Probabilistic Inference with Large Discrete Domains

by

Rita Sharma

M.Sc., University of British Columbia, 2001

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in
THE FACULTY OF GRADUATE STUDIES
(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

October 2006
© Rita Sharma, 2006
Abstract

The straightforward representation of many real world problems is in terms of discrete random variables with large or infinite domains. For example, in a domain where we are trying to identify a person, we may have variables that have as domains, a set of all names, a set of all postal codes, and a set of all credit card numbers. The task usually reduces to performing probabilistic inference, i.e., compute the probability of some values of some random variables given the values of some other variables. Bayesian networks are a compact way to represent joint probability distributions. This thesis is concerned with probabilistic inference in Bayesian networks that have discrete random variables with large or infinite domains.

Carrying out inference in Bayesian networks that have variables with large domains is a difficult problem. For efficient inference, we consider cases where there is some structure that can be exploited to make inference efficient. In this thesis we consider two kinds of structures that can be exploited for efficient inference. These structures allow us to partition the large number of values in equivalence classes. Rather than reasoning about every value of a variable individually, we can reason about a set of values in a single step.

We first consider the case where there are intensional definitions of the conditional probability distributions. To represent these conditional probabilities, we introduce a CPD language that allows us to define the conditional probabilities procedurally, in terms of predicates and functions. We present an inference algorithm, Large Domain VE, for the CPD language that uses this representation to partitions the domains of the variables dynamically. The partitions depend on what is observed and what is queried. We apply Large Domain VE to the person identification problem that has variables with large domains.

The second case we consider where there is a priori internal structure on the values of the variables. In particular, we consider the case where the values of
the variables are represented as tree hierarchies. We call such variables hierarchically structured variables. We present a language for representing the conditional probabilities of Bayesian networks with hierarchically structured variables. To perform inference in Bayesian networks with hierarchically structured variables we construct an abstract Bayesian network dynamically, given some evidence and a query, by collapsing the hierarchies to include only those values necessary to answer the query. We can answer the query from the abstract Bayesian network using any standard inference algorithm.

Finally, we show how both intensional definitions of the conditional probability distributions and hierarchically structured values can be put together to produce a general framework that can be applied to a more general class of problems.
Contents

Abstract ......................................................... ii

Contents ...................................................... iv

List of Tables ............................................... viii

List of Figures ................................................. ix

Acknowledgements ........................................... xiii

1 Introduction ................................................. 1
  1.1 Probabilistic Reasoning ............................... 1
  1.2 Thesis Overview ....................................... 6
  1.3 Summary of Thesis Contributions ................. 9
    1.3.1 Thesis Organization ............................ 10

2 Background ............................................... 11
  2.1 Introduction .......................................... 11
  2.2 Notation ............................................. 12
  2.3 Bayesian Networks .................................. 12
List of Tables

4.1 Conditional probability table $P(F1|H^a)$. .............. 150
4.2 Conditional probability table $P(P^a|H^a)$. .............. 150
List of Figures

2.1 An example of a Bayesian network. ........................................ 14
2.2 Conditional probability tables for the example network. .......... 15
2.3 The Variable Elimination Algorithm. ................................. 22
2.4 Initial set of factors for example 2.5.6. ............................... 23
2.5 (a) A simple Bayesian network modeling whether the person is injured. (b) The tabular representation of conditional probability \( P(I|C \land S) \). (c) The decision tree representation of conditional probability \( P(I|C \land S) \). .................................................... 32

3.1 Bayesian Network representation of attribute dependency for case \( X \neq Y \) (shaded variables are observed). ............................... 44
3.2 Bayesian Network representation of attribute dependency for case \( X = Y \) (shaded nodes are observed). ................................. 46
3.3 A CPD language representation of \( P(Fname.X|Afname \land Sex \land EF.X) \). .......................................................... 53
3.4 A decision tree representation of \( P(Fname.X|Afname \land Sex \land EF.X) \). 54
3.5 The decision tree representations \( T_1, T_2, \) and \( T_3 \) of conditional probabilities \( P(Fname,X|Afname\land Sex\land EF.X), P(Fname,Y|Afname\land Sex\land EF.Y), \) and \( P(Afname|Sex) \) respectively.

3.6 A BNF grammar of factor language.

3.7 A tree simplified by removal of redundant subtrees (triangles denote subtrees).

3.8 Algorithm for pruning the decision tree \( T \).

3.9 Merging trees \( T_1 \) and \( T_2 \). The leaf labels are combined using the plus function \( merge_2(T_1,T_2,+) \).

3.10 Merging trees \( T_1 \) and \( T_2 \). The leaf labels are combined using the multiplication function \( merge_2(T_1,T_2,\times) \).

3.11 Algorithm for merging merge \( T_1 \) and \( T_2 \) with operation \( Op \).

3.12 A decision tree representation, \( T \), of a factor corresponding to CPD \( P(Fname.X|Afname \land Sex \land EF.X) \).

3.13 Conditioned tree \( T_{Fname.X="david"} \), i.e., tree \( T \) after conditioning on \( Fname.X = \text{"david"} \).

3.14 Algorithm for conditioning the decision tree \( T \) on observations.

3.15 (a) A simple Bayesian network. (b) The decision tree representations of conditional probabilities \( P(X), P(W), \) and \( P(Y|W \land X) \).

3.16 The partitions of the values of \( W \) and \( X \).

3.17 Algorithm for summing out variables \( Y \) from tree \( T \).

3.18 Function \( \text{Compute.Evi.Factor} \) called by algorithm \( \text{Sum} \).

3.19 The Large Domain VE Algorithm.
3.20 The decision tree representations, \( T_1, T_2, \) and \( T_3, \) of factors \( f_1, f_2, \) and \( f_3 \) respectively. ................................................. 93

3.21 A decision tree representation, \( T, \) of new factor \( f \) after multiplying trees \( T_1, T_2, \) and \( T_3. \) ......................................................... 94

3.22 A decision tree representation, \( T', \) of new factor \( f' \) after summing out \( A\text{name} \) from tree \( T. \) ......................................................... 95

3.23 Recall versus precision for both attribute dependence and attribute independence. ........................................ 99

4.1 Part of a class hierarchy of living things (\( L. \) ......................................................... 110

4.2 (a) A Bayesian network with simple and hierarchical variables (b)
A part of the class hierarchy of \( R. \) ......................................................... 113

4.3 (a) A Bayesian network with simple and hierarchical variables. (b)
Part of the class hierarchies for variables \( P \) and \( Q. \) (c) parent context hierarchy of the given parent contexts that define \( P(Z|P \land Q). \) 124

4.4 The class hierarchy of \( L \) is divided into three regions \( R_{true}, R_{unknown}, \)
and \( R_{false} \) by positive observation (\( animal \)) and negative observations (\( bat, penguin \)) about \( L. \) ......................................................... 129

4.5 A Bayesian network with simple and hierarchical variables. ......................................................... 131

4.6 Algorithm Abstract_Bayes for constructing abstract Bayesian network. ......................................................... 137

4.7 Functions AbstractHier called by the algorithm Abstract_Bayes ......................................................... 138

4.8 Functions AbstractObs called by the algorithm Abstract_Bayes ......................................................... 139

4.9 Function Construct_Table called by the Abstract_Bayes. ......................................................... 142
4.10 Function Simple_Var_Table called by the Construct_Table. .... 142
4.11 Function Hier_Var_Table called by the Construct_Table. .... 143
4.12 A Bayesian network with simple and hierarchical variables (shaded
variables are observed) .................. 146
4.13 The Bayesian network of Figure 4.12 with trees hierarchies for the
hierarchical variables. Thin lines show the class-subclass relationship and
thick lines show the Bayesian network dependency. Exceptional classes are marked by *. .... 147
4.14 (a) Bayesian network representation of the conference - academics
domain. (b) A part of the class hierarchy for location in the world. ... 158
4.15 The average inference time of applying VE and Abstract_Bayes +
VE for $k = 2$ as a function of the depth of the hierarchy, and depth
of the observations. ................... 163
4.16 The average inference time of applying VE and Abstract_Bayes +
VE for $d = 8$ as a function of the number of academics and the depth
of the observations. ................... 164
5.1 (a) A Bayesian network modelling rock types in a region. (b) The
class hierarchies of Location and RockType. ............... 170
Acknowledgements

I am incredibly grateful to Dr. David Poole, my research supervisor, for the countless hours of discussions throughout the course of my work. His advice, comments, and suggestions concerning this thesis were invaluable. Without his extensive support (both financially and technically), patience and endless encouragement, my thesis would not have been possible.

Sincere thanks to Dr. Alan Mackworth, Dr. Nando de Freitas, and Dr. Anne Condon, members of my supervisory committee for their valuable feedback on my thesis draft, and great recommendations on this research project.

I would like to thank Dr. Marek J. Druzdzel (External Examiner), Dr. James J. Little (University Examiner), Dr. Paul Gutafson (University Examiner), and Dr. Bertrand Clarke (Examination Chair) for the time they spent reading and reviewing my work.

I am very grateful to Valerie McRae, the LCI Ex-Secretary for her support during the course of this thesis. She was always ready for me, providing help, proofreading (a lot), and supportive comments on a regular basis.

I thank my LCI friends Mark Crowley, Jacek Kisynski, and Michael Chi-ang, for their patience to listen my practice talks, providing feedback, and some valuable discussion. I would like to thank Craig Wilson for taking the time to proof read my thesis draft. Thanks to faculty members, staff, and fellow students in the Department for their help and encouragement.

I am indebted to my mom, who helps me in a different way to finish this thesis, in the course of this thesis she came to Vancouver three times, all the way from India to take care of my little one. Without her support, this work would not have been completed. I thank my father, father-in-law, and mother-in-law for their blessings and encouragement. I thank my friends and family for their encouragement and support.

Lastly but not least, I thank my husband, Sudhir, and my loving sons, Ashu
and Amit, for their sacrifices, unconditional love, encouragement and support.

RITA SHARMA

The University of British Columbia
October 2006
Chapter 1

Introduction

1.1 Probabilistic Reasoning

Reasoning under uncertainty is an important topic in artificial intelligence (AI), since any intelligent agent must be able to take action under uncertainty. Uncertainty is an unavoidable feature of the world. It arises in a variety of ways. An agent's sensors usually provide incomplete knowledge about the world, and the knowledge it does receive is often noisy. Also, its actions can have unpredictable effects.

Probability theory provides a sound mathematical basis for reasoning under uncertainty. It allows us to maintain our beliefs with incomplete (partial) knowledge. It allows us to update our beliefs by incorporating new pieces of evidence obtained from various sensors in a coherent way. Despite this, for many years the AI researcher did not use probabilistic models. The most important reason for not using probability theory was the inability to represent and exploit the structure of
large probabilistic models.

This situation changed in the late 1980s with the development of Bayesian networks [Pearl, 1988]. A Bayesian network is a directed acyclic graph of nodes, which represent random variables taking a set of values, and arcs, which represent direct probabilistic dependencies among these variables. Each node is associated with a *Conditional Probability Distribution (CPD)* defining the conditional distribution of the variable given its parents. The graph structure represents the conditional independence relationship that exists between different variables, and the parameters of conditional probability distributions represent the strength of probabilistic dependencies [Pearl, 1988]. Bayesian networks provide a compact representation for joint probability distributions (the probability distribution over the joint state space of all variables in the network) by a product of all conditional probabilities associated with the nodes in the network. Over the past fifteen years or so, Bayesian networks have emerged as the leading technology for probabilistic reasoning.

The fundamental issue for Bayesian networks is the problem of probabilistic inference. Given a Bayesian network, we would like to compute the probability distribution over the values of some random variables, given the fact that we know the values of some other variables. For example, in medical diagnosis we can query the probability of a particular disease given the results of a physical examination and tests performed. Probabilistic inference in a Bayesian network is computationally intensive and is known to be NP-hard in general [Cooper, 1990]. The computation cost grows exponentially with the size of a Bayesian network. Note that even
approximate inference is computationally difficult in the worst cases [Dagum and Luby, 1993].

Recently there has been interest in speeding up inference by extending the Bayesian networks representation through more structured representations of the conditional probability of a variable, given its parents (for example, in terms of causal independence [Zhang and Poole, 1996] or context specific independence (CSI) [Poole and Zhang, 2003; Boutilier, Friedman, Goldszmidt and Koller, 1996]). In all of these approaches, discrete random variables are considered to have a bounded number of values.

However, in many real problems the discrete random variables could have very large or even unbounded domains. For example, in a Bayesian network we may have random variables that have as domains the set of all names, all postal codes, and all phone numbers. The large number of values of the variables are a problem for many reasons:

- Specifying conditional probabilities in conditional probability tables (CPTs) requires a complete specification of CPT entries. The number of parameters required to describe the CPT of a variable $X$ is $|X| \times |Pa(X)|$, where $Pa(X)$ denotes the parents of $X$. For CPTs with large $|X|$ or $|Pa(X)|$, specifying all the parameters is not feasible.

- Even if one could specify all the parameters for big CPTs, a big tabular representation is not efficient in terms of memory.

- The complexity of exact probabilistic inference in a Bayesian network is exponential in tree width, where the base of the exponent is the domain size.
• One could potentially have a big tabular representation and use the previous approaches for probabilistic inference. However, we cannot represent the conditional probability distribution of a variable conditioned on a variable that has an unbounded number of values in tabular form. For example, consider a random variable that has as its domain the set of all first names. This domain may never be known to the full extent because people can make up names. We need a compact representation for representing the large (or unbounded) CPTs so that we can reason in an efficient manner.

In some problems we might have large-domain variables with a priori hierarchical structure on their values. Almost anything – objects, Geo-systems, university courses etc. – may be classified according to some taxonomic scheme. The classic example is the taxonomic classification of living things (i.e., the millions of species) according to Linnaean taxonomy hierarchy\(^1\). Living things are divided into kingdoms (e.g., plantae, animalia), classes (e.g., mammals, birds, fish), all the way down to species. The hierarchical structure over the values of a variable can be exploited to make the representation compact and reasoning efficient.

This thesis is concerned with the representations and inference algorithms for probabilistic reasoning in complex domains that have discrete random variables with large domains, including the case where there are unboundedly many discrete values.

To perform efficient inference in Bayesian networks that have discrete variables with large domains, the approach we propose in this thesis is based on a

\(^1\)http://www.wikipedia.org/wiki/Linnaean\_taxonomy
structured representation of the problem. The structured representation allows the exploitation of regularities and independencies in the problem that can reduce the "effective state space" of the variables. This reduction has an immediate effect on the computation of the query, and the storage required. We consider the following types of structures:

- We consider that there is some structure in the conditional probability tables that, given an observation and query, allows us to partition the large number of values of the variables in equivalence classes that have the same conditional probability. We can take advantage of the structure by reasoning about these classes together. In particular, we do not need to reason about every value of the large-domain variables individually. Rather, we can reason about a block of values together.

- We consider that there is a priori internal structure on the values of large domain variables. That is, the large number of values of the variables can be represented as specialization/feature hierarchies. We can exploit the structure provided by feature hierarchies in probabilistic inference. Feature hierarchies aggregate the values in meaningful ways. We can support the compact representation for the local probability model using inheritance, thus encoding constraints on the contexts in which the conditional probabilities are equal.

The idea of utilizing structure in the CPTs is standard [Boutilier et al., 1996; Zhang and Poole, 1996]. The difference is that we are applying it to a more general class of problems, where we can have discrete random variables with small, large
or unbounded domains. Similarly, feature hierarchies and semantic networks have been used in AI for a long time. These graph-based formalisms, are not easily amenable for representing uncertainty in an elegant and efficient manner. There has been some work that combines belief network and feature hierarchies [Pearl, 1986; Poh and Fehling, 1993; Heckerman, 1990]. But in these works, the Bayesian networks are restricted to a diagnostic network form (or naive Bayes classifier) with a single hypothesis variable. In this thesis, we utilize the feature hierarchies in a general Bayesian network that has both small and large-domain random variables, where any variable can have any type and any number of parents.

In summary, this thesis is concerned primarily with the representation and the probabilistic inference for Bayesian networks that contain both small and large-domain (or unbounded) discrete random variables, where there is some structure. The structure could either be in CPDs, or be due to the feature hierarchies, or both.

1.2 Thesis Overview

In this thesis, we present two structured probabilistic inference algorithms that exploit the structure in the problem for efficient reasoning in Bayesian networks that have discrete random variables with large domains.

To represent the conditional probabilities in a compact manner for Bayesian networks that have variables with large domain, we first consider that CPDs can be represented compactly using both intensional (in terms of functions and predicates) and extensional (by listing the values) definitions. The computation of intensional definitions may involve looking up the tables and computing the predicates and
functions. To capture such distributions, we propose a CPD language for representing the CPDs in a compact manner.

To deal with the complexity of inference, we exploit the fact that when there is no evidence or query to distinguish between all the values of a variable, we do not need to reason about each value separately. Rather, we can reason about a group of values together as a single entity. We develop a structured inference algorithm, Large Domain VE, that is based on the Variable Elimination algorithm, for making inference in Bayesian networks that have variables with large domain. Large Domain VE partitions the domains of the variables dynamically during the inference based on the evidence and query.

In this thesis, we motivate our work on large domain variables using the "person identification problem" [Gill, 1997; Bell and Sethi, 2001]. This is the problem of identifying whether two records that contain information about the demographic attributes of people (e.g., firstname, lastname, date of birth) refer to the same person. Standard methods [Fellegi and Sunter, 1969] treat the attributes as independent of each other, i.e., the matching on one attribute does not depend on other attributes. We have relaxed this assumption to model how the attributes are interdependent. Bayesian networks of attribute dependence contain many variables with large domains. For example, we have random variables that have as domains the set of all first names and the set of all postal codes. To represent the large CPTs of these networks, we need some compact representation. As we shall see in Chapter 3, we represent the large CPTs in a compact manner using intensional and extensional representation. We present the application of Large Domain VE to the person iden-
Chapter 4 considers the case where there is a priori hierarchical structure on the values of the variables. That is, the values of the variables can be represented as specialization / feature hierarchies. We call such variables hierarchically structured variables. To exploit the structure provided by feature hierarchies in probabilistic reasoning we need a representation language for representing the CPDs in a compact manner. We represent the distribution for the hierarchical variables by specifying, for each class, the probability distribution over its immediate subclasses. To represent the conditional probability distribution of any variable conditioned on a hierarchical variable, we use inheritance.

To perform efficient inference in Bayesian networks with hierarchically structured variables, we dynamically construct an abstract Bayesian network, given some evidence and a query, by collapsing the hierarchies to include only those values necessary to answer the query. We can answer the query from the abstract Bayesian network using any standard probabilistic inference algorithm. The domain size of the variables in the abstract Bayesian network is independent of the size of the hierarchies; it depends on how many of the classes in the hierarchies are supported directly by the evidence or are relevant to the query. Thus, the proposed approach is applicable even when the hierarchies are conceptually infinite.

Finally, we put both CPD language representation and feature hierarchies together to produce a general framework that can be applied to a more general class of problems that have both small and large-domain variables with or without feature hierarchies.
1.3 Summary of Thesis Contributions

The contributions of this thesis are summarized as follows:

- To represent the CPDs of a Bayesian network that has variables with large domains in a compact manner, we introduce a CPD language. The CPD language allows us to represent the large conditional probabilities compactly using intensional and extensional definitions. The CPD language generalizes the idea of context specific independence [Boutilier et al., 1996], because the contexts are not only given by expressions such as $variable_i = val_j$ but also by expressions such as $foo(variable_i, variable_j) = true$.

- We provide a structured efficient inference algorithm, Large Domain VE, that exploits the CPD language representation. Large Domain VE dynamically partitions the values of large-domain variables, based on the evidence and query.

- We demonstrate how to apply Large Domain VE to the person identification problem.

- We provide a systematic way of dealing with hierarchically structured variables in probabilistic reasoning. We provide the representation techniques for representing the CPDs in a compact manner.

- We provide an algorithm to perform efficient inference in Bayesian networks with hierarchically structured variables. The proposed approach is applicable even when the hierarchies are conceptually infinite.
We develop a general framework that exploits both intensional definition of the conditional probabilities and hierarchically structured values of the variables.

1.3.1 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, we provide a formal definition of Bayesian networks and describe an inference method. The contributions of this thesis begin in Chapter 3, where we consider how to exploit the structure in large discrete domains. We discuss in brief the "person identification problem". We present the representation and inference algorithm for making inference in Bayesian networks that have variables with large domains. In Chapter 4, we introduce the notion of hierarchically structured variable. We present the representation and inference algorithm for exploiting the hierarchical structure in probabilistic inference. We present some empirical results that show that hierarchically structured values of a variable can make reasoning more efficient than reasoning over the set of all the values. In Chapter 5, we integrate both kinds of structures in a common framework so that it can be applied to a more general class of problems. Finally, in Chapter 6, we summarize the main points of this thesis, and future directions that could be taken with this work.

Some of the ideas presented in the above chapters were previously published as technical papers in AI conferences [Sharma and Poole, 2003; Sharma, 2004; Sharma and Poole, 2005a; Sharma and Poole, 2005b].
Chapter 2

Background

2.1 Introduction

Bayesian networks (also known as Belief networks, Bayes nets) are a dominant technique for representing uncertainty knowledge in AI [Pearl, 1988]. Bayesian networks exploit conditional independence between variables to compactly represent the joint probability distribution. This chapter presents an introduction to Bayesian networks. We begin by defining Bayesian networks, then discuss the conditional independence relationships encoded in their structure, and finally an inference algorithm for performing probabilistic inference in Bayesian networks. None of the material presented in this chapter is new; readers who are familiar with this background can skip to the next chapter. In this chapter, we concentrate on discrete models where all the variables have small domains. The techniques presented in this chapter do not work very well with variables that have large domains. In subsequent chapters we shall concern ourselves with networks that have variables with
very large (or infinite) domains.

2.2 Notation

We denote a random variable by upper case letter (e.g., $X$, $Y_i$, $Z$), and the actual value of that variable by the same letter in lower case (e.g., $x$, $y_i$, $z$). We use $Val(X)$ to denote the set of all possible values that a random variable $X$ can take. We instantiate a variable $X$ by assigning it a value $x \in Val(X)$, denoted by $X = x$. We denote a set of random variables by bold-face upper case letter (e.g., $X$, $Y_1$, $Z$); the corresponding bold-face lower case letter (e.g., $x$, $y_1$, $z$) denotes an assignment for each variable in a given set. The function $Val(X)$ is the cross product of the domains of the variables in $X$. We instantiate a set of variables $X$ by assigning a value to each variable in the set, denoted by $X = x$.

We use $P(X)$ to denote the probability distribution for a variable $X$. We use $P(X = x)$ (or $P(x)$) to denote the probability that $X$ takes the value $x$. We use $P(X|Y)$ to denote the conditional probability of $X$ given $Y$, and $P(X|Y = y)$ denotes the conditional distribution of $X$ given $Y = y$. We use $Pa(X)$ to denote all the parents of a variable $X$ in a Bayesian network.

2.3 Bayesian Networks

A Bayesian network is a compact representation of a joint probability distribution over a set of random variables $X = \{X_1, \ldots, X_n\}$. A Bayesian network is a directed, acyclic graph $\mathcal{G}$ defining a probability distribution over $X$. Each node in $\mathcal{G}$
corresponds to a random variable, and edges in $\mathcal{G}$ represent the direct dependence between the variables. We use the terms node and random variable interchangeably. The parents of node $X$, denoted by $Pa(X)$, are all the nodes in $\mathcal{G}$ that have directed edges going into $X$. Each variable $X$ in a Bayesian network is associated with a conditional probability distribution (CPD), denoted by $P(X|Pa(X))$. If $X$ has no parents, then it is a prior probability of $X$, denoted by $P(X)$. The CPD for a variable $X$ is often specified explicitly in tabular form by listing the probability distribution over the values of $X$ given every possible instantiation of its parents $Pa(X)$. In this case, the CPD is called a *conditional probability table* (CPT).

The graph $\mathcal{G}$ represents a set of conditional independencies: every node in the graph is independent of its non-descendants, given its parents. If we total order the variables $X_1, \ldots, X_n$ such that $Pa(X_i) \subseteq \{X_1, \ldots, X_{i-1}\}$, then

$$P(X_i|X_{i-1}, \ldots, X_1) = P(X_i|Pa(X_i))$$

(2.1)

The joint probability distribution induced by a Bayesian network is defined using the *chain rule* and the conditional independence assumption. Using the chain rule, the joint probability distribution over $\{X_1 \ldots X_n\}$ can be computed as follows:

$$P(X_1 \ldots X_n) = \prod_{i=1}^{n} P(X_i|X_{i-1}, \ldots, X_1)$$

After applying the conditional independence assumption as defined in equation (2.1), the joint probability distribution $P(X_1 \ldots X_n)$ can be computed as follows:

$$P(X_1 \ldots X_n) = \prod_{i=1}^{n} P(X_i|Pa(X_i))$$

(2.2)

More formally, we can define a Bayesian network as follows:
Definition 2.3.1 A Bayesian network $\mathcal{B}$ over random variables $X = \{X_1, \ldots, X_n\}$ is a pair $(\mathcal{G}, \theta)$, where

- $\mathcal{G}$ is a directed acyclic graph over variables $X$, with one node for every variable $X_i$,
- $\theta$ is a set of conditional probability distributions:

$$\theta = \{ P(X_i | Pa(X_i)) \mid X_i \in X, \ Pa(X_i) \text{ are the parents of } X_i \text{ in } \mathcal{G} \}$$

The joint probability distribution represented by $\mathcal{B}$ is given by equation (2.2).

Example 2.3.2 Figure 2.1 presents a medical Bayesian network that models the lung cancer problem. The domain is modeled here through six binary variables; Smoking ($S$), Pollution ($P$), Bronchitis ($B$), Lung Cancer ($L$), Fatigue ($F$), and Positive chest X-Ray ($X$) are all Boolean \{true, false\} variables. This slightly modified Bayesian network model is taken from [Neapolitan, 2004]. The graph structure of a Bayesian network reflects the causal structure of the domain. Lung cancer and
Figure 2.2: Conditional probability tables for the example network.

Bronchitis are more often encountered in smokers, and we indicate this by setting $S$ to be a parent of both $L$ and $B$. Pollution can increase the probability of lung cancer, and we indicate this by setting $P$ to be a parent of $L$. Both lung cancer and bronchitis can cause fatigue, so the parents of $F$ are $L$ and $B$. A person with lung cancer usually has a positive chest X-ray, so the parent of $X$ is $L$.

Figure 2.2 shows a set of possible conditional probability tables for the Bayesian network shown in Figure 2.1. The entries in the CPT encode the parameters of the probability distribution. For example, $P(P = true) = 0.8$ and $P(B = true|S = true) = 0.25$. 

<table>
<thead>
<tr>
<th>$S$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>0.25</td>
</tr>
<tr>
<td>false</td>
<td>0.05</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$B$</th>
<th>$L$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>0.75</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>0.10</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>0.50</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>0.05</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
</tr>
<tr>
<td>0.80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$S$</th>
<th>$P$</th>
<th>$L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>0.05</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>0.03</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>0.02</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>0.001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$L$</th>
<th>$X$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>0.90</td>
</tr>
<tr>
<td>false</td>
<td>0.20</td>
</tr>
<tr>
<td>false</td>
<td>0.80</td>
</tr>
</tbody>
</table>
2.4 Conditional Independence

A Bayesian network is a compact representation of a joint probability distribution. A joint probability distribution over $n$ binary random variables needs $2^n - 1$ independent parameters. However, a Bayesian network representation over $n$ binary random variables requires at most $n \times 2^k$ independent parameters, where $k$ is the maximum number of parents of a variable. This is because the graph structure of a Bayesian network forces certain conditional independencies to hold, irrespective of the parameters of the conditional probability distributions.

**Definition 2.4.1** Let $X = \{X_1, \ldots, X_n\}$, $Y = \{Y_1, \ldots, Y_n\}$, and $Z = \{Z_1, \ldots, Z_n\}$ be three disjoint sets of random variables, then $X$ is conditionally independent of $Y$ given $Z$, if for all $x \in \text{Val}(X)$, $y \in \text{Val}(Y)$, and $z \in \text{Val}(Z)$,

$$P(X = x, Y = y | Z = z) = P(X = x | Z = z)P(Y = y | Z = z)$$

When $X$ and $Y$ are conditionally independent given $Z$, we write as $\text{I}(X, Y | Z)$.

For example, in the Bayesian network shown in Figure 2.1, variable $F$ is not independent of $S$; however, it is independent of $S$ given its parents $L$ and $B$, which means $\text{I}(F, S | \{L, B\})$.

2.5 Inference

Almost every system that uses Bayesian networks must be able to perform probabilistic inference, i.e., find the distribution over some variables given some evi-
dence. Given a Bayes net $B$ over the variables $X$, we typically need to compute the distribution $P(Q|evidence)$, where $Q \subseteq X$ is called query variables.

Very often, evidence is considered as an instantiation of some of the variables in the network, i.e., evidence denotes $E = e$, $E \subseteq X$, $e \in Val(E)$. However, one may also be interested in computing the posterior distribution of $Q$, given disjunctive evidence (more general evidence), e.g., $X_1 = x_{11} \lor X_1 = x_{12}$. That is, we only know that the value that a variable can be assigned belongs to a set of values, but we do not know which value. We therefore consider that $E \in e^*$, $e^* \subseteq Val(E)$. The new joint probability distribution after inserting the evidence can be computed as follows:

$$P(X_1, \ldots, X_n)_{E \in e^*} = \prod_{i=1}^{n} P(X_i|Pa(X_i))_{E \in e^*}$$

That is, inserting evidence into a joint probability distribution means that every conditional probability $P(X_i|Pa(X_i))$ is replaced by new conditional probability $P(X_i|Pa(X_i))_{E \in e^*}$, which can be computed as follows:

- If $(\{X_i\} \cup Pa(X_i)) \cap E = \emptyset$
  $$P(X_i|Pa(X_i))_{E \in e^*} = P(X_i|Pa(X_i))$$

- Otherwise, let $Z = (\{X_i\} \cup Pa(X_i)) \cap E$
  $$P(X_i|Pa(X_i))_{E \in e^*} = \begin{cases} P(X_i|Pa(X_i)) \forall z \in Val(Z) \text{ such that } z \in e^* \\ 0 & \text{otherwise} \end{cases}$$

$P(Q|E \in e^*)$ is the conditional probability distribution over the query variable $Q$, given the evidence $E \in e^*$. Let $\{Y_1, \ldots, Y_s\}$ be the non-query random variables.
variables of $B$. The query $P(Q|E \in e^t)$ can be computed as follows:

$$P(Q|E \in e^t) = \frac{\sum_{Y_1, \ldots, Y_n} P(X_1, \ldots, X_n)_{E \in e^t}}{\sum_Q \sum_{Y_1, \ldots, Y_n} P(X_1, \ldots, X_n)_{E \in e^t}}$$ \hspace{1cm} (2.3)

The denominator in equation (2.3) does not depend on $Q$. Thus, we only need to compute $\sum_{Y_1, \ldots, Y_n} P(X_1, \ldots, X_n)_{E \in e^t}$, which is a function of $Q$:

$$f(Q) = \sum_{Y_1, \ldots, Y_n} P(X_1, \ldots, X_n)_{E \in e^t}.$$ 

We can consider $P(Q|E \in e^t)$ as a function that assigns each $q \in Val(Q)$ a conditional probability $P(Q = q|E \in e^t)$. The conditional probability $P(Q = q|E \in e^t)$ can be computed as follows:

$$P(Q = q|E \in e^t) = \frac{f(Q = q)}{\sum_{Q \in Val(Q)} f(Q)}$$

where $\sum_{Q \in Val(Q)} f(Q)$ is a normalizing constant that normalizes the values of $f(Q)$ so that they sum to 1.

Thus, the problem of probabilistic inference reduces to the problem of computing $f(Q)$. Theoretically, $f(Q)$ can be computed from the joint probability distribution by summing out non-query variables one by one. For example, suppose we are interested in the query $P(X|F = true)$ in the Bayesian network shown in Figure 2.1. To compute $P(X|F = true)$, we need to compute $f(X)$, which can be computed as follows:

$$f(X) = \sum_{S,P,L,B} P(S,P,L,B,X,F)_{F \in \{true\}}$$

$$= \sum_{S,P,L,B} P(S) \times P(P) \times P(L|S,P) \times$$

$$P(B|S) \times P(F = true|B,L) \times P(X|L)$$

18
The variables $S$, $P$, $L$, and $B$ are Boolean variables. Thus, to compute $f(X)$ we need to sum 16 different terms, and that in general the number of terms we need to sum might be exponential in the number of variables in the network. Thus, summing out a variable from a joint probability distribution is not practically viable. It has been proved that the probabilistic inference in Bayesian networks is NP-hard [Cooper, 1990]. However, in many cases we can exploit the structure of the Bayesian network to perform probabilistic inference efficiently. In the next section, we discuss an exact inference algorithm.

### 2.5.1 Variable Elimination

The Variable Elimination (VE) algorithm [Zhang and Poole, 1994; Dechter, 1996] is a query oriented algorithm for probabilistic inference that exploits the conditional independence inherent in the Bayesian network for efficient inference.

As discussed above, to compute the query $P(Q|E \in \mathcal{e}^s)$ we need to compute the function $f(Q)$.

$$f(Q) = \sum_{Y_5} \cdots \sum_{Y_1} P(X_1, \ldots, X_n)_{\{E \in \mathcal{e}^s\}}$$

$$= \sum_{Y_5} \cdots \sum_{Y_1} \prod_{i=1}^{n} P(X_i|Pa(X_i))_{\{E \in \mathcal{e}^s\}} \quad (2.4)$$

where $Y_1, \ldots, Y_5$ are the non-query variables of the Bayesian network and $E \in \mathcal{e}^s$ is the evidence.

The problem of probabilistic inference is thus reduced to the problem of summing out variables from a product of functions. Before showing in detail how VE computes the function $f(Q)$, we introduce some terminology.
Definition 2.5.1 A factor over variables X is a function from $\text{Val}(X)$ into the real numbers. A factor can be represented as a $d$-dimensional table, where $d$ is the number of variables in the factor. Each row of the table corresponds to a specific instantiation of the factor variables.

Definition 2.5.2 Let $f$ be a factor over $n$ variables and $b$ be the maximum domain size of a variable in $f$. The size of factor $f$ is $O(b^n)$.

Definition 2.5.3 Let $f_1$ be a factor over variables $Y$, and $f_2$ be a factor over variables $Z$. The product of factors $f_1$ and $f_2$, denoted by $f_1 \otimes f_2$, is a factor over $Y \cup Z$ defined by:

$$f_1 \otimes f_2(Y \cup Z) = f_1(Y) \times f_2(Z)$$

The product of factors is commutative and associative. To construct the product of factors $f_1 \otimes f_2 \otimes \ldots \otimes f_k$ we need to create a factor $f$ over variables $Y$, where $Y$ is the union of all the variables in $f_1 \ldots f_k$. The table entry in $f$ for any assignment $y$ of $Y$, $y \in \text{Val}(Y)$, is the multiplication of $k$ table entries from $f_1, \ldots, f_k$ corresponding to the assignment $y$.

