition. Our goal is to show that there exists a subset S' whose elements sum up to L iff $P(B = 1 \mid Y = L) > 0.5$. Since $P(B)$ is uniform $P(B = 1 \mid Y = L) > 0.5$ iff:

$$\frac{P(B=1 \mid Y=L)}{P(B=0 \mid Y=L)} = \frac{P(Y=L \mid B=1)P(B=1)}{P(Y=L \mid B=0)P(B=0)} = \frac{P(Y=L \mid B=1)}{P(Y=L \mid B=0)} > 1$$

First, we compute $P(Y = L \mid B = 0)$:

$$P(Y=L \mid B=0) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(L-\sqrt{2n}-L)^2}{2}\right) = \frac{1}{\sqrt{2\pi}e^n}$$

Let $\boldsymbol{a}^k = \langle a_1, a_2, ..., a_k \rangle$ be some assignment to $\langle A_1, A_2, ..., A_k \rangle$ ($a_i \in \{0, 1\}$). Also, let $S(\boldsymbol{a}^k) = \sum_{i=1}^{k} a_i s_i$. It is easy to show that for $1 \le k \le n$:

$$P(X_k \mid \boldsymbol{a}^k) = \mathcal{N}\left(X_k; S(\boldsymbol{a}^k), k\sigma^2\right)$$

For the first direction, we assume that $\exists S' \subseteq S$ whose elements sum to L, and show that $P(B = 1 \mid Y = L) > 0.5$. Define $\boldsymbol{a}^n$ as $a_i = 1$ iff $s_i \in S'$. Clearly $S(\boldsymbol{a}^n) = L$, and $P(Y = L \mid B = 1, \boldsymbol{a}^n) = \mathcal{N}(L, (n+1)\sigma^2)$. Thus,

$$\begin{aligned} P(Y=L \mid B=1) &\ge P(Y=L, \boldsymbol{a}^n \mid B=1) \\ &= P(\boldsymbol{a}^n \mid B=1)P(Y=L \mid B=1, \boldsymbol{a}^n) \\ &= \frac{1}{2^n} \cdot \frac{1}{\sqrt{2\pi(n+1)\sigma}} \\ &= \frac{\sqrt{2Cn}}{2^n\sqrt{2\pi}} \end{aligned}$$

and in this case

$$\frac{P(Y=L \mid B=1)}{P(Y=L \mid B=0)} \geq \frac{e^n}{2^n}\sqrt{2Cn} > 1$$

Conversely, we assume that there does not exist such $S' \subseteq S$ and show that $P(B=1 \mid Y=L) < 0.5$. Since S' does not exist, we get that, $\forall, \boldsymbol{a}^n$ $S(\boldsymbol{a}^n)$ is an integer different from L, and therefore $\forall \boldsymbol{a}^n$ $|L - S(\boldsymbol{a}^n)| \geq 1$. In this case we get:

$$\begin{aligned}
P(Y=L \mid B=1) &= \sum_{\boldsymbol{a}^n} P(\boldsymbol{a}^n \mid B=1) P(Y=L \mid B=1, \boldsymbol{a}^n) \\
&= \sum_{\boldsymbol{a}^n} \frac{1}{2^n} \cdot \frac{1}{\sqrt{2\pi(n+1)}\sigma} \exp\left(-\frac{(L - S(\boldsymbol{a}^n))^2}{2(n+1)\sigma^2}\right) \\
&\leq 2^n \frac{1}{2^n} \frac{\sqrt{2Cn}}{\sqrt{2\pi}} \exp\left(\frac{-1}{2(n+1)\sigma^2}\right) \\
&= \frac{\sqrt{2Cn}}{\sqrt{2\pi}} e^{-Cn}
\end{aligned}$$

Therefore:

$$\frac{P(Y=L \mid B=1)}{P(Y=L \mid B=0)} \leq \frac{\sqrt{2Cn}e^n}{e^{Cn}} = \frac{\sqrt{2Cn}}{e^{(C-1)n}} < 1$$

We can now explain the role of the constant C. By making C large enough, $P(B=1 \mid Y=L)$ gets arbitrarily close to 1 if S' exists and arbitrarily close to 0 if it does not. We could then use a hypothetical approximation algorithm $\mathcal{A}$ with an absolute error of $\epsilon < 0.5$ to answer the decision problem: Construct a problem instance where $P(B=1 \mid Y=L) < 0.5 - \epsilon$ or $P(B=1 \mid Y=L) > 0.5 + \epsilon$ depending on the existence of S', and answer "yes" iff the algorithm answers that $P(B=1 \mid Y=L) > 0.5$.

To see why this is correct, let $P_{\mathcal{A}}$ be the probability distribution generated by $\mathcal{A}$, such that for every discrete event $|P - P_{\mathcal{A}}| \leq \epsilon$. Assume that S' exists, and therefore $P(B=1 \mid Y=L) > 0.5 + \epsilon$. Then

$$P_{\mathcal{A}}(B=1 \mid Y=L) \geq P(B=1 \mid Y=L) - \epsilon > 0.5 + \epsilon - \epsilon = 0.5$$

Similarly, we can show that if S' does not exist then $P_{\mathcal{A}}(B=1 \mid Y=L) < 0.5$. Therefore, unless P=NP, there does not exist a polynomial time approximate inference

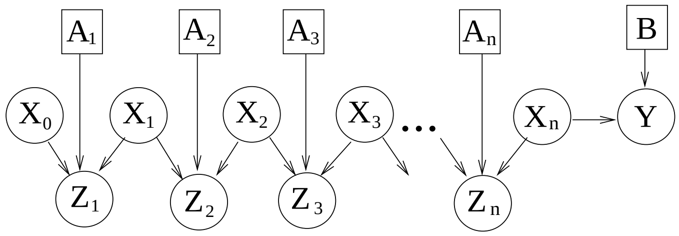


Figure 4.3: Network used in proof of Theorem 4.2

algorithm with an absolute error smaller than 0.5. ∎

In the network shown in Figure 4.1 all the discrete variables are ancestors of the continuous variable Y, and therefore the marginal distribution of Y is a mixture with exponentially many components. One might conjecture that if we restrict the marginals over the continuous variables to be "simple", i.e., to contain at most K components, then the inference problem becomes easier. The following theorem tells us that this is not the case, even for $K = 2$.

Theorem 4.2 *The following problem is NP-hard:*

- **Instance:** *A polytree CLG $\mathcal{T}$ with binary discrete variables such that every continuous node has at most one discrete ancestor, and some evidence $\boldsymbol{E} = \boldsymbol{e}$.*
- **Question:** *What is the posterior probability $P(A \mid \boldsymbol{e})$ for some discrete variable A?*

Furthermore, unless P=NP there does not exist any polynomial approximate inference algorithm with absolute error smaller than 0.5.

Proof: The reduction is very similar to the previous proof, but we have to use a different network structure. We use the structure shown in Figure 4.3. Again, all the discrete variables are binary with uniform distribution, and:

$$\begin{array}{rrcl} & P(X_0) & = & \mathcal{N}(0, \sigma^2) \\ \text{For } 1 \leq i \leq n\text{:} & P(X_i) & = & \mathcal{N}(M, \sigma_2^2) \\ \text{For } 1 \leq i \leq n\text{:} & P(Z_i) & = & \begin{cases} \mathcal{N}(X_i - X_{i-1}, \sigma_1^2) & A_i = 0 \\ \mathcal{N}(X_i - X_{i-1} - s_i, \sigma_1^2) & A_i = 1 \end{cases} \end{array}$$

$$P(Y) = \begin{cases} \mathcal{N}(L - \sqrt{2n}, 1) & B = 0 \\ \mathcal{N}(X_n, \sigma^2) & B = 1 \end{cases}$$

where $M = \sum_j s_j$. Our query is $P(B \mid \boldsymbol{Z}^n = \boldsymbol{0}, Y = L)$, where $\boldsymbol{Z}^k = \boldsymbol{0}$ is a notation for $Z_1 = 0, ..., Z_k = 0$.

The intuition behind the structure is that by setting the evidence $Z_k = 0$ and by having σ_1 be very small, Z_k will force X_k to be close to either X_{k-1} or to $X_{k-1} + s_k$ depending on the value of A_k, resulting in a similar situation to that of Theorem 4.1. In fact, if we could make X_i have exactly the distribution $\mathcal{N}(X_{k-1}, k\sigma^2)$ or $\mathcal{N}(X_{k-1} + s_k, k\sigma^2)$ then we could simply use the proof of Theorem 4.1. Although we cannot get exactly to the desired distribution we can get very close by choosing σ_2 to be very large and σ_1 to be very small. Namely, we can get exactly the desired variance and be within ϵ away from the mean. For some $\epsilon > 0$ we choose:

$$\sigma_1^2 = \sigma^2 \frac{\epsilon}{n \cdot M} \; ; \; \sigma_2^2 = \sigma^2 \left(1 + \frac{\sigma^2}{\sigma_1^2}\right)$$

By induction we can prove that $\forall 1 \leq k \leq n$

$$P(X_k \mid \boldsymbol{a}^k, \boldsymbol{Z}^k = \boldsymbol{0}) = \mathcal{N}(\mu_k, k\sigma^2)$$

where $|\mu_k - S(\boldsymbol{a}^k)| \leq \frac{k\epsilon}{n}$ (see Appendix A.2 for the details of the derivation). We are now almost in the situation of Theorem 4.1. Instead of $P(X_n \mid \boldsymbol{a}^n, \boldsymbol{Z}^k = \boldsymbol{0})$ having the distribution $\mathcal{N}(S(\boldsymbol{a}^n), n\sigma^2)$, it has the distribution $\mathcal{N}(\mu_n, \sigma^2)$, where $|\mu_n - S(\boldsymbol{a}^n)| \leq \epsilon$. If we choose $\epsilon = \frac{\sigma}{n}$, we can easily modify the inequalities from Theorem 4.1 and prove the desired result. ∎

From the reductions we get an immediate corollary regarding the problem of finding the MPE assignment to the discrete variables in a CLG.

Corollary 4.3 *The following problem is NP-hard:*

- **Instance:** *A polytree CLG $\mathcal{T}$ with binary discrete variables such that every continuous node has at most one discrete ancestor, and some evidence $\boldsymbol{E} = \boldsymbol{e}$.*

- **Question:** *What is the most likely assignment of the discrete variables in $\mathcal{T}$ given* $\mathbf{e}$*?*

Proof: This is a direct result of the reduction used in Theorem 4.2. If there exists a subset, then the most likely instantiation must correspond to such a subset and involve $B = 1$. If there does not exist such a subset, then any instantiation involving $B = 0$ is more likely than any instantiation involving $B = 1$. It follows that the most likely instantiation has $B = 1$ iff there exists a solution to the subset sum problem. ∎

4.2 Discussion

The fact that exact inference in polytree CLGs is NP-hard may not be very surprising by itself. Possibly the most popular CLG models are switching linear dynamic systems (SLDS), discussed in Section 8.6. An SLDS is a temporal model but, as we shall see in Chapter 8, it has a close relation to static CLGs, and in particular CLGs with the structure of the network in Figure 4.1 (although SLDSs are slightly more complex, as the discrete variable A_i often depends on the discrete variable A_{i-1}). The continuous variable at the end of the chain has all the discrete variables as ancestors, and therefore its distribution is a mixture of exponentially many Gaussians. It was therefore believed that inference in this case is hard, although no proof was ever given.

The results in this chapter go beyond giving a formal proof for the previously known intuition. First, we concentrate on queries involving just discrete variables whose posterior distribution is easy to represent (but not to infer). Thus, although it is very reasonable to assume that it is hard to represent the exponential mixture over the continuous variables, the case for the discrete variables is less obvious. Second, in Theorem 4.2, the prior distribution of every continuous variable is either a Gaussian or a mixture of two Gaussians. Thus, it is somewhat more surprising that the seemingly benign structure of the continuous variables can produce an NP-hard decision problem. However, perhaps the most important and least obvious fact proven by our analysis, is the fact that even the simplest type of approximate inference with an absolute error smaller than 0.5 is intractable. Consequently, one should not expect to find a polynomial approximate inference algorithm with a useful error bound without further restrictions on the structure or the parameters of the CLGs.[1]

As a technical note, we observe that the proof of the NP-hardness theorems was based on a reduction from Subset Sum, which can be solved using a pseudo-polynomial algorithm, i.e., an algorithm which is polynomial both in $n = |S|$ and in $m = \max_{s \in S}$. The consequence is that the Subset Sum problem is hard only if the magnitudes of numbers in the set S are exponential in n. Thus, when we reduce the problem into a CLG, we get a network in which the variance is exponentially small compared to the means. Although we believe that the problem remains NP-hard even if the means are bounded by a polynomial, we do not currently have a proof for this case.

[1]Knowing this fact in advance might have saved the author some time ...

Chapter 5

Enumeration Algorithm

In the previous chapter we saw that inference for very simple CLGs is NP-hard, and that even approximate inference is not tractable. Given these results, one might conclude that inference in CLG models is a lost cause. Fortunately, the problems we are faced with in practice are often simpler than the ones used in NP-hardness reductions, as they have some special structure or special features which can be exploited by carefully designed algorithms to allow efficient inference.

In this chapter, we concentrate on domains where the probability distribution of the discrete variables is skewed, i.e., a small number of assignments to the discrete variables dominate the rest. The advantage in this case is that we can well approximate an exponentially large mixture using a much smaller mixture of Gaussians.

The property of a skewed distribution is fairly common in practice and appears in many different domains. Consider, for example, the fault diagnosis domain. Discrete variables correspond to the various possible faults, and *a priori* each one of them has a skewed distribution (since faults are unlikely). We first have the null hypothesis, corresponding to the event of no new faults happening (note however, that the null hypotheses may contain faults that happened in the past and are likely to persist). Assuming we have n discrete variables, we then have n hypotheses corresponding to the occurrence of a single new fault, $O(n^2)$ hypotheses corresponding to a pair of new faults, $O(n^3)$ hypotheses corresponding to three new faults, and so on. The key insight is that a combination of many new faults is extremely unlikely. In practice,

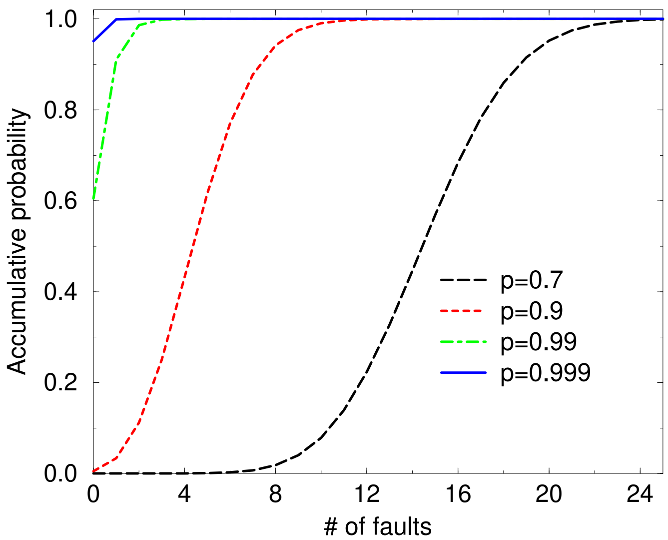


Figure 5.1: Total probability mass taken up by hypotheses of up to k faults out of 50 possible faults for different fault probabilities

it is often the case that *a priori* the null hypothesis combined with the single new fault hypotheses and the pair of new faults hypotheses completely dominate the other hypotheses.

The last observation is demonstrated in Figure 5.1. Here we assume that we have $n = 50$ binary fault variables $A_1, \ldots, A_{50}$ where $P(A_i = 1) = p$. The graph shows the total probability mass taken up by all the hypotheses of up to k faults for various values of p. For example, for $p = 0.99$, hypotheses with two or fewer faults account for 0.986 of the probability mass and hypotheses with three or fewer faults account for 0.998 of the probability mass. Thus, when p is very close to 1, it is reasonable to hope that by generating only $O(n^k)$ hypotheses (for a small k), we will get a good approximation for the exponentially large mixture.

Unfortunately, this observation does not have a direct implication on the size of the mixtures we wish to approximate, since we care about the posterior distribution rather than the prior one. In general, assignments which are likely *a priori* may not be likely *a posteriori*, and vice-versa. This is especially the case when dealing with hybrid domains. Recall that in the hybrid case the likelihood of the evidence, which corresponds to the density function, is not bounded. Thus, even a hypothesis which is very unlikely *a priori*, might have an extremely likely evidence and become the most likely hypothesis *a posteriori*. Furthermore, the evidence might "balance out" the prior likelihoods leading to a situation where we no longer have a small set of hypotheses that dominate the rest.

Nonetheless, in practice the situation appears to be better. It is usually the case that even *a posteriori* there is a small number of hypotheses that dominate the rest. It is also the case that, on expectation, hypotheses which are likely *a priori* remain likely *a posteriori*, and thus concentrating on these hypotheses is a reasonable heuristic, which we shall explore in this chapter. In the fault diagnosis domain, the idea of concentrating on likely hypotheses with a small number of faults is very natural, and was used in [dKW87], although the probabilistic model there was discrete and some strong independence assumptions were made.

We note that the phenomenon of a small number of hypotheses that dominate the rest is not unique to the fault diagnosis domain. As another example, the situation also arises in visual tracking, where given the past evidence most of the hypotheses are very unlikely. For example, given that two people are having a conversation, it is unlikely that one person would start jumping while the other would start dancing.

5.1 Inference Algorithms

We begin by presenting a general framework for our enumeration algorithm as well as other methods such as Rao-Blackwellized sampling. For simplicity, we will assume that the CLG has just one continuous connected component. If this is not the case we can apply our algorithm in every continuous connected component and then use message passing over the discrete variables. One can think of this as building a strongly rooted clique tree where for every continuous connected component we have precisely one clique with all the continuous variables and their discrete neighbors. We use one of the algorithms discussed in this chapter to approximate the distribution within each such clique and then calibrate the tree using message passing. However, we note that in many examples of interest, including the RWGS discussed in Chapter 10, the continuous nodes indeed form one continuous connected component.

We will denote the discrete variables as $\mathbf{\Delta}$, the continuous variables as $\mathbf{\Gamma}$ and their discrete neighbors as $\mathbf{\Delta}_{DN} \subseteq \mathbf{\Delta}$. The joint distribution $P(\mathbf{\Delta}, \mathbf{\Gamma})$ is a mixture of Gaussians with one Gaussian corresponding to each assignment of the discrete variables. Some of these Gaussians might be identical — every two assignments over $\mathbf{\Delta}$ that agree on the variables in $\mathbf{\Delta}_{DN}$ have the same Gaussian.

5.1.1 The Setup

Consider the general query of the form $P(\boldsymbol{Q}_\Delta, \boldsymbol{Q}_\Gamma \mid \boldsymbol{e}_\Delta, \boldsymbol{e}_\Gamma)$ where $\boldsymbol{Q}_\Delta, \boldsymbol{e}_\Delta$ are discrete and $\boldsymbol{Q}_\Gamma, \boldsymbol{e}_\Gamma$ are continuous. The distribution is a mixture of Gaussians, but note that, if $\mathbf{\Delta}_{DN} \nsubseteq \boldsymbol{Q}_\Delta \cup \boldsymbol{E}_\Delta$, then for each combination of $\boldsymbol{Q}_\Delta \cup \boldsymbol{E}_\Delta$ we get a mixture of Gaussians rather than a single Gaussian. Thus, instead of approximating $P(\boldsymbol{Q}_\Delta, \boldsymbol{Q}_\Gamma \mid \boldsymbol{e}_\Delta, \boldsymbol{e}_\Gamma)$ it is simpler to approximate $P(\boldsymbol{Q}_\Delta, \mathbf{\Delta}_{DN}, \boldsymbol{Q}_\Gamma \mid \boldsymbol{e}_\Delta, \boldsymbol{e}_\Gamma)$. The reason is that, by including $\mathbf{\Delta}_{DN}$, we make sure that for every combination of the discrete variables we have just one Gaussian. Note that the addition of $\mathbf{\Delta}_{DN}$ to the query does not cost us anything, since in any case we still need to generate Gaussians for the possible assignments to $\mathbf{\Delta}_{DN}$. Once we have $P(\boldsymbol{Q}_\Delta, \mathbf{\Delta}_{DN}, \boldsymbol{Q}_\Gamma \mid \boldsymbol{e}_\Delta, \boldsymbol{e}_\Gamma)$ it is easy to answer our query by marginalizing out the extra variables. We can use weak marginalization if we only care about the first two moments, but we can also keep the original Gaussians without collapsing them, potentially leading to a mixture of Gaussians for every combination of the discrete query variables.

For example, consider the network in Figure 5.2. Here $\mathbf{\Delta} = \{A, B, C, D, E\}$, $\mathbf{\Gamma} = \{X, Y, Z\}$ and $\mathbf{\Delta}_{DN} = \{D, E\}$. Assume that $\boldsymbol{Q}_\Delta = \{A, E\}$, $\boldsymbol{Q}_\Gamma = \{X\}$, $\boldsymbol{E}_\Delta = \{C\}$ and $\boldsymbol{E}_\Gamma = \{Z\}$, i.e., we are interested in $P(A, E, X \mid c, z)$. For every combination of $\{A, E\}$ we have a mixture of Gaussians corresponding to the possible values of D. Since $A \notin \mathbf{\Delta}_{DN}$, we get different Gaussians only for different values of E, i.e., all the combinations of $\{A, E\}$ that share the same value for E have the same Gaussians in their mixtures. Nonetheless, the weights of the mixture components are different for different values of A. When we add $\mathbf{\Delta}_{DN}$ to the query we need to approximate the

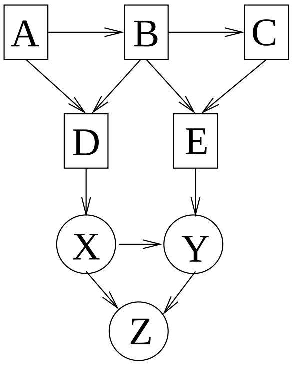


Figure 5.2: A CLG example

distribution $P(A, D, E, X \mid c, z)$.

Finally, in order to find $P(\boldsymbol{Q}_\Delta, \boldsymbol{\Delta}_{DN}, \boldsymbol{Q}_\Gamma \mid \boldsymbol{e}_\Delta, \boldsymbol{e}_\Gamma)$ it is convenient to first approximate $P(\boldsymbol{Q}_\Delta, \boldsymbol{\Delta}_{DN}, \boldsymbol{Q}_\Gamma, \boldsymbol{E}_\Gamma \mid \boldsymbol{e}_\Delta)$. Thus, in our example we will try to approximate $P(A, D, E, X, Z \mid c)$. This distribution is a mixture of Gaussians where the variables of each Gaussian are $\boldsymbol{Q}_\Gamma \cup \boldsymbol{E}_\Gamma$. Conditioning on $\boldsymbol{E}_\Gamma = \boldsymbol{e}_\Gamma$ can be done one mixture component at a time, in a way similar to conditioning RB-samples on evidence (see Section 3.4.3). We multiply the weight of the mixture component by the likelihood of the evidence and then condition the Gaussian on the evidence. If we re-normalize the mixture weights we get $P(\boldsymbol{Q}_\Delta, \boldsymbol{\Delta}_{DN}, \boldsymbol{Q}_\Gamma \mid \boldsymbol{e}_\Delta, \boldsymbol{e}_\Gamma)$.

We are now ready to discuss inference algorithms to evaluate $P(\boldsymbol{Q}_\Delta, \boldsymbol{\Delta}_{DN}, \boldsymbol{Q}_\Gamma, \boldsymbol{E}_\Gamma \mid \boldsymbol{e}_\Delta)$. In fact, we have already encountered this problem, and discussed it in Chapter 3. Obviously, if Lauritzen's algorithm is feasible for our particular problem then we can simply use it. However, in this chapter our goal is to tackle problems where Lauritzen's algorithm leads to clique trees which are too large, and we have to resort to approximate inference techniques.

We shall present three possible algorithms. The first two are based on RB-sampling, and were already discussed in Section 3.4.3. We then discuss our new algorithm, which is an alternative to the sampling approach. All the algorithms are based on the observation that, given an assignment to the discrete variables, the distribution over the continuous variables is a Gaussian. Thus, all the algorithms treat the discrete and continuous variables in an asymmetric way: We keep a set of weighted *particles* where every particle is defined by an assignment to the discrete variables and a probability distribution over the continuous ones.[1]

5.1.2 Rao-Blackwellized Likelihood Weighting

Our first algorithm to approximate $P(\boldsymbol{Q}_\Delta, \Delta_{DN}, \boldsymbol{Q}_\Gamma, \boldsymbol{E}_\Gamma \mid \boldsymbol{e}_\Delta)$ is Rao-Blackwellized Likelihood Weighting. Recall that the variables of the CLG are partitioned into two sets $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$, where we sample $\boldsymbol{X}_1$ and then keep a distribution $P(\boldsymbol{X}_2 \mid \boldsymbol{x}_1)$. Here we choose $\boldsymbol{X}_1 = \boldsymbol{\Delta}$ and $\boldsymbol{X}_2 = \Gamma$, i.e., we sample the discrete variables and given a sample $\boldsymbol{x}_1$ of $\boldsymbol{\Delta}$ we keep a multivariate Gaussian over $\boldsymbol{X}_2$. When sampling from $\boldsymbol{\Delta}$ we use LW with the evidence $\boldsymbol{e}_\Delta$. Alternatively, if the network structure of the discrete variables is simple we can perform probabilistic inference and sample directly from $P(\boldsymbol{\Delta} \mid \boldsymbol{e}_\Delta)$. Note that, after generating a sample $\boldsymbol{\Delta} = \boldsymbol{\delta}$, we can remove from $\boldsymbol{\delta}$ every discrete variable not in $\boldsymbol{Q}_\Delta \cup \boldsymbol{\Delta}_{DN}$ and similarly we can remove from the Gaussian every variable not in $\boldsymbol{Q}_\Gamma \cup \boldsymbol{E}_\Gamma$.

5.1.3 Rao-Blackwellized MCMC

Our second algorithm for approximate inference in CLGs is Rao-Blackwellized MCMC, and in particular Rao-Blackwellized Gibbs sampling. Recall that in Gibbs sampling we need to find the conditional distribution of some discrete variable A_i given the rest of the variables in $\boldsymbol{\Delta}$ and the continuous evidence. Let $\boldsymbol{\delta}_{-i}$ be an assignment to the discrete variables $A_1, \ldots, A_{i-1}, A_{i+1}, A_n$. We can then write:

$$P(A_i = a \mid \boldsymbol{\delta}_{-i}, \boldsymbol{e}_\Gamma) \propto P(\boldsymbol{e}_\Gamma \mid A_i = a, \boldsymbol{\delta}_{-i}) P(A_i = a \mid \boldsymbol{\delta}_{-i})$$

It is easy to compute $P(A_i = a_i \mid \boldsymbol{\delta}_{-i})$, just like in Gibbs sampling for discrete networks. To find $P(\boldsymbol{e}_\Gamma \mid A_i = a, \boldsymbol{\delta}_{-i})$ we build the Gaussian that corresponds to the discrete instantiation and then find the likelihood of the evidence. Having computed $P(\boldsymbol{e}_\Gamma \mid A_i = a, \boldsymbol{\delta}_{-i}) P(A_i = a \mid \boldsymbol{\delta}_{-i})$ for all the possible values of A_i we normalize the probabilities, which leads to a probability distribution over the next state.

[1]In the context of RB-sampling methods, the particles are identical to the RB-samples.

5.1.4 The Enumeration Algorithm

Recall the intuition of first considering the hypotheses which were likely *a priori*. RB-LW is one way to apply this idea in the context of sampling. Our third and final algorithm applies this intuition in the context of deterministic search rather than sampling. With this method the algorithm uses all its running time to consider as many distinct hypotheses as possible without the duplicate hypotheses that come up when using sampling method.

As a first step, assume we wish to enumerate the most likely instantiations of $P(\mathbf{\Delta} \mid \boldsymbol{e}_\Delta)$. An important observation is that we can ignore all the continuous variables in the BN for the purposes of the enumeration: We only care about the discrete problem of enumerating from $P(\Delta \mid \boldsymbol{e}_\Delta)$. We can therefore use the algorithm presented in Section 2.5.1 to enumerate the K most likely assignments of $P(\Delta \mid \boldsymbol{e}_\Delta)$. For each such instantiation we compute the relevant multivariate Gaussian and set its weight to be $P(\boldsymbol{\delta} \mid \boldsymbol{e}_\Delta)P(\boldsymbol{e}_\Gamma \mid \boldsymbol{\delta})$.[2] Note that, although we create a clique tree to generate our samples, the tree has only discrete variables and thus there is no need for strong triangulation as with Lauritzen's algorithm. Thus, if the structure of the discrete part of the Bayesian network is simple (as in fault diagnosis, where many fault variables are independent and merely influence some continuous variables) then the resulting clique tree will be simple, and the enumeration can be done efficiently.

It is sometimes possible to do better than this. Recall that we are not interested in enumerating the instantiations of $\mathbf{\Delta}$ per se. What we are interested in are the instantiations of $\boldsymbol{Q}_\Delta \cup \mathbf{\Delta}_{DN}$. Thus, it would be useful the marginalize out of our clique tree all the variables in $\Delta - (\boldsymbol{Q}_\Delta \cup \mathbf{\Delta}_{DN})$. Unfortunately, the problem of generating the K most likely hypotheses for a subset of the variables can be significantly more difficult than generating the K most likely hypotheses for all the variables [Par02]. However, in some cases, if the network structure is truly simple the clique tree over $\boldsymbol{Q}_\Delta \cup \mathbf{\Delta}_{DN}$ is still relatively small and then it is possible to generate efficiently the K most likely samples for $\boldsymbol{Q}_\Delta \cup \mathbf{\Delta}_{DN}$.

In fact, it is even enough to enumerate just the instantiations of $\mathbf{\Delta}_{DN}$ without

[2] These weights need to be re-normalized in order to serve as a probability distribution.

$\boldsymbol{Q}_\Delta$, since for every such instantiation we can use discrete Bayes net inference to find $P(\boldsymbol{Q}_\Delta \mid \boldsymbol{\Delta}_{DN})$. We can view this as another instance of Rao-Blackwellization where we add variables $\boldsymbol{Q}_\Delta - \boldsymbol{\Delta}_{DN}$ to the set of unsampled variables. In other words, for each assignment to $\boldsymbol{\Delta}_{DN}$ we keep a probability distribution over the continuous variables as well as over $\boldsymbol{Q}_\Delta$. This idea can lead to significant savings sometimes, but again we are faced with the potential of larger clique trees, so using the idea is not always possible. In any case, in all the examples considered in this thesis, we always have $\boldsymbol{Q}_\Delta \subseteq \boldsymbol{\Delta}_{DN}$, and therefore $\boldsymbol{Q}_\Delta - \boldsymbol{\Delta}_{DN} = \emptyset$.

5.1.5 The Single Representative Algorithms

The RB-LW and RB-MCMC algorithms can be viewed as having two sources of randomness: which particles are generated and how many times each given particle is sampled. Recall that it is possible to generate duplicate samples of the same particle and that the effect of the duplicate samples is to increase the weight of the particle. In other words, the weight of a particle corresponding to the assignment $\boldsymbol{x}$ given the evidence $\boldsymbol{e}$ is

$$\frac{\#\text{ of times } \boldsymbol{x} \text{ was sampled}}{\#\text{ of total samples}} P(\boldsymbol{e} \mid \boldsymbol{x}).$$

We can view the first term as an estimator for the prior probability $P(\boldsymbol{x})$. Under this view, our enumeration algorithm modifies the sampling algorithms in two ways

- Generating the particles in a deterministic way rather than by sampling.
- Avoid generating duplicate samples. The term $\frac{\#\text{ of times } \boldsymbol{x} \text{ was sampled}}{\#\text{ of particles generated}}$ is replaced by $P(\boldsymbol{x})$.

This view naturally leads to an intermediate algorithm, which we call the *single representative* algorithm. The single representative algorithm keeps the first source of stochasticity but avoids the second: It generates the particles stochastically as in the sampling approach, but ignores the duplicate particles and uses $P(\boldsymbol{x})$ instead of $\frac{\#\text{ of times } \boldsymbol{x} \text{ was sampled}}{\#\text{ of particles generated}}$. In fact, we define two different *single representative* algorithms:

- **Single Representative RB-LW** — in this version the particles are generated using LW sampling as in Section 5.1.2.
- **Single Representative RB-LW** — in this version the particles are generated using MCMC sampling as in Section 5.1.3.

To summarize, the single representative algorithms generate the particles using either LW or MCMC, but unlike the sampling algorithms, duplicate samples are ignored — the weight of each particle is set to be $p(\boldsymbol{x})P(\boldsymbol{e} \mid \boldsymbol{x})$. Note that computing $P(\boldsymbol{x})$ is easy under the assumption that all the parents of a sampled variables are also sampled — in this case computing $P(\boldsymbol{x})$ reduces to multiplying the relevant CPD entries. This is the case in all our examples since we sample all the discrete variables and there are no discrete variables with continuous parents.

Unlike RB-LW and RB-MCMC, The single representative algorithms are biased. However, the hope is that the added bias is compensated by the reduction in the variance, which stems from the fact that the randomness of the sampling process does not influence the (unnormalized) weight of a particle once the particle is generated. The single representative RB-LW is an intermediate algorithm between RB-LW and our enumeration algorithm, and therefore examining its performance can help us to understand better the difference in performance between the two algorithms.

5.1.6 An Optimization Trick

All the algorithms that we described share a common procedure: Given an assignment to $\boldsymbol{\Delta}_{DN}$ they need to generate a multivariate Gaussian over the continuous variables. Although it is certainly possible to generate each Gaussian independently, it might be possible to speed up significantly the process by using previously generated Gaussians.

As an example, consider a large CLG where we have some discrete variable A that influences some continuous variable X, where X is a leaf. Assume we already created a multivariate Gaussian P_1 for some assignment of $\boldsymbol{\Delta}_{DN} = \boldsymbol{\delta}_1$ and we would like to construct a new Gaussian P_2 corresponding to some assignment $\boldsymbol{\delta}_2$, where $\boldsymbol{\delta}_1$ and $\boldsymbol{\delta}_2$ represent the same assignment except for the variable A. Since no continuous variables other than X are influenced by A, the Gaussians P_1 and P_2 would be exactly

Optimization of Gaussian generation

  $\{\langle P_1, \boldsymbol{\delta}_1 \rangle, \ldots, \langle P_N, \boldsymbol{\delta}_N \rangle\}$: Previously generated Gaussians

  $\boldsymbol{\delta}$: A new assignment of Δ_{DN}

1. Let best $= 1$
2. Let $\boldsymbol{X}_{\texttt{best}} = \boldsymbol{\Gamma}$ /* Every continuous variable */
3. For $i = 1$ to N
    4. Let $\boldsymbol{A}$ be variables with different assignments in $\boldsymbol{\delta}_i$ and $\boldsymbol{\delta}$
    5. Let $\boldsymbol{X}$ be the continuous descendants of $\boldsymbol{A}$
    6. If $(|\boldsymbol{X}| < |\boldsymbol{X}_{\texttt{best}}|)$
        7. Let best $= i$
        8. Let $\boldsymbol{X}_{\texttt{best}} = \boldsymbol{X}$
    9. End-if
10. End-for
11. Let $P = \texttt{Copy}(P_{\texttt{best}})$
12. Recompute moments for $\boldsymbol{X}_{\texttt{best}}$ in P using Theorem 3.3

Figure 5.3: Speeding up Gaussian generation using previously created Gaussians

the same except for the moments that involve X, i.e., the mean and variance of X and the covariances between X and other variables. Thus, instead of creating P_2 from scratch, it seems reasonable to start with P_1 and change it only for X using the equations from Theorem 3.3. Using this idea we generate only $O(n)$ entries in the covariance matrix instead of $O(n^2)$ entries (where n is the number of continuous variables).

In general assume that we generated a Gaussian P_1 for the discrete assignment $\boldsymbol{\Delta}_{DN} = \boldsymbol{\delta}_1$ and we would like to use it in order to generate a Gaussian P_2 for the discrete assignment $\boldsymbol{\Delta}_{DN} = \boldsymbol{\delta}_2$. Let $\boldsymbol{A}$ be the set of variables which have different values in $\boldsymbol{\delta}_1$ and $\boldsymbol{\delta}_2$. If we start with P_1 we only have to recompute the means, variances and covariances for continuous variables which are descendants of some variable in $\boldsymbol{A}$. Thus, if we have enough space to keep at least some of the generated Gaussians in memory we can use them to speed up our algorithm, using the algorithm shown in Figure 5.3. Among the N previously generated Gaussians we find the Gaussian P_{best} that is "best" for our needs, in the sense that our Gaussian and P_{best} have the largest number of continuous variables that share the same moments. We

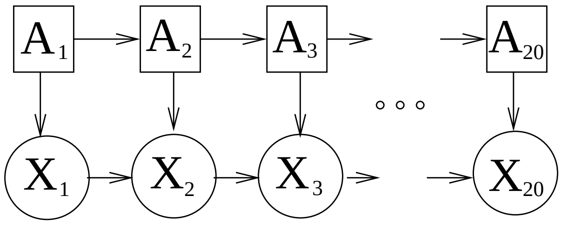


Figure 5.4: Network used for the experiments in this chapter

then copy the Gaussian in order not to change the previously generated Gaussian, and make the necessary changes in the copy.

One might wonder how useful this idea is. The answer is that the computational gain depends on the domain and on the way we generate Gaussians, but it can be very significant. Consider for example the fault diagnosis domain and assume we are using the enumeration algorithm to generate the Gaussians systematically. After generating the most likely hypothesis, corresponding to zero faults, we would start generating single fault hypotheses. Each one of these differs from the no-fault hypothesis by just one discrete variable. In many cases this single variable represents a sensor fault, in which case it typically influences a single leaf continuous variable (representing the sensor reading). In this case $|\boldsymbol{A}| = 1$ and better yet, $|\boldsymbol{X}| = 1$ and we get a dramatic speedup. A similar situation arises when we generate hypotheses corresponding to two faults, since each one of the two-faults hypotheses differs from some single fault hypothesis by just one variable. When we used this idea on a large real-world network in Chapter 10 we achieved a speedup factor of 16, i.e., we were able to consider 16 times as many hypotheses using the same computation time.

5.2 Empirical Comparison

In the last section we described five algorithms: RB-LW, single representative RB-LW, RB-MCMC, single representative RB-MCMC, and our enumeration algorithm. To compare these algorithms, we generated random Bayesian networks with a structure shown in Figure 5.4. This structure resembles the one used for the NP-hardness reduction, but the discrete variables are not independent *a priori*. The domain of all the discrete variables is $\{0, 1\}$; the evidence in all the experiments was X_{20}.

Note the dramatic difference between the clique tree used by the enumeration algorithm and the strongly rooted tree used by Lauritzen's algorithm. Since all the continuous variables are in one continuous connected component, any strongly rooted tree must have a clique that contains all the discrete variables, and therefore the size of the tree is exponential in n (where n is the size of the network, and in our case $n = 20$). In contrast, when we ignore the continuous variables, the discrete variables form a simple chain, and all the cliques in the tree contain only two discrete variables. Thus, the size of the tree used for the enumeration algorithm is only $O(n)$.

We chose to run experiments on this particular network because we were able to use a special ad hoc algorithm to compute the distribution over the discrete variables. Recall that

$$P(\boldsymbol{A} \mid x_{20}) \propto P(x_{20} \mid \boldsymbol{A})P(\boldsymbol{A})$$

where $\boldsymbol{A} = \{A_1, \ldots, A_{20}\}$. To evaluate this expression we enumerate all the instantiations $\boldsymbol{a}$ of $\boldsymbol{A}$. Computing $P(\boldsymbol{a})$ is easy since all we have to do is to multiply the relevant CPD entries. Since $P(X_{20} \mid \boldsymbol{a})$ is a Gaussian, in order to compute $P(x_{20} \mid \boldsymbol{a})$ it is enough to compute the mean and variance of X_{20}. This can be done efficiently: Given an assignment to $\boldsymbol{A}$, we first compute the mean and variance of X_1, from it we compute the mean and variance of X_2, from it the mean and variance of X_3, and so on. Note that because of the simple network structure, there is no need to compute the full covariance matrix.

Thus, for every assignment $\boldsymbol{a}$ we can compute the (unnormalized) probability $P(\boldsymbol{a} \mid x_{20})$ efficiently. Once this is computed for every $\boldsymbol{a}$ we normalize the result to get the exact posterior distribution $P(\boldsymbol{A} \mid x_{20})$. Therefore, although $\boldsymbol{A}$ has 2^{20} possible instantiations, using this method we were still able to compute the exact posterior distribution in a matter of seconds. This enabled us to compare the various approximate inference algorithms to the exact posterior distribution.

The parameters of the network were set as follows. We set $P(A_1 = 1) = p$ and for $2 \leq i \leq 20$ we set $P(A_i = A_{i-1}) = p$. The value of p determines how skewed the distribution is: There is one assignment with probability p^{20}, 20 assignments with probability $p^{19}(1-p)$, $\binom{20}{2}$ assignments with probability $p^{18}(1-p)^2$, and so on. Thus, we are in the same situation illustrated in Figure 5.1.

The other parameters in the network are in the CPDs of the continuous variables X_i's. Recall that the linear CPD of X_i is defined as $P(X_i \mid A_i = k, X_{i-1}) = \mathcal{N}\left(X_i; \alpha_{i,k} + \beta_{i,k}X_{i-1}, \sigma^2_{i,k}\right)$ where $k \in \{0,1\}$ (for $i = 1$ the parameters $\beta_{1,k}$ do not exist, as X_1 does not have a continuous parent). We sampled values for $\alpha_{i,k}$ using a uniform distribution over $[0, M]$ (where we varied M between $100,000$ and $1,000,000$) and values for $\beta_{i,k}$ using a uniform distribution over $[0, 2]$. For all i and k we set $\sigma_{i,k} = 1$.

We ran our experiments for various combinations of M and p. For each such combination we generated 200 sets of parameters for the Bayesian network. We sampled an instantiation from each one of the Bayes nets and used the sampled value for X_{20} as its observed value. We then computed the true posterior distribution over the discrete variables. We compared our algorithms using two criteria:

- L1-error over the discrete variables between the correct distribution P and the distribution P' generated by the approximation algorithm:

$$\sum_{\boldsymbol{a}} |P(\boldsymbol{A} = \boldsymbol{a}) - P'(\boldsymbol{A} = \boldsymbol{a})|$$

- Whether the most likely assignment according to the approximation algorithm was one of the three most likely assignments in the correct posterior distribution.

It is hard to come up with a fair comparison between the algorithms in terms of the number of particles, because in general duplicate samples are much cheaper than generating the particle for the first time — once we generate a duplicate sample there is no need to create the Gaussian again, but just increment the counter of the particle. Thus, if we compare the algorithms relative to all the samples including duplicates, we give the enumeration algorithm an unfair advantage since all its particles are distinct Gaussians and are more expensive to compute than duplicate particles. On the other hand, if we count only distinct Gaussians and ignore the duplicates we give the sampling algorithms an unfair advantage, because we ignore the time used to

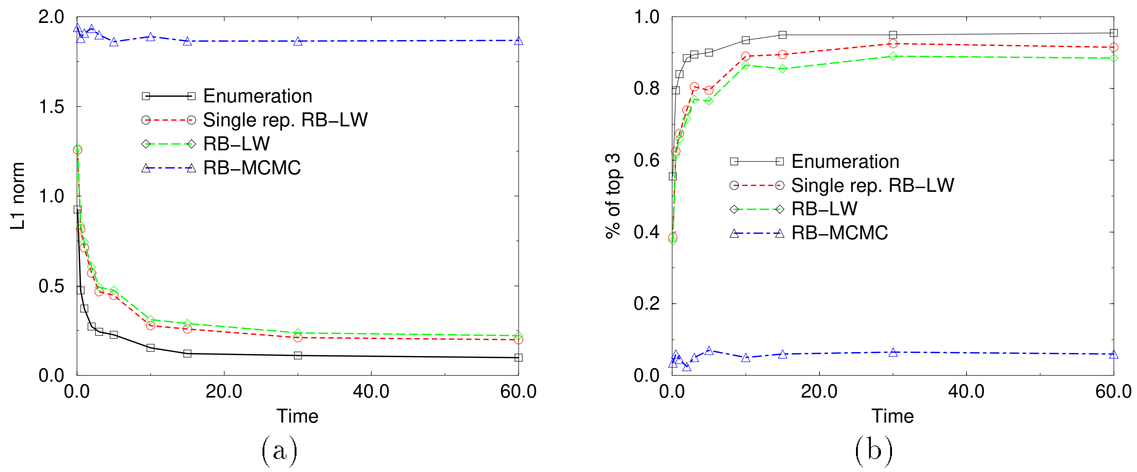


Figure 5.5: Results of the various algorithms on random networks with $p = 0.9$ and $M = 100000$ (a) L1 norm (b) Percentage of runs where the most likely hypothesis generated by the algorithm is one of the three most likely assignments

generate duplicate particles. Therefore, in order to make a fair comparison between the algorithms we measured their performance relative to the running time.

The results for different M's and p's were quite similar. Figure 5.5 shows the results for $M = 100,000$ and $p = 0.9$. Figure 5.5(a) shows the L1-error which is always between 0 and 2 (0 being good) and Figure 5.5(b) shows the top-3 criterion, i.e., the percentage of runs where the most likely assignment according to the algorithm was one of the top 3 most likely assignments in the correct posterior distribution (1 being good). The line for single representative RB-MCMC was almost on top of the line for RB-MCMC and is therefore not shown in the graphs. The reason for this phenomenon will be discussed later in the context of RB-LW, where we shall see that when the performance of the sampling algorithm is poor, it is very similar to the performance of the single representative algorithm.

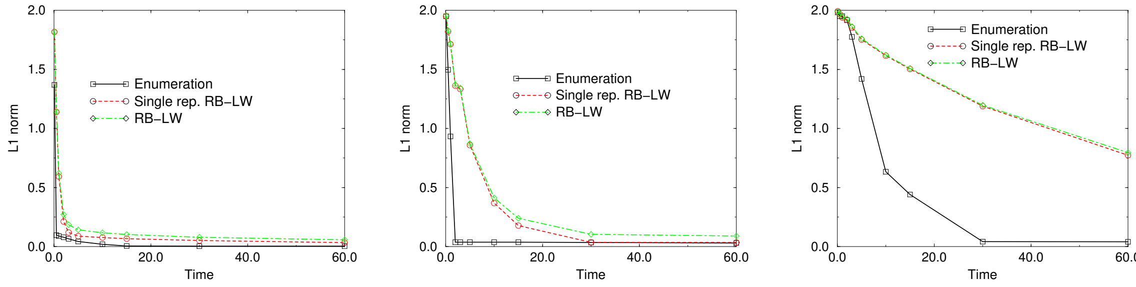
The most striking observation in Figure 5.5 is the poor performance of both RB-MCMC and single representative RB-MCMC, which indeed was the case in all the experiments we ran. The reason is that the space of discrete assignments has many local maxima which are hard to get out of. For example, assume we have just 2 nodes where

$$\begin{aligned}
P(X_1 \mid A_1 = 0) &= \mathcal{N}(X_1; 0, 1) \\
P(X_1 \mid A_1 = 1) &= \mathcal{N}(X_1; 10, 1) \\
P(X_2 \mid A_2 = 0, X_1) &= \mathcal{N}(X_2; X_1, 1) \\
P(X_2 \mid A_2 = 1, X_1) &= \mathcal{N}(X_2; X_1 + 15, 1)
\end{aligned}$$

Further, assume that the evidence is $X_2 = 15$ and that our initial assignment for MCMC is $A_1 = 1, A_2 = 0$. The most likely assignment is $A_1 = 0, A_2 = 1$, but it may take the chain a very long time before generating this sample. The reason is that when using Gibbs sampling we change one variable at a time, and therefore we can only consider transitions to $A_1 = 0, A_2 = 0$ or $A_1 = 1, A_2 = 1$. Both of these assignments are much less likely than our initial assignment, and therefore the probability of transitioning into either one of them is extremely small.[3] Thus, the mixing rate of the chain is very small and the performance of MCMC will be poor. The problem is much worse when we have 20 nodes instead of 2, since there are many more local maxima, i.e., assignments from which taking any single transition significantly reduces the probability of the evidence. The algorithm is likely to get stuck in one of the local maxima and not get to the likely assignments in any reasonable time. Since the performance of the MCMC algorithms was consistently poor, we ignore it for the rest of this section and concentrate on the enumeration algorithm, RB-LW and single representative RB-LW.

Figure 5.5 shows that among the remaining three algorithms, the enumeration algorithm had the best performance, RB-LW had the worst performance and single representative RB-LW was in between. To further explore the differences between the algorithms, we tested the performance of the algorithms relative to the likelihood

[3]For our initial assignment the mean of X_2 is 10, for the assignment $A_1 = 0, A_2 = 0$ the mean of X_2 is 0 and for the assignment $A_1 = 1, A_2 = 1$ the mean of X_2 is 25. Thus, for the initial assignment the evidence $X_2 = 15$ is 5 standard deviations away from the mean, for the assignment $A_1 = 0, A_2 = 0$ it is 15 standard deviations away from the mean, and for the assignment $A_1 = 1, A_2 = 1$ it is 10 standard deviations away from the mean. Therefore, although our initial assignment is quite unlikely, it is still much more likely than the other two candidates.



(a) (b) (c)

Figure 5.6: L1 norm as a function of the prior likelihood of the discrete assignment (a) Prior of 0.0015 (b) Prior of 0.000167 (c) Prior of 0.0000185

of the discrete events. To do so we used random networks with $p = 0.9$ and $M = 100,000$ and sampled assignments to both the discrete and continuous variables. We then partitioned our samples into subsets based on the likelihood of the discrete assignments, and used 35 such samples in every subset. We tested the performance of the algorithms for each subset separately. For RB-LW and single representative RB-LW we performed 10 runs for each one of the assignments. Note that we have one assignment ($\{A_1 = 1, \ldots, A_{20} = 1\}$) with probability $0.9^{20} \approx 0.122$, 20 assignments with probability $0.9^{19} \cdot 0.1 \approx 0.0135$, $\binom{20}{2}$ assignments with probability $0.9^{18} \cdot 0.1^2 \approx 0.0015$, $\binom{20}{3}$ with probability $0.9^{17} \cdot 0.1^3 \approx 0.000167$, and so on. Further note that, although the first subset has just one discrete assignment, it is possible to generate more than one sample, since the sample also includes the assignment to the continuous variables. Figure 5.6 and Figure 5.7 show the results of these experiments.

As expected, all three algorithms perform better when the likelihood of the discrete event is higher. The reason is that discrete events with high likelihood are enumerated early on and are more likely to be sampled early in the sampling process. However, note that the degradation in the performance of RB-LW and single representative RB-LW is much more severe than the degradation in the performance of the enumeration algorithm. The reason is that when the discrete event is unlikely *a priori*, the sampling approaches often spend a lot of time in generating duplicate samples of the likely assignments. On the other hand the enumeration algorithm does not waste any extra time on the likely assignments after they have been generated and can efficiently

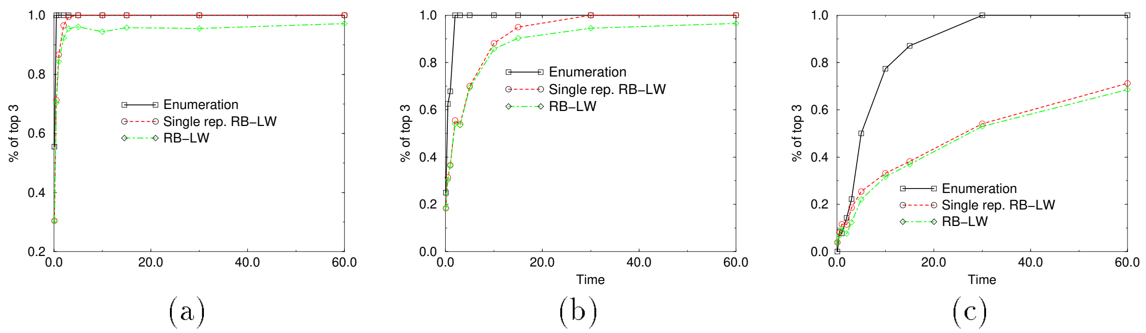
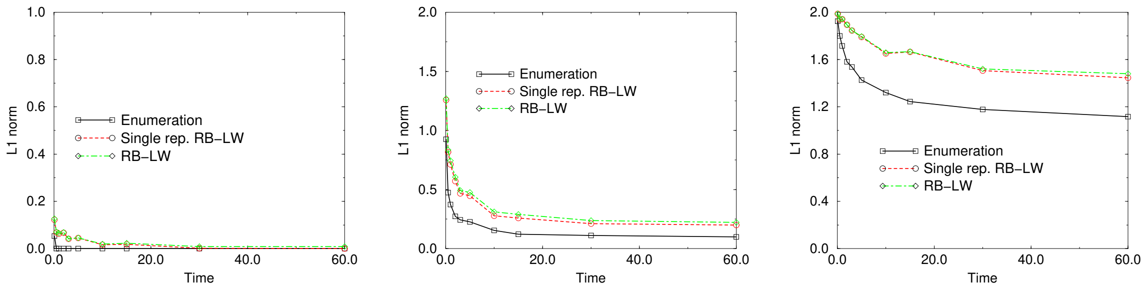


Figure 5.7: Top 3 percentage as a function of the prior likelihood of the discrete assignment (a) Prior of 0.0015 (b) Prior of 0.000167 (c) Prior of 0.0000185

spend its time on generating more and more unlikely assignments.

