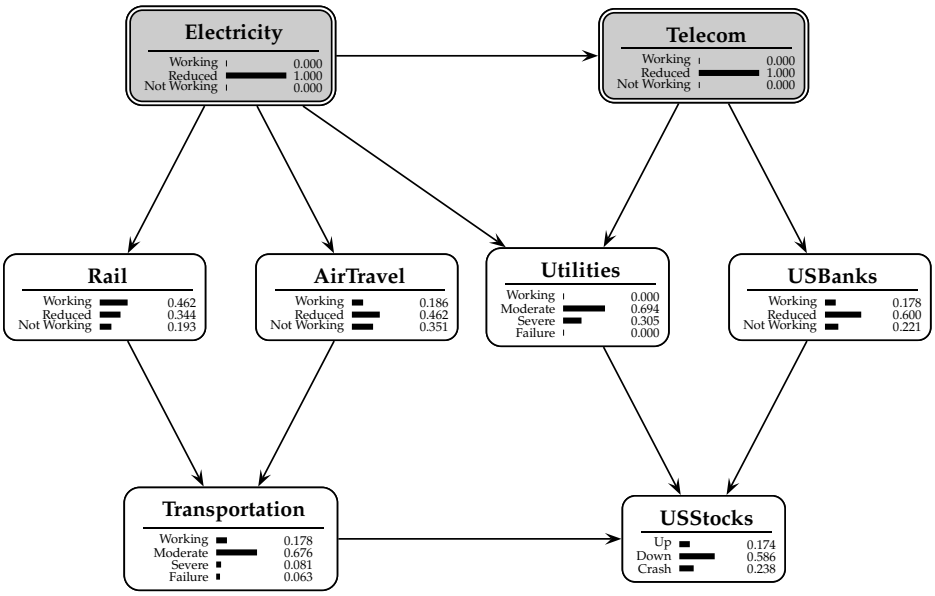
Bayesian networks

- a self-contained introduction with implementation remarks



Henrik Bengtsson <hb@maths.lth.se>
Mathematical Statistics
Centre for Mathematical Sciences
Lund Institute of Technology, Sweden

Abstract

This report covers the basic concepts and theory of Bayesian Networks, which are graphical models for reasoning under uncertainty. The graphical presentation makes them very intuitive and easy to understand, and almost any person, with only limited knowledge of Statistics, can for instance use them for decision analysis and planning. This is one of many reasons to why they are so interesting to study and use.

A Bayesian network can be thought of as a compact and convenient way to represent a joint probability function over a finite set of variables. It contains a qualitative part, which is a directed acyclic graph where the vertices represent the variables and the edges the probabilistic relationships between the variables, and a quantitative part, which is a set of conditional probability functions.

Before receiving new information (evidence), the Bayesian network represents our a priori belief about the system that it models. Observing the state of one of more variables, the Bayesian network can then be updated to represent our a posteriori belief about the system. This report shows a technique how to update the variables in a Bayesian network. The technique first compiles the model into a secondary structure called a junction tree representing joint distributions over non-disjoint sets of variables. The new evidence is inserted, and then a message passing technique updates the joint distributions and makes them consistent. Finally, using marginalization, the distributions for each variable can be calculated. The underlying theory for this method is also given.

All necessary algorithms for implementing a basic Bayesian network application are presented along with comments on how to represent Bayesian networks on a computer system. For validation of these algorithms a Bayesian network application in Java was implemented.

Keywords: Bayesian networks, belief networks, junction tree algorithm, probabilistic inference, probability propagation, reasoning under uncertainty.

Contents

Introduction

This Master's Thesis covers the basic concepts and theory of Bayesian networks along with an overview on how they can be designed and implemented on a computer system. The project also included an implementation of a software tool for representing Bayesian networks and doing inference on them. The tool is referred to as $\texttt{hb}^{\texttt{BN}}$.

In the expert system area the need to coordinate uncertain knowledge has become more and more important. Bayesian networks, also called Bayes' nets, belief networks or probability networks. Since they were first developed in the late 1970's [Pea97] Bayesian networks have during the late 1980's and all of the 1990's emerged to become a general representation scheme for uncertainty knowledge. Bayesian networks have been successfully used in for instance medical applications (diagnosis) and in operating systems (fault detection) [Jen96].

A Bayes net is a compact and convenient representation of a joint distribution over a finite set of random variables. It contains a *qualitative* part and a *quantitative* part. The *qualitative* part, which is a directed acyclic graph (DAG), describes the structure of the network. Each vertex in the graph represents a random variable and the directed edges represent (in some sense) informational or causal dependencies among the variables. The *quantitative* part describes the strength of these relations, using conditional probabilities.

When one or more random variables are observed, the new information propagates in the network and updates our belief about the non-observed variables. There are many propagation techniques developed [Pea97, Jen96]. In this report, the popular junction-tree propagation algorithm was used. The unique characters of this method are that it uses a secondary structure for making inference and it is also quite fast. The update of the Bayesian network, i.e. the update of our belief in which states the variables are in, is performed by an *inference engine* which has a set of algorithms that operates on the secondary structure.

Bayesian networks are not primarily designed for solving classification problems, but to explain the relationships between observations [Rip96]. In occasions where the decision patterns are complex BNs are good in explaining why something occurred, e.g. explaining which of the variables that did change in order to reach the current state of some other variable(s) [Rip96]. It is possible to learn the conditional probabilities, which describes the relation between the variables in the network, from data [RS98, Hec95]. Even the entire structure can be learned from data that is fully given or contains missing data values [Hec95, Pea97, RS97].

This report is written to be a self-contained introduction covering the theory of Bayesian networks, and also the basic operations for making inference when new observations are included. The majority of the algorithms are from [HD94]. The application developed is making use of all the algorithms and functions described in this report. All Bayesian-network figures found in this report are (automatically) generated by $\texttt{hb}^{\texttt{BN}}$. Also, some problems that will arise during the design and implementation phase are discussed and suggestions on how to overcome these problems are given.

Purpose

One of the research projects at the department concerns computational statistics and we felt that there was big potential for using Bayesian network. There are two main reasons for this project and report. Firstly, the project was designed to give an introduction into the field of Bayesian networks. Secondly, the resulting report should be a self-contained tutorial that can be used by others that have no or little experience in the field.

Method

Before the project started, neither my supervisor nor I was familiar with the concept of Bayesian networks. For this reason, it was hard for us to actually come up with a problem that was surely reasonable in size, time and difficulty and still wide enough to cover the main concepts of Bayes nets. Having a background in Computer Science, I though it would be a great idea to implement a simple Bayesian network application. This approach offers a deep insight in the subject and also some knowledge about the real-life problems that exist. After some literature studies and time estimations, we decided to use the development of an application as the main method for discovering the field of Bayesian networks. The design of the tool is object oriented and it is written in 100%-Java.

Outline of report

In chapter 1, two simple examples are given to show what Bayesian networks are about. This section also includes calculations showing how propagation of new information is performed.

In chapter 2, all graph theory needed to understand the Bayesian network structure and the algorithms are presented. Except for the introduction of some important notations the reader familiar with graph theory can skip this section.

In chapter 3, a graphical model called Markov network is defined. Markov networks are not as powerful as Bayesian networks, but because they carry Markov properties, the calculations are simple and straightforward. Along with defining Markov networks, the concept of conditional independence is defined. Markov networks are interesting since they carry the basics of the Bayesian networks and also because the secondary structure used to update the Bayes net can be seen as a multidimensional Markov tree, which is a special case of a Markov network.

In chapter 4, the different ways information can propagate through the network are described. This section does not cover propagation in the secondary structure (which is done in chapter 5), but in the Bayesian network. There are basically three different types of connections between variables; serial, diverging and converging. Each connection has its own propagation properties, which are described both formally and using the examples given in chapter one.

In chapter 5, the algorithms for junction-tree propagation are described step by step. The algorithm to create the important secondary structure from the Bayesian network structure is thoroughly explained. This chapter should also be very helpful to those who want to implement their own Bayesian networks system. The initialization of this secondary structure is described. Parallel with the algorithm described, a Bayesian network model is also used as an example on which all algorithms are carried out and explicitly explained. In addition, ways to keep it consistent are shown. Finally, there are methods showing how to introduce observations and how they are updating the quantitative part of the network and our belief about the non-observed variables. The chapter ends by illustration different scenarios using the network model.

In chapter 6, an interesting example where Bayesian networks outperformed predicate logic and normal probability models is presented. It is included to convince the reader that Bayesian networks are useful and encourage to further readings.

In chapter 7, suggestions of what steps to take next after learning the basics of Bayes nets are given, along with this further suggested readings.

In appendix A, a discussion how multi-way arrays can be implemented on a computer system can be found. Multi-way arrays are the foundation for probability distributions and potentials. Also, implementation comments on potential and conditional probability functions are given.

In appendix B, the file format used by `hb`$^{\texttt{BN}}$ to load and save Bayesian networks to the file system are described. The file system is called the XML Belief Network File Format (XBN) and is based on markup language XML.

In appendix C, some of the Bayesian networks used in the report are given in XBN format.

In appendix D, a simple self-defined ad hoc script language for the `hb`$^{\texttt{BN}}$ tools is shown by some examples. There is no formal language specified and for this reason this section is included for those who are curious to see how to use `hb`$^{\texttt{BN}}$.

Acknowledgments

During the year 1996/97 I was studying at University of California, Santa Barbara, and there I also met Peter Kärcher who at the time was in the Computer Science Department. One late night during one of the many international student parties, he introduced the Bayesian networks to me. After that we discussed it just occasionally, but when I returned to Sweden I got more and more interested in the subject. This work would not have been done if Peter never brought up the subject at that party.

Also thanks to all people at my department helping me out when I got stuck in "unsolvable" problems, especially Professor Björn Holmquist that gave me valuable suggestions how to implement multidimensional arrays. I also want to thank Lars Levin and Catarina Rippe for their support. Thanks to Martin Depken and Associate Professor Anders Holtsberg for interesting discussions and for reviewing the report. Of course, also thanks to Professor Jan Holst for initiating and supervising this project.

Thanks to the Uncertainty in Artificial Intelligence Society for the travel grant that made it possible for me to attend the UAI'99 Conference in Stockholm. Thanks also to my department for sending me there.