Definition 2.5.4 Let $f$ be a factor over variables $Y$, and $Z$ some variable in $Y$. The summing out of variable $Z$ from factor $f$, is a factor $g$ over variables $W = Y - \{Z\}$ such that

$$g(W) = \sum_{Z} f(Y) = \sum_{z \in \text{Val}(Z)} f(W \land Z = z)$$

The table entry in $g$ for any assignment $w$ of $W$ is the summation of all those entries from $f$ that correspond to the assignment $w$. That is, the values of the resulting factor $g$ are obtained by adding the values of factor $f$ for each value of $Z$.  

Definition 2.5.5 Let $f$ be a factor over variables $X$ and $Z \in X$. Suppose $Z \in z^x$, $z^x \subseteq Val(Z)$. The conditioning of $f$ on $Z \in z^x$ is the factor $f_{Z \in z^x}$ over $X$ defined as follows:

$$f_{Z \in z^x}(X) = \begin{cases} f(X) & \text{if } Z \in z^x, z^x \subseteq Val(Z) \\ 0 & \text{otherwise} \end{cases}$$

This definition includes the conditioning of $f$ on $Z = z$ as special case because $Z = z$ is an abbreviation of $Z \in \{z\}$.

We have defined a factor, along with some basic operations on factors. Now, we will discuss how VE computes the function $f(Q)$ as a series of operations on factors. The VE algorithm works by keeping a set of factors. Initially, the factors are the conditioned CPDs of the Bayesian network. The non-query variables of the Bayesian network are eliminated one by one. The order in which the variables are eliminated is called the elimination ordering. To compute $f(Q)$ efficiently, factors are rearranged in equation (2.4) and the summations are pushed inside so that inside the sum are only those factors that involve the summing variable. For example, suppose we want to sum out variable $Y_i$, and $f_1, \ldots, f_k$ are the factors that are multiplied together. Suppose $f_1, \ldots, f_m$ are those factors that do not involve $Y_i$, and $f_{m+1} \ldots f_k$ are those that do involve $Y_i$. Then,

$$\sum_{Y_i} f_1 \ldots f_k = f_1 \ldots f_m \sum_{Y_i} f_{m+1} \ldots f_k$$

We explicitly construct a factor for $\sum_{Y_i} f_{m+1} \ldots f_k$. After summing out all the non-query variables, we can compute the posterior distribution by multiplying the remaining factors and normalizing the remaining factor. The VE algorithm is summarized in Figure 2.3.
Function Variable Elimination \((B, Q, E, e^f)\)

**Input:** \(B\): Bayesian network, \(Q\): query variables, \(E\): observed variables, \(e^f\): set such that \(e^f \subseteq \text{Val}(E)\)

**Output:** A normalized factor over variables \(Q\)

Let \(F\) be the factors corresponding to the CPDs in \(B\)
Replace each \(f \in F\) that involves some \(E_i \in E\) with \(f_{E_i e^f}\)
Let \(Z\) be the variables of \(B\)
for each \(Y \in Z - Q\) according to some elimination ordering do
\(\quad F \leftarrow \text{Eliminate}(Y, F)\)
end for
return \(\text{Normalize}(F, Q)\)

**Function Eliminate** \((Y, F)\)

partition \(F\) into:
\(\{f_1, \ldots, f_m\}\) that do not involve \(Y\)
\(\{f_{m+1}, \ldots, f_k\}\) are those that do involve \(Y\)
\(f \leftarrow \sum_Y f_{m+1} \times \ldots \times f_k\)
return \(\{f_1, \ldots, f_m, f\}\)

**Function Normalize** \((\{f_1, \ldots, f_r\}, X)\)
\(f \leftarrow f_1 \times \ldots \times f_r\)
\(c \leftarrow \sum_X f\)
return \(f/c\)

Figure 2.3: The Variable Elimination Algorithm.
Example 2.5.6 We illustrate the VE algorithm by computing the answer to the query $P(L = \text{true}|X = \text{true}, F = \text{false})$ in the Bayesian network shown in Figure 2.1. After conditioning on the evidence, we have the set of factors as shown in Figure 2.4. All entries that are 0 have been omitted from factors $f_4(L,X)$ and $f_6(B,L,F)$. We will eliminate the variables $X$, $F$, $P$, $S$, and $B$ in that order. Variable $X$ is involved in only one factor $f_4(L,X)$, and the value of $X$ is fixed by evidence. Eliminating $X$ replaces $f_4(L,X)$ by the factor $f_7(L)$ defined as follows:

$$f_1(S) = \begin{array}{c|c}
\text{S} & \text{true} \\
\text{false} & \text{false} \\
\end{array} \begin{array}{c} 0.2 \\
0.8 \\
\end{array}$$

$$f_3(S, P, L) = \begin{array}{c|c|c}
\text{S} & \text{P} & \text{L} \\
\text{true} & \text{true} & \text{true} \\
\text{true} & \text{true} & \text{false} \\
\text{true} & \text{false} & \text{true} \\
\text{false} & \text{false} & \text{true} \\
\end{array} \begin{array}{c} 0.05 \\
0.95 \\
0.03 \\
0.97 \\
0.02 \\
0.98 \\
0.001 \\
0.999 \\
\end{array}$$

$$f_2(P) = \begin{array}{c|c}
\text{P} & \text{true} \\
\text{false} & \text{false} \\
\end{array} \begin{array}{c} 0.9 \\
0.1 \\
\end{array}$$

$$f_4(L,X) = \begin{array}{c|c|c|c}
\text{L} & \text{X} \\
\text{true} & \text{true} \\
\text{false} & \text{false} \\
\end{array} \begin{array}{c} 0.8 \\
0.2 \\
\end{array}$$

$$f_3(B, L, F) = \begin{array}{c|c|c|c}
\text{B} & \text{L} & \text{F} \\
\text{true} & \text{true} & \text{false} \\
\text{false} & \text{true} & \text{false} \\
\end{array} \begin{array}{c} 0.25 \\
0.9 \\
0.5 \\
\end{array}$$

$$f_6(S, B) = \begin{array}{c|c|c|c}
\text{S} & \text{B} \\
\text{true} & \text{true} \\
\text{false} & \text{false} \\
\end{array} \begin{array}{c} 0.25 \\
0.75 \\
0.95 \\
\end{array}$$
Variable $F$ is involved only in one factor $f_5(B, L, X)$, and its value is fixed by evidence. Eliminating $F$ replaces $f_5(B, L, F)$ by the factor $f_6(B, L)$ defined as follows:

<table>
<thead>
<tr>
<th>$L$</th>
<th>0.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>0.8</td>
</tr>
<tr>
<td>false</td>
<td>0.2</td>
</tr>
</tbody>
</table>

To eliminate $P$ we need to multiply factors $f_2(P)$ and $f_3(S, P, L)$ because $P$ appears in these two factors. Multiplying these factors together results an intermediate factor $f_9(S, L, P)$, defined as follows:

<table>
<thead>
<tr>
<th>$S$</th>
<th>$P$</th>
<th>$L$</th>
<th>0.045</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>0.045</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>0.855</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>0.003</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>0.097</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>0.018</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>0.882</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>0.0001</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>0.0999</td>
</tr>
</tbody>
</table>
We then sum out $P$ from $f_9(S, L, P)$, and replace $f_2(P)$ and $f_3(S, P, L)$ with $f_{10}(S, L)$, defined as follows:

$$
\begin{array}{c|c|c}
S & L & f_{10}(S, L) \\
\hline
true & true & 0.048 \\
true & false & 0.952 \\
false & true & 0.0181 \\
false & false & 0.9819 \\
\end{array}
$$

Next, we eliminate $S$, which is involved in factors $f_1(S), f_6(S, B)$, and $f_{10}(S, L)$. We multiply these together to obtain an intermediate factor $f_{11}(S, B, L)$ defined as follows:

$$
\begin{array}{c|c|c|c}
S & B & L & f_{11}(S, B, L) \\
\hline
true & true & true & 0.0024 \\
true & true & false & 0.0476 \\
true & false & true & 0.0072 \\
true & false & false & 0.1428 \\
false & true & true & 0.000724 \\
false & true & false & 0.03927 \\
false & false & true & 0.013756 \\
false & false & false & 0.74624 \\
\end{array}
$$

We then sum out $S$ from $f_{11}(S, B, L)$, and replace $f_1(S), f_6(S, B)$ and $f_{10}(S, L)$ with $f_{12}(B, L)$, defined as follows:
To eliminate $B$, we multiply together $f_8(B, L)$ and $f_{12}(B, L)$, and get $f_{13}(B, L)$ defined as follows:

<table>
<thead>
<tr>
<th>B</th>
<th>L</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>0.000781</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>0.07818</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>0.01045</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>0.8445</td>
</tr>
</tbody>
</table>

We then sum out $B$ from $f_{13}(B, L)$, replacing $f_8(B, L)$ and $f_{12}(B, L)$ with a factor $f_{14}(L)$ over $L$ defined as follows:

<table>
<thead>
<tr>
<th>L</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>0.0113</td>
</tr>
<tr>
<td>false</td>
<td>0.9227</td>
</tr>
</tbody>
</table>

Now, we are left with factors $f_{14}(L)$ and $f_7(L)$, which are factors over $L$ alone. Multiplying these two factors together, we obtain a factor $f_{15}(L)$ defined as follows:

<table>
<thead>
<tr>
<th>L</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>0.0102</td>
</tr>
<tr>
<td>false</td>
<td>0.1845</td>
</tr>
</tbody>
</table>
Now, we normalize $f_{15}$ to obtain

$$P(L = true | X = true, F = false) = 0.0523$$

$$P(L = false | X = true, F = false) = 0.9476$$

In summary, we have performed the following computation:

$$f_9(S, P, L) = f_2(P) \times f_3(S, P, L)$$

$$f_{10}(S, L) = \sum_P f_9(S, P, L)$$

$$f_{11}(S, B, L) = f_1(S) \times f_6(S, B) \times f_{10}(S, L)$$

$$f_{12}(B, L) = \sum_S f_{11}(S, B, L)$$

$$f_{13}(B, L) = f_{12}(B, L) \times f_5(B, L)$$

$$f_{14}(L) = \sum_B f_{13}(B, L)$$

$$f_{15}(L) = f_{14}(L) \times f_7(L)$$

As discussed above, eliminating a variable in VE involves a series of multiplications to compute an intermediate factor, followed by a number of summations. The number of multiplications is proportional to the size of the intermediate factor times the number of factors to be multiplied. This is because for each assignment of the variables of intermediate factor we need to do $k$ multiplications, where $k$ is the
number of factors to be multiplied. Note that the number of summations is also pro-
portional to the size of the intermediate factor, as each entry (row) of intermediate
factor contributes to one summation term. Thus, the cost of eliminating a variable
in VE is proportional to the size of the intermediate factors times the number of
factors to be multiplied.

**Proposition 2.5.7** Let \( N \) be the number of variables in the network, \( b \) be the maxi-
mum number of values that a variable can have, and \( M \) be the maximum number of
variables involved in an intermediate factor produced during the computation. The
complexity of variable elimination is \( O(N \times b^M) \).

It is evident that both \( M \) and \( b \) are crucial in determining the complexity of
the VE algorithm. The number of variables involved in intermediate factors depends
upon the ordering we choose to eliminate the variables. For example, if we first
eliminate the variable \( S \) in Example 2.5.6, then we get an intermediate factor over
variables \( S, B, L \) and \( P \) that is larger than any factor we actually had. Unfortunately,
finding the optimal elimination ordering, i.e., the ordering that results in the smallest
factors, is NP-complete [Arnborg, Corneil and Proskurowski, 1987]. However, one
can use some simple heuristics to produce good elimination ordering. One such
simple heuristic is the *minimum discrepancy* heuristic. According to this heuristic,
at each time we choose to eliminate the variable that adds the fewest number of new
edges to \( G \) [Kjaerulff, 1990].

Note that performing inference in networks that have variables with very
large \( b \), even if we can reduce \( M \), could be very expensive. If exact inference is too
expensive in a network, we can use approximate inference algorithms, which we
have not discussed here. Usually there is a good deal of structure in the networks that can be exploited for performing efficient inference.

2.6 Pruning Irrelevant Variables

Variable Elimination is a query oriented algorithm that processes one query at a time. We can use the structure of the graph to simplify the task of computing the query as much as possible. We can simplify the task by pruning from the Bayesian network all variables that are irrelevant to the query. One of the properties of the Bayesian network that can be utilized by the VE algorithm is known as barren variables. Shachter [1986] introduced the concept of barren variables.

Definition 2.6.1 Let Q be the query variables in a Bayesian network and E be the observed variables. A variable $X$ is a barren variable with respect to query nodes $Q$, if $X \not\in Q$, $X \not\in E$, and all descendants of $X$ are barren.

Barren variables are a consequence of one of the basic axioms of probability theory:

Axiom 2.6.2

$$\sum_X P(X|Y) = 1$$

When a barren variable is summed out, it always gives a unity factor equal to one. Hence, barren variables are computationally irrelevant and can be removed from a Bayesian network without affecting the posterior probability distribution of the query variables.
2.7 Context Specific Independence

Bayesian networks exploit conditional independence to represent joint probability distribution in a compact manner. They do not place any restriction on how a variable depends on its parents. This means that to specify the conditional probability table of a variable $X$, we need to specify $|X| \times |Pa(X)|$ conditional probabilities. Often, however, there is much structure in the conditional probability tables that can be exploited for efficient inference. In particular, Bayesian networks cannot capture certain independence: independence that holds only in certain contexts, for example, given a specific assignment of values to certain variables.

In the past, extensive work has been done to extend Bayesian network representation in order to capture this additional independence. Many different approaches have been proposed for representing and reasoning with independence that are more general than conditional independence [Boutilier et al., 1996; Zhang and Poole, 1996; Poole and Zhang, 2003; Heckerman and Breese, 1994]. In this section, we discuss context specific independence [Boutilier et al., 1996], which we will apply in subsequent sections.

Boutilier et al. [1996] formalized the notion of context specific independence (CSI). CSI refers to the fact that some random variables are independent of each other only in a certain context.

**Definition 2.7.1** Given a set of variables $C$, a **context** on $C$ is an assignment of one value to each variable in $C$, denoted by $C = c$, where $c \in Val(C)$. We say that $C$ are the variables of the context.

**Definition 2.7.2** Let $X$, $Y$, $Z$, and $C$ be pairwise disjoint sets of variables. We say
that $X$ and $Y$ are contextually independent [Boutilier et al., 1996], given $Z$ and the context $C = c$, $c \in \text{Val}(C)$, if

$$P(X|Z, C = c, Y) = P(X|Z, C = c) \text{ whenever } P(Y, Z, C = c) > 0$$

Thus, the independence relation between $X$ and $Y$ need not hold for all values of $C$.

For example, consider the Bayesian network shown in Figure 2.5 (a). The network represents the domain that if a person goes climbing ($C$) and the rocks are slippery ($S$), then that person will probably get injured ($I$). The domain is captured here through three Boolean random variables $S$, $I$, and $C$, with domain $\{\text{true, false}\}$. The tabular representation of $P(I|C \land S)$ is shown in Figure 2.5 (b). Note that the last two rows in the tabular representation of $P(I|C \land S)$ are the same. This represents that the probability of getting injured does not depend on whether the rocks are slippery or not, if a person does not go climbing. That is, variable $I$ is independent of $S$ in the context $C = \text{false}$.

If there is context specific independence, the regularities in the conditional probability tables can be captured for representing the conditional probabilities in a compact manner. A very natural representation for capturing the common entries in the CPTs is by decision tree [Boutilier et al., 1996].

**Definition 2.7.3** A decision tree representing the conditional probability distribution of a variable $X$ is a rooted tree $T_x$, where each internal node represents $X_i \in (\{X\} \cup \text{Pa}(X))$ and each leaf node represents a real number between $[0,1]$. Each internal node has a set of outgoing arcs to its children, and each arc in $T_x$ is associated with a unique variable assignment $X_i = x_i, x_i \in \text{Val}(X_i)$. 31
Figure 2.5: (a) A simple Bayesian network modeling whether the person is injured. (b) The tabular representation of conditional probability $P(I|C \wedge S)$: (c) The decision tree representation of conditional probability $P(I|C \wedge S)$. 
A branch in a decision tree represents a **path** beginning at the root and ending at a leaf node. The parent context defined by a path is the set of variable assignments $X_i = x_i, X_i \neq X$ encountered on the arcs along the path.

The decision tree representation of $P(I \mid C \land S)$ is shown in Figure 2.5 (c). Boutilier et al. [1996] present two inference algorithms that exploit the decision tree representation for the efficient inference in Bayesian networks that we have not discussed (see [Boutilier et al., 1996] for details). Poole and Zhang [2003] exploit context specific independence in the variable elimination algorithm.

### 2.8 Conclusion

Bayesian networks have been used with great success for modeling many real world applications. For example, Bayesian networks are used for troubleshooting in Microsoft Windows [Heckerman, Breese and Rommelse, 1994], for predicting the likelihood of meeting attendance [Horvitz, Koch, Kadie and Jacobs, 2002], and in molecular biology for analyzing expression data [Friedman, Linial, Nachman and Pe'er, 2000]. They are potentially applicable anywhere there is uncertainty about the state of the world from observations. The key to the success of Bayesian networks is the representation of conditional independence, which enables the compact representation of the joint probability distribution and efficient inference algorithms. It is our goal in subsequent chapters to extend Bayesian networks to more complex domains, where discrete random variables can have large (or unbounded) number of values.
Chapter 3

Exploiting Structure in Large Discrete Domains

3.1 Introduction

As described in the previous chapter, Bayesian networks have emerged as a viable technology for representing and reasoning about probabilistic models. As we have seen, the complexity of probabilistic inference grows with the domain size of the variables. We now want to take the next step, to extend Bayesian networks to more complex problems, where discrete random variables can have a large (or possibly infinite) number of values. For example, in the person identification problem, where given a pair of records we are trying to identify whether two records refer to the same person, we may have variables that have as domains the set of all names, the set of all postal codes, and the set of all credit card numbers.

The large number of values of the variables are a problem for the following
reasons:

- Representing conditional probabilities in conditional probability tables (CPTs) requires a complete specification of CPT entries. The number of parameters required to describe the CPT of a variable \(X\) is \(|X| \times |Pa(X)|\). For the CPTs with large \(|X|\) or \(|Pa(X)|\), specifying all the parameters is not feasible.

- Even if one could specify all the parameters for the big CPTs, a big tabular representation is not efficient in terms of memory.

- The complexity of exact probabilistic inference in a Bayesian network is exponential in tree width, where the base of the exponent is domain size.

- One could potentially have a big tabular representation and use the previous approaches for probabilistic inference. However, we cannot represent the conditional probability distribution (CPD) of a variable conditioned on a variable that has an infinite number of values in a tabular form. For example, consider a random variable that has as its domain the set of all first names. This domain may never be known to the full extent because people can make up names. We need a compact representation for representing the large (or unbounded) CPTs so that we can reason in an efficient manner.

Carrying out inference in Bayesian networks that have variables with large domain is a difficult problem. For efficient inference in Bayesian networks that have large-domain discrete random variables, we consider cases where there is some structure that can be exploited to make inference efficient. In many cases, there is some structure in the conditional probability tables that allows us to partition
the large number of values of the variables into equivalence classes. Rather than reasoning about every value of these variables individually, we can reason about a set of values together. The fundamental idea is that in any conditional probability table, we partition the values of each large-domain variables into disjoint subsets (equivalence classes) that have the same conditional probability for particular values of other variables. We would like to represent the CPDs in a compact manner and an inference algorithm that can utilize this compact representation in probabilistic inference for computational gain.

We assume that we have a procedural way of generating the prior probabilities of the large domain variables (perhaps conditioned on other variables). This may include looking up tables. For example, the U.S. Census Bureau\(^1\) publishes a list of all first names, conditioned by gender, together with probabilities that covers 90% of the probability mass for both male and female first names. Together with a method for estimating the probability of a new name, this can be used as the basis for \(P(FirstName|Sex)\). We may also have a model of how postal codes are generated to give a procedure that estimates the probability of a given postal code.

To represent the conditional probability of a variable conditioned on variables that have large domains we assume that we can write them compactly using predicates and functions. We also assume that we have the procedures for computing the predicates (and functions) and to count the number of values for which they are true. For example, consider a binary variable \(CommonName\) that has only one parent \(FirstName\). \(CommonName\) is true when the value taken by \(FirstName\) is a common male or female name. We may consider the top 20 names in the male and

\(^1\)http://www.census.gov/genealogy/names/
female name file as the common male and female names. Now, we can write the
the conditional probability $P(\text{CommonName}|\text{FirstName})$ compactly as follows:

$$P(\text{CommonName}|\text{FirstName}) = \begin{cases} 
1 & \text{if } \text{common}(\text{FirstName}) \\
0 & \text{otherwise}
\end{cases}$$

where, $\text{common}(\text{FirstName})$ is a predicate that is true if $\text{FirstName}$ is a common
male or female name. Predicate $\text{common}(\text{FirstName})$ is true for the 40 values of
$\text{FirstName}$.

In this chapter, we propose an approach for making efficient inference in
Bayesian networks that have discrete random variables with large domains. We be­
gin in Section 3.2 by presenting a motivating problem – the person identification
problem. In Section 3.3, we discuss an approach for representing the CPDs in a
compact manner. We represent the CPDs using both intensional (i.e., in terms of
procedures) and extensional (by listing the values) definitions. To deal with the
complexity of inference, we consider an evidence and query oriented approach.
Our approach is based on the fact that when there is no evidence to distinguish
between all the values of a variable, we do not need to reason about each value sep­
arately. Rather, we can reason about a group of values together as a single entity.
For effective inference, we utilize a Bayesian network representation of conditional
independence, while also taking advantage of the additional structure in the CPDs.
In Section 3.4 we present an inference algorithm, Large Domain VE, that is based
on the variable elimination algorithm for making inference in Bayesian networks
that have variables with large domains. In Section 3.5, we present the application
of Large Domain VE to the person identification problem that shows that by parti-
tioning the large number of values of a variable we can make the inference efficient.

### 3.2 A Motivating Example: The Person Identification Problem

The person identification problem that we consider is to determine whether two records containing information about the demographic attributes of a person (e.g., name, age, phone number, etc.) refer to the same person. It is used for comparing records in one or more data files, removing duplications, and determining if a new record refers to a person already in the database, or to a new person. The person identification problem occurs in many person-centric applications. For example, it occurs in health care applications [Gill, 1997; Bell and Sethi, 2001] when identifying patients, in social services applications in relation to clients, in commerce applications with customers, even in criminal contexts when identifying suspects [Wang, Chen and Atabakhsh, 2004].

This problem has been studied independently under various names by different user communities. In statistics, it has been studied as record linkage since at least 1969 [Fellegi and Sunter, 1969]. In the Fellegi-Sunter method, for each pair of records, agreement and disagreement probabilities (matching weights) for each attribute are computed using frequency counts and error rates. The values of these match weights are used to designate a pair of records as a match, a possible match or a nomatch. Possible matches are those pairs for which clerical review is needed to decide their final status.
In the computer science literature, this problem has been studied as duplicate detection [Monge and Elkan, 1997], the merge/purge problem [Hernandez and Stolfo, 1995], or identity uncertainty [Pasula, Marthi, Milch, Russell and Shpitser, 2002]. Hernandez and Stolfo developed the sorted neighbourhood method for limiting the number of potential duplicate pairs requiring distance computation. Monge and Elkan proposed the use of the Smith-Waterman algorithm for computing minimum edit-distance to recognize pairs of approximately duplicate records. Pasula et al. [2002] proposed the use of a relational probability model with MCMC for matching scientific citations.

The standard methods for person identification [Fellegi and Sunter, 1969] consider that the attributes are independent: the matching of one attribute does not depend on other attributes. However, this assumption is often faulty as the following examples demonstrate:

- People living in the same household have the same address, phone number and often the same last name. In this situation, if we assume that the last name, address and phone number are independent of each other, it becomes more likely that we have a false positive match.

- Twins usually have the same date of birth and last name. In this situation, the independence assumption can again cause a false positive match.

- If a person moves to a different city, his address, phone number, and postal code change together. In this situation, the independence assumption can cause a false negative match.
If two descriptions refer to the same person, we expect that the attribute values should be the same for both descriptions. However, the attributes may be different because of errors, for example: typographical errors, misspellings, swapping first and last names, moving and so on. Rather than considering the attribute as independent of each other, we consider here that attributes depend on each other. We propose the use of Bayesian networks for modelling the dependence/independence among the attributes.

3.2.1 Decision Making in Person Identification

The core sub-problem of person identification is to determine whether two records refer to the same person or not. Let the set of field values (attribute values) for a record be called a description\(^2\). Let \(X\) and \(Y\) be two records, which refer to the people to be compared, with \(Desc_X\) and \(Desc_Y\) denoting their corresponding descriptions. There are two hypotheses when we compare the descriptions \(Desc_X\) and \(Desc_Y\):

- the records refer to the same person \((X = Y)\)
- the records refer to different people \((X \neq Y)\)

Let \(P_{same}\) be the posterior probability that records \(X\) and \(Y\) refer to the same person, given their descriptions and \(P_{diff}\) be the posterior probability that records \(X\) and \(Y\) refer to different people, given their descriptions. That is,

\[
P_{same} = P(X = Y|Desc_X, Desc_Y)
\]

\(^2\)We want to distinguish a record from its description because two records can have the same description.
\[ P_{\text{diff}} = P(X \neq Y | \text{Desc}_X, \text{Desc}_Y) \]

The odds, \( \text{Odds} \), for hypotheses \( X = Y \) and \( X \neq Y \):

\[
\text{Odds} = \frac{P_{\text{same}}}{P_{\text{diff}}}
= \frac{P(X = Y) \times P(\text{Desc}_X \land \text{Desc}_Y | X = Y)}{P(X \neq Y) \times P(\text{Desc}_X \land \text{Desc}_Y | X \neq Y)}
= \frac{P(X = Y) \times P(\text{Desc}_Y | X = Y) \times P(\text{Desc}_X | \text{Desc}_Y \land X = Y)}{P(X \neq Y) \times P(\text{Desc}_Y | X \neq Y) \times P(\text{Desc}_X | \text{Desc}_Y \land X \neq Y)} \tag{3.1}
\]

We would expect that record \( Y \)'s attribute value is independent of \( X = Y \) or \( X \neq Y \) given no information about the other person's attribute value. That is,

\[
P(\text{Desc}_Y | X \neq Y) = P(\text{Desc}_Y | X = Y) \tag{3.2}
\]

After applying equation (3.2) in equation (3.1), we have the following:

\[
\text{Odds} = \frac{P(X = Y)}{P(X \neq Y)} \times \frac{P(\text{Desc}_X | \text{Desc}_Y \land X = Y)}{P(\text{Desc}_X | \text{Desc}_Y \land X \neq Y)} \tag{3.3}
\]

The ratio \( \frac{P(X = Y)}{P(X \neq Y)} \) is a prior odds and the ratio \( \frac{P(\text{Desc}_Y | \text{Desc}_Y \land X = Y)}{P(\text{Desc}_Y | \text{Desc}_Y \land X \neq Y)} \) is a likelihood ratio.

There are three possible actions (decisions) that we can take for records \( X \) and \( Y \) [Fellegi and Sunter, 1969]:

- **match** (decide that \( X \) and \( Y \) refer to the same person)

- **possible match** (hold for a clerical review)

- **nomatch** (decide that \( X \) and \( Y \) refer to different people)

The decision can be made using decision theory [Duda, Hart and Stork, 2000], given the cost of false positive, true positive, true negative and false negative match. An action with minimum cost is selected. Suppose we have a cost
function $E(\alpha|\omega)$ that describes the cost of action $\alpha$ when $\omega$ is true in the world. If the assignment is *match*, the expected cost $E_{\text{match}}$ is:

$$E_{\text{match}} = E(\text{match}|\text{same}) \times P_{\text{same}} + E(\text{match}|\text{diff}) \times P_{\text{diff}}$$

If the assignment is *possible match*, the expected cost $E_{\text{posmatch}}$ is:

$$E_{\text{posmatch}} = E(\text{posmatch}|\text{diff}) \times P_{\text{diff}} + E(\text{posmatch}|\text{same}) \times P_{\text{same}}$$

If the assignment is *nomatch* the expected cost $E_{\text{nomatch}}$ is:

$$E_{\text{nomatch}} = E(\text{nomatch}|\text{diff}) \times P_{\text{diff}} + E(\text{nomatch}|\text{same}) \times P_{\text{same}}$$

We select the assignment (action) for which the error cost is minimum. Let us assume the following:

- $E(\text{match}|\text{same}) < E(\text{posmatch}|\text{same}) < E(\text{nomatch}|\text{same})$, and
- $E(\text{nomatch}|\text{diff}) < E(\text{posmatch}|\text{diff}) < E(\text{match}|\text{diff})$

The conditions for actions *match*, *possible match*, and *nomatch* are the following:

- Action *match* is optimal if $\frac{P_{\text{same}}}{P_{\text{diff}}} \geq \max(C_1, C_2)$
- Action *possible match* is optimal if $\min(C_2, C_3) \leq \frac{P_{\text{same}}}{P_{\text{diff}}} \leq \max(C_1, C_2)$
- Action *nomatch* if $\frac{P_{\text{same}}}{P_{\text{diff}}} \leq \min(C_2, C_3)$

where,

$$C_1 = \frac{E(\text{match}|\text{diff}) - E(\text{posmatch}|\text{diff})}{E(\text{posmatch}|\text{same}) - E(\text{match}|\text{same})}$$

$$C_2 = \frac{E(\text{match}|\text{diff}) - E(\text{nomatch}|\text{diff})}{E(\text{nomatch}|\text{same}) - E(\text{match}|\text{same})}$$

$$C_3 = \frac{E(\text{posmatch}|\text{diff}) - E(\text{nomatch}|\text{diff})}{E(\text{nomatch}|\text{same}) - E(\text{posmatch}|\text{same})}$$
The constant prior odds can be merged with constants $C_1$, $C_2$, and $C_3$. We then just need the likelihood ratio for making the decision.

Traditional methods [Fellegi and Sunter, 1969] treat the attributes as independent of each other, whether the descriptions refer to the same person or not. In the Fellegi-Sunter method, for each pair of records, agreement and disagreement probabilities for each attribute are computed using frequency counts and error rates.

We have relaxed this assumption, and model the dependence/independence between the attributes for both cases $X = Y$ and $X \neq Y$ using a similarity network representation [Heckerman, 1990]. This representation exploits the hypothesis-specific independence between variables. In particular, separate Bayesian networks are constructed for each hypothesis.

### 3.2.2 Modelling of Attribute Dependence/Independence

We model the dependence/independence between the attributes. The Bayesian networks of attribute dependence contain many variables that have very large domains.

**The Bayesian Network Model of Attribute Dependence for $X \neq Y$**

To identify a person, we consider the following seven attributes: social security number (SSN), first name (Fname), last name (Lname), date of birth (DOB), gender (Gen), phone number (Phone), and postal code (Post). The statistical dependence among the attributes that we assume is shown in Figure 3.1. Dependence among the attributes is modelled using the hidden variables that are the unshaded nodes in

---

3The approach can be extended to any number of attributes.

4The dependence may be known by the domain experts, or potentially can be learned.
The proposition *twin* is true if the people referred by $X$ and $Y$ are twins.

The proposition *relative* is true if the people referred by $X$ and $Y$ are relatives.

The proposition *samehousehold* is true if the people referred by $X$ and $Y$ live in the same household.

The proposition *samelastname* is true if the people referred by $X$ and $Y$ have the same last name.

We assume here that the gender of two different people is independent of each other\(^5\). Attribute $SSN$ does not depend on the considered relations. However, we cannot assume that the $SSN$ of two different people is independent. Knowing

\[^5\text{A more detailed model may specify that twins are more likely to be of the same gender.}\]
one person's SSN changes our belief about another person's SSN, because we ex­pect that they should not be the same (as two persons are not allowed to have the same SSN). Hence,

\[
P(\text{SSN}_X|\text{SSN}_Y \land X \neq Y) = \begin{cases} 
  r & \text{if } \text{SSN}_X = \text{SSN}_Y \\
  P(\text{SSN}_X) & \text{if } \text{SSN}_X \neq \text{SSN}_Y 
\end{cases}
\]

where \( r \) denotes the probability that two different persons have the same SSN recorded (due to some errors), which is very, very small.

**The Bayesian Network Model of Attribute Dependence for \( X = Y \)**

If records \( X \) and \( Y \) refer to the same person (the numerator of the Odds formula), we expect that the attribute values should be the same for both \( X \) and \( Y \). However, there may be differences because of attribute errors: typing errors, phonetic errors, nicknames, swapping first and last names, change of address and so forth.

As explained earlier, we do not want to assume that the attributes are inde­pendent. We model their dependence using the hidden variables that explain the dependence among the attributes. Suppose the attributes are dependent because the data-entry person was sloppy, or because the person moved to a new residence be­tween the times that the records were input. If the move is true, we consider that the record \( X \) was input before the move and that \( Y \) was input after the move.

We model the dependence among attributes using their true values\(^6\), the slop­piness of the data-entry person, and the possibility of moving. Here, we consider

\(^6\)For some attributes, the true values change because of the move, e.g., phone number. For such attributes, we consider the true values with respect to \( Y \).
the change in phone number and postal code because of the move. The dependence between attributes is shown in Figure 3.2. The unshaded nodes show the hidden variables, and we assume that the shaded nodes are observed. The variable Sloppy $X$ represents whether the person who recorded the attribute values of record $X$ was sloppy or not. The variable $EF.X$ represents which error was made in recording the first name for record $X$. To make this example more readable, we consider the following errors (values of $EF.X$):

- **copy error (ce):** an error where a person copies a correct name, but from the wrong row of a table.

- **single digit/letter error (sle):** an error where a person copies all the letters (digits) correctly except one letter (digit).

- **no error (noerr):** no errors.

The variable move represents whether the person moved to a different address between the two records. The variable Aphone.$Y$ represents the true phone number after the move, if move is true. The variable Afname represents the actual
first name. The random variables $Fname_X$, $Fname_Y$, and $Afname$ have, as domains, all first names. The random variables $Phone_X$, $Phone_Y$, and $Aphone_Y$ have, as domains, all phone numbers.

For the probability $P(Afname|Sex)$, we can use the first name lists available from the U.S. Census Bureau\(^7\). There are two first name lists with associated probabilities: one for female names, and the other for male names. The probability $P(Afname|Sex = male)$ is computed using the male name file. The probability $P(Afname|Sex = female)$ is computed using the female name file. We need a different mechanism for names that do not appear in these lists, as we cannot assign zero probabilities to such names. A number of techniques have been proposed to estimate the probabilities of new events [Chen and Goodman, 1998; Good, 1953; Friedman and Singer, 1998] that can be used to compute the probability of a new name (names that did not appear in the database). The data available from the U.S. Census Bureau is too noisy and incomplete to apply any of these techniques. In our implementation, we use a very small probability as the estimate of the probability of a new name.