Figure 5.7 demonstrates another interesting phenomenon. When the performance of the algorithms is poor, the performance of RB-LW and single representative RB-LW is very similar, as can be seen in Figure 5.7(c) and the left side of Figure 5.7(b). In these cases, both algorithms are significantly outperformed by the enumeration algorithm. However, when the results of the sampling algorithms become more reliable, single representative RB-LW works much better than RB-LW until its performance is similar to the performance of the enumeration algorithm, as can be seen in Figure 5.7(a) and the right side of Figure 5.7(b). We conjecture that the reason for this phenomenon is that when the performance is poor, it mostly stems from not sampling the correct discrete instantiations. Since both RB-LW and single representative RB-LW generate the samples in an identical way, when we do not have the correct assignments in one we are likely not to have the correct assignments in the other. However, when the performance is better, the correct samples are generated and then the question is whether they have the correct weight or not. By eliminating the sampling noise for the sample weights, single representative RB-LW effectively behaves like the enumeration algorithm in this case, and thus significantly outperforms the RB-LW algorithm.

In our final set of experiments we tested the performance of the algorithms relative to how skewed the distribution is. Recall that when the parameter p is closer to 1, the distribution is more skewed and a smaller number of hypotheses account for more



(a) (b) (c)

Figure 5.8: L1 norm as a function of the parameter p (a) $p = 0.99$ (b) $p = 0.9$ (c) $p = 0.75$

probability mass. As shown in Figure 5.8 and Figure 5.9, the closer p is to 1, the better the three algorithms perform, as expected. Furthermore we see that no matter what the value of p is, the enumeration algorithm always has the best performance and that single representative RB-LW outperforms RB-LW.

In conclusion, our results seem to indicate that avoiding the stochasticity of RB-LW is a useful idea in skewed distributions. The enumeration algorithm, which is fully deterministic, works best. The single representative RB-LW, which is stochastic in its choice of particles but deterministic in determining their weight, enjoys some of the performance improvement, but not all of it.

5.3 Discussion

In this section we discuss three issues relating to our enumeration algorithm: its relation to RB-LW, bounding the error of the approximation and a comparison with Lauritzen's algorithm.

In our presentation so far, we viewed the enumeration algorithm as a very close relative to RB-LW, namely as a deterministic alternative to LW. Although this is certainly a reasonable view, it is not the only one, and it can be argued that RB-LW and the enumeration algorithm are fundamentally different. At a very basic level, Rao-Blackwellized LW, like any other sampling method, is motivated by the weak law of large numbers — when the number of samples grows it gets more and more

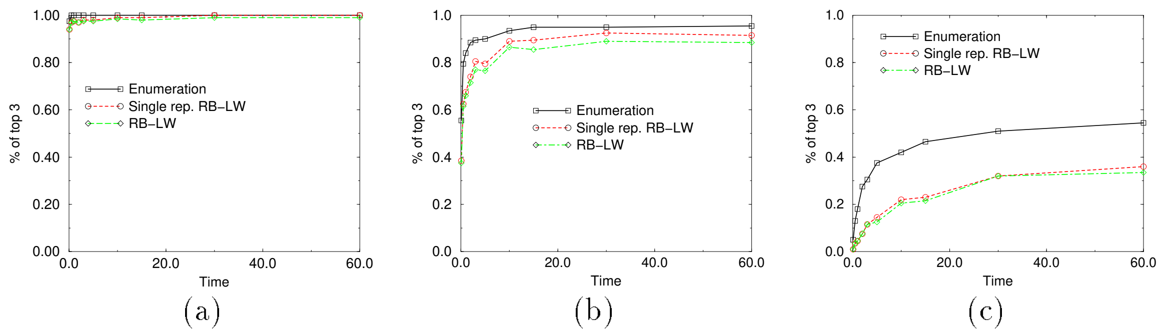


Figure 5.9: Top 3 percentage as a function of the parameter p (a) $p = 0.99$ (b) $p = 0.9$ (c) $p = 0.75$

accurate and at the limit it is exact. Our algorithm is a biased estimator that cannot be directly motivated by the law of large numbers, and thus, it can viewed as a very different alternative to sampling.

5.3.1 Error Bounds

An important issue is bounding the error of our approximation. One possible approach is to bound the probability mass of hypotheses which were not generated. We can partition the set of instantiations of $\mathbf{\Delta}_{DN}$ into $A \cup B$ where A represents the instantiations that were generated so far and B represents the instantiations that were not generated so far. The unnormalized probability mass of hypotheses which were not generated so far is:

$$\sum_{\boldsymbol{\delta} \in B} P(\boldsymbol{\delta} \mid \boldsymbol{e}_\Delta) P(\boldsymbol{e}_\Gamma \mid \boldsymbol{\delta})$$

To bound this sum, we need to bound the densities $P(\boldsymbol{E}_\Gamma \mid \boldsymbol{\delta})$ (which may be larger than 1) over the possible instantiations of $\mathbf{\Delta}_{DN}$ (or preferably, over the possible instantiations of $\mathbf{\Delta}_{DN}$ consistent with some assignment in B). If we can find such a bound K, then we can write:

$$\sum_{\boldsymbol{\delta} \in B} P(\boldsymbol{\delta} \mid \boldsymbol{e}_\Delta) P(\boldsymbol{e}_\Gamma \mid \boldsymbol{\delta}) \le \sum_{\boldsymbol{\delta} \in B} P(\boldsymbol{\delta} \mid \boldsymbol{e}_\Delta) K = \left(1 - \sum_{\boldsymbol{\delta} \in A} P(\boldsymbol{\delta} \mid \boldsymbol{e}_\Delta)\right) K$$

Given K, this bound can be computed efficiently based on our enumeration of instantiations in A. Note that the bound decreases monotonically when we generate instantiations, since by doing so we move them from B to A. The big question, of course, is how to compute a bound K. One possible approach is to bound the density of the Gaussian $P(\boldsymbol{E}_\Gamma \mid \boldsymbol{\delta})$. Assume that we can find the density at the mean of this Gaussian. No matter what the evidence is, its density cannot be larger than the density at the mean, and thus, we get an upper bound. Furthermore, we will take the approach of trying to bound K with respect to all the possible assignments of $\boldsymbol{\Delta}_{DN}$, and not just those consistent with B.

Under these assumptions, we can bound K efficiently if the network is a polytree and $|\boldsymbol{E}_\Gamma| = 1$. Consider a univariate Gaussian $\mathcal{N}(\mu, \sigma^2)$. The density of this Gaussian at the mean μ is $\frac{1}{\sqrt{2\pi}\sigma}$. Thus, in order to find an upper bound on the density $P(\boldsymbol{E}_\Gamma \mid \boldsymbol{\delta})$, we need to find a lower bound on the variance of $\boldsymbol{E}_\Gamma$ for every assignment $\boldsymbol{\delta}$. This can be done easily by traversing the tree in topological order and greedily finding the smallest variance of every continuous variable.

However, if we want to bound the density of $\boldsymbol{E}_\Gamma$ when the network is not a polytree or when $\boldsymbol{E}_\Gamma$ has more than one variable, we must consider the covariances between variables, and the problem becomes more difficult. Using a simple reduction from 3-SAT we can show that finding the smallest variance of a single variable (over all the possible instantiations of Δ_{DN}) in general networks is NP-hard. Currently, we do not know what is the complexity of finding the upper bound K in two cases:

- Polytrees where $|\boldsymbol{E}_\Gamma| > 1$
- Networks with cycles but with a low tree width. In other words, these are networks for which we can build a clique tree with a small number of variables in every clique when we do not perform strong triangulation.

The second case is quite interesting, and finding an efficient algorithm in this case would make the enumeration algorithm more reliable, as it would give an indication on the quality of the approximation.

5.3.2 Comparison with Lauritzen's algorithm

We conclude this chapter by comparing the enumeration algorithm to Lauritzen's exact inference algorithm. If we give our algorithm enough time it would enumerate all the possible discrete instantiations in every continuous connected component, and would come up with the exact answer. Thus, for simple networks, where it is possible to enumerate all the discrete instantiations of the various continuous connected components, the complexity of our algorithm is comparable to the complexity of Lauritzen's algorithm. Both algorithms enumerate the same number of hypotheses, either directly using the enumeration algorithm or by creating a Gaussian factor as in Lauritzen's algorithm. The two main differences are:

- Our algorithm creates Gaussians over all the continuous variables in the continuous connected component, as opposed to Lauritzen's algorithm where only some of the continuous variables are in the strong root. The disadvantage of our approach is that it creates larger Gaussians. However, by doing so our algorithm computes the exact distribution rather than weak marginals and avoids the costly operations involved in the message passing algorithm, such as matrix inversion.
- Our algorithm has the overhead of creating the clique tree for the discrete variables and enumerating the instantiations from it. However, in most cases this extra complexity is negligible compared to the operations involved in manipulating the Gaussians. Furthermore, if we know in advance that we have enough time to enumerate all the instantiations, there is no need to enumerate them by their prior likelihood, and we can completely avoid the overhead involved in enumeration by the prior.

Thus, when both algorithms have enough computational resources to come up with the exact answer their performance is comparable and can differ only by a polynomial factor. The choice between the algorithms will depend on the particular network structure, implementation details and whether we need weak or strong marginals.

However, our algorithm is not designed for small networks where Lauritzen's algorithm is tractable, but rather for large networks. In this case, our algorithm enjoys two important advantages:

- Our algorithm is an anytime algorithm — it can give some answer (albeit an approximate one) whenever we request it.
- Our algorithm has a much better space complexity. Recall that Lauritzen's algorithm often results in exponentially sized cliques even for simple networks such as the polytrees discussed in Chapter 4. In contrast, our storage requirements are dictated by the size of the query and are often exponentially smaller. For example, if our query involves only discrete variables then we can just keep a factor over these discretes and update it as we generate more Gaussians, without keeping the Gaussians that we generate. As another example, if we want to know the first two moments of some continuous variables $\boldsymbol{X}$ then we can keep a single Gaussian P over $\boldsymbol{X}$ — at every iteration we simply collapse P and the current Gaussian generated by the enumeration algorithm. However, if we want the true marginal over $\boldsymbol{X}$, we may have to keep a mixture of Gaussians over $\boldsymbol{X}$, and our space requirements will be significantly larger.

Thus, in many real life networks where Lauritzen's algorithm is impractical, we can use our enumeration algorithm and get at least an approximate answer. We shall do exactly that in order to perform fault diagnosis on a real life physical system in Chapter 10.

Chapter 6

Non-linear CPDs

So far, we have concentrated on CLG models, where all the continuous CPDs are linear CPDs. Unfortunately many real world domains have non-linear dependencies between their variables. The RWGS, discussed in Chapter 10, contains many examples of such non-linear dependencies. In this chapter, we concentrate on non-linear CPDs for continuous variables. In Chapter 7, we will also consider non-linear CPDs that involve continuous parents of discrete children.

In the presence of non-linear CPDs, the resulting joint distribution is no longer a mixture of Gaussians, but rather a mixture of distributions which in general are non-Gaussian and cannot be represented in closed form. For simplicity, we start our discussion with a purely continuous model, without any discrete variables. As we shall see, in the presence of discrete variables we can use the techniques from Chapter 5.

One very standard approach when the joint distribution is non-Gaussian is to approximate it as a Gaussian, and the task is to find a Gaussian distribution which is as close as possible to the original distribution. We first discuss the traditional approach, the *Extended Kalman Filter*, and then see how it can be improved by viewing the problem as a numerical integration problem.

```
Extended Kalman Filter for static BNs
  1. Sort nodes in topological order X_1, ..., X_n
  2. For i = 1..n do
    3. If CPD of X_i is not linear
      4. Approximate CPD as linear using Taylor series expansion
    5. Compute first two moments of P(X_1, ..., X_i) using Theorem 3.3
```

Figure 6.1: The extended Kalman filter adapted for static Bayesian networks

6.1 Extended Kalman Filter Approach

The *Extended Kalman Filter (EKF)* [BSLK01] is an algorithm that was developed in the context of dynamic models rather than static networks. However, the same approach can also be used for static Bayesian networks, and we shall present it in this context.

Let Y be some variable in the Bayes net with parents $\boldsymbol{X}$ and assume that $\boldsymbol{X}$ have a known Gaussian distribution $P(\boldsymbol{X}) = \mathcal{N}(\boldsymbol{X}; \boldsymbol{\mu}, \Sigma)$. We further assume that the CPD of Y is $Y = f(\boldsymbol{X})$ where f is a non-linear deterministic function. The assumption that f is deterministic does not restrict the generality of the method, since we can always add any source of stochasticity as extra variables to f. For example, if $Y = \sqrt{X_1^2 + X_2^2} + W$ where $P(W) = \mathcal{N}(W; 0, \sigma_W^2)$ then we can view f as a deterministic function with 3 variables $Y = f(X_1, X_2, W)$.

Our task is to find a Gaussian approximation for $P(\boldsymbol{X}, Y)$. The approach taken by EKF is to replace the function f by a simpler function $\hat{f}$, i.e., $Y = \hat{f}(\boldsymbol{X})$. The idea is that $\hat{f}$ is simple enough so that finding the Gaussian approximation to $P(\boldsymbol{X}, Y)$ becomes easy. The most popular choice for $\hat{f}$ is a linear function, with the appealing property that once we make a linear approximation, the distribution $P(\boldsymbol{X}, Y)$ is indeed a Gaussian and can be computed using Theorem 3.3. EKF uses what is perhaps the most standard linear approximation for $f(\boldsymbol{X})$ — the Taylor series expansion around the mean of $\boldsymbol{X}$:

$$Y \approx \hat{f}(\boldsymbol{X}) = f(\boldsymbol{\mu}) + \nabla f|_{\boldsymbol{\mu}} (\boldsymbol{X} - \boldsymbol{\mu}) \tag{6.1}$$

The Taylor series approximation is the basis for the EKF algorithm shown in Figure 6.1. We sort the nodes in topological order $X_1, \ldots, X_n$ and increasingly build a multivariate Gaussian over the nodes in this order. Note that, when we get to the node X_i, we already have a Gaussian approximation for all its parents and therefore we can use the Taylor series expansion as defined in Equation 6.1.

The only small addition that we need to make is to compute the covariances between the node X_i and other nodes that appear before it in the topological order but are not directly its parents. Let Z be such a node. One simple way of finding $\mathrm{Cov}(Z, Y)$ is to to add Z as a parent of Y, i.e., represent Y as $Y = \beta_0 + \sum_i \beta_i X_i + \beta Z + W$ where $\beta_Z = 0$ and then use Theorem 3.3. The only problem with this approach is that when we get to the later variables we will end up with many parents, and the algorithm will become inefficient. We will discuss an alternative approach in Section 6.2.5.

It is important to emphasize that using the Taylor series approximation does not lead to an optimal first order approximation, in the sense that the resulting Gaussian approximation to the distribution $P(\boldsymbol{X}, Y)$ might be sub-optimal. For example, consider the function $Y = X^2$ with $P(X) = \mathcal{N}(X; 0, 1)$. The mean of Y is easy to find, since $E[Y] = E[X^2] = 1$. The variance of Y is also easy to compute in closed form, since $\mathrm{Var}(Y) = E[Y^2] - E[Y]^2 = E[X^4] - E[X^2]^2 = 3 - 1^2 = 2$. Thus, the best Gaussian approximation for $P(Y)$ is $\mathcal{N}(1, 2)$. The first order Taylor series approximation at the mean value $X = 0$ is $Y = 0$, leading to the Gaussian approximation for $P(Y)$ as $\mathcal{N}(0, 0)$, i.e., a delta function where all the mass is located at $Y = 0$. Obviously, this is an extremely poor approximation.

EKF has two serious disadvantages. The first is its accuracy (or lack thereof). The quality of the approximation depends on how well $\hat{f}$ approximates f in the local area around the mean of $\boldsymbol{X}$ (the size of this local area is determined by the variance of $\boldsymbol{X}$). The EKF approximation is a good approximation only if the second and higher order terms in this area are negligible. In many practical situations, this is not the case and using the EKF leads to a poor approximation (as we have just seen for the case of $Y = X^2$).

The second problem with EKF is the need to compute the gradient. Some non-linear functions may not be differentiable (e.g., the max function), preventing the use of EKF. Furthermore, even if f is differentiable, computing the derivatives may still be hard, depending on f and the way it is represented. The function f may not be given explicitly as an analytical function, but rather as a lookup table or as a function implemented in some programming language, and in general f might be a black box that simply generates a value for some given inputs. If this is the case, it seems that there is no simple way to compute the derivatives of the Taylor series expansion without resorting to numerical differentiation methods.

To improve the accuracy of EKF, it is possible to extend it by taking into account higher order terms in the Taylor series expansion, but the formulas involved tend to become much more complex. The first complication is that we must compute many more partial derivatives (for example, for the second order approximation we must compute the Hessian). The second complication is that even after computing $\hat{f}$ the resulting distribution is still not a Gaussian — we cannot use Theorem 3.3 and must derive a different, and more complicated, set of formulas. In practice, even the second order approximation is not commonly used and higher order approximations are almost never used.

6.2 Numerical Integration

6.2.1 Problem Formulation

The extended Kalman filter uses what may be considered an indirect approach of first simplifying the non-linear function and only then computing the resulting distribution. A more direct approach is to try to directly approximate the distribution resulting from the original non-linear function. Again we start with the case of $Y = f(\boldsymbol{X})$ where f is a non-linear deterministic function and the goal is to approximate $P(\boldsymbol{X}, Y)$ as a Gaussian. We note that all the quantities we are looking for can be expressed as integrals:

$$E[Y] = \int_{-\infty}^{\infty} f(\boldsymbol{x})P(\boldsymbol{x})d\boldsymbol{x} \quad (6.2)$$

$$E[Y^2] = \int_{-\infty}^{\infty} f^2(\boldsymbol{x})P(\boldsymbol{x})d\boldsymbol{x} \quad (6.3)$$

$$E[X_iY] = \int_{-\infty}^{\infty} x_i f(\boldsymbol{x})P(\boldsymbol{x})d\boldsymbol{x} \quad (6.4)$$

Note that we use the notation $\int_{-\infty}^{\infty} f(\boldsymbol{x})d\boldsymbol{x}$ to represent the integral of f over all of $\mathbb{R}^n$. If we could evaluate these integrals, we would have the correct first and second order moments of $P(\boldsymbol{X}, Y)$, and thus we would be able to construct the best Gaussian approximation to the resulting distribution. Dealing with non-linear continuous CPDs can therefore be reduced to an integration problem, and the question is how to solve the integrals.

We first note that it is sometimes possible to solve the integrals in closed form, leading to an efficient and optimal way of computing the best Gaussian approximation. In fact we already saw an example when we considered the function $Y = X^2$. Equation 6.2 reduces to computing $E[X^2]$, Equation 6.3 reduces to computing $E[X^4]$, and Equation 6.4 reduces to computing $E[X^3]$, all of which have closed form formulas.

Unfortunately, in most cases there are no closed form solutions and we must resort to numerical integration techniques. There is a vast literature on these techniques, many of which can potentially be used for our purposes. Our particular integrals always have the form of a product between a Gaussian and some other function, and thus are particularly suitable for the *Gaussian Quadrature* and *Exact Monomials* methods [DR84].

6.2.2 Gaussian Quadrature

We begin our discussion with one dimensional integrals. Gaussian quadrature approximates integrals of the form $\int_a^b W(x)f(x)dx$ where $W(x)$ is a known non-negative function (in our case a Gaussian). Note that the approach is general since we can always set $W(X) \equiv 1$. Based on the function W, we choose n points $x_1, \ldots, x_n$ and

n weights $w_1, \ldots, w_n$ and approximate the integral as:

$$\int_a^b W(x) f(x) dx \approx \sum_{j=1}^{n} w_j f(x_j) \tag{6.5}$$

The points and weights are chosen such that the integral is exact if f is a polynomial of degree $2n-1$ or less. Consider for example the case of $n = 2$, and assume that the integration rule is chosen such that it is exact for $f_0(x) = 1$, $f_1(x) = x$, $f_2(x) = x^2$ and $f_3(x) = x^3$. Since we can represent any polynomial of the form $f(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3$ as a linear combination $f = \sum_i \alpha_i f_i(x)$ we get:

$$\int_a^b W(x) f(x) dx = \int_a^b W(x) \sum_i \alpha_i f_i(x) dx = \sum_i \alpha_i \int_a^b W(x) f_i(x) dx$$

In addition, we have:

$$\sum_{j=1}^{n} w_j f(x_j) = \sum_{j=1}^{n} w_j \sum_i \alpha_i f_i(x_j) = \sum_i \alpha_i \sum_{j=1}^{n} w_j f_i(x_j)$$

Therefore, if our integration rule is exact for $f_0, \ldots, f_3$ then it is exact for general polynomials of degree 3 or less. In order for the rule to be exact for $f_0, \ldots, f_3$ it must be the case that for $i = 0, .., 3$ we have $\int_a^b W(x) f_i(x) dx = w_1 f_i(x_1) + w_2 f_i(x_2)$. Assuming $W(x) = \mathcal{N}(0, 1)$, $a = -\infty$, and $b = \infty$, we get the following set of four non-linear equations

$$\begin{aligned}
w_1 + w_2 &= \int_{-\infty}^{\infty} \mathcal{N}(x; 0, 1)\, dx = 1 \\
w_1 x_1 + w_2 x_2 &= \int_{-\infty}^{\infty} \mathcal{N}(x; 0, 1)\, x dx = 0 \\
w_1 x_1^2 + w_2 x_2^2 &= \int_{-\infty}^{\infty} \mathcal{N}(x; 0, 1)\, x^2 dx = 1 \\
w_1 x_1^3 + w_2 x_2^3 &= \int_{-\infty}^{\infty} \mathcal{N}(x; 0, 1)\, x^3 dx = 0
\end{aligned}$$

The solution for these equations (up to swapping x_1 and x_2) is $w_1 = w_2 = 0.5$,

$x_1 = -1$ and $x_2 = 1$. Thus, for $n = 2$ we get the following integration rule

$$\int_{-\infty}^{\infty} \mathcal{N}(x; 0, 1)\, f(x)dx \approx 0.5f(-1) + 0.5f(1)$$

In general we get a set of $2n$ non-linear equations, and the existence of a solution is not at all obvious. Fortunately, it is possible to construct a different set of equations based on a set of orthogonal polynomials with respect to W [Atk89], and come up with an efficient procedure to solve the equations and generate the integration points [PTVF88]. The resulting integration rules are very accurate and efficient.

We can already point out one of the major advantages of the numerical integration approach compared to the EKF approach. Recall that, for the extended Kalman filter we relied on a Taylor series expansion, and had to compute the relevant derivatives. There is no such need in this case — all we need is to evaluate the function f over a set of points. This property is particularly advantageous if f is not differentiable or is given implicitly as a C function or a lookup table, where it is easy to evaluate f on a given point but hard to compute its derivatives.

Suppose now that the integral is in $\mathbb{R}^d$ rather than $\mathbb{R}$ and that the weight function is $\mathcal{N}(\mathbf{0}, I)$ (we show how to relax this assumption and use any Gaussian as the weight function in Section 6.2.4). We can still use the Gaussian Quadrature method by using our points as a grid of n^d points, with the resulting rule still being accurate for polynomials of degree $2n - 1$ or less. This approach can work reasonably well if d is small, but obviously it is not practical for integrals in high dimensions. Indeed, the problem of numerical integration is much harder in high dimensions, but fortunately we can still do much better than using n^d points, by choosing a different set of points.

6.2.3 Exact Monomials and the Unscented Filter

To avoid the exponential number of points of the Gaussian Quadrature method for multi-dimensional integrals, we can try to choose the points directly in $\mathbb{R}^d$ instead of using a grid of points chosen in $\mathbb{R}$ [DR84]. Our goal would be to find rules which are exact for some m monomials, and in particular we will consider the sets of all monomials of degree p or less, i.e., monomials of the form $\prod_{i=1}^{d} x_i^{a_i}$ where the powers

a_i are nonnegative integers and $\sum_{i=1}^{d} a_i \leq p$. Such rules are said to have *precision* or *degree* p (formally, there should also be a monomial of degree $p+1$ for which the rule is not exact). Note that the integration rule that uses a combination of one-dimensional Gaussian Quadrature rules of precision p is exact for a much larger set of monomials, namely monomials of the form $\prod_{i=1}^{d} x_i^{a_i}$ where $a_i \leq p$. For example, if we combine one-dimensional Gaussian Quadrature rules of precision 2, we will get a rule which is exact for all monomials of precision 2, but is also exact for some higher degree monomials such as $x_1^2 x_2^2$ (however, it will not be exact for the degree 3 monomial x_1^3).

In general there are $\binom{d+p}{p}$ distinct monomials of precision p or less. When we use n points there are $n + nd$ parameters that we can choose (n weights and nd coordinates for the points). Therefore one can expect that in order to achieve precision p, one would have to use $n \geq \frac{1}{d+1}\binom{d+p}{p}$ points. However this is not the case. Since we get a set of non-linear equations there is no guarantee that a solution exists. On the other hand, it is often possible to find rules that use significantly fewer points than $\frac{1}{d+1}\binom{d+p}{p}$, called *hyperefficient rules*. For example, for $p = 3$ we have a rule that uses only $2d + 1$ points, i.e., $2d^2 + 3d + 1$ free parameters, and is exact for all $O(d^3)$ monomials of degree three or less.

McNamee and Stenger [MS67] present a suite of exact monomial rules for *fully symmetric* weight functions. We say that a function W is fully symmetric if $W(\boldsymbol{x}) = W(\boldsymbol{y})$ for every $\boldsymbol{y}$ that is obtained from $\boldsymbol{x}$ by permutations and/or changes in the sign of the coordinates of $\boldsymbol{x}$. For example, the weight function $\mathcal{N}(\mathbf{0}, I)$ is fully symmetric. McNamee and Stenger develop rules for precisions 3, 5, 7, and 9 and a general procedure to find rules for precision $p = 2k + 1$ with $O(\frac{(2d)^k}{k!})$ points. Here we show the rules for precisions 3 and 5 with the weight function $\mathcal{N}(\mathbf{0}, I)$ as examples. In the next section we will show how the points can be transformed to general Gaussians.

Definition 6.1 *A point* $\boldsymbol{u} = (u_1, \ldots, u_r, 0, \ldots, 0) \in \mathbb{R}^d$ *where* $0 < u_i \leq u_{i+1}$ *will be called a* generator *and denoted as* $[\pm \boldsymbol{u}]$ *or* $[\pm u_1, \ldots, \pm u_r]$. *It represents the set of points that can be obtained from* $\boldsymbol{u}$ *by permutations and changing the sign of some coordinates.*

For example, the generator $[0]$ contains the point $\mathbf{0} \in \mathbb{R}^d$, and the generator

$[\pm 1, \pm 1]$ in $\mathbb{R}^3$ represents the set of points

$$\{ \quad (1,1,0),(-1,-1,0),(1,-1,0),(-1,1,0),(1,0,1),(-1,0,-1), \\ (1,0,-1),(-1,0,1),(0,1,1),(0,-1,-1),(0,1,-1),(0,-1,1) \quad \}$$

We will use the shorthand notation $f[\pm u_1, \ldots, \pm u_r]$ for the sum $\sum_{x \in [\pm u_1, \ldots, \pm u_r]} f(x)$, i.e., the sum of f applied to the points represented by the generator $[\pm u_1, \ldots, \pm u_r]$. We shall consider integration rules that are based on some set of generators, i.e., rules of the form:

$$\sum_k w_k f[\boldsymbol{u}_k] \tag{6.6}$$

Consider some monomial $f = \prod_{i=1}^{d} x_i^{a_i}$ with at least one odd a_j. Every generator $[\pm \boldsymbol{u}_k]$ is symmetric, i.e, if $\boldsymbol{x} \in [\pm \boldsymbol{u}_k]$ then there exists a $\boldsymbol{y} \in [\pm \boldsymbol{u}_k]$ which is the same as $\boldsymbol{x}$ except that the sign of the ith coordinate is switched. We have $f(\boldsymbol{x}) = -f(\boldsymbol{y})$ and therefore, when summing all the points we get $f[\pm \boldsymbol{u}_k] = 0$, and the result of the integration rule is $\sum_k w_k f[\boldsymbol{u}_k] = 0$. Consider now the integral $\int_{-\infty}^{\infty} W(\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x}$. If the weight function $W(\boldsymbol{x})$ is fully symmetric, then in particular it is symmetric for x_j. We get:

$$\begin{aligned}
\int_{-\infty}^{\infty} W(\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x} &= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} W(\boldsymbol{x}) \prod_{i=1}^{d} x_i^{a_i} dx_1 \cdots dx_d \\
&= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \prod_{i \neq j} x_i^{a_i} \left(\int_{-\infty}^{\infty} W(\boldsymbol{x}) x_j^{a_j} dx_j \right) dx_1 \cdots dx_{j-1} dx_{j+1} \cdots dx_d \\
&= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \prod_{i \neq j} x_i^{a_i} \cdot 0 dx_1 \cdots dx_{j-1} dx_{j+1} \cdots dx_d \\
&= 0
\end{aligned}$$

It follows that if our weight function is fully symmetric and the integration rule is of the form of Equation 6.6, then it is exact for every monomial with at least one odd power. Thus, we only have to consider monomials in which all the powers are even.

Let us develop a precision 3 rule with $2d+1$ points.[1] Our rule will be developed

[1] It is also possible to derive a rule with just $2d$ points but our rule will be more numerically stable.

for the fully symmetric function $W = \mathcal{N}(\mathbf{0}, I)$, and will have the form:

$$\int_{-\infty}^{\infty} \mathcal{N}(\boldsymbol{x}; \mathbf{0}, I) f(\boldsymbol{x}) d\boldsymbol{x} \approx w_0 f[0] + w_1 f[\pm u] \tag{6.7}$$

The only monomials of degree 3 or less without any odd power are $f(\boldsymbol{x}) = 1$ and $f(\boldsymbol{x}) = x_i^2$. All the monomials $f(\boldsymbol{x}) = x_i^2$ lead to the same equation, and we end up with just two equations for $f(\boldsymbol{x}) = 1$ and $f(\boldsymbol{x}) = x_1^2$

$$\begin{aligned} w_0 + 2dw_1 &= \int_{-\infty}^{\infty} \mathcal{N}(\boldsymbol{x}; \mathbf{0}, I)\, d\boldsymbol{x} = 1 \\ 2w_1 u^2 &= \int_{-\infty}^{\infty} \mathcal{N}(\boldsymbol{x}; \mathbf{0}, I)\, x_1^2 d\boldsymbol{x} = 1 \end{aligned}$$

The parameter u is a free parameter; after choosing some $u \neq 0$ we get $w_0 = 1 - \frac{d}{u^2}$ and $w_1 = \frac{1}{2u^2}$. Although we get a precision 3 rule for every value of $u \neq 0$, different choices of u do lead to different approximations. Small values of u lead to more local approximations which are based on the behavior of f near the mean of the Gaussian and are less affected by the higher order terms of f. Julier and Uhlmann [JU97] suggest to choose $u = \sqrt{3}$, but in general different values of u can lead to better or worse approximations depending on the function f.

For precision 5, we will look for a rule of the form:

$$\int_{-\infty}^{\infty} \mathcal{N}(\boldsymbol{x}; \mathbf{0}, I) f(\boldsymbol{x}) d\boldsymbol{x} \approx w_0 f[0] + w_1 f[\pm u] + w_2 f[\pm u, \pm u] \tag{6.8}$$

Note that we get 1 point from the generator $[0]$, $2d$ points from the generator $[\pm u]$ and $2d(d-1)$ points from the $[\pm u, \pm u]$ generator, for a total of $2d^2 + 1$ points. This time we have to make sure the rule is exact for monomials of precision 5 or less without any odd powers. Due to the symmetry of the generators, it is enough to consider the monomials $f(\boldsymbol{x}) = 1$, $f(\boldsymbol{x}) = x_1^2$, $f(\boldsymbol{x}) = x_1^4$ and $f(\boldsymbol{x}) = x_1^2 x_2^2$. We get the following set of equations:

$$\begin{aligned} w_0 + 2dw_1 + 2d(d-1)w_2 &= \int_{-\infty}^{\infty} \mathcal{N}(\boldsymbol{x}; \mathbf{0}, I)\, d\boldsymbol{x} = 1 \\ 2w_1 u^2 + 4(d-1)w_2 u^2 &= \int_{-\infty}^{\infty} \mathcal{N}(\boldsymbol{x}; \mathbf{0}, I)\, x_1^2 d\boldsymbol{x} = 1 \end{aligned}$$

$$\begin{aligned} 2w_1u^4 + 4(d-1)w_2u^4 &= \int_{-\infty}^{\infty} \mathcal{N}(\boldsymbol{x}; \mathbf{0}, I)\, x_1^4 d\boldsymbol{x} = 3 \\ 4w_2u^4 &= \int_{-\infty}^{\infty} \mathcal{N}(\boldsymbol{x}; \mathbf{0}, I)\, x_1^2 x_2^2 d\boldsymbol{x} = 1 \end{aligned}$$

For example, in the second equation we have the term $4(d-1)w_2u^2$ because there are $4(d-1)$ points with the first coordinate being $\pm u$ in the generator $[\pm u, \pm u]$. If we divide the third equation by the second we obtain $u^2 = 3$ and the other values follow immediately. We get $u = \sqrt{3}$, $w_0 = 1 + \frac{d^2-7d}{18}$, $w_1 = \frac{4-d}{18}$ and $w_2 = \frac{1}{36}$.

The technique of exact monomial rules provides a new and useful perspective on the *Unscented Filter (UF)* [JU97] which was suggested as an alternative to EKF for tracking non-linear dynamic systems. The UF can be extremely accurate, even in cases where the EKF leads to a poor approximation [JU97]. We point out that the Unscented Filter is exactly our precision 3 rule from Equation 6.7, and the superior performance compared to EKF is an instance of the generally better accuracy of the numerical integration approach compared to EKF, even when using one of the least accurate numerical integration rules.

This view of the Unscented Filter has immediate practical consequences: We can trade off between the accuracy of the computation and its computational requirements. For example, if we are interested in a more precise rule than the Unscented Filter and are willing to evaluate the function at $O(d^2)$ points then we can use the exact monomial rule of precision 5. Depending on the function, this may represent a significant gain in accuracy.

6.2.4 Dealing with General Gaussians

The integrals that appear in Equation 6.2 through Equation 6.4 have a weight function $W(\boldsymbol{X}) = \mathcal{N}(\boldsymbol{X}; \boldsymbol{\mu}, \Sigma)$ and in general represent a different weight function than $\mathcal{N}(\boldsymbol{X}; \mathbf{0}, I)$. Thus, we cannot directly use our set of points chosen for $\mathcal{N}(\mathbf{0}, I)$ for general Gaussians. Fortunately, we can transform these points to any general Gaussian while keeping the same accuracy guarantees.

Theorem 6.2 *Let $P_1(\boldsymbol{X}) = \mathcal{N}(\boldsymbol{X}; \boldsymbol{0}, I)$ and $P_2(\boldsymbol{X}) = \mathcal{N}(\boldsymbol{X}; \boldsymbol{\mu}, \Sigma)$. Then for every square matrix A such that $AA^T = \Sigma$:*

$$\int_{-\infty}^{\infty} P_2(\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x} = \int_{-\infty}^{\infty} P_1(\boldsymbol{x}) f(A\boldsymbol{x} + \boldsymbol{\mu}) d\boldsymbol{x}$$

Before proving the theorem let us consider its usefulness. First, since Σ is positive definite it is always possible to find such a matrix A, often called the square root of Σ. For example, we can use Cholesky's decomposition [PTVF88] and find such an A which is also lower triangular. Once we have A, Theorem 6.2 lets us transform the integral that involves a general Gaussian into an integral that involves the standard Gaussian $\mathcal{N}(\boldsymbol{0}, I)$. We can therefore use the same set of points that we found for $\mathcal{N}(\boldsymbol{0}, I)$ to compute an integral with any Gaussian — all we have to do is apply the linear transformation $A\boldsymbol{X} + \boldsymbol{\mu}$ to every point. Intuitively this linear transformation corresponds to scaling the point to the covariance matrix Σ and shifting it according to the mean $\boldsymbol{\mu}$.

Proof: Note that, from $AA^T = \Sigma$ it follows that $A^T = A^{-1}\Sigma = (\Sigma^{-1}A)^{-1}$. Also note that, since $|A| = |A^T|$, $|A| = \sqrt{|\Sigma|}$. We can therefore write:

$$\begin{aligned}
P_2(A\boldsymbol{y} + \boldsymbol{\mu}) &= \frac{1}{\sqrt{2\pi|\Sigma|}} \exp\left(-\frac{1}{2}(A\boldsymbol{y} + \boldsymbol{\mu} - \boldsymbol{\mu})^T \Sigma^{-1} (A\boldsymbol{y} + \boldsymbol{\mu} - \boldsymbol{\mu})\right) \\
&= \frac{1}{\sqrt{2\pi}|A|} \exp\left(-\frac{1}{2}\boldsymbol{y}^T A^T \Sigma^{-1} A\boldsymbol{y}\right) \\
&= \frac{1}{\sqrt{2\pi}|A|} \exp\left(-\frac{1}{2}\boldsymbol{y}^T I \boldsymbol{y}\right) \\
&= \frac{P_1(\boldsymbol{y})}{|A|}
\end{aligned}$$

To compute the integral $\int_{-\infty}^{\infty} P_2(\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x}$ we perform the change of variables $\boldsymbol{x} = A\boldsymbol{y} + \boldsymbol{\mu}$ and get:

$$\int_{-\infty}^{\infty} P_2(\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x} = \int_{-\infty}^{\infty} |A| P_2(A\boldsymbol{y} + \boldsymbol{\mu}) f(A\boldsymbol{y} + \boldsymbol{\mu}) d\boldsymbol{y}$$

```
Numerical Integration for non-linear CPDs
   1. Sort nodes in topological order X_1, ..., X_n
   2. For i = 1..n do
      3. If CPD of X_i is not linear
         4. Find moments of P(Par(X_i), X_i) and approximate as a Gaussian
         5. Find covariances between X_i and {X_1, ..., X_{i-1}} - Par(X_i)
            using Equation 6.9
      6. else
         7. Compute first two moments of P(X_1, ..., X_i) using Theorem 3.3
```

Figure 6.2: The numerical integration approach

$$
\begin{aligned}
&= \int_{-\infty}^{\infty} |A| \frac{P_1(\boldsymbol{y})}{|A|} f(A\boldsymbol{y} + \boldsymbol{\mu}) d\boldsymbol{y} \\
&= \int_{-\infty}^{\infty} P_1(\boldsymbol{y}) f(A\boldsymbol{y} + \boldsymbol{\mu}) d\boldsymbol{y}
\end{aligned}
$$

∎

6.2.5 Putting it all together

We can now adapt the algorithm in Figure 6.1 to the numerical integration approach, as shown in Figure 6.2. Again we sort the variables in topological order and for linear CPDs simply use Theorem 3.3. For non-linear CPDs we compute the moments of the CPD variable and its parents using one of our integration rules: We use the simple grid approach of the Gaussian Quadrature rule for low dimensions (up to $d = 2$ or $d = 3$) and for higher dimensions we use one of the exact monomial rules such as Equation 6.7 or Equation 6.8.

It is only left to explain how to compute the covariances with the variables in $\{X_1, \ldots, X_{i-1}\} - \mathrm{Par}(X_i)$. As before, we can add them to the CPD and include them in the integrals that we compute. However, this approach leads to integrals of very high dimensions, forcing us to use relatively inaccurate integration rules. Fortunately, we can use an alternative approach.

Let $Y = f(\boldsymbol{X})$ be some variable with a non-linear CPD, and let $Z \notin \boldsymbol{X}$ be some variable that appeared before Y in our topological order. We are interested in finding

$\mathrm{Cov}(Z, Y)$. Since Z was already processed, we have a Gaussian approximation for $P(\boldsymbol{X}, Z)$. We can therefore use Corollary 3.5 in order to represent Z as a linear combination of $\boldsymbol{X}$, i.e.,

$$Z = \beta_0 + \sum_i \beta_i X_i + W$$

Using this representation of Z we have:

$$\begin{aligned}
\mathrm{Cov}(Z, Y) &= E[ZY] - E[Z]E[Y] = \\
&= E[(\beta_0 + \sum_i \beta_i X_i + W)Y] - E[\beta_0 + \sum_i \beta_i X_i + W]E[Y] \\
&= \beta_0 E[Y] + \sum_i \beta_i E[X_i Y] + E[WY] - \\
&\quad\ \beta_0 E[Y] - \sum_i \beta_i E[X_i]E[Y] - E[W]E[Y] \\
&= \sum_i \beta_i (E[X_i Y] - E[X_i]E[Y]) + E[W]E[Y] - E[W]E[Y] \\
&= \sum_i \beta_i \mathrm{Cov}(X_i, Y) \qquad (6.9)
\end{aligned}$$

The reason that $E[WY] = E[W]E[Y]$ is that W is independent of $\boldsymbol{X}$ and therefore it is also independent of Y. We note that representing Z as a linear combination of $\boldsymbol{X}$ involves solving a set of linear equations $A\boldsymbol{x} = \boldsymbol{b}$ where A is the covariance matrix of $P(\boldsymbol{X})$ and $\boldsymbol{b} = (\mathrm{Cov}(X_1, Z), \ldots, \mathrm{Cov}(X_n, Z))$. If we need to perform this computation for more than one node we can do so efficiently by decomposing A once using Cholesky decomposition and then using it to solve each set of linear equations (see [PTVF88] for more details).

Computing $\mathrm{Cov}(Z, Y)$ using Equation 6.9 is exact under two conditions:

- There are no errors in $\mathrm{Cov}(X_i, Y)$
- The true joint probability distribution $P(\boldsymbol{X}, Z)$ is indeed a Gaussian

In other words, if $P(\boldsymbol{X}, Z)$ is not truly a Gaussian (because Z, $\boldsymbol{X}$ or one of their ancestors has a non-linear CPD) then we will not get the correct $\mathrm{Cov}(Z, Y)$ even if we were able to find the correct first two moments for $P(\boldsymbol{X}, Y)$. The reason is that we are using only an approximation for $P(\boldsymbol{X}, Z)$ and might suffer from errors resulting from this approximation. It is possible not to rely on the Gaussian estimate for $P(\boldsymbol{X}, Z)$ by using integrals that might involve more variables, such as the ancestors of $\boldsymbol{X}$. In practice we often end up with integrals of higher dimensions, and the inaccuracies that arise due to less accurate integration rules often negate the potential improvement of not using the Gaussian approximation for $P(\boldsymbol{X}, Z)$.

6.2.6 Encapsulated Variables

Just as we use the structure of the Bayesian network to decompose the dependency between the various variables, in many cases it is possible to further decompose the non-linear dependency $Y = f(\boldsymbol{X})$. For example, we may be able to decompose the non-linear function f as $f(\boldsymbol{X}) = g(g_1(\boldsymbol{X}_1), g_2(\boldsymbol{X}_2))$, where $\boldsymbol{X}_1, \boldsymbol{X}_2 \subseteq \boldsymbol{X}$.[2] Instead of directly approximating the Gaussian over $\{\boldsymbol{X}, Y\}$ we can define two extra variables: $T_1 = g_1(\boldsymbol{X}_1)$ and $T_2 = g_2(\boldsymbol{X}_2)$. We first find a Gaussian approximation to $P(\boldsymbol{X}_1, T_1)$. Next we approximate $P(\boldsymbol{X}_2, T_2)$ as a Gaussian and using Equation 6.9 we approximate $P(\boldsymbol{X}, T_1, T_2)$ as a Gaussian. Finally, we approximate $P(T_1, T_2, Y)$ as a Gaussian and again use Equation 6.9 to find the Gaussian approximation for $P(\boldsymbol{X}, T_1, T_2, Y)$. The same accuracy tradeoffs that were discussed in the previous section apply here: by reducing the dimension of the integrals we can solve each one more accurately, but may introduce further errors if the interactions between the extra variables are non-linear.

In principle, one could add T_1 and T_2 to the Bayesian network and treat them as regular variables. However, doing so increases the number of variables in our Gaussian, denoted as n in Figure 6.2, and thereby increases the algorithm's space complexity, which is $O(n^2)$ (as we need to represent a covariance matrix). It is better to treat the extra variables as local variables *encapsulated* within the CPD and unknown to the rest of the network. After computing the Gaussian approximation

[2] As an example, consider a gas flow sensor which is used in the RWGS system, described in Chapter 10. For a given gas flow, a flow sensor gives different readings depending on the gas type. Assume we have random variables that represent the total flow F and the compositions of the different gases in it α_1, α_2. The function g_i $(i = 1, 2)$ may be the product $\alpha_i F$, representing the net flow of each one of the gases. The function g would be a weighted sum of these flows where the weights correspond to the sensor's response for the different gases.

for the CPD variables, we simply marginalize out the encapsulated variables and do not compute the covariance between them and any other network variable. In our example, we reduce to Gaussian $P(\boldsymbol{X}, T_1, T_2, Y)$ to the Gaussian $P(\boldsymbol{X}, Y)$. In terms of the algorithm in Figure 6.2, this operation can be viewed as an implementation of line 4, where the encapsulated variables are not considered to be variables of the Bayes net.

Note that each encapsulated variable belongs to just one CPD, and therefore can have at most one non-encapsulated child. This approach is very similar to the local computations in an OOBN model [KP97], where some of the variables are local variables, encapsulated within a CPD.

6.3 Moments Correction

Unfortunately, our numerical integration approach has one significant problem, as nothing guarantees that the resulting covariance matrix will be positive definite or even positive semi-definite. For example, the estimated variance is the difference $E[Y^2] - E[Y]^2$. If we underestimate $E[Y^2]$ or overestimate $E[Y]$, we might end up with a negative variance for Y.

One possible way of dealing with this problem is to use a more accurate integration rule, which often solves the problem in practice. However, the problem may persist even when using a more accurate rule. Furthermore, a more accurate rule is computationally more expensive, and so it may be undesirable to use it.

An alternative approach is to find the "closest" positive definite covariance matrix. We cast this problem as a convex optimization problem following [BV03, Ber95]. Consider once again the problem of approximating $P(\boldsymbol{X}, Y)$ as a multivariate Gaussian, where Y is a nonlinear function of its parents $\boldsymbol{X} = \{X_1, \ldots, X_n\}$, i.e., $Y = f(\boldsymbol{X})$, and $P(\boldsymbol{X}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, A)$. Assume that the results of our numerical integration rule were $\bar{a} = E[Y]$, $\bar{b} = E[Y^2]$ and $\bar{c_i} = E[X_i Y]$. We also denote $\bar{\boldsymbol{c}} = (\bar{c_1}, \ldots, \bar{c_n})^T$. Recall that $\text{Var}(Y) = E[Y^2] - E[Y]^2$ and that $\text{Cov}(X_i, Y) = E[X_i Y] - E[X_i]E[Y]$. We therefore have the following covariance matrix:

$$\Sigma = \left[\begin{array}{c|c} A & \bar{\boldsymbol{c}} - \bar{a}\boldsymbol{\mu} \\ \hline (\bar{\boldsymbol{c}} - \bar{a}\boldsymbol{\mu})^T & \bar{b} - \bar{a}^2 \end{array}\right] \tag{6.10}$$

We would like to make sure that Σ is a legal covariance matrix, i.e., it positive definite. The following theorem is useful (its proof is given in Appendix A.3).

Theorem 6.3 *Let Σ be a symmetric matrix of the form*

$$\Sigma = \left[\begin{array}{c|c} A & \boldsymbol{u} \\ \hline \boldsymbol{u}^T & v \end{array}\right]$$

where A is a positive definite matrix of dimension $n \times n$, $\boldsymbol{u}$ is a vector of dimension n and v is a scalar. Then Σ is positive definite iff $v - \boldsymbol{u}^T A^{-1}\boldsymbol{u} > 0$.

Since the distribution of $\boldsymbol{X}$ is known and assumed to be legal, A and $\boldsymbol{\mu}$ in Equation 6.10 are fixed, and we are therefore only allowed to change $\bar{a}, \bar{b}$ and $\bar{\boldsymbol{c}}$. We would like to change them "as little as possible" in order to make Σ positive definite. We can formulate this as an optimization problem (using the L2-norm to express "closeness"):

$$\text{Minimize} \quad f(a, b, \boldsymbol{c}) = (a - \bar{a})^2 + (b - \bar{b})^2 + \| \boldsymbol{c} - \bar{\boldsymbol{c}} \|^2 \tag{6.11}$$

$$\text{Subject to} \quad f_1(a, b, \boldsymbol{c}) = (\boldsymbol{c} - a\boldsymbol{\mu})^T A^{-1}(\boldsymbol{c} - a\boldsymbol{\mu}) - (b - a^2) + \epsilon \leq 0 \tag{6.12}$$

Note that if $\epsilon = 0$ then we demand that Σ is only semi-positive definite. To make sure it is positive definite ϵ needs to be a small positive number. Also note that if the original estimates $\bar{a}, \bar{b}$ and $\bar{\boldsymbol{c}}$ result in a legal covariance matrix then they must be the solution of the optimization problem and we are done.

Since A and therefore A^{-1} are positive definite, both the functions f and f_1 are convex in a, b and $\boldsymbol{c}$, and we can use convex optimization techniques to solve our problem. We use the Lagrange multiplier λ and form the Lagrangian

$$
\begin{aligned}
L(a,b,\boldsymbol{c},\lambda) &= f(a,b,\boldsymbol{c}) + \lambda f_1(a,b,\boldsymbol{c}) \\
&= (a-\bar{a})^2 + (b-\bar{b})^2 + \parallel \boldsymbol{c} - \bar{\boldsymbol{c}} \parallel^2 + \qquad (6.13)\\
&\quad \lambda\left((\boldsymbol{c} - a\boldsymbol{\mu})^T A^{-1}(\boldsymbol{c} - a\boldsymbol{\mu}) + a^2 - b + \epsilon\right) \qquad (6.14)
\end{aligned}
$$

We now take the standard approach of solving the *dual problem*, i.e.:

$$
\begin{aligned}
&\text{Maximize} \quad g(\lambda) \stackrel{def}{=} \inf_{a,b,c} L(a,b,\boldsymbol{c},\lambda) && (6.15)\\
&\text{Subject to} \quad \lambda \geq 0 && (6.16)
\end{aligned}
$$

The function $g(\lambda)$ is called the *dual function.* Since L is convex with respect to $a, b, \boldsymbol{c}$ we can find $\inf_{a,b,c} L(a,b,\boldsymbol{c},\lambda)$ by setting the partial derivatives to 0:

$$
\begin{aligned}
\frac{\partial L}{\partial a} &= 2(a-\bar{a}) + \lambda\left(2\boldsymbol{\mu}^T A^{-1}(a\boldsymbol{\mu} - \boldsymbol{c}) + 2a\right) && (6.17)\\
\frac{\partial L}{\partial b} &= 2(b-\bar{b}) - \lambda && (6.18)\\
\frac{\partial L}{\partial \boldsymbol{c}} &= 2(\boldsymbol{c} - \bar{\boldsymbol{c}}) + 2\lambda A^{-1}(\boldsymbol{c} - a\boldsymbol{\mu}) && (6.19)
\end{aligned}
$$

Setting Equation 6.19 to 0, we get

$$
\begin{aligned}
\left(I + \lambda A^{-1}\right)\boldsymbol{c} &= \bar{\boldsymbol{c}} + \lambda A^{-1} a\boldsymbol{\mu} \\
\boldsymbol{c} &= \left(I + \lambda A^{-1}\right)^{-1}\left(\bar{\boldsymbol{c}} + \lambda A^{-1} a\boldsymbol{\mu}\right) && (6.20)
\end{aligned}
$$

When we set Equation 6.18 to 0 we get

$$
b = \bar{b} + \frac{\lambda}{2} \qquad (6.21)
$$

We now set Equation 6.17 to 0:

$$
\left(1 + \lambda + \lambda\boldsymbol{\mu}^T A^{-1}\boldsymbol{\mu}\right) a = \bar{a} + \lambda\boldsymbol{\mu}^T A^{-1}\boldsymbol{c} \qquad (6.22)
$$

Plugging Equation 6.20 into Equation 6.22 we get:

$$\begin{aligned}\left(1+\lambda+\lambda\boldsymbol{\mu}^T A^{-1}\boldsymbol{\mu}\right)a &= \bar{a}+\lambda\boldsymbol{\mu}^T A^{-1}\left(I+\lambda A^{-1}\right)^{-1}\left(\bar{\boldsymbol{c}}+\lambda A^{-1}a\boldsymbol{\mu}\right)\\ &= \bar{a}+\lambda\boldsymbol{\mu}^T(A+\lambda I)^{-1}\left(\bar{\boldsymbol{c}}+\lambda A^{-1}a\boldsymbol{\mu}\right)\\ &= \bar{a}+\lambda\boldsymbol{\mu}^T(A+\lambda I)^{-1}\bar{\boldsymbol{c}}+\lambda^2\boldsymbol{\mu}^T(A^2+\lambda A)^{-1}\boldsymbol{\mu}a \quad (6.23)\end{aligned}$$

Solving Equation 6.23 for a we get:

$$a=\frac{\bar{a}+\lambda\boldsymbol{\mu}^T(A+\lambda I)^{-1}\bar{\boldsymbol{c}}}{1+\lambda+\lambda\boldsymbol{\mu}^T A^{-1}\boldsymbol{\mu}-\lambda^2\boldsymbol{\mu}^T(A^2+\lambda A)^{-1}\boldsymbol{\mu}} \tag{6.24}$$

Equations 6.20, 6.21, and 6.24 define the infimum of a, b and $\boldsymbol{c}$ for every value of λ. We can therefore plug these equations into Equation 6.14 and get the function $g(\lambda)$ as defined in Equation 6.15. The function we get is concave in λ [BV03, Ber95] and therefore it is not hard to find the value of λ that maximizes it. In particular, we can find λ by solving the equation $g'(\lambda)=0$. Finding g' may appear cumbersome, but in fact it is not, since we can use the following lemma.