Chapter 1

Introductory Examples

This chapter will present two examples of Bayesian networks, where the first one will be returned to several times throughout this report. The second example complements the first one and will also be used later on.

The examples will introduce concepts such as evidence or observations, algorithms updating the distribution of some variables given evidence, i.e. to calculate the conditional probabilities. This to give an idea how complex everything can be when we have hundreds or thousands of variables with internal dependencies. Fortunately, there exist algorithms that can easily be run on a computer system.

1.1 Will Holmes arrive before lunch?

This example is directly adopted from [Jen96] and is implemented in `hb`$^{\texttt{BN}}$.
The story behind this example starts by police inspector Smith waiting for Mr Holmes and Dr Watson to arrive. They are already late and Inspector Smith is soon to have lunch. It is wintertime and he is wondering if the roads might be icy. If they are, he thinks, then Dr Watson or Mr Holmes may have been crashing with their cars since they are so bad drivers.
A few minutes later, his secretary tells him that Dr Watson has had a car accident and directly he draws the conclusion that "The roads must be icy!".

-"Since Holmes is such a bad driver he has probably also crashed his car", Inspector Smith, says. "I'll go for lunch now."
-"Icy roads?" the secretary replies, "It is far from being that cold, and furthermore all the roads are salted."
-"OK, I give Mr Holmes another ten minutes. Then I'll go for lunch.", the inspector says.

The reasoning scheme for Inspector Smith can be formalized by a Bayesian network, see figure 1.1. This network contains the three variables X_{Icy}, X_{Holmes} and X_{Watson}. Each is having two states; *yes* and *no*. If the roads are icy the variable X_{Icy} is equal to *yes* otherwise it is equal to *no*. When $X_{Watson} = yes$ it means that Dr Watson has had an accident. Same holds for the variable X_{Holmes}. Before observing anything, Inspector Smiths beliefs about the roads to be icy or not is described by $P(X_{Icy} = yes) = 0.70$

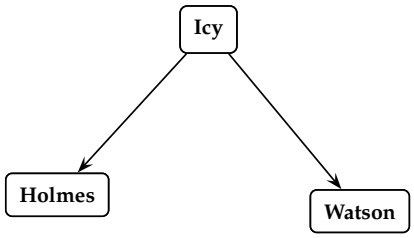


Figure 1.1: The Bayesian network "Icy Roads" contains three variables X_{Icy}, X_{Watson} and X_{Holmes}.

and $P(X_{Icy} = no) = 0.30$. The probability that Watson or Holmes has crashed depends on the road conditions. If the roads are icy, the inspector estimates the risk for Mr Holmes or Dr Watson to crash to be 0.80. If the roads are non-icy, this risk is decreased to 0.10. More formally, we have that $P(X_{Watson}|X_{Icy} = yes) = (0.80, 0.20)$ and $P(X_{Watson}|X_{Icy} = no) = (0.10, 0.90)$ and the same for $P(X_{Holmes}|X_{Icy})$.

How do we calculate the probability that Mr Holmes or Dr Watson has been crashing without observing the road conditions? Using Dr Watson as an example, we first calculate the joint probability distribution $P(X_{Watson}, X_{Icy})$ as

$$\begin{aligned} P(X_{Watson}|X_{Icy} = yes)P(X_{Icy} = yes) &= (0.8, 0.2) \cdot 0.7 = (0.56, 0.14) \\ P(X_{Watson}|X_{Icy} = no)P(X_{Icy} = no) &= (0.1, 0.9) \cdot 0.3 = (0.03, 0.27) \end{aligned}$$

From this we can marginalize X_{Icy} out of the joint probability. We get that $P(X_{Watson}) = (0.56 + 0.03, 0.14 + 0.27) = (0.59, 0.41)$. One will get the same distribution for the belief about Dr Holmes having a car crash or not. This is Inspector Smiths *prior belief* about the road conditions, and his belief in Holmes or Watson having an accident. It is graphically presented in figure 1.2.

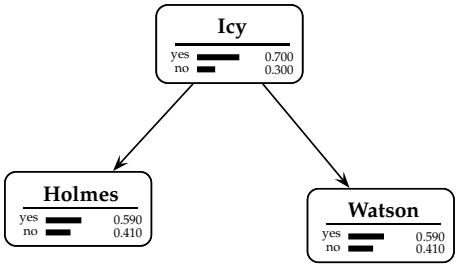


Figure 1.2: The a priori distribution of the network "Icy Roads". Before observing anything the probability for the roads to be icy is $P(X_{Icy} = yes) = 0.70$. The probability that Watson or Holmes has crashed is $P(X_{Watson} = yes) = P(X_{Holmes} = yes) = 0.59$.

When the inspector is told that Watson has had an accident, he *instantiates* the variable X_{Watson} to be equal to *yes*. The information about Watson's crash changes his beliefs about the road conditions and whether Holmes has crashed or not. In figure 1.3 the instantiated (observed) variable is double-framed and shaded gray and its distribution

is fixed to one value (*yes*). The posterior probability for icy roads is calculated using Bayes' rule

$$P(X_{Icy}|X_{Watson} = yes) = \frac{P(X_{Watson} = yes|X_{Icy})P(X_{Icy})}{P(X_{Watson} = yes)} = \\ = \frac{(0.80 \cdot 0.70, 0.10 \cdot 0.30)}{0.59} = \frac{(0.56, 0.03)}{0.59} = (0.95, 0.05).$$

The a posteriori probability that Mr Holmes also has had an accident is calculated by marginalizing X_{Icy} out of the (conditional) joint distribution $P(X_{Holmes}, X_{Icy}|X_{Watson} = yes)$ which is now

$$P(X_{Holmes}, X_{Icy} = yes|X_{Watson} = yes) = (0.8, 0.2) \cdot 0.95 = (0.760, 0.190)$$
$$P(X_{Holmes}, X_{Icy} = no|X_{Watson} = yes) = (0.1, 0.9) \cdot 0.05 = (0.005, 0.045).$$

We get that $P(X_{Holmes} = yes|X_{Watson} = yes) = 0.765$.

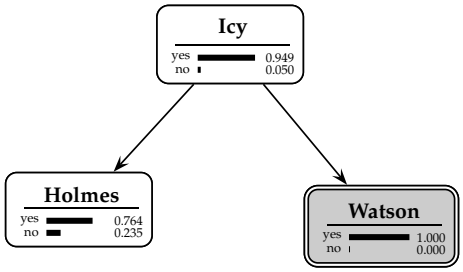


Figure 1.3: The a posteriori distribution of the network "Icy Roads" after observing that Watson had a car accident. Instantiated (observed) variables are double-framed and shaded gray. The new distributions becomes $P(X_{Icy} = yes|X_{Watson} = yes) = 0.950$ and $P(X_{Holmes} = yes|X_{Watson} = yes) = 0.765$.

Just as he drew the conclusion that the roads must be icy, his secretary told him that the roads are indeed not icy. In this very moment Inspector Smith once again receives evidence. In the Bayesian network found in figure 1.4, there are now two instantiated variables; $P(X_{Watson} = yes) = 1$ and $P(X_{Icy} = no) = 1$. The only non-fixed variable is X_{Holmes} which will have its distribution updated. The probability that Mr Holmes also had an accident is now one out of ten, since $P(X_{Holmes} = yes|X_{Icy} = no) = 0.10$. The inspector waits another minute or two before leaving for lunch. Note that, the knowledge about Dr Watson's accident does not affect the belief about Mr Holmes having a accident *if* we know the road conditions. We say that X_{Icy} separated X_{Holmes} and X_{Watson} if it is instantiated (known). This will be discussed further in section 4.2 and 3.2.1.

In this example we have seen how new information (*evidence*) is inserted into a Bayes net and how this information is used to update the distribution of the unobserved variables. Even though it is not exemplified, it is reasonable to say that the order in which evidence arrives does not influence our final belief about having Holmes arrive before lunch or not. Sometimes this is not the case though. It might happen that the order in which we receive the evidences affects our final belief about the world, but

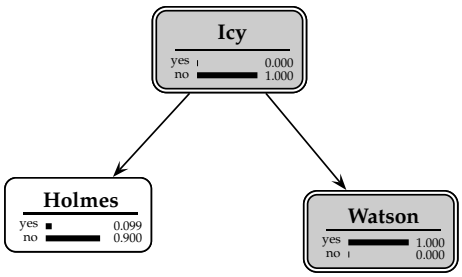


Figure 1.4: The posteriori distribution of the network "Icy Roads" after observing both that Watson has had a car accident and the roads are not icy. The probability that X_{Holmes} had a car crash too is now lowered to 0.10.

that is beyond this report. For this example, we also understand that after observing that the roads are icy, our initial observation that Watson has crashed does not change (of course), and that it does not even effect our belief about Holmes. This is a simple example of a property called *d-separation*, which will discussed further in chapter 4.

1.2 Inheritance of eye colors

The way in which eye colors are inherited is well known. A simplified example can be generated if we assume that there exist only two eye colors; *blue* and *brown*. In this example, the eye color of person is fully determined by the two *alleles*, together called the *genotype* of the person. One allele comes from the mother and one comes from the father. Each allele can be of type b or B, which are coding for blue eye colors and brown eye colors, respectively[1]. There are four different ways the two alleles can be combined, see table 1.1.

♀\♂	b	B
b	bb	bB
B	Bb	BB

Table 1.1: Rules for inheritance of eye colors. B represents the allele coding for brown and b the one coding for blue. It is only the bb-genotype the codes for blue eyes, all other combinations code for brown eyes since B is a dominant allele.

What if a person has one allele of each type? Will she have one blue and one brown eye? No, in this example we define the B-allele to be *dominant*, i.e. if the person has at least one B-allele her eyes will be brown. From this we conclude that, a person with blue eyes can only have the genotype bb and a person with brown eyes can have one of three different genotypes; bB, Bb, and BB, where the two former are identical. This is the reason why two parents with blue eyes can not get children with brown eyes. This is an approximation of how it works in real life, where things are a little bit more complicated. However this is roughly how it works.

[1]The eye color is said to be the *phenotype* and is determined by the corresponding genotype.