To compute the probability $P(Aphone.Y)$ a model for generating phone numbers can be used. There are rules to generate the valid phone numbers for a city, province, and so forth. We use a simple procedure to compute the probability $P(Aphone.Y)$. Let $P$ be the total number of legal phone numbers, then

\[
P(Aphone.Y) = \begin{cases} 
1/P & \text{if } Aphone.Y \text{ is a legal phone number} \\
0 & \text{otherwise}
\end{cases}
\]

To make the decision whether records $X$ and $Y$ refer to the same person, we

\(^7\)http://www.census.gov/genealogy/names/
need to compute the likelihood ratio (LR) as given in equation 3.3. Suppose \( B_1 \) and \( B_2 \) denote the Bayesian networks shown in Figures 3.1 and 3.2, respectively. Let \( M \) and \( N \) be the variables of \( B_1 \) and \( B_2 \), respectively. Since \( B_1 \) and \( B_2 \) refer to a similarity network, \( B_1 \) encodes the conditional distribution \( P(M \mid X \neq Y) \), and \( B_2 \) encodes \( P(N \mid X = Y) \) [Geiger and Heckerman, 1996].

To compute the likelihood ratio we need to condition on the observations and marginalize over the unobserved variables in the networks shown in Figures 3.1 and 3.2. After conditioning on the observed variables and summing out the hidden variables from the network shown in Figure 3.1, we get the likelihood of the observed data, given the hypothesis \( X \neq Y \) [Geiger and Heckerman, 1996]. Note that this is different from the standard set-up of probabilistic inference where we want to query some variable(s); here, we want to compute the probability of the data given the hypothesis that is not explicitly represented in the network.

For the Bayesian network shown in Figure 3.2, we cannot represent the conditional probabilities in tabular form because some of the variables have very large domains. The conditional probability \( P(Fname \mid X \wedge Sex \wedge EF \cdot X) \) cannot be represented in tabular form, as we do not know all names, and even if we could, the domains of \( Fname \) and \( Fname \cdot X \) would be very large. The tabular representation of \( P(Fname \mid Sex) \) is also very large. To represent these large CPTs we need a compact representation. The previously existing Bayesian network inference algorithms do not work on this network. In the following sections, we develop a methodology for representing large CPTs and detail the probabilistic inference algorithm that exploits the compact representation of the large CPTs.
3.3 Structured Representation of Conditional Probabilities

To compactly represent the conditional probabilities for Bayesian networks that have large domain variables, the approach we propose is based on a structured representation of the problem. The structured representation allows exploitation of regularities and independencies in the problem that can reduce the effective state space of the variables. Our idea for specifying the conditional probabilities in a compact manner is that: the conditional probabilities can be represented compactly using both intensional (in terms of functions and predicates) and extensional (by listing the values) definitions. The computation of intensional definitions may involve looking up the tables, and computing the predicates and functions.

Example 3.3.1 Consider the conditional probability \( P(Fname \mid Afname \land Sex \land EF.X) \) from the Bayesian network shown in Figure 3.2. Suppose we observed \( Fname.X = \text{"dave"} \). If the data-entry person makes a single letter error (\( EF.X = sle \)), the conditional probability \( P(Fname.X = \text{"dave"} | Afname \land Sex \land EF.X = sle) \) has the same value for all those values of \( Afname \) that are a single letter apart from “dave”. Note that there are 100 words that are single letter apart from “dave”, as each letter in “dave” can be replaced by 25 other possible letters. Let \( S \) be the set of names that are single letter apart from “dave”:

\[
S = \{ x \mid x \text{ is a single letter apart from "dave"} \}
\]
Then,

\[
P(Fname_X = \text{"dave"} \mid Afname \land Sex \land EF_X = \text{sle}) = \begin{cases} 
1/100 & \text{if } Afname \in S \\
0 & \text{otherwise}
\end{cases}
\]

This above assumes that given \(EF_X = \text{sle}\), \(Fname_X\) is independent of \(Sex\). Thus, given \(EF_X = \text{sle}\), we can represent

\[
P(Fname_X = \text{"dave"} \mid Afname \land Sex \land EF_X = \text{sle})
\]

compactly using the intensional definition:

\[
Afname \in S, \ S = \{x \mid x \text{ is a single letter apart from "dave"}\}
\]

In a Bayesian network that has variables with large domains, we can represent the conditional probabilities in a compact manner using intensional (using the predicates and functions) and extensional representations (by listing the values). To capture such distributions, we introduce a \(CPD\) \textit{language} that allows us to define the CPDs procedurally, in terms of predicates and functions. Let \(Y\) be a variable and \(Z_1, \ldots, Z_k\) be the parents of \(Y\). In CPD language, to describe the conditional probability of \(Y\), we have two cases:

\textbf{Case 1 (Y does not have large domain):} We represent \(P(Y|Z_1, \ldots, Z_k)\) as a sentence as follows:

\[
P(Y|Z_1, \ldots, Z_k) = \langle\text{Cp1}\rangle
\]

where

\[
\langle\text{Cp1}\rangle ::= \langle\text{number}\rangle, |
\]

\[
\langle\text{if C(Z_1, \ldots, Z_n, Y) then C1 else C1}\rangle
\]

50
Here, \textit{number} is a real constant in the closed interval \([0, 1]\); it represents the probability of a single value of \(Y\). \(C(Z_1, \ldots, Z_n, Y)\) is a condition that describes the subsets of the variables. The condition \(C(Z_1, \ldots, Z_n, Y)\) can be described either:

- \textit{intensionally}, as a rule (predicate) which implicitly describes the subsets for the variables. We assume that there is a procedure to efficiently compute each predicate. For example, in Example 3.3.1 the condition \(\text{Afname} \in \{x \mid x \text{ is single letter apart from "dave"}\}\) is an intensional definition.

- \textit{extensionally}, by listing the values of the variables. For example, \(X_i \in \{v_1, \ldots, v_k\}\) or \(X_i = v_{12}\). Note that this is a particular form of intensional definition that needs to be distinguished because it allows more efficient inference.

**case2 (\(Y\) has large domain):** We represent \(P(Y|Z_1, \ldots, Z_k)\) as a sentence as follows:

\[
P(Y|Z_1, \ldots, Z_k) = \langle Cp \rangle
\]

where

\[
\langle Cp \rangle ::= \langle \text{number} \rangle | \\
\langle [p_Y, pm] \rangle | \\
\langle \text{if } C(Z_1, \ldots, Z_n, Y) \text{ then } Cp \text{ else } Cp \rangle
\]

Here, \textit{number} and \(C(Z_1, \ldots, Z_n, Y)\) have the same meaning as discussed above in Case1. In pair \([p_Y, pm]\), \(p_Y\) represents the probability of a single value of
and \( pm \), a real constant between \([0, 1]\), represents the probability mass of
a set of values of \( Y \). The value \( pr \) could be defined extensionally (by listing
the value) or intensionally using a function, e.g., \( f(Y) \), that returns a real con-
tant between \([0, 1]\) – we assume that we have procedures for computing the
functions.

The conditional probability of \( Y \) can be defined using both intensional and
extensional definitions. The CPD language generalizes the notion of context spe-
cific independence [Boutilier et al., 1996], because the contexts are not only given
by the assignment of the values to the variables such as \( X_i \in \{ v_1, \ldots, v_k \} \), but also
by expressions such as \( foo(X_1, \ldots, X_k) = v_i \).

**Example 3.3.2** Consider the representation of conditional probability distribution
\( P(Fname_X|Afname \land Sex \land EF_X) \) of the Bayesian network shown in Figure 3.2.
As mentioned in Section 3.2.2, variable \( Fname.X \) represents the first name for
record \( X \), \( Afname \) represents the actual first name of the person and \( EF.X \) repre-
sents the type of error made in first name for record \( X \). The conditional probability
\( P(Fname.X|Afname \land Sex \land EF_X) \) can be represented compactly using the CPD lan-
guage, as shown in Figure 3.3. It can also be seen as a decision tree [Quinlan, 1986]
as shown in Figure 3.4, where the internal nodes represent the conditions and the
leaf nodes represent the real numbers or the pairs of functions and numbers, e.g.,
\[ [\, prsing(Fname.X), \, 1\, ] \.\]

As shown in Figures 3.3 and 3.4 \( P(Fname.X|Afname \land Sex \land EF_X) \) is repre-
sented compactly using the intensional and extensional definitions, by conditioning
on the values of \( EF.X \). Figure 3.4 shows that there are three main cases. The
\[
P(Fname_X|Afname \land Sex \land EF.X) = \langle \begin{array}{l}
\text{if } (EF.X = \text{noerr}) \text{ then } C_1 \text{ else } C_2 \end{array}\rangle
\]

Figure 3.3: A CPD language representation of \( P(Fname_X|Afname \land Sex \land EF.X) \).

Conditional probabilities for these cases are computed as follows:

**Case 1:** \( EF.X = \text{noerr} \)

If the data-entry person who entered the attribute values for record \( X \) did not make an error in the first name, the first name that appears in record \( X \) must be the actual first name. The predicate \( equal(N1, N2) \) as shown in Figure 3.3 is true when \( N1 \) and \( N2 \) have the same values. We assume that given \( EF.X = \text{noerr} \), \( Fname.X \) is independent of \( Sex \). Thus,

\[
P(Fname.X|Afname \land Sex \land EF.X = \text{noerr}) = \begin{cases} 
1 & \text{if } equal(Afname, Fname.X) \\
0 & \text{otherwise}
\end{cases}
\]

If \( Fname.X \) (\( Afname \)) is observed, predicate \( equal \) implicitly partitions the values of \( Afname \) (\( Fname.X \)) into the observed value of \( Fname.X \) (\( Afname \)) and the other values.
Figure 3.4: A decision tree representation of $P(Fname_X|Fname \land Sex \land EF_X)$.

**Case 2: $EF_X = sle$**

If the data-entry person who entered the values for record $X$ mistyped a single letter in the first name ($EF_X = sle$), the first name that appears in record $X$ must be one letter different from the actual name, but has the same number of letters. The predicate $\text{singlet}(N_1, N_2)$ as shown in Figure 3.3 is true when $|N_1| = |N_2|$ and $N_1$ is a single letter different than $N_2$. The function $\text{prsing}(Fname_X)$ computes the probability when $EF_X = sle$ and $\text{singlet}(Fname_X, Fname) = true$. Function $\text{prsing}(Fname_X)$ is computed as follows:

$$\text{prsing}(Fname_X) = \frac{1}{25 \times |Fname_X|}$$

If $Fname_X = \text{"dave"}$, $\text{prsing}(Fname_X) = \frac{1}{100}$

We again assume that given $EF_X = sle$, $Fname_X$ is independent of $Sex$, as this typing error does not depend on the sex. Given $EF_X = sle$, the probability mass of all the values of $Fname_X$ that have the same number of letters as in $Fname_X$
and are a single letter different from Afname is 1. Thus,

$$P(Fname.X|Afname \land Sex \land EF.X = sle) =$$

$$\begin{cases} 
[\text{prsing}(Fname.X), 1] & \text{if } \text{singlet}(Fname.X, Afname) \\
0 & \text{otherwise}
\end{cases}$$

If Fname.X is observed, predicate singlet implicitly partitions the values of Afname into two sets: one consists of those values of Afname that have the same number of letters as in the observed value of Fname.X and are a single letter different from the value of Fname.X; and the second consists of the rest of the values of Afname.

**Case 3:** EF.X = ce

If the data-entry person makes a copying error, we can consider that the value of Fname.X is distributed according to the distribution over names. That is, Fname.X is independent of Afname given EF.X = ce. We can use male and female name files for the distribution over names.

The predicate intable (Fname.X, male) is true when the value of Fname.X exists in the male name file. The function lookup(Fname.X, male) computes the probability when EF.X = ce and intable (Fname.X, male) = true. The function lookup(Fname.X, male) computes the probability P(Fname.X) by looking in the male name file. If Fname.X is not in the male name file, we use function fnew(Fname.X), which returns the probability of a new name. The probability mass

---

When a person makes a copying error there are more chances for a common name to be typed than an uncommon name, because a common name appears more frequently in the database than an uncommon name.
of all the male names that are in the male name file available from U.S. Census Bureau is 0.9. Also, the probability mass of all the female names that are in the female name file available from U.S. Census Bureau is 0.9. Thus,

\[ \text{Sex} = \text{male} \]

\[ P(Fname_X| \text{Afname} \land \text{Sex} = \text{male} \land EF.X = ce) = \]

\[ \begin{cases} 
\text{lookup}(Fname_X, \text{female}), 0.9 & \text{if intable}(Fname_X, \text{male}) \\
\text{fnew}(Fname_X), 0.1 & \text{otherwise}
\end{cases} \]

Sex = female

\[ P(Fname_X| \text{Afname} \land \text{Sex} = \text{female} \land EF.X = ce) = \]

\[ \begin{cases} 
\text{lookup}(Fname_X, \text{female}), 0.9 & \text{if intable}(Fname_X, \text{female}) \\
\text{fnew}(Fname_X), 0.1 & \text{otherwise}
\end{cases} \]

### 3.3.1 Input Requirements for Inference

Let \( \mathcal{B} \) be a Bayesian network whose CPDs are represented compactly using intensional and extensional definitions in the CPD language form discussed in Section 3.3. To perform inference in \( \mathcal{B} \), we assume that the "Input Condition" defined below is true (the reason for this condition is explained later in Section 3.4.4):

**Input Condition:** Let \( Y \) be a variable with large domain. Suppose \( T_Y \) denotes the tree representation of \( P(Y|Pa(Y)) \). If \( T_Y \) contains condition \( C(X, Y) \) such that \( X \neq \{\} \land X \subseteq Pa(Y) \), \( Y \) must be observed.
To perform inference in $B$, user needs to provide some procedures (the reason for these procedures is also explained later in Section 3.4.4). Let $Y$ be a minimal non-empty, non-query, non-observed, set of large-domain variables of $B$ that do not share a predicate with any non-observed variable outside of $Y$. Suppose $X_Y$ is the set of children of elements of $Y$, i.e.,

$$X_Y = \{ X \mid \exists Y \in Y, Y \in Pa(X) \}$$

Let $Y_1, \ldots, Y_k$ be the variables in $Y$ and $X_1, \ldots, X_m$ be the variables in $X_Y$. We define the following:

- $involve(T, \alpha, Z)$: a function that returns a set of conditions of path $\alpha$ of $T$ that contain variable $Z \in Z$.
- $notinvolve(T, \alpha, Z)$: a function that returns a set of conditions of path $\alpha$ of $T$ that do not contain variable $Z \in Z$.

$SY$: a set of contexts computed as follows:

$$SY = \{ c \mid c \text{ is } C_Y \cup C_X, c \text{ is consistent} \}$$

where,

$$C_Y = notinvolve(T_{Y_1}, \alpha_{i_1}, Y) \cup \ldots \cup notinvolve(T_{Y_k}, \alpha_{i_k}, Y)$$

$$C_X = notinvolve(T_{X_1}, \beta_1, Y) \cup \ldots \cup notinvolve(T_{X_m}, \beta_m, Y)$$

For each $Y$, for each context $c \in SY$, user needs to provide the procedures defined as follows:

Let $P_c$ be a set of predicates defined as follows:

$$P_c = involve(T_{X_1}, p_{i_1}, Y) \cup \ldots \cup involve(T_{X_m}, p_{i_m}, Y)$$
where $p_i$ is a path in $T_{X_i}$ that is consistent with context $c$.

If $P_c$ contains $m$ predicates $B_1(Y_1, X_1) \ldots B_m(Y_m, X_m)$, where $Y_i \subseteq Y$, $X_i \in X_Y$. Suppose $B(Z_1, \ldots, Z_j)_{obs}$ denotes the predicate $B(Z_1, \ldots, Z_j)$ after each observed variable $Z_i$ is replaced by its value. Then, for each $j = 1, m$ there exist procedures for computing the following conditional probabilities:

$$P \left( B_j(Y_j, X_j)_{obs} = \text{true} \mid B_1(Y_{j-1}, X_{j-1})_{obs} \wedge \ldots \wedge B_{j-1}(Y_1, X_1)_{obs} \wedge \text{Cond} \right)$$

where $\text{Cond} = \text{invol}(T_{Y_1}, p_i, Y) \cup \ldots \cup \text{invol}(T_{Y_k}, p_i, Y)$, $p_i$ is a path in $T_Y$ that is consistent with context $c$.

However, if $P(X_i|Pa(X_i) \wedge c \wedge \text{invol}(T_{X_i}, p_i, Y)) = 0$ for each value of $X_i$ that satisfy condition $\text{invol}(T_{X_i}, p_i, Y)$ ($p_i$ is consistent with $c$), then user does not need to provide the procedures for conditional probabilities that involve $B \in \text{invol}(T_{X_i}, p_i, Y)$.

**Example 3.3.3** Consider the Bayesian network shown in Figure 3.2. The variable $Afname$ has large domain (all first names) and does not share predicate with any non-observed variables. The variables $Fname.X$ and $Fname.Y$ are observed variables. The decision tree representations $T_1$, $T_2$, and $T_3$ of $P(Afname|Afname \wedge EF.X \wedge Sex)$, $P(Afname|Afname \wedge EF.X \wedge Sex)$, and $P(Afname|Sex)$ respectively are shown in Figure 3.5.

Suppose we have observed $Fname.X = \text{"david"}$ and $Fname.Y = \text{"davig"}$. To perform inference in the network shown in Figure 3.2, user needs to provide some procedures. The set of contexts $S$ for variable $Afname$ has more than one elements. The context as given below is one of them:

$$\text{con} = \left( \text{Sex} = \text{male} \right) \wedge \left( EF.X = sle \right) \wedge \left( EF.Y = sle \right)$$

58
Figure 3.5: The decision tree representations $T1$, $T2$, and $T3$ of conditional probabilities $P(Fname_X|Afname \land Sex \land EF.X)$, $P(Fname.Y|Afname \land Sex \land EF.Y)$, and $P(Afname|Sex)$ respectively.
Let us consider the procedures that need to be provided for variable Afname, for context con. The paths path1, path2 in the tree representations T1, T2, and T3 as shown in Figure 3.5 are consistent with context con.

\[
\begin{align*}
&\text{involve}(T_{Afname}, \text{path1}, \text{Afname}) : (\text{intable}(\text{Afname}, \text{male}) = \text{yes}) \\
&\text{involve}(T_{Afname}, \text{path2}, \text{Afname}) : (\text{intable}(\text{Afname}, \text{male}) = \text{no}) \\
&\text{involve}(T_{Fname_X}, \text{path1}, \text{Afname}) : (\text{singlet}(\text{Afname}, Fname_X) = \text{yes}) \\
&\text{involve}(T_{Fname_X}, \text{path2}, \text{Afname}) : (\text{singlet}(\text{Afname}, Fname_X) = \text{no}) \\
&\text{involve}(T_{Fname_Y}, \text{path1}, \text{Afname}) : (\text{singlet}(\text{Afname}, Fname_Y) = \text{yes}) \\
&\text{involve}(T_{Fname_Y}, \text{path2}, \text{Afname}) : (\text{singlet}(\text{Afname}, Fname_Y) = \text{no})
\end{align*}
\]

For context con, set \( P_c \) contains two predicates as shown below (after conditioning on observations):

\[
P_c = \{\text{singlet}(\text{Afname, david}), \text{singlet}(\text{Afname, davig})\}
\]

From tree representations T1 and T2 we have the following:

\[
\begin{align*}
P(Fname_X | \text{Afname} \wedge \text{con} \wedge \text{singlet}(\text{Afname}, Fname_X) = \text{no}) &= 0 \\
P(Fname_Y | \text{Afname} \wedge \text{con} \wedge \text{singlet}(\text{Afname}, Fname_Y) = \text{no}) &= 0
\end{align*}
\]

Thus, for context con, and condition \( Cond = (\text{intable}(\text{Afname}, \text{male}) = \text{yes}) \) user needs to provide the procedures for computing the conditional probabilities \( p1 \) and \( p2 \) defined as follows:

\[
p1: P(\text{singlet}(\text{Afname, david}) = \text{yes}|Cond), \text{ and } \\
p2: P(\text{singlet}(\text{Afname, davig}) = \text{yes}|(\text{singlet}(\text{Afname, david}) = \text{yes}) \wedge Cond)
\]
The conditional probability $p_1$ can be computed as follows: The predicate $\text{singlet}(\text{Afname}, \text{david})$ is $\text{true}$ for 125 values of $\text{Afname}$. Out of these 125 values, only one value “davis” exists in male name file. The percentage frequency of name “davis” from male name file is 0.0001. As discussed before, the probability mass of all the names that exist in male name file is 0.9. Thus,

$$P(\text{singlet}(\text{Afname}, \text{david}) = \text{true} | \intable(\text{Afname}, \text{male}) = \text{yes}) = 0.0001/0.9$$

The probability $p_2$ can be computed as follows: As discussed in the computation of $P1$, the conditions $(\text{singlet}(\text{Afname}, \text{david}) = \text{true})$ and $(\intable(\text{Afname}, \text{male}) = \text{true})$ hold only when $\text{Afname}$ is “davis”. This means $\text{Afname}$ can have only one value “davis”. Now, the predicate $\text{singlet}(\text{“davis”}, \text{“davig”})$ is $\text{true}$ as “davis” is single letter apart from “davig”. Thus,

$$P(\text{singlet}(\text{Afname}, \text{“davig”}) = \text{true} | \text{singlet}(\text{Afname}, \text{“david”}) = \text{yes} \land \text{Cond}) = 1$$

In the next section, we present an inference algorithm that using the above mentioned “Input Condition” and procedures can perform efficient inference in $B$.

### 3.4 Inference algorithm: Large Domain VE

Let $B$ be a Bayesian network with $n$ discrete random variables, $\mathcal{X} = \{X_1, \ldots, X_n\}$. Suppose we want to answer some probabilistic query, $P(\Q | E = e)$, where $\Q, E \subseteq \mathcal{X}$. $\Q$ denotes the query variables. We observed that $E = e$, $e \in \text{domain}(E)$, $E = e$ is the evidence. As discussed in Chapter 2, from $B$ we can recursively prune any variable that has no children, and is not observed or queried [Geiger, Verma and Pearl, 1990; Pearl, 1988].
In a Bayesian network that has variables with large domain if a query variable has only a few values, we consider computing the probability distribution over the values of the query variable. If a query variable has a large (or unbounded) number of values, we consider computing the probability of a particular subset (or value) from the domain of the query variable, and the subset can be described extensionally or intensionally. If a query asks for the probability of a particular subset $S$ of query variable $X_b$, we create an extra Boolean child $X_q$ of $X_b$, which is true exactly when the query is true. That is,

$$P(X_q = true|X_b) = \begin{cases} 1 & \text{if } X_b \in S \\ 0 & \text{otherwise} \end{cases}$$

The variable $X_q$ has the same probability as the probability of $X_b \in S$. We thus reduce the problem to one of computing the probability distribution over the values of a variable that has only a few values.

Let $X_1, \ldots, X_s$ be the non-query random variables of $B$, and suppose $X_i$’s are ordered according to some elimination ordering. As discussed in Chapter 2, we can compute the query from a function $f(Q)$, which can be computed from the conditioned joint probability distribution by summing out the variables $X_i$’s in order.

$$f(Q) = \sum_{X_s} \cdots \sum_{X_1} \prod_{i=1}^n P(X_i|Pa(X_i))_{\{E=e\}}$$

The subscripted probabilities have the same meaning as discussed in Chapter 2.

As discussed in Section 3.1, we want to exploit the structure of the problem to perform efficient inference in Bayesian networks that have variables with large domains. To compute the function $f(Q)$ efficiently, our approach is based
on the fact that if the evidence or query does not need to distinguish between each value of a variable, we do not need to reason about each value of the variable separately. Rather, for efficient inference we can treat those values that do not need to be distinguished as a single value. This will reduce the effective domain size of the variables.

**Definition 3.4.1** A **partition** of the domain of a variable \(X\), denoted by \(\text{part}(X)\), is a collection of \(b^1, \ldots, b^k\) of nonempty, mutually disjoint subsets of \(\text{Val}(X)\) such that \(\text{Val}(X) = \bigcup_{i=1}^{k} b^i\). The sets \(b^i\) are called **blocks** of the partition.

**Lemma 3.4.2** Let \(B\) be a Bayesian network with \(n\) discrete random variables, \(X = \{X_1, \ldots, X_n\}\). Suppose there exists a partition of each non-observed variable \(X\) of \(B\) such that: \(\forall Y\) if parents of \(Y\) are \(X_1, \ldots, X_k\) and \(b_i \in \text{part}(X_i)\) for each \(i\). If for context \(C = c\) and \(\forall i\) \(v_i, w_i \in b_i\),

\[
P(Y|X_1 = v_1 \ldots X_k = v_k \land C = c) = P(Y|X_1 = w_1 \ldots X_k = w_k \land C = c)
\]

then,

\[
\sum_{X_s} \prod_{i=1}^{n} P(X_i|\text{Pa}(X_i) \land C = c) = \sum_{b_i \in \text{part}(X_i)} P(b_i|\text{Pa}(X_s) \land C = c) \prod_{i=1, i \neq s}^{n} P(X_i|\text{Pa}(X_i) \land C = c)
\]

**Proof** Summing over all the values of \(X_s\) is equivalent to summing over the partition of the values of \(X_s\). Thus,

\[
\sum_{X_s} \prod_{i=1}^{n} P(X_i|\text{Pa}(X_i) \land C = c) = \sum_{b_s \in \text{part}(X_s)} \sum_{X_s \in b_s} \prod_{i=1}^{n} P(X_i|\text{Pa}(X_i) \land C = c)
\]

63
To compute \( \sum_{X_t \in b_t} \prod_{i=1}^n P(X_i | Pa(X_i) \land C = c) \), from the product \( \prod_{i=1}^n P(X_i | Pa(X_i) \land C = c) \) we can distribute out all of the conditional probabilities that do not involve \( X_s \) out of the sum. After distributing out the conditional probabilities, inside the sum \( \sum_{X_t \in b_t} \) we have conditional probability of \( X_s \) given its parents and the conditional probabilities of \( X_s \)'s children. Suppose \( X_1, \ldots, X_p \) are the children of \( X_s \). Then, inside the sum \( \sum_{X_t \in b_t} \) we have the following terms:

\[
\sum_{X_t \in b_t} P(X_s | Pa(X_s) \land C = c) \prod_{j=1}^p P(X_j | Pa(X_j) \land C = c)
\]

If equation (3.4) is true, all the children \( X_j \)'s of \( X_s \) have the same conditional probabilities for each value \( x_j \in b_j \), we can distribute the product \( \prod_{j=1}^p P(X_j | Pa(X_j) \land C = c) \) out of the sum, leaving the term \( \sum_{X_t \in b_t} P(X_s | Pa(X_s) \land C = c) \), which is equal to \( P(b_s | Pa(X_s) \land C = c) \). Thus,

\[
\sum_{X_t} \prod_{i=1}^n P(X_i | Pa(X_i) \land C = c)
= \sum_{b_s \in \text{part}(X_s)} P(b_s | Pa(X_s) \land C = c) \prod_{i=1, i \neq s}^n P(X_i | Pa(X_i) \land C = c)
\]

When there is structure in the problem that allows us to partition the domain of variables in subsets, the above lemma provides a way for summing such variables efficiently. If there is structure, given the observation and query, we need to construct such partitions dynamically.

To perform inference in Bayesian networks whose CPDs are represented using the CPD language representation discussed in Section 3.3, we introduce an inference algorithm based on the variable elimination (VE) algorithm [Zhang and
Poole, 1994], which we call Large Domain VE. Large Domain VE partitions the
domains of the large-domain variables dynamically based on the contexts.

In VE, a factor is the unit of data used during computation. A factor is a
function over a set of variables. The core of Large domain VE is the same as the
VE algorithm. The VE algorithm requires three basic operations over factors:

- conditioning on observations
- multiplying factors
- summing out a variable or variables from a factor

We adapt the above three operations for Large Domain VE so that they can
be applied to the richer representation of the factors. Initially, the factors represent
the CPDs in CPD language form. After conditioning on the observations the factors
are represented in factor language form. Suppose $X_1, \ldots, X_n$ are the variables of
a factor. The BNF of the factor language is shown in Figure 3.6. In the factor
language representation $number$ is a real constant between $[0, 1]$; $C(X_1, \ldots, X_n)$ is
a condition can be described intensionally or extensionally as discussed in Section
3.3.

\[
\langle Cp \rangle ::= \langle number \rangle |
\langle \text{if } C(X_1, \ldots, X_n) \text{ then } \langle Cp \rangle \text{ else } \langle Cp \rangle \rangle
\]

Figure 3.6: A BNF grammar of factor language.

A Large Domain VE factor is more complicated than a VE factor repre-
sented as a table. In Large Domain VE, we represent the factors as sentences in
factor language. As discussed in Section 3.3, a sentence in CPD language can be seen as a decision tree shown in Figure 3.4. Thus, a sentence in factor language representation can also be seen as a decision tree, where a leaf has only one probability value.

**Definition 3.4.3** A factor that involves variables with large domain is termed a **big factor** if the number of conditions in its factor language representation is linear in the domain size of its any variable with large domain.

For the intermediate factors that are created by marginalizing and multiplying factors, we partition the domains of large-domain variables dynamically.

### 3.4.1 Constraint Elimination Ordering

The main message of the Lemma 3.4.2 is that to sum out a variable $X$ with large domain, we need to compute the probability masses for the values of $X$ that are in some sets. To compute the probability mass efficiently (without enumerating the values of $X$), we need procedures that can compute the probability of a set of values of $X$, conditioned on other sets of values of $X$ (this is explained in Section 3.4.4 how we use these.). These procedures are provided by the user as part of the input as discussed in Section 3.3.1.

When the parents of $X$ have been summed out before $X$, a new partition of $X$ will be created. In this case, when we sum out $X$, we need different procedures to compute the probability of a set of values of $X$ conditioned on the newly created set of values of $X$. Since we cannot expect a user to anticipate these intermediate
partitions (new conditions), we assume that a user provides us with the local information. Because of this locality constraint, in Large Domain VE we assume that large-domain variables are summed out before their parents.

3.4.2 Operations Over Trees

The operations of multiplying factors together and summing out a variable (or variables) from a factor are based on two tree operations. In this section, we describe these two operations.

Tree Pruning (simplification)

Tree pruning is used to remove redundant interior nodes or redundant subtrees of the interior nodes of a tree. We prune branches that are incompatible with the ancestors in the tree. If the condition is of the form $X \in S$ and the ancestor specifies $X \in S'$, we can carry out an intersection, $S'' = S \cap S'$, to determine the effective constraints. We can then prune any branch where the effective constraint is that a variable is a member of the empty set (which means $S''$ is empty). For example,
consider the subtree of $X \in \{1, 2\}$ of the tree as shown in Figure 3.7 (a). Here, the ancestor specifies $X \in \{1, 2\}$ and one descendant in this subtree specifies $X \in \{3\}$, the descendant can be pruned. Similarly, for the “else” case in this subtree, we can do set difference to determine the effective constraints. The effective constraint for the “else” case is $X \in (\{1, 2\} - \{1\} - \{3\}) = X \in \{2\}$.

The tree shown in Figure 3.7 (a) contains multiple interior nodes labeled $X$ along a single branch. We can simplify it to produce a new tree in which the subtrees of the subsequent occurrence of $X$ that are not feasible are removed. The new simplified tree is shown in Figure 3.7 (b).

The algorithm for pruning a tree is shown in Figure 3.8. $T$ is the tree to be pruned, and $A_{in}$ and $A_{notin}$ denote the set of $(\text{var}, \text{set})$ pairs. A var-set pair $(X, S)$ in $A_{in}$ denotes that $X \in S$, while $(X, S)$ in $A_{notin}$ denotes that $X \notin S$, $X$ is an ancestor variable of $T$. We start with $\text{Prune}(T, \{\}, \{\})$. We traverse the tree $T$ in a top-down manner. At each node, we determine the type of $T$. We test if $T$ is of the form: $(\text{If } C \text{ then } T_1 \text{ else } T_2 )$. If so, we test if the condition $C$ is of the form $X \in S$. If $C$ is of the form $X \in S$, there are three cases:

- If $(X, S') \in A_{in}$, we compute the intersection $S'' = S \cap S'$. There are three cases:
  - If $S''$ is empty, there is no “then” case, so the result is $\text{Prune}(T_2, A_{in}, A_{notin})$.
  - If $S' \subseteq S$, there is no “else” case, so the result is $\text{Prune}(T_1, A_{in}, A_{notin})$.
  - Otherwise, the result is a tree with condition $C$, if branch of the tree is $\text{Prune}(T_1, A_{in} - \{(X, S')\} \cup \{(X, S'')\}, A_{notin})$ and the else branch is $\text{Prune}(T_1, A_{in} - \{(X, S')\} \cup \{(X, S' - S)\}, A_{notin})$.  