Lemma 6.4 *Consider the convex optimization problem*

$$\begin{array}{ll}\textit{Minimize} & f(\boldsymbol{x})\\ \textit{Subject to} & f_1(\boldsymbol{x})\le 0\end{array}$$

where both f and f_1 are convex and differentiable. For $\lambda>0$ define the Lagrangian as $L(\boldsymbol{x},\lambda)=f(\boldsymbol{x})+\lambda f_1(\boldsymbol{x})$ and the dual function $g(\lambda)=L(\boldsymbol{x}(\lambda),\lambda)$ where we define the function $\boldsymbol{x}(\lambda)$ as $\boldsymbol{x}(\lambda)=\arg\inf_{\boldsymbol{x}} L(\boldsymbol{x},\lambda)$. Then $g'(\lambda)$ is a monotonically decreasing function and $g'(\lambda)=f_1(\boldsymbol{x}(\lambda))$.

Proof: Since g is concave we know that g' is monotonically decreasing. To show that $g'(\lambda)=f_1(\boldsymbol{x}(\lambda))$ note that $g(\lambda)=f(\boldsymbol{x}(\lambda))+\lambda f_1(\boldsymbol{x}(\lambda))$ and therefore

$$\begin{aligned}g'(\lambda) &= \nabla f'(\boldsymbol{x}(\lambda))\cdot\nabla\boldsymbol{x}(\lambda)+f_1(\boldsymbol{x}(\lambda))+\lambda\nabla f_1(\boldsymbol{x}(\lambda))\cdot\nabla\boldsymbol{x}(\lambda)\\ & f_1(\boldsymbol{x}(\lambda))+\nabla\boldsymbol{x}(\lambda)\cdot[\nabla f(\boldsymbol{x}(\lambda))+\lambda\nabla f_1(\boldsymbol{x}(\lambda))]\end{aligned}$$

But $\boldsymbol{x}(\lambda)$ was the defined as the infimum of $f(\boldsymbol{x}) + \lambda f_1(\boldsymbol{x})$. Since both f and f_1 are convex, the gradient with respect to $\boldsymbol{x}$ at the infimum must be zero, i.e., $\nabla f(\boldsymbol{x}(\lambda)) + \lambda \nabla f_1(\boldsymbol{x}(\lambda)) = 0$. Therefore $g'(\lambda) = f_1(\boldsymbol{x}(\lambda))$. ∎

Thus, to find to optimal value of λ we need to solve the equation

$$g'(\lambda) = f_1(a(\lambda), b(\lambda), \boldsymbol{c}(\lambda)) = 0 \tag{6.25}$$

where f_1 is defined in Equation 6.12 and the functions a, b and $\boldsymbol{c}$ are defined using Equations 6.20, 6.21, and 6.24. Since g' is monotonically decreasing in λ, we can solve Equation 6.25 using bisection. To do so we must find one value $\lambda = \lambda_0$ for which g' is non-negative and one value $\lambda = \lambda_1$ for which g' is non-positive. For the value of λ_0 note that $a(0) = \bar{a}, b(0) = \bar{b}$ and $\boldsymbol{c}(0) = \bar{\boldsymbol{c}}$. Since $\bar{a}, \bar{b}, \bar{\boldsymbol{c}}$ violate the constraint of Equation 6.12 we know that $f_1(\bar{a}, \bar{b}, \bar{\boldsymbol{c}}) > 0$ and thus, we can simply choose $\lambda_0 = 0$.

To find a value for λ_1 it is convenient to define the function h:

$$\begin{aligned}
f_1(a(\lambda), b(\lambda), \boldsymbol{c}(\lambda) &= h(a(\lambda), \boldsymbol{c}(\lambda)) - b(\lambda) \\
h(a(\lambda), \boldsymbol{c}(\lambda)) &= (\boldsymbol{c}(\lambda) - a(\lambda)\boldsymbol{\mu})^T A^{-1} (\boldsymbol{c}(\lambda) - a(\lambda)\boldsymbol{\mu}) + a(\lambda)^2 + \epsilon
\end{aligned}$$

Since f_1 is monotonically decreasing in λ, so is h, and we can therefore write:

$$\begin{aligned}
g'(\lambda) &= f_1(a(\lambda), b(\lambda), \boldsymbol{c}(\lambda)) \\
&= h(a(\lambda), \boldsymbol{c}(\lambda)) - b(\lambda) \\
&= h(a(\lambda), \boldsymbol{c}(\lambda)) - \bar{b} - \frac{\lambda}{2} \\
&\leq h(a(0), \boldsymbol{c}(0)) - \bar{b} - \frac{\lambda}{2} \\
&= f_1(\bar{a}, \bar{b}, \bar{\boldsymbol{c}}) - \frac{\lambda}{2}
\end{aligned}$$

It follows that if we choose $\lambda_1 = 2 f_1(\bar{a}, \bar{b}, \bar{\boldsymbol{c}})$ we have $g'(a(\lambda_1), b(\lambda_1), \boldsymbol{c}(\lambda_1)) \leq 0$. Thus, we can use λ_0 and λ_1 as a lower and an upper bound for the actual value of λ and then use bisection in order to solve Equation 6.25. Once we have λ we use Equations 6.20, 6.21, and 6.24 to compute a, b and $\boldsymbol{c}$ and plug them into Equation 6.10

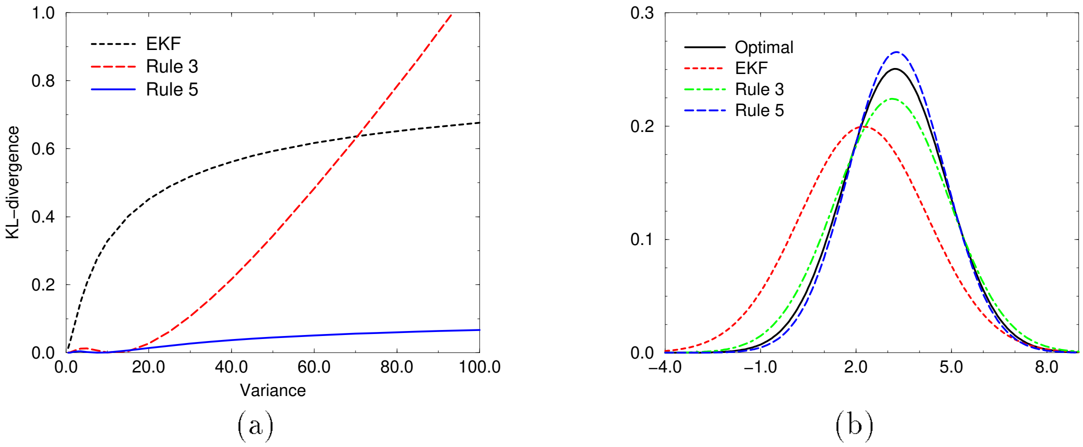


Figure 6.3: Comparison of Gaussian approximations for $Y = \sqrt{X_1^2 + X_2^2}$ (a) KL-divergence from optimal approximation (b) Resulting distribution when $\mathrm{Var}(X_1) = \mathrm{Var}(X_2) = 4$

to get our new covariance matrix. This represents an efficient algorithm to correct the integration results, guaranteeing a legal covariance matrix.

In terms of the algorithm from Figure 6.2, we can plug in this procedure between lines 4 and 5, correcting the moments of $P(\mathrm{Par}(X_i), X_i)$. Note that if $P(\mathrm{Par}(X_i), X_i)$ represents a legal Gaussian and $P(X_1, \ldots, X_{i-1})$ represents a legal Gaussian, then $P(X_1, \ldots, X_i)$ generated in line 5 will also be a legal Gaussian.

6.4 An Example

We compare the EKF approach and the exact monomial rules using a simple non-linear function $Y = \sqrt{X_1^2 + X_2^2}$. For simplicity we assume that X_1 and X_2 are independent and that $P(X_1) = \mathcal{N}(X_1; 0, \sigma^2)$ and $P(X_2) = \mathcal{N}(X_2; 2, \sigma^2)$. We computed a Gaussian approximation for the distribution of Y using EKF, an exact monomial rule of precision 3 and an exact monomial rule of precision 5. In this example the precision 5 rule always led to legal covariance matrices, but the precision 3 rule resulted in illegal covariance matrices for $\sigma > 0.23$. Thus, the results reported here for the precision 3 rule include a correction using the technique described in Section 6.3.

	Mean	Variance	KL-divergence
Optimal	3.235	2.538	0
EKF	2.236	4	0.169
Rule 3	3.142	3.173	0.013
Rule 5	3.277	2.263	0.00381

Table 6.1: Comparison of Gaussian approximations for $\mathrm{Var}(X_1) = \mathrm{Var}(X_2) = 4$

Figure 6.3(a) shows the KL-divergence for $P(Y)$ between the optimal approximation and the various methods. As our optimal approximation we used a very accurate Gaussian Quadrature rule which used a grid of 100×100 integration points. In general, the quality of the approximation of every method degrades as we increase σ^2. This is to be expected since all the methods are accurate for low order polynomials, and the larger σ^2 is, the larger is the contribution of the higher order terms.

For small and medium variances, EKF is the least exact of the three methods. For large variances, the precision 3 rule becomes the least accurate but the precision 5 rule maintains a much better precision than EKF. The reason for the precision 3 rule behavior is the illegal covariance matrices for $\sigma > 0.23$ — the moments correction is capable of dealing with this problem as long as σ is not too large, but for $\sigma > 4$ the inaccuracies become more and more significant. However, it is important to note that for high variances, the Gaussian approximation is not a very good approximation to begin with and therefore it is not as important to be very close to the optimal Gaussian approximation. For low variances, where the approximation is a good one even the corrected precision 3 rule significantly dominates the EKF approximation. Figure 6.3(b) shows the distribution of Y for $\sigma = 2$ — the precision 5 rule is quite accurate but even the corrected precision 3 rule is significantly more accurate than the EKF. The results are also summarized in Table 6.1, including the KL-divergence from the optimal approximation.

6.5 Combination with the Enumeration Algorithm

So far, we have focused on networks that contain only continuous variables, and saw how to approximate their joint distribution as a multivariate Gaussian. In hybrid networks, these ideas can be combined very easily with the enumeration algorithm as presented in Section 5.1.4.

Recall that the enumeration algorithm enumerates discrete assignments from the prior and for each one computes the corresponding Gaussian. Adding non-linear CPDs does not change the prior distribution of the discrete variables, and therefore, we can use the exact same algorithm for enumerating the discrete assignments. Given a discrete assignment we are back to the case of a distribution over the continuous variables, where some of the CPDs are not linear. Thus, for every discrete instantiation we can use the numerical integration approach (or EKF) in order to approximate the induced distribution over the continuous variables as a Gaussian. We can also use our moments correction algorithm in order to make sure that all the covariance matrices are positive definite. Note that using the same approach we can also combine RB-LW or any other algorithm that generates discrete assignment with our techniques to deal with non-linear CPDs.

Chapter 7

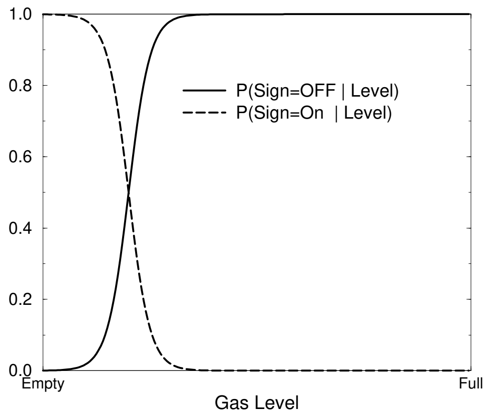
Augmented CLGs

One of the main restrictions of CLGs is that the graphical model does not allow discrete variables to have continuous parents, a dependency that arises in many domains. For example, consider a feedback control loop involving a thermostat, which controls the room temperature by turning on or off a heating device and a cooling system. The thermostat should be modeled using a discrete variable ("heating on", "cooling on", and "idle") which depends on the continuous variable representing the room temperature. As another example, consider the modeling of sensors such as the warning sign which appears when the gas level in a car's gas tank gets low. Once again, we have a discrete variable ("sign on" or "sign off") which depends on a continuous one (gas level).

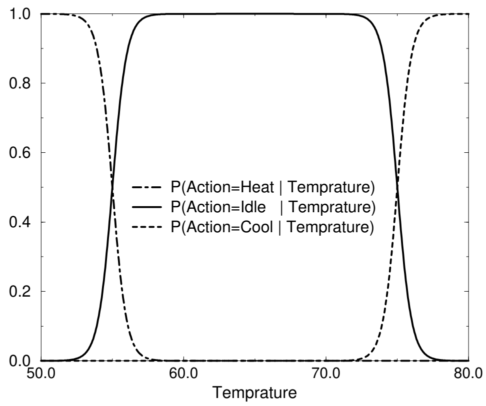
In this chapter we define a class of *augmented* CLGs and discuss inference in these models. In augmented CLGs the CPDs of the continuous nodes are linear CPDs, but we also allow discrete nodes to depend on continuous parents. We use *continuous-discrete (CD)* to refer to such CPDs.

7.1 Representation

There are many possible functional forms that can be used to represent CD CPDs. One of the most useful is a *softmax* or *logistic* function. Let A be a discrete node



(a)



(b)

Figure 7.1: Examples of softmax CPDs: (a) The low gas level warning sign (b) The thermostat

with the possible values $a_1, ..., a_m$, and let $X_1, ..., X_k = \boldsymbol{X}$ be its parents. We define:

$$P(A = a_i \mid x_1, ..., x_k) = \frac{\exp\left(w_0^{(i)} + \sum_{l=1}^{k} w_l^{(i)} x_l\right)}{\sum_{j=1}^{m} \exp\left(w_0^{(j)} + \sum_{l=1}^{k} w_l^{(j)} x_l\right)}. \tag{7.1}$$

The case where A also has discrete parents is modeled as in CLGs: We define a different softmax function for every combination of the discrete parents.

It is possible to eliminate one of the linear combinations by dividing both numerator and denominator by $\exp\left(w_0^{(m)} + \sum_{l=1}^{k} w_l^{(m)} x_l\right)$, leaving us with $m-1$ sets of parameters (for the m-th set we get that after the division $w_l^{(m)} = 0$ for $0 \leq l \leq k$). In particular, when A is binary, this new form simplifies to a standard sigmoid function. For $0 \leq l \leq k$ define $w_l = w_l^{(0)} - w_l^{(1)}$ and $\boldsymbol{w}^T = (w_1, \ldots, w_k)$. Then:

$$P(A = a_1 \mid x_1, ..., x_k) = \frac{1}{1 + \exp\left(w_0 + \sum_{l=1}^{k} w_l x_l\right)} = \frac{1}{1 + \exp\left(w_0 + \boldsymbol{w}^T \boldsymbol{x}\right)}. \tag{7.2}$$

Figure 7.1 shows two examples of softmax CPDs. In Figure 7.1(a) we have a softmax function for a binary variable, corresponding to the low gas level warning sign. When the tank is full the sign is always off, and when the tank is empty the

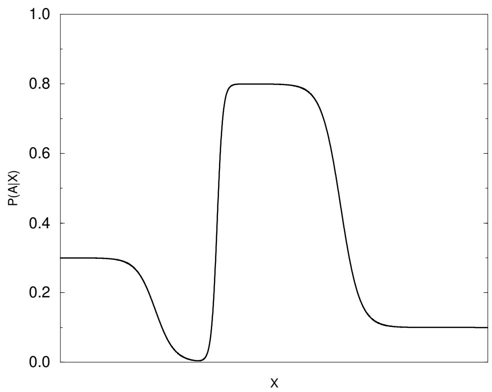


Figure 7.2: Generalized softmax CPD

sign is always on. The parameters of the softmax CPD determine the slope of the transition and its location: w_1 determines the slope and $-w_0/w_1$ determines the location. Figure 7.1(b) models the thermostat. When the temperature is below 55^o, the thermostat is likely to turn on the heating and when it is above 75^o, it is likely to turn on the cooling system. In between, the thermostat keeps both appliances off. The location of the transitions is determined by the relative values of the w_l's, and the slope of the transition, which corresponds to the noise level, by their magnitude.

It is possible to generalize the softmax functions to express fairly complex distributions. Intuitively, the generalized softmax CPD defines a set of R regions (for some parameter R of our choice). The regions are defined by a set of R linear functions over the continuous variables. A region is characterized as the part of the space where one particular linear function is larger than all the others. Each region is also associated with some distribution over the values of the discrete child; this distribution is the one used for the variable within this region. The actual CPD is a continuous version of this region-based idea, allowing for smooth transitions between the distributions in neighboring regions of the space.

More precisely, let A be a discrete variable, with the continuous parents $\boldsymbol{X} = \{X_1, \ldots, X_k\}$. Assume that A has m possible values, $\{a_1, a_2, \ldots, a_m\}$. Each of the R regions is defined via two vectors of parameters $\boldsymbol{w}^{(r)}, \boldsymbol{p}^{(r)}$. The vector $\boldsymbol{w}^{(r)}$ is a vector of weights $w_0^{(r)}, w_1^{(r)}, \ldots, w_k^{(r)}$ specifying the linear function associated with the region. The vector $\boldsymbol{p}^{(r)} = \{p_1^{(r)}, \ldots, p_m^{(r)}\}$ is the probability distribution over $a_1, \ldots, a_m$ associated with the region, i.e., $\sum_{j=1}^{m} p_j^{(r)} = 1$. The CPD is now defined as:

$$P(A = a_j \mid \boldsymbol{x}) = \sum_{r=1}^{R} \alpha^{(r)} p_j^{(r)}$$

where

$$\alpha^{(r)} = \frac{\exp(w_0^{(r)} + \sum_{i=1}^{k} w_i^{(r)} x_i)}{\sum_{q=1}^{R} \exp\left(w_0^{(q)} + \sum_{i=1}^{k} w_i^{(q)} x_i\right)}$$

In other words, the distribution is a weighted average of the region distributions, where the weight of each "region" depends exponentially on how large the value of its defining linear function is, relative to the rest.

The power to choose the number of regions R to be as large as we wish is the key to the rich expressive power of the generalized softmax CPD. Figure 7.2 demonstrates this expressivity. In this example we have four different regions. The choice of $\boldsymbol{w}^{(r)}$ determines both the regions and the slope of the transitions between them; the choice of $\boldsymbol{p}^{(r)}$ determines the distribution defining each region.

7.2 Lauritzens's Algorithm and Augmented CLGs

Our approach for inference in augmented CLGs is based on the observation that a Gaussian can be a good approximation to the product of a Gaussian and a softmax CPD [Mur99]. The quality of the approximation depends on the slope of the logistic function compared to the variance of the Gaussian, as is demonstrated in Figure 7.3. The sharper the logistic function is, the worse the approximation becomes, but in many cases the accuracy of the approximation is very high.

The observation that a product of a Gaussian and a logistic function can be approximated as a Gaussian suggests that one can extend both Lauritzen's algorithm and the enumeration algorithm to the case of augmented CLGs. Unfortunately, unlike the case of continuous non-linear CPDs, when we use augmented CLGs the enumeration algorithm becomes impractical, as we shall discuss in Section 7.3. In this section we concentrate on extending Lauritzen's algorithm to augmented CLGs using the numerical integration approach. Our goal is to come up with an algorithm that is exact in terms of the first two moments, up to numerical integration errors.

7.2.1 Variational Approach

We first review the algorithms of Murphy and Wiegerinck [Mur99, Wie00] for extending Lauritzen's algorithm to augmented CLGs using the variational approach. Roughly speaking the idea is to build a strong clique tree where we introduce *variational parameters* in the clique potentials, i.e., the canonical factors (defined in Section 3.3.2) are parameterized by some extra parameters. We then try to find a set of parameters that represents a good approximation to the distribution represented by the Bayesian network. When using the variational approach, we limit ourselves to networks where all discrete variables with continuous parents are binary and all the CD CPDs are of the form of Equation 7.2.

More formally, let $\sigma(x)$ be the sigmoid function $\sigma(x) \stackrel{def}{=} \frac{1}{1+\exp(-x)}$. Murphy [Mur99] suggests the use of the following bound on σ:

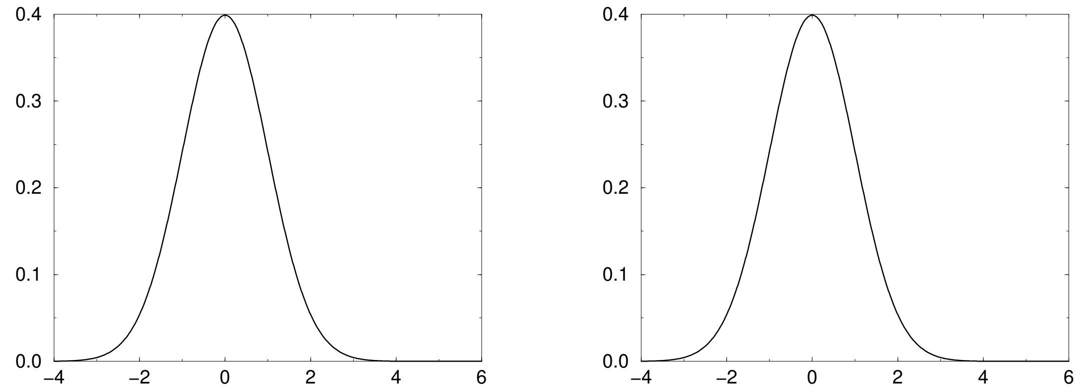
$$\log \sigma(x) \geq \lambda(\xi)x^2 + \frac{x}{2} + \log \sigma(\xi) - \frac{\xi}{2} - \xi^2\lambda(\xi) \tag{7.3}$$

where ξ can take any value and

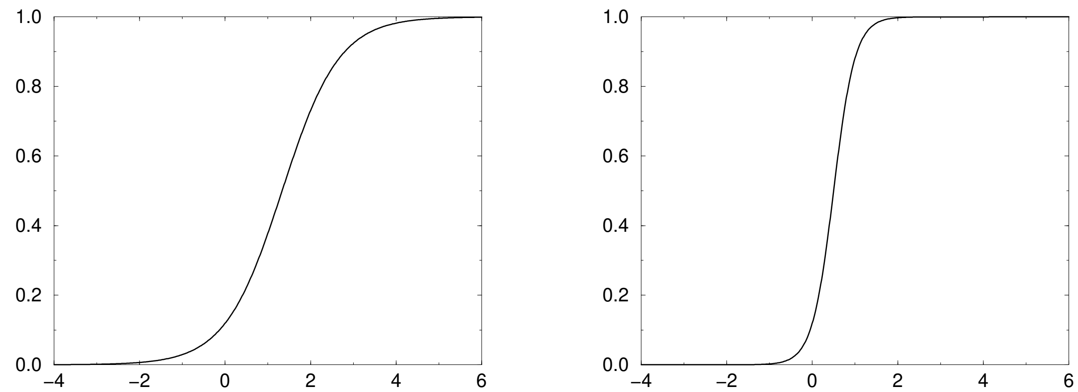
$$\lambda(\xi) = \frac{\frac{1}{2} - \sigma(\xi)}{2\xi}$$

Combining the bound from Equation 7.3 and our representation for a binary softmax CPD from Equation 7.2, we get the following bound for softmax CPDs (where $r \in \{0, 1\}$):

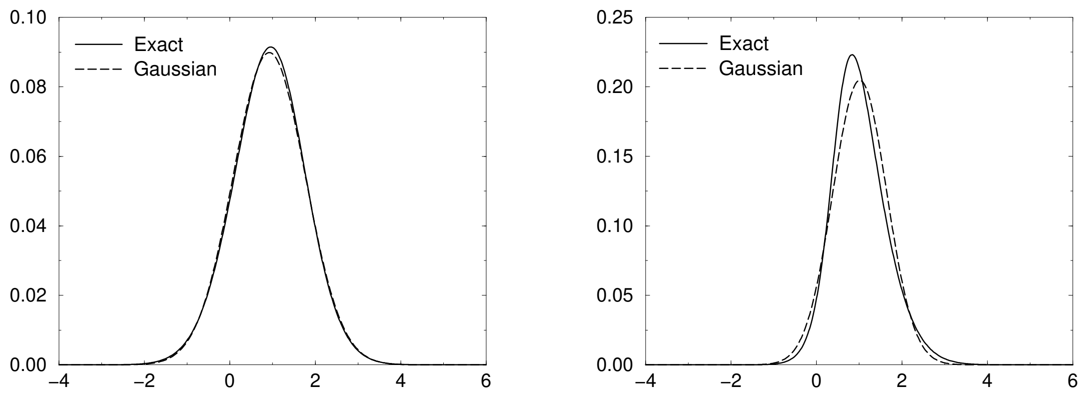
$$\begin{aligned} P(A = a_r \mid \boldsymbol{x}) &\geq -\frac{1}{2}\boldsymbol{x}^T K \boldsymbol{x} + \boldsymbol{h}^T \boldsymbol{x} + g \\ K &= -2\lambda(\xi)\boldsymbol{w}\boldsymbol{w}^T \\ \boldsymbol{h} &= \left(\frac{1}{2}(2r-1) + 2\lambda(\xi)b\right)\boldsymbol{w} \end{aligned} \tag{7.4}$$



(a) $\mathcal{N}(0,1)$



(b) Logistic CPD $w_0 = 2, w_1 = -1.5$ (c) Logistic CPD $w_0 = 2, w_1 = -4$



(d) Product of (a) and (b) (e) Product of (a) and (c)

Figure 7.3: Approximating the product of a Gaussian and a logistic CPD by a Gaussian

$$g \;=\; \log\sigma(\xi) + \frac{2r-1}{2}\cdot b - \frac{\xi}{2} + \lambda(\xi)(b^2-\xi^2)$$

where $\boldsymbol{w}^T = (w_1, \ldots, w_k)$. Equation 7.4 is a quadratic form. For every value of the variational parameter ξ we get a bound on the softmax CPD. We can multiply this quadratic form into some potential in our strongly rooted tree; the tree would then represent a distribution which is parameterized by the variational parameters. It is important to note that, even with the introduction of the variational parameters, our representation is still a mixture of Gaussians.

The goal is to find the "best" variational parameters that give us the best possible approximation. The variational approach does this by minimizing the KL-divergence $D(Q \parallel P)$ between the distribution Q represented by the tree and the true underlying distribution P.[1] This can be done by an iterative algorithm. For the variational parameter corresponding to Equation 7.4 we use the following update rule:

$$\xi^2 \leftarrow E[(\boldsymbol{w}^T\boldsymbol{X} + w_0)^2] = \boldsymbol{w}^T E[\boldsymbol{X}^2]\boldsymbol{w} + 2b\boldsymbol{w}^T E[\boldsymbol{X}] + w_0^2$$

where the moments of $\boldsymbol{X}$ are computed using Lauritzen's algorithm. Murphy [Mur99] discusses how to choose good initial values for the variational parameters which is an important problem since bad values might lead to a poor local maximum. Wiegerinck [Wie00] shows how the approximation can often be significantly improved by using different variational parameters for every Gaussian in the canonical factor, rather than using just one variational parameter for each softmax CPD.

While the variational approach is quite elegant it suffers from a few problems. First, it is currently limited to binary softmax distributions and seems to be quite hard to extend to generalized softmax CPD. More importantly, note that this method is based on optimizing the bound in Equation 7.3, and therefore can offer no real guarantees: the bound may not be tight, and even if it is, finding a lower bound for $\log\sigma(x)$ is not guaranteed to result in a good approximation for the posterior distribution. Furthermore, the iterative algorithm used to optimize the parameters

[1] It is more natural to minimize the other KL-divergence, i.e., $D(P \parallel Q)$, but it is much harder to do so.

is not guaranteed to converge to the global maximum, and we might end up with a poor local maximum. There is currently no method of estimating how close the result is to the best possible approximation within the same expressive power, i.e., the distribution which approximates the posterior as a mixture of Gaussians with the correct first two moments for every mixture component.

7.2.2 Numerical Integration Approach

It is possible to extend Lauritzen's algorithm to deal with augmented CLGs without performing a variational approximation by taking the more direct approach of numerical integration. The resulting algorithm can be made to be exact up to errors introduced by numerical integration. To simplify the presentation, we take the same approach as in Chapter 3 and present our algorithm in the context of canonical forms. It is then a simple matter to adapt the algorithm to work with conditional forms as we shall show in Section 7.2.5.

We begin with a simple motivating example. Consider the network $X \to A$, where X has a Gaussian distribution given by $P(X) = \mathcal{N}(X; \mu, \sigma)$ and the CPD of A is a softmax given by $P(A = 1 \mid X = x) = 1/(1 + e^{ax+b})$. The clique tree has a single clique (X, A), whose factor should contain the product of these two CPDs. Thus, it should contain two continuous functions — $P(x)P(A = 1 \mid x)$ and $P(x)P(A = 0 \mid x)$ — each of which is a product of a Gaussian and a sigmoid.

To approximate the resulting distribution as a mixture of Gaussians we need to compute the marginal distribution of A and the first two moments of X conditioned on the different values of A:

$$\begin{aligned}
P(A = a) &= \int_{-\infty}^{\infty} P(A = a \mid x)P(x)dx && (7.5)\\
E[X \mid A = a] &= \int_{-\infty}^{\infty} xP(x \mid A = a)dx \\
&= \frac{1}{P(A = a)} \int_{-\infty}^{\infty} xP(A = a \mid x)P(x)dx && (7.6)\\
E[X^2 \mid A = a] &= \int_{-\infty}^{\infty} x^2P(x \mid A = a)dx
\end{aligned}$$

$$= \frac{1}{P(A=a)} \int_{-\infty}^{\infty} x^2 P(A=a \mid x) P(x) dx \tag{7.7}$$

This basic idea leads us to the following outline for an algorithm. We roughly follow Lauritzen's algorithm, diverging only in cases where a clique contains CD CPDs; in this case, we approximate its factor as a mixture of Gaussians, where the mixture has one Gaussian — with the correct first and second moments — for each instantiation of the discrete variables. In the next section, we "fill in" the details of this algorithm, addressing the subtleties that arise.

7.2.3 The Algorithm

Using integrable distributions

The first difficulty in applying our algorithm arises from the observation that Equations 7.5 through 7.7 compute expectations relative to $P(x)$: To evaluate these expressions at a clique, we must have a probability distribution over X in that clique. Unfortunately, the message passing algorithm does not guarantee that these distributions are available initially. Consider the network $X \rightarrow Y \rightarrow A$ where X and Y are continuous nodes and A is discrete. The clique tree for this network consists of two cliques: (X, Y) and (Y, A). In Lauritzen's algorithm, the clique tree is initialized by incorporating all CPDs into their corresponding cliques. In our case, we should incorporate $P(A \mid Y)$ into the clique (Y, A) by computing the relevant expectations. However, at the initialization phase, the message passing has not yet been performed. As such, Y is given in canonical form and does not yet represent a Gaussian distribution, preventing us from performing this integration; thus, we cannot multiply the CPD $P(A \mid Y)$ into the clique at this stage.

We address this problem by introducing a preprocessing phase, which serves to guarantee that all cliques contain an *integrable distribution* — a Gaussian distribution relative to which we can compute the relevant expectations, rather than a non-Gaussian canonical form. To do so, we build the standard clique tree for our BN, but do not initialize the clique potentials. We then insert all the CPDs except for the CD CPDs. The resulting network is equivalent to a CLG, so we can calibrate it using

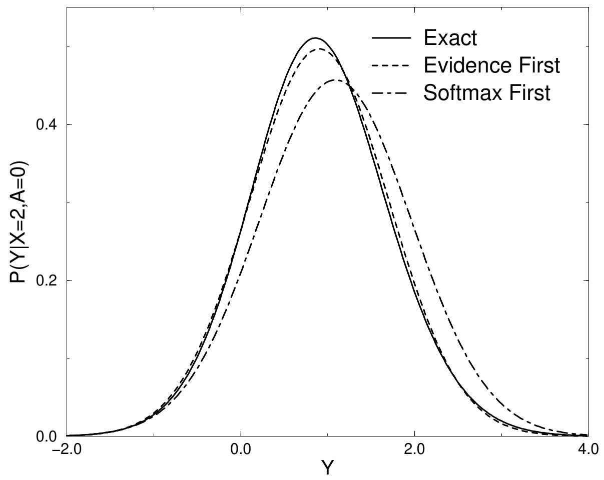


Figure 7.4: Incorporating softmax CPDs before and after the evidence

Lauritzen's algorithm, resulting in probability distributions in each clique. Finally, we insert the remaining CD CPDs and re-calibrate the tree. Note that the cliques in our tree were designed to accomodate this insertion operation. Since we now have integrable distributions, we can perform the approximation.

In our example, we will first calibrate the clique tree without the CPD for A, obtaining a distribution over Y. We can then introduce A's CPD into the clique, using the distribution over Y as our integration distribution to compute the necessary expectations. Thus, we ensure that we have a well defined distribution in every clique in the tree.

This approach raises the following question: Can we always use the distribution in some clique in the tree as our integration distribution? Unfortunately, there are several reasons why the use of this distribution is an approximation which can lead to errors. We now discuss each of these, and show how to correct them.

Conditioning on the evidence

In Lauritzen's algorithm it is possible to condition the distribution on the evidence during any part of the algorithm. However, once we need to deal with non-linear CPDs this is no longer true. Consider, for example, the network $X \to Y \to A$, and assume that X is observed. The minimal cliques are (X, Y) and (Y, A). Following our current algorithm, we would first enter the CPDs $P(X)$ and $P(Y \mid X)$ and the calibrate the tree. We would then insert the CPD $P(A \mid Y)$ and calibrate the tree, approximating the distribution as a mixture of Gaussians. If we now enter the observed evidence X, we would be incorporating it into an approximate distribution rather than the true one, potentially leading to a sub-optimal approximation. Figure 7.4 shows an example of this phenomenon, where the approximation obtained by first integrating the CD CPD and then conditioning on the evidence is a sub-optimal approximation. The optimal approximation uses the posterior over Y directly as our integration distribution.

Our solution to this problem is straightforward: We not only ensure that each clique has a Gaussian distribution in it, we ensure that it has the Gaussian distribution that accounts for the evidence. Thus, we incorporate the evidence and propagate it before entering the CD CPDs. We note that only evidence about continuous variables can cause difficulties with our approximation: Evidence over discrete variables merely changes the probabilities of the mixture components and can thus be incorporated either before or after the integration.

Restricting the integration cliques

A more subtle problem with our choice of integration distribution relates to the use of collapsing within Lauritzen's algorithm. Consider a network $A \to X \to Y \to B$, and assume that the clique tree has the cliques (X, Y, B) and (A, X, B) (note that the tree $(A, X), (X, Y), (Y, B)$ is inconsistent with strong triangulation). According to our current algorithm, we calibrate the clique tree with the CPDs for A, X, and Y, and then insert the CPD $P(B \mid Y)$ into the clique (X, Y, B). However, the distribution in this clique is not the correct prior distribution over Y. The correct prior distribution of Y has two modes (one for every value of A); but, as A does not appear in the clique, Lauritzen's algorithm collapses the two modes into a single Gaussian, losing its bimodal nature. Although Lemma 3.14 ensures that this approximation can be used without introducing new errors if the functions are linear, it does not hold for

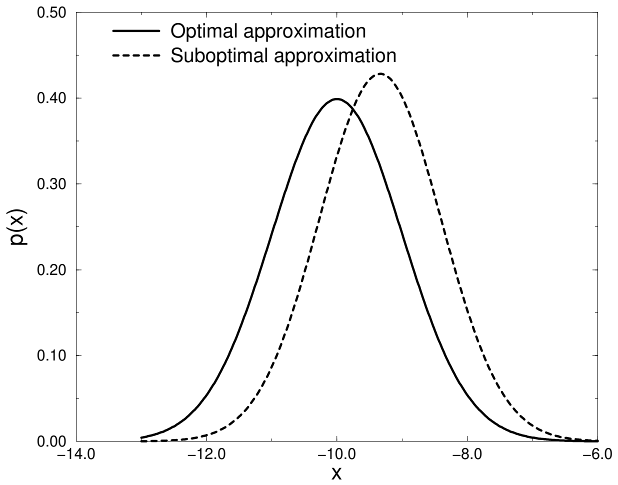


Figure 7.5: Error introduced if the discrete neighbors are not in the same clique as the sigmoid CPDs

non-linear functions. Thus, since the CPD $P(B \mid Y)$ is not linear, we may not get the optimal approximation for the first two moments. Figure 7.5 shows an example, where our approximation is worse when we use the collapsed distribution for Y as the integration distribution.

Once again, the solution is to enter the CD CPD into a clique where the integration distribution over the continuous parents is correct. Let $\boldsymbol{X}$ be the variables of some maximal continuous connected component. Denote by $\mathrm{CD}(\boldsymbol{X}) \subseteq \boldsymbol{X}$ all the variables in $\boldsymbol{X}$ (continuous or discrete) that participate in some CD CPD. As in Definition 3.17, let $\mathrm{DN}(\boldsymbol{X})$ be the discrete neighbors of $\boldsymbol{X}$. As we discussed, $\mathrm{CD}(\boldsymbol{X})$ will have one mode for every assignment to $\mathrm{DN}(\boldsymbol{X})$. Hence, if we want to represent the exact multi-modal distribution for $\mathrm{CD}(\boldsymbol{X})$, it is necessary and sufficient to have a clique containing the variables $\mathrm{CD}(\boldsymbol{X}) \cup \mathrm{DN}(\boldsymbol{X})$. Of course, this requirement could result in a larger clique tree; however, the overhead is not large. As we discussed in Section 3.3.5, $\mathrm{DN}(\boldsymbol{X})$ must already be in some clique in any strongly rooted tree. Thus, at worst, we only add some continuous variables to some of the cliques. Since the representation of canonical forms (and multivariate Gaussians) is quadratic in the number of variables,

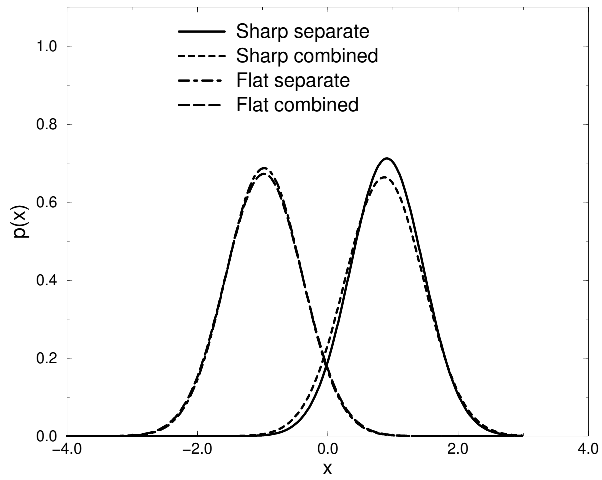


Figure 7.6: Error introduced if the sigmoid CPDs are entered separately

the size of the tree can only grow by a polynomial factor at worst.

Note that this modification to the clique tree is necessary only if we want to guarantee the optimal approximation. The algorithm remains well defined if we use an approximate integration distribution, only the quality of our approximation can degrade. Therefore, we can use a clique tree where the clique that contains $\mathrm{CD}(\boldsymbol{X})$ contains only a subset of the variables in $\mathrm{DN}(\boldsymbol{X})$. Thus, we obtain a spectrum of approximations, with a tradeoff between complexity and accuracy.

Simultaneous insertion of CD CPDs

The final problem arises when there is more than one CD CPD. Most simply, we can insert each CD CPD sequentially. We insert each CD CPD, approximate the resulting joint distribution as a mixture of Gaussians, and proceed to use that mixture as the basis for inserting the next CD CPD. Unfortunately, this approach has a problem: The integration distribution used for the CD CPDs inserted later is only an approximation to the correct non-Gaussian distribution resulting from the insertion of the earlier CD CPDs.

The solution to the problem is to integrate all the CD CPDs in the same continuous connected component in one operation. In Figure 7.6, we show the difference on the network $A \leftarrow X \rightarrow Y \rightarrow B$. We tried inserting both softmax CPDs into the clique containing X and Y together, and separately. We experimented both with step-like transitions ("sharp" sigmoids), and smoother transitions. The latter allows for a better approximation as a Gaussian, and therefore less error by doing the approximation step by step. This difference is clearly manifested in the figure.

While the idea of joint integration seems expensive, we note that the relevant CPDs must already be in the same clique (with their discrete neighbors), so we do not increase the size of the tree. However, we do pay the price of computing integrals in higher dimensions. We can reduce this cost by integrating only some of the CD CPDs together. Again, we get a spectrum of approximations, with a tradeoff between complexity and accuracy: If we want to avoid the high dimensional integration, we can approximate it by inserting the CPDs separately or in small groups. However, it is also possible to use the exact monomials methods discussed in Section 6.2.3, and get efficient integration rules even in high dimensions.

The full algorithm for inference in augmented CLGs is presented in Figure 7.7. We are given a hybrid Bayesian network B, evidence $\boldsymbol{e}$ and a query $\boldsymbol{Q}$ and wish to compute $P(\boldsymbol{Q}, \boldsymbol{e})$. Note that the CD CPDs should be integrated together to achieve the best approximation but can also be integrated separately as discussed above.

7.2.4 In-clique Integration

Having defined the overall structure of the algorithm, it remains only to discuss the integration process within the cliques. Since our integrals still have the form of a product between a Gaussian and some other function we can use the integration techniques discussed in Chapter 6. As always, it is desirable to reduce the dimension of the integrals involved if possible: doing so will speed up the computation or alternatively will let us achieve better accuracy without spending more time on the computation of the integral. Reducing the dimension of the integrals seems to have a special importance in our case, as the cliques in our algorithm often contain many

```
Extending Lauritzen's algorithm to augmented CLGs
   1.  Construct a strongly rooted tree s.t. for all maximal connected
       components X_i, DN(X_i) ∪ CD(X_i) belong to some clique C_i
   2.  Insert all CPDs except for softmax CPDs
   3.  Insert the (continuous and discrete) evidence
   4.  Calibrate the tree using Lauritzen's algorithm
   5.  Instantiate the CD CPDs with the continuous evidence
   6.  For each maximal connected component X_i
      7.  Find all softmax CPDs S_1,...,S_n that can go into C_i
      8.  Insert S_1,...,S_n into C_i using multi-dimensional integration
   9.  Re-calibrate the tree
   10. Return the distribution over Q
```

Figure 7.7: Inference algorithm for augmented CLGs

continuous variables (particularly if we keep a clique which contains an entire continuous connected component). Fortunately, we can use the properties of our augmented CLGs to significantly reduce the computational burden.

Our first observation exploits the fact that we are dealing with Gaussian distributions by using a similar trick to the one discussed in Section 6.2.5. Assume that the continuous variables can be partitioned into the sets $\boldsymbol{Y}$ and $\boldsymbol{Z}$ where only variables from $\boldsymbol{Y}$ appear in CD CPDs. Recall that we can represent the multivariate Gaussian over the variables $\{\boldsymbol{Y}, \boldsymbol{Z}\}$ as a linear Gaussian network with the structure $\boldsymbol{Y} \rightarrow \boldsymbol{Z}$, i.e., there are no edges from a variable in $\boldsymbol{Z}$ to a variable in $\boldsymbol{Y}$. The CD CPDs change the distribution over $\boldsymbol{Y}$, and since every variable in $\boldsymbol{Z}$ depends linearly on the variables in $\boldsymbol{Y}$, Lemma 3.14 ensures that having the first two moments for $\boldsymbol{Y}$ is enough to infer the first two moments for $\boldsymbol{Z}$ without any further numerical integration. Thus, to incorporate the variable A into the CPD, the required integration dimension is exactly the number of continuous parents of A.

Interestingly, we can substantially improve even on this idea, in the case where the CPD of A is a softmax. The softmax for a node A is a soft threshold defined via a set of linear functions f_a over the continuous parents $\boldsymbol{Y}$ of A; we have one function for each value a of A, although we can eliminate one of them as discussed in Section 7.1. Thus, from now on we shall assume that $f_m = 0$ (where $m = |A|$) and we are left with $m-1$ functions $\{f_1,\ldots,f_{m-1}\}$.

We can now define a set of new variables Z_a which are a (deterministic) linear function of the variables $\boldsymbol{Y}$: $Z_a = f_a(\boldsymbol{Y}) = w_0^{(a)} + \sum_l w_l^{(a)} Y_l$. We can then reinterpret the softmax as a CPD whose parents are the variables Z_a. Note that, as the Z_a's are linear functions of the parents $\boldsymbol{Y}$, a Gaussian distribution over $\boldsymbol{Y}$ induces a Gaussian distribution over the Z_a's, and we can therefore use the distribution over the Z_a's as our integration distribution. Since we can represent the variables $\boldsymbol{Y}$ as a linear combination of the Z_a's, we can once again apply Lemma 3.14 and find the correct distribution over $\boldsymbol{Y}$ (up to the first two moments). The dimension of the integrals we have to perform is at most $|A|-1$, where $|A|$ is the number of values of A. Indeed, we have our choice of methods, so we can perform the insertion of A into the clique using an integration dimension which is $\min(|A|-1, |\boldsymbol{Y}|)$, where $|\boldsymbol{Y}|$ is the number of continuous parents of A. When dealing with binary variables, this approach can result in dramatic savings.

More generally, we can use any set of variables $\boldsymbol{Z}$ which are linear combinations of $\boldsymbol{Y}$ such that every $f_a(\boldsymbol{Y})$ can be represented as a linear combination of $\boldsymbol{Z}$, and the integration dimension would then be $|\boldsymbol{Z}|$. In particular we can choose $\boldsymbol{Z} = \boldsymbol{Y}$ or $\boldsymbol{Z} = \{f_1,\ldots,f_{m-1}\}$ (recall that $f_m = 0$) and therefore we can choose $|\boldsymbol{Z}|$ to be $\min(|A|-1, |\boldsymbol{Y}|)$. However, if it is possible to represent some of the functions f_i as linear combinations of the others, i.e., they represent vectors which are linearly dependent, then we may find $\boldsymbol{Z}$ with even a smaller dimension.

As an example, assume that $m = |A| = 4$ and $k = |\boldsymbol{Y}| = 4$. Using the notation of Equation 7.1 we have the following set of parameters (where $\boldsymbol{w}^{(i)} = (w_1^{(i)},\ldots,w_k^{(i)})$)

$$
\begin{array}{ll}
w_0^{(1)} = 3 & \boldsymbol{w}^{(1)} = (2,1,6,2) \\
w_0^{(2)} = 2 & \boldsymbol{w}^{(2)} = (1,-3,4,0) \\
w_0^{(3)} = -2 & \boldsymbol{w}^{(3)} = (2,5,4,-2) \\
w_0^{(4)} = 1 & \boldsymbol{w}^{(4)} = (1,-1,3,-2)
\end{array}
$$

As a first step, we make sure that $f_4 \equiv 0$ by subtracting $w_j^{(4)}$ from all the coefficients, i.e., dividing the exponents by $\exp(w_0^{(4)} + \sum_j w_j^{(4)} Y_j)$. We get:

$$\begin{array}{ll} w_0^{(1)} = 2 & \boldsymbol{w}^{(1)} = (1,2,3,4) \\ w_0^{(2)} = 1 & \boldsymbol{w}^{(2)} = (0,-2,1,2) \\ w_0^{(3)} = -3 & \boldsymbol{w}^{(3)} = (1,6,1,0) \\ w_0^{(4)} = 0 & \boldsymbol{w}^{(4)} = (0,0,0,0) \end{array}$$

Note that this set of parameters represent exactly the same softmax distribution as the original set. In principle we can now define $\boldsymbol{Z} = \{Z_1, Z_2, Z_3\}$ for the 3 non-zero functions. However, we can use the fact that $\boldsymbol{w}^{(3)} = \boldsymbol{w}^{(1)} - 2\boldsymbol{w}^{(2)}$ and define $\boldsymbol{Z} = \{Z_1, Z_2\}$, with the third function being defined as $Z_1 - 2Z_2$.

Thus, define $Z_1 = Y_1 + 2Y_2 + 3Y_3 + 4Y_4$ and $Z_2 = -2Y_2 + Y_3 + 2Y_4$. Using these variables we can define the softmax CPD with the parents Z_1, Z_2 and a new set of parameters that still represents the same distribution:

$$\begin{array}{ll} w_0^{(1)} = 2 & \boldsymbol{w}^{(1)} = (1,0) \\ w_0^{(2)} = 1 & \boldsymbol{w}^{(2)} = (0,1) \\ w_0^{(3)} = -3 & \boldsymbol{w}^{(3)} = (1,-2) \\ w_0^{(4)} = 0 & \boldsymbol{w}^{(4)} = (0,0) \end{array}$$

For example,

$$P(A = a_3 \mid \boldsymbol{Z}) = \frac{\exp(-3 + Z_1 - 2Z_2)}{\exp(2 + Z_1) + \exp(1 + Z_2) + \exp(-3 + Z_1 - 2Z_2) + 1}$$

The integrals are now defined over Z_1 and Z_2 only. We thus reduced the dimension of the integration from 4 to 2.

Of course, one can still construct networks where the integration dimension is very large: networks where the discrete variable A has many values and continuous parents, or where there are many CD CPDs that all need to be integrated into the same clique. In these cases, we can choose to use the exact monomials method with a relatively low precision rule.

7.2.5 Using Conditional Forms

As discussed in Section 3.2.3, canonical forms suffer from some well known numerical instabilities which our extension to Lauritzen's algorithm would inherit. Lauritzen and Jensen's modified algorithm [LJ01] improves upon the original one by using conditional forms, which are more numerically stable and can incorporate deterministic CPDs.

To enjoy the benefit of numerical stability, we can adapt our algorithm to work with conditional forms. In this version of Lauritzen's algorithm, the clique potentials are conditional Gaussians except for the strong root, where the potential is a multivariate Gaussian. All we need to do is to ensure that CD CPDs are inserted in a clique which is a strong root, guaranteeing it has an integrable distribution. But using the algorithm in Figure 7.7 we already have this property, as it is forced in line 1.

Note that even if we have a tree in which the strong root does not contain the continuous variables from the CD CPDs (in which case the algorithm is still well defined for canonical forms, although it is not guaranteed to give the correct first two moments), we can change the clique tree using PUSH operations [LJ01]. These operations "push" continuous variables into the strong root, and thus can be used to ensure that the property in line 1 of the algorithm holds. The only possible consequence is the addition of continuous variables to some cliques. Therefore, the added complexity in the worst case is polynomial in the number of continuous variables.

7.2.6 Analysis

We now show that our algorithm is "exact", up to errors caused by numerical integration. We use "exact" in the same sense used in Lauritzen's algorithm: It computes the correct distribution over the discrete nodes, and the correct first and second moments for the continuous ones.

Theorem 7.1 *Let $\mathcal{B}$ be some augmented CLG. Let $\boldsymbol{Q}$ be a query such that $\boldsymbol{Q} \subseteq \boldsymbol{C}$ where $\boldsymbol{C}$ is some clique in the tree and let $\boldsymbol{e}$ be some evidence. The algorithm in Figure 7.7 computes a distribution $P(\boldsymbol{Q}, \boldsymbol{e})$ which is exact for discrete variables* *in* $\boldsymbol{Q}$ *and has the correct first two moments for continuous variables in* $\boldsymbol{Q}$*, up to inaccuracies caused by numerical integration.*

Proof: We start by showing that the algorithm is exact when the clique tree contains just one clique. The first step involves inserting discrete and (conditional) linear CPDs into the clique tree. Since all the variables are in one clique and there is no need for collapsing, we get that the clique potential represents the exact product of the CPDs that were inserted. Note that the product of the clique potential and the CD CPDs is the exact prior distribution.

The next step involves incorporating the evidence. As we saw in Section 2.3, the discrete evidence can be viewed as extra factors that are multiplied into the tree (the entry in the factor corresponding to the evidence is 1 and the rest are 0). Multiplying these factors into the tree is an exact operation, and does not introduce any inaccuracies. We now consider the continuous evidence. Setting the evidence is equivalent to setting the relevant values in every function of the product of the tree and the CD CPDs. In the clique potential, we simply condition our canonical factors or our conditional factors on the evidence. In the CD CPDs we are guaranteed that any continuous evidence variables that appear in the CPD must be parents; Thus, we can simply set their value and get a new conditional distribution not involving them, which is either a CD CPD or just a discrete factor.

The last step is the insertion of the CD CPDs into the clique potential. Had we been able to represent the answer exactly, then the clique potential would have been the exact posterior distribution. In practice, the true posterior distribution is a mixture of non Gaussians, and we approximate each of the mixture components as a Gaussian. Thus, the third step produces the correct first two moments, up to inaccuracies caused by numerical integration.

We now remove the assumption of a single clique in our tree by assuming that we have some general strongly rooted tree. Let C be a clique as defined in line 1 of the algorithm in Figure 7.7. Consider the network $\mathcal{B}'$ that we get from $\mathcal{B}$ by removing all the CD CPDs. $\mathcal{B}'$ is a CLG and our clique is a strong root for $\mathcal{B}'$. Note that C contains all the discrete neighbors of a maximal connected components, and therefore can represent the exact distribution for its continuous variables without collapsing. From that, and from Theorem 3.13, we know that after line 4 of the algorithm the distribution in C represents the correct posterior distribution for $\mathcal{B}'$ over the variables in C. Thus, similarly to the single node case, when we multiply C by the CD CPDs using numerical integration, we get the correct distribution over the discrete variables and the correct first two moments for the continuous variables, up to inaccuracies caused by numerical integration.