In figure 1.5, a family is presented. Starting from the bottom we have an offspring with blue eyes carrying the genotype *bb*. Above in the figure, is her mother and father, and her grandparents.

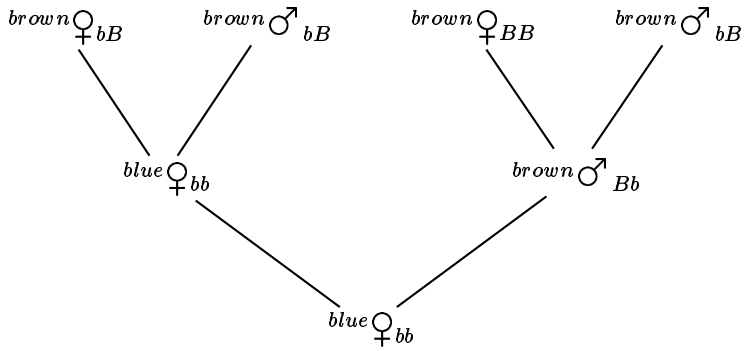


Figure 1.5: Example of how eye colors are inherited. This family contains of three generations; grandparents, parents and their offspring.

Considering that the genotypes bB and Bb are identical, then we can say that there exists three (instead of four) different genotypes (states); bb, bB and BB. Since the child will get one allele from each parent and each allele is chosen out of two by equal probability ($= 1/2$), we can calculate the conditional probability distribution of the child's genotype; $P(X_{child}|X_{mother}, X_{father})$, see table 1.2.

$X_{mother} \backslash X_{father}$	bb	bB	BB
bb	$(1; 0; 0)$	$(1/2; 1/2; 0)$	$(0; 1; 0)$
bB	$(1/2; 1/2; 0)$	$(1/4; 1/2; 1/4)$	$(0; 1/2; 1/2)$
BB	$(0; 1; 0)$	$(0; 1/2; 1/2)$	$(0; 0; 1)$

Table 1.2: The conditional probability distribution of the child's genotype $P(X_{child}|X_{mother}, X_{father})$.

Using this table we can calculate the a priori distribution for the genotypes of a family consisting of a mother, father and one child. In figure 1.6 the Bayesian network is presented. The distributions are very intuitive and it is natural that our belief about different person's eye colors phenotypes or genotypes should be equally distributed if we know nothing about the persons. In this case, not even the information that they belong to the same family will give us any useful information.

Consider another family. Two brown eyed parents with genotypes BB and bB, respectively ($X_{mother} = BB$ and $X_{father} = bB$), are expecting a child. What eye colors will it have? According to table 1.2, the probability distribution function for the expected genotype will be $(0; 1/2; 1/2)$, i.e. the child will have brown eyes. Using the inference engine in `hb`$^{\texttt{BN}}$ we will get exactly the same result. See figure 1.7.

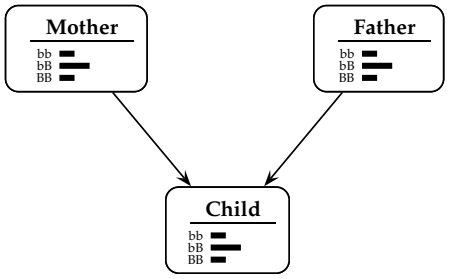


Figure 1.6: The Bayes' net "EyeColors" contains three variables X_{mother}, X_{father} and X_{child}.

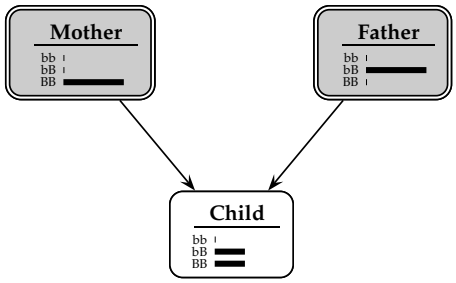


Figure 1.7: What eye colors the child get if the mother is BB and the father is bB? Our belief, after observing the genotypes of the parents, is that the child get the genotypes bB or BB with a chance of fifty-fifty, i.e. in any case it will get brown eyes.

But, what if we only know the eye colors of the parents and not the specific genotype, what happens then? This is a case of *soft evidence*, i.e. it is actually not an instantiation (observation) by definition, since we do not know the exact state of the variable. Soft evidence will not be covered in this report, but it will be exemplified here. People with brown eyes can have either genotype bB or BB. If we make the assumption the allele b and B are equally probable to exists (similar chemical and physical structure etc.), it is also reasonable to say that the genotype of a brown eyed person is bB in $\frac{2}{3}$ of the cases and BB in $\frac{1}{3}$:

$$P(X_{allele} = b) = P(X_{allele} = B) = \frac{1}{2} \Longrightarrow \begin{cases} P(X = bB|\hat{X} = brown) = \frac{2}{3} \\ P(X = BB|\hat{X} = brown) = \frac{1}{3} \end{cases}$$

These assumptions are made about the world *before observing it* and are called the *prior knowledge*. Entering this knowledge into a Bayesian network software we get that the belief that the child will have blue eyes is 0.11. See figure 1.8.

How can we calculate this by hand? We know that the expecting couple can have four different genotype pairs (bB,bB), (bB,BB), (BB,bB) or (BB,BB). The probability for the genotype pairs to occur are $\frac{4}{9}$, $\frac{2}{9}$, $\frac{2}{9}$ and $\frac{1}{9}$, respectively. The distribution of our belief of the child's eye colors will then become

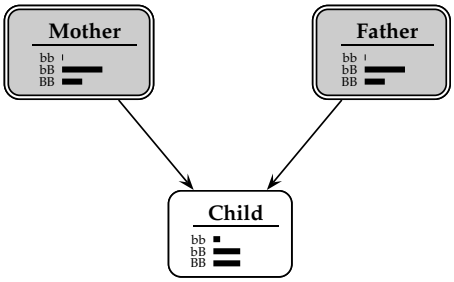


Figure 1.8: What color of the eyes will the child get if all we know is that both parents have brown eyes, i.e. $\hat{X}_{mother} = brown, \hat{X}_{father} = brown$ (soft evidence)? Our belief that the child will get brown eyes is 0.89, i.e. $P(\hat{X}_{child} = brown) = 0.89$.

$$
\begin{aligned}
&P(X_{child}|\hat{X}_{mother} = brown, \hat{X}_{father} = brown) = \\
&= \frac{4}{9}P(X_{child}|X_{mother} = bB, X_{father} = bB) + \frac{2}{9}P(X_{child}|X_{mother} = bB, X_{father} = BB) + \\
&+ \frac{2}{9}P(X_{child}|X_{mother} = BB, X_{father} = bB) + \frac{1}{9}P(X_{child}|X_{mother} = BB, X_{father} = BB) = \\
&= \frac{4}{9}(1/4; 1/2; 1/4) + 2 \cdot \frac{2}{9}(0; 1/2; 1/2) + \frac{1}{9}(0; 0; 1) = (1/9; 4/9; 4/9)
\end{aligned}
$$

From this we conclude that the child will have blue eyes with a probability of $\frac{1}{9}$.

The child is born and it got blue eyes ($\hat{X}_{child} = blue$). How does this new information (*hard evidence*) change our knowledge about the world (genotypes of the parents)? We know from before that a blue eyed person must have the genotype *bb*, i.e. its parents must have at least one *b*-allele each. We also knew that the parents were brown eyed, i.e. they had either *bB* or *BB* genotypes. All this together infers that both parents must be *bB*-types, see figure 1.9.

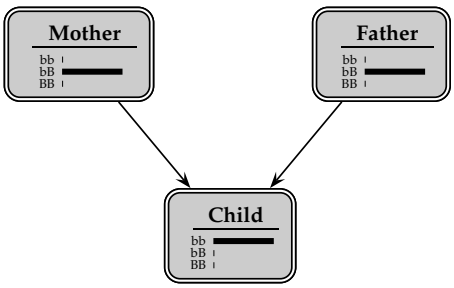


Figure 1.9: The observation that the child has blue eyes ($\hat{X}_{child} = blue$), updates our previous knowledge about the brown-eyed parents ($\hat{X}_{mother} = brown, \hat{X}_{father} = brown$). Now we know that both must have genotype *bB*.

This example showed how physical knowledge could be used to construct a Bayesian network. Since we know how eye colors are inherited, we could easily create a Bayesian network that represents our knowledge about the inheritance rules.

Chapter 2

Graph Theory

A Bayesian network is a directed acyclic graph (DAG). A DAG, together with all other terminology found in the graph theory are defined in this section. Those who know graph theory can most certainly skip this chapter. The notation and terminology used in this report are mainly a mixture taken from [Lau96] and [Rip96].

2.1 Graphs

A *graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a set of *vertices*, $\mathcal{V}$, and a set of *edges*, $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$[1]. In this report, a vertex is denoted with lower case letters; u, v, and a, b, c etc. Each edge $e \in \mathcal{E}$ is an ordered pair of vertices (u, v) where $u, v \in \mathcal{V}$. An *undirected* edge $u - v$ has both $(u, v) \in \mathcal{E}$ and $(v, u) \in \mathcal{E}$. A *directed* edge $u \rightarrow v$ is obtained when $(u, v) \in \mathcal{E}$ and $(v, u) \notin \mathcal{E}$. If all edges in the graph are undirected we say that the graph in *undirected*. A graph containing only directed edges is called a *directed* graph. When referring to a graph with both directed and undirected edges we use the notation *semi-directed* graph. Self-loops, which are edges from a vertex to itself, are not possible in undirected graphs because then the graph is defined to be (semi-) directed.

If there exists an edge $u \rightarrow v$, u is said to be a *parent* of v, and v a *child* of u. We also say that the edge *leaves* vertex u and *enters* vertex v. The set of parents of a vertex u is denoted $pa(u)$ and the set of children is denoted $ch(u)$. If there is an ordered or unordered pair $(u, v) \in \mathcal{E}$, u and v are said to be *adjacent* or *neighbors*, otherwise they are *non-adjacent*. The set of neighbors of u is denoted as $ne(u)$. The *family*, $fa(u)$, of a vertex u is the set of vertices containing the vertex itself and its parents. We have

$$\begin{aligned} pa(v) &= \{u \in \mathcal{V} | (u, v) \in \mathcal{E} \wedge (v, u) \notin \mathcal{E}\} \\ ch(u) &= \{v \in \mathcal{V} | (u, v) \in \mathcal{E} \wedge (v, u) \notin \mathcal{E}\} \\ ne(u) &= \{v \in \mathcal{V} | (u, v) \in \mathcal{E} \vee (v, u) \in \mathcal{E}\} \\ fa(u) &= \{u\} \cup pa(u) \end{aligned}$$

The adjacent (neighbor) relation is symmetric in an undirected graph.