68
Function Prune \((T, A_{\text{in}}, A_{\text{notin}})\) returns a decision tree \(T'\)

**Input:** \(T\) is the sentence, \(A_{\text{in}}\) and \(A_{\text{notin}}\) are sets of \((\text{var}, \text{set})\) pairs (for \(A_{\text{in}}\) var-set pair \((\text{var}, \text{set})\) denotes \(\text{var} \in \text{set}\). For \(A_{\text{notin}}\) var-set pair \((\text{var}, \text{set})\) denotes \(\text{var} \notin \text{set}\).)

```plaintext
1: if \(T\) is of form \(\langle \text{if } C \text{ then } T_1 \text{ else } T_2 \rangle\) then
2:   if \(C\) is of form \(X \in S\) then
3:     if \(\langle X, S'\rangle \in A_{\text{in}}\) then
4:       Let \(S'' = S \cap S'\)
5:       if \(S'' = \{\}\) then
6:         return \(\text{Prune}(T_2, A_{\text{in}}, A_{\text{notin}})\)
7:       else if \(S' \subseteq S\) then
8:         return \(\text{Prune}(T_1, A_{\text{in}}, A_{\text{notin}})\)
9:     else
10:    \(T_1' \leftarrow \text{Prune}(T_1, A_{\text{in}} - \{\langle X, S'\rangle\} \cup \{\langle X, S''\rangle\}, A_{\text{notin}})\)
11:   \(T_2' \leftarrow \text{Prune}(T_2, A_{\text{in}} - \{\langle X, S'\rangle\} \cup \{\langle X, S' \cap S''\rangle\}, A_{\text{notin}})\)
12:   return \(\langle \text{if } C \text{ then } T_1' \text{ else } T_2' \rangle\)
13: end if
14: else if \(\langle X, S'\rangle \in A_{\text{notin}}\) then
15:   Let \(S'' = S - S'\)
16:   if \(S'' = \{\}\) then
17:     return \(\text{Prune}(T_2, A_{\text{in}}, A_{\text{notin}})\)
18:   else
19:     \(T_1' \leftarrow \text{Prune}(T_1, A_{\text{in}} \cup \{\langle X, S''\rangle\}, A_{\text{notin}} - \{\langle X, S'\rangle\})\)
20:   \(T_2' \leftarrow \text{Prune}(T_2, A_{\text{in}}, A_{\text{notin}} - \{\langle X, S'\rangle\} \cup \{\langle X, S' \cap S''\rangle\} \cup \{\langle X, S''\rangle\})\)
21: return \(\langle \text{if } C \text{ then } T_1' \text{ else } T_2' \rangle\)
22: end if
23: else
24:   return \(\langle \text{if } C \text{ then } \text{Prune}(T_1, A_{\text{in}} \cup \{\langle X, S\rangle\}, A_{\text{notin}})\) \text{ else } \text{Prune}(T_2, A_{\text{in}}, A_{\text{notin}}) \text{ \rangle}\)
25: end if
26: else
27:   return \(\langle \text{if } C \text{ then } \text{Prune}(T_1, A_{\text{in}}, A_{\text{notin}}) \text{ else } \text{Prune}(T_2, A_{\text{in}}, A_{\text{notin}}) \text{ \rangle}\)
28: end if
29: else
30:   \% \(T\) is not of the form: \(\langle \text{if } C \text{ then } T_1 \text{ else } T_2 \rangle\)
31: return \(T\)
32: end if
```

Figure 3.8: Algorithm for pruning the decision tree \(T\).
• If \( \langle X, S' \rangle \in A_{\text{notin}} \), we compute the set difference \( S'' = S - S' \). There are two cases:

  - If \( S'' \) is empty, the result is \( \text{Prune}(T_2, A_{\text{in}}, A_{\text{notin}}) \).

  - Otherwise, the result is a tree with condition \( C \), if branch of the tree is \( \text{Prune}(T_1, A_{\text{in}} \cup \{ \langle X, S'' \rangle \}, A_{\text{notin}} - \{ \langle X, S' \rangle \}) \) and the else branch is \( \text{Prune}(T_1, A_{\text{in}}, A_{\text{notin}} - \{ \langle X, S' \rangle \} \cup \{ \langle X, S' \cup S'' \rangle \}) \).

• If \( X \) is not assigned in \( A_{\text{in}} \) and \( A_{\text{notin}} \) (which means that \( X \) didn’t appear extensionally in the ancestors of \( T \) in the tree), the result is a tree with condition \( C \), if branch of the tree is \( \text{Prune}(T_1, A_{\text{in}} \cup \{ \langle X, S \rangle \}, A_{\text{notin}}) \) and the else branch is \( \text{Prune}(T_1, A_{\text{in}}, A_{\text{notin}} \cup \{ \langle X, S \rangle \}) \).

  For any variable \( X \), \( \langle X, S \rangle \) appears at most once in \( A_{\text{in}} \) and \( A_{\text{notin}} \). Note that \( X \) itself only appears either in set \( A_{\text{in}} \) or in \( A_{\text{notin}} \), and only once (see line 10,11,19,20, 24,25 and 28 in Figure 3.8, where a var-set pair \( \langle X, S \rangle \) is assigned either to \( A_{\text{in}} \) or \( A_{\text{notin}} \)).

  If \( C \) is not of the form \( X \in S \) (which means that \( C \) is an intensional definition), the result is a tree with condition \( C \), if branch of the tree is \( \text{Prune}(T_1, A_{\text{in}}, A_{\text{notin}}) \) and the else branch is \( \text{Prune}(T_1, A_{\text{in}}, A_{\text{notin}}) \).

  The correctness of the algorithm does not depend on whether we do complete pruning. We haven’t consider checking for compatibility of intensional representations (which may require some theorem proving); whether the algorithm can be more efficient with such operations is still an open question.
Merging Trees

In VE, we need to multiply factors and sum out a variable from a factor. Both of these operations in Large Domain VE are built upon the merging trees operation.

Two trees $T_1$ and $T_2$ can be merged using operation $Op$ to form a single tree that makes all the distinctions made in any of $T_1$ and $T_2$, and with $Op$ applied to the leaves. When we merge $T_1$ and $T_2$, we replace the leaves of tree $T_1$ by the structure of tree $T_2$. The new leaves of the merged tree are labeled with the function, $Op(p_1, p_2)$, where $p_1$ and $p_2$ are the labels of the leaf in $T_1$ and $T_2$, respectively. We write $merge_2(T_1, T_2, Op)$ to denote the resulting tree. If the labels of the leaves are numbers, the leaf value of the new merged tree is evaluated while merging the trees. If the leaf labels are intensional functions, one of the choices is when to evaluate the intensional function. When to evaluate the intentional functions can be considered as a secondary optimization problem. We always apply the pruning operation to the merged tree.

For example, Figure 3.9 shows tree $T_2$ being merged to tree $T_1$ with the addition (+) operator being applied. When we merge two trees and the $Op$ is a multiplication operator ($\times$) then if the value at any leaf of $T_1$ is zero, we keep that leaf of $T_1$ unchanged in the merged tree. We do not put the structure of $T_2$ at that leaf (as shown in Figure 3.10).

The algorithm $merge_2(T_1, T_2, Op)$ for merging two trees $T_1$ and $T_2$ is shown in Figure 3.11. $merge_2(T_1, T_2, Op)$ calls $mergeT(T_1, T_2, op)$ for merging trees $T_1$ and $T_2$. After merging two trees $T_1$ and $T_2$, $merge2$ calls Prune function to remove the redundant nodes or subtrees from the merged tree.
The leaf labels are combined using the plus function $\text{merge2}(T_1, T_2, +)$.

In $\text{mergeT}$ we traverse the tree $T_1$ in a top-down manner. At each node we determine the type of $T_1$. We test if $T_1$ is of the form: $(\text{if } C \text{ then } T' \text{ else } T'')$. If so, the result is a tree with condition $C$, if branch of the tree is $\text{mergeT}(T', T_2, \text{Op})$ and the else branch is $\text{mergeT}(T'', T_2, \text{Op})$.

If $T_1$ is a leaf node (which means that $T_1$ is a number or function), there are two cases:

- If $T_1$ is zero and $\text{Op}$ is $\times$, the result is $T_1$.
- Otherwise, we call $\text{ReplaceLeaves}(T_2, T_1, \text{Op})$ to replace each leaf $p$ of $T_2$ with $\text{op}(p, T_1)$. The result is the updated tree $T_2$.

The function $\text{ReplaceLeaves}(T_1, T_2, \text{Op})$ as shown in Figure 3.11 replaces each leaf $p$ of $T_1$ with $\text{Op}(p, T_2)$, where $T_2$ is a number or function. We traverse
Function $\text{merge2} (T_1, T_2, Op)$ returns a merged decision tree $T'$

Input: $T_1, T_2$, the root of the decision tree, $Op$, the operation

1. $T_p \leftarrow \text{mergeT}(T_1, T_2, Op)$
2. $T \leftarrow \text{Prune}(T_p, \{\}, \{\})$
3. return $T$

Function $\text{mergeT} (T_1, T_2, Op)$ returns a merged decision tree $T'$

Input: $T_1, T_2$, the root of the decision tree, $Op$, the operation

if $T_1$ is of form: (if $C$ then $T'$ else $T''$) then
    return (if $C$ then $\text{mergeT}(T', T_2, Op)$ else $\text{mergeT}(T'', T_2, Op)$)
else if $(T_1 == .0) \land (Op == \times)$ then
    return $T_1$
else
    return $\text{ReplaceLeaves}(T_2, T_1, Op)$
end if

Function $\text{ReplaceLeaves} (T_1, T_2, Op)$ returns a decision tree $T'$$$

Input: $T_1, T_2$, the root of the decision tree, $Op$, the operation

if $T_1$ is number then
    $N \leftarrow \text{simplify} T_1 \ Op \ T_2$
    return $N$
else % $T_1$ is of form: (if $C$ then $T'$ else $T''$)
    return (if $C$ then $\text{ReplaceLeaves}(T', T_2, Op)$ else $\text{ReplaceLeaves}(T'', T_2, Op)$)
end if

Figure 3.11: Algorithm for merging trees $T_1$ and $T_2$ with operation $Op$.  

73
the tree $T_1$ in a top-down manner. At each node we determine the type of $T_1$. If $T_1$ is a number or function, the result is $Op(T_1, T_2)$ (if both $T_1$ and $T_2$ are numbers, $Op(T_1, T_2)$ is evaluated). Otherwise, $T_1$ is of the form: \{ if $C$ then $T'$ else $T''$\}, the result is a tree with condition $C$, if branch of the tree is $ReplaceLeaves(T', T_2, Op)$ and the else branch is $ReplaceLeaves(T'', T_2, Op)$.

We can extend the $merge_2$ operator to a set of trees. Let $Ts$ be a set of trees, $Ts = \{T_0, \ldots, T_n\}$. We can define $merge(Ts, Op)$, where $Op$ is an associative and commutative operator, as follows. We choose a total order of the set, and carry out the following recursive procedure:

$$merge(\{T_0, \ldots, T_n\}, Op) =$$

$$\begin{cases} 
T_0 & \text{if } n = 0 \\
merge2(merge(\{T_0, \ldots, T_{n-1}\}, Op), T_n, Op) & \text{if } n > 0 
\end{cases}$$

### 3.4.3 Conditioning on Observations

When we observe the values taken by certain variables, we need to incorporate the observation into the factors. Given a tree $T$, a set of observed variables $X_j$, $x_j \in Val(X_j)$, let $T^{X_j=x_j}$ denotes the conditioned tree, i.e., tree $T$ after conditioning on $X_j = x_j$.

To compute the conditioned tree $T^{X_j=x_j}$ we traverse $T$ in a top-down manner. If a node in $T$ splits on a function of observed variables $X_k \in X_j$, the observed values $x_k$ of $X_k$ are incorporated by replacing the occurrences of $X_k$ by their observed values $x_k$. To simplify the tree, we compute those conditions (or predicates) that do not contain any variables. However, the computation of the intensional rep-
resentations that involve variables is delayed until we sum out those variables.

**Example 3.4.4** Consider a factor \( f \) corresponds to CPD \( P(Fname.X | Afname \land Sex \land EF.X) \) as shown in Figure 3.12. Suppose we observe \( Fname.X = "david" \). Tree \( T^{Fname.X="david"} \) is shown in Figure 3.13.

![Figure 3.12: A decision tree representation, \( T \), of a factor corresponding to CPD \( P(Fname.X | Afname \land Sex \land EF.X) \).](image)

![Figure 3.13: Conditioned tree \( T^{Fname.X="david"} \), i.e., tree \( T \) after conditioning on \( Fname.X = "david" \).](image)

The computation of predicates *equal* and *singlet* is delayed until we sum out the variable *Afname*. However, to simplify the tree after conditioning, the predicates *intable("david", male)* and *intable("david", female)* are computed. The subtree at
node `intable("david", male)` is replaced by the value of `lookup("david", male)`, which is 0.02363, as `david` appears in the male name file.

The subtree at node `intable("david", female)` is replaced by the probability of a new name, `Pnew`, as `david` does not appear in the female name file. The function `prsing(Fname,X)` is computed, and replaced by $1/125$.

The predicate `equal` gives us the possible value for `Afname` which is equal to `david`. That is, in the context of $EF.X = noerr$, we are implicitly partitioning `Afname` into `{david}` and all of the other names. Similarly, for $EF.X = sle$, we are implicitly partitioning the values of `Afname` into those names that are a single letter apart from `david`, and all of the other names.

The algorithm for conditioning on observations is shown in Figure 3.14. $T$ is the tree to be conditioned, $A$ is the set of `(var, set)` pairs, $Z$ is the random variable such that $T$ represents the conditional probability of $Z$. A var-set pair `(X, s)` in $A$ denotes that $X$ is an observed variable with value $s$. We traverse the tree $T$ in a top-down manner. At each node, we determine the type of $T$. We test if $T$ is of the form: `(if C then T1 else T2 )`. If so, there are three cases:

- $C$ is of the form $Y \in S$ and $(Y, s') \in A$. We compute the intersection $S'' = S \cap \{s'\}$. There are three cases:
  - if $S'' = \{\}$, there is no “then” case, so the result is `CondObs(T2, A)`.
  - if $\{s'\} \subseteq S$, there is no “else” case, so the result is `CondObs(T1, A)`.
  - Otherwise, we replace $S$ in $C$ by $S''$ and the result is a tree with new condition $C$, if branch of the tree is `CondObs(T1, A)` and the else branch
Function \texttt{CondObs}(T, A, Z) returns a conditioned decision tree \( T' \)

\textbf{Input:} \( T \) is the sentence, \( A \) is a set of \( \langle \text{var}, s \rangle \) pairs ( \( \text{var} \) is an observed variable and \( s \) is its observed value.)

\( Z \) is a random variable and \( T \) represents the conditional probability of \( Z \).

1: if \( T \) is of form: \( \langle \text{if } C \text{ then } T_1 \text{ else } T_2 \rangle \) then
2: \hspace{1em} if \( C \) is of form \( Y \in S \) and \( \langle Y, s' \rangle \in A \) then
3: \hspace{2.5em} Let \( S'' = S \cap \{s'\} \)
4: \hspace{2.5em} if \( S'' = \{\} \) then
5: \hspace{3.5em} return \( \text{CondObs}(T_2, A) \)
6: \hspace{2.5em} else if \( \{s'\} \subseteq S \) then
7: \hspace{3.5em} return \( \text{CondObs}(T_1, A) \)
8: \hspace{2.5em} else
9: \hspace{3.5em} \( C \leftarrow \) replace \( S \) in \( C \) by \( S'' \)
10: \hspace{3.5em} return \( \langle \text{if } C \text{ then } \text{CondObs}(T_1, A) \text{ else } \text{CondObs}(T_2, A) \rangle \)
11: end if
12: else if \( C \) is of intensional form, \( C(X_1, \ldots, X_n) \) then
13: \( C \leftarrow \) instantiate each \( X_i \) of \( C \) with \( s_i \) such that \( \langle X_i, s_i \rangle \in A \)
14: \hspace{1em} if \( C \) does not have any variable then
15: \hspace{2.5em} Val \leftarrow \) compute predicate \( C \)
16: \hspace{2.5em} if \( Val \) is true then
17: \hspace{3.5em} return \( \text{CondObs}(T_1, A) \)
18: \hspace{2.5em} else
19: \hspace{3.5em} return \( \text{CondObs}(T_2, A) \)
20: end if
21: end if
22: end if
23: return \( \langle \text{if } C \text{ then } \text{CondObs}(T_1, A) \text{ else } \text{CondObs}(T_2, A) \rangle \)
24: else if \( T \) is of the form \( \langle \text{number} \rangle \) then
25: return \( T \)
26: else \% \( T \) is of the form \( \langle p_X, \text{number} \rangle \)
27: if \( Z \) is not an observed variable then
28: return \( \text{number} \)
29: else
30: \( p'_X \leftarrow \) instantiate \( X \) of \( p_X \) with \( s \) such that \( \langle X, s \rangle \in A \)
31: \( N \leftarrow \) call procedure to compute \( p'_X \)
32: return \( N \)
33: end if
34: end if

Figure 3.14: Algorithm for conditioning the decision tree \( T \) on observations.
is CondObs(T2, A).

- if \( C \) is a predicate (intensional), we instantiate each observed variable of \( C \) with its observed value. If \( C \) does not have any variables, we compute \( C \); if \( C \) is true, the result is CondObs(T1, A); otherwise the result is CondObs(T2, A).

- Otherwise, the result is a tree with condition \( C \), if branch of the tree is CondObs(T1, A) and the else branch is CondObs(T2, A).

If \( T \) is a number, we return \( T \). Otherwise, \( T \) is of the form \( \langle p_X, \text{number} \rangle \). If \( Z \) is not an observed variable, we return \( \text{number} \). Otherwise, we have two cases:

- If \( p_X \) is not a function, we return \( p_X \).

- Otherwise, we instantiate the observed variables of \( p_X \) with their observed values, compute the function and return it.

### 3.4.4 Eliminating Variables

In variable elimination, to eliminate a variable \( Y \), we multiply all of the factors that contain \( Y \), then sum out \( Y \) from the resulting factor. When a variable has a very large domain, we want to sum out all of those values that are in one partition in a single step. Let \( f_Y \) be the factor that involves \( Y \) and its ancestors, and \( f_1, \ldots, f_k \) be the other factors that involve \( Y \). Suppose \( \text{part}(Y) \) denotes the partition of the values of \( Y \) such that the children of \( Y \) have the same conditional probability for each value in block \( b^i \in \text{part}(Y) \). Suppose \( p_j^i \) denotes factor \( f_j \) under condition \( b^i \). Let \( f \) be the
resulting factor after summing out Y. Then,

\[ f = \sum_{Y} f_Y \times f_1 \times \ldots \times f_k \]

\[ = \sum_{b' \in \text{part}(Y)} \sum_{Y \in b'} f_Y \times f_1 \times \ldots \times f_k \]

\[ = \sum_{b' \in \text{part}(Y)} p_{1}^1 \times \ldots \times p_{k}^k \sum_{Y \in b'} f_Y \]

To compute the resulting factor \( f \) in an efficient manner, we consider that the sum \( \sum_{Y \in b'} f_Y \) should be computed without enumerating the values of \( Y \). To make this possible, the input condition as discussed in Section 3.3.1 should be true, and is why it is included.

Before discussing how the domain of \( Y \) is partitioned dynamically, the following example illustrates why we need to sum out more than one variable at a time.

**Motivation for Summing Out Multiple Variables Together**

**Example 3.4.5** Consider the Bayesian network \( B \) shown in Figure 3.15 (a). The variables \( Y \) and \( Z \) are Boolean variables. The variables \( X \) and \( W \) have large domains, suppose \( X \) and \( W \) both have discrete integer values 1...1000. The tree representations of \( P(X) \), \( P(W) \), and \( P(Y|X \wedge W) \) are shown in Figure 3.15 (b). The predicate \( \text{lesseq}300(X) \) means \( X \leq 300 \), \( \text{lesseq}500(W) \) means \( W \leq 500 \), and the predicate \( \text{lessthan}(W, X) \) means \( W < X \). Suppose \( X \) and \( W \) are uniformly distributed in regions \( X \leq 300, X > 300, W \leq 500, \) and \( W > 500 \).

The leaf values in the tree representation of \( P(X) \) and \( P(W) \) represent the probability masses of all those values of \( X \) and \( W \) that reach to that leaf. Suppose
Figure 3.15: (a) A simple Bayesian network. (b) The decision tree representations of conditional probabilities $P(X)$, $P(W)$, and $P(Y|W \land X)$.

we have observed the value of $Z$ and suppose $Y$ is the query variable. To compute the query, suppose we first want to sum out $X$ from $B$. To sum out $X$, we need to multiply $P(Y|X \land W)$ and $P(X)$. Let $f(W, Y)$ be the resulting factor after summing out $X$. Then,

$$f(W, Y) = \sum_X P(X) \times P(Y|W \land X)$$

Note that $P(Y|W \land X)$ has same distribution over the values of $Y$ for both cases: $W < X$ and $W \geq X$. Thus,

$$f(W, Y) = \begin{cases} 
P(Y|W \land X)_{W<X} \sum_{X : W<X} P(X) \\
+ P(Y|W \land X)_{W\geq X} \sum_{X : W\geq X} P(X)
\end{cases} \quad (3.5)$$

where $P(Y|W \land X)_C$ denotes $P(Y|W \land X)$ under condition $C$.  

80
To compute the factor \( f(W, Y) \), we need to compute the terms \( \sum_{X: W < X} P(X) \) and \( \sum_{X: W \geq X} P(X) \) as shown in equation (3.5). Note that to compute \( \sum_{X: W < X} P(X) \) we need to enumerate the values of \( W \) because the set of values of \( X \) that satisfy \( W < X \) is different for each value value of \( W \). We do not want to enumerate the values of \( W \), and we therefore want to delay the computation of terms \( \sum_{W < X} P(X) \) until we sum out \( W \), in which case we are effectively summing out \( W \) and \( X \) simultaneously.

Let \( f(Y) \) be the resulting factor after we sum out together \( X \) and \( W \). Then,

\[
f(Y) = \sum_{X} \sum_{W} P(Y|W \wedge X) \times P(W) \times P(X)
\]

\[
= P(Y|W \wedge X)_{W < X} \sum_{X: W < X} P(W) \times P(X)
\]

\[
+ P(Y|W \wedge X)_{W \geq X} \sum_{X: W \geq X} P(W) \times P(X)
\]

From the tree representations of \( X \) and \( W \), it is evident that \( X \) and \( W \) do not have uniform distribution over their values. This means to compute the terms such as \( \sum_{X, W: W < X} P(X) \times P(W) \), the values of \( X \) and \( W \) need to be further partitioned. Thus,

\[
f(Y) = P(Y|W \wedge X)_{W < X} \sum_{X, W: X \leq 300, W \leq 500, W < X} P(X) \times P(W)
\]

\[
+ P(Y|W \wedge X)_{W \geq X} \sum_{X, W: X \leq 300, W \leq 500, W \geq X} P(X) \times P(W)
\]

\[
+ P(Y|W \wedge X)_{W < X} \sum_{X, W: X < 300, W > 500, W < X} P(X) \times P(W)
\]

\[
+ P(Y|W \wedge X)_{W \geq X} \sum_{X, W: X < 300, W > 500, W \geq X} P(X) \times P(W)
\]

\[
+ P(Y|W \wedge X)_{W < X} \sum_{X, W: X > 300, W \leq 500, W < X} P(X) \times P(W)
\]

\[
+ P(Y|W \wedge X)_{W \geq X} \sum_{X, W: X > 300, W \leq 500, W \geq X} P(X) \times P(W)
\]

\[
81
\]
To compute factor \( f(Y) \), we need the probability mass of \( X \) and \( W \) under various conditions as shown in equation 3.6. For example, consider the computation of probability mass of \( X \) and \( W \) under condition \( X < 300, W < 500, W < X \), which can be calculated as follows:

\[
\sum_{X,W:X<300,W<500,W<X} P(X) \times P(W)
\]

\[
= P(W < X | X \leq 300 \land W \leq 500) \times \sum_{X,W:X<300,W<500} P(X) \times P(W)
\]

\[
= P(W < X | X \leq 300 \land W \leq 500) \times \sum_{X} P(X) \times \sum_{W} P(W)
\]

From the tree representation of \( P(X) \) and \( P(W) \) shown in Figure 3.15 (b), we have the following: \( \sum_{X;X \leq 300} P(X) = 0.8 \), and \( \sum_{W;W \leq 500} P(W) = 0.7 \). Now, knowing the conditional probability \( P(W < X | X \leq 300 \land W \leq 500) \), we can compute \( \sum_{X,W:X<300,W<500,W<X} P(X) \times P(W) \).

Figure 3.16 shows the partitions of the values of \( X \) and \( W \). The area of the triangle \( ABC \) excluding the values that are on line \( AC \) \( (W = X) \) represents the number of values of \( W \) and \( X \) that satisfy the conditions \( W < X, X \leq 300, \) and \( W \leq 500 \). As discussed before, \( X \) and \( W \) both have uniform distribution in the region \( X \leq 300 \) and \( W \leq 500 \). Thus,

\[
P(W < X | X \leq 300 \land W \leq 500) = \frac{(0.5 \times 300 \times 300 - 300)}{(300 \times 500)}
\]

As discussed in Section 3.3.1, we assume that there exists procedures that provide us the probability such as \( P(W < X | X \leq 300 \land W \leq 500) \) for computing.
Figure 3.16: The partitions of the values of $W$ and $X$.

To sum out the variables efficiently (without enumerating the values), we sum out together all those variables whose partitions depend on each other.

**Algorithm**

Let $Y$ be the set of variables to be summed out. $Y$ is a minimal non-empty set of variables that do not share a predicate with a variable outside of $Y$. Suppose $Y_1, \ldots, Y_k$ are the variables in $Y$. Since the parents of $Y$ have not been summed out yet, there will be $k$ factors $f_{Y_1}, \ldots, f_{Y_k}$, where $f_{Y_i}$ corresponds to $P(Y_i|Pa(Y_i))$, and some other factors, say $f_1, \ldots, f_n$, that involve $Y \in Y$. Thus, there are total $n + k$ factors that involve variables $Y$. Suppose $T_{Y_1}, \ldots, T_{Y_k}$ denote the tree representation of $f_{Y_1} \ldots f_{Y_k}$ and $T_1, \ldots, T_n$ denote the tree representations of $f_1, \ldots, f_n$. In Large Domain VE to multiply the factors we can use the “Merging Trees” operation to
multiply the corresponding trees.

Suppose $T$ represents the product of $T_{Y_1}, \ldots, T_{Y_k}, T_1, \ldots, T_n$. A path in tree $T$ corresponds to the blocks (partitions) of the summing variables and their parents. Suppose $\text{Cond}_P$ denotes all the conditions of path $P$ in $T$. The conditions in $\text{Cond}_P$ can be divided into two sets:

$\text{Cond}^{up}_P$: conditions that also appear in trees $T_{Y_1}, \ldots, T_{Y_k}$

$\text{Cond}^{down}_P$: conditions that also appear in trees $T_1, \ldots, T_n$

Note that the conditions in $\text{Cond}^{down}_P$ represent the partitions of the domains of only summing variables, while $\text{Cond}^{up}_P$ represents the partitions of the domains of both summing variables and their parents. Suppose $(f_j)_C$ denotes factor $f_j$ under condition $C$. Then, the leaf value $l_P$ of path $P$ is a number which represents:

$$\prod_{j=1}^n (f_j)_{\text{Cond}^{up}_P} \times \sum_{Y: \text{Cond}^{up}_P} f_{Y_1} \times \ldots \times f_{Y_k}$$

The values of $Y$ represented by $\text{Cond}_P$ are the subset of the values represented by $\text{Cond}^{down}_P$. Thus, $(f_j)_{\text{Cond}_P} = (f_j)_{\text{Cond}^{down}_P}$.

To sum out $Y$ from $T$, for each path $P$ we need to compute the term

$$\prod_{j=1}^n (f_j)_{\text{Cond}_P} \sum_{Y: \text{Cond}_P} f_{Y_1} \times \ldots \times f_{Y_k}$$

that we call the evidenced leaf value of path $P$ and can be computed as follows:

$$\prod_{j=1}^n (f_j)_{\text{Cond}_P} \times \sum_{Y: \text{Cond}_P} f_{Y_1} \times \ldots \times f_{Y_k}$$

$$= \prod_{j=1}^n (f_j)_{\text{Cond}^{down}_P} \times \sum_{Y: \text{Cond}^{up}_P \cap \text{Cond}^{down}_P} f_{Y_1} \times \ldots \times f_{Y_k}$$

$$= P(\text{Cond}^{down}_P|\text{Cond}^{up}_P) \times \prod_{j=1}^n (f_j)_{\text{Cond}^{down}_P} \times \sum_{Y: \text{Cond}^{up}_P} f_{Y_1} \times \ldots \times f_{Y_k}$$
\[ P(\text{Cond}_P^{\text{down}}|\text{Cond}_P^{\text{up}}) \times l_P \]

where \( l_P \) is the leaf value of path \( P \).

Thus, we can compute the term \( \prod_{j=1}^n (f_j)_{\text{Cond}_P} \sum_{Y: \text{Cond}_P} f_{\text{Y}} \times \ldots \times f_{Y_k} \) by multiplying the leaf value \( l_P \) of path \( P \) by the probability \( P(\text{Cond}_P^{\text{down}}|\text{Cond}_P^{\text{up}}) \). We term the probability \( P(\text{Cond}_P^{\text{down}}|\text{Cond}_P^{\text{up}}) \) evidence factor of path \( P \). Suppose \( \text{Cond}_P^{\text{down}} \) has \( n \) \((\text{condition}, \text{value})\) pairs, \( b_1, \ldots, b_n \), in some order. Then,

\[ P(\text{Cond}_P^{\text{down}}|\text{Cond}_P^{\text{up}}) = P(b_1 \land \ldots \land b_n|\text{Cond}_P^{\text{up}}) \]

\[ = \prod_{i=1}^n P(b_i|b_{i-1} \land \ldots \land b_1 \land \text{Cond}_P^{\text{up}}) \]

We assume that when \( b_i \) is of the form \((\text{pred}_i, \text{true})\), there exists procedures that compute the probabilities \( P(b_i|b_{i-1} \land \ldots \land b_1 \land \text{Cond}_P^{\text{up}}) \) (this is the input requirement as discussed in Section 3.3.1). If \( b_i \) is of the form \((\text{pred}_i, \text{false})\), suppose \( b'_i \) denotes \((\text{pred}_i, \text{true})\). Then,

\[ P(b_i|b_{i-1} \land \ldots \land b_1 \land \text{Cond}_P^{\text{up}}) = 1 - P(b'_i|b_{i-1} \land \ldots \land b_1 \land \text{Cond}_P^{\text{up}}) \]

To sum out \( Y \) from tree \( T \), at each leaf we need to compute the evidenced leaf value. Once we have the evidenced leaf value at each leaf, we need to sum the subtrees that correspond to different blocks for a partition of some \( Y_i \in Y \). We need to do this for every context (i.e., for every assignment of ancestors). Thus, summing out \( Y \) from \( T \) involves two steps: 1. computing the evidenced leaf value; and 2. summing the subtrees that corresponds to different partition of \( Y_i \in Y \).

The algorithm \( \text{Sum}(T, Y, \text{Context}_{\text{up}}, \text{Context}_{\text{down}}) \) that combines both computing the evidenced leaf value and summing the subtrees steps is shown in Figure 3.17. \( Y \) is the set of variables that needs to be summed out from \( T, \text{Context}_{\text{up}} \) and
**Function Sum**(\(T, Y, Context_{up}, Context_{down}\)) **returns** a tree

**Input**: \(T\), a node of the decision tree, \(Y\), the summing variables

\(Context_{up}, Context_{down}\) : set of \((condition, value)\) pairs.

1:  if \(T\) is of the form: \(\langle \text{if } C \text{ then } T_1 \text{ else } T_2 \rangle\) then
2:      if \(C\) contains some \(Y \in Y\) then
3:        if \(C\) is in \(P(Y|Pa(Y))\), \(Y \in Y\) then
4:          \(T_1' \leftarrow \text{Sum}(T_1, Y, Context_{up} + \{(C, true)\}, Context_{down})\)
5:          \(T_2' \leftarrow \text{Sum}(T_2, Y, Context_{up} + \{(C, false)\}, Context_{down})\)
6:        else
7:          \(T_1' \leftarrow \text{Sum}(T_1, Y, Context_{up}, Context_{down} + \{(C, true)\})\)
8:          \(T_2' \leftarrow \text{Sum}(T_2, Y, Context_{up}, Context_{down} + \{(C, false)\})\)
9:      end if
10:     return \(\text{merge}\) (\(\{T_1', T_2'\}\), +)
11:  else
12:    \(T_1' \leftarrow \text{Sum}(T_1, Y, Context_{up}, Context_{down})\)
13:    \(T_2' \leftarrow \text{Sum}(T_2, Y, Context_{up}, Context_{down})\)
14:    return \(\langle \text{if } C \text{ then } T_1' \text{ else } T_2' \rangle\)
15:  end if
16:  else % \(T\) is a leaf node
17:      \(\text{Let } L_T\text{ be the value at leaf node } T\)
18:      if \(L_T == 0\) then
19:          return \(L_T\)
20:    else
21:      \(\text{evfactor} \leftarrow \text{Compute_Evi_Factor}(Context_{up}, Context_{down})\)
22:      return \(L_T \times \text{evfactor}\)
23:    end if
24:  end if

Figure 3.17: Algorithm for summing out variables \(Y\) from tree \(T\).
Function Compute_Evi_Factor\((Context_{up}, Context_{down})\) returns a number

Input: \(Context_{up}, Context_{down}\): set of \((condition, value)\) pairs.

Preconditions: There exists procedures for computing the probabilities \(P(b_i|b_{i-1} \land \ldots \land b_1 \land Context_{up})\).

1: Let \(b_1 \ldots b_n\) be the \((condition, value)\) in \(Context_{down}\) in some order
2: \(evfactor \leftarrow 1\)
3: for all \(b_i\) do % we need to compute \(P(b_i|b_{i-1} \land \ldots \land b_1 \land Context_{up})\)
4: if \(b_i\) is of the form \(\langle cond, false \rangle\) then
5: \(b'_i \leftarrow \langle cond, true \rangle\)
6: \(p_m \leftarrow\) Call procedure to compute \(P(b'_i|b_{i-1} \land \ldots \land b_1 \land Context_{up})\)
7: \(evfactor \leftarrow evfactor \times (1 - p_m)\)
8: else
9: \(p_m \leftarrow\) Call procedure to compute \(P(b_i|b_{i-1} \land \ldots \land b_1 \land Context_{up})\)
10: \(evfactor \leftarrow evfactor \times p_m\)
11: end if
12: end for
13: return \(evfactor\)

Figure 3.18: Function Compute_Evi_Factor called by algorithm Sum.

\(Context_{down}\) are set of \((condition, value)\) pairs that are above \(T\) in the tree. \(Context_{up}\) records those conditions that appear in \(P(Y_i|Pa(Y_i))\) \(Y_i \in Y\), while \(Context_{down}\) records those conditions that do not appear in \(P(Y_i|Pa(Y_i))\) \(Y_i \in Y\). In particular, \(Context_{up}\) and \(Context_{down}\) track the predicates (or conditions) and their values for computing the evidence factors of paths in \(T\).

We start with \(Sum(T, Y, \{\}, \{\})\). We traverse \(T\) in a top-down manner. At each node, we determine the type of \(T\). If \(T\) is of the form: \(\langle if C then T1 else T2 \rangle\), there are two cases:

- If condition \(C\) contains some \(Y \in Y\), we sum out \(Y\) from \(T1\) and \(T2\). Let \(T1'\) and \(T2'\) be the trees after summing out \(Y\) from \(T1\) and \(T2\) respectively. There are two cases:
If the condition C also appears in \( P(Y_i|Pa(Y_i)) \), \( Y_i \in Y \). Then,

\[
T_1' = \text{sum}(T_1, Y, \text{Context}_\text{up} + \{(C, \text{true})\}, \text{Context}_\text{down})
\]

\[
T_2' = \text{sum}(T_2, Y, \text{Context}_\text{up} + \{(C, \text{false})\}, \text{Context}_\text{down})
\]

- Otherwise,

\[
T_1' = \text{sum}(T_1, Y, \text{Context}_\text{up}, \text{Context}_\text{down} + \{(C, \text{true})\})
\]

\[
T_2' = \text{sum}(T_2, Y, \text{Context}_\text{up}, \text{Context}_\text{down} + \{(C, \text{false})\})
\]

We merge \( T_1' \) and \( T_2' \) using plus operator, and the merged tree is the resulting tree after eliminating \( Y \).