We now perform a downward pass in the tree, i.e., we send messages from C outward. The proof that we get the correct first two moments in the downward pass (up to numerical integration inaccuracies) in some arbitrary clique V is similar to the proof in Theorem 3.13. Again, we use induction on the length of the path between C and V. For C itself we already know that we have the correct marginal. For some other node V, we define W to be the first node on the path from V to C and define S to be the sepset between V and W. The induction hypothesis is that the canonical factor in W represents the correct discrete distribution and the correct first two moments (up to numerical integration errors).

Using the same idea as the proof of Theorem 3.13, we first use Equation 3.12 to show that variables in V that belong to S have the same (weak) marginal as they have in W. Since these variables have the correct weak marginal in W up to numerical integration errors, the same also holds in V. Thus, it is left to prove the claim for variables in $V - S$. Because of strong triangulation, all the variables in $V - S$ must be continuous (unless all the variables in S are discrete, in which case the message passing operation is exact and the claim becomes trivial). We now observe that because of the way the tree was built in line 1 of Figure 7.7, all the the continuous variables in V that appear in some CD CPD must be in the clique C. Thus, none of the variables in $V - S$ can appear in a CD CPD, and therefore the variables in $V - S$ must have a linear dependency on the variables in S. Since the dependency is linear, Lemma 3.14 applies and therefore we have the correct first two moments for all the variables in V (up to numerical integration inaccuracies). ∎

We point out that sometimes the numerical integration errors can be magnified by unlikely discrete evidence. Recall that in order to get the conditional distribution we must renormalize the discrete weights to sum up to 1, and we do so by dividing by the

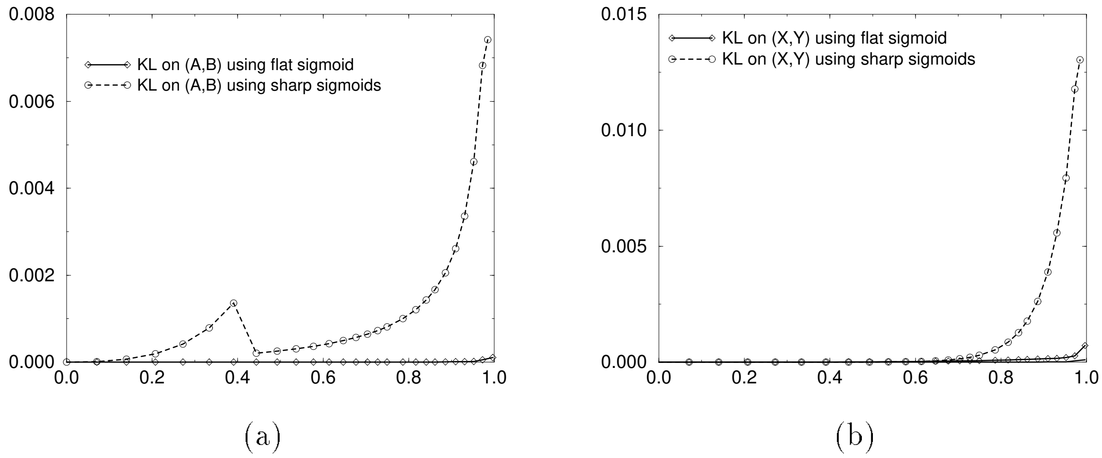


Figure 7.8: Experimental results: error caused by inserting CD CPDs separately (a) KL-divergence for discrete variables (b) KL-divergence for continuous variables

likelihood of the evidence. If we divide by a small number, even slight errors in our approximation can become significant. It is possible to reduce the effect of unlikely evidence by a two-step process: We run our algorithm to obtain a first estimate of the posterior over the discrete variables and then rerun it, using more precise (and more expensive) integration rules for mixture components that are more likely in the posterior distribution.

7.2.7 Experimental Results

Perhaps the most inconvenient part about our extension to Lauritzen's algorithm is the need to perform the integration of CD CPDs simultaneously in order to get the optimal approximation. It is therefore interesting to examine how important it to insert the CD CPDs together versus one at a time.

We experimented with the network $A \to X \to Y \to B$ used in Figure 7.6. In Figure 7.8 we show the KL-divergence, for both the discrete and the continuous variables, between the distributions obtained by inserting the softmax CPDs together versus one at a time. The error depends on the strength of the correlation between X and Y, so we considered different correlation values ranging from 0 to 1. We also

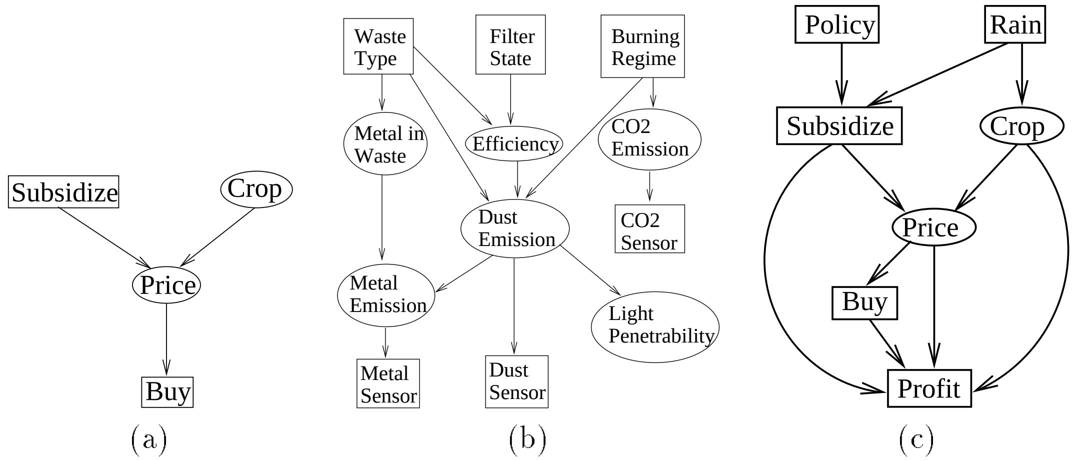


Figure 7.9: Augmented CLGs (a) The original Crop network (b) The extended Emission network (c) The extended Crop network

changed the parameters of the softmax CPDs, allowing for smooth transitions in one case (a flat sigmoid) and step-like transitions in another case. We see that the error is larger when there is a strong correlation between variables influenced by different CD CPDs, and when the sigmoids are sharp, making the Gaussian approximation worse. Interestingly, however, the accuracy is very high in all cases; Thus, even if we insert CD CPDs separately, with the associated computational benefits, we often get very accurate results.

We compared the performance of our algorithm to that of others on examples that have been considered by other researchers. We first tested our algorithm on the Crop network [BKRK97] shown in Figure 7.9(a). We compared our results with the results reported by Murphy [Mur99] and with running Gibbs sampling using BUGS [GTS94]. It turns out that Murphy's variational algorithm performs quite poorly when the posterior distribution is multi-modal, achieving L1-errors over the binary discrete variables of 0.28–0.38. On the other hand, both our algorithm and BUGS performed very well on this simple network, giving the correct result almost instantaneously. We also note that Wiegerinck [Wie00] reports good results for this network using his variational approach.

To test the algorithm on a larger network, we used the *Emission* network [Lau92], which models the emission of heavy metals from a waste incinerator. The original network is a CLG. We augmented it with three extra discrete binary variables as shown in Figure 7.9(b). The additional variables correspond to various emission sensors and each has a CD CPD: a dust sensor ($w_0 = -3, w_1 = 1$), a CO_2 sensor ($w_0 = 6, w_1 = 3$) and a metal emission sensor ($w_0 = -5.6, w_1 = 2$).

We experimented with various queries using our algorithm and compared it to a few runs of Gibbs sampling using BUGS. We found that BUGS seemed unstable and produced results which differed significantly from ours. As an example, we queried for the distribution over the emission of dust after setting both the metal emission sensor and the CO_2 sensor to *High*. Our algorithm returned a mean of 3.419 and a variance of 1.007. BUGS converged after about 500,000 samples to a mean of 3.31 and a variance of 0.31, which did not change substantially even after 1,000,000 samples.

To understand the discrepancy, we used likelihood weighting on the same query. After 500,000 samples, the estimated mean was 3.418 and the estimated variance was 0.999 which agree quite closely with the results produced by our algorithm.[2] For our algorithm, we were able to converge using Gaussian Quadrature as a numerical integration technique with only 3 points per dimension, and with the highest integration dimension being 2. Hence, our algorithm was almost instantaneous, and was much faster than both BUGS and likelihood weighting.

As a final example, we tried our algorithm with a network containing non-softmax CD CPDs. We augmented the crop network with three more variables, as shown in Figure 7.9(c). One of them is the *Profit* variable, which depends on a product of the *Crop* and *Price* variables. The parameters of the extended network appear in Table 7.1, where

$$\begin{aligned} P(\mathit{Profit}{=}\mathit{Negative}) &= \frac{f_l}{f_l + 1 + f_p} \\ P(\mathit{Profit}{=}\mathit{Even}) &= \frac{1}{f_l + 1 + f_p} \end{aligned}$$

[2]We further investigated this discrepancy and discovered that BUGS also returns answers that disagree with those appearing in [Lau92] even for the original *Emission* network without the CD CPDs. For example the standard deviation for *DustEmission* converged to 0.85 instead of 0.77 (the mean was correct).

Node	Distribution		
Policy			(0.5, 0.5)
Rain			(0.35,0.6,0.05)
Subsidize	Drought	Liberal	(0.4, 0.6)
	Drought	Conservative	(0.3, 0.7)
	Average	Liberal	(0.95, 0.05)
	Average	Conservative	(0.95, 0.05)
	Floods	Liberal	(0.5, 0.5)
	Floods	Conservative	(0.2, 0.8)
Crop	Drought		$\mathcal{N}(3, 0.5)$
	Average		$\mathcal{N}(5, 1)$
	Floods		$\mathcal{N}(2, 0.25)$
Price	Yes		$\mathcal{N}(9 - C, 1)$
	No		$\mathcal{N}(12 - C, 1)$
Buy			$w_0 = -1, w_1 = 7$
Profit	Sub=Yes	Buy=Yes	f_l=exp(13-2P-PC)
			f_p=exp(3P+PC-23)
	Sub=Yes	Buy=No	f_l=exp(13-2P)
			f_p=exp(3P-23)
	Sub=No	Buy=Yes	f_l=exp(13-PC)
			f_p=exp(PC-23)
	Sub=No	Buy=No	f_l=exp(13)
			f_p=exp(-23)

Table 7.1: Parameters for the augmented Crop network

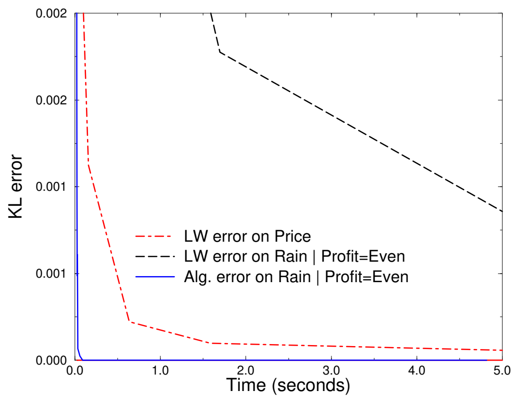


Figure 7.10: Experimental results: Comparison with Likelihood Weighting

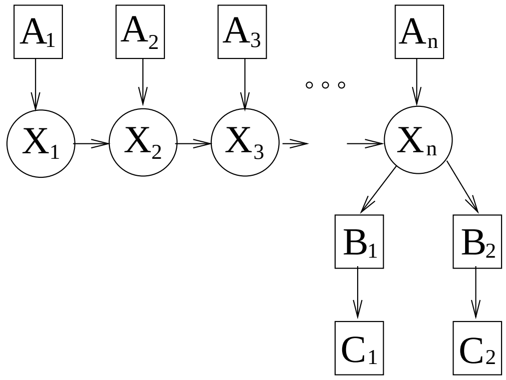
$$P(\mathit{Profit}{=}\mathit{Positive}) = \frac{f_p}{f_l + 1 + f_p}$$

Having experienced problems using BUGS, we compared our results to likelihood weighting. We tested two scenarios, one without evidence, and one with the evidence *Profit=Even*, and compared the accuracy of both algorithms on various queries. We used numerical integration with 150 points per dimension as our ground truth. We ran LW and our algorithm for the same amount of time, and then measured the KL-divergence between the "ground truth" and the results. For LW, we averaged over 10–500 runs (for short runs, where the number of samples was smaller and the variance larger, we averaged our results over a larger number of runs). Figure 7.10 shows the results for the KL-error for *Price* with no evidence and for *Rain* given *Profit=Even*. Only three lines are visible: the KL-error of our algorithm in the case of no evidence is too close to zero to be visible in the graph. It is clear that our algorithm converges much faster than LW, especially when we have evidence. This is to be expected — as we discussed, sampling methods can take a lot of time to converge, and the performance of LW deteriorates when there is evidence.

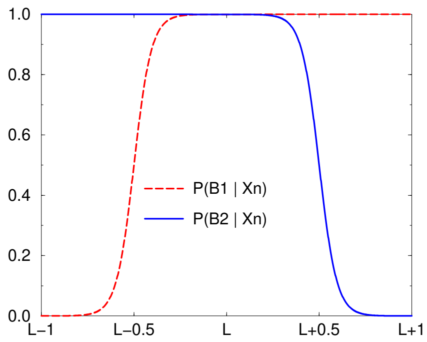
7.3 Enumeration Approach

The main problem with the approach of generalizing Lauritzen's algorithm to augmented CLGs is that, as we saw in Section 3.3.5, Lauritzen's algorithm does not scale up well, even in the case of regular CLGs. Recall from Chapter 5 that we dealt with this problem by using the enumeration algorithm, and we saw in Section 6.5 how to extend this algorithm to non-linear CPDs involving continuous variables. One could hope that we could use similar ideas for augmented CLGs as well — after all the resulting distribution of augmented CLGs is again a mixture distribution where we can approximate every mixture component as a Gaussian.

Unfortunately, enumerating the discrete hypotheses by their prior is a much harder task in augmented CLGs than in CLGs. In fact, finding just the most likely discrete instantiation in polytree augmented CLGs is already NP-hard. Intuitively, the reason is that once some of the discrete variables depend on continuous variables we can no



(a)



(b)

Figure 7.11: (a) Augmented CLG used in the reduction for Theorem 7.2 (b) Distributions of $P(B_1|X_n)$ and $P(B_2|X_n)$ in the reduction

longer ignore the continuous parts of the network for the purposes of enumeration by prior, and we have to pay the price in terms of significant added complexity.

Theorem 7.2 *The following problem is NP-hard:*

- **Instance:** *A polytree augmented CLG $\mathcal{T}$ with binary discrete variables.*
- **Question:** *What is the most likely assignment of the discrete variables in $\mathcal{T}$?*

Proof: The reduction is similar to the one used in Theorem 4.1. Again, we show a reduction from the subset sum problem where we have a set of non-negative integers $S = \{s_1, s_2, ..., s_n\}$, and another positive integer L and we are asked whether there exists a subset $S' \subseteq S$ such that $\sum_{s_i \in S'} = L$.

We reduce the problem to an augmented CLG polytree with the structure shown in Figure 7.11(a). Before going into the details, we explain the general idea of the proof. The variables A_i and X_i have the same distribution as in Theorem 4.1. The variables A_i represent a "selector" function that determines whether $s_i \in S'$ or not. Every assignment of the A_i variables induces a Gaussian for X_n and therefore X_n is a mixture of Gaussians with a mode at L iff there exists an S' such that $\sum_{s_i \in S'} = L$.

We use the variables B_1 and B_2 as noisy indicators for the value of X_n: B_1 is a noisy indicator for $X_n > L - 0.5$ and B_2 is a noisy indicator for $X_n < L + 0.5$. Note that $B_1 = B_2 = 1$ is likely only when $X_n = L$, and thus $B_1 = B_2 = 1$ will be a part of a likely assignment iff there exists a solution to the subset sum problem. The nodes C_1 and C_2 will be used to make sure that if there is a solution to the subset sum problem, then the assignment with $B_1 = B_2 = 1$ is not only a likely assignment, but the most likely one. We will therefore show that there exists an S' that is a solution to the subset sum problem iff the most likely discrete assignment in the augmented CLG has $B_1 = B_2 = 1$.

More formally, we define the CPDs of the variables. The discrete variables are all binary. The variables A_i have a uniform distribution, i.e., $P(A_i = 1) = \frac{1}{2}$. For the continuous variables:

$$P(X_1 \mid A_1) = \begin{cases} \mathcal{N}(0, \sigma^2) & A_1 = 0 \\ \mathcal{N}(s_1, \sigma^2) & A_1 = 1 \end{cases}$$

where $\sigma = \frac{1}{c\sqrt{n}}$. For $i = 2, 3, ..., n$:

$$P(X_i \mid A_1, X_{i-1}) = \begin{cases} \mathcal{N}(X_{i-1}, \sigma^2) & A_i = 0 \\ \mathcal{N}(X_{i-1} + s_i, \sigma^2) & A_i = 1 \end{cases}$$

The CPDs for B_1 and B_2 are logistic CPDs shown in Figure 7.11(b). Note again that B_1 and B_2 are likely to both be 1 only when $X_n = L$.

$$\begin{aligned} P(B_1 = 1 | X_n = x_n) &= \frac{1}{1 + \exp\left(-\alpha\left(x_n - L + \frac{1}{2}\right)\right)} \\ P(B_2 = 1 | X_n = x_n) &= \frac{1}{1 + \exp\left(\alpha\left(x_n - L - \frac{1}{2}\right)\right)} \end{aligned}$$

Finally, we have the CPDs for C_i where $i = 1, 2$:

$$\begin{aligned} P(C_i = 1 | B_i = 0) &= \frac{1}{2} \\ P(C_i = 1 | B_i = 1) &= 1 \end{aligned}$$

We will set the values of our constants as $c = 12$ and $\alpha = 24$ — these values will become clear later in the proof. For now we just assume that both c and α are positive.

First, let us assume that there exists an S' which is a solution to the subset sum problem. We must show that $B_1 = B_2 = 1$ is a part of the most likely discrete assignment. Let $\boldsymbol{a'} = \langle a_1, a_2, ..., a_n \rangle$ be some assignment to $\langle A_1, A_2, ..., A_n \rangle$ corresponding to S', i.e., $a_i = 1$ iff $s_i \in S'$. Note that for similar arguments to the ones used in Theorem 4.1

$$P(X_n|\boldsymbol{a'}) = \mathcal{N}\left(X_n; L, n\sigma^2\right) = \mathcal{N}\left(X_n; L, \frac{1}{c^2}\right) \tag{7.8}$$

Consider the assignment $\langle \boldsymbol{a'}, B_1 = 1, B_2 = 1, C_1 = 1, C_2 = 1 \rangle$, which we will denote as $\langle \boldsymbol{a'}, b_1, b_2, c_1, c_2 \rangle$. We have:

$$\begin{aligned}
P(\boldsymbol{a'}, b_1, b_2, c_1, c_2) &= P(\boldsymbol{a'})P(b_1, b_2 \mid \boldsymbol{a'})P(c_1 \mid \boldsymbol{a'}, b_1, b_2)P(c_2 \mid \boldsymbol{a'}, b_1, b_2, c_1) \\
&= 2^{-n} P(b_1, b_2 \mid \boldsymbol{a'}) \cdot 1 \cdot 1
\end{aligned}$$

Thus, we need to find $P(b_1, b_2 \mid \boldsymbol{a'})$. Note that B_1 and B_2 are independent given x_n, and therefore:

$$\begin{aligned}
P(b_1, b_2 \mid \boldsymbol{a'}) &= \int_{-\infty}^{\infty} P(b_1, b_2 \mid \boldsymbol{a'}, x_n) P(x_n \mid \boldsymbol{a'}) dx_n \\
&= \int_{-\infty}^{\infty} P(b_1 \mid x_n) P(b_2 \mid x_n) \mathcal{N}\left(X_n; L, \frac{1}{c^2}\right) dx_n \\
&= \int_{-\infty}^{\infty} \frac{\mathcal{N}\left(x_n; L, \frac{1}{c^2}\right)}{\left\{1 + \exp\left(-\alpha\left(x_n - L + \frac{1}{2}\right)\right)\right\}\left\{1 + \exp\left(\alpha\left(x_n - L - \frac{1}{2}\right)\right)\right)} dx_n \\
&\geq \int_{L-\frac{1}{4}}^{L+\frac{1}{4}} \frac{\mathcal{N}\left(x_n; L, \frac{1}{c^2}\right)}{\left\{1 + \exp\left(\alpha\left(L - x_n - \frac{1}{2}\right)\right)\right\}\left\{1 + \exp\left(\alpha\left(x_n - L - \frac{1}{2}\right)\right)\right\}} dx_n \\
&\geq \left(\frac{1}{1 + \exp(-\alpha/4)}\right)^2 \int_{L-\frac{1}{4}}^{L+\frac{1}{4}} \mathcal{N}\left(x_n; L, \frac{1}{c^2}\right) dx_n \\
&= \left(\frac{1}{1 + \exp(-\alpha/4)}\right)^2 \left(\Phi\left(\frac{L + \frac{1}{4} - L}{\frac{1}{c}}\right) - \Phi\left(\frac{L - \frac{1}{4} - L}{\frac{1}{c}}\right)\right) \\
&= \left(\frac{1}{1 + \exp(-\alpha/4)}\right)^2 \left(\Phi(c/4) - \Phi(-c/4)\right)
\end{aligned}$$

where we use the notation $\Phi(t) = \int_{-\infty}^{t} \mathcal{N}(x; 0, 1)\, dx$. For our choices of $\alpha = 24$ and $c = 12$ we get that $P(b_1, b_2 \mid \boldsymbol{a}') = 0.9951 \cdot 0.9974 = 0.9925$, and therefore

$$P(\boldsymbol{a}', b_1, b_2, c_1, c_2) > 0.99 \cdot 2^{-n}$$

Consider any other assignment, where $B_1 = 0$ or $B_2 = 0$. For any such assignment we must have $P(C_1 \mid B_1 = 0) = \frac{1}{2}$ or $P(C_2 \mid B_2 = 0) = \frac{1}{2}$, and therefore for the likelihood of such an assignment we have:

$$P(A_1, \ldots, A_n, B_1, B_2, C_1, C_2) \leq P(A_1, \ldots, A_n) P(C_1 \mid B_1) P(C_2 \mid B_2) \leq \frac{1}{2} \cdot 2^{-n}$$

Since $0.99 \cdot 2^{-n} > \frac{1}{2} \cdot 2^{-n}$, it must be the case that the most likely assignment has $B_1 = B_2 = 1$.

Conversly, we must show that if there does not exist a solution to the subset sum problem then the most likely assignment has $B_1 = 0$ or $B_2 = 0$. Consider some assignment $\langle \boldsymbol{a}, b_1, b_2, C_1, C_2 \rangle$ involving $B_1 = B_2 = 1$ where $\boldsymbol{a} = \langle a_1, \ldots, a_n \rangle$. Let $M = S(\boldsymbol{a}) = \sum_i a_i s_i \neq L$. Assume first that $M < L$. Then:

$$P(\boldsymbol{a}, b_1, b_2, C_1, C_2) \leq P(\boldsymbol{a}) P(b_1 \mid \boldsymbol{a}) = 2^{-n} P(b_1 \mid \boldsymbol{a})$$

Using the fact that $M - L \leq -1$ we get:

$$\begin{aligned}
P(b_1 \mid \boldsymbol{a}) &= \int_{-\infty}^{\infty} P(b_1 \mid x_n) P(x_n \mid \boldsymbol{a}) dx_n \\
&= \int_{-\infty}^{M+\frac{1}{4}} \frac{1}{1 + \exp\left(-\alpha\left(x_n - L + \frac{1}{2}\right)\right)} P(x_n \mid \boldsymbol{a}) dx_n + \\
&\quad \int_{M+\frac{1}{4}}^{\infty} P(b_1 \mid x_n) \mathcal{N}\left(x_n; M, \frac{1}{c^2}\right) dx_n \\
&\leq \int_{-\infty}^{M+\frac{1}{4}} \frac{1}{1 + \exp\left(-\alpha\left(M - L + \frac{3}{4}\right)\right)} P(x_n \mid \boldsymbol{a}) dx_n + \\
&\quad \int_{M+\frac{1}{4}}^{\infty} \mathcal{N}\left(x_n; M, \frac{1}{c^2}\right) dx_n \\
&\leq \frac{1}{1 + \exp(\alpha/4)} \int_{-\infty}^{M+\frac{1}{4}} P(x_n \mid \boldsymbol{a}) dx_n + \int_{M+\frac{1}{4}}^{\infty} \mathcal{N}\left(x_n; M, \frac{1}{c^2}\right) dx_n
\end{aligned}$$

$$\leq \frac{1}{1+\exp(\alpha/4)} + \Phi(-c/4)$$

Again, using our choices for α and c we get that the probability of any assignment for which $S(\boldsymbol{a}) < L$ and $B_1 = 1$ is at most $0.0038 \cdot 2^{-n}$. To realize this is not the most likely assignment we only need to remember that no matter what assignment $\boldsymbol{a}$ we choose there are 16 combinations of B_1, B_2, C_1, C_2 and since $\sum P(B_1, B_2, C_1, C_2 \mid \boldsymbol{a}) = 1$ there must be at least one combination of $B_1 = b_1', B_2 = b_2', C_1 = c_1', C_2 = c_2'$ for which $P(b_1', b_2', c_1', c_2' \mid \boldsymbol{a}) \geq \frac{1}{16}$. Since for this combination $P(\boldsymbol{a}, b_1', b_2', c_1', c_2') \geq \frac{1}{16} \cdot 2^{-n} > 0.0038 \cdot 2^{-n}$ we get that the most likely assignment does not have $B_1 = B_2 = 1$.

The case for $M > L$ is similar where we use B_2 instead of B_1. ∎

Thus, enumeration by prior can be hard even for simple augmented CLGs, and in general we cannot use it as a part of an efficient approximate inference algorithm. However, this does not imply that one cannot use some different version of the enumeration algorithm for augmented CLGs. After all, enumeration by prior is nothing more than a heuristic that tries to achieve an enumeration order that is close to the order induced by the posterior distribution. It is possible to come up with other heuristics that also try to approximate the order induced by the posterior distribution, but are tractable even for augmented CLGs. We leave the problem of finding such tractable heuristics as an open issue which we do not address in this thesis.

Chapter 8

Dynamic Bayesian Networks

So far, we have considered only static environments, i.e., environments that do not evolve over time. However to perform many tasks, including fault diagnosis, we need to model stochastic processes, where the various variables have different values over time. In this chapter we discuss how to extend Bayesian networks to the dynamic case and some of the important inference algorithms. Sections 8.1, 8.2, 8.3, 8.5 and 8.7, are based on a draft of the textbook of Koller & Friedman.

8.1 Modeling Stochastic Processes

Given some stochastic process, it is convenient to view its state in terms of a snapshot, which encodes the process state at some given instant. Each state is represented as usual: an assignment of values to some set of random variables. We denote the random variables that represent the state at time t as $X_1^{(t)}, \ldots, X_n^{(t)}$ (and accordingly $\boldsymbol{X}^{(t)}$ represents the entire state at time t).

Our first representational simplification is to model time as proceeding in discrete regularly-scheduled *time slices*. That is, instead of viewing time as continuous, we assume that t can only take on integer values; thus, $\boldsymbol{X}^{(0)}$ is the state at the initial time slice, $\boldsymbol{X}^{(1)}$ is the state after a single time increment, and so on. This assumption is often consistent with the way sensors are used in practice, where the value of the sensor is sampled at some constant frequency — this frequency is a natural candidate to represent the discrete time units in our model.

Using this assumption, we can represent the process using a probability distribution over $\boldsymbol{X}^{(0)}, \boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \ldots$. This can be done by directly representing the distribution $P(\boldsymbol{X}^{(0)})$ and for $t > 0$ representing the conditional distribution $P(\boldsymbol{X}^{(t)} \mid \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(t-1)})$. Note that this representation encodes the fact that time moves forward, as the state at time t depends only on the history of the system before time t.

We now make a second simplifying assumption. Instead of letting the $\boldsymbol{X}^{(t)}$ depend on the entire history of the system from time step 0 all the way to time step $t-1$ we assume that the dependency goes back only k time steps into the past, i.e., we assume

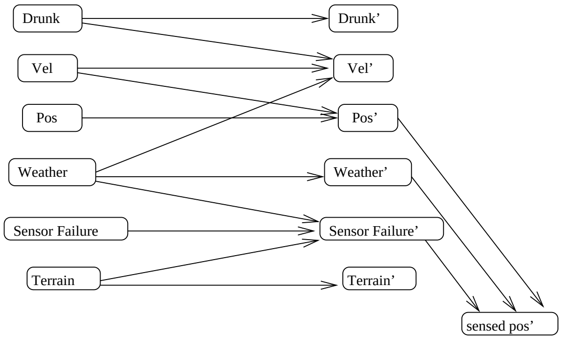
$$P(\boldsymbol{X}^{(t)} \mid \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(t-1)}) = P(\boldsymbol{X}^{(t)} \mid \boldsymbol{X}^{(t-k)}, \ldots, \boldsymbol{X}^{(t-1)})$$

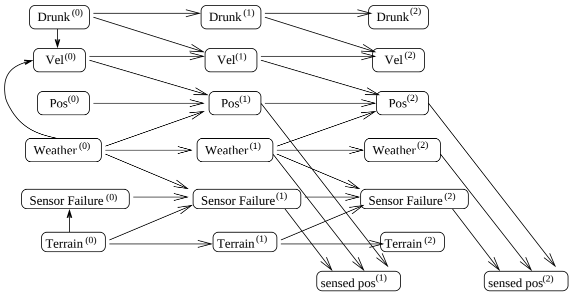
In this thesis, we will almost always use the strongest form of this assumption in which $k = 1$. This special case is called the *Markov* assumption.

Definition 8.1 *We say that a dynamic system over the variables* $\boldsymbol{X}$ *satisfies the Markov assumption if, for all* $t \geq 0$,

$$P(\boldsymbol{X}^{(t)} \mid \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(t-1)}) = P(\boldsymbol{X}^{(t)} \mid \boldsymbol{X}^{(t-1)})$$

Another way to state the Markov assumption is that $\boldsymbol{X}^{(t+1)}$ is independent of $\boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(t-1)}$ given $\boldsymbol{X}^{(t)}$, i.e., the future is independent of the past given the present. Like any conditional independence assumption, the Markov assumption may be reasonable or not. For example, assume that we are tracking a moving object and our only variable is its location. The location at time $t+1$ is not independent of the history given the location at time t because we need to know in which direction the object is moving and at what velocity. We can, however, make the Markov assumption more realistic by adding these variables to our model. We can make our model even more accurate by also adding variables that represent the acceleration of the object. In general we can transform any model in which $P(\boldsymbol{X}^{(t)})$ depends on $\boldsymbol{X}^{(t-k)}, \ldots, \boldsymbol{X}^{(t-1)}$ for some fixed k into a Markov model by treating the variables $\boldsymbol{X}^{(t-k)}, \ldots, \boldsymbol{X}^{(t-2)}$ as a part of the system state, i.e., the system state has a "memory"





(a) (b)

Figure 8.1: (a) A highly simplified 2-TBN for monitoring a vehicle. (b) The unrolled DBN for three time slices

of $k-1$ time slices into the past.

Since the process is unbounded, we still need to represent infinitely many conditional probabilities $P(\boldsymbol{X}^{(t)} \mid \boldsymbol{X}^{(t-1)})$. Therefore, we usually make one last simplifying assumption:

Definition 8.2 *We say that a Markovian process is* time-invariant *(or* homogeneous *or* stationary*) if $P(\boldsymbol{X}^{(t+1)} \mid \boldsymbol{X}^{(t)})$ is the same for all t. In this case, we represent the model using a* transition model $P(\boldsymbol{X}' \mid \boldsymbol{X})$, *where $\boldsymbol{X}$ represents the variables at the current time slice and $\boldsymbol{X}'$ represents the variables at the next one.*

8.2 Definition of Dynamic Bayesian Networks

Under the Markov and the stationarity assumptions, the key to representing stochastic dynamic processes is the representation of the transition model $P(\boldsymbol{X}' \mid \boldsymbol{X})$, which is a conditional probability distribution. We can represent such a model using a *2-time-slice Bayesian network (2-TBN).*

Definition 8.3 *A* 2-time-slice Bayesian network (2-TBN) *for a process whose state variables are $\boldsymbol{X}$ is a Bayesian network* fragment *defined as follows. The graph structure is a directed acyclic graph whose nodes are $\boldsymbol{X} \cup \boldsymbol{X}'$, where nodes in $\boldsymbol{X}$ have no parents. The nodes in $\boldsymbol{X}'$ are annotated with CPDs, $P(X' \mid Par(X'))$. The 2-TBN* *represents a* conditional distribution $P(\boldsymbol{X}' \mid \boldsymbol{X})$ *via the* chain rule for 2-TBNs*:*

$$P(\boldsymbol{X}' \mid \boldsymbol{X}) = \prod_{i=1}^{n} P(X_i' \mid Par(X_i')).$$

Figure 8.1(a) shows a very simple example of a 2-TBN for a vehicle monitoring application. The 2-TBN models the behavior of the vehicle, the environment and an on-board camera that tries to estimate the vehicle's position (in order to keep it in its lane).

Based on the stationarity property, this model defines the probability distribution $P(\boldsymbol{X}^{(t+1)} \mid \boldsymbol{X}^{(t)})$ for any t. If we put in our distribution over the initial states, it defines the distribution over arbitrarily long sequences of time slices resulting in the Dynamic Bayesian network model [DK89]:

Definition 8.4 *A* Dynamic Bayesian Network (DBN) *is a pair* $\langle \mathcal{B}_0, \mathcal{B}_\rightarrow \rangle$*, where* $\mathcal{B}_0$ *is a Bayesian network over* $\boldsymbol{X}^{(0)}$*, representing the initial distribution over states, and* $\mathcal{B}_\rightarrow$ *is a 2-TBN for the process. For any* T*, the distribution over* $\boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(T)}$ *is defined as:*

$$P\left(\boldsymbol{X}^{(0)} = \boldsymbol{x}^{(0)}, \ldots, \boldsymbol{X}^{(T)} = \boldsymbol{x}^{(T)}\right) = P\left(\boldsymbol{x}^{(0)}\right) \cdot \prod_{t=1}^{T} P\left(\boldsymbol{X}' = \boldsymbol{x}^{(t)} \mid \boldsymbol{X} = \boldsymbol{x}^{(t-1)}\right).$$

Given some desired window length, we can easily compose the initial Bayesian network $\mathcal{B}_0$ and the transition network $\mathcal{B}_\rightarrow$ to define a Bayesian network over all the variables within that window. This process is called *unrolling* the DBN. For example, Figure 8.1(b) shows a Bayesian network obtained by unrolling the DBN over three time slices. The edges in the first time slice are the ones present in $\mathcal{B}_0$ (not shown separately). The CPDs for $X_i^{(0)}$ is the CPD of X_i in $\mathcal{B}_0$; the CPD for $X_i^{(t)}$ for $t > 0$ is copied from the CPD of X_i' in $\mathcal{B}_\rightarrow$.

Note that some of the edges go between time slices, whereas others are between nodes in the same time slice. Intuitively, this depends on the temporal aspect of the interaction. If the effect is immediate — much shorter than the difference between time slices — the influence is manifested (roughly) within a time slice. If the effect is longer-term, the influence is manifested from one time slice to the next. In addition some of the arcs between time slices are called *persistence arcs*, representing the tendency of certain variables (e.g., weather) to persist over time with high probability.

Just like a Bayes net, a DBN can encode sophisticated reasoning patterns as long as they are made to be an explicit part of the model. For example, by explicitly encoding sensor failure, we can reach the conclusion that the sensor has failed. Thus, for example, if we get a reading that tells us something unexpected, e.g., the car is suddenly 15 feet to the left of where we thought it was 0.1 seconds ago, then in addition to considering the option that the car has suddenly teleported, we will also consider the option that the sensor has simply failed. Note that the model only considers options that are built into it. If we had no "sensor failure" node, and had the sensor reading depend only on the current location, then there would be no way to explain unlikely sensor readings except by the trajectory.

8.3 The Inference Task

Inference in DBNs can be usually categorized into one of four possible types of queries:

- **Prediction:** Given a probability distribution $P(\boldsymbol{X}^{(t)})$ over the current state, compute the distribution over some set of variables at $t' > t$.
- **Monitoring (tracking):** Given an observation $\boldsymbol{o}^{(t)}$ that comes in at every time slice t, maintain a distribution over the current state given the evidence seen so far. This distribution

$$\varphi^{(t)}\left(\boldsymbol{X}^{(t)}\right) \stackrel{def}{=} P\left(\boldsymbol{X}^{(t)} \mid \boldsymbol{o}^{(1)}, \ldots, \boldsymbol{o}^{(t)}\right)$$

  is called the *belief state* at time t.
- **Probability estimation:** Given a sequence of observations $\boldsymbol{o}^{(1)}, \ldots, \boldsymbol{o}^{(T)}$, what is the distribution over each intermediate state $\boldsymbol{X}^{(t)}$.
- **Explanation:** Given an initial state and an observations sequence $\boldsymbol{o}^{(1)}, \ldots, \boldsymbol{o}^{(t)}$, what is the most likely sequence of states $\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(t)}$ explaining the observations. This is the DBN version of the MPE problem from Section 2.5.1.

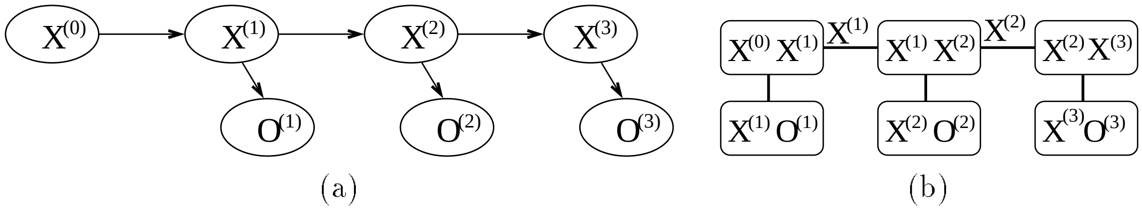


Figure 8.2: (a) Unrolling the DBN where $\boldsymbol{X}$ are all the state variables (b) The resulting clique tree

Prediction, monitoring and probability estimation can all be solved using clique tree inference. We simply unroll the DBN and then perform inference on the resulting Bayes net. For example, if we want to perform monitoring, we can unroll the DBN up to time t, set all the observations $\boldsymbol{o}^{(1)}, \ldots, \boldsymbol{o}^{(t)}$ and then use our inference engine to find the probability distribution over $\boldsymbol{X}^{(t)}$.

The problem with this approach is that it might become very inefficient. Suppose that we monitor the condition of some patient given information from various sensors. The process can run for very long, and thus the unrolled DBN can become very large (in our example, perhaps we monitor the patient at time intervals of 5 seconds and we can do it over days). It seems wasteful and not practical to unroll the DBN and perform the probabilistic inference from scratch for every t. Instead we would like to use our results from time $t-1$ in order to speed up the computations at time t.

To understand the online version of the algorithm it is useful to think of the DBN as having just one state variable and one observation variable (this special case of DBNs is called *Hidden Markov Models (HMMs)* [Rab89]). Figure 8.2(a) shows an example of such a DBN rolled over four time slices, and Figure 8.2(b) shows the resulting clique tree.

It is easy to verify that the monitoring task is equivalent to an upstream pass in the clique tree of Figure 8.2(b) (where upstream is the left to right pass): the message passed from the $\{X^{(t-1)}, X^{(t)}\}$ clique to the $\{X^{(t)}, X^{(t+1)}\}$ clique is the probability of $X^{(t)}$ *and* all of the downstream evidence $o^{(1)}, \ldots, o^{(t)}$. To get the appropriate conditional probability, we simply renormalize the factor.

The key point is that the messages in the clique tree are simply the (unnormalized) belief state $\varphi^{(t)}(X^{(t)})$, leading to a simple on-line algorithm that avoids the full unrolling of the DBN for every time slice. Assume, by induction, that we have already calculated $\varphi^{(t)}$. To compute $\varphi^{(t+1)}$ based on $\varphi^{(t)}$ and the evidence $o^{(t+1)}$, we first propagate the state forward:

$$\begin{aligned}
\tau^{(t+1)}\left(X^{(t+1)}\right) &\stackrel{def}{=} P\left(X^{(t+1)} \mid o^{(1)}, \ldots, o^{(t)}\right) \\
&= \sum_{X^{(t)}} P\left(X^{(t+1)} \mid X^{(t)}, o^{(1)}, \ldots, o^{(t)}\right) P\left(X^{(t)} \mid o^{(1)}, \ldots, o^{(t)}\right) \\
&= \sum_{X^{(t)}} P\left(X^{(t+1)} \mid X^{(t)}\right) \varphi^{(t)}\left(X^{(t)}\right) \qquad (8.1)
\end{aligned}$$

where the simplification in the last step is due to the Markov assumption. Then, we condition on the evidence $o^{(t+1)}$ and using Bayes' law we get:

$$\begin{aligned}
\varphi^{(t+1)}\left(X^{(t+1)}\right) &= P\left(X^{(t+1)} \mid o^{(1)}, \ldots, o^{(t+1)}\right) \\
&= \frac{P\left(o^{(t+1)} \mid X^{(t+1)}, o^{(1)}, \ldots, o^{(t)}\right) P\left(X^{(t+1)} \mid o^{(1)}, \ldots, o^{(t)}\right)}{P\left(o^{(1)}, \ldots, o^{(t)}\right)} \\
&= \frac{P\left(o^{(t+1)} \mid X^{(t+1)}\right) \tau^{(t+1)}\left(X^{(t+1)}\right)}{P\left(o^{(1)}, \ldots, o^{(t)}\right)} \qquad (8.2)
\end{aligned}$$

Thus we can simply multiply $\tau^{(t+1)}(X^{(t+1)})$ by $P(o^{(t+1)} \mid X^{(t+1)})$ and renormalize. The process of computing $\varphi^{(t+1)}$ from $\varphi^{(t)}$ is often called *filtering*.

To solve the probability estimation task we need to calibrate the tree and find the clique potentials for the various $X^{(t)}$. We can do this by our standard clique tree algorithm, first doing an upstream pass and then doing a downstream pass. In fact, this algorithm was independently developed in the HMM community before the theory of clique trees was developed and was called the *forward-backward* algorithm for obvious reasons.

In principle, these computations extend easily to general DBNs, with the only difference being that the belief state is now defined as the joint distribution over all the state variables, i.e., $\varphi^{(t)}(\boldsymbol{X}^{(t)})$. The key question when doing inference in DBNs is how to represent this joint distribution. As we shall see in the next sections, the answer to this question strongly depends on the type of DBN we are dealing with (continuous, discrete or hybrid). However, no matter the type of the DBN, the belief state has an unfortunate property: In almost every DBN, after a relatively small number of time slices the belief state has no conditional independence structure, because usually all the variables are correlated through the past [Ast65].

As an example, consider the network of Figure 8.1 and consider the two variables $Terrain^{(2)}$ and $Pos^{(2)}$ which are a part of the belief state $\varphi^{(2)}$. Intuitively it seems that these variables are independent, but in fact they are not. The reason is that $Terrain^{(2)}$ is correlated with $SensorFailure^{(1)}$ (both are influenced by $Terrain^{(0)}$), which in turn is correlated with $Pos^{(1)}$ (both are parents of the observed variable $SensedPos^{(1)}$). Since $Pos^{(1)}$ is correlated with $Pos^{(2)}$ we get that $Terrain^{(2)}$ is correlated with $Pos^{(2)}$ where all the correlations come from past variables, i.e., $Terrain^{(2)}$ and $Pos^{(2)}$ are not conditionally independent given any other variables in the time 2 belief state.

8.4 The Kalman Filter

Perhaps the most important type of DBNs is also the simplest type, where all the variables are continuous and all the CPDs in $\mathcal{B}_0$ and $\mathcal{B}_\rightarrow$ are linear CPDs. These models, often called *linear dynamic systems* or simply *linear systems*, as well as models based on them were investigated and used in many communities long before DBNs were introduced. For example, tracking using linear systems is at the heart of most target tracking systems [BSLK01], e.g., tracking submarines using sonar data. It is also widely used in computer vision, computer graphics, economics, and control.

The popularity of linear models comes from their simplicity, and more importantly the simplicity of the tracking algorithm. The key property of linear systems is that *the belief state is always a multivariate Gaussian.* We can show this by induction. Since $\mathcal{B}_0$ is a linear Gaussian the initial belief state is indeed a Gaussian. Assume now that $\varphi^{(t)}$ is a Gaussian. When we propagate the belief state using Equation 8.1 we get that $\tau^{(t+1)}$ is also a Gaussian (since all the CPDs in $\mathcal{B}_\rightarrow$ are linear). Similarly, when we condition on the evidence all the CPDs are linear and therefore $\varphi^{(t+1)}$ is also a Gaussian.

Thus the belief state can be kept in a compact representation of a mean vector and a covariance matrix, whose representation size is only quadratic in the number of state variables. Propagating the belief state involves various manipulations of Gaussians and linear Gaussians and can be done in time which is cubic in the number of state variables. We therefore have a polynomial algorithm for monitoring linear systems. This algorithm is known as the *Kalman Filter (KF)* [Kal60].

If the variables are continuous but the CPDs are not linear then in general the belief state is not a Gaussian. The standard approach is to approximate the belief state as a Gaussian using the ideas discussed in Chapter 6, i.e., using EKF or algorithms based on numerical integration rules such as the the Unscented Filter.

8.5 Inference in Discrete DBNs

We now turn our attention to the representation of the belief state for discrete DBNs. As we have seen at the end of Section 8.3, we cannot use the structure of the DBN to simplify the representation of the belief state. Since the belief state has no conditional independencies, there is no better way to represent it than a table with one number for every possible combination of the discrete variables. Obviously this is not a practical method for reasonably sized networks, i.e., more than 15–20 variables. Thus, exact inference is usually not possible for DBNs, and we must resort to approximations.

In this section we concentrate on a specific type of approximation presented and analyzed by Boyen and Koller [BK98]. The idea is that, although the belief state is fully correlated, some of the correlations are weak and can be approximated as independencies. Recall our example from Figure 8.1. Although, as we have seen, the variables $Terrain^{(2)}$ and $Pos^{(2)}$ are correlated, their correlation is weak, and it may be reasonable to approximate it as an independence. As another example, consider monitoring a physical system such as an engine. Although eventually all the belief state variables become correlated it may make sense to approximate the state of various subsystems (such as the gas subsystem and the cooling subsystem) as independent or alternatively as conditionally independent (e.g., given the state of the electric subsystem).

We begin with the case where the belief state is approximated as a product of separate distributions, i.e., we approximate the various subsystems as fully independent rather than conditionally independent. More precisely, we define a set of disjoint clusters $\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_n$ such that $\boldsymbol{X} = \cup_i \boldsymbol{Y}_i$. We maintain an approximate belief state:

$$\varphi^{(t)}(\boldsymbol{X}^{(t)}) \approx \hat{\varphi}^{(t)}(\boldsymbol{X}^{(t)}) = \prod_{i=1}^{n} \hat{\varphi}^{(t)}(\boldsymbol{Y}_i^{(t)})$$

If we were to propagate this approximate belief state forward, all of our variables would become correlated as before. Therefore, we perform the following process: at each time t, we take $\hat{\varphi}^{(t)}$, propagate it forward to time $t+1$, and condition the resulting distribution on the evidence. The result is a new distribution $\tilde{\varphi}^{(t+1)}$ (note that we do not get the correct belief state $\varphi^{(t+1)}$ since we started with an approximate belief state $\hat{\varphi}^{(t)}$). The distribution $\tilde{\varphi}^{(t+1)}$ may not be factored as a product of independent marginals over the clusters $\boldsymbol{Y}_i^{(t+1)}$, but we can approximate it as one. This approximation step is straightforward: We simply compute $\tilde{\varphi}^{(t+1)}(\boldsymbol{Y}_i^{(t+1)})$ for every i; the product of these marginals is $\hat{\varphi}^{(t+1)}$.

We can use the clique tree algorithm to implement this process. We generate a BN over $\boldsymbol{X}^{(t)}, \boldsymbol{X}^{(t+1)}$ where the distribution over $\boldsymbol{X}^{(t)}$ is $\hat{\varphi}^{(t)}$; this BN will have a fully connected subgraph over the variables in each cluster $\boldsymbol{Y}_i^{(t)}$, but no edges between variables in different clusters $\boldsymbol{Y}_i^{(t)}$ and $\boldsymbol{Y}_j^{(t)}$. The connections between the two time slices are as in the original 2-TBN. We then generate a clique tree where we make sure that the variable in any cluster $\boldsymbol{Y}_i^{(t+1)}$ are together in at least one clique. We set the evidence of the time $t+1$ variables and calibrate the clique tree. When the calibration is complete, we can read $\tilde{\varphi}^{(t+1)}(\boldsymbol{Y}^{(t+1)})$ from the appropriate cliques in the tree. The marginals over the clusters give us $\hat{\varphi}^{(t+1)}$.

We now consider the case where the $\boldsymbol{Y}_i's$ are overlapping, leading to conditional independencies rather than simple direct independencies. The idea is to think of the various $\boldsymbol{Y}_i$'s as the nodes in a clique tree and add the sepsets $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_{n-1}$ to the tree (where $\boldsymbol{Z}_i = \boldsymbol{Y}_i \cap \boldsymbol{Y}_{i+1}$).[1] The belief state is therefore represented as the product of

[1]We restrict ourselves to subsets $\boldsymbol{Y}_i's$ that can be used as cliques in a clique tree, i.e., they can be arranged in a tree such that the running intersection property holds.

the cliques, divided by the product of the sepsets:

$$\hat{\varphi}^{(t)}(\boldsymbol{X}^{(t)}) = \frac{\prod_{i=1}^{n} \hat{\varphi}^{(t)}(\boldsymbol{Y}_i)}{\prod_{i=1}^{n-1} \hat{\varphi}^{(t)}(\boldsymbol{Z}_i)}$$

The propagation step can once again be carried out using the clique tree algorithm. We can either build a Bayes net that represents $\hat{\varphi}^{(t)}(\boldsymbol{X}^{(t)})$ or better yet, plug the belief state directly into a clique tree over $\{\boldsymbol{X}^{(t)}, \boldsymbol{X}^{(t+1)}\}$ with the appropriate structure [BK98].

It is not immediately clear whether this algorithm provides a useful approximation. Intuitively, the process introduces errors into the belief state at every time step. If the errors accumulate, our results might become less and less relevant. Luckily, this is not the case. Consider two parallel processes: the (hypothetical) process of exact tracking (with correct belief states), and the (actual) process of approximate tracking. We would like to analyze how far they can diverge from each other. There are two opposing forces at work. The stochastic transition from time t to time $t+1$ adds noise to both belief states, causing the difference between them to decrease. Conditioning both on the same evidence also causes the error to decrease (on expectation). The approximation is the other opposing force, which causes the error to increase. It can be shown that, if the process is stochastic enough, these two opposing forces remain balanced [BK98]. Thus, the error remains bounded indefinitely over time.

8.6 Inference in Hybrid DBNs

The third and last type of DBNs we discuss in this chapter is hybrid DBNs, containing both discrete and continuous variables. In particular, we are interested in CLG DBNs, i.e., DBNs where both $\mathcal{B}_0$ and $\mathcal{B}_{\rightarrow}$ are CLGs. These models are often called *switching linear systems* or *Switching Linear Dynamic Systems (SLDS)*. Bar Shalom *et al.* [BSLK01] provide a good introduction to these models, and Ghahramani and Hinton [GH00] provide an up-to-date literature survey. The reason for the name is that, given the discrete variables, the system is linear; "switches" in the value of the discrete variable correspond to switches in the system behavior.