[1] In some literature *nodes* and *arcs* are used as synonyms for vertices and edges, respectively.

In figure 2.1, a semi-directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V} = \{a, b, c, d\}$ and edge set $\mathcal{E} = \{(a, b), (a, c), (c, a), (c, b), (c, d), (d, c)\} = \{a \rightarrow b, a - c, c \rightarrow b, c - d\}$ is presented. Consider vertex b. Its parent set is $pa(b) = \{a, c\}$ and its child set is empty; $ch(b) = \emptyset$. The neighbors of b are $ne(b) = \{a, c\}$. Moving the focus to vertex c, we see that its parents set is empty, but its child set is $ch(c) = \{b\}$ and $ne(c)$ is $\{a, b, d\}$.

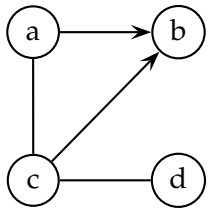


Figure 2.1: A simple graph with both directed and undirected edges.

2.1.1 Paths and cycles

A *path* of *length* l from a vertex u to a vertex u' is a sequence $\langle v_0, v_1, \ldots, v_l \rangle$ of vertices such that $u = v_0$ and $u' = v_l$, and $(v_{i-1}, v_i) \in \mathcal{E}$ for all $i = 1, 2, \ldots, l$. The vertex u' is said to be *reachable* from u via p, if there exists a path p from u to u'. Both directed and undirected graphs can have paths. A *simple* path $p = \langle v_0, v_1, \ldots, v_l \rangle$ is a path where all vertices are distinct, i.e. $v_i \neq v_j$ for all $i \neq j$.

A *cycle* is a path $p = \langle v_0, v_1, \ldots, v_l \rangle$ (with $l \geq 1$) where $v_0 = v_l$. A self-loop is the shortest cycle possible. In an undirected graph, a cycle must also conform to the fact that all vertices are distinct. As an example, some of the cycles in figure 2.2 are $p_1 = \langle b, f, g, b \rangle$, $p_2 = \langle c, e, h, d, b, c \rangle$ and $p_3 = \langle g, d, h, e, c, b, f, g \rangle$. There is no cycle containing vertex a.

2.1.2 Common structures

A directed (undirected) *tree* is a connected and directed (undirected) graph without cycles. In other words, undirecting a graph, it is a tree if between any two vertices a *unique* path exists. A set of trees is called a *forest*.
A *directed acyclic graph* is a directed graph without any cycles. If the direction of all edges in a DAG are removed and the resulting graph becomes a tree, then the DAG is said to be *singly connected*. Singly connected graphs are important specific cases where there is no more than one undirected path connecting each pair of vertices.

If, in a DAG, edges between all parents with a common child are added (the parents are *married*) and then the directions of all other edges are removed, the resulting graph is called a *moral graph*. Using Pearls notation, we call the parents of the children of a vertex the *mates* of that vertex.

DEFINITION 1 (TRIANGULATED GRAPH)
A triangulated *graph is an undirected graph where cycles of length three or more always have a* chord*. A chord is an edge joining two non-consecutive vertices.*

2.1.3 Clusters and cliques

A *cluster* is simply a subset of vertices in a graph. If all vertices in a graph are neighbors with each other, i.e. there exists an edge (u, v) or (v, u) for $\forall u, v \in \mathcal{E}$, then the graph is said to be *complete*. A *clique* is a maximal set of vertices that are all pairwise connected, i.e. a maximal subgraph that is complete. In this report, a cluster or a clique is denoted with upper case letters; A, B, C etc. The graph in figure 2.1 has one clique with more than two vertices, namely the clique $C_1 = \{a, b, c\}$. There is also one clique with only two vertices; $C_2 = \{c, d\}$. Note that for instance $C = \{a, b\}$ is not a clique, since it is not a *maximal* complete set.

Define the *boundary* of a cluster, $bd(A)$, to be the set of vertices in the graph $\mathcal{G}$ such that they are not in A, but they have a neighbor or a child in A. The *closure* of a cluster, $cl(A)$, is the set containing the cluster itself and its boundary. We also define parents, children and neighbors in the case of clusters . $pa(A)$, $ch(A)$, $ne(A)$, $bd(A)$ and $cl(A)$ are formally defined as

$$
\begin{aligned}
pa(A) &= \bigcup_{u \in A} pa(u) \backslash A \\
ch(A) &= \bigcup_{u \in A} ch(u) \backslash A \\
ne(A) &= \bigcup_{u \in A} ne(u) \backslash A \\
bd(A) &= pa(A) \cup ne(A) \\
cl(A) &= A \cup bd(A)
\end{aligned}
$$

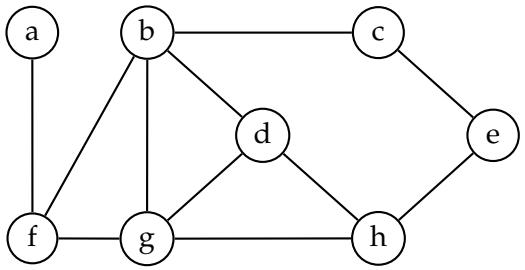


Figure 2.2: Example of neighbor set and the boundary set of clusters in an undirected graph. The neighbor and boundary set of $\{b, g\}$ are both $\{c, d, f, h\}$.

Consider the graph in figure 2.2. Note that there exists no parents or children in this graph since its undirected. Neighbors and boundary vertices exist, though. Let A be the set $\{b, g\}$. The neighbor set of A is $ne(A) = \{c, d, f, h\}$ and the boundary is of course

the same since we are dealing with an undirected graph. The cliques in this graph are (in order of number of vertices): $C_1 = \{a, f\}, C_2 = \{b, c\}, C_3 = \{c, e\}, C_4 = \{e, h\}, C_5 = \{b, d, g\}, C_6 = \{b, f, g\}$, and $C_7 = \{d, g, h\}$.

DEFINITION 2 (SEPARATOR SETS)
Given three clusters A, B and C, we say that C separates *A and B in $\mathcal{G}$ if every path from any vertex in A to any vertex in B goes through C. C is called a* separator set *or shorter* sepset.

In left graph of figure 2.3, vertex a and c are separated by vertex b. Vertex b does also separate cluster $\{a, b\}$ and $\{b, c\}$. In the right graph, the cliques $A = \{a, b, c\}$ and $B = \{b, d, e\}$ are separated by any of the three clusters $C_1 = \{b, c\}$, $C_2 = \{b, d\}$, and $C_3 = \{b, c, d\}$.

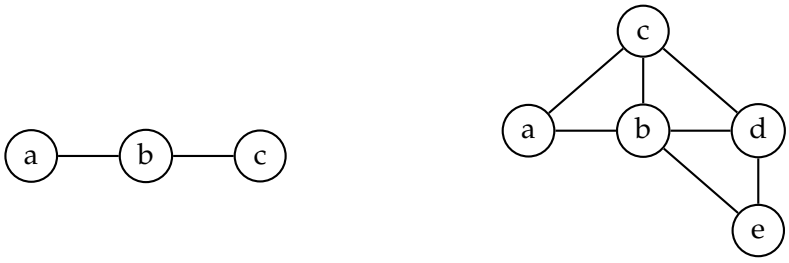


Figure 2.3: Left: Vertex b separates vertex a and vertex c. It is also the case that vertex b separates the clusters $A = \{a, b\}$ and $B = \{b, c\}$. Right: The clusters $C_1 = \{b, c\}$, $C_2 = \{b, d\}$, and $C_3 = \{b, c, d\}$ do all separate the cliques $A = \{a, b, c\}$ and $B = \{b, d, e\}$.

Chapter 3

Markov Networks and Markov Trees

Before exploring the theory of Bayesian networks, this chapter introduces a simpler kind of graphical models, namely the Markov trees. Markov trees are a special case of Markov networks and they have properties that make calculations straightforward. The main drawback of the Markov trees is that they can not model as complex systems as can be modeled using Bayes nets. Still they are interesting since they carry the basics of the Bayesian networks. Also, the secondary structure into which the junction tree algorithm (see chapter 5) is converting the belief network can be seen as a multidimensional Markov tree. One reason why the calculations on a Markov tree are so straightforward is that a Markov networks has special properties which are based on the definition of independence between variables. These properties are called Markov properties. A Bayesian network does normally not carry these properties.

3.1 Overview

3.1.1 Markov networks

A Markov network is an undirected graph with the vertices representing random variables and the edges representing dependencies between variables they connects. In figure 3.1, an example of a Markov network is given. The vertex labeled a represent the random variable X_a and similar for the other vertices. In this report variables are denoted with capital letters; X, Y and Z.

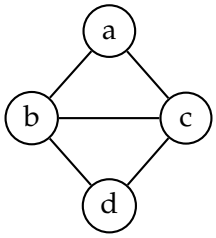


Figure 3.1: An example of a Markov network with four vertices $\{a, b, c, d\}$ representing the four variables $\{X_a, X_b, X_c, X_d\}$.

3.1.2 Markov trees

A Markov tree is a Markov network with tree structure. A Markov chain is a special case of a Markov tree where the tree does not have any branches.

A rooted (or directed) Markov tree can be constructed from a Markov tree by selecting any vertex as the *root* and then direct all edges outwards from the root. To each edge, $(pa(u), u)$, a conditional probability is assigned; $P(X_u|X_{pa(u)})$. In the same way as we do calculations on a Markov chain, we can do calculations on a Markov tree. The methods for the calculations proceed towards or outwards from the root. The distribution of a variable X_u in a rooted Markov tree is fully given by its parent $X_{pa(u)}$, i.e. $P(X_u|X_{\mathcal{V}\setminus\{u\}}) = P(X_u|X_{pa(u)})$. With this notation, the joint distribution for *all* variables can be calculated as

$$P(\mathbf{X}_\mathcal{V}) = \prod_{u\in\mathcal{V}} P(X_u|X_{pa(u)}) \tag{3.1}$$

where $P(X_u|X_{pa(u)}) = P(X_u)$ if u is the root.