- Otherwise, we return \( \langle \text{if } C \text{ then } T_1 \text{ else } T_2 \rangle \)

where,

\[
T_1 = \text{Sum}(T_1, Y, \text{Context}_\text{up}, \text{Context}_\text{down})
\]

\[
T_2 = \text{Sum}(T_2, Y, \text{Context}_\text{up}, \text{Context}_\text{down})
\]

Otherwise, \( T \) is a leaf node. Let \( L_T \) be the leaf value at \( T \). If \( L_T \) is zero, we return \( L_T \). Otherwise, we compute the evidence factor at \( T \). We call function \( \text{Compute.Evi.Factor(Context}_\text{up}, \text{Context}_\text{other}) \) for computing the evidence factor at \( T \). Let \( \text{evfactor} \) be the evidence factor return by the function \( \text{Compute.Evi.Factor} \). Then, the evidenced leaf value of \( T \) is \( L_T \times \text{evfactor} \).

### 3.4.5 Computing the Posterior Distribution

The procedure for computing the posterior distribution in Large Domain VE is the same as in VE. To compute the posterior distribution, we first condition on the
observed variables and then sum out all non-query variables one by one. We can compute the posterior distribution by multiplying the remaining factors and normalizing the remaining factor. The Large Domain VE algorithm is summarized in Figure 3.19.

3.4.6 Complexity of Large Domain VE

Let $B$ be a Bayesian network with large-domain variables made up of $N$ discrete random variables. Let $d$ be the maximum domain size of a variable in $B$. Large Domain VE partitions the domains of large-domain variables dynamically during the inference and sum out all of the values that are in one block in a single step.

**Definition 3.4.6** Let $X$ be a random variable in $B$. The **induced domain size** of $X$ is the number of blocks (disjoint subsets) in the partition of the values of $X$, created by Large Domain VE during the inference. If $X$ is a small-domain variable, the induced domain size of $X$ is same as its domain size.

**Definition 3.4.7** The **induced treewidth** of a graph for a specified elimination ordering is the maximum number of variables in an intermediate factor (minus 1).

**Definition 3.4.8** The **treewidth** of a graph is the minimum induced treewidth over all elimination ordering.

**Definition 3.4.9** The **constrained treewidth** of a graph is the minimum induced treewidth over all elimination ordering that satisfy the constraint.

As discussed in Chapter 2, the complexity of VE on $B$ is $O(Nd^W)$, where $W$ is the treewidth of the graph. Let $b$ be the maximum induced domain size of
Function LargeDomainVE(B, Q, E, e)
Input: B: Bayesian network, Q: query variables, E: observed variables, e: values of E, e ∈ Val(E)
Output: A factor over variables Q

1: Let F be the factors corresponding to original CPDs
2: Let T₅ be the trees corresponding to F
3: Replace each T ∈ T₅ that involves some Eᵢ ∈ E with Tₑᵢ=ₑᵢ.
4: Let Z be the variables of B
5: Z' ← Z − Q
6: while Z' is not empty do
7:      select Y ∈ Z' such that Y is a minimal non-empty set of variables that do not share a predicate with a variable outside of Y
8:      Tₛ ← Eliminate(Y, Tₛ)
9:      Z' ← Z' − Y
10: end while
11: return Normalize(Tₛ, Q)

Function Eliminate(Y, Tₛ)
1: partition Tₛ into:
2: {T₁, . . . , Tₘ} that do not involve Y
3: {Tₘ₊₁ . . . Tₖ} are those that do involve Y
4: Tₘ ← merge({Tₘ₊₁, . . . , Tₖ}, ×)
5: Tₛ ← Sum(Tₘ, Y, {}, {})
6: return {T₁, . . . , Tₘ, Tₛ}

Function Normalize({T₁, . . . , Tₙ}, X)
1: T' ← merge({T₁, . . . , Tₙ}, ×)
2: N ← Sum(T', X, {}, {})
3: T ← ReplaceLeaves(T', N, ÷)
4: return T

Figure 3.19: The Large Domain VE Algorithm.
a variable in $B$. Suppose $W_c$ is the constrained treewidth of the graph. Then, the complexity of Large Domain VE on $B$ is $O(Nb^{W_c})$. Because of the constrained elimination ordering, the constrained graph treewidth $W_c$ could be larger than graph treewidth $W$. However, the induced domain size of a large-domain variable can be significantly smaller than its domain size.

3.5 Application of Large Domain VE to Person Identification

In this section we discuss how to apply Large Domain VE to the Person Identification problem. Let $X$ and $Y$ be two records, which refer to the people to be compared. As discussed in Section 3.2.2, to make the decision we need to condition on the observations and marginalize over the unobserved variables in the Bayesian networks shown in Figures 3.1 and 3.2. Consider the elimination of variable $Afname$ from Bayesian network as shown in Figure 3.2. To eliminate variable $Afname$ we need to multiply all the factors that contain variable $Afname$. Factors that contain variable $Afname$ are the following:

- $f1 (Fname_X, EF_X, Sex, Afname)$
- $f2 (Fname_Y, EF_Y, Sex, Afname)$ and
- $f3 (Afname, Sex)$

As shown in Figure 3.20, T1, T2 and T3 are the decision tree representations of factors $f1$, $f2$, and $f3$ respectively. Suppose for records $X$ and $Y$ we have
observed $F_{name.X} = \text{"david"}$ and $F_{name.Y} = \text{"davig"}$. After conditioning on the observations $T_1^{F_{name.X} = \text{"david"}}$ and $T_2^{F_{name.Y} = \text{"davig"}}$ are shown in Figure 3.21. After multiplying $T_1^{F_{name.X} = \text{"david"}}$, $T_2^{F_{name.Y} = \text{"davig"}}$, and $T_3$ we get a new factor $f (EF.X, EF.Y, Sex, Fname)$. A part of the tree representation $T$ of $f$ is shown in Figure 3.21. After multiplying the factors we want to sum out the variable $Fname$ from factor $f$. That is, we need to sum out $Fname$ from tree $T$. After we sum out the variable $Fname$ from $f$ we get a new factor $f' (EF.X, EF.Y, Sex)$. A part of the tree representation $T'$ of new factor $f'$ is shown in Figure 3.22. In the next section, we discuss how the leaf $p'$ of $T'$ as shown in Figure 3.22 is computed by Large Domain VE.

### 3.5.1 Evaluation of $p'$

The leaf value $p'$ of $T'$ as shown in Figure 3.22 is the sum of the the evidenced leaf values of $p_1$ and $p_2$ of $T$ as shown in Figure 3.22. Let $p_1'$ and $p_2'$ be the evidenced leaf values corresponding to leaf values $p_1$ and $p_2$ respectively. Then, $p' = p_1' + p_2'$.

Let us first consider the computation of the evidenced leaf value $p_1'$. The evidenced leaf value $p_1'$ is computed by multiplying the leaf value $p_1$ with the evidence factor $evfactor_{p1}$ corresponding to $p_1$. As shown in Figure 3.22, there are six conditions in the path from root to leaf $p_1$:

$C_1 : (Sex = male)$, $C_2 : (intable(Fname, male) = true)$,
$C_3 : (EF.X = sle)$,
$C_4 : (singlet(Fname, \text{"david"}) = true)$, $C_5 : (EF.Y = sle)$, and
$C_6 : (singlet(Fname, \text{"davig"}) = true)$
Figure 3.20: The decision tree representations, $T_1$, $T_2$, and $T_3$, of factors $f_1$, $f_2$, and $f_3$ respectively.
Figure 3.21: A decision tree representation, $T$, of new factor $f$ after multiplying trees $T_1$, $T_2$, and $T_3$. 

94
Thus,

\[ evfactor_p_1 = P(\text{singlet}(\text{Afname}, \text{"david"}) = \text{true}|\text{intable}(\text{Afname}, \text{male}) = \text{true}) \times P(\text{singlet}(\text{Afname}, \text{"davig"}) = \text{true}|C4 \land C2) \]

We call the procedures that compute the conditional probabilities required in the computation of \( evfactor_{p_1} \). These procedures are provided by the user as a part of the input. As discussed in Section 3.3.1:

\[ P(\text{singlet}(\text{Afname}, \text{"david"}) = \text{true}|\text{intable}(\text{Afname}, \text{male}) = \text{true}) = 0.0001/0.9 \]

and

\[ P(\text{singlet}(\text{Afname}, \text{"davig"}) = \text{true}|\text{singlet}(\text{Afname}, \text{"david"}) = \text{true} \land C2) = 1 \]

Thus,

\[ p_1' = p_1 \times 0.0001/0.9 \]
Let us now consider the computation of evidenced leaf value, $p2'$. As shown in Figure 3.22, there are six conditions in the path from root to leaf $p2$:

- $C1' : (\text{Sex} = \text{male})$
- $C2' : (\text{intable}(\text{Afname, male}) = \text{false})$
- $C3' : (\text{EF}_X = \text{sle})$
- $C4' : (\text{singlet}(\text{Afname, "david"}) = \text{true})$
- $C5' : (\text{EF}_Y = \text{sle})$
- $C6' : (\text{singlet}(\text{Afname, "davig"}) = \text{true})$

Thus,

$$\text{evfactor}_{p2} = P(\text{singlet}(\text{Afname, "david"}) = \text{true}|\text{intable}(\text{Afname, male}) = \text{false}) \times P(\text{singlet}(\text{Afname, "davig"}) = \text{true}|C4' \land C2')$$

The probability $P(\text{singlet}(\text{Afname, "david"}) = \text{true}|\text{intable}(\text{Afname, male}) = \text{false})$ can be computed as follows: The predicate $\text{singlet}(\text{Afname, "david"})$ is true for 125 values of $\text{Afname}$. As discussed before only one value “davis” out of these values exist in male name file. That is, there are 124 values that are single letter apart from $\text{david}$ and are not in male name file. As discussed before, the probability mass of all the names that are not in male name file is 0.1. Suppose $P_{new}$ is the probability (a very small number) of a new name that is not in male name file and single letter apart from “david”. Then,

$$P(\text{singlet}(\text{Afname, "david"}) = \text{true}|\text{intable}(\text{Afname, male}) = \text{false}) = \frac{124 \times P_{new}}{0.1}$$

The probability $P(\text{singlet}(\text{Afname, "davig"}) = \text{true}|C4' \land C2')$ can be computed as follows: To count efficiently the number of values of $\text{Afname}$ that are single letter apart from both $\text{david}$ and $\text{davig}$, we can first generate the patterns of names...
that are a single letter apart from *david*. For example, *?avid*, where *?* is any letter except *d*. After generating these patterns we test which of these patterns makes the predicate $\text{singlet}(\text{Afname}, \text{davig}) = \text{yes}$. Here, the pattern *?avid* satisfies both predicates $\text{singlet}(\text{Afname}, \text{david})$ and $\text{singlet}(\text{Afname}, \text{davig})$ if (\(? \neq d\land ? \neq g\)). Thus, the possible number of values for *Afname* is 24 that are a single letter apart from both *david* and *davig*\(^9\). Out of these 24 values of *Afname* we have already found that one value exist in male name file (during the computation of $p1'$). Thus, there are only 23 values of *Afname* that are single letter apart from *david* and *davig* and are not in male name file. Thus,

$$
P(\text{singlet}(\text{Afname}, "\text{davig}")) = \text{true} | \text{singlet}(C4' \land C2') = 23/124
$$

and

$$
p2' = p2 \times (23/124) \times (124 \times P_{\text{new}} / 0.1)
$$

### 3.5.2 Experiment

In this section we describe an experiment which we designed to test that by modelling attribute dependence gives better results over the traditional approach which consider the attributes independence. To test the proposed approach, as real databases are confidential, we model a reasonably realistic distribution of attribute values by modelling the people in a set of households and model, for example, how twins are born. We model a small town of 1500 households. We generate the population of the town. The generated population was intended to be a good model of real world population. Persons living in the same household have the same address and phone

\(^9\)As there are 26 letters.
number. The probability that a single person lives in a house is 0.4. The probability that a person is living with a partner is 0.6. For a single person there is a 30% chance of having one child. The chances for the subsequent child is 10%. For each birth there is a 3% chance that twins will be born.

The probability that partners have the same last name is 0.5. For partners there is a 70% chance of having one child. The chances for a subsequent child is 30%. When both partners have different last names then the probability that the child will have mother’s or father’s last name is the same. For each birth there is a 3% chance that twins will be born. Each record of the population contain seven fields: social security number, first name, last name, gender, date of birth, phone number, and postal code. Personal first names and last names are chosen according to the distribution from U.S. census file\textsuperscript{10}.

After creating the true population, we made two datasets, $D_A$ and $D_B$. To create $D_A$ we randomly took 600 records from the true population. We corrupt the records using the database generator of Hernandez and Stolfo [Hernandez and Stolfo, 1995], using typographical errors and movement into the true record. The typographical errors introduced by the generator occur with relative frequency known from previous research on spelling correction algorithms [Peterson, 1986; Pollock and Zamora, 1987]. We place these corrupted records in dataset $D_A$. Similarly, we made the database $D_B$ but we took 1500 records from the true population. We compared each record of dataset $D_A$ with each record of dataset $D_B$. There were 900,000 comparisons in total. In these comparisons there were only 227 duplicate cases. We compute the likelihood ratio considering both attribute dependence and

\footnote{http://www.census.gov/genealogy/names/}
After computing the likelihood ratio between all pairs of records, we set the deciding threshold equal to the maximum of maximum likelihood ratio from both cases. The pair of records with likelihood ratio greater than the deciding threshold were taken as duplicates. We compute the precision and recall.

\[
\text{Precision} = \frac{\text{# of Correctly Identified Duplicate Pairs}}{\text{# of Identified Duplicate pairs}}
\]

\[
\text{Recall} = \frac{\text{# of Correctly Identified Duplicate Pairs}}{\text{# of True Duplicate pairs}}
\]

We reduce the deciding threshold with a step of 1 until the deciding threshold is equal to the minimum of minimum likelihood ratio from both cases. For each value of threshold we compute the precision and recall for both cases.

Figure 3.23: Recall versus precision for both attribute dependence and attribute independence.
As more pairs with lower similarity are labeled as duplicates, recall increases, while precision begins to decrease. Figure 3.23 shows the precision versus recall for both cases. The resulting recall/precision curve shows that with attribute dependence the precision of the prediction is 95% with 100% recall, while with attribute independence precision is 70% for 100% recall. Also, with attribute dependence the 100% accuracy is achieved with more coverage than attribute independence. Results of the experiment show that the performance is improved when attribute dependence is considered over attribute independence for person identification problem.

These experiments are not possible without the inference algorithm in this chapter.

3.6 Conclusion

In this chapter we have argued for making inference in Bayesian networks that contain random variables with large or even unbounded domains. To make the inference efficient we exploit the structure in the problem. The main idea is to partition the domain of variables in equivalence classes that have the same conditional probability for particular values (or particular subset) of other random variables. These equivalence classes can be described extensionally and intensionally. To represent the conditional probability distributions in a compact manner we propose a CPD language that allows us to represent the CPDs using intensional and extensional definitions. We present an inference algorithm, Large Domain VE, which is based on the variable elimination algorithm, that exploit the structure for efficient infer-
ence. Large Domain VE partitions the domains of the variables dynamically during the inference based on what is observed and what is queried.
Chapter 4

Inference with Hierarchically Structured Variables

4.1 Introduction

In the previous chapter, we provide a representation and an inference algorithm for Bayesian networks that have discrete random variables with large domains. To handle the large number of values, we consider that the conditional probabilities can be represented compactly using both intensional (in terms of procedures) and extensional (by listing the values) definitions. The domains of the variables are partitioned at run time depending on what is observed and what is queried. In many problem domains there is a priori structure on the values of a random variable. In particular, the large number of values of a variable is represented a priori as an abstraction hierarchy (or an is-a hierarchy or tree hierarchy) [Pearl, 1986; Mackworth, Mulder and Havens, 1985]. An abstraction hierarchy is a tree where the
nodes represent concepts and the nodes at the top are less specific than the nodes at the bottom. In abstraction hierarchies, information from high-level concepts is inherited by more specific concepts. We hope that the large number of values of a variable can be managed efficiently by considering a hierarchical structure on its values.

The large number of values of many discrete random variables can be represented naturally in a tree hierarchy form. For example, consider a random variable *living things* that describes the types of species on the earth. The values of the *living things* can be represented into a tree hierarchy according to Linnaean taxonomy\(^1\). Living things are divided into kingdoms (e.g., *plantae, animalia*), classes (e.g., mammals, birds, fish), all the way down to species. We call such discrete random variables *hierarchically structured variables* (or *hierarchical variables*). We call discrete random variables that are not hierarchical *simple variables*. The reason we consider hierarchically structured variables is they typically have a good deal of structure; there are some features that are common to higher-level concepts in the hierarchy. For example, birds have feathers and lay eggs. The feature information needs to be stored only at the highest possible level of abstraction, thus achieving economy of storage.

There is very limited work on combining abstraction hierarchies and probabilistic reasoning. Pearl [1986] considered the problem of evidential reasoning in a taxonomic hierarchy of hypotheses using probability theory. Pearl presents an approach for updating probabilities in a taxonomic hierarchy given the observation. However, he considered a very restricted, naive Bayes classifier network form,

\(^1\)http://www.wikipedia.org/wiki/Linnaean\_taxonomy
where there is only one hierarchical variable representing the hypotheses. Pearl's approach cannot be used for making inference in a general Bayesian network setting.

We can use existing inference algorithm for making inference in Bayesian networks with hierarchical variables. However, the basic difficulty is that existing algorithms [Lauritzen and Spiegelhalter, 1988; Zhang and Poole, 1994] flatten the domain of such variables, removing the hierarchical structure and treating them as variables that have as domains all the leaf values. This flattening has several important adverse implications. From a representation point of view, if the domain of the variable is very large, the tabular representation of the conditional probability distribution is very big. For example, consider a hierarchical variable that represents the living things on the earth. After flattening, this variable has millions of species as its values. As another example, consider a hierarchical variable \textit{location} that represents the location of an object on the earth. After flattening, the domain of \textit{location} is unmanageable because it can, for example, take each square centimeter of the earth as a value.

When there is a lot of structure because of the hierarchies (e.g., the features depend on the classes, not on leaves) flattening introduces duplicate entries in the tabular representation and we lose the good properties that we were hoping to achieve from the hierarchically structured values. In addition to the representational inadequacies of Bayesian networks for such variables, there is also the problem of inference. As discussed in Chapter 2, the cost of exact inference in Bayesian networks is exponential in tree width, where the domain size is the base of the exponent.
The hierarchical structure between the values of a variable can be exploited to make the representation compact and reasoning efficient.

In this chapter, we propose an approach for making efficient inference in Bayesian networks that have both simple and hierarchical variables. We begin in Section 4.3 by defining hierarchically structured variables. To represent the hierarchical variables we try to exploit any structure they may have. The structure can be represented in tree hierarchies by considering that there are some features that are common to higher-level concepts. In Section 4.5, we discuss our approach for representing the CPDs of a Bayesian network that has hierarchical variables in a compact manner. We represent the distribution of the hierarchical variables by specifying, for each class, the probability distribution over its immediate subclasses. To represent the conditional probability of a variable conditioned on a hierarchical variable, we use inheritance. That is, we associate the probabilities with the classes (group of values) rather than with each value of the variable.

In Section 4.8, we present an approach for reasoning in Bayesian networks with hierarchically structured variables that dynamically constructs an abstract Bayes network, given some evidence and a query, by collapsing the hierarchies to include only those values necessary to answer the query. This can be done with a single pass over the network. We can answer the query from the abstract Bayesian network using any standard probabilistic inference algorithm, such as variable elimination or stochastic simulation. The domain size of the variables in the abstract Bayesian network is independent of the size of the hierarchies; rather it depends on how many of the classes in the hierarchies are supported directly by the evidence or relevant
to the query. The proposed approach is therefore applicable even when the hierarchies are conceptually infinite. In Section 4.10, we present some empirical results that show that hierarchically structured values of a variable can make the reasoning more efficient than reasoning over the set of all values.

4.2 Related Work

Abstraction hierarchies and semantic nets have been used in artificial intelligence since at least 1968 [Quillian, 1968] for representing concepts and categorical knowledge. The early work that combined tree hierarchies and uncertainty was motivated by the need for diagnosing medical domains that are hierarchical in nature. In a hierarchical hypothesis space, the evidence can appear at any level of abstraction. The earlier rule-based expert systems (e.g., MYCIN) were not capable of combining the evidence that are at different levels of abstraction in a consistent way [Buchanan and Shortliffe, 1984].

Motivated by the inadequacy of expert systems for handling evidence that is at different levels of abstraction, Gordon and Shortliffe [1985] first studied the problem of combining evidence in a hierarchical hypothesis space using the Dempster-Shafer theory. They developed an approximate algorithm for computing the D-S belief functions on tree hierarchies. Shenoy and Shafer [1986] improved the algorithm proposed by Gordon and Shortliffe and showed that belief functions can be propagated using local computations.

Pearl [1986] proposed a Bayesian approach for combining evidence in a hierarchical hypothesis space. Let $H$ be a finite set of mutually exclusive and exhaus-
tive hypotheses, \( H = \{ h_1, h_2, \ldots, h_n \} \). The subsets of the values of \( H \) are organized in a tree hierarchy. That is, each node in the tree represents a set of values of \( H \). \( H \) is the root of the hierarchy and \( h_i \)'s are the leaves in the tree hierarchy. Each leaf \( h_i \) is assigned with a measure of belief \( BEL(h_i) \), denoting the prior probability of \( h_i \). The belief of the intermediate nodes in the hierarchy can be computed from the sum of the belief of its children in a recursive manner.

Suppose we have a new evidence, \( e \), that affects one of the node \( N \) in the hierarchy but does not say anything about the descendants of \( N \) in the hierarchy. Pearl's algorithm [Pearl, 1986] for computing the updated belief of every hypothesis in the tree hierarchy consists of three main steps:

- **Estimation**: Let \( S \) be the hypothesis set, represented by \( N \), that is affected by evidence \( e \). We compute the likelihood ratio \( \lambda_s \) that determines the degree to which the evidence confirms or disconfirms \( S \).

\[
\lambda_s = \frac{P(e|S)}{P(e|\neg S)}
\]

- **Weight distribution**: Each hypothesis \( h_i \in H \) is assigned a weight \( W_i \), as follows:

\[
W_i = \begin{cases} 
\lambda_s & \text{if } h_i \in S \\
1 & \text{otherwise}
\end{cases}
\]

- **Belief updating**: In this step, the updated belief of each hypothesis \( h_i \in H \), denoted by \( BEL'(h_i) \), is computed as follows:

\[
BEL'(h_i) = P(h_i|e) = \alpha_s W_i \times BEL(h_i)
\]

where \( \alpha_s \) is a normalizing factor:

\[
\alpha_s = \frac{1}{\sum_i W_i \times BEL(h_i)}
\]
The updated belief of the intermediate nodes in the tree can be computed from the sum of the belief of its children in a recursive way.

Pearl considered a naïve Bayes classifier network form where there is only one hierarchical variable. We consider a general Bayesian network setting that can have both hierarchical and simple variables. We consider both parents and children of hierarchical variables, which can themselves be simple or hierarchical variables.

In more recent work, Koller and Pfeffer [1998] described a representation language for combining frame-representation systems and Bayesian networks to represent complex structured domains. Their framework uses is-a and part-of hierarchies in an object-oriented way. The work by Koller and Pfeffer is different than ours in that they look at different aspects of the problem, concentrate on multiple objects, and combine first order logic representation with probabilities. In contrast, we are working with propositional logic. We consider the case where the domain of a random variable is hierarchically structured. Our work will allow inheritance in a single node of a Bayesian network. We will exploit inheritance to represent the conditional probabilities in a compact manner and in the probabilistic inference algorithm. Future work can combine these two ideas.

4.3 Hierarchical Variables

In this section, we introduce the notion of hierarchical variables. The hierarchically structured values for a variable define an abstraction hierarchy for mutually exclusive subsets of the values of the variable [Pearl, 1986]. The mutual exclusivity implies that the abstraction hierarchy forms a tree – we do not allow the situation
where a node is a direct descendant of more than one node. Note that there is also no need for the leaves – the tree could be infinitely big (i.e., no need to \textit{a priori} discretize an hypothesis space). Thus, it is possible to model hierarchies that are infinitely detailed (e.g., spatial hierarchies).

\textbf{Definition 4.3.1} In a \textbf{hierarchical variable}, the subsets of the values of the variable are represented hierarchically in a tree. The nodes in the tree represent subsets of the values of the variable, and satisfy the following conditions:

1. The root of the tree represents the set of all the values of the variable.

2. The children of a node correspond to a partitioning of the set of values of the node. That is, the subsets represented by the children of a node in the tree are mutually disjoint and any node in the tree represents the union of its children in the tree.

In the field of Semantic web, the ontologies representations (e.g., OWL, RDFs) are used to represent hierarchical knowledge. Classes in ontologies are hierarchically organised through \textit{is-a} (part-of or subclass-of) relationship, with a partial-order relation. Using the same terminology as in OWL, we refer to the tree hierarchy of a hierarchical variable as a \textbf{class hierarchy} and the nodes in the tree as a \textbf{class}. A class represents the set of values of the variable.

If a class $C_j$ in the class hierarchy is a child of another class $C_i$ in the hierarchy, written as $\text{child}(C_j, C_i)$, we say that $C_j$ is an \textit{immediate subclass} of $C_i$. The \textit{subclass} relation, written as $\leq$, is the transitive closure of the immediate subclass relation; if $C_j \leq C_i$, we say that $C_j$ is a subclass of $C_i$. The inverse of subclass is \textit{superclass}. 

109
The conjunction of two classes $C_j$ and $C_k$, written as $C_j \land C_k$, denotes the set intersection of sets represented by $C_j$ and $C_k$. This is empty unless one is a superclass of the other. If $C_j \land C_k$ is empty, we say that $C_j$ and $C_k$ are disjoint. The set difference of two classes $C_j$ and $C_k$, denoted by $C_j - C_k$, represents the set of all the values that are in the set represented by $C_j$ but not in the set represented by $C_k$.

For example, suppose $L$ is a hierarchical variable that represents the living things on the earth. In the class hierarchy of $L$ we may have the root class livingthing that represents all the values of $L$. We can have plant, and animal as the immediate subclasses of livingthing. The class animal in turn has the immediate subclasses mammal, bird, fish, etc. A part of the class hierarchy of $L$ is shown in Figure 4.1.

4.3.1 Hierarchical Variable and Distribution

Intuitively, a hierarchical variable can be considered as a simple variable that has a large number of values. It may seem natural to define the probability distribution for a hierarchical variable as it is defined for a simple variable. Unfortunately, if there
are infinitely many values, the probability of any value is typically zero. Instead, we need to define a probability measure over the set of values of a hierarchical variable. A probability measure assigns a probability to a set of values rather than to individual values.

Let $X$ be a random variable and $S$ be the set of all possible values that $X$ can take. Let $\mathcal{A}$ be a $\sigma$ algebra over the subset of $S$. A $\sigma$ algebra over a set $S$, denoted by $\mathcal{A}$, is a non-empty collection of subsets of $S$ that is closed under countable unions and formation of complements. Each set $E \in \mathcal{A}$ is a measurable event.

**Definition 4.3.2** A probability measure $P$ on $(S, \mathcal{A})$ is a function that satisfies the following axioms:

- $P(A) \geq 0$ for all $A \in \mathcal{A}$
- $P(S) = 1$
- $P(A \cup B) = P(A) + P(B)$ for all $A, B \in \mathcal{A}$ and $A \cap B = \emptyset$

Let $H$ be a hierarchical variable and $S$ be the set of values of $H$. If, for example, $S$ is finite the power set of $S$, $\mathcal{P}(S)$, is a $\sigma$ algebra over $S$. Then, from definition 4.3.2, the probability of each element $E \in \mathcal{P}(S)$ is between $[0, 1]$, $0 \leq P(E) \leq 1$. The sets denoted by the classes in the hierarchy, closed under countable unions and formation of complements, can form a $\sigma$ algebra over which we can have a probability distribution.
4.4 Bayesian Networks with Hierarchical Variables

In a Bayesian network that has hierarchical variables, we consider that a hierarchical variable can have both hierarchical and simple variables as its parents and children. We assume that the class hierarchies of the hierarchical variables are fixed, i.e., do not change with the context.

Figure 4.2 shows a simple example of a Bayesian network that has both simple and hierarchical variables. The variables *Flying* and *Lay Eggs* are binary variables with values *true* and *false*. The simple variable *Season* represents the seasons on earth. We consider that *Season* has only two values: (northern hemisphere) *summer* and *winter*. The variables *L* and *R* are hierarchical variables. The hierarchical variable *L* represents living things on the earth. The class hierarchy of *L* is shown in Figure 4.1. The hierarchical variable *R* represents regions on the earth. Figure 4.2 (b) shows part of the hierarchy of *R*. At the root, all possible values of *R* are grouped together into the class *earth*. Below this there are two classes – *land* and *ocean*. The class *land* is further subdivided into *northern hemisphere*, *equatorial*, and *southern hemisphere*. Further, each *hemisphere* can be divided into *temperate* and *polar* and so on. The distribution of living things on the earth changes with the season, for example in summer there are more insects than in winter. Similarly, the distribution of living things is different in different regions of the earth (e.g., in the ocean we mostly have fish), so the parents of *L* are *Season* and *R*.

An important aspect of any probabilistic representation is the support for inference; having made some observations, how to condition on these observations and compute the posterior probability distributions. As discussed in Section 4.2,
we cannot use Pearl's algorithm [Pearl, 1986] for making inference in Bayesian networks with hierarchical variables. Existing inference algorithms are not capable of exploiting hierarchical structure; we need to "flatten" the hierarchical variables by removing the hierarchical structure, and treat them as simple variables that have as their domain all of the leaf values. That is, we lose all of the good properties that we were hoping to achieve from the hierarchically structured values.

In this chapter, we look at two related problems with hierarchical variables in Bayesian networks. The first is the compact representation of conditional probability distributions. The second is how to exploit that representation in probabilistic inference for computational gain.
4.5 Specifying the Conditional Probability Distribution

In this section, we discuss how to represent the conditional probability distribution in a compact manner that utilises the structure provided by the class hierarchy. There are two main issues to be considered to represent the CPDs of Bayesian networks with hierarchical variables:

C1: specifying the conditional probability for hierarchical variables;

C2: specifying how variables are conditioned on hierarchical parents.

The simplest case of C1 is how to specify the prior probability of a hierarchical variable. The simplest case of C2 is how to specify the conditional probability of a simple variable conditioned on a hierarchical variable. The more complicated cases are built from these two cases. First, we discuss how we can represent probability distribution for these two cases. Then, we present the general case using the representations developed for these two cases.

4.5.1 Specifying the Prior Probability Distribution for a Hierarchical Variable

The prior probability distribution for a hierarchical variable can be represented in a bottom-up manner, as discussed in the work of Pearl [Pearl, 1986]. In this approach the leaves in the class hierarchy are assigned with the probabilities. Knowing this,
we can compute the probability of any class \((C_k)\) in the class hierarchy by summing the probabilities of the immediate subclasses of that class \((C_k)\) in a recursive manner. Thus,

\[
P(C_k) = \sum_{C_j: \text{child}(C_j, C_k)} P(C_j)
\]

This approach assumes that we can enumerate the probabilities for the leaves. However, if the hierarchies are very big (or even infinite) it may not be practically or theoretically possible to enumerate the leaves. For example, the class hierarchy of a hierarchical variable that describes the location of a person in the world could be infinitely detailed; we may not \textit{a priori} choose a lowest level resolution. To allow for the compact specification of any class that does not depend on the lowest level details, we consider a top-down approach for specifying the prior probability distribution of a hierarchical variable.

In our approach, for each class in the class hierarchy of the variable that has subclasses, we specify a probability distribution over its immediate subclasses; this is the probability of a class given its immediate superclass in the class hierarchy. The top-down approach is a more natural and elegant way of defining the prior probability distribution, and does not assume that we can enumerate the leaves. More formally,

**Definition 4.5.1** Let \(H\) be a hierarchical variable. The prior probability distribution for \(H\) is defined as follows: each link representing \textit{child}(\(C_j, C_k\)) (which means \(C_j\) is the child of \(C_k\)) in the class hierarchy of \(H\) is associated with a value that gives the conditional probability \(P(C_j|C_k)\) (i.e., the conditional probability that value \(C_j\)
holds, given that value $C_k$ holds) such that:

$$\sum_{C_j: \text{child}(C_j,C_k)} P(C_j|C_k) = 1$$

**Example 4.5.2** Suppose we want to specify the prior probability distribution over the hierarchical variable $L$, the class hierarchy of $L$ is shown in Figure 4.1. The root of the hierarchy is the class *livingthing*. Suppose the class *livingthing* has only two children. We specify the probability distribution of *livingthing* over its immediate subclasses:

$$P(\text{animal}|\text{livingthing}) = 0.4$$

$$P(\text{plant}|\text{livingthing}) = 0.6$$

We can specify the probability distribution over the immediate subclasses of *animal*. The class *animal* has many other children that are not shown in Figure 4.1. Suppose we have as part of this distribution:

$$P(\text{mammal}|\text{animal}) = 0.3$$

$$P(\text{bird}|\text{animal}) = 0.2$$, etc.

We can specify the probability of an immediate subclass of *mammal* given *mammal*, with probabilities such as:

$$P(\text{bat}|\text{mammal}) = 0.1$$

In this approach, the prior probability of any class can be computed in a recursive manner by multiplying the probability of class given its immediate superclass and the probability of its immediate superclass. The probability of the root class is 1, since it represents all the values.
Example 4.5.3 Given the probabilities as in Example 4.5.2, $P(bat)$ can be computed as follows:

$$P(bat) = P(bat|mammal) \times P(mammal|animal) \times P(animal|livingthing) \times P(livingthing)$$

$$= 0.1 \times 0.3 \times 0.4 \times 1$$

$$= 0.012$$

Computing the probability of a class involves a series of multiplications. The number of multiplications is proportional to the depth of the class, so the cost of computing the probability of a class is proportional to the depth of the class. We therefore have the following:

Proposition 4.5.4 The time to compute the probability of any class $C_k$ in the class hierarchy is $O(d)$, where $d$ is the depth of $C_k$ in the hierarchy.