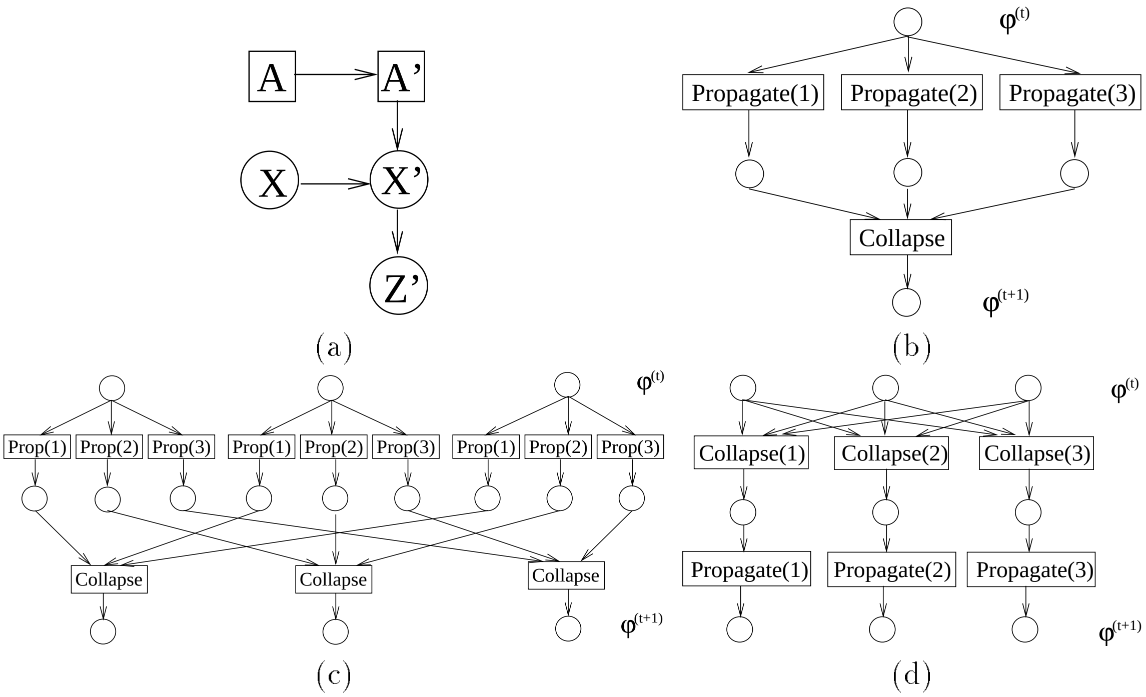


Figure 8.3: (a) An SLDS (b) GPB1 algorithm (c) GPB2 algorithm (d) IMM algorithm

Unfortunately, just like the situation with static Bayesian networks, the combination of discrete and continuous variables in DBNs is significantly harder than any one of them separately. Consider, for example, the SLDS in Figure 8.3(a) where the belief state has the discrete variables A (with M values) and the continuous variable X (the continuous variable Z is observed). $\mathcal{B}_0$ has no edges, and therefore the distribution of $X^{(0)}$ in $\varphi^{(0)}(A^{(0)}, X^{(0)})$ is a Gaussian. When we propagate the belief state to time slice 1, we get a different Gaussian for $X^{(1)}$ for every value of $A^{(1)}$. Thus the belief state $\varphi^{(1)}(A^{(1)}, X^{(1)})$ is a mixture of M Gaussians, one for every value of A. When we propagate to time slice two, each one of the M Gaussians in time slice 1 results in M different Gaussians over $X^{(2)}$ (one for every value of $A^{(2)}$), and together $\varphi^{(2)}(A^{(2)}, X^{(2)})$ is a mixture of M^2 Gaussians. It is not hard to verify that the belief state $\varphi^{(t)}(A^{(t)}, X^{(t)})$ has a mixture of M^t Gaussians, where each Gaussian corresponds to a particular combination of $A^{(1)}, \ldots, A^{(t)}$. For similar reasons, if each time slice has n binary variables then the exact belief state at time t is a mixture of up to $(2^n)^t$ Gaussians. We conclude that there is no hope of representing the exact belief state for anything but trivial SLDSs over a small number of time slices.

There has been much work trying to deal with this problem, e.g., [BSLK01, SS91, Kim94, GH00]. Most of this work assumes that the number of discrete combination at every time slice is relatively small, and the difficulty comes mostly from the exponential dependence on t. In this section, we review some of the techniques to approximate the belief state under these conditions. In Chapter 9, we discuss the case where the number of discrete combinations in every time slice is too large, preventing us from directly using these ideas.

To make the propagation algorithm in SLDSs practical, we must make sure that the mixture of Gaussians represented by the belief state does not get too large. The two main techniques to do so are *pruning* and *collapsing*. Pruning algorithms reduce the size of the belief state by discarding some of its Gaussians. The standard pruning algorithm simply keeps the N Gaussians with the highest probabilities, discards the rest and then renormalizes the probabilities such that they sum up to 1.

Collapsing algorithms work by partitioning the mixture of Gaussians into subsets and then collapsing all the Gaussians in each subset into one Gaussian. Thus, if the belief state was partitioned into N subsets, the result would be a belief state with exactly N Gaussians. The different collapsing algorithms usually differ in the number of subsets, how they are chosen and exactly when the collapsing is performed. In this section, we present three algorithms: GPB1, GPB2, and IMM [BSLK01].

The *General Pseudo-Bayesian (GPB)* algorithms limit the number of Gaussians in the belief state to M^{k-1} where M is the number of possible system modes, i.e., the number of discrete combinations that induce a different behavior for the continuous variables in the 2-TBN, and k is a positive integer. For example, in the SLDS of Figure 8.3(a), $M = |\mathrm{Dom}(A)|$. Note that if $\varphi^{(t)}$, the belief state at time t, has M^{k-1} Gaussians then after one step of propagation, $\varphi^{(t+1)}$ will have M^k Gaussians. The GPB algorithms partition these Gaussians into M subsets, where the elements in each subset have the same trajectory $k-1$ time steps into the past. In the example of Figure 8.3(a) Gaussians in the same subset will correspond to the same values of $A^{(t-k+3)}, \ldots, A^{(t+1)}$

To understand this better, let us examine the special cases of $k = 1$ and $k = 2$. In the case of $k = 1$ we get the GPB1 algorithm, that keeps exactly one Gaussian in the belief state. Assuming we have M different modes of behaviors and one Gaussian in the time t belief state we get M Gaussians at time $t+1$ which are collapsed into one Gaussian. This algorithm is shown in Figure 8.3(b) for the case of $M = 3$ (each circle represents a Gaussian). Note that all the collapsed Gaussian correspond to different discrete values at time $t+1$; thus, their trajectories differ on the most current discrete instantiation.

Consider now the case of $k = 2$, which corresponds to the GPB2 algorithm. Here we keep M Gaussians in the belief state. After propagation we get M^2 Gaussians, each one corresponding to one of M discrete values at time t and one of M discrete values at time $t+1$. We now partition this mixture into M subsets, where each subset has Gaussians that have the same $t+1$ discrete value (and different time t values). Each subset is collapsed, resulting in a belief state with M Gaussians. The entire process is shown in Figure 8.3(c) (again for $M = 3$). Note that the trajectories of collapsed Gaussians agree on the one most recent time step.

The problem of the GPB2 algorithm is that at every propagation step we must generate M^2 Gaussians, which is often too computationally expensive. If we cannot use GPB2 but would like to do better than GPB1 we can use the *Interacting Multiple Model (IMM)* algorithm which is more efficient than GPB2 but more accurate than GPB1.

Just like the GPB2 algorithm, the IMM algorithm keeps M Gaussians in the belief state for time t, but unlike GPB2, these Gaussians are collapsed before propagating them into time $t+1$. It seems that this should give us the same behavior as GPB1, but the difference is that we collapse the Gaussians in different ways depending on the discrete value at time $t+1$. For every one of the M values of the discrete variables at time $t+1$, we compute a different set of probabilities for our time t mixture and then collapse the Gaussians using these probabilities.

For example, consider again the SLDS of Figure 8.3(a). Our belief state has the distribution $\varphi^{(t)}(A^{(t)}, X^{(t)}) = P(A^{(t)})P(X^{(t)} \mid A^{(t)})$ where $P(X^{(t)} \mid A^{(t)})$ is a Gaussian. We now consider each one of the possible values of $A^{(t+1)}$. For each one of these values we compute the distribution

$$\begin{aligned} P\left(A^{(t)}, X^{(t)} \mid A^{(t+1)}\right) &= P\left(A^{(t)} \mid A^{(t+1)}\right) P\left(X^{(t)} \mid A^{(t)}, A^{(t+1)}\right) \\ &= P\left(A^{(t)} \mid A^{(t+1)}\right) P\left(X^{(t)} \mid A^{(t)}\right) \end{aligned} \tag{8.3}$$

Thus for every value of $A^{(t+1)}$ we have a mixture of M Gaussians but with different probabilities from the belief state probabilities. Intuitively, the new probabilities better correspond to the specific mode of operation. Note that computing the new probabilities involves only discrete variables and therefore is simple as long as M is not too large. We then collapse the mixture using the new probabilities and propagate the Gaussian to time $t+1$ resulting in just one Gaussian over time $t+1$ variables. After doing this for every value of $A^{(t+1)}$, we end up with M Gaussians at time $t+1$ which are our new time $t+1$ belief state. This process is illustrated in Figure 8.3(d).

Although we collapse our belief state in M different ways when using the IMM algorithm, we only create M Gaussians at time $t+1$, as opposed to M^2 Gaussians in GPB2. The extra work compared to GPB1 is the computation of the new mixture probabilities and collapsing M mixtures instead of just one, but usually this extra computational cost is small relative to the cost of computing the M Gaussians at time $t+1$. Therefore the computational cost of the IMM algorithm is only slightly higher than the GPB1 algorithm, and is significantly lower than GPB2. In practice, it seems that IMM often performs significantly better than GPB1 and almost as well as GPB2 [BBS88]. Thus, the IMM algorithm appears to be a good compromise between complexity and performance.

8.7 Particle Filters

As we have seen in previous chapters, sampling techniques offer an alternative approach to other approximate inference algorithms for Bayesian networks. The situation is similar for Dynamic Bayesian networks. The most popular sampling technique for DBN inference is *Particle Filters* [GSS93, Kit96], which will be described in this section.

The naive way to use sampling techniques in DBNs is to apply LW (see Section 2.4.1) to the unrolled DBN, where each sample corresponds to one possible trajectory of the system $\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(t)}$. To generate a sample, we initially sample $\boldsymbol{X}^{(0)}$ from $\mathcal{B}_0$. Given $\boldsymbol{X}^{(0)}$ we can now sample $\boldsymbol{X}^{(1)}$ using $\mathcal{B}_{\rightarrow}$ and weigh the sample by the likelihood of the evidence $\boldsymbol{o}^{(1)}$. We continue this process sampling the variables one time slice at a time and multiplying the sample weight by the probability of the evidence.

It is possible to modify this procedure and get an online algorithm that we can use for monitoring. We maintain a set of samples at each time slice t: $\boldsymbol{x}^{(t)}[1], \ldots, \boldsymbol{x}^{(t)}[N]$. Each of these is associated with a weight $w^{(t)}[j]$. At each time slice, we take each of the samples, propagate it forward to sample the variables at time $t+1$, and adjust its weight to suit the new evidence at time $t+1$. We can then use our set of time $t+1$ samples to answer various queries.

Unfortunately, this simple approach does not work in practice. Recall that if all the evidence is at the leaves, LW generates the samples according to the prior distribution and the evidence affects only the weights. For example, in the car surveillance DBN (Figure 8.1), all the evidence is at the leaves. Thus, we would generate completely random trajectories for the car, and only use the evidence to influence the weights. Unfortunately, most random trajectories would not make the evidence very likely. Thus, each sample would have very low weight. While this is a problem also in regular BNs, in DBNs it leads to a complete breakdown of the algorithm: As time evolves, more and more trajectories deviate from the true system trajectory and are never corrected by the evidence. The samples become less and less reliable as an approximation of the system's hidden state and LW diverges rapidly as time goes by.

The *Particle Filtering (PF)* algorithm is a way to deal with the problem. The key intuition is to think of the samples at time t as an approximation of the belief state at time t. What we truly want to do is not to propagate each sample but propagate the belief state as a whole, i.e., take i.i.d. samples from the belief state and propagate those new samples forward.

Assume our belief state has N samples $(\boldsymbol{x}^{(t)}[1], \ldots, \boldsymbol{x}^{(t)}[N])$ with N associated weights $(w^{(t)}[1], \ldots, w^{(t)}[N])$. We sample N i.i.d. samples $\boldsymbol{y}^{(t)}[1], \ldots, \boldsymbol{y}^{(t)}[N]$ (also called *particles*) from the belief state:

$$P\left(\boldsymbol{Y}^{(t)}[j] = \boldsymbol{x}^{(t)}[i]\right) = w^{(t)}[i].$$

We then propagate each one of the $\boldsymbol{y}^{(t)}[j]$ particles to time $t+1$ and weigh them by the time $t+1$ evidence. This new set of samples and weights serves as our approximation to the time $t+1$ belief state. Note that a sample $\boldsymbol{x}^{(t)}[i]$ with a large $w^{(t)}[i]$ is likely to be sampled more than once while a sample with a low $w^{(t)}[i]$ is likely to not be sampled even once.

Another way to view this algorithm is that some samples are "better" than others. In particular, samples that have larger weights explain the evidence so far better, and are likely to be closer to the current state. Thus, we want to use a sample $\boldsymbol{x}^{(t)}[i]$ more often as a starting point for propagation if it has a large weight. Particle filtering works much better than LW in practice, as the quality of the approximation does not degrade over time.

In SLDS models it is very natural to combine particle filtering with the Rao-Blackwellization techniques (Section 3.4.3) leading to the *Rao Blackwellized Particle Filter (RBPF)* [DGA00]. The idea is to sample only the discrete variables. Given an assignment to the discrete variables, the continuous variables have a normal distribution which we can compute and represent exactly. Thus, each particle is represented as an assignment to the time t discrete variables and a Gaussian over the time t continuous variables. The tradeoffs are similar to the RB tradeoffs: each RBPF particle is more expensive to generate but contains more information. In practice it seems that RBPF particles are well worth the extra computational complexity needed to compute them.

Chapter 9

Scaling up Hybrid DBNs

In Section 8.6 we discussed some of the standard techniques for inference in SLDS models. Recall that if we have n binary discrete variables within every time slice and t time steps, then the exact belief state is a mixture of $(2^n)^t$ Gaussians. All the techniques described in Section 8.6 assumed that n is a small number. Under this assumption, a mixture of 2^n Gaussians was assumed to be small enough to be tractable, and the concern was to deal with the temporal blowup of raising 2^n to the power of t.

Unfortunately, in many domains it is not realistic to assume that n is small. For example, in the fault diagnosis domain, we have discrete variables that represent different faults (e.g., a broken sensor) and various modes of behavior (e.g., valve open or valve closed). Obviously, in a reasonably sized system we can have many such variables. The upshot is that we must deal with the case of 2^n being too large by itself, even if we ignore the temporal blowup. This is the subject of this chapter.

9.1 A Collapsing Algorithm

In Section 8.6 we discussed how the belief state in hybrid DBNs can be represented as a mixture of Gaussians. Recall that in GPB1 the belief state has just one Gaussian while in GPB2 and IMM the belief state has 2^n Gaussians. Unfortunately, none of these options is useful for us. One Gaussian is not enough for our needs — in fault

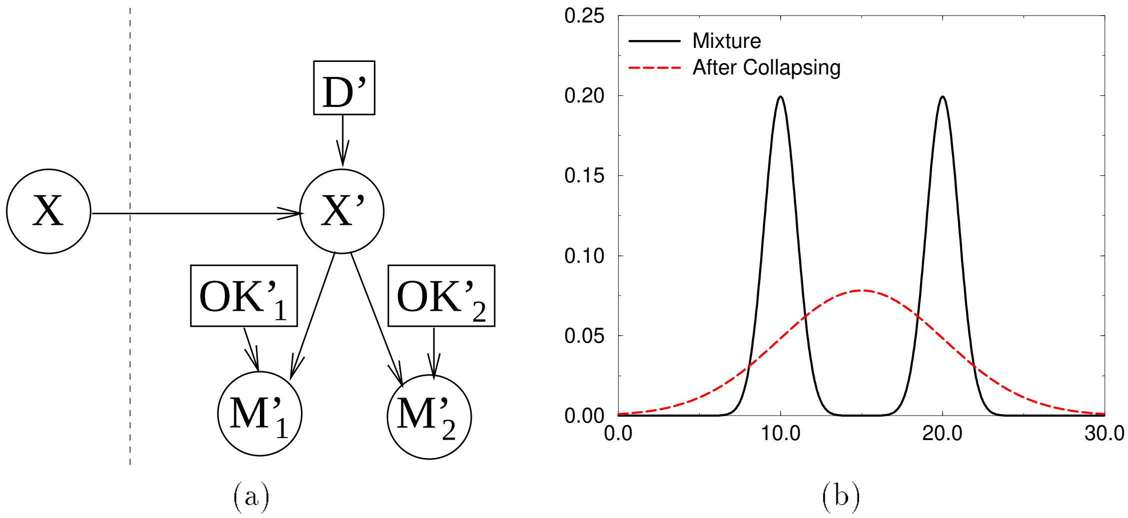


Figure 9.1: (a) A simple hybrid DBN, describing the movement of a robot (b) Collapsing a mixture of two different hypotheses

diagnosis it is often important to track a few distinct hypotheses over time in order to decide which is the correct one, and thus, we cannot afford to collapse all the hypotheses into just one Gaussian. On the other hand, a mixture of 2^n Gaussians is too large for our needs, since it is often not practical to keep such a large mixture.

In fact, even if could keep 2^n Gaussians in our belief state, the collapsing scheme of GPB2 may not be ideal. In GPB2 we collapse together Gaussians which share the same assignment to time $t+1$ discrete variables (and have different assignments to time t discretes). However, some of the Gaussians which we collapse may be very different from each other, and worthwhile keeping as distinct hypotheses. On the other hand, it is possible that among the 2^n Gaussians that GPB2 keeps some are very similar to each other and can be collapsed without a significant loss of accuracy.

For example, assume that we have a very simple hybrid DBN shown in Figure 9.1(a) that models the movement of a robot in a one-dimensional world. The variable X indicates the location of the robot. Depending on the value of the discrete variable D the robot can move either left or right. There are two sensors that measure the robot's location represented by M_1' and M_2'. Both sensors can either work properly or be broken — this behavior is represented by the discrete variable OK'_i for $i = 1, 2$. If the sensor is working, i.e., $OK' = \mathit{True}$, then we get some noisy reading of the robot's location. If the sensor is broken then we get a reading from $\mathcal{N}(0, 1)$, and get no information about the robot's position. The two sensors differ in accuracy and reliability — the first sensor is very accurate but tends to break down a lot while the second sensor is much more noisy but very rarely gets broken.[1]

Consider two hypotheses that agree that at time $t+1$ the robot moved left and that both sensors did not work, namely $D^{(t+1)} = \mathit{Left}$, $OK_1^{(t+1)} = \mathit{False}$ and $OK_2^{(t+1)} = \mathit{False}$. Note that the hypotheses agree on the time $t + 1$ variables and therefore will be collapsed by GPB2. Assume now that the two hypotheses start with two different robot locations at time t. Although the robot is assumed to move left in both cases, the two hypotheses will represent two very different locations at time $t + 1$ (as there is no sensor reading to change the estimates of either hypothesis). For example, one hypotheses may have $P(X^{(t+1)}) = \mathcal{N}\left(X^{(t+1)}; 10, 1\right)$ and the other may have $P(X^{(t+1)}) = \mathcal{N}\left(X^{(t+1)}; 20, 1\right)$. If we collapse these hypotheses (assuming they have the same likelihood) we get the hypotheses $P(X^{(t+1)}) = \mathcal{N}\left(X^{(t+1)}; 15, 26\right)$. Figure 9.1(b) shows the mixture of the two original hypotheses as well as the collapsed hypotheses. As we can see from the figure, not only we have lost our two distinct hypotheses, but according to the collapsed hypotheses, we now believe that the robot is likely to be at $X = 15$, which is unlikely according to both of the original hypotheses. This can be especially problematic when dealing with physical systems. The reason is that some values, such as $X = 15$, may violate some physical constraints. While the original mixture may reflect these constraints, after collapsing we get a hypothesis in which much of the probability mass is in the "forbidden zone". Thus, it is often desirable to avoid collapsing distinctly different hypotheses.

On the other hand, consider now two hypotheses at time $t+1$ which originate from the same robot location at time t, have the same direction of the robot movement, have sensor 1 (the accurate sensor) as working but differ on whether sensor 2 is working. Both hypotheses may have a very similar estimate for the robot's location

[1]In a more realistic model the sensor faults would be persistent over time. We will discuss such models later in this chapter.

at time $t+1$, since when sensor 1 is working, the noisy reading of sensor 2 has very little effect on the estimate for the robot's location. Thus, it may make sense to collapse these two similar hypotheses into one, instead of keeping them as separate hypotheses as GPB2 would do.

The conclusion is that, even when our belief state can keep 2^n Gaussians, using GPB2's collapsing scheme may not be optimal. Instead we may want to consider which Gaussians are similar and which are different before deciding which to collapse. To formalize this intuition, we first need to define similarity between two Gaussians $P_1 = \mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1)$ and $P_2 = \mathcal{N}(\boldsymbol{\mu}_2, \Sigma_2)$. We choose to define similarity in terms of KL-divergence between the Gaussians, i.e., we define the distance between P_1 and P_2 as:

$$J(P_1, P_2) = \frac{1}{2}(D(P_1 \parallel P_2) + D(P_2 \parallel P_1)) \tag{9.1}$$

Our measure has some desired properties:

- It is symmetric: $J(P_1, P_2) = J(P_2, P_1)$
- $J(P_1, P_2) \geq 0$ with equality iff $P_1 \equiv P_2$
- $J(P_1, P_2)$ can be computed in closed form [CT91]:

  $$J(P_1, P_2) = \frac{1}{2}\left(\mathrm{tr}(\Sigma_2^{-1}\Sigma_1) + \mathrm{tr}(\Sigma_1^{-1}\Sigma_2) - 2d\right) + \frac{1}{4}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^t(\Sigma_1^{-1} + \Sigma_2^{-1})(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

  where d is the dimension of the Gaussians.

Having defined a distance measure we are ready to present our collapsing algorithm, shown in Figure 9.2. We assume we are given a mixture of Gaussians and would like to collapse it to a smaller mixture of at most K Gaussians. Our algorithm partitions the mixture into at most $K+1$ subsets of Gaussians. The first K subsets represent Gaussians that are collapsed together, i.e., all the Gaussians in subset i for $1 \leq i \leq K$ are collapsed into one Gaussian in the resulting mixture. Gaussians in subset $K+1$ are discarded. The question, of course, is how to create the partition scheme.

We use a greedy approach, that takes into consideration both the likelihood of the

Collapsing Algorithm

$\mathcal{A} = \{\langle w_i, P_i \rangle_{i=1,\ldots,n}\}$: a mixture of n Gaussians

K: Maximal size of mixture after collapsing

c: threshold of distance matrix

1. Sort $\mathcal{A}$ in decreasing order of weights w_i
2. Mark all Gaussians in $\mathcal{A}$ as unused
3. Set $\mathcal{B} = \emptyset$
4. While $|\mathcal{B}| < K$ and $\mathcal{A}$ has unused Gaussians
    5. Let $\langle w, P \rangle$ be the first unused Gaussian in $\mathcal{A}$
    6. Find all $l \in L$ s.t. $\langle w_l, P_l \rangle \in \mathcal{A}$ is unused and $J(P, P_l) < c$
    7. Mark $\langle w, P \rangle$ and $\langle w_l, P_l \rangle_{l \in L}$ as used
    8. Let $w' \leftarrow w + \sum_{l \in L} w_l$
    9. Let P' be the Gaussian resulting from collapsing $\langle w, P \rangle$ and $\langle w_l, P_l \rangle_{l \in L}$ according to Equations 3.10 and 3.11
    10. Let $\mathcal{B} \leftarrow \mathcal{B} \cup \{\langle w', P' \rangle\}$
11. end-while

Figure 9.2: Collapsing a mixture of Gaussians into K Gaussians

different Gaussians and their similarity to each other. We first sort the different hypotheses by their likelihoods. Then, starting from the most likely Gaussian P, we find the Gaussians $P_1, \ldots, P_l$ that are "similar" to P and create the subset $\{P, P_1, \ldots, P_l\}$. We then find the next most likely Gaussian in our list that was not used so far and continue in the same manner. We stop after using up all the Gaussians in the original list or generating K subsets.

We define two Gaussians P_1 and P_2 as similar if $J(P_1, P_2) < c$. Note the importance of the parameter c. If we set $c = \infty$ then all the Gaussians are similar to each other and all will be collapsed together. Thus, for $c = \infty$ we effectively get the GPB1 algorithm. On the other hand, if $c = 0$ then no two different Gaussians are similar to each other and therefore every subset will contain exactly one Gaussian. Thus, for $c = 0$ our algorithm behaves like a simple pruning algorithm, keeping the K most likely Gaussians as they are while discarding all the rest.

As an example, assume we have three binary discrete variables A, B and C such that we have a Gaussian for every combination of A, B and C. Let the original mixture be

A	B	C	$P(A,B,C)$	Corresponding Gaussian
False	*False*	*False*	0.2	P_1
False	*False*	*True*	0.1	P_2
False	*True*	*False*	0.05	P_3
False	*True*	*True*	0.23	P_4
True	*False*	*False*	0.12	P_5
True	*False*	*True*	0.08	P_6
True	*True*	*False*	0.07	P_7
True	*True*	*True*	0.15	P_8

Suppose that we use our collapsing algorithm on this mixture with the size of the resulting mixture bounded by $K = 3$. The most likely Gaussian is P_4 and we now look for a similar Gaussian. Assume that only P_6 is similar enough, leading to our first subset $\{P_4, P_6\}$. We next move to the remaining most likely Gaussian, P_1. Assuming that out of the remaining Gaussians P_1 is similar to P_2 and P_8, we get the subset $\{P_1, P_2, P_8\}$. The next most likely remaining Gaussian is P_5, and in this case it is not similar to any of the remaining Gaussian, leading to the subset $\{P_5\}$. We now have generated 3 subsets and therefore we stop. The output of our algorithm are the subsets $\{P_4, P_6\}$, $\{P_1, P_2, P_8\}$ and $\{P_5\}$. The Gaussians P_3 and P_7 are discarded.

Assume we have N d-dimensional Gaussians in the original mixture and would like to collapse them into a mixture of K Gaussians. Sorting the Gaussians by their weights takes $O(N \log N)$. To compute the KL-divergence between two Gaussians it is necessary to invert to covariance matrices, which can be done once for each of the original N Gaussians. Once the covariance matrices are inverted, evaluating Equation 9.1 takes $O(d^2)$. Inverting all the original Gaussians takes $O(Nd^3)$. Equation 9.1, i.e., line 6 of Figure 9.2, needs to be evaluated in K iterations for $O(N)$ Gaussians per iteration leading to a complexity of $O(NKd^2)$. The collapsing operation in line 9 can be done in $O(M_i d^2)$ where M_i is the number of Gaussians that are being collapsed in iteration i. Since $\sum_{i=1}^{K} M_i \leq N$ all the collapsing operations can be done in $O(Nd^2)$. Thus, the total complexity of the algorithm is $O(N \log N + Nd^3 + NKd^2)$.

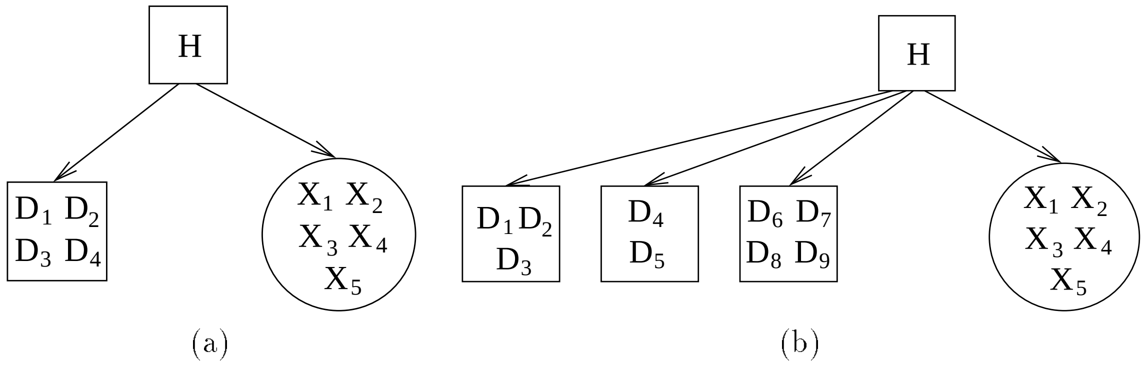


Figure 9.3: (a) Belief state representation (b) Belief state representation with a decomposed probability distribution over the discrete variables

9.2 The Hypothesis Variable

In the previous section we discussed the collapsing algorithm while implicitly assuming that all the variables in the belief state are continuous. As we already mentioned, this assumption is not realistic, as the belief state often contains persistent discrete variables as well as continuous ones. We therefore need to discuss how to represent both the discrete and continuous variables in the belief state and the relation between them.

For GPB1 and GPB2 the representation is quite simple. In GPB1, where all the Gaussians are collapsed together, we can represent the belief state as a product between a multivariate Gaussian over the continuous variables and a factor over the discrete variables, since by keeping just one Gaussian the continuous variables are assumed to be independent of the discrete variables. In GPB2 every hypothesis corresponds to a full instantiation over the discrete variables, and thus, the belief state can be represented as a Gaussian factor where we keep one Gaussian for every combination of the discrete variables.

It is less clear how the belief state should look like after using our collapsing algorithm. In this case, every Gaussian corresponds to some set of discrete assignments and to their weights. In other words, each Gaussian P induces a probability distribution over the discrete variables where every assignment that was not collapsed into P has probability zero and the assignments that were involved in P have a probability which is proportional to their weight.

As an example, consider again the example from the end of Section 9.1. Assume that the discrete variables A and B are in the belief state while C is not. We end up with three Gaussians that correspond to the subsets $\{P_4, P_6\}$, $\{P_1, P_2, P_8\}$ and $\{P_5\}$. These Gaussians induce the following distributions on the discrete variables (after renormalization):

$\{P_4, P_6\}$

A	B	$P(A, B)$
False	*False*	0
False	*True*	0.742
True	*False*	0.258
True	*True*	0

$\{P_1, P_2, P_8\}$

A	B	$P(A, B)$
False	*False*	0.667
False	*True*	0
True	*False*	0
True	*True*	0.333

$\{P_5\}$

A	B	$P(A, B)$
False	*False*	0
False	*True*	0
True	*False*	1
True	*True*	0

The obvious question is how to represent this type of belief state. It is convenient to introduce a new random variable H, called the *hypothesis variable*. The hypothesis variable is a discrete variable with one value for every Gaussian in the belief state. We define the belief state using the graphical model shown in Figure 9.3(a) where the continuous variables in the belief state are $X_1, \ldots, X_n$ and the discrete variables are $D_1, \ldots, D_m$. Both the discrete and continuous variables depend on the hypothesis variable: For every value of the hypothesis variable there is a Gaussian distribution defined over the continuous variables and some factor defined for the discrete variables. It is important to note how the hypothesis variable correlates the continuous variables and the discrete ones: Every Gaussian in the mixture corresponds to one value of the hypothesis variable which in turn induces a probability distribution on the discrete variables.

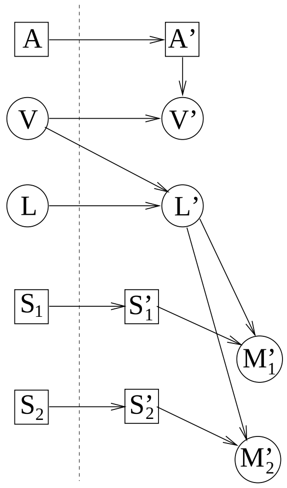


Figure 9.4: A model for a one-dimensional robot including its velocity

9.3 Empirical Comparison

To compare the collapsing algorithm with GPB2 and GPB1 we slightly extended our one-dimensional robot model from Figure 9.1 into the model shown Figure 9.4. To make sure it is possible to run GPB2 we kept the belief state to a minimum with only 12 discrete states.

There are two continuous variables in the belief state: the robot's *Location (L)* and the robot's *Velocity (V)*. Positive velocity implies that the robot is moving to the right while negative velocity implies that the robot is moving to the left. At every time step the robot can take one of three *Actions (A)* that influence its velocity: it can accelerate to the right, i.e., increase its velocity, accelerate to the left, i.e., decrease its velocity by adding some negative constant to it, or decelerate, i.e., decrease the absolute value of its velocity. Note that, depending on its current velocity the result of accelerating to one direction can be similar to decelerating. There are two sensors that measure the robot's location: M_1 and M_2. Both sensors can be in two possible *Statuses* (S_1 and S_2): *Failed* or *Okay*. Sensor 1 is less likely to be working than sensor 2, but if it is working, then its reading is less noisy. The only observed nodes in the network are M_1 and M_2. The exact parameters of the network are given in Table 9.1. For example, we see that $P(Action_1 = \mathit{Acc\text{-}Left} \mid Action_0 = \mathit{Acc\text{-}Right}) = 0.05$.

Node	Discrete Parents	Distribution
$Action_0$	—	(0.333, 0.333, 0.333)
$Status1_0$	—	(0.5, 0.5)
$Status2_0$	—	(0.5, 0.5)
$Location_0$	—	$\mathcal{N}(0, 1)$
$Velocity_0$	—	$\mathcal{N}(0, 1)$
$Action_1$	$Action_0 = \textit{Slow-Down}$	(0.6, 0.2, 0.2)
	$Action_0 = \textit{Acc-Right}$	(0.25, 0.7, 0.05)
	$Action_0 = \textit{Acc-Left}$	(0.25, 0.05, 0.7)
$Status1_1$	$Status1_0 = \textit{Failed}$	(0.9, 0.1)
	$Status1_0 = \textit{Okay}$	(0.3, 0.7)
$Status2_1$	$Status2_0 = \textit{Failed}$	(0.6, 0.4)
	$Status2_0 = \textit{Okay}$	(0.2, 0.8)
$Location_1$	—	$\mathcal{N}(L_0 + V_0, 0.0025)$
$Velocity_1$	$Action_1 = \textit{Slow-Down}$	$\mathcal{N}(0.5V_0, 0.0001)$
	$Action_1 = \textit{Acc-Right}$	$\mathcal{N}(V_0 + 1, 0.0016)$
	$Action_1 = \textit{Acc-Left}$	$\mathcal{N}(V_0 - 1, 0.0016)$

Table 9.1: Parameters for a simple model for a one-dimensional robot

Unfortunately, even for this extremely small example, exact inference is intractable for any reasonably long sequence. Since we wanted to conduct our experiments with sequences of hundreds of time steps we had to evaluate our algorithms based on criteria which can be computed without the exact posterior distribution. We used two such criteria.

The first criterion was comparison to the "omniscient Kalman Filter". An omniscient Kalman filter gets to observe all the discrete variables. Given these variables we are left with a simple linear system on which we can use the Kalman filter algorithm. Clearly, the omniscient Kalman filter should do a better job of tracking the system than any method based on imperfect, partial information, but comparing to it provides some indication on the quality of the approximation.

Thus, given a sampled trajectory we used the omniscient Kalman filter to compute $P(\mathbf{\Gamma}^{(t)})$, the belief state over the continuous variables for every t. For each inference algorithm A we also computed $P_A(\mathbf{\Gamma}^{(t)})$, i.e., the belief state according to A. We then computed the KL-divergence $D(P \parallel P_A)$ averaged over t. When computing

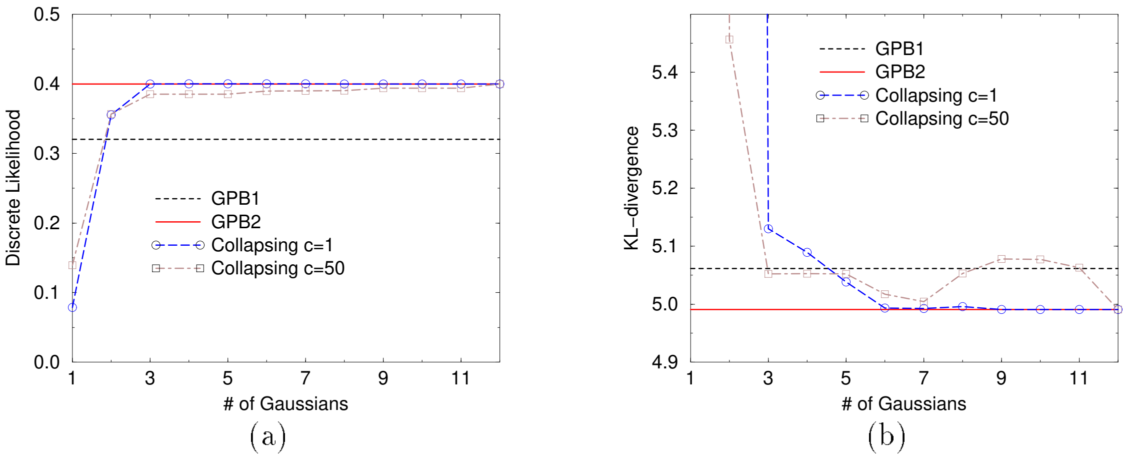


Figure 9.5: Results for the one-dimensional robot model (a) KL-divergence from the omniscient Kalman Filter (b) Likelihood of the correct discrete trajectory

the KL-divergence we collapsed P_A into a single Gaussian in order to have a simple expression for the KL-divergence.

The second criterion was based on the likelihood of the discrete variables. From our generated data we knew $\boldsymbol{\delta}^{(t)}$, the assignment to the discrete variables at time t. Thus, our second criterion was $P_A(\boldsymbol{\delta}^{(t)})$ averaged over t, i.e., the likelihood that algorithm A assigned to the correct discrete event.

We emphasize that none of these criteria is perfect, since sometimes unlikely events happen. For example, it is certainly possible that the most likely discrete event according to the exact belief state is not the actual discrete assignment. In these cases, the likelihood of the "correct" discrete event should be small, and therefore maximizing the second criterion is not always the right thing to do. The same holds true for the first criterion — when the unlikely events are the correct ones, the omniscient Kalman Filter may be a poor approximation to the belief state. Still, on average the two criteria should provide a reasonable indication for the quality of the algorithms. Thus, we should treat the results as a noisy sensor for the quality of the algorithms.

The results are shown in Figure 9.5, where Figure 9.5(a) shows the KL-divergence criterion and Figure 9.5(b) shows the criterion based on the likelihood of the discrete trajectory. In both graphs the results were averaged over 100 trajectories of 200 time steps each. The graphs show the performance of the GPB1 and GPB2 algorithms as well as the collapsing algorithm. For the collapsing algorithm we show the performance for two values of the threshold parameter c (as used in Figure 9.2): $c = 1$ and $c = 50$. The X axis represents the number of Gaussians in the belief state — this axis is relevant only for the collapsing algorithm since GPB1 always has one Gaussian and GPB2 always has 12 Gaussians (thus, GPB1 and GPB2 are shown as horizontal lines).

The graphs show that GPB2 beats GPB1, as expected. The performance of the collapsing algorithm is very poor when we use just one Gaussian, and not as good as GPB2 when using two Gaussians. Note that, for any $c < \infty$, the collapsing algorithm does not behave like GPB1, but for $c = \infty$ we get the exact same performance of GPB1 with just one Gaussian. However, with three or more Gaussians in the belief state, the performance of the collapsing algorithm is comparable to GPB2 and in some cases even slightly better.

Interestingly, for $c = 50$ the KL-divergence criterion actually detoriates when we use the collapsing algorithm and go beyond 7 Gaussians (unless we go as high as 12 Gaussians). We conjecture that, in this system, 7 Gaussians are usually enough to keep the correct hypothesis in the belief state, and adding more Gaussians increases the KL-divergence from the omniscient Kalman Filter, since we add hypotheses which are different from the correct one. However, this seems to be more of a symptom of the KL-divergence criterion which compares the distribution only with the correct hypothesis rather than comparing it with the true belief state, and less of an indication of a problem with the collapsing algorithm.

Given the noise of our criteria it is very hard to say whether GPB2 performs better or worse than the collapsing algorithm when using at least 3 Gaussians. However, it seems that at least for $c = 1$ both GPB2 and the collapsing algorithm outperform GPB1 and that their performance is comparable. This by itself is a win for the collapsing algorithm that achieves similar performance to GPB2 with a much smaller belief state. Thus, it is reasonable to believe that when the number of Gaussians required by GPB2 is too large, the collapsing algorithm provides a useful alternative by offering comparable performance to GPB2 and outperforming the GPB1 alternative.

9.4 Decomposition of the Belief State

There is a potential problem with the belief state representation described in Section 9.2, as the size of the factor over the discrete variables is exponential in the number of variables. Thus, if the number of discrete variables in the belief state is too large, it may not be practical to keep a factor that represents the joint distribution over them. In this case we can use similar ideas to the BK algorithm discussed in Section 8.5. We impose some independence or conditional independence assumptions on the probability distribution over the discrete variables in order to get a more compact representation for it. As an example, in Figure 9.3(b) the probability distribution over the discrete variables is decomposed. Instead of keeping a factor over $\{D_1, \ldots, D_9\}$ we keep three much smaller factors over $\{D_1, D_2, D_3\}$, $\{D_4, D_5\}$ and $\{D_6, D_7, D_8, D_9\}$, leading to significant savings in space. In this example, we have each subset of the discrete variables independent of the other subsets given the hypothesis variable. Just like the BK algorithm for discrete DBNs, we can also choose overlapping subsets, leading to conditional independencies between the subsets rather than direct independencies.

Given the decomposition of the distribution over the discrete variables, it is natural to ask whether it is useful to have an analogous decomposition for the continuous variables as well. The answer is yes and no: an exactly analogous decomposition is not very useful but a somewhat different decomposition is useful.

In an exactly analogous decomposition, instead of keeping one multivariate Gaussian for every value of the hypothesis variable, we decompose it as a product of smaller Gaussians or, if we want to keep some of the dependencies between the continuous variables, a product of conditional forms. However, unlike discrete factors that grow exponentially with the number of variables, the space required in order to represent a multivariate Gaussian is only quadratic in the number of variables, and therefore we can use our Gaussian representation even if the belief state contains tens or even hundreds of continuous variables. Thus, unless the system is extremely large, it is feasible to represent the belief state over the continuous variables given the hypothesis variable as a multivariate Gaussian, and there is no real need for decomposition.

However, we can use a different type of decomposition, where we decompose the hypothesis variable as well as the continuous variables. This type of decomposition lets us represent a much larger mixture of Gaussians, effectively keeping more hypotheses in our belief state. To see this, consider a mixture of Gaussians over $\{X_1, \ldots, X_n\}$ where the X_i's are independent of each other. Assume that the marginal probability for every X_i ($1 \leq i \leq n$) is a mixture of two Gaussians: $\mathcal{N}(X_i; -1, 1)$ and $\mathcal{N}(X_i; 1, 1)$, where the weight of each Gaussian is 0.5. It is easy to see that the joint distribution over $\{X_1, \ldots, X_n\}$ is a mixture of 2^n Gaussians. The weight of each mixture component is 2^{-n}, the covariance matrix is I and the mean vector looks like $(\pm 1, \ldots, \pm 1)$, i.e., the absolute value of each coordinate in the mean vector is 1. If we naively represent the joint distribution, then we need to represent 2^n different Gaussians and for each one the space requirement is $O(n^2)$. On the other hand, if we use the independencies between the variables we can represent the joint distribution as a product of n marginals. Each marginal is represented as a mixture of 2 univariate Gaussians, and therefore has a space requirement of only $O(1)$. Thus, by using the independencies between the continuous variables we can dramatically reduce the space requirements from $O(2^n n^2)$ to $O(n)$.

The reason for this savings is that for different X_i's we need different collapsing schemes, and this can be done only if we keep marginals rather than the full joint distribution. For example, assume that $n = 2$ and that the joint distribution is therefore a mixture of four Gaussians: $P_1 = \mathcal{N}((1,1), I)$, $P_2 = \mathcal{N}((1,-1), I)$, $P_3 = \mathcal{N}((-1,1), I)$ and $P_4 = \mathcal{N}((-1,-1), I)$. If we only consider the marginal of X_1 then $P_1 = P_2$ and $P_3 = P_4$ and therefore collapsing P_1 with P_2 and P_3 with P_4 does not change the marginal for X_1. If we consider the marginal of X_2 then we should collapse P_1 with P_3 and P_2 with P_4. However, if we operate on the full joint distribution then no two Gaussians are similar and we should not collapse anything. Thus, in case some subsets of the continuous variables are (close to) independent from each other, it is useful to decompose the distribution and define a separate collapsing scheme for every subset of continuous variables.

The last insight leads us to a further decomposition of the belief state. We partition the variables in our belief state into *subsystems*, where every subsystem contains

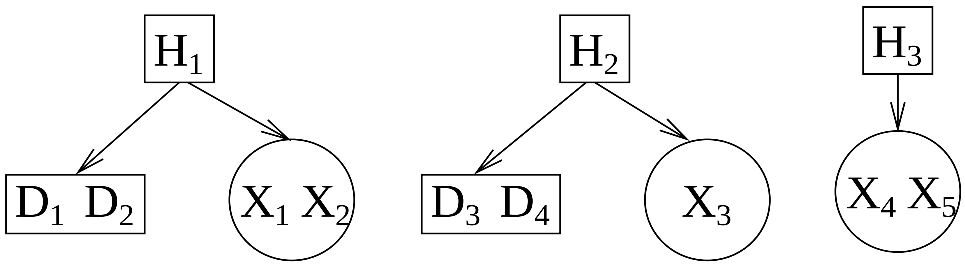


Figure 9.6: Belief state with three hypothesis variables. Variables in the different subsystems are assumed to be independent.

both discrete and continuous variables. For example, if we model a car, we can have one subsystem for the variables relating to the electric system, one subsystem for variables relating to the gas system, another one for the engine, and so on. We assign one hypothesis variable to each subsystem, and represent the distribution over a subsystem using a graphical model such as the ones shown in Figure 9.3(a) or Figure 9.3(b). Note that in each subsystem we can use a different collapsing scheme.

In the simplest case, shown in Figure 9.6, we assume that the variables in different subsystems are independent of each other. We can relax this assumption and let the various subsystems be only conditionally independent of each other using a graphical model such as the one shown in Figure 9.7, where we let variables in one subsystem influence variables in different subsystems. For the continuous variables we can represent such a dependency using conditional Gaussians.

Formally, we define the belief state using a graphical model, where we have n hypothesis variables $H_1, \ldots, H_n$. We partition the variables in the belief state into n subsets such that each subset is influenced by one of the hypothesis variables. In addition, for every $1 \le i, j \le n$ we let discrete variables in subsystem i influence the discrete variables in subsystem j, and similarly for every $1 \le i, j \le n$ we let continuous variables in subsystem i influence the continuous variables in subsystem. The only constraint we impose on the graphical model is that it contains no cycles.

There is a subtle difference between our representation and the representation used in the discrete version of BK. In the discrete case, the BK algorithm represents

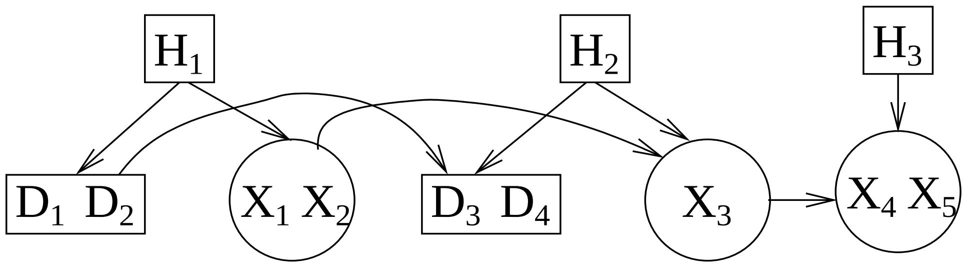


Figure 9.7: Belief state with three hypothesis variables and interdependencies between the sub-systems

conditional independencies by having overlapping sets of variables in the various subsystems, while making sure that every two subsystems agree on the marginal distribution of their intersection. This is problematic in our case. Our decomposition is useful mainly if we use different collapsing schemes in different subsystems. However, by using different collapsing schemes, we get different mixtures over the continuous variables, and in general the marginal distributions of the intersection would be different in different subsystems. Thus, if we keep overlapping sets of continuous variables and use different collapsing schemes in different subsystems, the joint distribution may not be well defined.

In contrast, the semantics of the graphical model in Figure 9.7 is well defined: the collapsing scheme for $\{X_4, X_5\}$ is defined using H_3. For different hypotheses in subsystem 3, i.e., different values of H_3, we can represent different dependencies of $\{X_4, X_5\}$ on $\{X_3\}$. Note that the conditional Gaussians do not depend on the hypothesis variable H_2, and thus are the same for all the hypotheses of subsystem 2.

It is left to discuss the modifications in the collapsing algorithm to account for the conditional Gaussians. Consider, for example, subsystem 3 in Figure 9.7. The conditional Gaussians in the mixture are defined over the variables $\{X_3, X_4, X_5\}$. We call these variables the *relevant* variables and denote them as $\boldsymbol{R}_i$.

Definition 9.1 *The* relevant *variables of subsystem i, denoted by $\boldsymbol{R}_i$, are all the variables in the belief state of subsystem i and all their direct parents in the decomposed belief state, excluding any hypothesis variables.*

As another example, in Figure 9.7 the relevant variables in subsystem 2 are $\boldsymbol{R}_2 =$

$\{D_1, D_2, D_3, D_4, X_1, X_2, X_3\}$. We denote the continuous variables in $\boldsymbol{R}_i$ as $\boldsymbol{\Gamma}_i$ and partition them as $\boldsymbol{\Gamma}_i = \boldsymbol{\Gamma}_i^+ \cup \boldsymbol{\Gamma}_i^-$ where $\boldsymbol{\Gamma}_i^+$ are the continuous variables in subsystem i and $\boldsymbol{\Gamma}_i^-$ are the continuous variables which are not in subsystem i, but are relevant to subsystem i. In our example of Figure 9.7, $\boldsymbol{\Gamma}_3 = \{X_3, X_4, X_5\}$, $\boldsymbol{\Gamma}_3^+ = \{X_4, X_5\}$ and $\boldsymbol{\Gamma}_3^- = \{X_3\}$. Similarly, we denote the discrete variables in $\boldsymbol{R}_i$ as $\boldsymbol{\Delta}_i$ and partition them as $\boldsymbol{\Delta}_i = \boldsymbol{\Delta}_i^+ \cup \boldsymbol{\Delta}_i^-$ where $\boldsymbol{\Delta}_i^+$ are the discrete variables in subsystem i and $\boldsymbol{\Delta}_i^-$ are the other discrete variables in $\boldsymbol{\Delta}_i$. Again, in our example $\boldsymbol{\Delta}_2 = \{D_1, D_2, D_3, D_4\}$, $\boldsymbol{\Delta}_2^+ = \{D_3, D_4\}$ and $\boldsymbol{\Delta}_2^- = \{D_1, D_2\}$.

Recall that, in our collapsing algorithm, we collapsed together Gaussians that had a small KL-divergence between them, i.e., Gaussians that had similar probability distributions. Now however, we are interested in collapsing together Gaussians that have a similar conditional distribution, even if their joint distribution is significantly different. More formally, for two Gaussians P_1 and P_2 we do not demand that $P_1(\boldsymbol{\Gamma}_i^+, \boldsymbol{\Gamma}_i^-)$ and $P_2(\boldsymbol{\Gamma}_i^+, \boldsymbol{\Gamma}_i^-)$ would be similar; instead we would like $P_1(\boldsymbol{\Gamma}_i^+ \mid \boldsymbol{\Gamma}_i^-)$ and $P_2(\boldsymbol{\Gamma}_i^+ \mid \boldsymbol{\Gamma}_i^-)$ to be close, allowing for the possibility that $P_1(\boldsymbol{\Gamma}_i^-)$ and $P_2(\boldsymbol{\Gamma}_i^-)$ are significantly different. Fortunately, there is a natural metric that we can use, based on the *conditional KL-divergence*, which is defined as:

$$D(P_1(\boldsymbol{y} \mid \boldsymbol{x}) \parallel P_2(\boldsymbol{y} \mid \boldsymbol{x})) = \int P_1(\boldsymbol{x}) \int P_1(\boldsymbol{y} \mid \boldsymbol{x}) \log \frac{P_1(\boldsymbol{y} \mid \boldsymbol{x})}{P_2(\boldsymbol{y} \mid \boldsymbol{x})} d\boldsymbol{y} d\boldsymbol{x}$$

It is easy to verify [CT91] that

$$D(P_1(\boldsymbol{y} \mid \boldsymbol{x}) \parallel P_2(\boldsymbol{y} \mid \boldsymbol{x})) = D(P_1(\boldsymbol{x}, \boldsymbol{y}) \parallel P_2(\boldsymbol{x}, \boldsymbol{y})) - D(P_1(\boldsymbol{x}) \parallel P_2(\boldsymbol{x}))$$

Intuitively, the last expression means that the conditional KL-divergence ignores the differences between $P_1(\boldsymbol{x})$ and $P_2(\boldsymbol{x})$ and leaves only the differences between $P_1(\boldsymbol{y} \mid \boldsymbol{x})$ and $P_2(\boldsymbol{y} \mid \boldsymbol{x})$. Thus, we can use the collapsing algorithm shown in Figure 9.2 but we replace the metric J defined in Equation 9.1 by the new metric J' defined as:

$$J'(P_1(\boldsymbol{\Gamma}_i^+, \boldsymbol{\Gamma}_i^-), P_2(\boldsymbol{\Gamma}_i^+, \boldsymbol{\Gamma}_i^-)) = \frac{1}{2}(D(P_1(\boldsymbol{\Gamma}_i^+ \mid \boldsymbol{\Gamma}_i^-) \parallel P_2(\boldsymbol{\Gamma}_i^+ \mid \boldsymbol{\Gamma}_i^-)) +$$

$$D(P_2(\mathbf{\Gamma}_i^+ \mid \mathbf{\Gamma}_i^-) \parallel P_1(\mathbf{\Gamma}_i^+ \mid \mathbf{\Gamma}_i^-))) \tag{9.2}$$

After we decide on the collapsing scheme we need to transform our distribution into a conditional distribution if $\mathbf{\Gamma}_i^- \neq \emptyset$ or $\Delta_i^- \neq \emptyset$. If $\Delta_i^- \neq \emptyset$ we must transform the factor over $\mathbf{\Delta}_i$ from the distribution $P(\Delta_i^+, \mathbf{\Delta}_i^-)$ into the conditional distribution $P(\Delta_i^+ \mid \mathbf{\Delta}_i^-)$. We can do that by simply dividing $P(\Delta_i^+, \Delta_i^-)$ by the marginal $P(\mathbf{\Delta}_i^-)$. Similarly, if $\mathbf{\Gamma}_i^- \neq \emptyset$ we must transform the Gaussian over $P(\mathbf{\Gamma}_i^+, \mathbf{\Gamma}_i^-)$ into the conditional Gaussian $P(\mathbf{\Gamma}_i^+ \mid \mathbf{\Gamma}_i^-)$. Again, we can do that by dividing $P(\mathbf{\Gamma}_i^+, \mathbf{\Gamma}_i^-)$ by the marginal $P(\mathbf{\Gamma}_i^-)$. To do that, we need to represent the belief state using either canonical forms (Section 3.2.2) or conditional forms (Section 3.2.3).