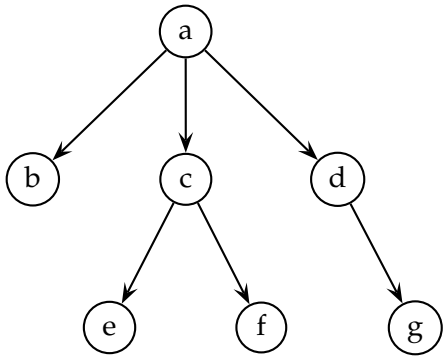


Figure 3.2: An example of a directed Markov tree constructed from an undirected Markov tree. In this case vertex a was selected to be the root vertex.

Consider the graph in figure 3.2. The joint probability distribution over $\mathcal{V}$ is

$$P(\mathbf{X}_\mathcal{V}) = P(X_a)P(X_b|X_a)P(X_c|X_a)P(X_d|X_a)P(X_e|X_c)P(X_f|X_c)P(X_g|X_d).$$

From this one can see that the Markov tree offers a compact way of representing a joint distribution. If all variables in the graph have ten states each, the state space of the joint distribution will have 10^7 states. If one state requires one byte (optimistic though) the joint distribution would require 10Mb of memory. The corresponding Markov tree would require much less since each conditional distribution $P(X_u|X_v)$ has $10\cdot 10 = 100$ states and $P(X_a)$ only has ten. We get $10 + 6\cdot 100 = 610$ states which requires 610 bytes of memory. This is one of the motivations for using graphical models.

3.1.3 Bayesian networks

Bayesian networks are directed acyclic graphs (DAGs)[1] in which vertices, as for Markov

[1]This property is very important in the concept of Bayesian networks, since cycles will induce redundancy, which is very hard to deal with.

networks, represent uncertain *variables* and the edges between the vertices represent probabilistic interactions between the corresponding variables. In this report the words variable and vertices are used interchangeable. A Bayesian network is parameterized (quantified) by giving, for each variable X_u, a conditional probability given its parents, $P(X_u|\mathbf{X}_{pa(a)})$. If the variable does not have any parents, a prior probability $P(X_u)$ is given.

Doing probability calculations on Bayesian networks are far from as easy as it is on Markov trees. The main reason for this is the tree structure that Bayes nets normally do not have. Information can be passed in both directions of an edge (see examples in chapter 1), which means that information in a Bayesian network can be propagated in *cycles* which is hard to deal with. Fortunately, there are some clever methods that handle this problem, see figure 3.3. One such method is the junction tree algorithm discussed in chapter 5.

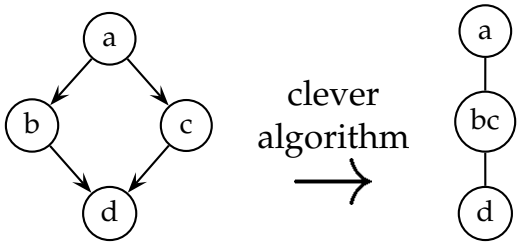


Figure 3.3: A clever algorithm to convert a Bayesian network into a Markov tree with multi-dimensional variables would help us get around the problem with cycles.

The theory of Markov trees is not restricted to one-dimensional variables, but multi-dimensional variables can also be used and represented by the vertices. This means that if we can build a Markov tree where each vertex is a cluster of variables, which then has a joint distribution, we can instead do the inference on the new Markov tree. This is exactly what used by the junction tree algorithm (see chapter 5); by clustering variables in a special way we can create a structure on which we easily can do conditional probability calculations.

3.2 Theory behind Markov networks

3.2.1 Conditional independence

DEFINITION 3 (CONDITIONAL INDEPENDENT VARIABLES)
Given three discrete *variables X, Y and Z, we say that X* is conditionally independent of Y given Z, *if*

$$P(X = x, Y = y|Z = z) = P(X = x|Z = z) \cdot P(Y = y|Z = z); \;\; \forall z \; s.t. \; P(Z = z) > 0.$$

This property is written as

$$X \perp\!\!\!\perp Y|Z,$$

and it follows directly from the definition that

$$X \perp\!\!\!\perp Y|Z \iff Y \perp\!\!\!\perp X|Z.$$

Note that Z can be the empty set. In that case we have $X \perp\!\!\!\perp Y|\emptyset$ which is equivalent with $X \perp\!\!\!\perp Y$. Two special cases are of interest for further reading and discussion. For any discrete variable X it holds that:

$$(i) \quad X \perp\!\!\!\perp X|X \text{ is } \textit{always} \text{ true} \tag{3.2}$$

$$(ii) \quad X \perp\!\!\!\perp X \text{ is } \textit{only} \text{ true iff } X \text{ is a constant variable.} \tag{3.3}$$

These results are very intuitive, and a proof will not be given. Later in this section similar results ((3.4) and (3.5)), but also more general, are presented and prooven.

We extend this formulation to cover sets of variables. First, in the same way as the variable represented by vertex a is denoted X_a, the multidimensional variable represented by the cluster A is denoted $\mathbf{X}_A$.

DEFINITION 4 (CONDITIONAL INDEPENDENT CLUSTERS)
Given three clusters A, B and C and their associated discrete *variables* $\mathbf{X}_A$, $\mathbf{X}_B$ *and* $\mathbf{X}_C$, *we say that* $\mathbf{X}_A$ is conditionally independent of $\mathbf{X}_B$ given $\mathbf{X}_C$, *if*

$$P(\mathbf{X}_A = \mathbf{x}_A, \mathbf{X}_B = \mathbf{x}_B|\mathbf{X}_C = \mathbf{x}_C) = P(\mathbf{X}_A = \mathbf{x}_A|\mathbf{X}_C = \mathbf{x}_C) \cdot P(\mathbf{X}_B = \mathbf{x}_B|\mathbf{X}_C = \mathbf{x}_C)$$

for all $\mathbf{x}_C$ *such that* $P(\mathbf{X}_C = \mathbf{x}_C) > 0$. *This property is written as*

$$\mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_B|\mathbf{X}_C,$$

and obviously

$$\mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_B|\mathbf{X}_C \iff \mathbf{X}_B \perp\!\!\!\perp \mathbf{X}_A|\mathbf{X}_C.$$

We can show that for any discrete variable X, the following is valid:

$$(i) \quad X \perp\!\!\!\perp (X, X, \dots)|X \text{ is } \textit{always} \text{ true} \tag{3.4}$$

$$(ii) \quad X \perp\!\!\!\perp (X, X, \dots) \text{ is } \textit{only} \text{ true iff } X \text{ is a constant variable.} \tag{3.5}$$

PROOF (i) $X \perp\!\!\!\perp (X, X, \dots)|X$ is equivalent with the statement $P(X = x_1, (X, X, \dots) = = (x_2, x_3, \dots)|X = x) = P(X = x_1|X = x) \cdot P((X, X, \dots) = (x_2, x_3, \dots)|X = x)$. When $x_1 = x_2 = x_3 = \dots = x$ the expression is equal to $1 = 1$ and otherwise $0 = 0$. Hence, (i) is proven. (ii) $X \perp\!\!\!\perp (X, X, \dots)$ is equivalent with the statement $P(X = x_1, (X, X, \dots) = (x_2, x_3, \dots)) = P(X = x_1) \cdot P((X, X, \dots) = (x_2, x_3, \dots))$ for all $(x_1, x_2, x_3, \dots)$. In the special case when $x_1 = x_2 = x_3 = \dots = x$ we have that $P(X = x, (X, X, \dots) = (x, x, \dots)) = P(X = x) \cdot P((X, X, \dots) = (x, x, \dots))$ This tells us that $P(X = x) = 0$ or $P(X = x) = 1$, which is the same as saying that X is a constant. It is trivial that when X is a constant, $X \perp\!\!\!\perp (X, X, \dots)$ also holds. Hence, (ii) is proven. □

DEFINITION 5 (INDEPENDENT MAP OF DISTRIBUTION)
A undirected graph $\mathcal{G}$ is said to be an I-map *(independent map) of the distribution if $\mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_B | \mathbf{X}_C$ holds for any clusters A and B with separator C.*

Note that all complete graphs are actually I-maps since there exist no separators at all which implies that all clusters are independent given its separators.

3.2.2 Markov properties

DEFINITION 6 (MARKOV PROPERTIES)
The Markov properties *for a probability distribution $P(\cdot)$ with respect to a graph $\mathcal{G}$ are*

(G) **Global Markov** – *for* any *disjoint clusters A, B and C such that C is a separator between A and B it is true that $\mathbf{X}_A$ is independent of $\mathbf{X}_B$ given $\mathbf{X}_C$, i.e.*

$$\mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_B | \mathbf{X}_C;\ \forall A, B, C \text{ s.t. } A \cap B = B \cap C = C \cap A = \emptyset\ \wedge\ C \text{ separates } A \text{ and } B$$

(L) **Local Markov** – *the conditional distribution of* any *variable X_a given $\mathbf{X}_{\mathcal{V}\setminus\{X_a\}}$ depends only on the boundary of a, i.e.*

$$X_a \perp\!\!\!\perp \mathbf{X}_{\mathcal{V}\setminus cl(a)} | \mathbf{X}_{bd(a)};\ \forall a \in \mathcal{V}$$

(P) **Pairwise Markov** – *for* all *pair or vertices a and b such that there is no edge between them, the variables X_a and X_b are conditionally independent given all other variables, i.e.*

$$X_a \perp\!\!\!\perp X_b | \mathbf{X}_{\mathcal{V}\setminus\{a,b\}};\ \forall a, b \in \mathcal{V} \text{ s.t. } (a,b) \notin \mathcal{E}\ \wedge\ (b,a) \notin \mathcal{E}$$

To say that a graph is global Markov is the same as saying that the graph is an I-map, and vice versa.