4.5.2 Hierarchical Variable Is the Parent of a Simple Variable

Suppose that $H$ is a hierarchical variable, $F$ is a simple variable, and $H$ is the only parent of $F$. To exploit the hierarchical structure, we assume that the influence on $F$ is local in the hierarchy of $H$, so that, generally, related values of $H$ have the same distribution over $F$. We assume that the probability distribution over $F$ given a class in the hierarchy of $H$ is either specified or can be inherited from a superclass of that class in the hierarchy. Using this convention, we do not need to specify the probability distribution over $F$ for each class in the hierarchy.
To specify the conditional probability distribution \( P(F|H) \) in a compact manner, we use the notion of a default probability distribution. The underlying idea is that the default probability distribution of \( F \) for class \( C_j \), written as \( P_d(F|C_j) \), is assumed to be inherited by all subclasses of \( C_j \), unless overridden by a more specific default probability distribution. More formally,

**Definition 4.5.5** The conditional probability distribution \( P(F|H) \) is specified by specifying the default distribution \( P_d(F|C_j) \) for some classes \( C_j \) in the class hierarchy of \( H \) such that there is at least one \( C_j \) along every path from the root. That is, every path down the tree must contain a default \( (P_d) \) distribution. Note that this condition holds trivially if the \( P_d \) value is defined for the root class. These classes \( C_j \) are called exceptional classes of \( H \) with respect to \( F \), denoted by \( e^H_F \). Note that different \( F \)'s may have different exceptional classes.

The conditional probability of \( F \) for any class \( C_k \) of \( H \), \( P(F|C_k) \), can be computed as follows:

- if there is no \( C \in e^H_F \) such that \( C \) is a strict subclass of \( C_k \), \( P(F|C_k) \) is inherited from the default probability distribution of \( F \) for the class \( C_j \in e^H_F \), such that \( C_j \) is the lowest exceptional superclass of \( C_k \),

\[
P(F|C_k) = P_d(F|C_j)
\]

- otherwise, \( P(F|C_k) \) can be derived from \( C_k \)'s immediate subclasses:

\[
P(F|C_k) = \sum_{C_i \text{: child}(C_i,C_k)} P(F|C_i) \times P(C_i|C_k)
\]

118
Associating the default probability distributions with exceptional classes is a compact representation over defining the distribution of $F$ for each value of $H$, when many of the subclasses share the same conditional probability distribution.

Example 4.5.6 Consider representing the conditional probability distribution $P(Flying|L)$ in the Bayesian network as shown in Figure 4.2 (a). To represent $P(Flying|L)$, we need to define the default probability distributions over $Flying$ for the exceptional classes of $L$. For example, we can state that living things have a low probability of flying by default, but bird and insect are exceptional because they have a high probability of flying. From the children of class mammal, a bat is exceptional because it has a high probability of flying. From the children of class bird, a penguin is exceptional because it has a very low probability of flying. Thus, to represent $P(flying|L)$ we could have:

\[
\begin{align*}
P_d(Flying = true|livingthing) &= 0.00001 \\
P_d(Flying = true|bird) &= 0.5 \\
P_d(Flying = true|bat) &= 0.3 \\
P_d(Flying = true|penguin) &= 0.00001 \\
P_d(Flying = true|insect) &= 0.4
\end{align*}
\]

From this we can infer:

- $P(Flying = true|sparrow) = 0.5$, as it inherits this value from bird

- $P(Flying = true|platypus) = 0.00001$, as it inherits this value from living-

\[\text{thing}\]

\[\text{We use a very small probability rather than zero for specifying the default probability of $Flying$ for living things that do not fly. This is because 0 means that flying is impossible. Similarly, we do not have 1.0 for living things that fly. This is to indicate that individuals could be exceptional.}\]
\[
P(Flying = true | penguin) = 0.00001, \text{ as it inherits this value from } penguin
\]

We can compute \( P(Flying | bird) \) from the probabilities of its exceptional subclasses as follows:

\[
P(Flying | bird) = \sum_{C \in \{-penguin \& bird, penguin\}} P(Flying | C) \times P(C | bird)
\]

\[
= P(Flying | \neg penguin \& bird) \times P(\neg penguin \& bird | bird) + P(Flying | penguin) \times P(penguin | bird)
\]

\[
= P_d(Flying | bird) \times (1 - P(penguin | bird)) + P_p(Flying | penguin) \times P(penguin | bird)
\]

\[
= 0.5 \times (1 - P(penguin | bird)) + 0.00001 \times P(penguin | bird)
\]

\[
= 0.5 \times (1 - 0.01) + 0.00001 \times 0.01
\]

\[
= 0.495
\]

Note that \( P(Flying = true | bird) \neq 0.5 \) as the class \( bird \) contains \( penguin \), which have a much lower probability of flying.

The use of "inheritance" with abstraction hierarchies is similar to the use of "context specific independence" (CSI) for the compact representation of the CPTs in Bayesian networks [Boutilier et al., 1996]. In CSI, the CPTs are partitioned into contexts in which the conditional probabilities are equal, and can be represented compactly using decision trees. With inheritance, the CPDs are represented by specifying the default probability distributions for some of the classes. The default probability distribution for a class is inherited by all its subclasses, unless overridden by a more specific default probability distribution.
4.5.3 The General Case

The general case is when a random variable (hierarchical or simple) can have any number and any kind of variables as parents. To represent the conditional probability distribution of a hierarchical variable $H$ conditioned on its parents, we assume that each class in the hierarchy of $H$ has a probability distribution over its immediate subclasses that is conditioned on (some of) the parents of $H$. We can treat each of these classes as a simple variable. Thus, the problem of representing a (conditional) probability distribution over a hierarchical variable reduces to the problem of representing a (conditional) probability distribution over simple variables.

Let $F$ be a simple variable that has both simple and hierarchical parents. Suppose $Pa^s(F)$ denotes its simple parents and $Pa^h(F)$ denotes its hierarchical parents. To represent the conditional probability distribution of $F$ given its parents, we introduce the notion of parent context.

**Definition 4.5.7** Let $F$ be a simple variable that has $n$ hierarchical parents $H_1, \ldots, H_n$. A parent context, denoted by $\mathcal{X}$, is an assignment of a class to each hierarchical parent of $F$. Thus,

$$\mathcal{X} = (c^1_x, \ldots, c^n_x)$$

where $c^i_x$ is a class in the class hierarchy of $H_i$.

Note that if a hierarchical variable is not relevant for some context, this is equivalent to having the root of that hierarchical variable in the parent context.

**Definition 4.5.8** A parent context $\mathcal{X}$ is a superparent context of parent context $\mathcal{Y}$, written as $\mathcal{Y} \leq \mathcal{X}$, if $\forall i \ c^i_y \leq c^i_x$. $\mathcal{X}$ is a proper superparent context of parent context $\mathcal{Y}$, written as $\mathcal{Y} < \mathcal{X}$, iff $\mathcal{Y} \leq \mathcal{X}$ and $\mathcal{Y} \neq \mathcal{X}$ (i.e., $\exists i \ c^i_y < c^i_x$).
The conditional probability of $F$ given its parents, $P(F|\text{pa}^\epsilon(F) \land \text{pa}^h(F))$, is specified by specifying the default probability distribution over $F$, $P_d(F|\text{pa}^\epsilon(F) = x \land \mathcal{X})$, for each assignment $x$ of $\text{pa}^\epsilon(F)$, for some given parent contexts $\mathcal{X}$. These parent contexts $\mathcal{X}$ are called \textbf{default parent contexts} of $F$. The parent contexts $\mathcal{X}$ are such that there is at least one default parent context along every path from the root in $\text{tree}(H_1) \times \ldots \times \text{tree}(H_n)$, where $\text{tree}(H_i)$ represents the class hierarchy of $H_i$.

\textbf{Definition 4.5.9} A \textbf{parent context hierarchy}, denoted by $C_p$, is a partial ordered set of default parent contexts, $C_p = (Z_p, \leq)$, where $Z_p$ is the set of default parent contexts and $\leq$ is a partial order over $Z_p$.

\textbf{Example 4.5.10} Consider the Bayesian network as shown in Figure 4.2. The probability distribution of class \textit{animal} of $L$ over its immediate subclasses can be conditioned on some of its parents. The distribution of class \textit{animal} changes with the values of \textit{Season} and $R$, because in \textit{ocean} we mostly have fish; in the northern hemisphere we have more insects in summer than in winter; and, in the polar region birds migrate in winter to a warmer area.

Suppose the distribution of class \textit{animal} in the \textit{equatorial} region is independent of season, then for parent context $\mathcal{X} = (\text{equatorial})$ we could have a default distribution such as:

\begin{align*}
P_d(\text{reptile}|\text{animal} \land \text{equatorial}) &= 0.1 \\
P_d(\text{insect}|\text{animal} \land \text{equatorial}) &= 0.4 \\
P_d(\text{mammal}|\text{animal} \land \text{equatorial}) &= 0.2
\end{align*}
\[ P_d(\text{fish|animal} \wedge \text{equatorial}) = 0.1 \]
\[ P_d(\text{bird|animal} \wedge \text{equatorial}) = 0.2 \]

In the polar regions, the distribution of class \textit{animal} over its children depends on the season, so for parent context \( \mathcal{X} = (\text{N-polar}) \) we could have a default distribution such as:

\[ P_d(\text{reptile|animal} \wedge \text{Season = summer} \wedge \text{N-polar}) = 0.01 \]
\[ P_d(\text{mammal|animal} \wedge \text{Season = summer} \wedge \text{N-polar}) = 0.04 \]
\[ P_d(\text{insect|animal} \wedge \text{Season = summer} \wedge \text{N-polar}) = 0.6 \]
\[ P_d(\text{fish|animal} \wedge \text{Season = summer} \wedge \text{N-polar}) = 0.05 \]
\[ P_d(\text{birds|animal} \wedge \text{Season = summer} \wedge \text{N-polar}) = 0.2 \]

When a simple variable \( F \) has more than one hierarchical parent, we cannot always compute the conditional probability distribution of \( F \) given a parent context \( \mathcal{X} \) in the same way as discussed in Section 4.5.2, because there could be a problem of multiple inheritance. This arises when conditional distribution over \( F \) for parent context \( \mathcal{X} \) can be inherited from more than one superparent contexts of \( \mathcal{X} \) in context hierarchy \( C_p \) of \( F \).

**Example 4.5.11** Suppose simple variable \( Z \) has two hierarchical parents \( P \) and \( Q \), as shown in Figure 4.3 (a). The class hierarchies for \( P \) and \( Q \) are shown in Figure 4.3 (b). Suppose that in order to define \( P(Z|P \wedge Q) \) we have defined the default distribution over \( Z \) for the parent contexts \( (p,q) \), \( (p_{11},q_{21}) \) and \( (p_{21},q_{11}) \). The parent context hierarchy \( C_p \) of the given parent contexts is shown in Figure 4.3 (c). In this case, for the context \( (p_{21},q_{21}) \) the default distribution over \( Z \) is not defined, it can be
Figure 4.3: (a) A Bayesian network with simple and hierarchical variables. (b) Part of the class hierarchies for variables $P$ and $Q$. (c) Parent context hierarchy of the given parent contexts that define $P(Z|P \land Q)$. 
inherited from the two most specific superparent contexts, \((p_{11}, q_{21})\) and \((p_{21}, q_{11})\), which might specify different distributions.

The above example illustrates the situation where the problem of multiple inheritance can arise. Rather than trying to define a way of combining different distributions, we explicitly disallow this. We assume that there is only one most specific superparent context. If there is more than one most specific compatible superparent context, with different default probability distributions, there must be a default distribution that disambiguates the two default distributions. We are thus forcing the user to disambiguate cases in which there would otherwise be a problem of multiple inheritance.

The conditional distribution over \(F\) for a parent context \(X\) is computed as follows:

- if \(X\) does not have any strict subparent contexts in the parent context hierarchy \(C_p\) of \(F\), \(P(F|X)\) is inherited from the default probability distribution of \(F\) for the superparent context \(Z\) of \(X\) in \(C_p\) that is not overridden by a more specific default probability distribution of \(F\) for parent context \(Y\) such that \(X \leq Y < Z\). Thus,
  \[
P(F|X) = P_d(F|Z)
\]
- otherwise, \(P(F|X)\) can be derived from the children of \(X\) in \(C_p\)
  \[
P(F|X) = \sum_{Z : \text{child}(Z,X)} P(F|Z) \prod_{i=1}^{n} P(c_i^j|c_x^i) + P_d(F|X) \left( 1 - \sum_{Z : \text{child}(Z,X)} \prod_{i=1}^{n} P(c_i^j|c_x^i) \right)
  \]
We have shown how to represent the conditional probability distribution for a Bayesian network that has hierarchical variables in a compact manner. In order to make the inference in a hierarchical Bayesian network, we first discuss what is an observation and query in a Bayesian network that has hierarchical variables.

4.6 Observation in Bayesian Networks with Hierarchical Variables

An observation of a simple variable is an instantiation of the variable. That is, the variable is assigned one value from its domain. However, for a hierarchical variable an observation might support a single value or a set of the values. For example, if a hierarchical variable denotes the possible species of living things, the evidence might support a single species emperor penguin, or might support a set of species such as penguin or penguin & bird, which corresponds to disjunction of its element. We assume that the observation for a hierarchical variable is specified in terms of the classes in its hierarchy. In order to support a set of values that is not defined as a class, evidence can support a class and refute particular subclasses of the class. We term evidence that supports a class a positive observation and evidence that refutes particular values a negative observation. More formally,

**Definition 4.6.1** Let $H$ be a hierarchical variable. An observation about $H$ is positive when we observe that $H$ is in $C_{obs}$, where $C_{obs}$ is a class in the hierarchy of $H$. This means that $H$ has a value $v_h$ such that $v_h \in V$, where $V$ is a set of values referred by $C_{obs}$.  

126
Definition 4.6.2 Let $H$ be a hierarchical variable. An observation about $H$ is **negative** when we observe that $H$ is $\neg C_{obs}$. This means that $H$ has a value $v_h$ such that $v_h \notin V$, where $V$ is a set of values referred by $C_{obs}$.

Without loss of generality, we can assume that there is always one positive observation about $H$. For example, suppose we have two positive observations $C_1$ and $C_2$ about $H$. This means that $v_h \in V_C$, where $V_C$ is a set of values referred by $C_1 \land C_2$. We have two cases:

- If $C_1$ and $C_2$ are disjoint, $C_1 \land C_2 = \emptyset$. This means $v_h \in \emptyset$. That is, the observation about $H$ is not consistent.

- Classes $C_1$ and $C_2$ are not disjoint, i.e., one is a superclass of another. Suppose $C_1$ is a subclass of $C_2$, $C_1 \land C_2 = C_1$. Thus, $v_h \in V_{C_1}$, where $V_{C_1}$ is a set of values referred by $C_1$.

It is sufficient to assume that there is always one positive observation about $H$, denoted by $C_{pos}$. If there are only negative observations about $H$, we assume root class is the positive observation. There can be multiple negative observations about $H$ that are descendants of $C_{pos}$.

Definition 4.6.3 An **allowable observation** in a Bayesian network that has both simple and hierarchical variables is a conjunction of sentences of the form:

- $X = x_i$, where $X$ is a simple variable and $x_i \in Val(X)$.

- $H$ is in $C_{pos}$ and not in $\{C_{neg}^1, \ldots, C_{neg}^k\}$, where $C_{pos}$ is the positive observation about $H$ and $C_{neg}^1, \ldots, C_{neg}^k$ are the negative observations about $H$. 

127
A class $C$ in the hierarchy of $H$ is **true** for the given observations, if the value $v_h$ that $H$ can have is in the set represented by $C$. A class $C$ in the hierarchy of $H$ is **false** for the given observations, if the value $v_h$ that $H$ can have is not in the set represented by $C$. Given the observations about a hierarchical variable $H$, we can divide its class hierarchy into three regions:

**$R_{true}$**: Consists of those classes of $H$ that are true for the given observation (i.e., set of superclasses of $C_{pos}$). Thus,

$$R_{true} = \{C_k \mid C_{pos} \subseteq C_k\}$$

**$R_{false}$**: Consists of those classes of $H$ that are false for the given observation. Thus,

$$R_{false} = \{C_k \mid C_k \not\subseteq C_{pos}, C_{pos} \not\subseteq C_k\} \cup \{C_k \mid C_k \subseteq (\text{negative observations})\}$$

**$R_{unknown}$**: Consists of those classes of $H$ that are not in $R_{true}$ and $R_{false}$. We know for certain that the classes along one path in $R_{unknown}$ are true, but we do not know which path this is.

**Example 4.6.4** Suppose for the hierarchical variable $L$ we have one positive observation, $animal$, and two negative observations, $(bat, penguin)$. That is, $L$ has a value $v$ such that $v \in (V_{animal} - V_{bat} - V_{penguin})$, where $V_{animal}, V_{bat},$ and $V_{penguin}$ are sets corresponding to the classes $animal, bat,$ and $penguin$. We know that $v$ is a member of all the sets corresponding to the superclasses of $animal$, and is not a member of the sets corresponding to the classes that are neither an ancestor nor a descendant of the class $animal$. The descendants $bat$ and $penguin$ of class $animal$ are also false. The regions $R_{true}$ and $R_{false}$ are shown by the bold lines in the class hierarchy of $L$. 

128
Figure 4.4: The class hierarchy of L is divided into three regions $R_{true}$, $R_{unknown}$, and $R_{false}$ by positive observation (animal) and negative observations (bat, penguin) about L.

as shown in Figure 4.4. The classes that are outside of the regions $R_{true}$ and $R_{false}$ are denoted by $R_{unknown}$.

4.7 Query in Bayesian Networks with Hierarchical Variables

Let $B$ be a Bayesian network made up of $n$ discrete random variables, $X = \{X_1, \ldots, X_n\}$. The inference task is to compute $P(Q|e)$, where $Q$ is a simple variable or a class in the hierarchy of hierarchical variable and $e$ is an allowable observations as defined in Section 4.6.

If the query is a simple variable, we compute the probability distribution over the values of the simple variable. For a hierarchical variable, we consider comput-
ing the probability of a particular class that can be at any level of abstraction in the
class hierarchy of hierarchical variable. We gain here because we can compute the
probability of a particular class without computing the probability distribution over
all the values.

Example 4.7.1 Consider the Bayesian network as shown in Figure 4.2. Suppose
we have observed $Flying = true$ and $Lay.Eggs = true$. Given the observations, one
query could be computing the probability that a living thing is a bird.

$$P(L = bird | Flying = true \land Lay.Eggs = true) = ???$$

Another query could be computing the probability that a living thing is a particular
species of the bird. For example,

$$P(L = royal penguin \mid Flying = true \land Lay.Eggs = true) = ???$$

4.8 Inference in Bayesian Network with Hierarchical
Variables

As discussed in the introduction to this chapter, we wish to exploit the hierarchical
structure on the values of the variables to perform efficient inference in Bayesian
networks with hierarchical variables. In particular, we wish to exploit the fact that
for a given problem, only certain classes in the class hierarchies of the hierarchical
variables are supported directly by the evidence or are relevant to the query.

Example 4.8.1 Consider the Bayesian network shown in Figure 4.5. The hierar-
chical variable $L$ represents living things on the earth. The hierarchical variable $R$
represents regions on the earth. The variables *Flying* and *Lay.Eggs* are Boolean variables with values *true* and *false*.

Suppose that the distribution of class *animal* is defined for only two classes in the hierarchy of *R*, *ocean* and *land*. Consider the computation of the likelihood $P(Flying = true)$. We can compute the likelihood $P(Flying = true)$ as follows:

$$P(Flying = true) = \sum_{L,R,Lay.Eggs} P(R) P(L|R) P(Flying = true|L) P(lay.eggs|L)$$

(4.1)

$$= \sum_{L} P(L) \sum_{L} P(Flying = true|L) P(L|R) \sum_{Lay.Eggs} P(Lay.Eggs|L)$$

Note that $\sum_{Lay.Eggs} P(Lay.Eggs|L) = 1$. Next, consider the computation of the term $\sum_{L} P(Flying = true|L) P(L|R)$ in equation (4.1). As discussed in Section 4.5.2, the conditional probability $P(Flying|L)$ can be computed by partitioning the values of $L$ into five disjoint and exclusive subsets, as follows:

$$\forall v \in V_1, \quad P(Flying|L = v) = P_d(Flying|bat)$$

$$\forall v \in V_2, \quad P(Flying|L = v) = P_d(Flying|bird)$$
\[
\forall v \in V_3, \quad P(Flying|L = v) = P_d(Flying|penguin) \\
\forall v \in V_4, \quad P(Flying|L = v) = P_d(Flying|insect) \\
\forall v \in V_5, \quad P(Flying|L = v) = P_d(Flying|livingthing)
\]

where \( V_1 \) is a set of values corresponding to class bat, \( V_2 \) is a set of values corresponding to class difference bird - penguin, \( V_3 \) is a set of values corresponding to class penguin, \( V_4 \) is a set of values corresponding to class insect and \( V_5 \) is a set of values corresponding to class difference livingthing - bat - bird - insect.

Now, the term \( \sum_L P(Flying = true|L) \times P(L|R) \) can be computed as follows:

\[
P(Flying = true|L) \times P(L|R) = P_d(Flying = true|bat) \sum_{v \in V_1} P(L = v|R) \\
+ P_d(Flying = true|bird) \sum_{v \in V_2} P(L = v|R) \\
+ P_d(Flying = true|penguin) \sum_{v \in V_3} P(L = v|R) \\
+ P_d(Flying = true|insect) \sum_{v \in V_4} P(L = v|R) \\
+ P_d(Flying = true|livingthing) \sum_{v \in V_5} P(L = v|R)
\]

Note that \( \sum_{v \in V} P(L = v|R) = P(V|R) \). Thus, we can replace the hierarchical variable \( L \) by a simple variable \( X \) such that:

- \( Val(X) = \{ \text{"insect"}, \text{"bat"}, \text{"penguin"}, \text{"bird - penguin"}, v_x \} \)

where \( v_x \) denotes “livingthing - bat - bird - insect”.

Note that the values \( v \in Val(X) \) are meaningless names that correspond to sets.

- The conditional probability distribution \( P(Flying|X) \) is given by:
\[ P(\text{Flying}|X = \text{"insect"}) = P_d(\text{Flying}|\text{insect}) \]
\[ P(\text{Flying}|X = \text{"bat"}) = P_d(\text{Flying}|\text{bat}) \]
\[ P(\text{Flying}|X = \text{"penguin"}) = P_d(\text{Flying}|\text{penguin}) \]
\[ P(\text{Flying}|X = \text{"bird - penguin"}) = P_d(\text{Flying}|\text{bird}) \]
\[ P(\text{Flying}|X = v_x) = P_d(\text{Flying}|\text{livingthing}) \]

Now, consider the computation of the term \( P(X|R) \). For example, \( P(X = \text{"insect"}|R) \) can be computed as follows\(^3\):

\[ P(X = \text{"insect"}|R) = P(\text{insect} | \text{animal} \land R) \times P(\text{animal} | \text{livingthing} \land R) \]

Note that only the distribution of class \textit{animal} over its children changes with the locations on the earth and is the same at any location on land and ocean. Let \( V_{\text{ocean}} \) and \( V_{\text{land}} \) be the sets corresponding to classes \textit{ocean} and \textit{land}, respectively. Then,

\[ \forall v \in V_{\text{ocean}}, \quad P(X = \text{"insect"}|R = v) \]
\[ = \quad P_d(\text{insect} | \text{animal} \land \text{ocean}) \times P_d(\text{animal} | \text{livingthing}) \]
\[ \forall v \in V_{\text{land}}, \quad P(X = \text{"insect"}|R = v) \]
\[ = \quad P_d(\text{insect} | \text{animal} \land \text{land}) \times P_d(\text{animal} | \text{livingthing}) \]

Similarly, we can compute \( P(X = v|R) \) for any other value \( v \in \text{Val}(X) \). Thus, we can replace the hierarchical variable \( R \) by a simple variable \( Y \), such that:

- \( \text{Val}(Y) = \{\text{"ocean"}, \text{"land"}\} \)
- and,

\(^3\)The value "insect" corresponds to the set of values represented by class \textit{insect}.
\[ P(Y = \text{"ocean")} = P_d(\text{ocean|earth}) \]
\[ P(Y = \text{"land")} = P_d(\text{land|earth}) \]

The hierarchical variables \( L \) and \( R \) can therefore be replaced by the simple variables \( X \) and \( Y \). Note that the domains of \( X \) and \( Y \) depend on those classes of \( L \) and \( R \) that are exceptional for their relevant children.

The above example illustrates that the partitioning of a hierarchical variable that has relevant hierarchical children is governed by those classes that affect the distribution of the superclasses of those exceptional classes of its relevant hierarchical children for which we have evidence. That is, the domain of a hierarchical variable can be partitioned efficiently after partitioning the domain of its relevant hierarchical children.

The idea of the structured inference algorithm is, given some evidence and a query, to construct an abstract Bayesian network by replacing the hierarchical variables with simple variables that only include those values necessary to answer the query. We can then answer the query from the abstract Bayesian network using any standard probabilistic inference algorithm.

**Definition 4.8.2** Let \( H \) be a hierarchical variable with values \( \text{Val}(H) \). An abstraction of the domain of \( H \), denoted by \( \text{part}(H) \), is a partition of \( \text{Val}(H) \). That is, \( \text{part}(H) \) is a set of subsets of \( \text{Val}(H) \) such that all sets in \( \text{part}(H) \) are mutually disjoint and the union of all the sets equals \( \text{Val}(H) \).

**Definition 4.8.3** Let \( H \) be a hierarchical variable and \( \text{part}(H) \) be a partition of its domain. An abstraction of hierarchical variable \( H \) for partition \( \text{part}(H) \) is a
simple variable $H^a$ with the following domain:

$$Val(H^a) = \{v^a \mid v^a \text{ is a meaningless name corresponds to set } b \in part(H)\}$$

We refer to $v^a$ as an abstract value of $H$.

**Definition 4.8.4** An abstract Bayesian network is a Bayesian network that is constructed from a Bayesian network with hierarchical variables by replacing each hierarchical variable by its abstraction.

**Definition 4.8.5** A flattened Bayesian network is a Bayesian network that is constructed from a Bayesian network with hierarchical variables by replacing each hierarchical variable with a simple variable that has all the leaf values as its domain.

**Definition 4.8.6** An exceptional class is relevant exceptional class if there is evidence about it.

### 4.8.1 Construction of an Abstract Bayesian Network

To construct an abstract Bayesian network from a Bayesian network that has hierarchical variables given some evidence and a query, we traverse the Bayesian network from the leaves upwards, prune those variables irrelevant to answer the query, and abstract the hierarchical variables. We abstract the hierarchical variables as part of a pass to remove variables that are irrelevant to the query. From the bottom-up, we can recursively prune any variable that has no children and is not observed or queried [Shachter, 1986; Geiger et al., 1990; Pearl, 1988] as well as abstract hierarchical variables when their children have been abstracted.
Definition 4.8.7 An abstract Bayesian network constructed from a Bayesian network $B$ is safe if it has the same posterior distribution over the query as by the flattened Bayesian network constructed from $B$.

After constructing the abstract Bayesian network, we can use variable elimination or stochastic simulation algorithm to solve the query. The algorithm for constructing an abstract Bayesian network, called Abstract Bayes, is discussed in the next section.

4.8.2 Abstract Bayes

As discussed before, to compute the efficient abstraction of hierarchical variables we need to abstract the hierarchical variables in a bottom-up manner. That is, when we abstract a hierarchical variable $H$, it has only simple children. The hierarchical variable can be a parent of several simple variables. The abstraction of a hierarchical variable $H$, denoted by $H^a$, depends upon the exceptional classes $C_k$ of $H$ with respect to each of these children (initially these are the classes associated with the evidence or the query). In order to develop a safe abstraction, we should follow the following constraints when computing the abstraction for the hierarchical variables:

1. For every evidence or query, there must be an abstract value that directly associates with that evidence/query and conveys its effect.

2. All abstract values must be mutually disjoint and exhaustive.

The algorithm Abstract Bayes is shown in Figure 4.6. It consists of three phases:
Algorithm Abstract_Bayes\((B, e, Q)\)

**Input**  
\(B\): A Bayesian network, \(e\): evidence, \(Q\): query

**Output**  
\(B^a\): An Abstract Bayesian network with respect to \(e\) and \(Q\)

1. **Phase0**:  
2. if \(Q\) is of form \(H = C_q\) then
3.  
4. for all \(H \in H\) do
5.  
6. \(C_q \leftarrow \) query class of \(H\) in \(C_q\)
7. \(C_{root} \leftarrow \) root class in the hierarchy of \(H\)
8. \(X_q \leftarrow \) new Boolean child of \(H\)
9. \(P_d(X_q = true|C_q) \leftarrow 1, P_d(X_q = false|C_{root}) \leftarrow 1\)
10. end for
11. end if

12. **Phase1: Abstract**  
13. Iterate over variables \(H\) of \(B\) in a bottom-up manner
14. if \(H\) is barren variable, Prune \(H\)
15. if \(H\) is hierarchical variable then
16. \(V_{ex} \leftarrow \) set of exceptional classes of \(H\) for its relevant children
17. if \(H\) is observed then
18. \(C_{pos} \leftarrow \) positive observation about \(H\)
19. \(C_{neg} \leftarrow \) set of negative observations about \(H\)
20. \(Val \leftarrow \) AbstractObs\((H, V_{ex}, B, C_{pos}, C_{neg})\)
21. else
22. \(Val \leftarrow \) AbstractHier\((H, V_{ex}, B)\)
23. end if
24. \(H^a \leftarrow \) new simple variable with domain \(Val\)
25. else
26. \(H^a \leftarrow H\)  
27. end if
28. end if

29. **Phase2: Construct Tables**  
30. for all variables \(H^a\) do
31. \(P(H^a|Pa(H^a)) \leftarrow \) ConstructTable\((H^a, B)\)
32. end for
33. \(B^a \leftarrow \) abstract Bayesian network with nodes representing variables \(H^a\) and
34. CPTs \(P(H^a|Pa(H^a))\)
35. Return \(B^a\)

Figure 4.6: Algorithm Abstract_Bayes for constructing abstract Bayesian network.
Function AbstractHier($H, V_{ex}, B$)

**Input** $H$: A Hierarchical variable, $V_{ex}$: A set of exceptional classes of $H$, $B$: A Bayesian network.

**Output** $Val$: A set of abstract values of $H$

1. $V_{affect}(H) \leftarrow \{\}$, $Val \leftarrow \{\}$, $num \leftarrow 0$
2. for all $C_k \in V_{ex}$ do
3.  $C_1, \ldots, C_m \leftarrow$ highest strict subclasses of $C_k$ that are in $V_{ex}$
4.  if $(C_k - C_1 - \ldots - C_m)$ does not represent an empty set then
5.  
6.  $Val[num ++] \leftarrow "C_k - C_1 - \ldots - C_m"
7.  $V_{affect}(H) \leftarrow V_{affect}(H) \cup \{C \mid C_k < C\}$
8. end if
9. end for
10. $Pa^h(H) \leftarrow$ hierarchical parents of $H$ in $B$
11. for all $Y \in Pa^h(H)$ do
12.  Mark those classes of $Y$ exceptional that affect the probability distribution of $C \in V_{affect}(H)$ over its children. % this is for the next step
13. end for
14. Return $Val$

Figure 4.7: Functions AbstractHier called by the algorithm Abstract_Bayes

**Phase 0:** If a query of a Bayesian network asks for the probability of a particular class in hierarchical variable $H$, we create an extra binary child $X_q$ of $H$, which is true exactly when the query is true (i.e., $P_d(X_q = false|C_{root}) = 1$ and $P_d(X_q = true|C_q) = 1$, where $C_{root}$ is the root class and $C_q$ is the query class in the class hierarchy of $H$). The variable $X_q$ has the same probability as the query class. We thus reduce the problem to one of computing the probability of a simple variable.

**Phase 1: Abstract** In this phase, we traverse Bayesian network $B$ from the leaves upwards. We prune a variable that is not queried or observed and does not have any children [Geiger et al., 1990; Pearl, 1988]. For each unpruned hierarchical variable $H$, we compute its abstraction $H^a$. Let $V_{ex}$ be the set of exceptional classes of $H$...
Function AbstractObs \((H, V_\text{ex}, B, C_{\text{pos}}, C_{\text{neg}})\)

**Input** \(H\): A hierarchical variable, \(V_\text{ex}\): A set of exceptional classes of \(H\)
\(B\): A Bayesian network, \(C_{\text{pos}}\): A positive observation about \(H\)
\(C_{\text{neg}}\): A set of negative observations about \(H\)

**Output** \(\text{Val}\): A set of abstract values of \(H\)

1. \(C_{\text{root}} \leftarrow \text{root class of } H, Pa^h(H) \leftarrow \text{hierarchical parents of } H \text{ in } B\)
2. \(V_{\text{affect}}(H) \leftarrow \{\}, \text{Val} \leftarrow [], \text{num} \leftarrow 0\)
3. \(R_{\text{unknown}} \leftarrow \{C | C < C_{\text{pos}}\} - C_{\text{neg}}\)
4. \(V^R_\text{ex} \leftarrow \{C | C \in V_\text{ex} \text{ and } C \in R_{\text{unknown}}\} \cup \{C_{\text{pos}}\}\)
5. for \(\text{all } C_k \in V^R_\text{ex} \) do
6. \(C_1, \ldots, C_m \leftarrow \text{highest strict subclasses of } C_k \text{ that are in } V^R_\text{ex} \cup C_{\text{neg}}\)
7. if \((C_k - C_1 - \ldots - C_m) \text{ does not represent an empty set}\) then
8. \(\text{Val}[\text{num} + +] \leftarrow "C_k - C_1 - \ldots - C_m"\)
9. \(V_{\text{affect}}(H) \leftarrow V_{\text{affect}}(H) \cup \{C | C_k < C\}\)
10. end if
11. end for
12. for \(\text{all } Y \in Pa^h(H) \) do
13. Mark those classes of \(Y\) exceptional that affect the distribution of \(C \in V_{\text{affect}}(H)\)\% this is for the next step
14. end for
15. \(\text{Val}[\text{num} + +] \leftarrow "(C_{\text{root}} - C_{\text{pos}}) \cup C_{\text{neg}}"\)
16. Return \(\text{Val}\)

Figure 4.8: Functions AbstractObs called by the algorithm Abstract_Bayes
with respect to its relevant children. The domain for $H^a$ is computed as follows:

**Case 1: $H$ is not observed** The domain of $H^a$ is the set of non-empty abstract values for each exceptional class. We compute the abstract value, $v^a$, for each class $C_k \in V_{ex}$ as follows:

- $v^a = "C_k"$, if $C_k$ does not have any exceptional strict subclasses.
- Otherwise, $v^a = "C_k - C_1 - \ldots - C_m"$, where $C_1, \ldots, C_m$ are the highest strict subclasses of $C_k$ that are in $V_{ex}$. The abstract value $v^a$ represents the set of all of the values that are in class $C_k$ and are not covered by other abstract values.

The abstract values that denote empty sets are removed from the domain of $H^a$. To abstract the hierarchical parent of $H$, we find their exceptional classes because of $H$. Let $V_{affect}(H)$ be the set of strict super classes of $C_k$'s for which we have the abstract values. Let $Pa^h(H)$ be the set of hierarchical parents of $H$. The classes of $Y \in Pa^h(H)$ that affect the distribution of $C \in V_{affect}(H)$ are marked exceptional because of $H$.