9.5 Putting it all together

We now have the tools necessary to scale up the algorithms discussed in Section 8.6 to large scale hybrid DBNs. There are four crucial components in our propagation algorithm:

- The representation of the belief state discussed in Sections 9.2 and 9.4.
- The enumeration algorithm from Section 5.1.4.
- The numerical integration techniques discussed in Chapter 6 which are used together with the enumeration algorithm.
- The collapsing algorithm from Sections 9.1 and 9.4

In each propagation step, we are given the decomposed belief state for time t $P(\boldsymbol{X})$, the the 2-TBN $\mathcal{B}_{\rightarrow}$, and $\boldsymbol{E}^{(t+1)}$, the evidence for time $t+1$. The task is to create the decomposed belief state for time $t+1$. We first create the full Bayesian network $\mathcal{B}$ by combining the belief state $P(\boldsymbol{X})$ with the 2-TBN $\mathcal{B}_{\rightarrow}$. We then create the decomposed belief state one subsystem at a time. For each subsystem we first use the enumeration algorithm in order to approximate the distribution $P(\boldsymbol{R}_i^{(t+1)} \mid \boldsymbol{E}^{(1)}, \ldots, \boldsymbol{E}^{(t+1)})$, and then use the collapsing algorithm on the resulting mixture (using the distance metric of either Equation 9.1 or Equation 9.2, depending on whether $\mathbf{\Gamma}_i^- = \emptyset$ or not). We get a different collapsing scheme and a different

```
One step inference
     P(X):  time t belief state
     B→:  the 2-TBN
     e^(t+1):  evidence at time t+1
  1. Plug P(X) into B→ to create the BN B
  2. For each subsystem i do
    3. Create the subnetwork B_i over R_i ∪ E_i and their ancestors
    4. Use enumeration algorithm (Section 5.1.4) to create a mixture
       of Gaussians over {X^(t+1), E^(t+1)}
    5. Condition on the evidence E^(t+1) = e^(t+1)
    6. Use collapsing algorithm (Figure 9.2) to create a collapsing
       scheme and the hypothesis variable for subsystem i, H_i^(t+1)
    7. Generate time t+1 belief state as discussed in Section 9.4.
       If needed generate conditional distributions
```

hypothesis variable in each subsystem. If $\mathbf{\Gamma}_i^- \neq \emptyset$ we generate a conditional Gaussian and if $\mathbf{\Delta}_i^- \neq \emptyset$ we generate a discrete factor that represents a conditional distribution.

The only issue that we have not discussed yet is how to take advantage of the subsystem decomposition for generating the Gaussians. The simple approach is to first generate a mixture of Gaussians over all the variables in the network $\mathcal{B}$ and condition it on the evidence $\boldsymbol{e}^{(t+1)}$. Next, each subsystem $1 \leq i \leq n$ marginalizes out all the variables which are not in $\boldsymbol{R}_i$, leading to n mixtures of Gaussians, where mixture i is defined over $\boldsymbol{R}_i$. We then perform the collapsing algorithm independently within each subsystem. Thus, although the different subsystems get their Gaussian from the same source (the original mixture over all the system variables), they can define different collapsing schemes, based on their own $\boldsymbol{R}_i$.

A different approach is to let each subsystem generate its own Gaussians. This approach is useful if the subsystems are relatively small compared to the entire system. Since the size of the covariance matrix is quadratic in the number of variables, if we have many variables it may be more efficient to generate relatively small Gaussians for each subsystems, instead of Gaussians over all the variables in $\mathcal{B}$. In this case subsystem i must generate Gaussians over the relevant variables $\boldsymbol{R}_i$ and every observation variable that is not independent of $\boldsymbol{R}_i$, denoted as $\boldsymbol{E}_i$. To do so we define the Bayesian network $\mathcal{B}_i$. The nodes in $\mathcal{B}_i$ are $\boldsymbol{R}_i \cup \boldsymbol{E}_i$ and all their ancestors in the DBN. The edges between the nodes in $\mathcal{B}_i$ are the same edges as in the DBN and so are the CPDs. In other words, $\mathcal{B}_i$ is the original DBN restricted to $\boldsymbol{R}_i \cup \boldsymbol{E}_i$ and all their ancestors. To generate Gaussians for subsystem i we can use the enumeration algorithm on $\mathcal{B}_i$, ignoring the rest of the DBN. If all the $\mathcal{B}_i$'s are small compared to the full DBN, this approach will be computationally more efficient. The resulting algorithm is shown in Figure 9.5.

Unfortunately, in practice it is often the case that even if $\boldsymbol{R}_i$ contains relatively few variables, $\boldsymbol{E}_i$ contains many of the observations in the system, forcing $\mathcal{B}_i$ to be almost as large as the entire DBN. As an example, the RWGS system, discussed in detail in Chapter 10, has a chemical reactor that creates a chemical reaction between the gases flowing in the system. The chemical reactor can be modeled as a relatively small subsystem. However the efficiency of the reaction depends on the incoming flow F, which in turn depends on many flows and pressures around the system. Thus, F is dependent on every flow and pressure sensor in the system. The result is that the network $\mathcal{B}_i$ for the reactor subsystem will include all the variables that influence the flows and pressures in the system, which are almost all the variables in the model.

However, even in cases where many observation nodes influence the variables in some subsystem, it is often the case that some of the dependencies are extremely weak, and can be ignored almost without changing the posterior distribution. In the chemical reactor example, if we have a measurement M of the incoming flow we may be able to ignore all the other flow and pressure measurements in the system without introducing any significant errors.[2] Thus, it may be a reasonable approximation to ignore some of the nodes in $\boldsymbol{E}_i$ in order to reduce the size of $\mathcal{B}_i$ and decrease the time it takes to generate a Gaussian. By doing so we may be able to generate more Gaussians given the same amount of computation time, compensating for the errors introduced by ignoring some of the observations.

[2]Note that, the flow is not truly independent of the other measurements in the system, even given the measurement M. Since M, like any other measurement, is noisy, even given M there is still some uncertainty over the actual incoming flow, and other measurements can influence our belief over f.

9.6 Smoothing

So far we concentrated on the problem of tracking, i.e., the evidence that was available to us was the observations up to the current t. In some applications we are also interested in the probability estimation task, defined in Section 8.3. Here we have a sequence of observations $\boldsymbol{o}^{(1)}, \ldots, \boldsymbol{o}^{(T)}$ and we would like to find the distribution over $\boldsymbol{X}^{(t)}$ for some $1 \leq t \leq T$. Probability estimation comes up when we perform offline inference, perhaps in order to learn the parameters of the model. However, even if we restrict ourselves to online applications, sometimes it is possible to look into the future. For example, assume that we get a new observation every second. If we are allowed a short delay of five second before we are required to give our estimate for the state of system, then we can use five "future" observations in order to improve our estimate. The process of using future observations in order to improve our estimate for the present is called *smoothing*.

Smoothing can be quite important even in tracking applications since the supporting evidence for some hypotheses may be present only in the future. Consider, for example, the case of a leak in some water tank, and assume we measure the water level in the tank. When the leak starts it has no effect on the water level, and thus the leak hypothesis is not supported by the present evidence. Only by looking at future evidence we can realize that the leak hypothesis is much more likely than it currently seems to be.

In discrete DBNs, one can perform smoothing using message passing. Consider again the DBN of Figure 8.2(a) and the resulting clique tree of Figure 8.2(b). Assume that we need to find the belief state over $\boldsymbol{X}^{(2)}$ and that in addition to the evidence $\boldsymbol{O}^{(2)}$ we also have the evidence $\boldsymbol{O}^{(3)}$ available. We can now perform smoothing using an algorithm which is called the *forward-backward* algorithm, which is identical to the clique tree calibration algorithm. We first send messages forward in time, i.e., from the clique $\{\boldsymbol{X}_0, \boldsymbol{X}_1\}$ to the clique $\{\boldsymbol{X}_1, \boldsymbol{X}_2\}$ and from it to the clique $\{\boldsymbol{X}_2, \boldsymbol{X}_3\}$, and then we send messages backward in time. After we send the message from $\{\boldsymbol{X}_2, \boldsymbol{X}_3\}$ back to the clique $\{\boldsymbol{X}_1, \boldsymbol{X}_2\}$, the clique $\{\boldsymbol{X}_1, \boldsymbol{X}_2\}$ has the correct posterior distribution given all the evidence.

Unfortunately, we cannot use the same algorithm for hybrid DBNs. The reason is that the clique tree in Figure 8.2(b) is not strongly rooted, since Definition 3.11 does not hold for any clique in the tree. Intuitively, in a strongly rooted tree, we can perform an upstream pass without collapsing any Gaussians, but when we send the forward messages in a clique tree resulting from a hybrid DBN, we have to collapse the Gaussians in the belief state to avoid the exponential blowup. Thus, using the forward-backward algorithm in hybrid DBNs may not be well defined. For example, it is easy to convert the example from Section 3.3.4 that resulted in a negative variance, to a tree that represents a DBN.

However, it is possible to salvage some part of the smoothing algorithm. Since the problems in the backward pass involve illegal Gaussians we can ignore the continuous variables in the backward pass and send messages that involve only the discrete variables. The result would be a belief state at time t where the discrete variables would have a distribution which reflects the future evidence as well at the past and present evidence. Thus, by using smoothing over the discrete variables we do not change the actual Gaussians in our belief state, but we do change their weights.

We can use the new distribution over the discrete variables in two ways:

- **Improve our estimate for the state of the system.** As we have seen in the leak example, we need the future evidence in order to come up with an accurate estimate for the current state.
- **Improve our collapsing scheme.** Recall that the collapsing scheme is influenced by the likelihood of the hypotheses, as it attempts to keep the more likely hypotheses relatively unchanged in the belief state. If we sort the hypotheses based on their new weights, we are more likely to keep in the belief state hypotheses which may not be very likely given the present evidence but become more likely given the future evidence. Obviously keeping such hypotheses in our belief state is a desired feature.

It is important to make sure we do not double count the future evidence. The belief state used for the next propagation step should only reflect the past and present evidence and not the future evidence. Thus, the distribution over the discrete variables that is sent forward should be the distribution before the smoothing step. In the collapsing algorithm, we can use the smoothed distribution as a heuristic for deciding which hypotheses should remain in the belief state, but when we actually do collapse hypotheses we must use their weights before the smoothing step. In other words, we can use the new weights in line 1 of Figure 9.2 but we must use the original weights in lines 5–10.

Chapter 10

Application: The RWGS System

So far we have considered some theoretical aspects of hybrid Bayesian networks and developed some algorithms which were tested on relatively simple examples and synthetic data. In this chapter we put our algorithm to a real test, namely performing fault diagnosis for the RWGS system, a complex real-world physical system. We begin with a quick overview of the system and then discuss some of the issues that had to be addressed while modeling the system. We then test our inference algorithm and our model on both synthetic data and actual data collected during system runs.

10.1 The RWGS System

The *Reverse Water Gas Shift (RWGS)* system shown in Figure 10.1 is a complex physical system designed to extract oxygen from carbon dioxide. NASA foresees a number of possible uses for the RWGS system, including producing oxygen from the atmosphere on Mars and converting carbon dioxide to oxygen within closed human living quarters. In a Mars mission, the RWGS is supposed to operate for 500 or more days without human intervention [LG00], thus making a fault diagnosis system a necessary component. A prototype of the system was constructed at NASA's Kennedy Space Center (KSC).

The RWGS works by decomposing carbon dioxide (CO_2), which is abundant on Mars, into oxygen (O_2) and carbon monoxide (CO). The system, whose schematic is

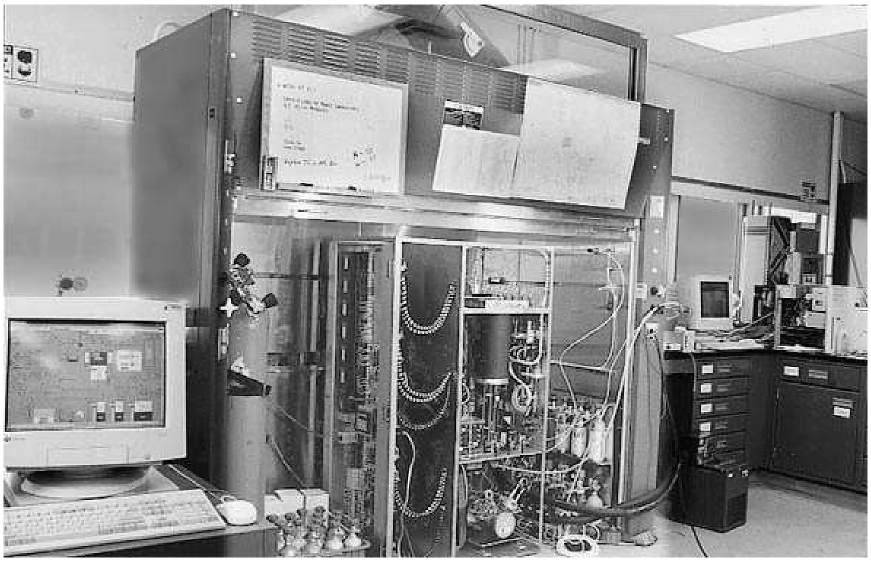


Figure 10.1: The prototype RWGS system

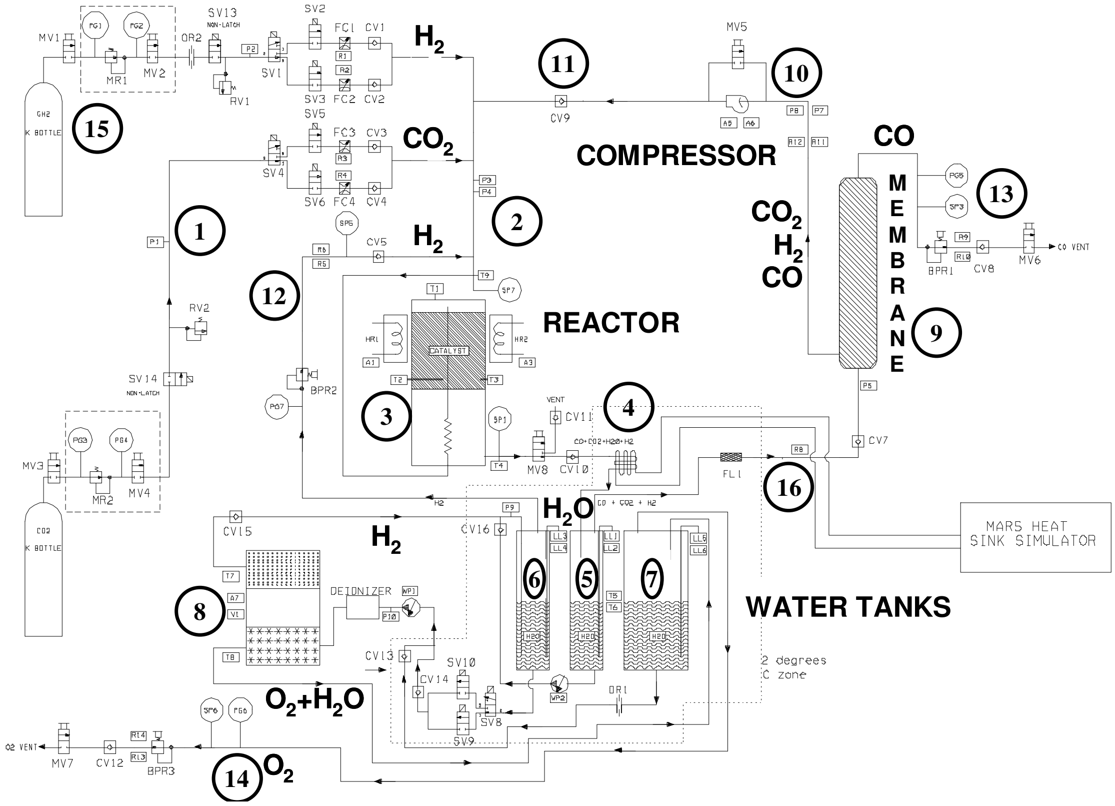


Figure 10.2: The RWGS schematic

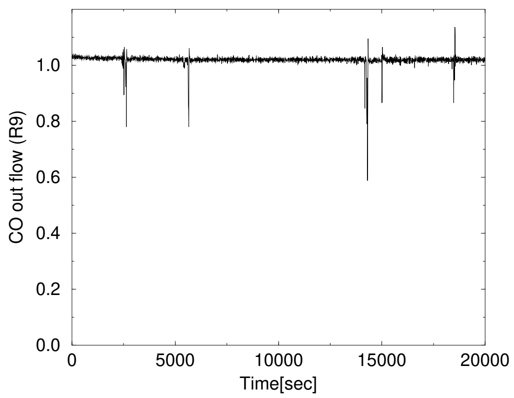
shown in Figure 10.2 [Goo02], contains two loops: a gas loop that converts CO_2 and hydrogen (H_2) into water (H_2O) and CO, and a water loop that electrolyzes the H_2O to produce O_2 and H_2. Under normal operation, CO_2 at line (1) is combined with H_2 returned from the electrolyzer via line (12), and a mixture of CO_2, H_2, and CO from the reactor recycle line (11). This mixture enters a catalyzed reactor (3) heated to 400°C. Approximately 10% of the CO_2 and H_2 react to form CO and H_2O:

$$CO_2 + H_2 \rightleftharpoons CO + H_2O$$

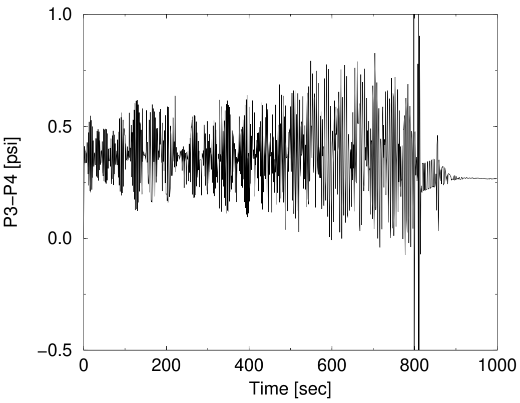
The H_2O is condensed at (4) and is stored in a tank (5). The remaining gas mixture passes through a separation membrane (9), which sends a fraction of the CO to the vent (13) while directing the remaining mixture into the recycle line (11). A compressor (10) is used to maintain the necessary pressure differential across the membrane. In the water loop, the H_2O in tank (5) has some CO_2 dissolved in it, which would be detrimental to the electrolyzation process. To remedy this, the H_2O is pumped into a second tank (6), and has H_2 bubbled through it to purge the CO_2. From there, the H_2O is pumped into the electrolyzer (8), which separates a portion of it into O_2 and H_2. The H_2 re-enters the gas loop via (12), while the remaining H_2O, along with the O_2, goes into tank (7), where the mixture is cooled and separated. The H_2O returns to the electrolyzer, while the O_2 leaves the system through (14). The entire process is controlled by a computer program that opens and closes valves, turns water pumps, the condenser and the electrolyzer on and off, controls gas flow and reactor temperatures, and so on.

In addition to its normal operating mode, the system may operate without the electrolyzer and water pumps. In this mode, the H_2 for the reaction is supplied by a supply line (15) paralleling the CO_2 supply line. This option is not feasible for operation on Mars, but has proven useful for testing the physical system while under development.

The RWGS is an interconnected nonlinear system where the various components influence each other in complicated and sometimes unexpected ways. For example, during runs without the electrolyzer, it is necessary to empty the water tank (5)



(a)



(b)

Figure 10.3: (a) Effects of emptying a water tank (b) Pressure difference between P_3 and P_4

periodically, to prevent water from accumulating and eventually overflowing the tank. This causes the gases in the tank to expand, and thus creates a significant and sudden pressure drop, which affects the flow throughout the whole system. This phenomenon is demonstrated in Figure 10.3(a) [Whi01]. The graph shows the flow through the CO vent (13) as it evolves over time — the spikes correspond to emptying the water tank.

10.2 Modeling the RWGS

We model the RWGS using a hybrid DBN. The 2TBN has 313 nodes, out of which 66 are discrete and 247 are continuous. All the discrete variables are persistent and belong to the belief state, i.e., we have 33 discrete variables in the belief state. Out of the 33 discrete variables, 8 correspond to various modes of operation, e.g., compressor on or off, valves being open or closed, and so on. The other 25 discrete variables correspond to sensor faults. Note that the variables indicating the mode of operation can also represent faults, for example, it is possible that the valve is supposed to be open but it is stuck closed. We model this by keeping all the discrete variables hidden, i.e., if the valve is supposed to be open then with high probability we have *Valve=Open* but it is also possible that *Valve=Closed*.

The continuous variables in our model capture the continuous-valued elements of our system, e.g., pressure at various points in the system, flow rates, temperatures, gas composition, and so on. Of the 247 continuous variables, 49 represent the time t belief state and 198 represent the variables at time $t+1$. Of the latter, 49 variables are belief state variables for time $t+1$, 71 variables are *encapsulated* variables, as discussed in Section 6.2.6, 32 variables are observed variables, and the rest are transient variables. Within the 198 variables at time step $t+1$, the CPDs for 73 variables are non-linear CPDs. Examples for non-linear functions used in our CPDs include a multiplication function, a division function, a max function, and other more special-purpose functions such as the gas loop equations discussed in Section 10.2.3.

10.2.1 Sensor Modeling

As in any real system, the RWGS sensors do not record the underlying state exactly. In addition to some important quantities, such as the gas compositions, which are not measured at all, the existing sensors are noisy and biased. The noise level of the sensors depends on many factors and can change over time. An example is shown in Figure 10.3(b), where the difference in the readings of the pressure sensors P_3 and P_4 (both located at (2) in Figure 10.2) is plotted over time. The main reason for the noise in time steps 0–800 is the physical proximity of the sensors to the compressor, which sends pressure waves throughout the system. Since the sensors are not synchronized with the compressor, they take measurements at various phases of the pressure waves and thus measure significantly different values. After 796 seconds the compressor shuts down and the noise level decreases dramatically.[1] More interestingly, we note that the two sensors are placed very close together and therefore at least the average difference should be zero. However, as the plot demonstrates, this is not the case, indicating that the sensors are not well calibrated and some bias is present. Furthermore this bias depends on the system state, as shown by the change

[1]The sensor's noise is literally noise that can be heard — the pressure waves are the sound waves generated by the compressor.

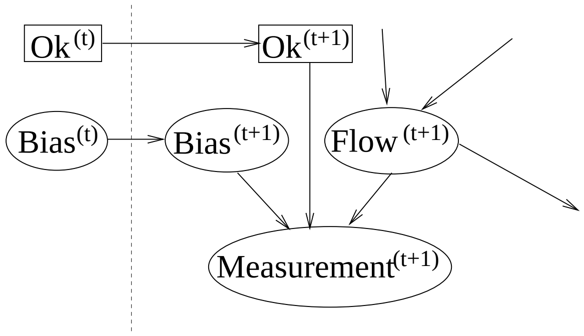


Figure 10.4: The sensor model

in the average difference when the compressor shuts off.

We deal with noisy sensors in the obvious way: by increasing the variance of the predicted measurement values to match the noise level in the data. Sensor biases present a more interesting modeling problem. The biases are not easily modeled using a simple parameter since they are unknown and can drift over time. Instead, we address the problem by adding hidden variables to the belief state that model the different biases of the sensors. Biases start with zero mean and a reasonably large variance and persist over time, i.e., the CPD for $Bias^{t+1}$ is a linear CPD of the form $Bias^{t+1} = Bias^{t} + V$. The variable V represents Gaussian noise with a relatively small variance, allowing for some amount of drift to occur over time. Figure 10.4 shows our sensor model, using a flow sensor as an example. The belief state contains two variables for every sensor: the status of the sensor (working or failed) and the bias variable. If the sensor is working then its reading at time $t+1$ is a sum of the actual quantity (the flow in Figure 10.4) and the sensor's bias.

This idea works quite well, but it tends to overfit the data: By letting the bias account for every discrepancy between the model predictions and the actual sensor measurements, the tracking algorithm might settle to an incorrect steady state. For example, assume we have some flow sensors each one with some bias. If we add some constant c to all the biases then all the flow measurements can be explained by the flows being smaller by c than their true value, i.e., if the flow sensor has bias c and it measures the flow F then we predict the actual flow to be $F - c$. This leads to a state that is almost consistent with the measurements (it is not fully consistent because other sensors may indicate some discrepancies) but might be very different from the actual state.

To fix the problem we must make sure that the model biases reflect true sensor biases — biases should be kept as small as possible and be allowed to grow only if there is a real reason. We implement this idea by introducing a contraction factor $\gamma < 1$ (empirically set to be 0.9) into the bias formula: $Bias^{t+1} = \gamma \cdot Bias^{t} + V$. Thus, biases tend to go to zero unless doing so introduces a systematic discrepancy with the predicted system state.

10.2.2 Sensitivity and Unidentifiability

One of the challenging aspects about the RWGS system is that some parts of the system state are very sensitive to input conditions, and yet are not measured directly by the existing sensors. As an example, consider the H_2 and CO_2 compositions in the gas loop. Under normal operating conditions, the gas flow into the reactor has equal amounts of H_2 and CO_2 added to it via the CO_2 line (1), the recycle line (12) and the H_2 supply (15). The reactor itself takes almost the same amounts of H_2 and CO_2 back out of the mixture. Thus, aside from a very small difference in these two quantities, the only changes in the compositions of H_2 and CO_2 in the gas loop arise from small amounts of H_2 and CO_2 that leave the system through the vent line (13). Therefore, the changes in the compositions of the gases are very slow. The system takes a long time to reach equilibrium, and the equilibrium itself is exceptionally sensitive to the input conditions and to the behavior of the membrane. To make matters worse, the gas compositions are not measured directly in real time. The reason is that the only existing sensors capable of making this sort of measurement are application-specific, and require specific types of lasers, complicated optical setups, and significant amounts of time from specialists in order to set the system up. Using such sensors is entirely impractical for a prototype system, and may not be feasible even for the actual system that needs to work autonomously on a different planet.

Even if we use the most exact form of the equations governing the H_2/CO_2 balance in the gas loop in our model, we will end up with (at least) the same level of sensitivity to slight variations in the input conditions. In addition, the model predictions will suffer from high sensitivity to inherent errors in the model such as small errors in physical parameters, approximate equations for complex components such as the separation membrane, and so on.

We therefore use equations for the H_2/CO_2 balance that contain an intentionally non-physical component — a *stabilizing* term — that reduces the sensitivity. This term drives the balance to a pre-determined point, which in this case is our expected value for the balance. In other words, the variable $C^{(t+1)}$, representing the composition of a certain gas at time $t+1$, is a weighted average $C^{(t+1)} = wM^{(t+1)} + (1-w)S$ where $M^{(t+1)}$ represents the composition predicted by the model and S represents the pre-determined stabilization term. The weight w was manually adjusted to provide a good tradeoff between physical accuracy and model stability and was set at 0.995, i.e., the prediction of the model contributes much more than the stabilizing term.

10.2.3 Differing Time Scales

The RWGS system contains different phenomena that manifest themselves over at least three different time scales. Pressure waves in the RWGS propagate essentially instantaneously (at the speed of sound). The time it takes for the gas flow to circulate around the gas loop is on the order of seconds. Finally, as we have just discussed, gas compositions in the gas loop take a very long time to reach a steady state (on the order of hours). Meanwhile, the sensors in the system collect data at a sampling rate of one second.

The naive approach is to model the DBN at finest time granularity, i.e., with less than 1ms per time step. Doing so enables us to model the pressure waves that go around the system, at least in principle. However this idea is not useful: If we model the system at time steps of 1ms we get one observation for every 1000 steps. Even if we can find a way to create an accurate model (which is very hard since the data is too sparse to be used effectively for training), without sensor data to correct our predictions, using the finer time granularity does not help us to achieve more accurate predictions. In other words, there is no point in modeling the system at a finer granularity than the sensor data.

Thus, we choose to model our DBN with time granularity of 1 second per time step. We approximate the pressure waves as occurring instantaneously and, instead of modeling their transient behavior, we model the quasi-steady-state of the system after the pressure waves traveled around the system and have reached an equilibrium. The equations in this case are substantially simpler, and require far fewer empirical constants.

The difficulty, however, is that a change in any part of the system will affect all the other parts at a much faster rate than our time steps, i.e., they happen instantaneously from the point of view of the model. In fact, this situation leads to cyclical influences between our variables. For example, the gas flow entering the membrane (16) influences the pressure there. A change in this flow would change the pressure at (16) causing an almost instantaneous pressure wave around the gas loop. This change in the pressures would change the gas flows in the loop, including the flow at (16). Thus, we have a situation where the flow influences the pressures which influence the flow. Since all these changes occur within one time step, we get a cycle in our DBN, which is not legal in our framework.

The solution is to solve the equilibrium equations of the gas loop simultaneously. There are five variables that represent the pressures and flows around the gas loop, denoted as $\boldsymbol{Y}$. The value of $\boldsymbol{Y}$ can be determined if we know the outside conditions denoted as $\boldsymbol{X}$, e.g., gas flows into the system, the pressure at the vent pipe (13), and so on. The equations to compute $\boldsymbol{Y}$ from $\boldsymbol{X}$ include both the compressor equation and an approximation to the membrane equations developed by Whitlow [Whi01]. They are fairly large and nonlinear, and no direct simultaneous solution form exists. Instead, we can solve these equations iteratively, as a set of fixed point equations.

How do we combine these fixed point equations with our DBN? We insert these fixed-point equations into a (nonlinear) CPD $\mathcal{C}$ in our DBN. Since we solve the equations simultaneously for the five variables $\boldsymbol{Y}$, $\mathcal{C}$ is in fact a vector CPD, used for all the variables $\boldsymbol{Y}$. The parents in $\mathcal{C}$ are the variables $\boldsymbol{X}$. The definition of the CPD is essentially procedural: given a value of the parents $\boldsymbol{X}$, it executes an iterative fixed-point computation until convergence, and outputs the values for all of its five variables $\boldsymbol{Y}$. Thus, we can view $\mathcal{C}$ as defining the non-linear function $\boldsymbol{Y} = f(\boldsymbol{X})$. The function has a complicated form, implemented as a C procedure in our code.

Approximating the distribution induced by $\mathcal{C}$ may seem like a daunting task. Indeed, had we been using the EKF algorithm (Section 6.1) this would have been a difficult task, since it is not easy to compute the derivatives of the function $f(\boldsymbol{X})$ represented by $\mathcal{C}$. However, if we use our numerical integration methods, and in particular the exact monomials method (Section 6.2.3) there is no difficulty in using a CPD such as $\mathcal{C}$. When using the exact monomials method we only need to evaluate the function f at some set of points, i.e., we need to solve the fixed point equations at some points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ determined by the numerical integration algorithm. Thus, we plug in our "procedural" CPD $\mathcal{C}$ into our numerical integration algorithm, and estimate the first two moments of the relevant variables. We can use the algorithm from Figure 6.2 almost unchanged — the only exception is that unlike other CPDs, $\mathcal{C}$ defines the moments of more than one variable.

10.3 Parameter Estimation

The model for the RWGS is quite complex and includes many parameters. It is useful to classify these parameters into different categories, as it helps to understand the different types of parameters involved in the model and the different methods used to estimate their values. We partition our parameters into seven classes:

1. Known physical constants
2. Unknown physical constants
3. Unknown physical quantities that change over time
4. Noise level parameters that are induced by the physical system
5. Noise level parameters that are induced by model inaccuracies
6. Persistence parameters
7. Model correction parameters

The first class is probably the simplest one. It involves constants such as the gravity of the earth and constants used for convergence between physical units. The convergence parameters are useful if we want to represent our variables using some convenient units, but the sensors are using some other units. For example, it may be more convenient to represent temperatures in Kelvin even though the sensors are in Celsius. Thus, when we model the sensor readings we must convert from Kelvin to Celsius. Since all the parameters in this category are known, they are simply set to their actual value.

The second class of unknown physical constants includes parameters such as the physical lengths of the pipes in the system (which we only have an estimate for), the rate in which the reactor heats up when the heater is on, the surface area of the separation membrane, and so on. Parameters in this class were estimated either by the NASA team (e.g., the length of the pipes) or from the data using least squares or other curve fitting techniques. For example, we plotted the cooling off curve of the reactor and tried to find a low order polynomial fit to this curve.

The third class includes parameters such as the resistance to the gas flow (which turned out to change over time) and the bias variables, discussed in Section 10.2.1. Since these quantities change over time they cannot be modeled as simple model parameters. Instead, we add them as hidden variables to the model with some initial distribution based on some heuristic. For example, the initial mean of all the bias parameters was set to zero and the initial resistance of the membrane was set to a rough estimate produced from data collected during steady state runs. We then let the inference algorithm take care of the rest, i.e., adjust the values of these variables over time.

The fourth and fifth classes correspond to the variances of certain variables in the model. The fourth class corresponds to noise that exists in the physical system. Examples include the noise levels of the sensors and the noise levels of various set values (e.g., the pressure valve located at (13) is supposed to set the pressure in the system to some pre-defined value X, but it may actually to set it to a value slightly different from X). These noise levels can usually be easily estimated from the data. For example, in order to estimate the sensor noise we can take short sub-sequences of the steady-state data where the actual measured quantity stays almost constant in each sub-sequence. We then compute the variance of the sensor readings in these sub-sequences, and use it as an estimate for the sensor's noise level. If the noise level of a certain sensor depends on some outside conditions such as the compressor working or not working, we perform the estimation procedure independently for each operating mode.

The fifth class accounts for model errors. For example, the equations that model the separation membrane are only an approximation to the actual membrane behavior, and thus, even if all the outside conditions are known exactly, there should still be some uncertainty on the predicted flows and pressures within the gas loop. Similarly, even if know exactly the temperature of the reactor and the gas composition, we can only give an approximate estimate of the efficiency of the reaction, and thus, there should be some uncertainty added to our predictions. Most of these parameters were estimated using "trial and error", i.e., we tried different values and chose the ones that led to the best results. Note that by adding these parameters to our model we artificially introduce more noise to compensate for the model inaccuracies.

The sixth class includes the persistence parameters for the discrete as well as the continuous variables. For example, we need a parameter that defines the probability of the sensor to work at time $t+1$ given that it was working at time t. An example for a persistence parameter that relates to continuous variables is the uncertainty over the sensor bias at time $t+1$, given that we know the sensor bias at time t. These parameters were estimated using various methods and then tuned using trial and error. For example, for the persistence discrete parameter we estimated how often sensors break or start working in the data that we had. We then used our training data to improve the value of this parameter, finally setting it to 0.9999.

The last class includes parameters for the various stabilization mechanisms that were added to the model. Examples for these parameters include the weight of the stabilization term described in Section 10.2.2 and the decay parameter γ of the sensor biases, described in Section 10.2.1. Again, these parameters were estimated using trial and error.

Parameter estimation turned out to be perhaps the most difficult task involved in modeling the RWGS. The reason is that there is relatively very little data and many parameters to estimate. Furthermore, the system has some modes of behavior for which we had no data, making the estimation of some of the parameters very hard. For example, we do not have data for the case that the H_2 flow into the system is turned off while there is still CO_2 flowing into the system. We feel that, at this point, coming up with a good model with good parameters is the single biggest problem that must be solved in order to use hybrid models for fault diagnosis in practice. We discuss this issue further in Section 11.2.3.

10.4 Experimental Results using Synthetic Data

We tested our algorithm with both real data and simulated data that was generated from our model. Although running with real data is the real test for our approach, running with simulated data is also of interest. The reason is that there are two sources of errors when using real data: model inaccuracies and errors induced by the algorithm. When using simulated data, only errors of the second type are present and we can better test the performance of the algorithm. In this section we present the results of the experiments with synthetic data. All our experiments were run on a Pentium III 700MHz.

10.4.1 Testing the Gaussian Approximation

Our first test was whether the belief state, as defined by the model, can be well approximated as a Gaussian, and whether our particular approximation algorithm that uses the numerical integration approach is a good one. To represent the belief state without approximating it as a Gaussian, we generated a set of samples from the model. We did not introduce any evidence so the samples were indeed sampled from the correct joint distribution.

In Figure 10.5 we show the results for two particular variables: the flow at point (16) and the pressure at point (2). These variables were chosen because of their

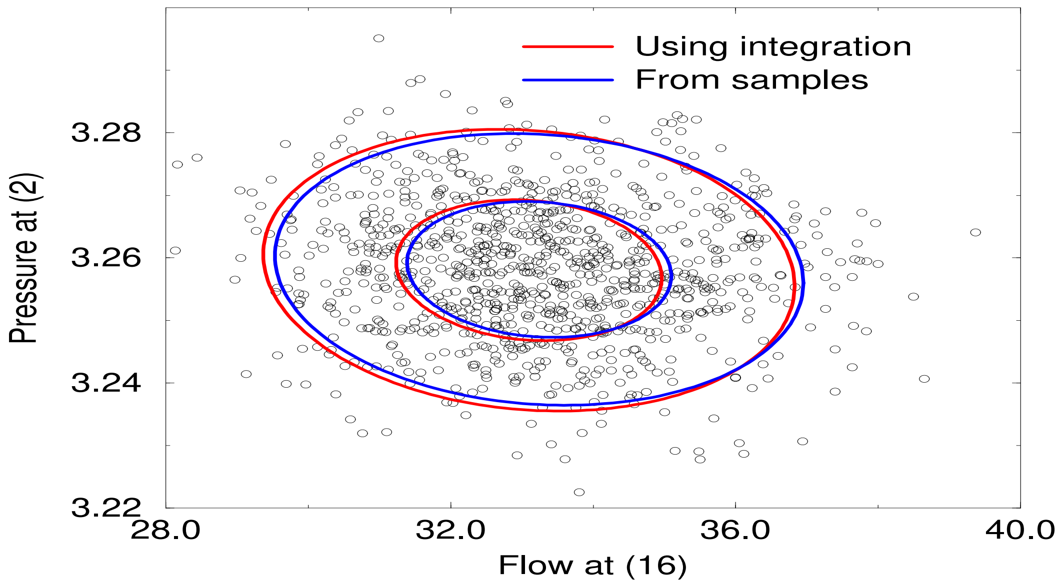


Figure 10.5: Random samples from the RWGS model and Gaussian estimates for the distribution

dependency on the non-linear CPD of the membrane; other variables produced similar results. The samples appear to be drawn from a distribution that is either a Gaussian or close to one. Furthermore, our estimate for the joint distribution, as depicted by the contours for one and two standard deviations, is very close to the Gaussian that was estimated directly from the samples. Thus, it is reasonable to expect that our techniques will lead to good approximations of the belief state.

10.4.2 Comparison with Particle Filtering

The most popular alternative to our inference algorithm is based on particle filters (PF) (Section 8.7) with or without the addition of Rao-Blackwellization. The reason we may want to use simple PF rather than RBPF is that the system is non-linear and when we use PF we do not impose any restrictions on the belief state — with enough samples we can represent any distribution, no matter how non-Gaussian it is.

To compare our algorithm with PF, we generated a trajectory of 500 time steps

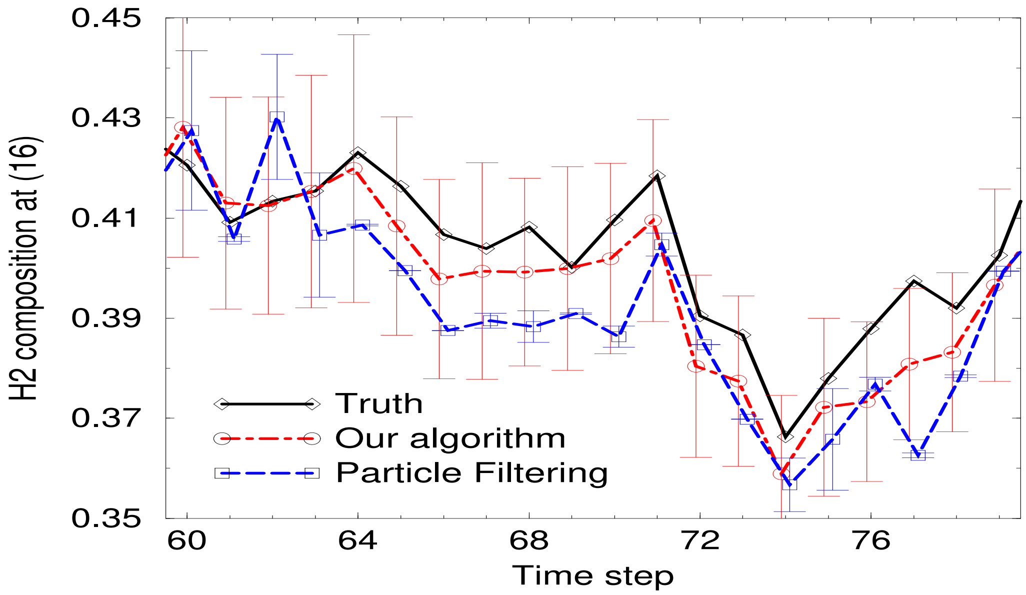


Figure 10.6: Comparison with particle filtering on simulated data

from our model and tested both algorithms on it. To make the task particularly simple, we assumed that all the discrete variables were observed — as we shall see, even this simple case was quite challenging for PF.

Our algorithm took 20ms per time step, which included computing the Gaussian approximation to the belief state, with numerical integration when necessary, and conditioning on the evidence. In comparison, generating a sample using particle filtering took 1.5ms. Thus, one step of our algorithm took as much time as generating 13 samples. However, with just 13 samples PF performed extremely poorly and therefore in our experiments we used 10,000 samples at every time step, giving PF a somewhat unfair advantage.

Figure 10.6 shows the estimates for the percentage of H_2 in the flow at point (16) that were computed by our algorithm and by particle filtering, as well as the actual value (known from the simulated data). We report the results on this particular variable because the gas compositions are not measured by any sensors and are therefore a potential challenge to any algorithm. The X axis represents time, and the Y axis the percentage of H_2 in the flow at point (16). To increase readability, we shifted the estimates generated by our algorithm by 0.1 to the left and those generated by particle filtering by 0.1 to the right. The error bars represent the uncertainty of the estimates as plus and minus two standard deviations. For our algorithm we simply used the variance of the Gaussian to compute the standard deviation. For particle filtering we computed the standard deviation using the variance induced by the weighted samples, i.e., if given a set of particles $\{\langle \boldsymbol{x}[1], w[1]\rangle, \ldots, \langle \boldsymbol{x}[N], w[N]\rangle$, the variance was estimated as $\sum_{i=1}^{N} w[i](\boldsymbol{x}[i] - \bar{\boldsymbol{x}})^2$ where $\bar{\boldsymbol{x}} = \sum_{i=1}^{N} w[i]\boldsymbol{x}[i]$.

As Figure 10.6 demonstrates, the estimates of particle filtering are not as good as the estimates of our algorithm. Over the entire sequence the average error of our algorithm was 0.009 while the average error of particle filtering was 0.013, although under our setup, particle filtering was slower than our algorithm by a factor of 750. Nevertheless, the more dramatic difference is in the estimates of the variance. Often, the estimated variance for particle filtering is extremely small, even when the estimated value is not very accurate (e.g., time steps 72 and 73). In fact, over the entire sequence, according to the estimated distribution of our algorithm, the correct value of the H_2 composition was within two standard deviations 96% of the time (this is consistent with the fact that the probability mass within two standard deviations from a Gaussian mean is 95%). In comparison, for particle filtering, the true value was within two estimated standard deviations only 20% of the time.

The difference was even more apparent when we considered the average log-likelihood of the true value given the two possible estimates for the belief state. Note that, for particle filtering, we could not estimate the log-likelihood directly from the samples, since the probability of the ground truth being exactly equal to a value in one of the samples is zero. Therefore, we used the samples generated by particle filtering to estimate the first two moments of the belief state distribution. We used these moments to approximate the distribution as a Gaussian and compute the log-likelihood of the ground truth. For our algorithm the average log-likelihood was 3.1 while for particle filtering it was only $-5.59 \cdot 10^{11}$.

The reason for this problem is the relatively high dimension of the evidence which leads to a very high variance for the weights of the samples. Although we generated

10,000 samples at each time step, only a very small number of them had a significant effect on the estimate. Over the entire sequence, in 65% of the time steps one sample accounted for more than 0.5 of the total probability mass, in 27% one sample accounted for more than 0.9 of the mass, and in 15% one sample accounted for more than 0.99. Obviously, in cases where one sample completely dominates the rest, the estimates of PF are not very reliable and in particular the variance estimates can be extremely small and misleading. Thus, not only is our algorithm faster than particle filtering with 10,000 samples by a factor of 750, its estimates are much more reliable.

10.4.3 Comparison with Rao Blackwellized Particle Filtering

The results from the previous section are a strong indication that RBPF should be preferred over PF in the RWGS domain. Therefore, we next compared the performance of RBPF to the performance of our enumeration algorithm. To make a direct comparison between the two algorithms, in both cases we used our algorithm for inference in hybrid DBNs from Figure 9.5, with the only difference being that in one case we implemented line 4 using the enumeration algorithm while in the other case we used RB-LW to generate the Gaussians.

The belief state was represented using a decomposed belief state as described in Section 9.4. We used three hypothesis variables, where the subsystems were modeled as independent of each other. The sub-systems corresponded to the following partition of the model:

- **The reactor subsystem** — this subsystem contained variables that describe the state of the reactor (area (3) in Figure 10.2), mainly variables that relate to the temperature of the reactor. This was the smallest subsystem with just 8 belief state variables (3 discrete and 5 continuous).
- **The water subsystem** — this subsystem contained all the variables that describe the state of the water tanks and the electrolyzer. Specifically, it involved the variables in areas (5)–(8), (12), and (14) in Figure 10.2. This subsystem contained 24 belief state variables (12 discrete and 12 continuous).

- **The gas loop subsystem** — this subsystem contained all the the other variables in the belief state, which correspond to the gas flows in the system. It was compromised of areas (1),(2),(4),(9)–(11), (13) and (15). This was the largest subsystem with 50 belief state variables (18 discrete and 32 continuous).

For the sake of having a fair comparison, we gave the two algorithms the same amount of time to generate the Gaussians: 3 seconds for the gas loop subsystem, 0.7 seconds for the water subsystem and 0.3 seconds for the reactor subsystem — we allocated more time to the gas loop subsystem since it is by far the largest one.

We ran a few synthetic trajectories, each one corresponding to some specified sequence of faults. We used two methods to evaluate the algorithms. The first was to examine if the algorithm converged to the correct discrete hypothesis and how quickly it did so. The second was comparison with the omniscient Kalman Filter, described in Section 9.3. The omniscient Kalman Filter gets to observe all the discrete variables; we cannot expect any algorithm which does not get to observe the discrete variables to perform as well as the omniscient Kalman Filter, but we can evaluate the quality of an algorithm by how close it gets to the omniscient Kalman Filter.

The no faults sequence

Our first sequence was the simplest possible sequence, where no faults occurred. In this case both RBPF and the enumeration algorithm performed well — both gave a very high probability to the no-fault hypothesis, and the distribution over the continuous variables in the belief state was almost indistinguishable from the omniscient Kalman Filter.

Turning off the heater

Our second sequence involved a single fault: at time step 11 the reactor heater (3) was turned off. Figure 10.7 shows the results of tracking the variable T_R, representing the temperature of the reactor, using both the enumeration algorithm and the omniscient Kalman Filter (for reference, we also plot the actual value of T_R, which was known

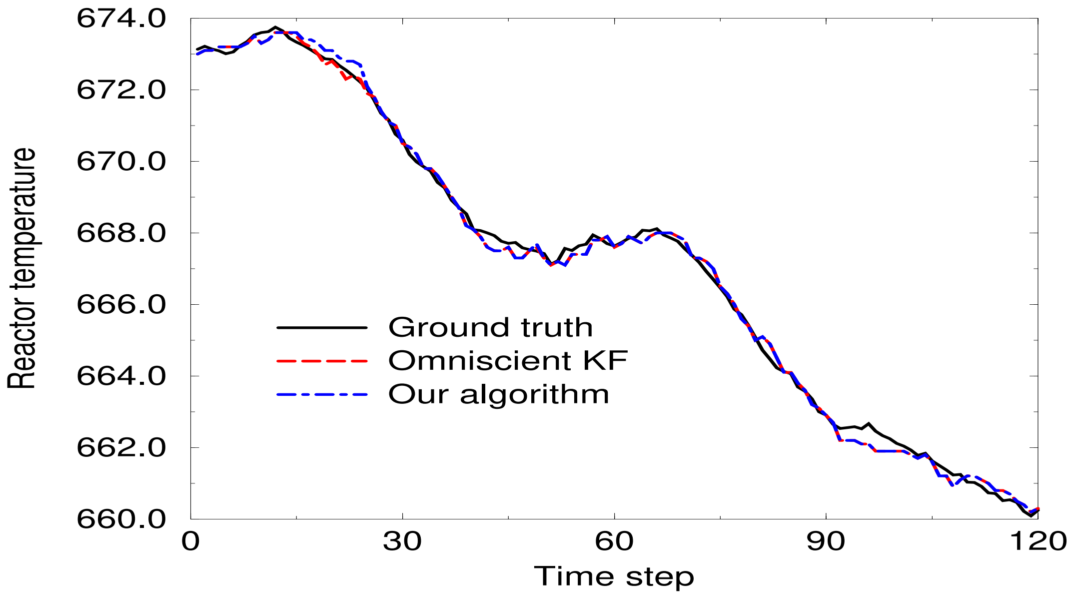


Figure 10.7: Heater shutdown: the enumeration algorithm and the omniscient KF

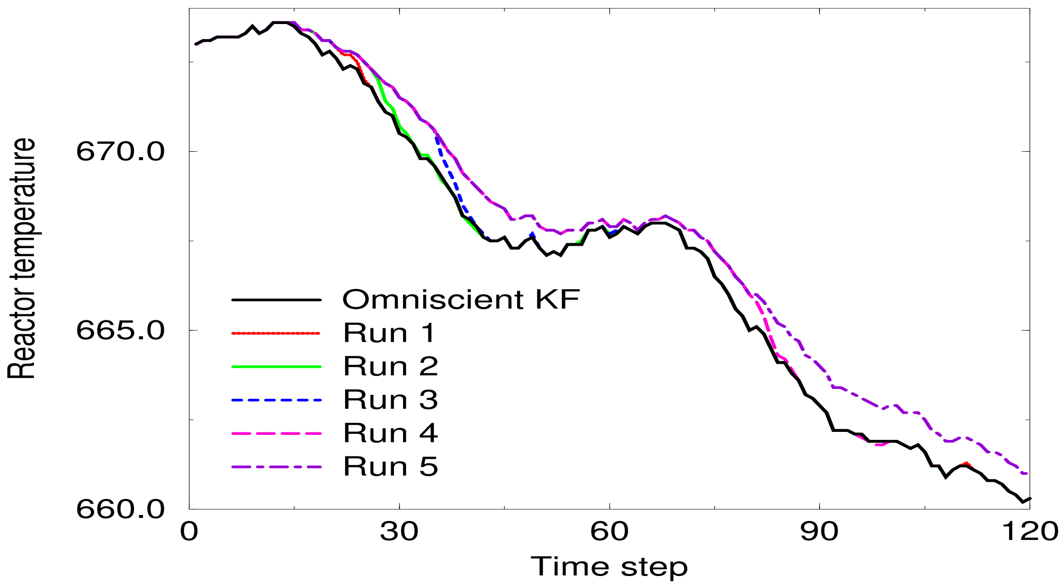


Figure 10.8: Heater shutdown: RBPF and the omniscient KF

from our synthetic data). To create the plots, we ran the algorithms and at every time step computed the mean of T_R from the belief states.

Up to time step 14, the estimates for the mean of T_R, as computed from the belief states of the enumeration algorithm and the omniscient Kalman Filter, are the same. At time step 15 the two estimates diverge but become identical at time step 26 and remain so until the end of the sequence (even when looking at closeups of the graph).

This behavior can be best understood when considering Figure 10.9 which shows the probability $P(\mathit{Heater}^{(t)}{=}\mathit{Off})$ as computed by the enumeration algorithm. Initially, the probability of the heater being off is very low. It remains so even after the heater is turned off at time step $t = 11$. The reason is that the initial drop in the reactor temperature is better explained as noise (either in the sensors, the model equations, or the physical system itself) rather than a fault of the heater device. However, after a few seconds, when the temperature consistently keeps dropping, the hypothsis of the heater being off becomes more and more likely: $P(\mathit{Heater}^{(26)}{=}\mathit{Off}) \approx 0.93$ and after that the probability is almost 1. Thus, between time steps 11 and 25 the enumeration algorithm "believes" in the wrong hypothesis of the heater being on, and therefore the estimated mean of T_R in these time steps does not agree with the omniscient Kalman Filter.[2] In other words, early in the run and after time step 25, the enumeration algorithm gives a very high probability to the correct discrete hypothesis. Since the omniscient KF knows the correct discrete hypothesis, they agree on the belief state. When the enumeration algorithm does not know the correct discrete hypothesis, it disagrees with the omniscient KF.