THEOREM 1 (ORDER OF THE MARKOV PROPERTIES)
The internal relation between the Markov properties is

$$(G) \Longrightarrow (L) \Longrightarrow (P)$$

PROOF For a proof, see [Lau96] □

An example where the local property holds but not the global is adopted from [Rip96]. This is the case for the graph found in figure 3.4 if we put the *same* non-constant variable at all four vertices; $X = X_a = X_b = X_c = X_d$.

Figure 3.4: This graph has local, but not global Markov property, if all vertices are assigned the same (non-constant) variable.

Since the boundary of any of the vertices in the graph is a single vertex and all vertices have the same variable X, $\mathbf{X}_{bd(u)}$ is equal to $\{X\}$ for all $u \in \mathcal{V}$. The closure of a and b are equal; $cl(a) = cl(b) = \{a, b\}$. Similarly, $cl(c) = cl(d) = \{c, d\}$. Further, we have that $\mathcal{V}\backslash\{a, b\} = \{c, d\}$ and $\mathcal{V}\backslash\{c, d\} = \{a, b\}$. Now, since $\mathbf{X}_{\mathcal{V}\backslash\{a,b\}} = \mathbf{X}_{\mathcal{V}\backslash\{c,d\}} = (X, X)$, the local Markov property

$$X_u \perp\!\!\!\perp \mathbf{X}_{\mathcal{V}\backslash cl(u)} | \mathbf{X}_{bd(u)}; \forall u \in \mathcal{V}$$

can be written as $X \perp\!\!\!\perp (X, X)|X$ where (X, X) denotes the collection of variables over any pair of vertices in the graph. From (3.2) we get that this is always true. Hence, we have shown that the local Markov property holds for the graph. The global Markov property does not hold, though. $\{b\}$ and $\{c\}$ is separated by $\emptyset$, but from (3.3) we get that $X_b \perp\!\!\!\perp X_c|\emptyset$ is false since $X_b = X_c = X$ and X is a non-constant random variable. An example adopted from [Lau96] presents a case where the graph has pairwise but not local Markov properties. Consider the graph in figure 3.5.

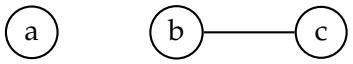


Figure 3.5: This graph has pairwise, but not local Markov property, if all vertices are assigned the same (non-constant) variable.

Let $X = X_a = X_b = X_c$ and put for instance $P(X = 1) = P(X = 0) = \frac{1}{2}$. We now have a case when the pairwise Markov property holds but not the local. There are only two pairs of vertices such that there is no edge between them, namely (a, b) and (a, c). We have that $X_a \perp\!\!\!\perp X_b|X_c \iff X \perp\!\!\!\perp X|X$. The latter is always true, according to (3.2). The same holds for the pair (a, c). Hence, the graph is pairwise Markov. But from (3.5) we have that $X_a \perp\!\!\!\perp \mathbf{X}_{\mathcal{V}\backslash cl(a)}|\mathbf{X}_{bd(a)} \iff X \perp\!\!\!\perp (X, X)|\emptyset$ is not true. Hence, the graph is not local Markov.

The last two examples are at a first glance quite strange. Why construct a model where the graph does not reflect the dependency between the variables and vice verse? The reason for this is to show that inconsistent models *can* exist. Given a Bayesian network we can not be sure that it has global or even local Markov properties. But, to be able to do inference on our network it would be nice if it is global Markov. The junction tree algorithm proposed by the ODIN group at Aalborg University [Jen96] makes inference by first transforming the Bayesian network into a graph structure which has global Markov properties. There are also some more restrictions on the junction tree. In chapter 5, junction trees will be discussed further.

The following statement is important and it comes originally from [Mat92]. As we will

see in chapter 5, it is one of the foundations of the junction tree algorithm.

THEOREM 2 (MARKOV PROPERTIES EQUIVALENCE)
All three Markov properties are equivalent for any *distribution if and only if every subgraph on three vertices contains two or three edges.*

PROOF See [Rip96]. □

The following two theorems are important and are taken from [Rip96]. They are often attributed to Hammersley & Clifford in 1971, but were unpublished for over two decades, see [Cli90].

THEOREM 3 (EXISTENCE OF A POTENTIAL REPRESENTATION)
Let us say we have a collection of discrete random variables, with distribution function $P(\cdot)$, defined on the vertices of a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$. Then, given that the joint distribution is strictly positive, the pairwise Markov property implies that there are positive functions ϕ_C, such that

$$P(\mathbf{X}_\mathcal{V}) \propto \prod_{C \in \mathcal{C}} \phi_C(\mathbf{x}_C) \tag{3.6}$$

where $\mathcal{C}$ is the set of cliques of the graph. (3.6) is called a potential representation.

The functions ϕ_C are not uniquely determined [Lau96].

DEFINITION 7 (FACTORIZATION OF A PROBABILITY DISTRIBUTION)
A distribution function $P(\cdot)$ on $\mathbf{X}_\mathcal{V}$ is said to factorize *according to $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ if there exists a potential representation (see 3.6). If $P(\cdot)$ factorizes, we say that $P(\cdot)$ has property (F).*

Comparing (3.1) with (3.6), it is easily found that former equation actually describes a potential representation of the distribution $P(\mathbf{X}_\mathcal{V})$.

THEOREM 4 (POTENTIALS IMPLIES GLOBAL MARKOV)
The potential representation (3.6) implies the global Markov property for any distribution;

$$(F) \Longrightarrow (G)$$

PROOF For a proof of these two theorems see [Rip96]. □

What THEOREM 3 says is that $(P) \Longrightarrow (F)$, so the cycle is closed. We can conclude that the following theorem holds.

THEOREM 5 (POTENTIALS IS EQUAL TO MARKOV PROPERTIES)
For any undirected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ *and any probability distribution on* $\mathbf{X}_{\mathcal{V}}$ *it holds that*

$$(F) \Longleftrightarrow (G) \Longleftrightarrow (L) \Longleftrightarrow (P)$$

In this chapter we have defined Markov properties and Markov trees. The last theorem summarizes the chapter by saying that if the probability distribution with respect to a graph that has Markov properties it also has a potential representation, and vice versa.

In chapter 5 we will show that the secondary structure (a junction tree), into which the Bayesian network is transformed, has such properties that there always exists a potential representation.

Chapter 4

Propagation of Information

4.1 Introduction

Before observing anything, the Bayesian network represents our *prior* belief about the world. When we later receive new information we are said to have a *posterior* belief about the world. This *belief update* is performed by well-defined knowledge propagation rules. In this chapter, these rules are presented and discussed. Once again, but now in the context of evidence and flow of information, the concept of independence between variables are discussed. How the actual computation is done is given in chapter 5.

4.2 Connectivity and information flow

4.2.1 Evidence

Evidence (sometimes called *finding*) on a variable is a statement of the certainties of its states. If the evidence tells us exactly which state the variable is in, it is called *hard* evidence (or *instantiation*) and we say that the variable is *instantiated*. In all figures in this report, instantiated variables are double-framed and colored *gray*. Evidence that are not hard are called *soft* evidence. An example of soft evidence on a variable X_a is when one of its descendants has received (hard or soft) evidence, see figure 4.4. Variables that have received soft evidence have vertices that are also double-framed but they are now dashed. The vertices are also colored with a somewhat lighter version of gray.

4.2.2 Connections and propagation rules

A *serial connection* in a belief network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a directed subgraph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ such that $\mathcal{V}' = \{a, b, c\} \subset \mathcal{V}$ and $\mathcal{E}' = \{(a, b), (b, c)\} \subset \mathcal{E}$. The connection can also be written as $X_a \rightarrow X_b \rightarrow X_c$. In figure 4.1, an example of a serial connection is given. A connection that is serial has the property that, if we receive evidence about X_a, it will update our knowledge about X_b, which then in turn will update our knowledge about X_c. This propagation of information is done by *marginalization*. Similar, if we receive evidence about X_c, it will, through *Bayes' Rule* update our knowledge about X_b and continue on to X_a (see left graph). If X_b is *instantiated*, all information will be *blocked* at X_b, e.g. new information about X_a will not reach X_c, since X_b is determined.

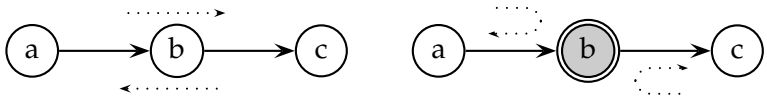


Figure 4.1: Serial connection. A vertex colored *gray* is an *instantiated* variable. Non-instantiated variables are white. In a serial connection $X_a \rightarrow X_b \rightarrow X_c$ the information can not pass X_b if it is instantiated.

The same is valid if we receive evidence about X_c. Our knowledge about X_a will not be changed (see right graph). Concluding, a serial connection has the property $X_a \perp\!\!\!\perp X_c | X_b$.

Consider the serial relation $X_{grandmother} \rightarrow X_{mother} \rightarrow X_{child}$ that exists in our example of eye colors given in section 1.2. If we know the genotype of the X_{mother}, let us say bB, then observing the genotype of the $X_{grandmother}$, e.g. $X_{grandmother} = bb$, will not change our belief about what genotype the X_{child} has. The same reasoning is valid in the opposite direction.

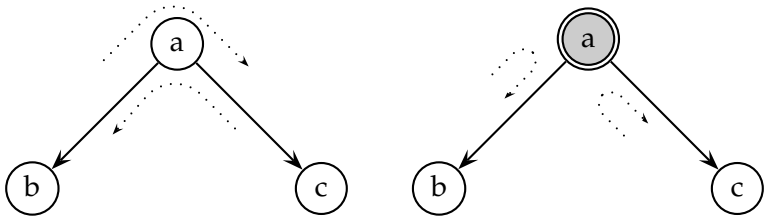


Figure 4.2: Diverging connection. In a diverging connection $X_b \leftarrow X_a \rightarrow X_c$ the information can not pass X_a if it is instantiated.