**Case 2: $H$ is observed** Let $C_{pos}$ be the positive observation about $H$ and $C_{neg}$ be the set of negative observations about $H$. The observations about $H$ divide its class hierarchy into three regions $R_{true}$, $R_{unknown}$, and $R_{false}$ as discussed in Section 4.6.

We do not need to distinguish between the values of $H$ that we know are false. Thus, one abstract value of $H$, denoted by $v_{false}$, corresponds to all
those values of $H$ that are false.

\[ v_{false} = \overline{(C_{root} - C_{pos}) \cup C_{neg}} \]

$v_{false}$ denotes the set of values represented by $(C_{root} - C_{pos}) \cup C_{neg}$.

To compute other abstract values of $H$, we need to consider exceptional classes from $V_{ex}$ that are in region $R_{unknown}$ in the hierarchy of $H$. Let $V_{ex}^R$ denotes the set of exceptional classes from $V_{ex}$ that are in region $R_{unknown}$. In order to have an abstract value corresponding to evidence, $C_{pos}$ is incorporated in set $V_{ex}^R$. We compute the abstract value, $v^a$, for each class $C_k \in V_{ex}^R$ as follows:

\[ v^a = \overline{C_k - C_1 - \ldots - C_m} \]

where $C_1, \ldots, C_m$ are the highest strict subclasses of $C_k$ that are in $V_{ex}^R$. The abstract value $v^a$ represents the set of all of the values that are in class $C_k$ and are not covered by other abstract values.

The abstract values that denote empty sets are removed from the domain of $H^a$. To abstract the parents of $H$, for each hierarchical parent $Y$ of $H$ we find $Y$'s exceptional classes because of $H$ in the same way as discussed in Case 1.

**Phase 2: Construct tables** In this phase, we construct the conditional probability table for each variable $H^a$. Suppose $Pa^a(H^a)$ denotes the abstracted parents of $H^a$, and $Pa^{na}(H^a)$ denotes the non-abstracted parents of $H^a$. We compute

\[ P(H^a = v^a | Pa^{na}(H^a) = V_{na} \land Pa^a(H^a) = V_a) \]

for each value $v^a$ of $H^a$, for each assignment $V_{na}$ of $Pa^{na}(H^a)$ and $V_a$ of $Pa^a(H^a)$ for the following three cases:
**Function Construct_Table**(\(H^a, B\))

**Input:** \(H^a\) abstracted variable, \(B\) Bayesian network

**Output:** \(P(H^a|Pa(H^a))\), conditional probability table of \(H^a\)

1. if \(H^a\) is not abstracted and has no abstracted parents then
2. \(P(H^a|Pa(H^a)) \leftarrow\) conditional probability of \(H^a\) in \(B\)
3. else if \(H^a\) is not abstracted but it has abstracted parents then
4. \(P(H^a|Pa(H^a)) \leftarrow Simple\_Var\_Table(H^a, B)\)
5. else
6. \(P(H^a|Pa(H^a)) \leftarrow Hier\_Var\_Table(H^a, B)\)
7. end if
8. Return \(P(H^a|Pa(H^a))\)

Figure 4.9: Function Construct_Table called by the Abstract_Bayes.

**Function Simple_Var_Table**(\(H^a, B\))

**Input:** \(H^a\) abstracted variable, \(B\) Bayesian network

**Output:** \(P(H^a|Pa(H^a))\), conditional probability table of \(X^a\)

1. \(Pa^a(H^a) \leftarrow\) abstracted parents of \(H^a\)
2. \(Pa^{na}(H^a) \leftarrow\) non-abstracted parents of \(H^a\)
3. \(Val \leftarrow [\ ]\)
4. for all assignment \(V_{na}\) of \(Pa^{na}(H^a)\) do
5.   for all assignment \(V_a\) of \(Pa^a(H^a)\) do
6.     context \(\leftarrow [\ ]\)
7.     for all \(Y \in Pa^a(H^a)\) do
8.       \(y \leftarrow\) value of \(Y\) in \(V_a\)
9.       \(C_y^v - C_1^v - \ldots - C_m^v\) \(\leftarrow\) class difference corresponds to \(y\)
10.      context \(\leftarrow\) add \(C_y^v\) in context
11.     end for
12.   end for
13. \(V_s \leftarrow\) inherit \(P(H^a|V_{na} \land context)\) \% the distribution over the values of \(H^a\)
14. \(Val \leftarrow\) add the values \(V_s\) in \(Val\)
15. end for
16. \(P(H^a|Pa(H^a)) \leftarrow\) construct CPT of \(H^a\) with values in \(Val\)
17. Return \(P(H^a|Pa(H^a))\)

Figure 4.10: Function Simple_Var_Table called by the Construct_Table.
Function Hier_Var_Table($H^a$, $B$)

**Input:** $H^a$ abstracted variable, $B$ Bayesian network

**Output:** $P(H^a|Pa(H^a))$, conditional probability table of $X_i^a$

1. $X \leftarrow$ variable corresponds to $H^a$ in $B$
2. $Pa^a(H^a) \leftarrow$ abstracted parents of $H^a$
3. $Pa_{na}(H^a) \leftarrow$ non-abstracted parents of $H^a$
4. $Val \leftarrow [\ ]; num \leftarrow 0$
5. for all assignment $V_{na}$ of $Pa_{na}(H^a)$ do
6. for all assignment $V_a$ of $Pa_{na}(H^a)$ do
7. context $\leftarrow [\ ]$
8. for all $Y \in Pa^a(H^a)$ do
9. $y \leftarrow$ value of $Y$ in $V_a$
10. $C^x_y - C_1^y - \ldots - C_m^y \leftarrow$ class difference corresponds to $y$
11. context $\leftarrow$ add $C^x_y$ in context
12. end for
13. for all $v \in Val(H^a)$ do
14. $C_k - C_1 - \ldots - C_m \leftarrow$ class difference corresponds to $v$
15. $p_i \leftarrow$ compute $P(C_i|V_{na} \land context)$
16. $Val[num + +] \leftarrow p_c - p_1 - \ldots - p_m$
17. end for
18. end for
19. $P(H^a|Pa(H^a)) \leftarrow$ construct CPT of $H^a$ from values in $Val$
20. Return $P(H^a|Pa(H^a))$

Figure 4.11: Function Hier_Var_Table called by the Construct_Table.
Case 0: $H^a$ is not abstracted and has no abstracted parents: Let $H$ be the variable in $B$ that corresponds to $H^a$. Then,

$$P(H^a | Pa(H^a)) = P(H | Pa(H))$$

Case 1: $H^a$ is not abstracted but has abstracted parents: Suppose $H^a$ has $k$ abstracted parents. The assignment $Pa^a(H^a) = V_a$ denotes the abstracted value for each abstracted parent of $H^a$, $V_a = (v^a_1, \ldots, v^a_k)$.

As discussed in Phase 1 of Abstract Bayes, each abstract value $v^i_a$ corresponds to set of values represented by an exceptional class and are not covered by other abstract values. Let $v^i_a$ corresponds to set of values represented by $C^i_{ex} - C^i_1 - \ldots - C^i_m$, where $C^i_{ex}$ is an exceptional classes in the hierarchy of $i^{th}$ hierarchical parent of $H^a$ and $C^i_1, \ldots, C^i_m$ are the highest exceptional strict subclasses of $C^i_{ex}$. Suppose $V$ denotes the parent context, $V = (C^1_{ex}, \ldots, C^k_{ex})$.

Then,

$$P(H^a | Pa^a(H^a) = V_a \wedge Pa^a(H) = V_a) = P(H^a | Pa^a(H^a) = V_a \wedge V)$$

As discussed in Section 4.5.3, $P(H^a = v^a | Pa^a(H^a) = V_a \wedge V)$ is inherited from the default probability distribution of $H^a$ for the super parent context $Z$ of $V$ in $C_{p}$ of $X$ that is not overridden by a more specific default probability distribution of $H^a$ for parent context $Y$, such that $Y \leq V < Z$.

$$P(H^a | Pa^a(H^a) = V_a \wedge Pa^a(H) = V_a) = P_d(H^a | Pa^a(H^a) = V_a \wedge Z)$$

Case 2: $H^a$ is an abstracted variable: Suppose $H^a$ has $k$ abstracted parents. As discussed in Case 1:

$$P(H^a = v^a | Pa^a(H^a) = V_a \wedge Pa^a(H) = V_a)$$

144
Let $\Delta$ denote the assignment: $\Delta = (Pa^{na}(H^a) = V_{na} \wedge V)$. As discussed in Phase 1, suppose the abstract value $v^a$ represents a set of values that corresponds to $C_k - C_1 - \ldots - C_m$, where $C_k$, $C_1, \ldots, C_m$ are the classes in the hierarchy of hierarchical variable corresponding to $H^a$. Thus,

$$P(H^a = v^a|\Delta) = P(C_k|\Delta) - P(C_1|\Delta) - \ldots - P(C_m|\Delta)$$

As discussed in Section 4.5.1, the probability of a class can be computed by multiplying the probabilities up in the class hierarchy. Thus, we can compute $P(C_k|\Delta)$ as follows:

$$P(C_k|\Delta) = P(C_k|C_{k-1} \wedge \Delta) \times P(C_{k-1}|\Delta)$$

where $C_{k-1}$ is the immediate superclass of $C_k$.

Therefore, to compute $P(C_k|\Delta)$ we need the probability distribution of all the strict super classes of $C_k$ over their immediate subclasses for parent context $V$. The probability distribution of a class over its immediate subclasses for parent context $V$ can be computed in the same way as described in Case 1.

We have presented the Abstract Bayes algorithm for constructing an abstract Bayesian network from a Bayesian network with hierarchical variables. Abstract Bayes has a desirable property: it is independent of the particulars of the inference algorithm that we use for computing the posterior distribution. Thus, it can be applied as preprocessing step before computing the posterior distribution (but note that the abstraction depends on both the observation and the query).
Example 4.8.8 We illustrate the Abstract_Bayes algorithm by applying it on a Bayes network that has four simple variables, C, F1, F2, and A, and three hierarchical variables, Q, H, and P as shown in Figure 4.12. The tree hierarchies for the hierarchical variables are shown in Figure 4.13. The thin lines in Figure 4.13 show the class-subclass relationships and the thick lines with arrows show the Bayesian network dependency. In the tree hierarchies, classes that are exceptional (i.e., those classes that have defined default distributions for at least one of the children of hierarchical variable) are marked by "*". In this example, we consider the following:

- The conditional probability $P(F1|H)$ is specified by the default probability distributions $P_d(F1|h)$, $P_d(F1|h1)$, and $P_d(F1|h7)$.

- The conditional probability $P(F2|H)$ is specified by the default probability distributions $P_d(F2|h)$ and $P_d(F2|h3)$.

- The default probability distributions of all the classes of $P$ are defined for the
Figure 4.13: The Bayesian network of Figure 4.12 with trees hierarchies for the hierarchical variables. Thin lines show the class-subclass relationship and thick lines show the Bayesian network dependency. Exceptional classes are marked by *.
root class $h$ of $H$. As shown in Figure 4.13, class $p1$ is affected by class $h3$ of $H$ so the default probability distribution of class $p1$ over its children is also defined for class $h3$. The default probability distribution of classes $p3$ and $p6$ of $P$ is also defined for class $h5$ of $H$.

- The default distribution of all the classes of variable $H$ are defined for the root class $q$ of $Q$. As shown in Figure 4.13, the probability distribution of class $h3$ of $H$ is affected by class $q5$ of $Q$, so the default probability distribution of class $h3$ is also defined for class $q5$. The default probability distributions of class $h8$ of $H$ is also defined for class $q10$.

- Suppose the query is $P(H = h10 | A = a1 \land P = p3)$.

**Construction of abstract Bayesian network:** Since the query is on a hierarchical variable, the algorithm first creates a simple child $X_q$ of $H$, such that:

$$P_d(X_q = true | h10) = 1, \text{ and } P_d(X_q = true | h) = 0$$

To construct the abstract Bayesian network from the Bayesian network $B$ as shown in Figure 4.13, we traverse $B$ from the leaves upwards. We leave the observed simple variable $A$ as it is. The simple variable $F2$ is pruned because it is irrelevant.

We abstract the hierarchical variable $P$.

**Abstraction of $P$ (observed hierarchical variable):** We have observed $P = p3$. $P$ is abstracted by a simple variable $P^a$ with domain $\text{Val}(P^a) = \{"p3","p-p3"\}$. After abstracting $P$, we abstract $H$.  

148
Abstraction of $H$: After pruning $F2$ and abstracting $P$, to abstract $H$ we find those classes of $H$ that are exceptional for $P$, $X_q$, and $F1$. Variable $H$ has five exceptional classes$^4$: $h$, $h1$, $h3$, and $h7$, and $h10$. $H$ is abstracted by a simple variable $H^a$ that has the following domain:

$$\text{Val}(H^a) = \{"h3", "h7", "h10", "h1 - h3 - h7 - h10", "h - h1"\}$$

Abstraction of $Q$: Finally, $Q$ is abstracted. As discussed before, to compute the abstraction of $Q$, only those values are needed that are necessary to compute the conditional probability distribution of $H^a$. Thus,

$$\text{Val}(Q^a) = \{"q5", "q10", "q - q5 - q10"\}$$

After abstracting the hierarchical variables, we compute the conditional probabilities for the abstract Bayesian network. The first case we consider is how $F1$ depends on $H^a$.

Construction of $P(F1|H^a)$: To compute $P(F1|H^a)$, we compute $P(F1|H^a = v^a)$ for each value $v^a$ of $H^a$. The conditional probability table $P(F1|H^a)$ is shown in Table 4.1. Note that the domain of $H^a$ was created such that we can have simple conditional probability tables for its children. There are some abstract values that represent the set of values that are equal to the set of values represented by exceptional classes for $F1$ (those classes that have different distribution over $F1$ in the above table).

$^4$Note that $h5$ is not exceptional. $h5$ tells us the distribution over the children of $p3$ or $p6$, but we have no evidence about either of these.
Table 4.1: Conditional probability table $P(F1|H^a)$.

| $H^a$       | $P(F1|H^a)$ |
|-------------|-------------|
| "h3"        | $P_d(F1|h1)$ |
| "h7"        | $P_d(F1|h7)$ |
| "h10"       | $P_d(F1|h1)$ |
| "h1 - h3 - h7 - h10" | $P_d(F1|h1)$ |
| "h - h1"    | $P_d(F1|h)$  |

Table 4.2: Conditional probability table $P(P^a|H^a)$.

| $H^a$       | $P(P^a = p3|H^a)$ |
|-------------|-------------------|
| "h3"        | $P_d(p3|p \land h3) \times P_d(p1|p \land h)$ |
| "h7"        | $P_d(p3|p \land h) \times P_d(p1|p \land h)$ |
| "h10"       | $P_d(p3|p \land h) \times P_d(p1|p \land h)$ |
| "h1 - h3-h7-h10" | $P_d(p3|p \land h) \times P_d(p1|p \land h)$ |
| "h - h1"    | $P_d(p3|p \land h) \times P_d(p1|p \land h)$ |

Construction of $P(P^a|H^a)$: To compute $P(P^a|H^a)$, we compute $P(P^a = \nu^a|H^a = \nu^a)$ for each value $\nu^a \in Val(P^a)$ and for each assignment $\nu^a$ of $H^a$. The conditional probability of $P^a$ given $H^a$ is shown in Table 4.2. Note that $P^a$ makes different distinctions in $H^a$ than $F1$. This reflects the different exceptional classes in $H$ for $P$ and $F1$. The domain of $H^a$ is the minimal set of values that preserve the distinctions needed and are adequate to answer the query.

The evidence for the observed hierarchical variable $H$ is translated into the corresponding abstract variable $H^a$ of the abstract Bayesian network. The evidence for hierarchical variable $H$ in abstract Bayesian network is the disjunction of all those abstract values of $H^a$ that are not false for the observation.

After constructing the abstract Bayesian network, we can answer the query using any standard probabilistic inference algorithm. However, it is not viable to use
the Junction Tree algorithm, as the construction of the abstract Bayesian network is
query oriented. We can use the Variable Elimination algorithm [Zhang and Poole, 1994] or stochastic simulation to sum out all non-observed, non-query variables
from the abstract Bayesian network. The abstraction is done as part of the standard
elimination of irrelevant variables.

4.8.3 Complexity

To construct an abstract Bayesian network, Abstract.Bayes traverses a Bayesian
network from the leaves upwards, prunes those variables that are irrelevant to the
query and abstracts the relevant hierarchical variables. As discussed in the algo-
rithm, the number of abstract values of a hierarchical variable is less than or equal
to the number of classes in its hierarchy that are exceptional with respect to its
relevant children.

Let $B$ be a Bayesian network with hierarchical variables and made up $N$
discrete random variables. Let $N_h$ be the number of hierarchical variables in $B$ that
need to be abstracted. Let $e$ be the maximum number of exceptional classes of a
relevant hierarchical variable in $B$, and $b_s$ be the maximum domain size of a simple
variable in $B$.

Proposition 4.8.9 Let $p$ be the maximum number of parents of a variable in $B$. Let
$l$ be the maximum domain size of a variable in the abstract Bayesian network. The
number of abstract values for a hierarchical variable is less than or equal to $e$. Thus,
$l = max(e, b_s)$. The size of the abstract Bayesian network $B^a$ constructed from $B$ by
Abstract.Bayes is $O(N \times l^{p+1})$. 

151
Computing the conditional probability table of a hierarchical variable involves a series of multiplications. The number of multiplications is proportional to the depth of the class corresponding to an abstract value, which in turn is the depth of the relevant exceptional class. We can have the following:

**Proposition 4.8.10** The cost of traversing $B$ is $O(N)$. The cost of constructing $B^a$ from $B$, given the observation and query, is $O(N + N_e \times e \times d)$, where $d$ is the largest depth of a relevant exceptional class in the hierarchy of a relevant hierarchical variable.

Suppose after constructing an abstract Bayesian network from a Bayesian network we use Variable Elimination to answer the query. Then the cost of solving a query is the sum of the cost of constructing an abstract Bayesian network and the cost of running Variable Elimination on the abstract Bayesian network. As discussed in Chapter 2, the cost of Variable Elimination is $O(Nb^M)$, where $N$ is the number of variables in the Bayesian network, $b$ is maximum domain size of a variable in the Bayesian network, and $M$ is the largest number of variables in an intermediate factor. We can have the following:

**Proposition 4.8.11** Let $l$ be the maximum domain size of a variable in the abstract Bayesian network. As discussed above, $l = \max(e, b)$. The cost of running Variable Elimination on the abstract Bayesian network is $O(Nl^M)$. 
4.9 Correctness of Abstract Bayesian Network

The abstract Bayesian network is constructed from a Bayesian network with hierarchical variables using the representation and algorithm as described in the previous Sections. The construction of an abstract Bayesian network would of course be useless, if the posterior distribution over the query in the abstract Bayesian network is not equal to the posterior distribution computed from the flattened Bayesian network.

Let $B$ be a Bayesian network with hierarchical variables made up of $n$ discrete random variables, $X = \{X_1, \ldots, X_n\}$, and we want to answer some probabilistic query, $P(Q|E)$, where $Q, E \subseteq X$, $Q$ denotes the query variables. We observed that $E$ is in the set $e$, and $e \subseteq \text{domain}(E)$, $E \in e$ is the evidence. From $B$ we can recursively prune any variable that has no children, and is not observed or queried [Geiger et al., 1990; Pearl, 1988].

Let $X_1, \ldots, X_s$ be the non-query random variables of $B$ and suppose $X_i$'s are ordered according to some elimination ordering. As discussed in Chapter 2, we can compute the query from a function $f(Q)$, which is proportional to $Q$'s posterior distribution.

$$f(Q) = \sum_{X_s} \cdots \sum_{X_1} \prod_{i=1}^n P(X_i|\text{Pa}(X_i))_{\{E \in e\}}$$

The subscripted probabilities have the same meaning as discussed in Chapter 2.

**Lemma 4.9.1** Let $B$ be a Bayesian network with $n$ discrete random variables, $X = \{X_1, \ldots, X_n\}$. Suppose a partition of the domain of each non-observed variable $X$ of $B$ exists such that: $\forall Y$ if the parents of $Y$ are $X_1, \ldots, X_k$ and $b_i \in \text{part}(X_i)$ for
each $i$. If $\forall i \ v_i, w_i \in b_i$,

$$P(Y|X_1 = v_1 \ldots X_k = v_k) = P(Y|X_1 = w_1 \ldots X_k = w_k) \quad (4.2)$$

then,

$$\sum_{X_s} \prod_{i=1}^{n} P(X_i|Pa(X_i)) = \sum_{b_s \in \text{part}(X_s)} \sum_{X_s \in b_s} \prod_{i=1, i \neq s}^{n} P(X_i|Pa(X_i)) \prod_{i=1}^{n} P(X_i|Pa(X_i))$$

**Proof** Summing over all the values of $X_s$ is equivalent to summing over the partition of the values of $X_s$. Thus,

$$\sum_{X_s} \prod_{i=1}^{n} P(X_i|Pa(X_i)) = \sum_{b_s \in \text{part}(X_s)} \sum_{X_s \in b_s} \prod_{i=1}^{n} P(X_i|Pa(X_i))$$

To compute $\sum_{X_s \in b_s} \prod_{i=1}^{n} P(X_i|Pa(X_i))$, from the product $\prod_{i=1}^{n} P(X_i|Pa(X_i))$ we can distribute out all of the conditional probabilities that do not involve $X_s$ out of the sum. After distributing out the conditional probabilities, inside the sum $\sum_{X_s \in b_s}$ we have conditional probability of $X_s$ given its parents and the conditional probabilities of $X_s$'s children. Suppose $X_1, \ldots, X_p$ are the children of $X_s$. Then, inside the sum $\sum_{X_s \in b_s}$ we have the following term:

$$\sum_{X_s \in b_s} P(X_s|Pa(X_s)) \prod_{j=1}^{p} P(X_j|Pa(X_j))$$

If equation (4.2) is true, all the children $X_j$'s of $X_s$ have the same conditional probabilities for each value $x_j \in b_j$ of $X_s$. We can distribute the product $\prod_{j=1}^{p} P(X_j|Pa(X_j))$ out of the sum, leaving the term $\sum_{X_s \in b_s} P(X_s|Pa(X_s))$, which is equal to $P(b_s|Pa(X_s))$. Thus,

$$\sum_{X_s} \prod_{i=1}^{n} P(X_i|Pa(X_i)) = \sum_{b_s \in \text{part}(X_s)} \sum_{X_s \in b_s} P(b_s|Pa(X_s)) \prod_{i=1, i \neq s}^{n} P(X_i|Pa(X_i))$$

154
The above lemma provides us with a way of proving the correctness of the abstract Bayesian network. We can do so by proving that Abstract Bayes constructs the abstraction of the hierarchical variables $X_k$ of $B$ from partition $\text{part}(X_k)$ such that the children of $X_k$ satisfy equation (4.2).

**Lemma 4.9.2** The Abstract Bayes algorithm partitions the domain of each hierarchical variable $X_h$ of $B$ such that all of $X_h$'s children satisfy the equation (4.2).

**Proof** Suppose $\text{part}(X_h)$ denotes the partition of the domain of hierarchical variable $X_h$ computed by Abstract Bayes. Let $X_h^a$ be the abstraction of $X_h$. If $X_h$ is a simple variable, $\text{part}(X_h) = \text{Val}(X_h)$. Abstract Bayes constructs the conditional probability tables for each abstracted variable $X_h^a$ of an abstract Bayesian network using "inheritance" for the following two cases:

**Case 1: $X_h^a$ is not abstracted but some of its parents are:** As discussed in Phase 2 (Construct Table) of Abstract Bayes the conditional probability table of $X_h^a$ is constructed as follows:

Suppose $X_h^a$ has $k$ abstracted parents. The assignment $P_a(X_h^a) = V_a$ denotes the abstracted value for each abstracted parent of $X_h^a$, $V_a = (v_a^1, \ldots, v_a^k)$.

As discussed in Phase 1, each abstract value $v_a^i$ corresponds to set of values represented by an exceptional class and are not covered by other abstract values: Let $v_a^i$ corresponds to set of values represented by $C_{ex}^i - C_1^i - \ldots - C_m^i$, where $C_{ex}^i$ is an exceptional classes in the hierarchy of $i^{th}$ hierarchical parent of $X_h^a$ and $C_1^i, \ldots, C_m^i$ are the highest exceptional strict subclasses of $C_{ex}^i$. Suppose $\mathcal{V}$ denotes
the parent context, \( V = (C_{ex1}, \ldots, C_{exk}) \). Then,

\[
P(X_h^n|Pa^{na}(X_h^n) = V_{na} \land Pa^a(X_h^n) = V_a) = P(X_h^n|Pa^{na}(X_h^n) = V_{na} \land V)
\]

As discussed in Section 4.5.3, \( P(X_h^n = v^a|Pa^{na}(X_h^n) = V_{na} \land V) \) is inherited from the default probability distribution of \( X_h^n \) for the super parent context \( Z \) of \( V \) in parent context hierarchy \( C_p \) of \( X_h \) that is not overridden by a more specific default probability distribution of \( X_h^n \) for parent context \( Y \), such that \( V \leq Y < Z \).

\[
P(X_h^n|Pa^{na}(X_h^n) = V_{na} \land Pa^a(X_h^n) = V_a) = P_d(X_h^n|Pa^{na}(X_h^n) = V_{na} \land Z)
\]

Let \( V_s \) be a set defined as follows:

\[
V_s = \{ \{x^1, \ldots, x^n\} | x^i \in \text{set of values represented by } v^i_a \}
\]

Note that \( C_{ex}^i \) is the lowest exceptional class of \( x^i \). Suppose \( Z \) are the hierarchical variables in \( B \) corresponding to \( Pa^a(X_h^n) \). Then,

\[
\forall v \in V_s, P(X_h^n|Pa^{na}(X_h^n) = V_{na} \land Z = v) = P_d(X_h^n|Pa^{na}(X_h^n) = V_{na} \land Z)
\]

Thus, the values of \( Z \) are partitioned such that \( X_h^n \) has the same probability distribution for all values \( v_i \in b_i \), where \( b_i \in part(Z_i), Z_i \in Z \). This implies equation (4.2).

**Case 2: \( X_h^n \) is abstracted:** As discussed in Phase2, to compute the conditional probability \( P(X_h^n = v^a|Pa^{na}(X_h^n) = V_{na} \land Pa^a(X_h^n) = V) \) we need the conditional distribution of some classes \( C_j \), from the tree hierarchy of \( X_h \), over their immediate subclasses. We can treat the classes \( C_j \) as simple variables. Thus, the proof follows from Case1.
Theorem 4.9.3 The posterior probability $P(Q|E)$ computed from the abstract Bayes network $B''$ of $B$ as constructed by Abstract Bayes is equal to the posterior probability computed from the flattened Bayesian network of $B$.

Proof The proof follows from Lemmas 4.9.1 and 4.9.2.

4.10 Empirical Results

We present some results testing whether the hierarchically structured values of the variables can make reasoning more efficient and to determine when the overhead of the abstraction is worth it. To evaluate our approach, we compare the cost of answering a query from a Bayesian network using the proposed approach, applying first Abstract Bayes then Variable Elimination (Abstract Bayes+VE), to applying Variable Elimination on a flattened Bayesian network (VE).

To test the proposed approach, we could not find any realistic networks. We notice that finding the real problem is like the chicken-and-egg problem, since in order to solve a Bayesian network that has hierarchical variables, one really needs a method for solving it. This problem is resolved by developing a method that can solve a Bayesian network with hierarchical variables. To test whether the hierarchically structured values of the variables can make reasoning more efficient, we model a simple conference - academics domain.
4.10.1 Conference - Academics Domain

Suppose the location of an academic in the world is influenced by the location of a particular conference, and the location of the spouse of the academic is influenced by the location of the academic. Suppose the location of the conference in the world does not depend on anything we are modelling. Let there be $k$ academics. We assume that the locations of the different academics are not independent; they depend on the conference location. We consider that all location variables (conference location or person locations) have the same domain and are hierarchical variables. The Bayesian network representation of this domain is shown in Figure 4.14 (a). The part of the class hierarchy for the location is shown in Figure 4.14 (b). Here we have $2k + 1$ hierarchical variables. The hierarchical variable $\text{conf}_\text{loc}$ denotes the conference location in the world. The hierarchical variables $\text{acam}_\text{loc}_1, \ldots, \text{acam}_\text{loc}_k$ denote the locations of the academics, and $\text{spou}_\text{loc}_1, \ldots, \text{spou}_\text{loc}_k$ denote the locations of their spouses.

We can specify the prior probability $P(\text{conf}_\text{loc})$ by specifying the distribu-
tion of each class in the class hierarchy of $\text{conf}_\text{loc}$ over its immediate subclasses. To specify the conditional probability $P(\text{acam}_\text{loc}_k|\text{conf}_\text{loc})$, we need to specify the default distribution of each class in the hierarchy of $\text{acam}_\text{loc}_k$ over its immediate subclasses for some parent contexts (for some classes in the hierarchy of $\text{conf}_\text{loc}$). We assume that the conference location influences each academic's location very locally; we need to specify the default distribution of the classes over their immediate subclasses for very few parent contexts.

**Example 4.10.1** Let us consider the distribution of class $uk$ in the hierarchy of academic location over its immediate subclasses. That is, we need to specify the conditional probabilities such as $P(\text{scotland}|uk \land \text{conf}_\text{loc})$. This means we know that the academic is in the UK but do not know whether he is in Scotland, Wales or England.

Now, knowing that the conference is taking place in the UK but not where in the UK does not provide us with any information about the location of the academic in the UK. However, if we know that the conference is taking place somewhere in Scotland, we can state that there is a high probability that the academic is in Scotland. That is, the conference location influences the academics location in the UK if the conference is taking place in the UK and we know where in the UK, otherwise it does not influence the location of the academic in the UK. We assume that knowing the conference location in Scotland (e.g., in Edinburgh), does not provide us with any extra information about where they may be if not in Scotland. That is, the distribution of class $uk$ in the hierarchy of $\text{acam}_\text{loc}_k$ over its immediate subclasses conditioned on $\text{conf}_\text{loc}$ needs to be defined for four parent contexts:
(world), (scotland), (england), and (wales).

We could have a default probability distribution of the class uk in the hierarchy of acam_loc.k over its immediate subclasses conditioned on conf_loc for parent context $X = \text{(scotland)}$ such as:

\[
P_d(\text{scotland}\mid\text{uk} \land \text{conf\_loc} = \text{scotland}) = 0.8
\]

\[
P_d(\text{england}\mid\text{uk} \land \text{conf\_loc} = \text{scotland}) = 0.1
\]

\[
P_d(\text{wales}\mid\text{uk} \land \text{conf\_loc} = \text{scotland}) = 0.1
\]

Similarly, for parent context $X = \text{(wales)}$ or $X = \text{(england)}$ such as:

\[
P_d(\text{scotland}\mid\text{uk} \land \text{conf\_loc} = \text{wales}) = 0.1
\]

\[
P_d(\text{england}\mid\text{uk} \land \text{conf\_loc} = \text{wales}) = 0.1
\]

\[
P_d(\text{wales}\mid\text{uk} \land \text{conf\_loc} = \text{wales}) = 0.8
\]

For parent context $X = \text{(world)}$ such as:

\[
P_d(\text{scotland}\mid\text{uk} \land \text{conf\_loc} = \text{world}) = 0.3
\]

\[
P_d(\text{england}\mid\text{uk} \land \text{conf\_loc} = \text{world}) = 0.4
\]

\[
P_d(\text{wales}\mid\text{uk} \land \text{conf\_loc} = \text{world}) = 0.3
\]

4.10.2 Results

To get an idea of the performance of Abstract.Bayes + VE compared to VE, we investigate how the performance of both algorithms varies with the number of hierarchical variables in the Bayesian network, with the depth of the tree hierarchies of
the hierarchical variables, and with the depth of the exceptional classes. We represent the abstraction hierarchy of all location variables by a binary tree of depth $d$, where $d$ is a parameter.

The probability distribution of any class $a$ in the hierarchy of academic acam_loc_k need to be specified for three parent contexts $c_0$, $c_{11}$, and $c_{12}$, where $c_0$ is the root class in the hierarchy of conf_loc, and $c_{11}$ and $c_{12}$ are the children of class $c$ in the hierarchy of conf_loc, where class $c$ corresponds to class $a$ in the hierarchy of acam_loc_k. We define the default distribution of class $a$ as follows:

$$
\begin{align*}
P_d(a_{11}|a \land \text{conf}_\text{loc} = c_0) &= 0.5 \\
P_d(a_{12}|a \land \text{conf}_\text{loc} = c_0) &= 0.5 \\
P_d(a_{11}|a \land \text{conf}_\text{loc} = c_{11}) &= 0.8 \\
P_d(a_{12}|a \land \text{conf}_\text{loc} = c_{11}) &= 0.2 \\
P_d(a_{11}|a \land \text{conf}_\text{loc} = c_{12}) &= 0.2 \\
P_d(a_{12}|a \land \text{conf}_\text{loc} = c_{12}) &= 0.8
\end{align*}
$$

In the same way, we define the distribution of classes in the hierarchy of spou_loc_k conditioned on acam_loc_k.

For the conference location variable, conf_loc, for each class $c$ in its hierarchy, we consider a uniform distribution over its children. Suppose $c_{11}$ and $c_{12}$ are the children of $c$. Then,

$$
\begin{align*}
P_d(c_{11}|c) &= 0.5 \\
P_d(c_{12}|c) &= 0.5
\end{align*}
$$
Suppose $k$ denotes the number of academics, $d$ denotes the depth of location hierarchies, and $d_{obs}$ denotes the depth of observation. For the experiments, we consider computing the probability of a class in the class hierarchy of conf_loc. We consider spou_loc_1, \ldots, spou_loc_k as the observed variables. We do so because we want some hierarchical variables to sum out. If the academic locations were observed, there are no hierarchical variables to sum out.

For the experiments, we consider computing the probability of a class that is at level $d = 1$ in the class hierarchy of conf_loc, given the evidence, by applying only $VE$ on the flattened network and $Abstract.Bayes + VE$. To determine how the inference time depends on the number of hierarchical variables, we vary $k$ from 1 to 8. To determine how inference time depends on the depth of the hierarchy ($d$), we vary $d$ from 1 to 10 with a step of 1. To determine how the inference time depends on the depth of the exceptional classes, for each value of $d$ we vary the depth of the observation ($d_{obs}$) for the observed variables from 1 to $d$. We consider only the positive observations about the hierarchical variables. For all hierarchical variables we keep the observations at the same level, but we randomly pick the observed classes at that level.

Figure 4.15 shows the average inference time of both $Abstract.Bayes + VE$ and $VE$ for the belief network that has only two academics as a function of the depth of the class hierarchies, as well as the depth of the observations. The error bars on $Abstract.Bayes + VE$, ($d_{obs} = d$) curve represent the standard deviations obtained from randomly picking the positions of the positive observation classes.