We ran the same experiment for RBPF, where we generated 10 different runs of RBPF on the same sequence. All the RBPF runs behaved initially like the enumeration algorithm — they were all identical to the omniscient KF until $t = 14$ and then diverged. However, different RBPF runs disagreed with the omniscient KF for different periods of time. Figure 10.8 shows the trace of five RBPF runs: the first agrees with the omniscient KF from time step $t = 26$ on (just like the enumeration algorithm) and the last does not agree with the omniscient KF even at time step 120

[2] In fact the estimates still agree until $t = 14$. The reason is that the model contains a built-in delay between the time the heater is turned off (or on) and the time the reactor temperature starts to change as a result.

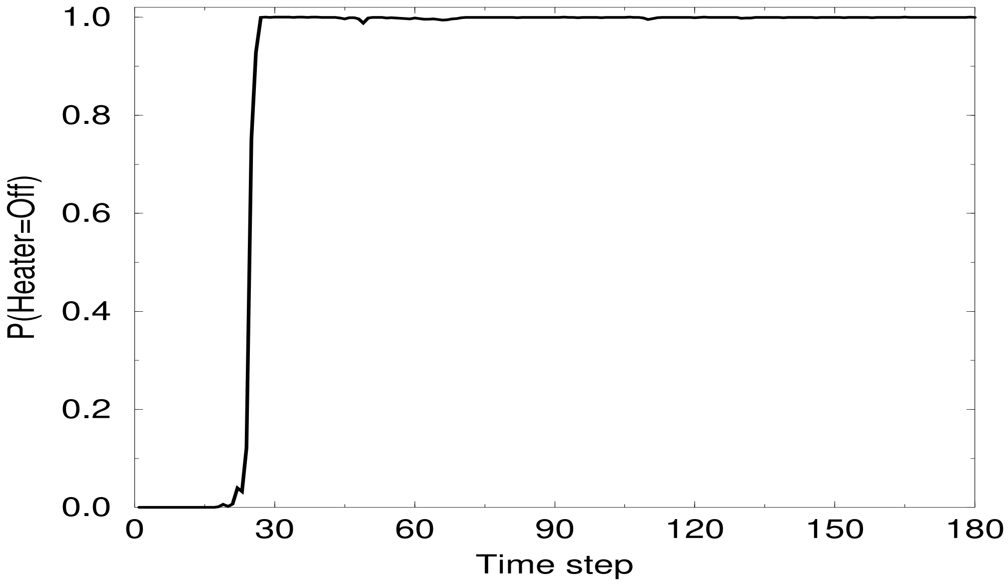


Figure 10.9: Heater shutdown: hypothesis likelihood (enumeration algorithm)

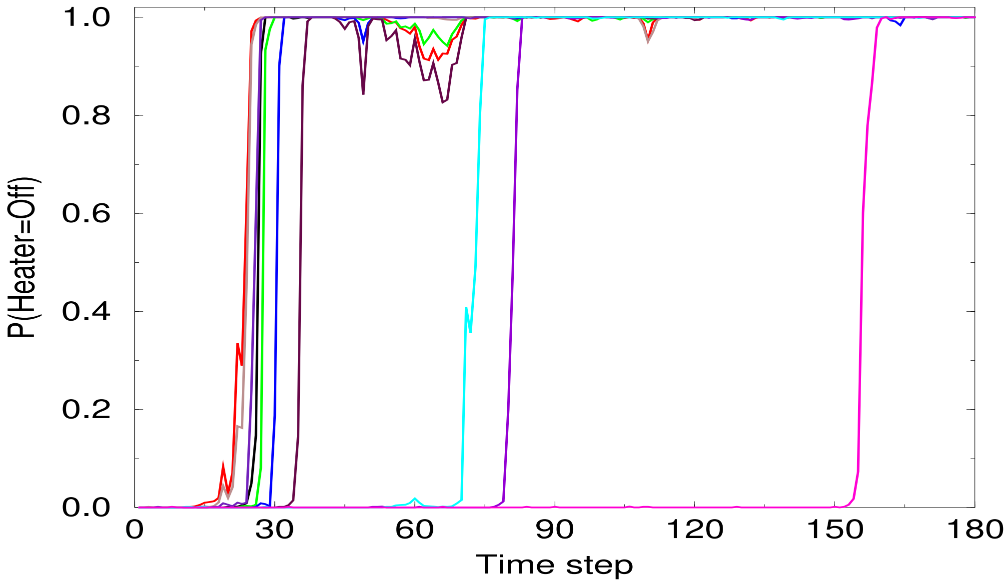


Figure 10.10: Heater shutdown: hypothesis likelihood (RBPF)

(it finally does converge to the omniscient KF at time step 160, which is not shown in order to make the graph more readable).[3]

Once again, the behavior can be best understood from the likelihood of the correct discrete hypothesis, shown in Figure 10.10. The graph plots $P(\mathit{Heater}^{(t)}{=}\mathit{Off})$ for 10 different runs of RBPF. In different runs the algorithm discovers that the heater is off at different time steps. In the best runs, the algorithm discovers the state of the heater around $t = 26$, just like the enumeration algorithm. In the worst run, the belief state indicates that the heater is on as late as $t = 155$.

To understand the behavior of RBPF, we must remember that every change in the temperature in a single time step can be explained as noise. It is the persistent drop of the temperature over a few time steps that indicates that the heater is off. Thus, in order for the algorithm to discover that the heater is off, it needs to first generate this hypothesis and then to propagate it over a few time steps. Both of these conditions are problematic for RBPF. Recall that the discrete persistence factor was set to 0.9999, and therefore almost all the samples are duplicate samples where the heater is on. Only a small fraction of the samples represent the *Heater=Off* hypothesis, and in fact there is no guarantee that such a hypothesis would be generated. Even if the correct hypothesis is generated, its weight would still not be significantly larger than the weights of the hypotheses where the heater is on, since the evidence of any single time step can be explained as noise. Thus, even if we have the *Heater=Off* hypothesis in our belief state for time t, we may not re-sample it when we propagate the belief state to time $t{+}1$, in which case the *Heater=Off* hypothesis would be lost. Intuitively, the low probability of generating the hypothesis representing the fault makes RBPF (at least in its standard version) an unreliable algorithm for fault diagnosis.

There is another interesting phenomenon in Figure 10.10. Around $t = 65$, 3 of the 7 runs that discovered that the heater is off significantly reduce the probability of the heater being off, while the 4 other runs keep the probability very close to 1. We believe that the reason for this is the variability of the sampling process, both in the hypotheses that were generated and in the number of duplicate samples per

[3]Interestingly, even without knowing the correct discrete hypothesis the mean of the variable does not diverge too much, since the measurements make sure that the temperature estimate stays close to the actual value.

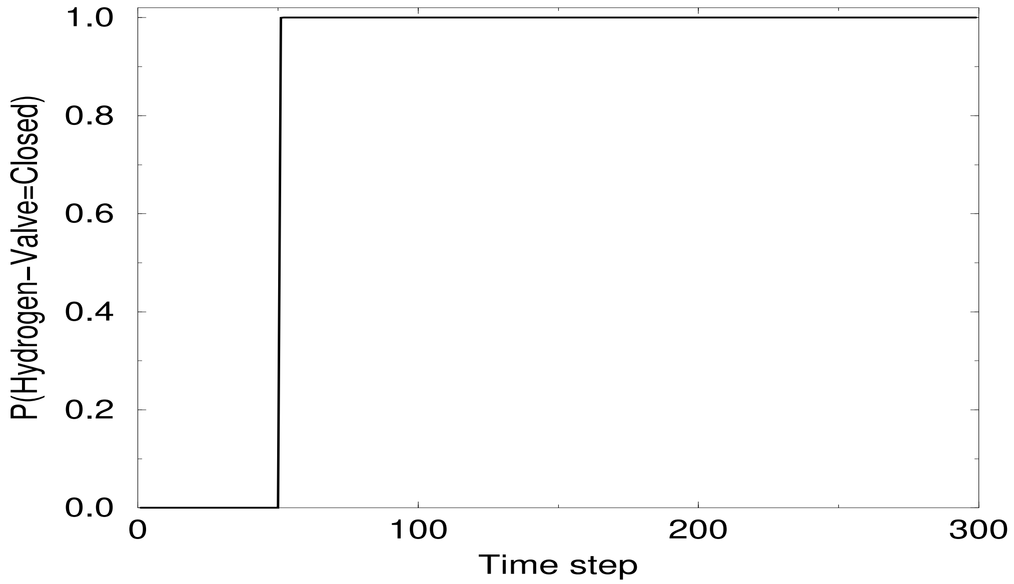


Figure 10.11: Hydrogen shutoff: hypothesis likelihood (enumeration algorithm)

hypothesis. Note that the problem does not exist in Figure 10.9 describing the behavior of the enumeration algorithm. This is another instance where the enumeration algorithm is more robust than RBPF.

Shutting off the hydrogen flow

Our third sequence involved a fault of the hydrogen valve (15), simulating the valve getting closed at time step 51. In this sequence we let the algorithms use the information from the flow sensor R_1 located on the hydrogen pipe. When the valve gets closed, R_1 indicates that the flow is approximately zero, and therefore this fault should be easier to diagnose than the heater being turned off.

Figures 10.11 and 10.12 show the performance of the enumeration algorithm and RBPF in this case. The enumeration algorithm diagnoses this fault as soon as it happens. On the other hand, the performance of RBPF varied in different runs. In the best run, RBPF diagnosed the fault in time step 51, just like the enumeration algorithms. In other runs, the fault was not diagnosed until much later in the sequence

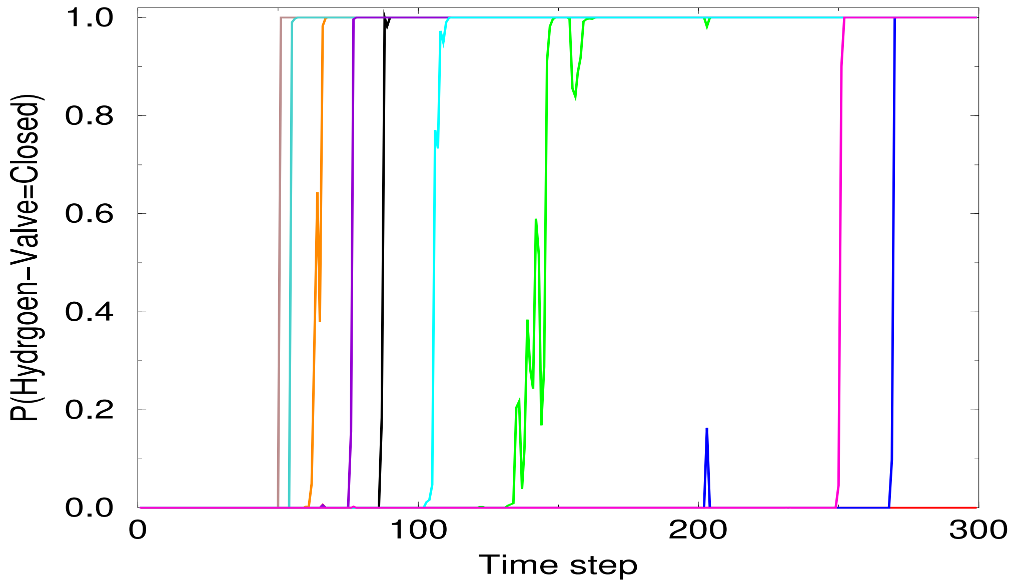


Figure 10.12: Hydrogen shutoff: hypothesis likelihood (RBPF)

— in one of the 10 runs the fault was not even diagnosed after 250 time steps. These results are very similar to the results we have seen for the heater example. Again, the reason is that because of the high discrete persistence the correct hypothesis is not always sampled at the right time.

In this particular sequence there is another possible way to explain the sensor readings, namely R_1 itself can be broken. This hypothesis is less likely than the valve being closed since a broken sensor can show any flow value and the fact that it shows 0 is likely not to be a coincidence and indicates that the flow has indeed stopped. For this reason the enumeration algorithm chooses the *Valve=Closed* hypothesis — the probability of R_1 being broken goes up to 0.00019 at time step $t = 51$ and drops back almost to zero afterwards. On the other hand, one might suspect that RBPF may consider the broken sensor hypothesis to be quite likely in runs where the closed valve hypothesis was not generated. Indeed, Figure 10.13 shows that this is the case. Here we plot the probability of R_1, the hydrogen sensor, to be broken for four runs of RBPF in which the closed valve fault was generated relatively late in the sequence.

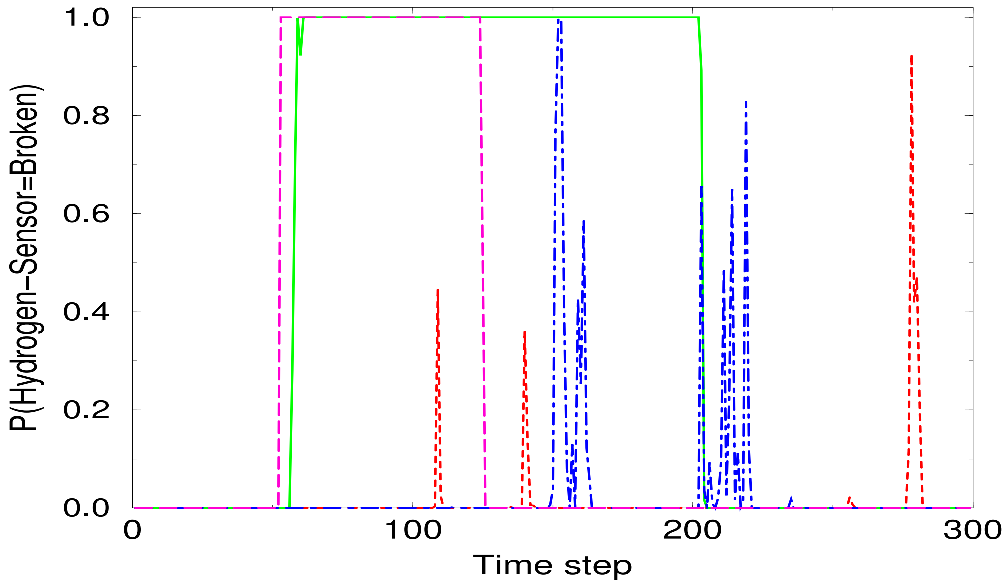


Figure 10.13: Hydrogen shutoff: probability for a broken sensor (RBPF)

Sensor Faults

Our next trajectory involves a related fault to the hydrogen valve fault, namely the flow sensor R_1 failing at $t = 10$. Sensor faults are quite easy to diagnose in our model, at least in principle. We model a failed sensor using a normal distribution with a large standard deviation of $\sigma = 100$, i.e., the distribution is $\mathcal{N}(0, 10000)$. In our trajectory, after giving readings which were consistently close to 1, at $t = 10$ R_1 suddenly generated the reading -27.3. This behavior can only be explained as a sensor fault, and indeed the enumeration algorithm had no problem with this fault, as it estimated $P(R_1^{(10)} = \mathit{Failed}) \approx 1$.

RBPF, on the other hand, had serious problems with this sequence. The faulty sensor hypothesis was usually not generated at $t = 10$. However, in this case, not generating this hypothesis was worse than in previous examples. In our model the noise level of R_1 when working has a standard deviation of 0.1. Thus, under the working sensor hypothesis we expect to get a reading from $\mathcal{N}(1, 0.01)$. The reading

-27.3 is 283 standard deviations away, and therefore under the working sensor hypothesis the probability of this evidence is astronomically low, or more precisely it is $\frac{1}{\sqrt{2\pi 0.1}}\exp(-0.5\cdot 283^2)\approx 10^{-17391}$. This number is too small to be represented using the standard 64 bits double representation for real numbers, and therefore is rounded to 0. Thus, in effect the likelihood of every hypothesis which does not include the sensor fault is zero.

In order for the propagation algorithm to be well defined, we must generate at least one hypothesis with a positive probability, otherwise the inference algorithm would not be able to continue. However, when using RBPF in our model with the discrete persistence factor of 0.9999, we are likely not to generate any faulty sensor hypothesis, in which case the probability of all the hypotheses is zero and the algorithm fails to continue. It is important to emphasize that the value of the discrete persistence factor was tuned to be a realistic model and to produce the best fault diagnosis results, and not to make the model easy or hard for a certain inference algorithm.

Going back to the enumeration algorithm, we observe that, while it can be reliably used to diagnose a fault of a single sensor, at some point the algorithm will break down. For example, if all the sensors fail simultaneously, the enumeration algorithm will not generate the correct hypothesis in a reasonable amount of time, since it is the least likely hypothesis.[4] Thus, in this case all the hypotheses generated will have a zero likelihood and the algorithm would fail, just like RBPF with a single sensor. The question is how far we can push the enumeration algorithm, i.e., how many simultaneous faults it can reliably diagnose.

Using the optimization techniques described in Section 5.1.6 we were able to generate about 320 Gaussians per second in the gas loop subsystem and about 1350 Gaussians per second in the heater and water subsystems. The gas loop subsystem contains 18 discrete variables, which is more than any other subsystem and is therefore the bottleneck in generating simultaneous-faults hypotheses. Using 3 seconds per

[4]This is true if all the faults are independent. However, note that when using DBNs we can model a system where certain faults are actually likely to happen simultaneously, e.g., because of some common cause. In this case, the enumeration algorithm will generate the hypothesis that corresponds to the the faults happening simultaneously before it generates the hypotheses that correspond to each fault happening by itself.

time step in the gas loop subsystem, the enumeration algorithm was able to generate all the double-fault hypotheses and a relatively small portion of the triple-fault hypotheses. Thus, under the setup we used in our experiments, our algorithm can diagnose two independent simultaneous sensor faults in real time, but not three.

Note that, if we are requested to diagnose more than two simultaneous faults, we can do several things. One option is to use a faster computer that can generate more Gaussians per second. Another option is to divide the gas loop subsystem into smaller subsystems, each one containing less discrete variables. Finally, we may be able to improve the inference algorithm — we discuss some ideas in Section 11.2.1.

On the other hand, if the faults happen sequentially then our algorithm can still be used, even if the faults happen at consecutive time steps. Our final synthetic trajectory describes a catastrophic scenario where we lose five flow sensors in just three time steps: at $t = 10$ the flow sensor R_1, located at (15), fails. At $t = 11$ the two flow sensors R_8 and R_{12}, located at (16) and (10) fail. Then, at $t = 12$, the two flow sensors R_9 and R_{10}, located at (13), fail. After these three time steps we have lost almost all the flow sensors in the gas subsystem, except for R_3, the flow sensor that measures the CO_2 flow, located at (1). Our algorithm had no problems diagnosing these faults and the relevant probabilities jumped from 0 to 1 at the appropriate time steps.

As a final challenge, the sequence included an H_2 shutoff at $t = 20$. Note that, by this point of the sequence, there are no working flow sensors that can give us useful information to diagnose the fault — the only working flow sensor at (1) continues to report the same amount of CO_2 flow going into the system. Thus, our only information comes from other sensors in the system, such as the pressure sensors in the gas loop, making the diagnosis task quite hard.

Figure 10.14 plots the likelihood of the hydrogen shutoff hypothesis as estimated by our algorithm. The algorithm reaches the correct conclusion, albeit after a fairly long time, which is to be expected given how little sensor data is available in the sequence. This can also be seen in Figure 10.15, which shows the actual flow at (16) combined with the estimates of the omniscient KF and our algorithm. As usual, our algorithm diverges from the omniscient KF as long as it gives high likelihood to

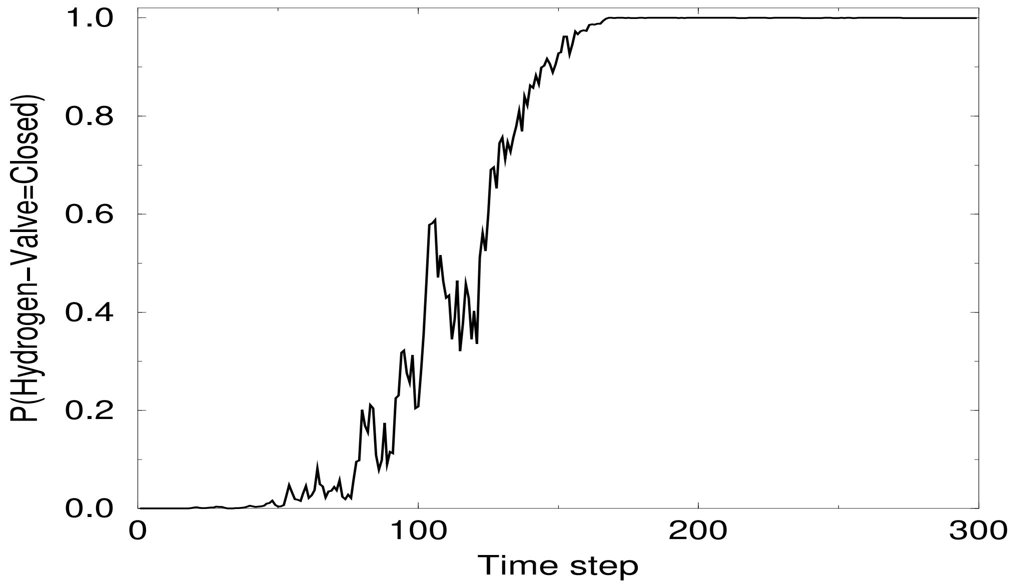


Figure 10.14: H_2 shutoff with broken flow sensors: hypothesis likelihood

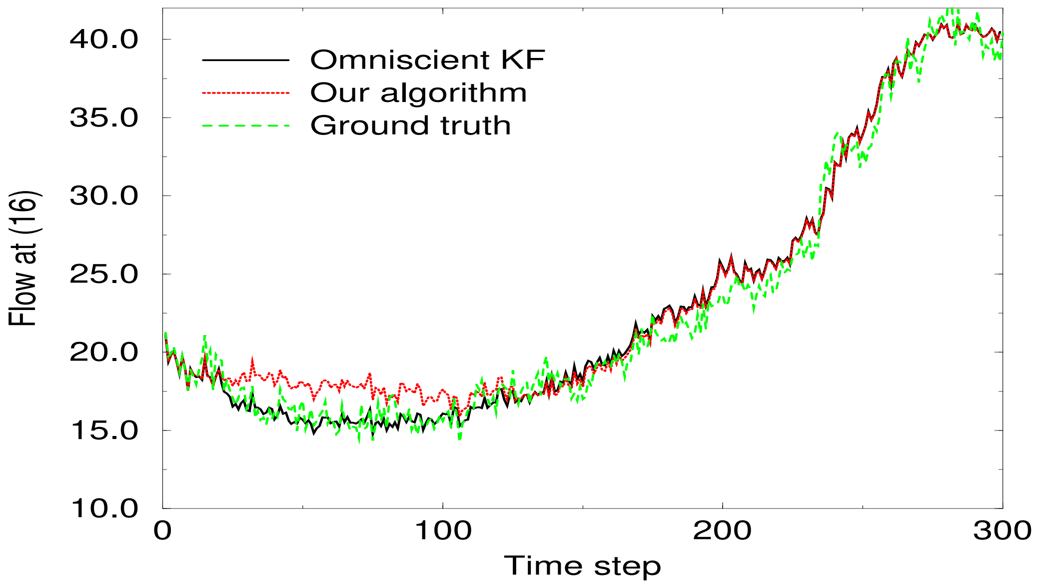


Figure 10.15: H_2 shutoff with broken flow sensors: flow estimates

the wrong discrete hypothesis (of no H_2 shutoff), but when our algorithm infers the correct discrete events it converges to the omniscient KF, and behaves like it from that point on.

Figure 10.15 also indicates that the ground truth trajectory sampled from the model is quite noisy, and in fact is much noisier than the actual data (we present some of the data in Figure 10.20 in the next section). The reason is the extra noise added to our model to compensate for its inaccuracies, as discussed in Section 10.3 When sampling a trajectory from our model, we use the adjusted noise levels, thus creating more jagged trajectories.

10.5 Results on Real Data

We now turn our attention to experiments that were run on real data collected for us by the KSC team. In the data that was available to us, the electrolyzer and the water pumps were not working, and therefore, although the water subsystem was a part of our model, it was never tested against real data. Its effect was mostly making the model as large as it should be.

Some of our data sets recorded the sensor readings at a frequency of once every 10 or 60 seconds. Since our model has a time step of 1 second, those data sets were only of limited use to us and were used solely for model estimation. Within the remaining data sets, only two were taken while the gas loop was operating with the compressor — in the rest the compressor was off and all the gas flow went out of the system through the ventilation pipe (13).

Thus, all the experiments in this section are reported on two data sets. The first was a long sequence of 4000 seconds that included 3430 seconds of steady state operation and 570 seconds of a CO_2 shutoff transition. The second included 600 seconds, out of which just 30 seconds were steady state and the other 570 seconds were a CO_2 shutoff transition. All our steady state results are reported on the first data set. In order not to use the same data for both training and testing the model, we used only 100 seconds of the steady state sequence for training, and the rest was used only for testing. For the CO_2 shutoff transition we treated the first data set as

Sensor		$t = 1$	$t = 2$	$t = 3$
Flow at (16)	R_8	0.001278	0.00001	0.000001
Flow at (13)	R_9	0.00064	0.000001	0
Flow at (13)	R_{10}	0.000593	0.000001	0
Flow at (10)	R_{11}	**1**	**1**	**1**
Flow at (10)	R_{12}	0.000705	0.000008	0.000001

Table 10.1: Probabilities of the flow sensors around the gas loop to be faulty at time steps 1–3

our training set and the second data set as our test set.

10.5.1 Steady State Experiments

Our first set of experiments with real data was performed with the steady state data. First, we tested the capability of the model to recognize faulty sensors. The data supplied to us by KSC actually included a faulty sensor, namely the flow sensor R_{11}, located at (10). In this experiment we initialized all the sensors (including the faulty one) to have a probability of 0.8 to be working and 0.2 to be broken.

As indicated by the synthetic data, our results confirmed that sensor faults are easy to diagnose. Table 10.1 shows the probabilities of the flow sensors around the gas loop to be faulty after 1,2 and 3 time steps. Indeed, our inference algorithm believes that all the sensors are working except for R_{11}.

In general, our model worked as expected on the steady state data with the correct discrete hypothesis always having a very high probability, and the continuous quantities being tracked well. To make the steady state data more challenging, we removed the data collected from the flow sensors R_1 and R_3 that measure the incoming H_2 and CO_2 flow at (15) and (1). Without these sensors, we have no measurements of the gas flowing into the gas loop, making various diagnoses harder. In particular, diagnosing whether the H_2 and CO_2 valves are open or closed becomes a more difficult problem.

Figure 10.16 shows the results of running our model on steady state data without R_1 and R_3. Here we plot the probability of both valves being open, which is the correct discrete hypothesis. With the exception of a small drop at the beginning of the sequence, the probability is practically 1 throughout the entire sequence (note

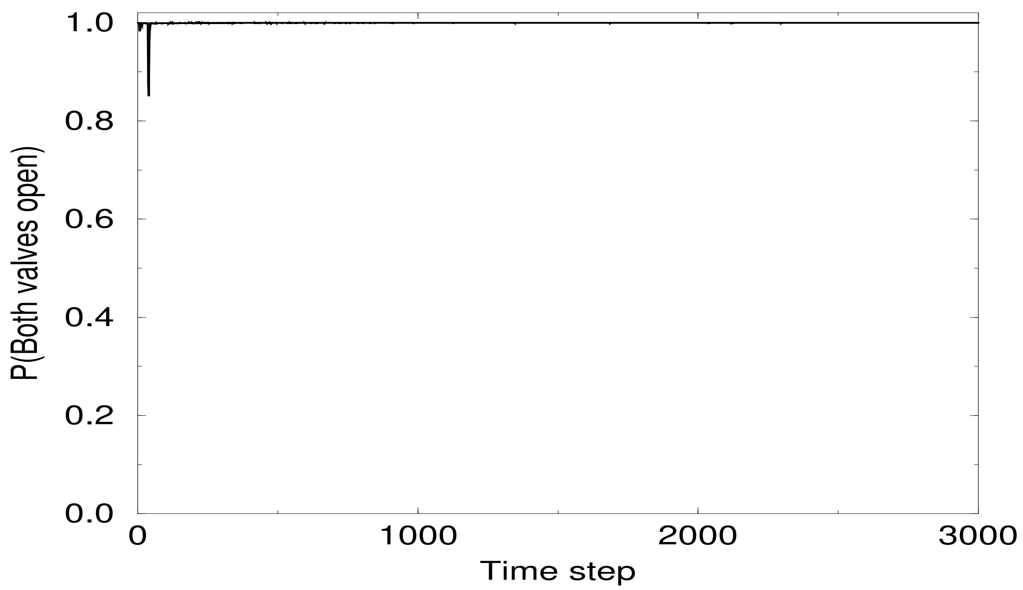


Figure 10.16: $P(\mathit{Valves\ open})$ plotted over 50 minutes of steady state data

that this the sequence represents 3000 seconds, i.e., 50 minutes in which the system was in steady state).

To better understand the drop at the beginning of the sequence, Figure 10.17(a) shows just the first 150 time steps of the sequence. Our conjecture is that the drop is caused by our initial conditions which do not represent the correct initial distribution. This is caused by a few factors:

- For some quantities in the model which are not directly measured, such as gas compositions, we used only rough estimates, which were probably somewhat wrong.
- Our initial values for all the sensor biases were zero, which we know to be wrong.
- We do not model any correlations between the variables in our initial belief state, i.e., we just defined a mean and a variance for each variable, setting all the covariances to zero. Again, we know this to be incorrect.

To test whether the initial drop is indeed caused by an incorrect initial belief state,

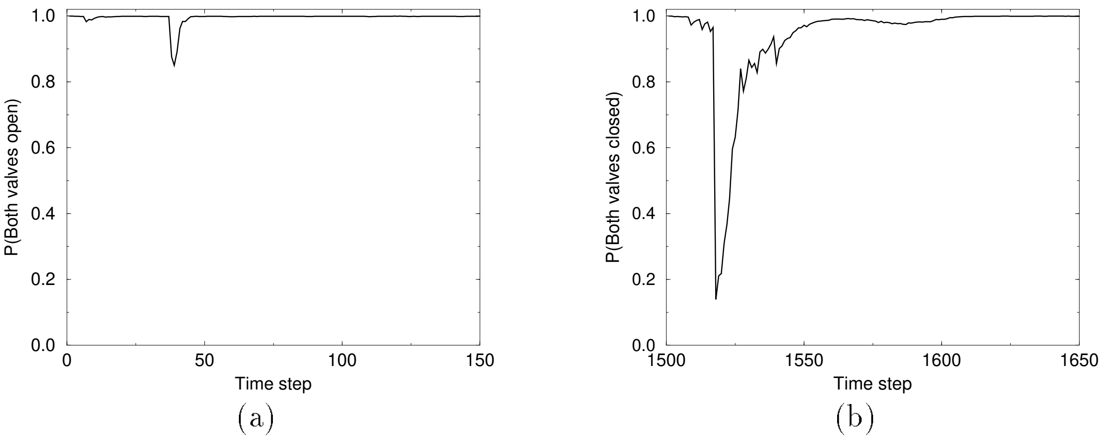


Figure 10.17: (a) The first 150 steps of the original sequence (b) First 150 steps of a sequence starting at $t = 1500$

we used the same sequence but this time started it at time step 1500. We modified our initial distribution to reflect the situation at $t = 1500$, but again we knew that we only have an approximation of the correct initial belief state. The results of this experiment are shown in Figure 10.17(b). Note that again the probability of the correct hypothesis drops at the beginning of the sequence before going back up. Since the original experiment had no problems around $t = 1500$, we view this as an indication that the problem is with the initial conditions. Clearly, if the system is to be used in practice it is necessary to have a better approximation of the initial belief state, although the task of estimating the belief state at the very beginning of the sequence, before the heater is turned on and there are any gas flows in the system, may be significantly easier.

Since in general the model worked well for steady state, we used the data to test various aspects of our model. Here, we report on experiments that test our sensor model. Recall that our sensor model includes two important components — the sensor noise and the sensor bias. We tested the effect of each one of these components.

We first demonstrate the effect of the noise level of the sensors. The standard deviation of the flow sensor noise was estimated from the data to be 0.1. We tried

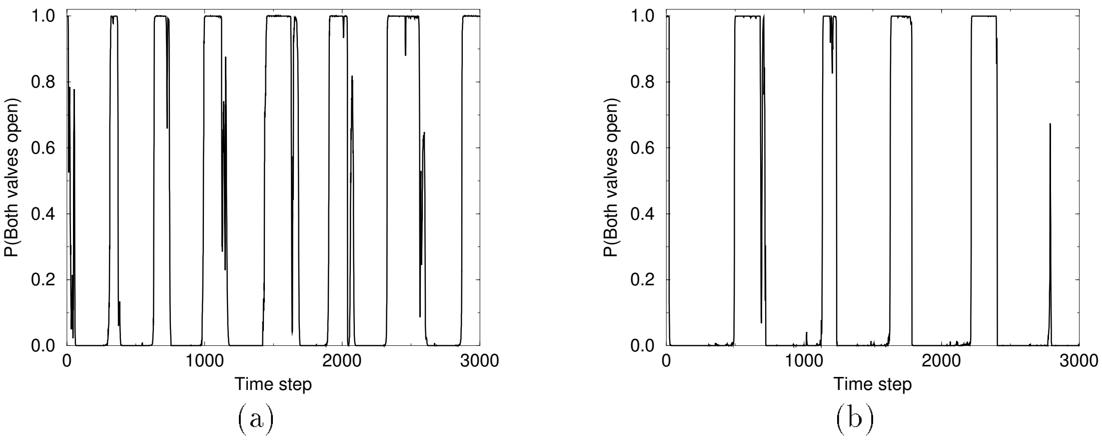


Figure 10.18: Wrong flow sensor noise level (a) Standard deviation too large: $\sigma = 0.4$ (b) Standard deviation too small: $\sigma = 0.01$

both to increase and decrease the standard deviation to 0.4 and 0.01, again not using the data from R_1 and R_3. In Figure 10.18 we plot the probability of both the CO_2 and H_2 valves to be open as estimated by the modified models.

It is clear that the performance for the different noise levels is quite poor and the predictions are not robust. When the noise level is too small the model is too sensitive to the noise in data, causing it to change the discrete hypothesis in order to explain the data noise. When the noise level is too high the model is too insensitive to the data — it does not adequately use the flow sensors readings to correct its predictions. The belief state becomes only a poor estimation of the state of the system, again leading us to choose the wrong discrete hypothesis. Thus, it is important to model the correct noise level of the sensors, and bad estimates for this parameter can lead to a serious degradation in the quality of the model's predictions.

Next, we set the standard deviation of the noise level back to 0.1, but removed all the bias variables of the flow sensors from our model. We used the same setup of removing the readings from R_1 and R_3 and plotting the probability of both valves to be open. The results are shown in Figure 10.19, again showing a dramatic degradation in the performance of the algorithm. Intuitively, without the bias variables, the belief

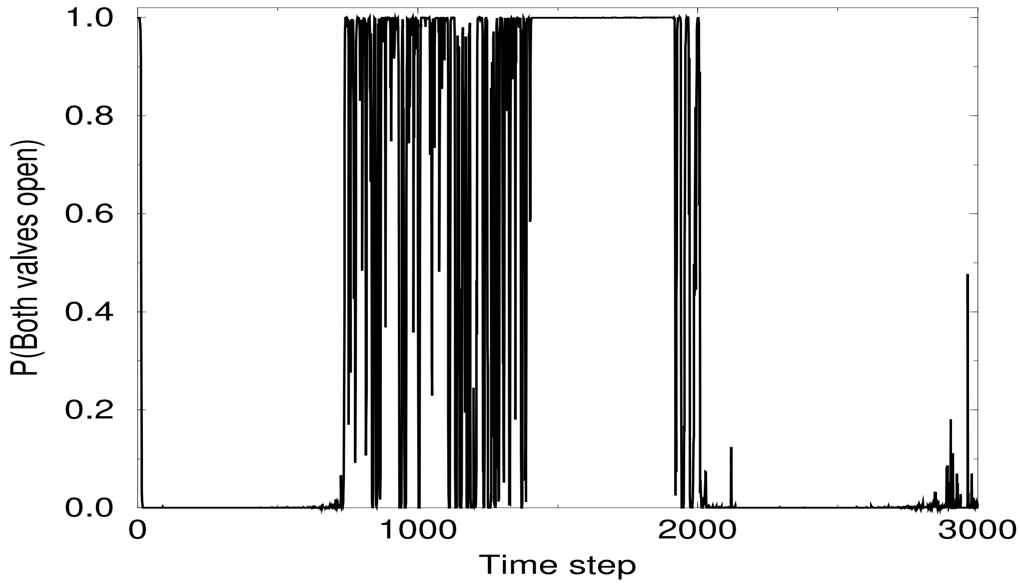


Figure 10.19: Modeling the flow sensors without the bias variables

state is not a reliable estimation of the state of the system. For example, consider the hidden flow variable F at (16). Without a bias variable for R_8, the flow sensor at (16), we force the value of F to be close to the value measured by R_8. Thus, if R_8 has a significant bias, the value predicted for F would not be close to the actual flow, and our predictions would not be reliable.

10.5.2 CO_2 Shutoff Experiments

Our last set of experiments involves the CO_2 shutoff sequence. We treat the shutoff as an unknown event, i.e., a fault, that we need to diagnose. All the results in this part were generated using the test set. We did not adjust our model for the test set, except for modeling the initial conditions at the beginning of the sequence.

To better understand our results we begin by examining some of the data that was generated in this sequence. Figure 10.20 shows the readings recorded by R_8, the flow sensor at (16) that measures the incoming flow into the membrane, and by R_{12}, the flow sensor at (10) that measures the outgoing flow from the membrane. Figure 10.21

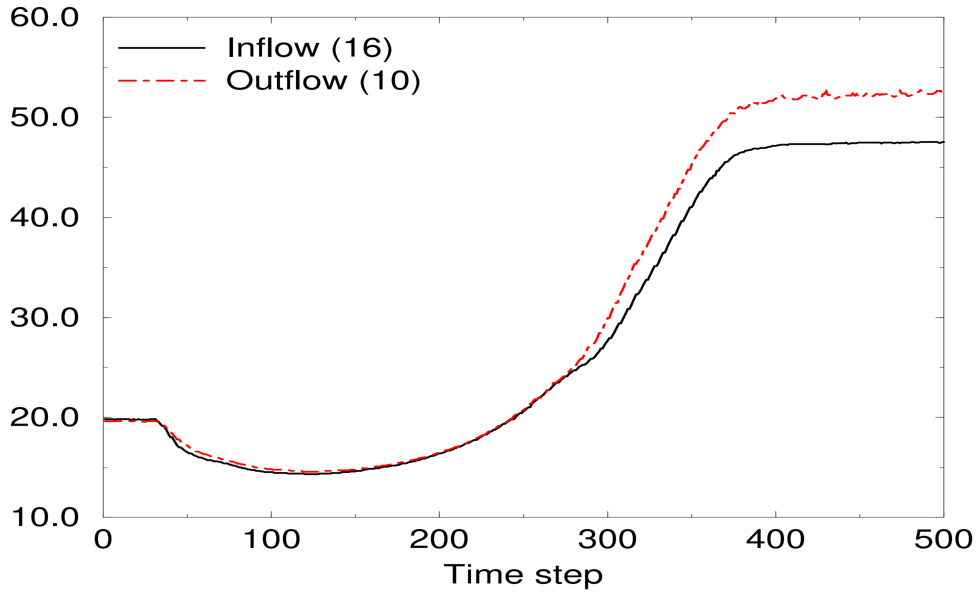


Figure 10.20: Flows at (16) and (10) during a CO_2 shutoff: the entire sequence

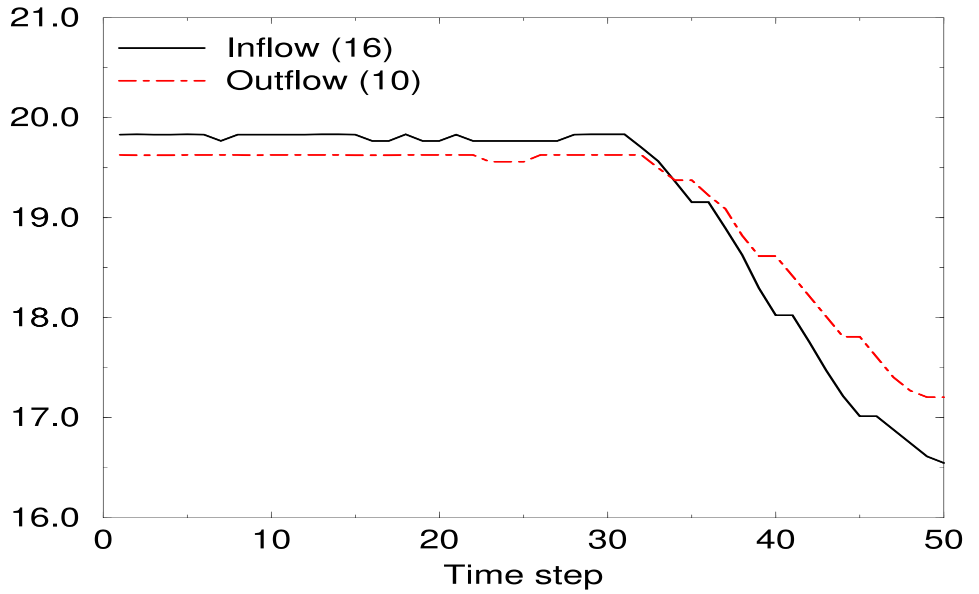


Figure 10.21: Flows at (16) and (10) during a CO_2 shutoff: the first 50 seconds

focuses on the first part of the sequence.

There are a few interesting phenomena in the data. The first is the qualitative behavior of the flows. Initially the flows are dropping, just as one would expect — turning off the CO_2 flow should decrease the flows in the system. However, 90 seconds into the transition the flows start to increase and even go beyond their original values. The reason is the different response of the separation membrane to different gases. After the CO_2 supply is cut off, the compositions of the gases in the gas loop change. Gradually, the CO and CO_2 in the system are vented out and the composition of H_2 in the flow increases. The resistance of the membrane to H_2 is smaller than the resistance to CO, and therefore the flows start to increase. Eventually, all the CO and CO_2 are vented and the system stabilizes again. Notice that our model can predict this subtle behavior of the system — the synthetic trajectory shown as the ground truth in Figure 10.15 has a similar qualitative behavior of first dropping, then rising and finally stabilizing (although, as we already discussed the noise levels in our model are larger and the simulated transition seems to happen slightly faster).

The second interesting phenomenon is the relation between the flows as measured by R_8 and by R_{12}. In general the flow at (16) is equal to the sum of the flows at (10) and (13). Thus, the flow measured by R_8 should be at least as large as the flow measured by R_{12}. In the actual sequence, the flow in R_8 is indeed larger than the flow at R_{12} during the first 30 seconds (see Figure 10.21). However, after the transition at $t = 30$, the readings of R_{12} become larger than the readings of R_8, and starting from time step $t = 280$, the difference becomes significant. The reason for this behavior is that apparently the sensors are not well calibrated and react differently to the changes in the gas compositions. Thus, although the actual flow in R_8 is slightly larger than the flow at R_{12}, the sensors indicate that it is significantly smaller.

Figure 10.22 shows how the readings of R_8 and R_{12} are reflected by the belief state of of our model. Here we plot the joint probability distribution over the status of the two sensors. Up to $t = 280$, the dominating hypothesis is that both sensors are working. The model explains the higher readings from R_{12} compared to R_8 by adjusting the bias variables of these sensors accordingly. However, the bias variables, as encoded in our model, are not likely to have large values. Since we did not model

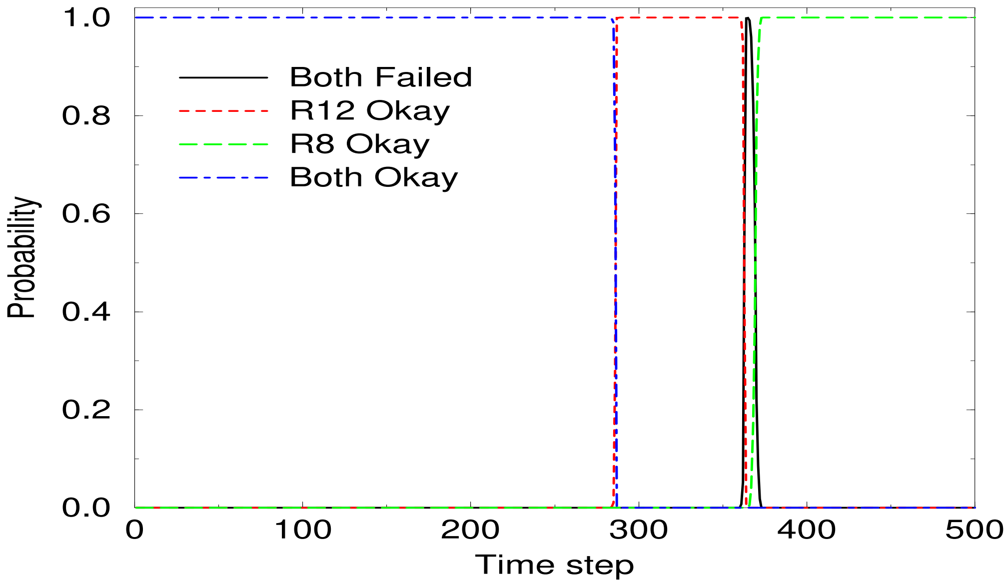


Figure 10.22: Predicted failures of R_8 and R_{12}

the dependency of the bias variables on the gas compositions, our model does not reflect the change in the biases after the gas compositions change. Thus, our model cannot explain the behavior of R_8 and R_{12} for $t > 280$.

The other possible explanation is that one of the two sensors is faulty, which is exactly the diagnosis our system makes. Initially, the likely hypothesis is that R_8 is faulty. When the flows level off at $t = 360$, our system seems to prefer the flow measured by R_8 and it therefore changes its diagnosis to R_{12} being faulty with high probability. For a few seconds it gives a high likelihood for both of the sensors to be faulty, but then it infers that R_8 is working after all.

This diagnosis seems quite reasonable, given that we did not explicitly model the influence of the gas compositions and the flow rates on the bias variables. In other words, given the sensor model we use, the behavior of R_8 and R_{12} should indeed be considered a fault, and the system makes a reasonable inference (which we did not expect beforehand) by assuming that one of the sensors is broken.

We now examine the performance of the model and our inference algorithm in

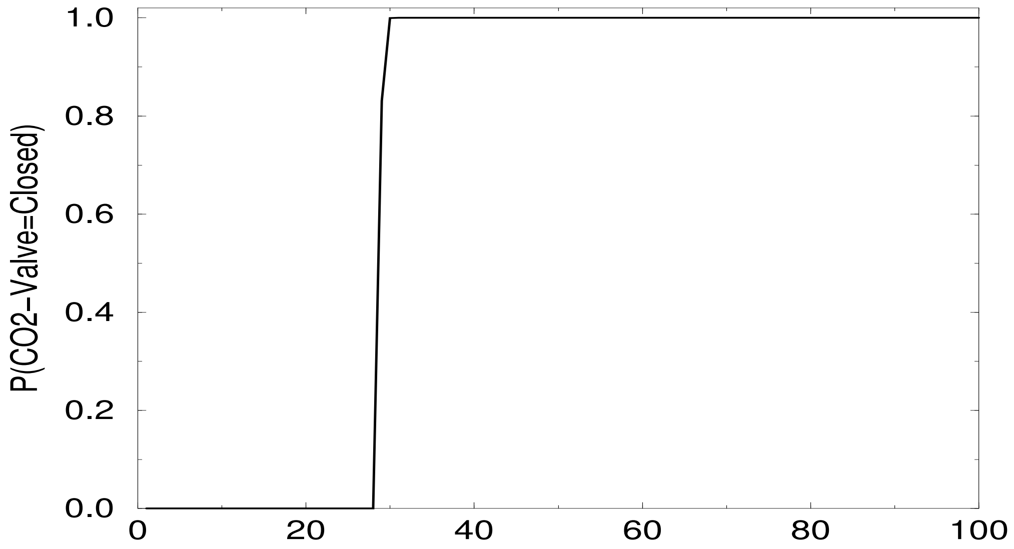


Figure 10.23: Predictions with all the flow sensors

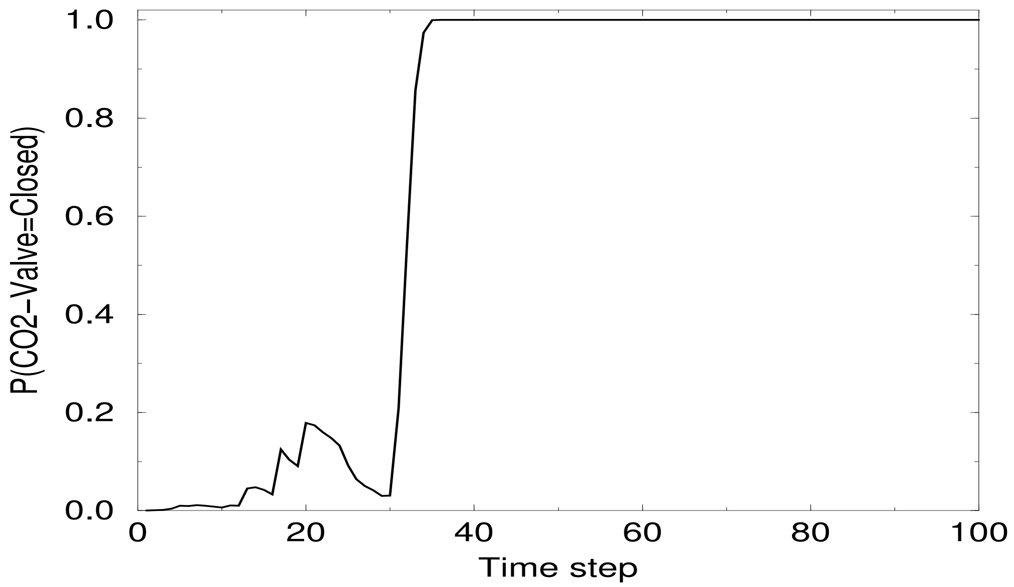


Figure 10.24: Prediction with all the flow sensors except for CO_2 readings

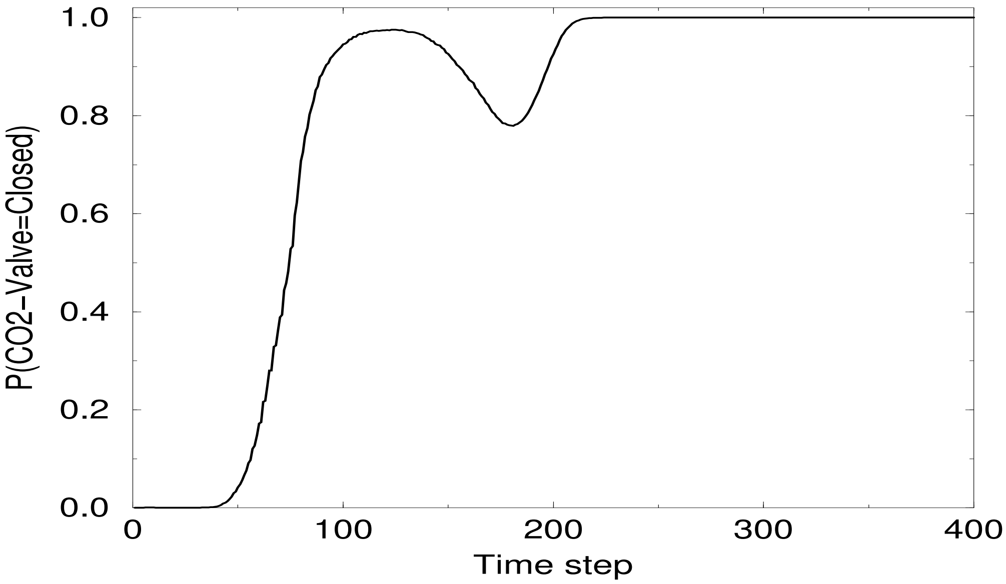


Figure 10.25: Prediction with no flow sensors except for H_2 readings

diagnosing the position of the valves. Figure 10.23 indicates that the correct diagnosis was reached, but as we already discussed in the case of the synthetic data, as long as we have the data from the CO_2 flow sensor R_3, located at (1), it is very easy to diagnose the CO_2 shutoff. Therefore, a more interesting test is to ignore the data from R_3 and test whether we can still diagnose the CO_2 shutoff. The results of this experiment are shown in Figure 10.24. Without the data from R_3, it takes a little more time to reach the right conclusion, but within a few time steps we reach the correct diagnosis and consider it to be extremely likely through the rest of the sequence.