A *diverging connection* in a Bayesian network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a directed subgraph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ such that $\mathcal{V}' = \{a, b, c\} \subset \mathcal{V}$ and $\mathcal{E}' = \{(a, b), (a, c)\} \subset \mathcal{E}$. Shortly, it can be written $X_b \leftarrow X_a \rightarrow X_c$. See also figure 4.2. The *diverging connection* has similar properties to the serial connection. If X_a is *not* instantiated, new information received about X_b or X_c will update our knowledge about X_a. Then the information will continue to X_c and X_b, respectively (see left graph). In the case that X_a is instantiated, information will not be able to pass through X_a. It is blocked at X_a (see right graph). Concluding, a diverging connection $X_b \leftarrow X_a \rightarrow X_c$ has the property $X_b \perp\!\!\!\perp X_c | X_a$.

Remember the "Icy Roads" example in section 1.1. The Bayes net itself was one single diverging connection, $X_{Holmes} \leftarrow X_{Icy} \rightarrow X_{Watson}$, and we also came to the conclusion that $X_{Holmes} \perp\!\!\!\perp X_{Watson} | X_{Icy}$.

In a Bayesian network, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a *converging connection* is a directed subgraph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ such that $\mathcal{V}' = \{a, b, c\} \subset \mathcal{V}$ and $\mathcal{E}' = \{(a, c), (b, c)\} \subset \mathcal{E}$. The connection can also be written as $X_a \rightarrow X_c \leftarrow X_b$. See figure 4.3. A converging connection will in some sense have opposite propagation properties compared to a serial or a diverg-

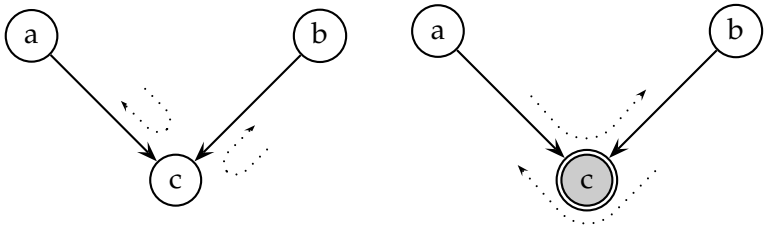


Figure 4.3: Converging connection. In a converging connection $X_a \rightarrow X_c \leftarrow X_b$ the information can *only* pass X_c if it has received some evidence.

ing connection. Evidence received in X_a or X_b will *only* pass X_c, if X_c has receive some evidence, hard (see right graph) *or* soft. We say that the evidence at X_c *opens* for information. If nothing is known about X_c, the new information will bounce back (see left graph) and we call the parents independent. Note that it is not necessary for X_c to be instantiated to make the parents dependent, only that we know *something* about X_c. In the case described in figure 4.4, there is an opening in the converging connection at variable X_c, because it has received soft evidence from its descendants, X_d and X_e. Variable X_e did first receive (hard) evidence. Concluding, a converging connection $X_a \rightarrow X_c \leftarrow X_b$ has the property $X_a \not\perp\!\!\!\perp X_b \mid X_c$.

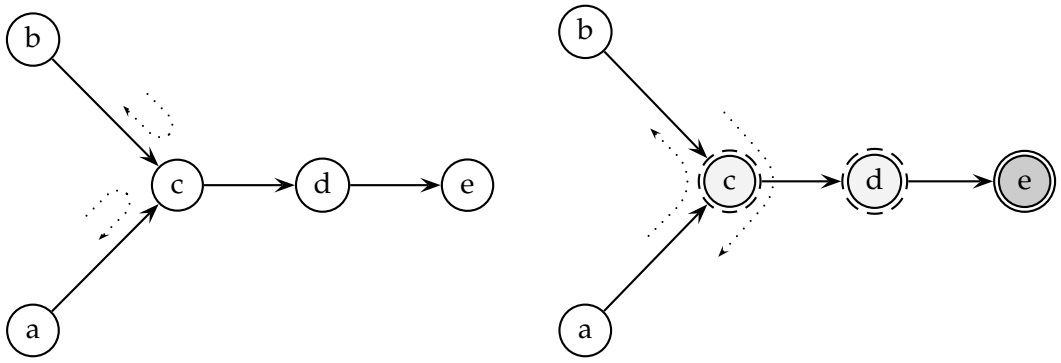


Figure 4.4: Converging connection. When X_e becomes instantiated both X_d and X_c receives (soft) evidence. This will open up the converging connection $X_a \rightarrow X_c \leftarrow X_b$ and hence X_a and X_b is connected. Vertices that are double-framed with the outer frame dashed represent variables that have received soft evidence. These vertices are also shaded gray, but with lighter version of gray than vertices with hard evidence.

Once again, go back to the eye color example and look at the converging connection $X_{mother} \rightarrow X_{child} \leftarrow X_{father}$. The knowledge about the X_{father} will not affect your knowledge about the X_{mother}, as long as the X_{child} is unobserved.

A comment about the eye color example and soft evidence: Note that the rules for serial and diverging connections are only valid if you observe the *genotype* (hard evidence). They

does not hold if you just observe the *phenotype*, i.e. the *eye color* (soft evidence).

4.2.3 d-connection and d-separation

In previous section some examples of how information is propagated between variables were given. These "rules" are intuitive and a property of human reasoning. They decide how and when information propagates through different types of connection. The rules can be summarized by the definition of *d-separation* (dependency separation).

DEFINITION 8 (D-SEPARATED VARIABLES)
Two variables X_a and X_b are d-separated *if for* all *paths between X_a and X_b there is an intermediate variable X_c such that either the connection is*

- *serial or diverging and the state of X_c is known.*
- *converging and neither X_c nor its descendants have received evidence.*

Two variables that are not d-separated are said to be d-connected.

Except from this, the report does not mention much about d-separation, but since it is so common in the literature the definition was included. It is interesting since it describes how two variables are related with each other given our knowledge about other variables.

Chapter 5

The Junction Tree Algorithm

It is possible to evaluate the marginal probability of all variables given observations on some other variables. The problem of conditioning Bayesian networks on observations is in general NP-hard [Rip96] (originally shown in [Coo90]), but experience shows that in many real systems the networks are sparsely connected and therefore the calculations are tractable [Rip96]. Several algorithms have been proposed, some more successful than others. One of the first algorithms [Pea88] designed operated directly on the Bayesian network structure. Each variable was given a process and each time a variable received some new evidence it sent a message to its neighbors, which sent new messages to their neighbors. After a while the Bayesian network is hoped to become stable. The most popular algorithm today is the junction tree algorithm designed by the Odin group at Aarhus University [Jen96, Jen94].

The junction tree algorithm, is a process in two major steps, *transformation* and *propagation*. The transformation step builds an undirected *junction tree* from the Bayesian network. The junction tree contains cliques of random variables. In most cases, this step is only performed once. The second step, propagation, is where we propagate received evidence and make inference about variables using only the junction tree. The propagation is made by sending messages and since it is done on a tree structure, it is enough to have a two step message passing function. This makes it a fast algorithm, which is one of the main reasons for its popularity. Most of the algorithms found in this chapter originates from [HD94] and to some extent from [Jen96]. It has been shown that the evaluation technique using the junction tree algorithm is equivalent to previously known evaluation technique, which perform directly on the Bayesian network [SAS94].

5.1 Theory

In this section, the fundamentals of a junction tree and its potentials are introduced and explained. When converting a Bayes net into a junction tree with potentials it is first triangulated. The triangulation of a graph is closely related to the decomposition of a graph and the decomposition of a distribution. All these concepts will be explained.

5.1.1 Cluster trees

Before defining what a junction tree is the definition of a cluster tree, which is a super class of the junction trees, are introduced.

DEFINITION 9 (CLUSTER TREE)
A cluster tree *over* $\mathbf{X}_\mathcal{V}$ *is a tree of clusters of variables from* $\mathbf{X}_\mathcal{V}$*. The clusters are subsets of* $\mathcal{V}$ *such that their union is equal to* $\mathcal{V}$*. The edges between the cluster in the tree is labeled with the intersection between two clusters* A *and* B*, i.e.* $S_{AB} = A \cap B$*. These intersections are by definition separator sets.*

Given a Bayesian network, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ over $\mathbf{X}_\mathcal{V}$, any cluster tree corresponding to $\mathcal{G}$ is a representation of $P(\mathbf{X}_\mathcal{V})$. An example of a cluster tree can be seen in figure 5.1.

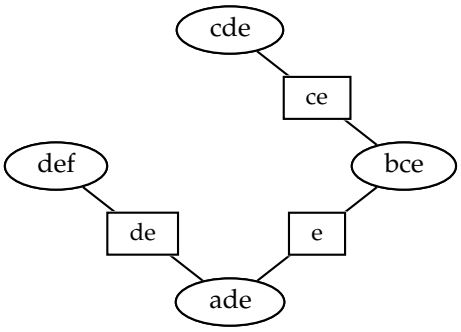


Figure 5.1: An example of a cluster tree. The clusters are displayed as ovals and the separators are displayed as squares.

5.1.2 Junction trees

A *junction tree* is a very special case of a cluster tree and it will be used often in this report. It is also called a *join tree*. This is the definition:

DEFINITION 10 (JUNCTION TREE)
A junction tree *over* $\mathbf{X}_\mathcal{V}$ *is a cluster tree such that all vertices between any two pair of vertices* A *and* B *contains the intersection* $A \cap B$*.*

In figure 5.2, an example of a junction tree is given. Compare this graph with the cluster tree in figure 5.1 which is not a junction tree since the cliques and sepsets along the path between clique *ade* and clique *cde* do not contain the intersection *de*.

5.1.3 Decomposition of graphs and probability distributions

DEFINITION 11 (GRAPH DECOMPOSITION)
Three disjoint subsets A*,* B *and* C *of an undirected graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *is said to form a* (weak) decomposition *of* $\mathcal{G}$ *if:*

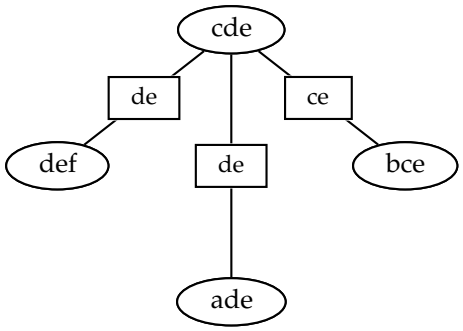


Figure 5.2: An example of a junction tree. The clusters are displayed as ovals and the separators are displayed as squares. Note that all vertices between any two clusters A and B contains the intersection $A \cap B$.