Figure 4.16 shows the average inference time of both $Abstract.Bayes + VE$
Figure 4.15: The average inference time of applying $VE$ and $Abstract Bayes + VE$ for $k = 2$ as a function of the depth of the hierarchy, and depth of the observations.
Figure 4.16: The average inference time of applying VE and Abstract Bayes + VE for $d = 8$ as a function of the number of academics and the depth of the observations.
and VE as a function of the number of academics and the depth of observations; here we consider the depth of the hierarchy is 8 \((d = 8)\). The error bars on Abstract_Bayes + VE, \((d_{obs} = 8)\) curve again represent the standard deviations obtained from by randomly picking the location of the positive observation classes at level 8 in the hierarchies of the spouse variables.

The error (standard deviation) arises because every time different observed classes are selected for the spouse variables. Sometimes the observed classes are very apart and make different exceptional classes in the hierarchies of the academics variables. If academics have different exceptional classes, they make different distinctions (different exceptional classes) in the hierarchy of conf_loc. If there are many exceptional classes in the hierarchy of conf_loc, the number of abstract values\(^5\) for variable conf_loc is also more that increases the processing time.

Figure 4.15 shows that the inference time for Abstract_Bayes+VE does not depend on the depth of the class hierarchy but increases as the depth of the observations increases. This increase is because the depth of the observations for spou_loc_1, \ldots, spou_loc_k variables increases the depth and the number of exceptional classes for acam_loc_1, \ldots, acam_loc_k and conf_loc variables. Notice that VE takes less time as the depth of observation increases. This is because the domain sizes of the spou_loc_1, \ldots, spou_loc_k variables reduce as the depth of the observation increases; after incorporating the observations in the flattened network the domain of each spou_loc_i variable is the disjunction of all leaf values that are descendants of the observed class (i.e., \(2^{(d-d_{obs})}\)^\(^6\).

\(^5\)The number of abstract values is proportional to the number of exceptional classes.

\(^6\)In a binary tree of depth \(d\), the number of values represented by a node that is at depth \(n\) from root is \(2^{d-n}\).
The results shown in Figures 4.15 and 4.16 give us some indication of the overhead involved in constructing an abstract Bayesian network and cases in which it may be more effective to use it rather than applying VE on a flattened network.

4.11 Conclusion

In this chapter, we argue that considering hierarchical structure on the values of a variable can make reasoning more efficient than reasoning over the set of all values. We show how, given a query and observations, we can dynamically construct an abstract Bayesian network from the given hierarchical Bayesian network that can be used in any standard probabilistic inference algorithm. The domain size of the variable in an abstract Bayesian network is independent of the size of the hierarchies; it depends on how many of the classes in the hierarchy are exceptional with respect to the children that are observed or have observed descendants in a Bayesian network. The running time of Abstract Bayes to abstract a hierarchical variable depends on the depth of the classes that are exceptional with respect to children that are observed or have observed descendants, as well as the depth of the query in the hierarchy, and is otherwise independent of the size of the hierarchy. Thus, it is possible to have hierarchical variables with unbounded or infinite domains. For example, with a spatial hierarchy, as long as we have a way to compute the probability distribution over subclasses, there is no reason not to have a hierarchy that is infinitely detailed. It is only when we ask a query of a class or make observations that we need to restrict the size of the hierarchy.
Chapter 5

Putting It All Together

5.1 Introduction

In Chapters 3 and 4, we consider the problem of performing inference in Bayesian networks that have variables with large domains. In Chapter 3, for handling the large number of values of the variables efficiently we consider that the conditional probabilities can be represented in a compact manner using both intensional and extensional definitions. We represent the conditional probabilities in CPD language form. We present an inference algorithm, Large Domain VE, that dynamically partitions the large number of values of the variables in equivalence classes, given the observation and query.

In Chapter 4, to handle the large number of values efficiently we consider the case where there is a priori hierarchical structure on the values of the variables, and called such variables hierarchical variables. To represent the CPDs compactly, we use inheritance. We present an approach for reasoning in Bayesian networks with
hierarchically structured variables that dynamically constructs an *abstract Bayesian network*, given some evidence and a query, by collapsing the hierarchies to include only those values necessary to answer the query. The query can be answered from the abstract Bayesian network using any standard probabilistic inference algorithm, such as variable elimination or stochastic simulation.

Given the close relationship\(^1\) between these two approaches, in this chapter we examine how they can be put together in a common framework; that is, how to perform inference in Bayesian networks that exploit both kinds of structures. To exploit both kinds of structures, one obvious setting is: consider a Bayesian network that contains both hierarchical and large-domain variables, and large-domain variables are represented using intensional and extensional definitions.

Let us consider another more general setting: consider a hierarchical variable such that for some of its children the hierarchical structure on its values does not need to be defined completely. That is, some children depend on each value represented by some of the nodes (classes) in the hierarchy, which means we need to consider the large number of values represented by some of the classes in the hierarchy. These large number of values can be represented efficiently using intensional definitions.

For example, consider the problem of identifying rock types in a region. There are a large number of rock types. The British Geological Survey provides a rock classification scheme\(^2\) that classifies rocks in a hierarchical manner. At the very high level, all types of rocks are grouped together as "allrocks" and then par-

---

\(^1\)Essentially, both of these approaches partition the large number of values of the variables into equivalence classes.

\(^2\)http://www.bgs.ac.uk/bgsres/
tioned into four groups: "igneous rocks", "metamorphic rocks", "sediments and sedimentary rocks", and "artificial man-made ground and natural superficial deposits". The rocks in each group are further partitioned. The types of rocks for some locations could be known to us because we have some prior knowledge about these locations. For locations that we know nothing about, we can find out the rock type by knowing the geological type of these locations, which can be determined from a geological map. The properties acidity and grain size of the rocks depend on rock type.

We can model this rock-identifying problem using the Bayesian network shown in Figure 5.1 (a). The variable Location is a hierarchical variable and represents all of the locations in Western Canada. The tree hierarchies of Location is shown in Figure 5.1 (b). At the very high level, all of the (X,Y) locations represented by Location are grouped together as western canada. Suppose all of the locations in Western Canada can be partitioned into two disjoint classes, bc and alberta. The values represented by the classes bc and alberta are further partitioned, and so on. The variable Legend is a simple variable representing the geological type of a location. The variable RockType is a hierarchical variable and represents the rock type of a location. The tree hierarchy of RockType is shown in Figure 5.1 (b).

The hierarchical structure on the values of Location is exploited by its child RockType but not by its other child Legend. This is because to find the geological type of a location we need to consult maps, which means we need to use the (X,Y) value of a location. We can use BC maps to find the geological type of any location in BC and Alberta maps to find the geological type of any location in
Figure 5.1: (a) A Bayesian network modelling rock types in a region. (b) The class hierarchies of Location and RockType.
Alberta. The conditional probability of Legend, \( P(\text{Legend} | \text{Location}) \) can be represented compactly, using the intensional definitions (using predicates and functions), which may involve looking up the locations in the BC map or Alberta map, in CPD language form.

We term those hierarchical variables whose classes can be represented extensionally or intensionally (using predicates and functions) generalised-hierarchical variables. Note that both large-domain and hierarchical variables are instances of generalised-hierarchical variables.

In this chapter, we consider the problem of performing inference in Bayesian networks that have generalised-hierarchical variables. We begin in Section 5.2 by defining the generalised-hierarchical variables. In Section 5.3, we discuss how to represent the CPDs of Bayesian networks with generalised-hierarchical variables in a compact manner using the representations that we developed in Chapters 3 and 4. In Section 5.4, we show how to perform inference in Bayesian networks with generalised-hierarchical variables using the inference algorithms that we developed in Chapters 3 and 4.

5.2 Generalised Hierarchical Variables

In Chapter 4, we introduced the notion of hierarchical variables. There we considered that the large number of values of a variable can be organized \textit{a priori} in a class hierarchy form, where the large number of values are grouped together in a meaningful way. However, in some problems for some children, the large number of values need to be grouped together up to some level (depth) in the class hierar-
chy, after that they need not be grouped together. That is, for some children, some internal nodes in the class hierarchy are considered as leaf-nodes. The large number of values of these nodes (or classes) are represented using intensional definitions. We term such hierarchical variables *generalised-hierarchical* variables.

**Definition 5.2.1** A *generalised-hierarchical variable* is a hierarchical variable with classes that can be represented extensionally or intensionally.

A large-domain variable can also be seen as a generalised-hierarchical variable with only one class (root class) represented intensionally. Thus, both hierarchical and large-domain variables are instances of generalised-hierarchical variables.

For example, consider the hierarchical variable *Location* shown in Figure 5.1 (a). The tree hierarchy of *Location* is shown in Figure 5.1 (b). At the very high level, all of the values of *Location* are grouped together as *western canada*. Then, *western canada* is partitioned into two disjoint classes, *bc* and *alberta*, and so on. As discussed before, *Location*’s one child *Legend* depends on each value represented by the classes *bc* and *alberta*. However, *Location*’s other child, *RockType*, exploits hierarchical structure on *Location*’s values. In this case, to define the conditional probability of *Legend* we need to consider all of the values represented by classes *bc* and *alberta*, i.e., we may need to use intensional definitions. However, to define the conditional probability of *RockType*, we do not consider the values represented by the classes in the hierarchy of *Location*. Thus, *Location* is a generalised-hierarchical variable.
5.3 Representation

In this section, we discuss how to represent the conditional probability distributions of Bayesian networks with generalised-hierarchical variables. There are two main issues that need to be considered:

C1: specifying the conditional probability for generalised-hierarchical variables

C2: specifying how variables are conditioned on generalised-hierarchical variables

The simplest case of C1 is how to specify the prior probability distribution for a generalised-hierarchical variable. The simplest case of C2 is how to specify the conditional probability of a simple variable conditioned on a generalised-hierarchical variable. The more complicated cases can be built from these two cases. First, we discuss how to represent the probability distributions for these two cases. Next, we present the general case using the representations developed for these two cases.

5.3.1 Specifying the Prior Probability Distribution for a Generalised Hierarchical Variable

In Chapter 4, we discussed a top-down approach for defining the prior probability distribution of a hierarchical variable. In the top-down approach, we consider that each class in the hierarchy defines a probability distribution over its children. To represent the prior probability distribution of the generalised-hierarchical variables, we can use the same approach. Note that for classes that have only few children, we
can define the probability distribution over their children extensionally (by listing the values). For the leaf-classes that represent a large number of values, we can define the probability distribution using the intensional definitions. We can define the distribution of a class over its children extensionally or intensionally using the CPD language representation as developed in Chapter 3.

For example, suppose we want to specify the prior probability distribution over the generalised-hierarchical variable *Location* as discussed in the previous section. Note that the domain of *Location* is all \((X, Y)\) locations in Western Canada. Suppose the class *western canada* has only two children. We specify the probability distribution of *western canada* over its immediate subclasses:

\[
P(bc| \text{western canada}) = 0.6
\]
\[
P(\text{alberta}| \text{western canada}) = 0.4
\]

We can specify the probability distribution over the immediate subclasses of *bc*. Suppose we have as part of this distribution:

\[
P(\text{north bc}|bc) = 0.4
\]
\[
P(\text{south bc}|bc) = 0.3
\]

Suppose the distribution of classes *north bc* and *south bc* over their values need to be represented intensionally. Suppose both *north bc* and *south bc* define a uniform distribution over their values. Then we can represent the distribution of *north bc* and *south bc* over their values as follows:

\[
P(\text{Location}|\text{Location} \in \text{north bc}) = \text{uniform}(\text{"north bc"})
\]
\[
P(\text{Location}|\text{Location} \in \text{south bc}) = \text{uniform}(\text{"south bc"})
\]
The function \textit{uniform}("north bc") defines an uniform distribution over all the locations in northern BC. Notice that how the representations that we developed in Chapters 3 and 4 can be combined to represent the prior probability distribution of a generalised-hierarchical variable.

\subsection*{5.3.2 Conditional Probability of a Simple Variable Conditioned on a Generalised-Hierarchical Variable}

Suppose that \( H \) is a generalised-hierarchical variable, \( F \) is a simple variable, and \( H \) is the only parent of \( F \). To represent the conditional probability of \( F \) in a compact manner, we use the approach discussed in Section 4.5.2. The conditional probability distribution \( P(F|H) \) is specified by specifying the default conditional probability distribution over \( F \) for some classes \( C_j \) in the hierarchy of \( H \). We have two cases:

- If the distribution over \( F \) does not depend on the set of values represented by \( C_j \), we define \( P_d(F|C_j) \).

- Otherwise, we specify \( P_d(F|H \land H \in C_j) \).

The default distribution \( P_d(F|H \land H \in C_k) \) can be inherited by the subclasses of \( C_k \) in the same way as \( P_d(F|C_j) \) is inherited by the subclasses of \( C_j \) as discussed in Section 4.5.2. The default distributions \( P_d \)'s can be specified extensionally or intensionally using the CPD language. The conditional probability of \( F \) for any class \( C_k \) of \( H \), \( P(F|C_k) \) (or \( P(F|H \land H \in C_k) \)), can be computed using inheritance, as discussed in Section 4.5.2.

\textbf{Example 5.3.1} Consider the problem of representing the conditional probability of
simple variable Legend conditioned on Location in the Bayesian network shown in Figure 5.1. Suppose Legend has four values: \{“mountain”, “hill”, “water”, “plain”\}; Legend is a deterministic function of Location that is obtained by consulting the appropriate map. To represent \(P(Legend|Location)\) we could have:

\[
P_d(Legend = v|Location \land Location \in bc) = \begin{cases} 1 & \text{if } (\text{consult.bc.map}(Location) = v) \\ 0 & \text{else} \end{cases}
\]

\[
P_d(Legend = v|Location \land Location \in alberta) = \begin{cases} 1 & \text{if } (\text{consult.alberta.map}(Location) = v) \\ 0 & \text{else} \end{cases}
\]

The function \text{consult.bc.map} returns the legend value of a location that is in British Columbia. We assume here that we have procedures that compute the functions \text{consult.alberta.map}(Location) and \text{consult.bc.map}(Location).

### 5.3.3 Arbitrary Probability Distribution

The general case is when a random variable (generalised-hierarchical, hierarchical or simple) can have any number and any kind of variables as its parents. Let \(H\) be a generalised hierarchical variable. Suppose \(Pa^h(H)\) denotes \(H\)'s hierarchical (and generalised-hierarchical) parents and \(Pa^s(H)\) denotes its simple parents. To represent the conditional probability of \(H\) conditioned on its parents, we can assume the following:

- Each non-leaf (internal) class in the hierarchy of \(H\) has a probability distribution over its immediate subclasses that is conditioned on (some of) the parents.
of \(H\). Let \(F\) be a variable corresponding to an internal class \(C_k\) in the hierarchy of \(H\). The values of \(F\) are the immediate subclasses of \(C_k\). For each \(F\) we specify the conditional probability \(P(F|\text{Pa }^h(H) \land \text{Pa }^a(H))\).

- Each leaf-class that represents a large number of values has a distribution over its values that is conditioned on (some of) the parents of \(H\), which is specified intensionally using the CPD language. Let \(C_l\) be a leaf class that represents a large number of values. For each \(C_l\) we specify the conditional probability \(P(H|H \in C_l \land \text{Pa }^h(H) \land \text{Pa }^a(H))\).

The conditional probability of \(F\) given its parents, \(P(F|\text{Pa }^h(H) \land \text{Pa }^a(H))\) (or \(P(H|H \in C_k \land \text{Pa }^h(H) \land \text{Pa }^a(H))\)), is specified by specifying the default probability distribution over \(F\), \(P_d(F|\text{Pa }^a(H) \land \text{Pa }^h(H) = \mathcal{X})\) using the CPD language representation, for some parent contexts \(\mathcal{X}\) as discussed in Section 4.5.3. The parent contexts \(\mathcal{X}\) are such that there is at least one default parent context along every path from the root in \(\text{tree}(H_1) \times \ldots \times \text{tree}(H_n)\), where \(H_1, \ldots, H_n\) are the generalised-hierarchical parents of \(F\), and \(\text{tree}(H_i)\) denotes the tree hierarchy of \(H_i\).

Note that the problem of multiple inheritance can arise with generalised-hierarchical variables, as discussed in Section 4.5.3; we explicitly disallow this to occur. We are thus forcing the user to disambiguate the cases in which there would otherwise be a problem of multiple inheritance.

The conditional probability of \(F\) (or \((H|H \in C_l)\)) for a parent context \(\mathcal{X}\) can be computed in the same way we computed it for hierarchical variables in Section 4.5.3.
In this section, we discuss how to represent the conditional probability distributions for Bayesian networks with generalised-hierarchical variables. In the next section, we discuss how to perform inference in Bayesian networks with generalised-hierarchical variables using the inference algorithms that we developed in Chapters 3 and 4.

5.4 Inference in Bayesian networks with Generalised-Hierarchical Variables

Similar to a hierarchical variable, observation about a generalised-hierarchical variable can also be positive and negative and involves classes from its tree hierarchy. Observation about a generalised-hierarchical variable can be interpreted in the same way we interpreted for hierarchical variables in Section 4.6. Let \( B \) be a Bayesian network with generalised-hierarchical variables. The inference task is to compute \( P(Q|e) \), where \( Q \) is a simple variable or a class in the hierarchy of hierarchical (generalised-hierarchical) variable and \( e \) is the allowable observations, as defined in Section 4.6.

For performing inference in Bayesian networks with generalised-hierarchical variables, we can use the same approach that we used with hierarchical variables. That is, given some evidence and a query, construct an abstract Bayesian network by replacing the generalised-hierarchical variables with large-domain variables whose values are partitioned into a minimum number of blocks to answer the query. We can then answer the query from the abstract Bayesian network. Note that
hierarchical variables are replaced by simple variables.

The abstract Bayesian network, constructed from the Bayesian network with generalised-hierarchical variables, involves variables with large domains; its conditional probabilities in CPD language form. We can use the Large Domain VE algorithm developed in Chapter 3 for making inference in the constructed abstract Bayesian network with large-domain variables. The algorithm for constructing an abstract Bayesian network from a Bayesian network with generalised-hierarchical variables is essentially the same as the algorithm Abstract_Bayes, except for the following step:

**Abstraction of generalised-hierarchical variables:** Let $H$ be a generalised-hierarchical variable that needs to be abstracted. Let $V_{ex}$ be the set of exceptional classes of $H$ with respect to its relevant children. Suppose $H^a$ denotes the abstraction of $H$, a large-domain variable. The domain of $H^a$ is partitioned into disjoint subsets (blocks). During the abstraction of $H$, we compute the blocks for $H^a$. We compute the set of values (a block) $v$ for $H^a$, for each class $C_k \in V_{ex}$ as follows:

- $v = V_{C_k}$, if $C_k$ does not have any exceptional strict subclasses. The set $V_{C_k}$ denotes the set of values represented by class $C_k$.

- Otherwise, $v = V_{C_k} - V_{C_1} - \ldots - V_{C_m}$, where $C_1, \ldots, C_m$ are the highest strict subclasses of $C_k$ that are in $V_{ex}$. The set $v$ contains all the values that are in $C_k$ and are not covered by other blocks.

The sets that are empty can be removed from the partition of $H^a$ and $H$ is replaced by $H^a$. The conditional probabilities for the abstract Bayesian network are
constructed as discussed in Section 4.8.1 for hierarchical variables.

**Example 5.4.1** Consider constructing an abstract Bayesian network from the Bayes network with generalised-hierarchical variables shown in Figure 5.1 (a). The variable *Location* is a generalised-hierarchical variable. The class hierarchies of *Location* and *RockType* are shown in Figure 5.1 (b). Suppose we have observed *RockType* ∈ *crystalline igneous rock*. The conditional probability $P(\text{Legend}|\text{Location})$ is specified using the intensional definitions discussed in Example 5.3.2. The query is: $P(\text{Location} \in \text{south bc}|\text{RockType} \in \text{crystalline igneous rock})$.

The conditional probability $P(\text{RockType}|\text{Legend} \land \text{Location})$ is specified by specifying the default distribution of each class in the hierarchy of *RockType* over their immediate subclasses for some of the classes in the hierarchy of *Location*. The default distribution of each class in the hierarchy of *RockType* is specified for the classes *bc* and *alberta* in the hierarchy of *Location*. The default distribution of each class in the hierarchy of *RockType* for the contexts *bc* and *alberta* depends on the values of *Legend*. Suppose for the context *bc*, *Legend* = “hill” the class *allrocks* defines the following distribution:

**context : (bc)**

\[
\begin{align*}
P_d(\text{metamorphic rock}| \text{allrocks} \land \text{bc} \land \text{Legend} = \text{“hill”}) &= 0.4 \\
P_d(\text{igneous rock}| \text{allrocks} \land \text{bc} \land \text{Legend} = \text{“hill”}) &= 0.3 \\
P_d(\text{sedimentary rock}| \text{allrocks} \land \text{bc} \land \text{Legend} = \text{“hill”}) &= 0.2 \\
P_d(\text{deposits}| \text{allrocks} \land \text{bc} \land \text{Legend} = \text{“hill”}) &= 0.1
\end{align*}
\]

The class *east squamish* in the hierarchy of *Location* affects the distribution of class *allrocks* in the hierarchy of *RockType*. The distribution of class *allrocks* for
the context east squamish does not depend on the values of Legend (we have some prior knowledge about east squamish). For the context east squamish the class allrocks defines the following distribution:

context : (east squamish)

\[ P_d(\text{metamorphic rock} | \text{allrocks} \land \text{east squamish}) = 0.2 \]
\[ P_d(\text{igneous rock} | \text{allrocks} \land \text{east squamish}) = 0.6 \]
\[ P_d(\text{sedimentary rock} | \text{allrocks} \land \text{east squamish}) = 0.1 \]
\[ P_d(\text{deposits} | \text{allrocks} \land \text{east squamish}) = 0.1 \]

Construction of an abstract Bayesian network: To construct an abstract Bayesian network from the Bayesian network shown in Figure 5.1 (a), we traverse the network from the leaves upwards. The simple variables Acidity and GrainSize are pruned because they are irrelevant. We abstract the hierarchical and generalised-hierarchical variables RockType and Location.

Abstraction of RockType (observed hierarchical variable): We have observed RockType $\in$ crystalline igneous rock. The variable RockType does not have any children, so RockType is abstracted by a simple variable RockType$^a$ with domain:

\[ \text{Val(RockType}^a) = \{"igneous rock type", "allrocks - igneous rock type"\} \]

After abstracting RockType we abstract Location.

Abstraction of Location: To abstract Location, we find those classes of Location that are exceptional for Legend and RockType. The variable Location has four exceptional classes: east squamish, south bc, bc, and alberta. We abstract Location
by large-domain variable Location\textsuperscript{a}. The values of Location\textsuperscript{a} are partitioned into four disjoint subsets:

\[ \text{Val(Location}^a) = \{V_{C_1}, V_{C_2}, V_{C_3}, V_{C_4}\} \]

where \( V_{C_1} \) is a set of values represented by class \textit{east squamish}, \( V_{C_2} \) is a set of values represented by class difference: \textit{(south bc – east squamish)}, \( V_{C_3} \) is a set of values represented by class difference: \textit{(bc – south bc)}, and \( V_{C_4} \) is a set of values represented by class \textit{alberta}.

After abstracting the hierarchical variables the algorithm compute the conditional probabilities for the abstract Bayesian network. The first case we consider is the prior probability of Location\textsuperscript{a}.

**Construction of \( P(\text{Location}^a) \):** The prior probability \( P(\text{Location}^a) \) is computed as follows:

\[
P(\text{Location}^a \in V_{C_1}) = P(\text{east squamish} | \text{south bc}) \times P(\text{south bc} | \text{bc}) \\
\times P(\text{bc} | \text{western canada})
\]

\[
P(\text{Location}^a \in V_{C_2}) = P(\text{south bc} | \text{bc}) \times P(\text{bc} | \text{western canada}) \\
- P(\text{Location}^a \in V_{C_1})
\]

\[
P(\text{Location}^a \in V_{C_3}) = P(\text{bc} | \text{western canada}) - P(\text{Location}^a \in V_{C_2})
\]

\[
P(\text{Location}^a \in V_{C_4}) = P(\text{alberta} | \text{western canada})
\]

Note that \( P(\text{Location}^a \in V_{C_i}) \) represents the probability mass of all the values of Location\textsuperscript{a} that are in set \( V_{C_i} \).
Construction of \( P(Legend|Location^a) \): To compute the conditional probability \( P(Legend|Location^a) \), the algorithm compute \( P(Legend|Location^a \in V) \) for each block \( V \) of \( Location^a \) using inheritance.

The conditional probability \( P(Legend|Location^a) \) is shown below:

\[
P(Legend = v|Location^a \in V_{C_1}) = \begin{cases} 1 & \text{if} \ (consult.bc.map(Location^a) = v) \\ 0 & \text{else} \end{cases}
\]

\[
P(Legend = v|Location^a \in V_{C_2}) = \begin{cases} 1 & \text{if} \ (consult.bc.map(Location^a) = v) \\ 0 & \text{else} \end{cases}
\]

\[
P(Legend = v|Location^a \in V_{C_3}) = \begin{cases} 1 & \text{if} \ (consult.bc.map(Location^a) = v) \\ 0 & \text{else} \end{cases}
\]

\[
P(Legend = v|Location^a \in V_{C_4}) = \begin{cases} 1 & \text{if} \ (consult.alberta.map(Location^a) = v) \\ 0 & \text{else} \end{cases}
\]

Construction of \( P(RockType^a|Location^a \land Legend) \): To compute the conditional probability \( P(RockType^a|Location^a \land Legend) \), the algorithm compute \( P(RockType^a = v_a|Location^a \in V^a \land Legend) \) for each value \( v_a \) of \( RockType^a \), for each block \( V^a \) of \( Location^a \). The conditional probability \( P(RockType^a|Location^a \land Legend) \) is shown below:

\[
P(RockType^a = \text{"crystalline igneous rock"}|Legend = V \land Location^a \in V_{C_1}) = P(\text{crystalline igneous rock}|igneous rock \land bc \land Legend = V) \times P(\text{igneous rock}|allrocks \land east squamish)
\]

\[
P(RockType^a = \text{"crystalline igneous rock"}|Legend = V \land Location^a \in V_{C_2})
\]
Note that variable $\text{RockType}^a$ is an observed variable, with assignment $\text{RockType}^a = \text{"crystalline igneous rock"}$.

After constructing the abstract Bayesian network we can answer the query using Large Domain VE algorithm to sum out all non-observed, non-query variables from the abstract Bayesian network. The abstraction is done as part of the standard elimination of irrelevant variables.

### 5.5 Conclusion

In this chapter, we discuss how the structures that we considered in Chapters 3 and 4 can be put together in a common framework that can be applied to a more
general class of problems. We extend the definition of hierarchical variables by considering that classes can be represented extensionally or intensionally. We show here that the representations that we have developed in Chapters 3 and 4 can be combined together to represent the CPDs of Bayesian networks with generalised-hierarchical variables. To perform inference in Bayesian networks that exploit both kinds of structures we can first construct an abstract Bayesian network that may have variables with large domains, and then use Large Domain VE to answer the query from the abstract Bayesian network.
Chapter 6

Conclusions

6.1 Summary

In this thesis, we have presented two approaches for probabilistic inference in complex systems that have discrete random variables with very large or infinite domains\(^1\). To perform efficient inference in Bayesian networks that have discrete variables with large domains, the structure we exploit is the ability to group the values. Rather than reasoning at the level of individual values, the structured representation allows us to partition the values into subsets and the values in these subsets are treated as if they were a single value.

We first consider the case where the conditional probabilities can be represented in a compact manner using both intensional (in terms of functions and predicates) and extensional (by listing the values) definitions. To represent such conditional probabilities, we developed a CPD language. To deal with the com-

\(^1\)The complexity of exact probabilistic inference in a Bayesian network is exponential in tree width, where the base of the exponent is the domain size.
plexity of inference, we exploit the fact when there is no evidence or query to dis­tinct between all the values of a variable, we do not need to reason about each value separately. Rather, we can reason about a group of values together as a single entity. We developed a structured inference algorithm, Large Domain VE, for mak­ing inference in Bayesian networks that have discrete variables with large domain. In this case the partitions are inferred at run time and depend on what is observed and what is queried.

We describe a motivating example application: person identification problem. We present the application of Large Domain VE to the person identification problem. Large Domain VE dynamically partitions the domains of large-domain variables for efficient inference. The point is that there is a natural Bayesian network representation for the person identification problem for which inference was not possible before.

In Chapter 4, we consider the case where there is a priori hierarchical structure on the values of the variables. That is, the large number of values of the variables can be represented as tree hierarchies. We call such variables hierarchically structured variables. To exploit the structure provided by tree hierarchies in probabilistic reasoning we developed a representation language for representing the conditional probability distributions in a compact manner. We represent the distribution of the hierarchical variables by specifying, for each class, the probability distribution over its immediate subclasses. To represent the conditional probability distribution of any variable conditioned on a hierarchical variable we use inheritance.
To perform efficient inference in Bayesian networks that have hierarchically structured variables, we construct an abstract Bayesian network dynamically, given some evidence and a query, by collapsing the hierarchies to include only those values necessary to answer the query. We can answer the query from the abstract Bayesian network using any standard probabilistic inference algorithm. The domain size of the variables in the abstract Bayesian network is independent of the size of the hierarchies; it depends on how many of the classes in the hierarchies are supported directly by the evidence or relevant to the query. Thus, the proposed approach is applicable even when the hierarchy of the variable is conceptually infinite. For example, with a spatial hierarchy, as long as we have a way to compute the probability distribution over subclasses, there is no reason not to have a hierarchy that is infinitely detailed. It is only when we ask a query given observations that we need to restrict the size of the hierarchy.

We presented experimental results that show that hierarchically structured values can make the reasoning more efficient than reasoning over the set of all values.

Finally, we described a framework that combines both intensional definitions of the conditional probability distributions and hierarchically structured values together so that it can be applied to a more general class of problems.

6.2 Future Work

This thesis has created many new opportunities for future work. This section suggests some possible directions future research may take.
The proposed approach for person identification involves pairwise matching. This approach works in two cases: first, when given a pair of records determining if they refer to the same person; and second, given two databases of records, where each record corresponds to a different person, determining matching records between them. However, this approach is not directly applicable when given a database we need to find which records refer to same people. One way our method can be extended is to build on the work of Pasula et al. [2002].

In our work with hierarchically structured variables in Chapter 4, we consider that the values of a hierarchically structured variable are arranged in a tree hierarchy form. We consider that each class in a tree hierarchy has only one parent. In the future, we would like to consider that the values of the variables can be arranged in a lattice hierarchy form. That is, we want to allow a class to have multiple parents. In this case, we will need a different mechanism for defining the prior (and conditional) distribution of a hierarchical variable. We also have to deal with the problem of multiple inheritance, since a class has more than one parent. We need techniques for handling multiple inheritance.

Recently, Poole [2003] presented first order probabilistic inference algorithms based on variable elimination and unification for reasoning about multiple individuals on the first-order level. He exploits the fact that all the individuals that we do not have any fact about can be consider as a group. In this thesis we consider how to combine all those values of a variable that are the same, given the evidence and query, as a single abstract value. We did not consider here how to combine all those individuals that are the same, given the evidence and query, as a group. In
the future, we would like to put these two works together in a common framework where we have large number of individuals and variables with large domains.

Many real problems require spatial hierarchies for modelling large domains. For example, consider the problem of decision making in forestry: which trees should be cut down in BC (Canada) given preferences and uncertainty. In this case we can represent the forest area in a tree hierarchy form. To make the decision, we may not be able to model this spatial hierarchy by a hierarchically structured variable. This is because in this case as an observation we can have number of locations (trees in that location) in the hierarchy that are infected with pine beetle, i.e., we have more than one positive observation for a hierarchical variable. However, for a hierarchically structured variable more than one value cannot be true at the same time. The proposed approach for handling a hierarchy as a hierarchical variable is not applicable with hierarchies where more than value is true. One idea for handling such hierarchies is to consider each class in the hierarchy as a node of the Bayesian network. In the future, we would like to extend our work to handle such hierarchies.

Many practical systems that are based on Bayesian networks are used in environments where the evidence arrives incrementally rather than coming all at once and is interleaved with belief updating. In the future, we would like to extend our work with hierarchically structured variables to incorporate evidence that arrives incrementally. To perform inference in a Bayesian network with hierarchically structured variables, we construct an abstract Bayesian network, given some evidence and query, by pruning the irrelevant variables and abstracting the hierarchical variables. In the case of incremental evidence we may not be able to prune
such variables. This is because the future observations can make these variables relevant. One idea is to consider these irrelevant variables as unit value variables (variables with only one value) and lazily revise the domain of these variables based on the new evidence. Thus, each new evidence can be viewed as invalidating some of the previously computed abstractions and conditional probability tables of the abstract Bayesian network. In this framework we can potentially use the Junction Tree algorithm. We can build the structure of Junction tree offline and we can revise the domain of the variables and conditional probability tables online as new evidence arrives.

One reason exact inference methods are developed is as basis of approximate inference methods. For example, both Rao-Blackwellization [G.Casella and Robert, 1996] and Mini-Buckets [Dechter, 1997] are approximate methods and use exact inference. In future one could explore new approximate algorithms, based on the algorithm proposed in this thesis, for solving networks that are too big for exact inference.

We would like to apply our work with hierarchical variables on more realistic problems. During the course of this thesis we were unable to find any realistic networks. We notice that finding a real network is like the chicken-and-egg problem. There is no point in building a network, if there is no method to solve it. However, there are domains that involve hierarchies where people need to make decisions. For example, the medical and geological domains often involve big hierarchies. Using the tools proposed in this thesis, people are now able to solve probabilistic models that involves hierarchies and so may start to build them.
With the development of the semantic web, people are using ontologies to describe the data (e.g., OWL, RDF). Our work with hierarchically structured variables, where the domain of a variable can be represented in a tree hierarchy form (e.g., in a taxonomy as part of the ontology), shows some progress in the direction of mixing ontologies and probabilities. Finally, we would like to extend our work so that it can be applied to the problems that involve ontologies and uncertainties.
Index

abstract Bayesian network, 135
abstract value, 135
abstraction of hierarchical variable, 134
allowable observation, 127
barren variable, 29
big factor, 66
blocks, 63
class, 109
class hierarchy, 109
constrained treewidth, 89
contextually independent, 31
default parent contexts, 122
default probability distribution, 118
elimination ordering, 21
evidence factor, 85
evidenced leaf value, 84
exceptional classes, 118
flattened Bayesian network, 135
generalised-hierarchical variable, 172
hierarchical variable, 109
induced domain size, 89
induced treewidth, 89
negative observation, 126
parent context, 121
parent context hierarchy, 122
partition, 63
path, 33
positive observation, 126
proper superparent context, 121
relevant exceptional class, 135
superparent context, 121
treewidth, 89
Bibliography


SIGMOD-1997 Workshop on Research Issues on Data Mining and Knowledge Discovery, pp. 23–29.