In the previous section, we tried to diagnose an H_2 shutoff event with only the CO_2 flow sensor working. We tried a similar experiment with real data, but since our sequence included a CO_2 shutoff rather than an H_2 shutoff, we removed all the sensors except for the H_2 sensor. The results are shown in Figure 10.25, and are quite similar to the results on the synthetic data in Figure 10.14 (except for the synthetic data being more noisy as we discussed above). The fact that we get the same qualitative

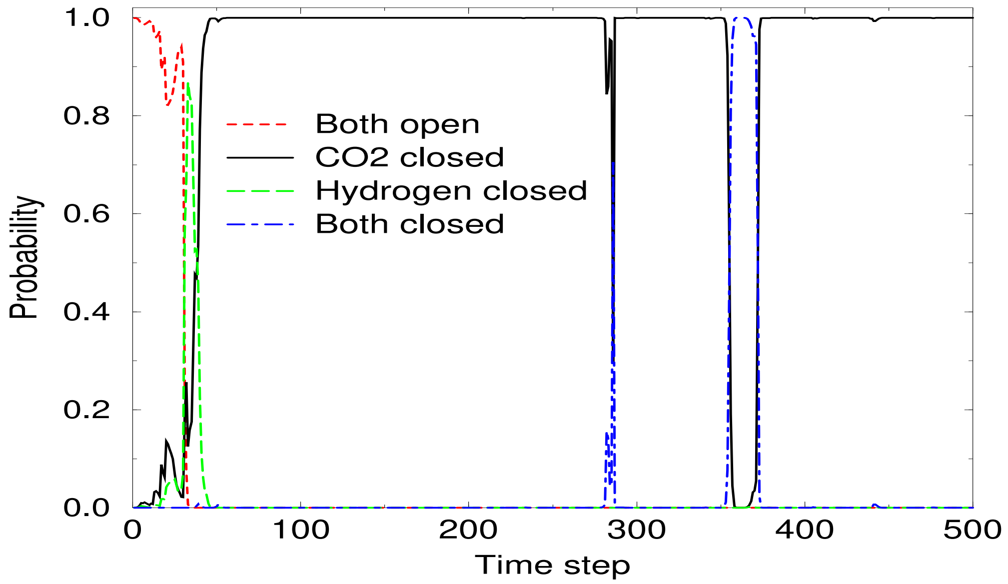


Figure 10.26: Prediction with no measurements of incoming flows

behavior for real and simulated data in this case is a positive indication for the quality of our model as well as the inference algorithm. Another positive indication is the fact that we are still able to come up with a useful diagnosis on a real data set even without many of the sensors.

As our final, and most challenging test, we ignored the data from both flow sensors R_1 and R_3, thus eliminating any measurement of incoming gas flows. Coming up with the correct diagnosis is much more challenging now, since, even if we infer that one of the valves is closed it is hard to infer which one, as the effects of a CO_2 shutoff and an H_2 shutoff are similar. The results of this experiment are plotted in Figure 10.26. Shortly after the transition, the probability of both valves being open goes down quickly and the two likely hypotheses are that the CO_2 valve was closed or that the H_2 valve was closed. For a short time the hypothesis of an H_2 shutdown is considered more likely but 15-20 seconds into the transition the probability of a CO_2 shutoff gets close to 1. It remains practically at 1 except for two short periods in the sequence. Interestingly the first period correspond to the time step where the model infers that

R_8 is faulty. The second corresponds to the short period where the model gives a high probability to the hypothesis of both R_8 and R_{12} being faulty.

These results are quite encouraging, although obviously there is some work to be done. We discuss some extensions to our framework in Chapter 11, but even with the current model and inference algorithm we were able to diagnose fairly complex faults that were not manifested directly in the sensor readings. In particular, the last two graphs, where important flow sensor readings were missing, demonstrate the power of hybrid DBNs in coming up with complex diagnoses based on the probabilistic model of the system.

10.6 Discussion

Previous work on fault diagnosis is based on two main approaches. Within the engineering community the fault diagnosis framework is usually based on SLDS models while the AI community has concentrated on constraint based methods, that work by discretizing the system. In this section we discuss these approaches and contrast them with our approach of using hybrid DBNs.

Within the fault detection and isolation engineering community, much of the research has focused on modeling the system as an SLDS, defined in Section 8.6 (e.g., [Fra90, Ise97, Ger88]). Non-linear systems are linearized, usually using the EKF. This approach is quite similar to our hybrid DBNs approach, with the main difference that there is no decomposition of the probability distribution: The model uses just one discrete variable that represents all the possible faults. As a result, we need to explicitly encode the behavior of all the continuous variables given every value of the discrete variable, forcing the model to have a relatively small number of possible faults.

The main alternative approach is to discretize the variables of the system, resulting in a discrete model which is often expressed as a set of constraints. A violated constraint indicates a fault, and the task of fault diagnosis is to find the likely explanations to the violated constraint. This is the classical approach used by the AI community (e.g., [dKW87, dKW89, CT94, McI98]).

To make the discussion more concrete, we focus on the *Livingstone* system [WN96, KN00] which is a fault diagnosis and control system developed at NASA. Livingstone is defined in two slightly different ways in the two references, but for our purposes Livingstone models a system as a tuple $(\boldsymbol{X}, \mathcal{M}_{\boldsymbol{X}}, \mathcal{M}_{\mathcal{T}})$ where:

- $\boldsymbol{X}$ is a set of discrete state variables, representing the state of the system. Just like DBNs, we have a set of variables for every time step, and in particular $\boldsymbol{X}^{(t)}$ represents the state of the system at time t. Any continuous quantity of interest is discretized into a small number of possible values. Examples of typical discretizations are two values such as *Zero, Nonzero* and three values such as *Low, Medium, High.* Note that every sensor observation is discretized as a pre-processing step.
- $\mathcal{M}_{\boldsymbol{X}}$ is a set of state constraints formalized as propositional logical formulas defined over $\boldsymbol{X}$. These constraints define a subset of the possible states of $\boldsymbol{X}$ as the feasible states of the system. For example, we can use the constraint

  $$(V_1 = \mathit{Closed}) \Longrightarrow (\mathit{flow}_{V_1} = \mathit{zero})$$

  where V_1 represents the state of some valve and flow_{V_1} represents the flow through V_1.
- $\mathcal{M}_{\mathcal{T}}$ is another set of constraints, constraining the possible transitions from $\boldsymbol{X}^{(t)}$ to $\boldsymbol{X}^{(t+1)}$. $\mathcal{M}_{\mathcal{T}}$ is defined as a set of propositional formulas over $\boldsymbol{X}^{(t)} \cup \boldsymbol{X}^{(t+1)}$. For example, we can state that if the sensor S_1 is broken then it remains broken:

  $$(S_1^{(t)} = \mathit{Broken}) \Longrightarrow (S_1^{(t+1)} = \mathit{Broken})$$

Livingstone uses a simple heuristic to model the probabilities of the possible trajectories in the system, based on the number of faults in a transition, where all the faults are assumed to be independent. A transition with k faults is not considered believable unless all the trajectories with less than k faults were ruled out due to the evidence and the constraints $\mathcal{M}_{\boldsymbol{X}}$ and $\mathcal{M}_{\mathcal{T}}$. Thus, Livingstone considers the nominal no-faults trajectory as the actual trajectory of the system if it is consistent with the constraints. If it is not, then the possible trajectories of the system are defined as all the single fault trajectories which are consistent with the constraints. If there are no such trajectories, the double fault trajectories are considered and so on.

The actual algorithm used to find the trajectories [KN00] is quite similar to our enumeration algorithm from Chapter 5. The algorithm maintains a belief state over time t variables and enumerates the transitions to time $t+1$ in order of their likelihood, ignoring all the transitions that are not consistent with the constraints. However, other than the fact that Livingstone uses only discrete variables, there is another important difference between Livingstone and our approach. In Livingstone the belief state is defined as a set of possible states, as opposed to our approach which maintains a probability distribution over the states. Consequently, while our approach uses the evidence to make some states more likely than others, Livingstone uses the evidence as hard constraints, making states either feasible or not.

Livingstone was applied by NASA to a few systems, including the propulsion system used in the Cassini spacecraft and the RWGS. While Livingstone was successful in the former application it was not in the latter [GK01]. The reason is the different nature of the two systems. The model of the propulsion system has two basic modes of operations, determined by the configuration of the valves: either there is flow of fuel through the system and thrust is produced, or there is no flow and no thrust. The typical faults in the system cause the flow to start or end unexpectedly, rather than causing the flow to increase or decrease the flow. Thus, there are many questions that can be answered by a very simple discretization of the flows to *Zero* and *Nonzero*.

The situation in the RWGS is more complex. As we have seen, faults can manifest themselves in changes in the flows and pressures rather than the flow going down to zero. If we use the *Zero* and *Nonzero* discretization in the RWGS, then the entire graph in Figure 10.20 would be reduced into the flow constantly being *Nonzero*, and the fault would be impossible to diagnose. It is possible to define a finer discretization but we are faced with similar problems to those discussed in Section 3.4.1: It is hard to find a good discretization scheme and inference becomes significantly more expensive. In addition, defining the model becomes harder. For example, assume that we partition the possible flows to ranges such as $\{[0,1),[1,2),\ldots,[59,60)\}$. We now have to decide whether to define a flow in the range $[19, 20)$ and a sensor reading of $[20, 21)$ as a fault, and if it is, how to express the fact that it is more likely than a fault resulting in a reading of $[40, 41)$. Diagnosing faults such as a slow drop in the temperature of the reactor as a result of the heater being turned off is extremely hard under this framework.

The deep reason for these problems is that the RWGS is truly a hybrid system, where the actual continuous quantities are important beyond the question of being zero or non-zero. Thus, although the classical constraint-based approach may be useful in some cases, it does not seem to work well in systems such as the RWGS. We feel that the results in this chapter indicate that modeling these systems as hybrid dynamic Bayesian networks is a natural alternative that has the potential of overcoming some of the limitations of the classical approach.

Chapter 11

Conclusion and Future Work

11.1 Summary

In this thesis, we have tried to achieve two main goals: develop a better understanding of hybrid Bayesian networks and show that hybrid models can be useful for practical applications, in particular in order to perform fault diagnosis on a complex real world physical system.

We first presented an overview of inference in both discrete and hybrid Bayesian networks. We showed that some of the insights for discrete networks can be used for hybrid networks but that there are also some important subtle differences. In particular we saw that it is possible to adapt the clique tree algorithm, developed for discrete networks, for CLGs. However, unlike the discrete case, not every elimination order can be used for CLGs. For some elimination orders, the operations involved in the message passing are not be well defined, and even if the operations are well defined they sometimes lead to illegal distributions, such as distributions with a negative variance.

To fix this problem, it is necessary to add a crucial restriction on the elimination order, called strong triangulation. We saw that with strong triangulation, we get a strongly rooted clique tree, where message passing is not only well defined, but is also guaranteed to find the correct (weak) marginal in every clique. Unfortunately, our complexity analysis showed that strong triangulation might dramatically increase the size of the clique tree. For many simple networks including polytrees, where without strong triangulation we get compact clique trees, strong triangulation leads to impractical clique trees.

The last observation led to an important and natural questions, namely, can strong triangulation be avoided. In other words, perhaps it is possible to find an algorithm which is not based on clique trees and does not suffer from the problems induced by strong triangulation. Alternatively, if we compromise on using approximate inference instead of exact inference, perhaps we can find some approximate inference algorithm which is efficient for networks such as polytrees and still provides a good approximation to the posterior distribution.

These questions were answered by our theoretical analysis in Chapter 4, and unfortunately, in general the answer is no. We showed that, even in polytrees, approximate inference in CLGs is NP-hard (unlike the discrete case where even exact inference in polytrees can be done in linear time). We proved that the NP-hardness holds even for a restricted class of polytrees where the marginal prior distribution for every continuous variables is a mixture of at most two Gaussians. A corollary is that finding the most likely discrete instantiation given some observations over the continuous variables is NP-hard, even for very simple network structures.

The last result appears quite discouraging in the context of fault diagnosis, where part of the task is to find the likely states of the system given the sensor readings. Our corollary tells us that finding even the single most likely state of the system (and certainly the K most likely states) is already NP-hard even in polytrees.

However, in Chapter 5 we showed that, in practice, the situation is not as bad. Real life domains do not look like NP-hardness reductions and we can often take advantage of their properties in order to come up with useful and efficient algorithms. In particular, in many domains the distribution over the discrete variables is very skewed. For example, in the fault diagnosis domain, the probability for faults is low and therefore the combination of many simultaneous independent faults is extremely unlikely. We presented an any-time algorithm that takes advantage of this phenomenon by deterministically enumerating the possible states of the system in order of their prior likelihood. Since the probability distribution is skewed, the hope is that considering a fairly small number of the most likely hypotheses provides a good approximation to the actual distribution. Our algorithm works well if at least some of the hypotheses that are likely *a priori*, are the most likely hypotheses *a posteriori*, and considering them produces a good approximation of the posterior distribution.

Our algorithm is an any-time algorithm and its space complexity is often exponentially smaller than Lauritzen's algorithm, and therefore we can use it and get some answer (albeit an approximate one) even in cases where Lauritzen's algorithm is completely impractical. Furthermore, we empirically showed that this algorithm outperforms the classical sampling algorithms on networks with a skewed distribution.

We next turned our attention to non-linear models. Once we introduce non-linear relations into our model, the resulting distribution is no longer a Gaussian (or a mixture of Gaussians), and can be arbitrarily complex. However, in many cases even if the distribution is non-Gaussian, it can be well approximated as a Gaussian. Thus, our approach was to approximate every non-Gaussian distribution as a Gaussian.

In Chapter 6, we described in detail two numerical integration techniques, namely Gaussian Quadrature and Exact Monomials. These techniques proved to be very useful for our purposes as they allowed us to compute efficiently the first two moments of the non-Gaussian distribution, often with a very high precision. We showed that our approach often outperforms the extended Kalman filter approach. We also showed that our approach presents a simple and coherent view of the successful Unscented Filter which turns out to be a special case of the Exact Monomials method. Once viewed as a numerical integration technique, it becomes clear how one can extend the Unscented Filter to achieve better precision. Our numerical integration approach can be naturally combined with the enumeration algorithm from Chapter 5.

In Chapter 7, we concentrated on augmented CLGs, i.e., CLGs that include discrete nodes with continuous parents. We presented a systematic way to deal with these networks by extending Lauritzen's algorithm using our numerical integration techniques. Our algorithm computes the exact distributions over the discrete nodes, and the exact first and second moments of the continuous ones, up to inaccuracies resulting from the numerical integration procedure. To achieve these properties, we had to address some subtle technical issues, and modify Lauritzen's algorithm is some non-trivial ways. We showed that, in the special case of softmax CPDs, the integration can often be done efficiently, and that using the first two moments leads to a particularly accurate approximation.

We also discussed how to extend our enumeration algorithm to augmented CLGs. We proved that enumeration from the prior distribution, which can be done efficiently for CLGs with a simple structure such as polytrees, is NP-hard for augmented CLGs with a similar structure. Therefore, our enumeration algorithm cannot be directly used in augmented CLGs, and some other heuristics are necessary.

Up to this point in the thesis we concentrated on static models. However, as we pointed out, we are interested is tracking stochastic processes over time. We therefore turned our attention to temporal models, namely to DBNs. In Chapter 8, we described three standard inference algorithms for hybrid DBNs: GPB1, GPB2, and IMM. Unfortunately, none of these algorithms is suitable for our needs, because of the number of Gaussians kept in the belief state. GPB1 keeps only one Gaussian in the belief state, which is not enough for the purposes of fault diagnosis, while GPB2 and IMM keep 2^n Gaussians (assuming the belief state has n binary variables). Clearly, these algorithms are not practical when n is large. In particular, models in the fault diagnosis domain can easily have tens of discrete variables in the belief state. For example, in the RWGS system discussed in Chapter 10 there are 33 discrete variables in the belief state.

Thus, in Chapter 9, we discussed how we can scale up inference in hybrid DBNs. We first observed that some of the Gaussians that are collapsed together in GPB2 may be very different from each other, and better kept as distinct hypotheses. On the other hand, some of the Gaussians that GPB2 keeps as separate hypotheses may be very similar to each other and can be collapsed without a significant loss of accuracy. Using this insight, we developed a new collapsing algorithm. Our algorithm takes a parameter K, representing the maximal number of Gaussians in the belief state, and tries to identify the K most likely distinct hypotheses and keep those in the belief state. A key part of our algorithm is that it collapses Gaussians together only if they are similar to each other in terms of KL-divergence.

We then discussed the implications of the collapsing algorithm on the representation of the belief state. We described the belief state using a graphical model that includes a special "hypothesis variable". Each value of the hypothesis variable corresponds to exactly one of the Gaussians in the belief state and induces a probability distribution on the discrete variables in the belief state. Thus, the hypothesis variable is a convenient way to correlate the mixture of Gaussians resulting from our collapsing algorithm with a distribution over the discrete variables.

In addition to approximating the belief state by collapsing together some of the hypotheses, it may be beneficial to decompose the belief state and keep it in a factored way, just like in discrete DBNs. We discussed various ways of doing so, including decomposing the mixture of Gaussians by using more than one hypothesis variable.

By this point in the thesis, we described all the techniques necessary to perform fault diagnosis in the RWGS model. In Section 9.5 we presented our tracking algorithm for hybrid DBNs that combines the following set of techniques:

- The enumeration algorithm from Chapter 5.
- The numerical integration techniques from Chapter 6.
- The collapsing algorithm and the representation of the belief state from Chapter 9.

Finally, we applied these techniques to the RWGS system in Chapter 10. The RWGS system is a complex physical system designed to extract oxygen from the Martian atmosphere, or any other carbon dioxide rich environment. We described the system and discussed some of the interesting issues that must be addressed in order to model the RWGS system as a hybrid DBN, including the problem of parameter estimation.

We presented experimental results both on synthetic data and real data collected during actual runs of the system. The experiments on the synthetic data compared the enumeration algorithm and the sampling approach, both with and without Rao-Blackwellization. We believe that these experiments demonstrate that the enumeration approach has some real advantages over the sampling approach. However, even the most successful experiments with synthetic data are not enough to demonstrate that the approach of modeling a system as a hybrid DBN can work in practice. In real world domains, the model is just an approximation of the system. Thus, real data never quite fits the model, and working with it presents an extra challenge over synthetic data. We feel that our experiments with real data sets demonstrate the feasibility of our approach under real-world conditions.

11.2 Limitations and Future Directions

It is our hope that this thesis demonstrates the usefulness of hybrid (dynamic) Bayesian networks and provides useful algorithms for inference in these models. Obviously, we have not answered all the questions that come up in these models, and there is still room for much work to be done. In this section, we review some exciting research directions that build on top of the work in this thesis.

11.2.1 Inference Issues

The heart of the algorithm used for tracking the RWGS system is the enumeration algorithm from Chapter 5 combined with the techniques of Chapter 6. Although this algorithm was proven to be capable of reliably dealing with a complex real-world system, it still can be improved in a few ways.

Combination with CD CPDs

Currently, our enumeration algorithm only handles non-linear relations between the continuous variables. In Chapter 7, we showed why it is not easy to combine the algorithm with CD CPD (CPDs of discrete nodes with continuous parents). Nonetheless, the need to do so still exists, since, as we discussed in Chapter 7, in some cases using CD CPDs is the most natural way to model a system. Thus, finding a way to incorporate CD CPDs with our enumeration framework is still an open problem.

Combination with Outside Experts

Our enumeration algorithm only uses information available directly from the probability distribution. However, in many cases we can get more information that may be useful and can improve the performance of our algorithm. For example, we may have an outside "expert" that uses its own algorithms to come up with possible hypotheses about the state of the system. It seems natural to combine such an expert with our algorithm, by combining the hypotheses generated by the expert with the most likely hypotheses enumerated by our algorithm. Defining such an architecture and studying its theoretical and empirical properties may prove to be a significant advance for using hybrid DBNs for fault diagnosis.

Adaptive Enumeration

The order in which the enumeration algorithm enumerates hypotheses is determined by the prior distribution over the discrete variables and by the discrete observations, if any exist. The enumeration order does not take into account any observations for the continuous variables. As we discussed in Chapter 5, we can view this as a similar approach to RB-LW which generates samples from a distribution that does not take into account the continuous observations.

The *adaptive importance sampling* approach [OK00, CD00] attempts to improve the sampling distribution over time. Roughly speaking, the idea is to use the samples that were already generated in order to get a better approximation of the posterior distribution. We then generate samples from our improved approximation of the posterior, use these samples to improve our approximation even further, and so on.

It should be possible to use the same idea for our enumeration algorithm. We start by enumerating hypotheses from the distribution P_0 which does not take into account any continuous evidence. After enumerating the K_0 most likely hypotheses from P_0, we use them to estimate a new distribution P_1, which should be a better approximation of the posterior distribution. We then stop enumerating hypotheses from P_0 and start enumerating hypotheses from P_1. Again, after we enumerate the K_1 most likely hypotheses from P_1 we use them to compute P_2, and continue in this fashion.

This approach may prove to be useful in dealing with simultaneous faults. Assume that four independent faults happen simultaneously. Since the probability of faults is low, it might take too long before our enumeration algorithm reaches the hypothesis that corresponds to all four faults. However, perhaps using the hypotheses that were already generated we discover that the probabilities for the faults are larger than our original estimates. We now learn a new distribution where each one of the faults is more likely. In this new distribution, the hypothesis corresponding to the four simultaneous faults is also more likely, and appears earlier in the enumeration order, which may enable the enumeration algorithm get to it at a reasonable time.

Exploiting Approximate Conditional Independencies

Perhaps the most important improvement to our enumeration algorithm is to combine it with a collapsing process. As an example, consider the RWGS system once again. The gas compositions in the gas loop depend on the temperature of the reactor, and thus depend on the discrete variable representing whether the heater is turned on or off. Thus, we must include the discrete variable representing the heater status among the variables for which we enumerate the hypotheses. However, if we know the current temperature of the reactor, this dependency disappears. In practice, we never know the exact temperature since all our sensors are noisy, but if we know the temperature with a high accuracy, perhaps we can still ignore the dependency of the gas compositions on the heater status. Thus, if the temperature sensor is working properly, there may be no need to enumerate the different hypotheses for the heater status — since they are similar for the purposes of our query we can collapse them into one, pick one of them, or even completely ignore this part of the network. On the other hand, if the temperature sensor is not working, perhaps it is important to consider the heater status, in which case we do want to enumerate the different hypotheses.

This observation suggests a sophisticated algorithm that takes advantage of approximate conditional independencies between the query variables and some other parts of the network, given some partial instantiation of the discrete variables in the network. To implement this algorithm, one must address many technical issues: in what order should the partial discrete instantiations be enumerated, how to identify the approximate conditional independence relations, how to actually deal with the variables that are approximately independent from our query variables, how to combine these techniques with non-linear dependencies, and so on.

Thus, coming up with an algorithm that takes advantage of our observation is a challenging problem, but we feel that the potential improvement in the performance of the algorithm can be dramatic and well justify the time spent on this problem. The reason is that, if we can find many partial instantiations for which large parts of the network become approximately independent from our query, we can get an exponential reduction in the number of hypotheses. In fact, using this approach we may be able to effectively enumerate all the distinct hypotheses for our query variables.

11.2.2 Modeling Long Term Changes

In the RWGS experiments we concentrated on diagnosing abrupt changes such as the CO_2 valve getting closed. In physical systems there is another important type of change, called *drifts*. Drifts occur slowly over a long period of time. An example of a drift would be the slow accumulation of dust in the chemical reactor, causing a very slow drop in the efficiency of the reactor. Drifts cause no noticeable change over a short period of time and become significant only if we consider a much less accurate time granularity. Over any short period of time, drifts cannot be detected, since the changes that they cause are negligible compared to other sources of variability in the system.

As a motivating example, consider the process of aging for human beings. As a thought experiment, assume that we are given a sequence of snapshots of some person's face taken at time intervals of one hour from each other. The snapshots will obviously be different from each other due to different light conditions, different facial expressions, and so on. If we look at snapshots taken hours or days apart from each other there would be no indication of the aging process — most likely we would not be able to tell the chronological order in which the snapshots were taken. However, if we look at snapshots taken decades apart the results of the aging process will be very clear.

Drifts present a challenge both from a modeling perspective and from a diagnosis perspective. From a modeling perspective, it is not clear how to best model drifts, given that they induce no perceptible change from one time step to the next. From a diagnosis perspective, it is not clear whether we can use our standard techniques of tracking the belief state in order to identify drifts or perhaps we need some different mechanism, perhaps equivalent to comparing the current snapshot to snapshots that were taken a long time ago.

11.2.3 Active Learning

The parameter estimation process for the RWGS system, as described in Section 10.3, was a very complicated and time-consuming process. It is quite certain that even after this investment, the model parameters can still be greatly improved. For example, we currently do not know how good the RWGS model is for H_2 shutoff, since we do not have such a sequence available, but it is very possible that the predictions of the model in this case are quite poor.

Obviously, there is a need for a more systematic way for parameter estimation and model learning in general. One possibility is to use the EM algorithm [DLR77] which is designed to estimate the parameters of the model from incomplete data, i.e., data in which not all the variables are observed. However, we we feel that in order to learn a good model for physical systems, it is important to direct the data collection process to include sequences that would be useful for modeling purposes (e.g., a sequence that includes a H_2 shutoff for the RWGS). In other words, we would like to perform *active learning*

In an active learning setup, it is possible to direct the data collecting process to collect data for specific system configurations. In active learning, the framework is that the learning algorithm can ask the data collecting process to first set some of the variables to some values, and then to generate a sample given these values.

There has been some work on active learning in the context of discrete Bayesian networks [TK01a, TK01b]. However, in order to successfully use active learning for our needs, some interesting problems must be addressed:

- How to perform active learning for hybrid networks rather than discrete ones.
- How to perform active learning for DBNs rather than Bayesian networks. Note that, instead of requesting a sample given a setting of some of the variable, in the dynamic setup we would most likely ask for a sequence describing the system behavior given some initial conditions that we impose and perhaps some more interventions along the way.
- In real life systems such as the RWGS, we must take into account some safety constraints. For example, we may not be able to ask to test the behavior of the system when the pressure grows to 100 atmospheres since the system would explode in this case. Thus, in order to use active learning for systems such as the RWGS, we must model the safety constraints and combine them with the active learning algorithm.

11.2.4 Control

Finally, in this thesis we focused on the task of fault diagnosis. In many practical cases we are not interested in fault diagnosis by itself but rather as a part of a control system: We do not want to just know about the system faults — we want to take actions to fix them or at the very least minimize the damage that they cause. Thus, there is an obvious need for a control component that would work hand in hand with our fault diagnosis tools.

Control is a difficult problem, even when the state of the system is fully known. The classical control theory [Oga01] usually deals with continuous systems, while the AI community often focuses on discrete systems, usually modeled as *Markov decision processes* [Put94]. Some work has been done on the problem of creating a control component for complex hybrid systems (e.g., [TPS98, PS96]), but the problem is still very much an open issue, especially if the state of the system is not fully known.

11.3 Conclusion

It is our hope that this thesis demonstrates that hybrid (dynamic) Bayesian networks are a powerful tool for reasoning about complex and realistic domains. They combine the explicit representation of uncertainty that has proved useful for Bayesian networks with enough expressive power to model discrete as well as continuous phenomena in the world. In this thesis, we concentrated on the use of hybrid models for fault diagnosis, but it is not hard to come up with other domains that call for hybrid models: visual tracking, speech recognition, robotics and many others.

In this thesis, we have answered some important and fundamental questions about hybrid models, which naturally led to some other questions, some of them just as important, or even more so. But perhaps even more importantly, in this thesis we go beyond theoretical analysis or testing our algorithms on synthetic data generated from toy examples — we take the bull by its horns and prove the applicability of our techniques on a real-world system with actual data. We believe this to be a strong indication that hybrid Bayesian networks are more than an intellectual exercise, and actually hold practical importance. We hope that this will serve as a motivation for other researchers to pursue some of the research questions related to hybrid Bayesian networks in combination of testing the applicability of hybrid models in other domains.

Appendix A

Some Proofs

A.1 Proof of Lemma 3.14

Proof: Let $\boldsymbol{X} = \{X_1, \ldots, X_n\}$ and $\boldsymbol{Y} = \{Y_1, \ldots, Y_m\}$. We prove the theorem by induction on m. Assume first $m = 1$, in which case we can write $P(Y_1 \mid \boldsymbol{X}) = \mathcal{N}(Y_1; a + \sum_i b_i X_i, c)$ or alternatively $P(Y_1 \mid \boldsymbol{X}) = Z + \sum_i b_i X_i$ where $P(Z) = \mathcal{N}(Z; a, c)$. We now have:

$$
\begin{aligned}
E[Y_1] &= E\left[Z + \sum_i b_i X_i\right] \\
&= E[Z] + E\left[\sum_i b_i X_i\right] \\
&= a + \sum_i b_i E\left[X_i\right] \\
E[Y_1^2] &= E\left[\left(Z + \sum_i b_i X_i\right)^2\right] \\
&= E\left[Z^2\right] + 2E\left[Z \sum_i b_i X_i\right] + E\left[\left(\sum_i b_i X_i\right)^2\right] \\
&= a^2 + c + \sum_{i,j} b_i b_j E[X_i X_j] \\
E[Y_1 X_j] &= E\left[\left(Z + \sum_i b_i X_i\right) X_j\right]
\end{aligned}
$$

$$
\begin{aligned}
&= E\left[ZX_i\right] + E\left[\sum_i b_i X_i X_j\right] \\
&= \sum_j b_i E\left[X_i X_j\right]
\end{aligned}
$$

We get that all the expectations depend only on the first two moments of $\boldsymbol{X}$ as required. The induction step is similar. Since $P(\boldsymbol{Y} \mid \boldsymbol{X})$ is linear we can always write Y_i as a linear combination of $\{\boldsymbol{X}, Y_1, \ldots, Y_{i-1}\}$ plus some Gaussian noise. By induction we already have the first two moments of $P(\boldsymbol{X}, Y_1, \ldots, Y_{i-1})$ and using the same argument as in the case of $m = 1$ we can show that the first two moments that involve Y_i are determined directly by the first two moments of $P(\boldsymbol{X}, Y_1, \ldots, Y_{i-1})$ ∎

A.2 Proof of claim used in Theorem 4.2

Proof: Recall the network shown in Figure 4.3 where the parameters are defined as:

$$
\begin{array}{rrcl}
 & P(X_0) & = & \mathcal{N}(0, \sigma^2) \\
\text{For } 1 \leq i \leq n\text{:} & P(X_i) & = & \mathcal{N}(M, \sigma_2^2) \\
\text{For } 1 \leq i \leq n\text{:} & P(Z_i) & = & \begin{cases} \mathcal{N}(X_i - X_{i-1}, \sigma_1^2) & A_i = 0 \\ \mathcal{N}(X_i - X_{i-1} - s_i, \sigma_1^2) & A_i = 1 \end{cases} \\
 & P(Y) & = & \begin{cases} \mathcal{N}(L - \sqrt{2n}, 1) & B = 0 \\ \mathcal{N}(X_n, \sigma^2) & B = 1 \end{cases}
\end{array}
$$

where

$$
M = \sum_j s_j \ ; \ \sigma_1^2 = \frac{\epsilon \sigma^2}{n \cdot M} \ ; \ \sigma_2^2 = \sigma^2 \left(1 + \frac{\sigma^2}{\sigma_1^2}\right)
$$

Our goal is to prove that $\forall 1 \leq k \leq n$

$$
P(X_k \mid \boldsymbol{a}^k, \boldsymbol{Z}^k = \boldsymbol{0}) = \mathcal{N}\left(\mu_k, \sigma^2\right)
$$

where $|\mu_k - S(\boldsymbol{a}^k)| \leq \frac{k\epsilon}{n}$.

We prove the claim by induction on k. For $k = 0$ we must show that $P(X_0) = \mathcal{N}(0, \sigma^2)$, which is given. For a general $k > 0$ we can use the conditional independencies induced by the network and write:

$$\begin{aligned} P(X_k, Z_k \mid \boldsymbol{a}^k, \boldsymbol{Z}^{k-1} = \mathbf{0}) &= \int P(X_{k-1}, X_k, Z_k \mid \boldsymbol{a}^k, \boldsymbol{Z}^{k-1} = \mathbf{0}) dX_{k-1} \\ &= \int P(X_{k-1} \mid \boldsymbol{a}^k, \boldsymbol{Z}^{k-1} = \mathbf{0}) P(X_k \mid X_{k-1}, \boldsymbol{a}^k, \boldsymbol{Z}^{k-1} = \mathbf{0}) \\ &\quad P(Z_k \mid X_k, X_{k-1}, \boldsymbol{a}^k, \boldsymbol{Z}^{k-1} = \mathbf{0}) dX_{k-1} \\ &= \int P(X_{k-1} \mid \boldsymbol{a}^{k-1}, \boldsymbol{Z}^{k-1} = \mathbf{0}) P(X_k) \\ &\quad P(Z_k \mid X_k, X_{k-1}, a_k) dX_{k-1} \end{aligned}$$

Using the induction hypothesis we have $P(X_{k-1} \mid \boldsymbol{a}^{k-1}, \boldsymbol{Z}^{k-1} = \mathbf{0}) = \mathcal{N}(\mu_{k-1}, \sigma^2)$ and therefore it is easy to explicitly write $P(X_k, Z_k, X_{k-1} \mid \boldsymbol{a}^k, \boldsymbol{Z}^{k-1} = \mathbf{0})$:

$$P(X_{k-1}, X_k, Z_k \mid \boldsymbol{a}^k, \boldsymbol{Z}^{k-1} = \mathbf{0}) = \mathcal{N}\left(\left(\begin{array}{c} \mu_{k-1} \\ M \\ M - \mu_{k-1} - \alpha \end{array} \right), \left[\begin{array}{ccc} \sigma^2 & 0 & -\sigma^2 \\ 0 & \sigma_2^2 & \sigma_2^2 \\ -\sigma^2 & \sigma_2^2 & \sigma^2 + \sigma_1^2 + \sigma_2^2 \end{array} \right] \right)$$

where $\alpha = 0$ if $A_k = 0$ and $\alpha = s_k$ if $A_k = 1$. Marginalizing out X_{k-1} is trivial and we get:

$$P(X_k, Z_k \mid \boldsymbol{a}^k, \boldsymbol{Z}^{k-1} = \mathbf{0}) = \mathcal{N}\left(\left(\begin{array}{c} M \\ M - \mu_{k-1} - \alpha \end{array} \right), \left[\begin{array}{cc} \sigma_2^2 & \sigma_2^2 \\ \sigma_2^2 & \sigma^2 + \sigma_1^2 + \sigma_2^2 \end{array} \right] \right) \quad \text{(A.1)}$$

To find $P(X_k \mid \boldsymbol{a}^k, \boldsymbol{Z}^k = \mathbf{0})$ we need to condition the distribution in Equation A.1 on $Z_k = 0$ and find the new mean and variance of X_k. Using Corollary 3.5, the posterior mean of X_k is

$$E[X_k \mid \boldsymbol{a}^k, \boldsymbol{Z}^k = \mathbf{0}] = M + \frac{\sigma_2^2}{\sigma^2 + \sigma_1^2 + \sigma_2^2} (0 - (M - \mu_{k-1} - \alpha)) \quad \text{(A.2)}$$

where

$$
\begin{aligned}
\frac{\sigma_2^2}{\sigma^2+\sigma_1^2+\sigma_2^2} &= \frac{\sigma^2\left(1+\frac{\sigma^2}{\sigma_1^2}\right)}{\sigma^2+\sigma_1^2+\sigma^2\left(1+\frac{\sigma^2}{\sigma_1^2}\right)} \\
&= \frac{\sigma^2\left(\sigma_1^2+\sigma^2\right)}{\sigma_1^2\left(\sigma_1^2+\sigma^2\right)+\sigma^2\left(\sigma_1^2+\sigma^2\right)} \\
&= \frac{\sigma^2}{\sigma_1^2+\sigma^2} \\
&= \frac{\sigma^2}{\frac{\epsilon}{nM}\sigma^2+\sigma^2} \\
&= \frac{1}{1+\frac{\epsilon}{nM}} \qquad \text{(A.3)}
\end{aligned}
$$

Combining Equation A.2 and Equation A.3 we get:

$$
\begin{aligned}
\left|E[X_k \mid \boldsymbol{a}^k, \boldsymbol{Z}^k = \mathbf{0}] - (\mu_{k-1}+\alpha)\right| &= \left|M - \frac{1}{1+\frac{\epsilon}{nM}}\left(M-\mu_{k-1}-\alpha\right) - (\mu_{k-1}+\alpha)\right| \\
&= \left|\left(1-\frac{1}{1+\frac{\epsilon}{nM}}\right)\left(M-\mu_{k-1}-\alpha\right)\right| \\
&\leq \frac{\frac{\epsilon}{nM}}{1+\frac{\epsilon}{nM}} \cdot M \\
&\leq \frac{\epsilon}{nM} \cdot M \\
&= \frac{\epsilon}{n}
\end{aligned}
$$

By the inductive hypothesis we know that $\left|S(\boldsymbol{a}^{k-1}) - \mu_{k-1}\right| \leq \frac{(k-1)\epsilon}{n}$. Furthermore, by our choice of α, $S(\boldsymbol{a}^k) = S(\boldsymbol{a}^{k-1}) + \alpha$. Therefore,

$$
\begin{aligned}
\left|S(\boldsymbol{a}^k) - E[X_k \mid \boldsymbol{a}^k, \boldsymbol{Z}^k = \mathbf{0}]\right| &= \left|S(\boldsymbol{a}^{k-1}) + \alpha - \mu_{k-1} + \mu_{k-1} - E[X_k \mid \boldsymbol{a}^k, \boldsymbol{Z}^k = \mathbf{0}]\right| \\
&\leq \left|S(\boldsymbol{a}^{k-1}) - \mu_{k-1}\right| + \left|\mu_{k-1} + \alpha - E[X_k \mid \boldsymbol{a}^k, \boldsymbol{Z}^k = \mathbf{0}]\right| \\
&\leq \frac{(k-1)\epsilon}{n} + \frac{\epsilon}{n} \\
&= \frac{k\epsilon}{n}
\end{aligned}
$$

We now turn our attention to computing the posterior variance of X_k, after conditioning on $\boldsymbol{a}^k$ and $\boldsymbol{Z}^k = \mathbf{0}$. Again, we use Corollary 3.5 and get:

$$\begin{aligned}
\mathrm{Var}(X_k \mid \boldsymbol{a}^k, \boldsymbol{Z}^k = \mathbf{0}) &= \sigma_2^2 - \frac{\sigma_2^4}{\sigma^2 + \sigma_1^2 + \sigma_2^2} \\
&= \sigma_2^2 \left(1 - \frac{\sigma_2^2}{\sigma^2 + \sigma_1^2 + \sigma_2^2}\right) \\
&= \sigma^2 \left(1 + \frac{\sigma^2}{\sigma_1^2}\right) \frac{\sigma^2 + \sigma_1^2}{\sigma^2 + \sigma_1^2 + \sigma^2 \left(1 + \frac{\sigma^2}{\sigma_1^2}\right)} \\
&= \sigma^2 \frac{\sigma^2 + \sigma_1^2}{\sigma_1^2} \frac{\sigma_1^2 \left(\sigma^2 + \sigma_1^2\right)}{\sigma^2\sigma_1^2 + \sigma_1^4 + \sigma^2\sigma_1^2 + \sigma^2} \\
&= \frac{\sigma^2 \left(\sigma^2 + \sigma_1^2\right)^2}{\left(\sigma^2 + \sigma_1^2\right)^2} \\
&= \sigma^2
\end{aligned}$$

and the proof is complete. ∎

A.3 Proof of Theorem 6.3

Proof: Σ is positive definite iff for every vector $\boldsymbol{w} \in \mathbb{R}^{n+1}$ such that $\boldsymbol{w} \neq \mathbf{0}$ we have $\boldsymbol{w}^T \Sigma \boldsymbol{w} > 0$. Let $\boldsymbol{w} \in \mathbb{R}^{n+1}$ be such a vector. We can describe $\boldsymbol{w}$ as

$$\boldsymbol{w} = \begin{pmatrix} \\ \boldsymbol{x} \\ \\ \hline y \end{pmatrix}$$

where $\boldsymbol{x}$ is a vector of dimension n and y is a scaler.

$$\begin{aligned}\boldsymbol{w}^T \Sigma \boldsymbol{w} &= \begin{pmatrix} \boldsymbol{x}^T & | y \end{pmatrix} \left[\begin{array}{c|c} A & \boldsymbol{u} \\ \hline \boldsymbol{u}^T & v \end{array}\right] \begin{pmatrix} \boldsymbol{x} \\ \hline y \end{pmatrix} \\ &= \boldsymbol{x}^T A \boldsymbol{x} + 2y\boldsymbol{x}^T \boldsymbol{u} + y^2 v \end{aligned} \tag{A.4}$$

We therefore have a quadratic function in $\boldsymbol{x}$. Since A is positive definite the function must be convex in $\boldsymbol{x}$ and we can find its minimum by setting the derivative with respect to $\boldsymbol{x}$ to 0:

$$\begin{aligned} 2A\boldsymbol{x} + 2y\boldsymbol{u} &= 0 \\ \boldsymbol{x} &= -yA^{-1}\boldsymbol{u} \end{aligned} \tag{A.5}$$

Plugging Equation A.5 into Equation A.4 (and using the fact that $(A^{-1})^T = A^{-1}$) we get that Σ is positive definite iff $\forall y$

$$y^2 \boldsymbol{u}^T A^{-1} \boldsymbol{u} - 2y^2 \boldsymbol{u}^T A^{-1} \boldsymbol{u} + y^2 v > 0 \tag{A.6}$$

Equation A.6 holds for every y iff

$$v - \boldsymbol{u}^T A^{-1} \boldsymbol{u} > 0 \tag{A.7}$$

∎

Bibliography

[ACP87] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal of Algebraic and Discrete Methods*, 8(2):277–284, 1987.

[Ast65] K. J. Aström. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10:174–205, 1965.

[Atk89] K. Atkinson. *An Introduction to Numerical Analysis.* Wiley, 1989.

[BB72] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming.* Academic Press, New York, 1972.

[BBS88] H. A. P. Blom and Y. Bar-Shalom. The interacting multiple model algorithm for systems with Markovian switching coefficients. *IEEE Transactions on Automatic Control*, 33(8):780–783, August 1988.

[Ber95] D. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, Massachusetts, second edition, 1995.

[BK98] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proceedings of the 14th Annual Conference on Uncertainty in AI (UAI)*, pages 33–42, 1998.

[BKRK97] J. Binder, D. Koller, S. Russell, and K. Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29(2-3):213–244, 1997.

[BSLK01] Y. Bar-Shalom, X. R Li, and T. Kirubarajan. *Estimation with Application to Tracking and Navigation*. John Wiley & Sons, 2001.

[BV03] S. Boyd and L. Vandenberghe. *Convex Optimization*. 2003. To appear.

[CD00] J. Cheng and M. J. Druzdzel. AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large bayesian networks. *Journal of Artificial Intelligence Research*, 13:155–188, 2000.

[CLR90] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1990.

[Coo90] G. Cooper. Probabilistic inference using belief networks is NP-hard. *Artificial Intelligence*, 42:393–405, 1990.

[CT91] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 1991.

[CT94] M. Cordier and S. Thiébaux. Event-based diagnosis for evolutive systems. Technical Report 819, IRISA, Cedex, France, 1994.

[Daw92] A. P. Dawid. Application of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, 2:25–36, 1992.

[DGA00] A. Doucet, S. Godsill, and C. Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10(3):197–208, 2000.

[DK89] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

[dKW87] J. de Kleer and B. C. Williams. Diagnosing multiple faults. In Matthew L. Ginsberg, editor, *Readings in Nonmonotonic Reasoning*, pages 372–388. Morgan Kaufmann, Los Altos, California, 1987.

[dKW89] J. de Kleer and B. C. Williams. Diagnosis with behavioral modes. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1324–1330, 1989.

[DL93] P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153, March 1993.

[DLR77] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39 (Series B):1–38, 1977.

[DR84] P. J. Davis and P. Rabinovitz. *Methods of Numerical Integration*. Academic Press, 1984.

[Fra90] P. M. Frank. Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy: a survey and some new results. *Automatica*, 26:459–474, 1990.

[Ger88] J. J. Gertler. *Fault Detection and Diagnosis in Engineering Systems*. Marcel Dekker, New York, 1988.

[Gew89] J. Geweke. Bayesian inference in econometric models using Monte Carlo integration. *Econometrica*, 57(6):1317–1339, November 1989.

[GG84] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, November 1984.

[GH00] Z. Ghahramani and G. E. Hinton. Variational learning for switching state-space models. *Neural Computation*, 12(4):831–864, 2000.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman, New York, 1979.

[GK01] C. Goodrich and J. Kurien. Continuous measurements and quantitative constraints — challenge problems for discrete modeling techniques. In *Proceedings of iSAIRAS*, 2001.

[Goo02] C. Goodrich. Reverse water gas shift system presentation. Stanford University, April 2002.

[GSS93] N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings-F*, 140(2):107–113, April 1993.

[GTS94] W. R. Gilks, A. Thomas, and D. J. Spiegelhalter. A language and program for complex Bayesian modelling. *The Statistician*, 43:169–78, 1994.

[HBH+98] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 256–265, 1998.

[HBR94] D. Heckerman, J. S. Breese, and K. Rommelse. Troubleshooting under uncertainty. Technical Report MSR-TR-94-07, Microsoft Research, 1994.

[HD96] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15:225–263, 1996.

[HKKJ02] E. Horvitz, P. Koch, C. Kadie, and A. Jacobs. Coordinate: Probabilistic forecasting of presence and availability. In *Proceedings of the 18th Annual Conference on Uncertainty in AI (UAI)*, pages 224–233, 2002.

[Ise97] R. Isermann. Supervision, fault-detection and fault-diagnosis methods — an introduction. *Control Engineering Practice*, 5(5):638–652, 97.

[Jel97] F. Jelinek. *Statistical Methods for Speech Recognition.* MIT Press, Cambridge, MA, 1997.

[Jen96] F. V. Jensen. *An Introduction to Bayesian Networks.* UCL Press, London, 1996.

[JU97] S. Julier and J. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defence Sensing, Simulation and Controls*, 1997.

[Kal60] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82:34–45, 1960.

[Kim94] C-J. Kim. Dynamic linear models with Markov-switching. *Journal of Econometrics*, 60:1–22, 1994.

[Kit96] G. Kitagawa. Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal Of Computational and Graphical Statistics*, 5(1):1–25, 1996.

[Kja90] U. Kjaerulff. Triangulation of graphs – algorithms giving small total state space. Technical Report TR R 90-09, Department of Mathematics and Computer Science, Strandvejen, Aalborg, Denmark, 1990.

[KK97] A. Kozlov and D. Koller. Nonuniform dynamic discretization in hybrid networks. In *Proceedings of the 13th Annual Conference on Uncertainty in AI (UAI)*, pages 314–325, 1997.

[KN00] J. Kurien and P. P. Nayak. Back to the future for consistency-based trajectory tracking. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI)*, pages 370–377, 2000.

[KP97] D. Koller and A. Pfeffer. Object-oriented Bayesian networks. In *Proceedings of the 13th Annual Conference on Uncertainty in AI (UAI)*, pages 302–313, 1997.

[Lau92] S. L. Lauritzen. Propagation of probabilities, means, and variances in mixed graphical association models. *JASA*, 87(420):1089–1108, 1992.

[Lau96] S. Lauritzen. *Graphical Models*. Oxford University Press, New York, 1996.

[Lei89] H.-G. Leimer. Triangulated graphs with marked vertices. In L. D. Andersen, I. T. Jakobsen, C. Thomassen, B. Toft, and P. D. Vestergaard, editors, *Graph Theory in Memory of G.A. Dirac*, volume 41 of *Annals of Discrete Mathematics*, pages 311–324. Elsevier/North-Holland, 1989.

[LG00] W. Larson and C. Goodrich. Intelligent systems software for human Mars missions. In *2000 International Aeronautical Foundation Congress*, 2000.

[Liu96] J. Liu. Metropolized independent sampling with comparisons to rejection sampling and importance sampling. *Statistics and Computing*, 6:113–119, 1996.

[LJ01] S. L. Lauritzen and F. Jensen. Stable local compuation with conditional Gaussian distributions. *Statistics and Computing*, 11:191–203, 2001.

[LMS+02] U. Lerner, B. Moses, M. Scott, S. McIlraith, and D. Koller. Monitoring a complex physical system using a hybrid dynamic Bayes net. In *Proceedings of the 18th Annual Conference on Uncertainty in AI (UAI)*, pages 301–310, 2002.

[LP01] U. Lerner and R. Parr. Inference in hybrid networks: Theoretical limits and practical algorithms. In *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)*, pages 310–318, 2001.

[LPKB00] U. Lerner, R. Parr, D. Koller, and G. Biswas. Bayesian fault detection and diagnosis in dynamic systems. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI)*, pages 531–537, 2000.

[LS88] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, 50:157–224, 1988.

[LSK01] U. Lerner, E. Segal, and D. Koller. Exact inference in networks with discrete children of continuous parents. In *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)*, pages 319–238, 2001.

[McI98] S. McIlraith. Explanatory diagnosis: Conjecturing actions to explain observations. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 167–177, 1998.

[Min01] T. P. Minka. Expectation propagation for approximate Bayesian inference. In *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)*, pages 362–369, 2001.

[MRR+53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[MS67] J. McNamee and F. Stenger. Construction of fully symmetric numerical integration formulas. *Numerische Mathematick*, 10:327–344, 1967.

[MSH+91] B. Middleton, M. Shwe, D. Heckerman, M. Henrion, E. Horvitz, H. Lehmann, and G. Cooper. Probabilistic diagnosis using a reformulation of the INTERNIST-1/QMR knowledge base II: Evaluation of diagnostic performance. *Methods of Information in Medicine*, 30:256–267, 1991.

[Mur98] K. Murphy. Inference and learning in hybrid bayesian networks. Technical Report CSD-98-990, Department of Computer Science, U.C. Berkeley, 1998.

[Mur99] K. Murphy. A variational approximation for Bayesian networks with discrete and continuous latent variables. In *Proceedings of the 15th Annual Conference on Uncertainty in AI (UAI)*, pages 467–475, 1999.

[Nea93] R. Neal. Probabilistic inference using Markov Chain Monte Carlo methods. Technical Report CRG-TR-93-1, University of Toronto, 1993.

[Nil98] D. Nilsson. An efficient algorithm for finding the M most probable configurations in probabilistic expert systems. *Statistics and Computing*, 8(2):159 – 173, 1998.

[Oga01] K. Ogata. *Modern Control Engineering*. Prentice Hall, Upper Saddle River, NJ, fourth edition, 2001.

[OK00] L. E. Ortiz and L. P. Kaelbling. Adaptive importance sampling for estimation in structured domains. In *Proceedings of the 16th Annual Conference on Uncertainty in AI (UAI)*, pages 446–454, 2000.

[Par02] J. Park. MAP complexity results and approximation methods. In *Proceedings of the 18th Annual Conference on Uncertainty in AI (UAI)*, pages 388–396, 2002.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, California, 1988.

[PPMH94] M. Pradhan, G. Provan, B. Middleton, and M. Henrion. Knowledge engineering for large belief networks. In *Proceedings of the 10th Canadian Conference on Artificial Intelligence*, pages 484–490, 1994.

[PRCM99] V. Pavlovic, J. Rehg, T.-J. Cham, and K. Murphy. A dynamic Bayesian network approach to figure tracking using learned dynamical models. In *Proc. ICCV*, 1999.

[PS96] G. J. Pappas and S. Sastry. Towards continuous abstractions of dynamical and control systems. In P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems IV*, volume 1273 of *Lecture notes in Computer Science*, pages 329–341. Springer Verlag, Berlin, Germany, 1996.

[PTVF88] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1988.

[Put94] M. Puterman. *Markov decision processes: Discrete stochastic dynamic programming.* Wiley, New York, 1994.

[Rab89] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[Rap02] C. Raphael. A hybrid graphical model for rhythmic parsing. *Artificial Intelligence*, 137(1–2):217–238, May 2002.

[SDHH98] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian approach to filtering junk e-mail. In *AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[SP89] R. Shacter and M. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Proceedings of the 5th Annual Conference on Uncertainty in AI (UAI)*, pages 221–230, 1989.

[SS90] P. P. Shenoy and G. R. Shafer. Axioms for probability and belief-function propagation. In *Proceedings of the 6th Annual Conference on Uncertainty in AI (UAI)*, pages 169–198, 1990.

[SS91] R. H. Shumway and D. S. Stoffer. Dynamic linear models with switching. *Journal of the American Statistical Association*, 86(415):763–769, September 1991.

[Sto96] C. Stone. *A Course in Probability and Statistics*. Duxbury Press, 1996.

[TK98] H. Taylor and S. Karlin. *An Introduction to Stochastic Modeling*. Academic Press, San Diego, third edition, 1998.

[TK01a] S. Tong and D. Koller. Active learning for parameter estimation in Bayesian networks. In *Advances in Neural Information Processing Systems 13*, pages 647–653, 2001.

[TK01b] S. Tong and D. Koller. Active learning for structure in Bayesian networks. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2, pages 863–869, 2001.

[TPS98] C. Tomlin, G. J. Pappas, and S. Sastry. Conflict resolution for air traffic management: A study in multi-agent hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):509–521, April 1998.

[TY84] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, August 1984.

[Whi01] J. E. Whitlow. Operation, modeling and analysis of the reverse water gas shift process. In *NASA CR-2001-(In Press)*, 2001.

[Wie00] W. Wiegerinck. Variational approximations between mean field theory and the junction tree algorithm. In *Proceedings of the 16th Annual Conference on Uncertainty in AI (UAI)*, pages 626–633, 2000.

[WN96] B. Williams and P. P. Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI)*, pages 971–978, 1996.

[ZP94] N. L. Zhang and D. Poole. A simple approach to Bayesian network computations. In *Proceedings of the 10th Canadian Conference on Artificial Intelligence*, pages 171–178, 1994.