(i) $\mathcal{V} = A \cup B \cup C$,

(ii) C *separates* A *and* B, *and*

(iii) C *is a complete subset.*

There is also a definition of *strong* decomposition, but it is only of interest when there are continuous variables in the network, more information can be found in [Lau96]. In this report, only discrete variables are discussed, though. When we in the following sections write decomposition we refer to weak decomposition.

DEFINITION 12 (DECOMPOSABLE GRAPH)
A graph *that can be decomposed into cliques by (weak) decompositions is called a* (weakly) decomposable graph.

An obvious result from this definition is that any subgraph $\mathcal{G}_A$ of a decomposable graph $\mathcal{G}$ is also decomposable.

THEOREM 6 (TRIANGULATED AND DECOMPOSABLE)
Given an undirected graph $\mathcal{G}$, *the condition that* $\mathcal{G}$ *is (weakly) decomposable is equivalent to the condition that* $\mathcal{G}$ *is triangulated.*

PROOF The proof is based on induction on the number of vertices in the graph and it can be found in [Lau96]. □

DEFINITION 13 (DECOMPOSABLE DISTRIBUTION)
A probability distribution *is said to be* decomposable *with respect to* $\mathcal{G}$ *if* $\mathcal{G}$ *is an I-map and triangulated.*

In figure 5.3, two probability distributions represented by graphs, one decomposable and one non-decomposable, are given. The graph to the left is a non-decomposable probability distribution since b and c are not married, which implies that it is not triangulated. The right graph is a decomposable probability distribution though and $\{b, c\}$ separates $\{a\}$ and $\{d\}$.

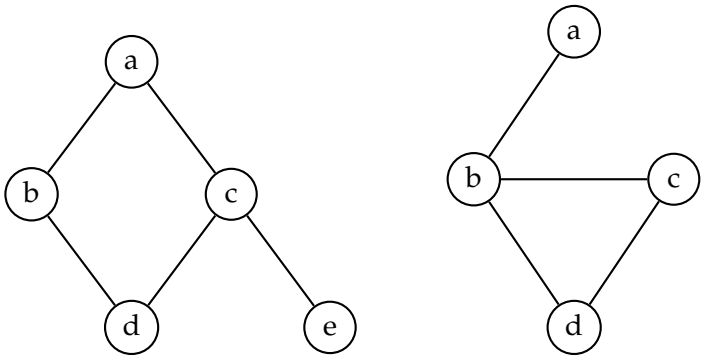


Figure 5.3: Left: The graph is a representation of a non-decomposable probability distribution since b and c are not married. In other words, it is not triangulated. Right: The graph is a representation of a decomposable probability distribution. $\{b, c\}$ separates $\{a\}$ and $\{d\}$.

5.1.4 Potentials

DEFINITION 14 (POTENTIAL)
A potential ϕ_A *over a set of variables* $\mathbf{X}_A$ *is a function that maps each instantiation of* $\mathbf{x}_A$ *into a non-negative real number. We denote the number that* ϕ_A *maps* $\mathbf{x}_A$ *into by* $\phi_A(\mathbf{x}_A)$.

How to represent a *discrete* potential on a computer system, see appendix A.

Operations on potentials

There are two basic operations on potentials, namely *marginalization* and *multiplication*. Suppose we have two clusters of random variables $\mathbf{X}_A$ and $\mathbf{X}_B$ where $\mathbf{X}_A \subseteq \mathbf{X}_B$. Then the marginalization of a potential and the multiplication of two potentials is defined as:

DEFINITION 15 (MARGINALIZATION OF A POTENTIAL)
The marginalization *of* ϕ_B *into* $\mathbf{X}_A$ *is a potential* ϕ_A*, and is denoted*

$$\phi_A = \sum_{\mathbf{X}_B \setminus \mathbf{X}_A} \phi_B.$$

Each $\phi_A(\mathbf{x}_A)$ *is computed as the sum* $\sum_i \phi_B(\mathbf{x}_{B,i})$ *where the* $\mathbf{x}_{B,i}$*:s are all instantiations of* $\mathbf{X}_B$ *such that they are consistent with* $\mathbf{x}_A$.

X_a	X_b	$\phi_A(\cdot)$
0	0	0.11
0	1	0.23
1	0	0.56
1	1	0.20

ϕ_A

X_a	X_b	X_c	$\phi_B(\cdot)$
0	0	0	0.02
0	0	1	0.23
0	1	0	0.11
0	1	1	0.10
1	0	0	0.09
1	0	1	0.13
1	1	0	0.31
1	1	1	0.01

ϕ_B

X_a	X_b	X_c	$\phi_C(\cdot)$
0	0	0	0.0022
0	0	1	0.0253
0	1	0	0.0253
0	1	1	0.0230
1	0	0	0.0504
1	0	1	0.0728
1	1	0	0.0620
1	1	1	0.0020

$\phi_C = \phi_A\phi_B$

Table 5.1: An example of a multiplication $\phi_C = \phi_A\phi_B$ between two potentials over two clusters $A = \{a, b\}$ and $B = \{a, b, c\}$.

DEFINITION 16 (MULTIPLICATION OF POTENTIALS)
The multiplication *of ϕ_A and ϕ_B is a potential ϕ_C, where $C = A \cup B$. It is denoted*

$$\phi_C = \phi_A\phi_B.$$

Each $\phi_C(\mathbf{x}_C)$ is computed as the multiplication $\phi_A(\mathbf{x}_A)\phi_B(\mathbf{x}_B)$ where both instantiation $\mathbf{x}_A$ and instantiation $\mathbf{x}_B$ are consistent with $\mathbf{x}_C$.

How to implement the multiplication algorithm is thoroughly discussed in appendix A.

As an example, suppose that we have a cluster $A = \{a, b\}$ and a cluster $B = \{a, b, c\}$ where each variable has two states; 0 and 1. The potential ϕ_A and ϕ_B can be seen as tables or multi-way arrays (see appendix A). The union set between A and B is equal to $C = \{a, b, c\}$ and the multiplication between $\phi_A(x_A)$ and $\phi_B(\mathbf{x}_B)$ is then a potential with $2 \times 2 \times 2$ state space. In table 5.1, a possible table representation of ϕ_A and ϕ_B is given together with the multiplication of them. Note that the potentials do *not* have to sum to one.

THEOREM 7 (POTENTIALS AND GLOBAL MARKOV PROPERTIES)
Having a potential representation and having global Markov properties are equivalent for all distributions on a graph, if and only if it is triangulated.

PROOF The proof for this can be found in [Rip96]. □

THEOREM 8 (FACTORIZATION OF POTENTIALS)
Given a decomposable probability distribution $P(\mathbf{X}_\mathcal{V})$ with respect to $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ it can be written as the product of all potentials of the cliques divided by the product of all potentials on

the separators:

$$P(\mathbf{X}_{\mathcal{V}}) = \frac{\prod_{C \in \mathcal{C}} \phi_C}{\prod_{S \in \mathcal{S}} \phi_S} \tag{5.1}$$

where $\mathcal{C}$ is the set of all cliques, $\mathcal{S}$ is the set of all separators, and ϕ_A is the potential over cluster A.

PROOF The proof for this can be found in [Rip96]. □

For more information on potentials, how they can be implemented, and methods and operations on them, see appendix A.

5.2 Transformation

The transformation of a Bayesian network into a junction tree is performed by doing a sequence of graphical operations. Do not misunderstand the word transformation, it means that a *new* graph (the junction tree) is created and that we still have the unchanged Bayesian network available. There are four main steps in this transformation from Bayesian network to junction tree:

ALGORITHM 1 (TRANSFORMATION)

1. *Transform the DAG of the Bayesian network into a* moral graph*, which is undirected.*
2. *Add arcs to the moral graph to form a* triangulated graph*.*
3. *Select* cliques *that are subsets of vertices in the triangulated graphs.*
4. *Using these cliques, create a* junction tree *by connecting the cliques with sepsets.*

Step 2 and 4 are NP-hard problems [HD94], but with some heuristic algorithms these steps can be completed with some efficiency. Since transformation into a junction tree is performed only once per Bayesian network, the speed is not that important if the network is moderate in size.

5.2.1 Moral graph

The first and also the less complicated step towards a triangulated graph (and later also a junction tree) is to moralize the Bayesian network. A *moral* graph is a graph where all parents with a common child have been connected with undirected edges and thereafter had all its directed edges made undirected. We say that the parents are *married*. Note that one parent can be married to many other vertices. In figure 5.4, to the left is the original directed graph and to the right is the moralized graph. Since vertex a and d are both parents to vertex b, the edge $a - d$ is added. The three edges $b - c$, $b - e$ and $c - e$ are added because vertex b, c and e are all parents to vertex f. In the figure added edges are dotted.
The algorithm for constructing a moral graph $\mathcal{G}_{\mathcal{M}}$ from a DAG $\mathcal{G}$ is

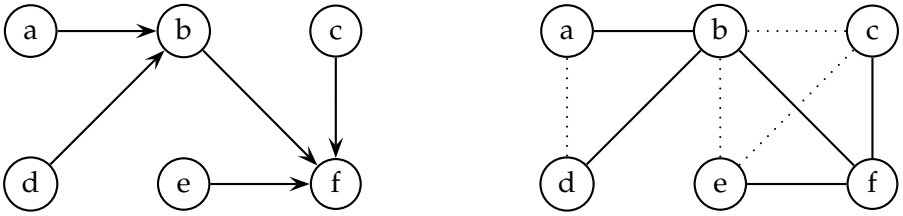


Figure 5.4: Left: The directed acyclic graph to be moralized. Right: The moralized graph. Dotted edges are the edges added during the moralization.

ALGORITHM 2 (MORALIZATION OF A DIRECTED ACYCLIC GRAPH)

1. *Create a copy $\mathcal{G}_{\mathcal{M}}$ of $\mathcal{G}$.*
2. *For each vertex u do*
   (a) *for each pair of vertices in the parent set of u, $pa(u)$, add an undirected graph to $\mathcal{G}_{\mathcal{M}}$.*
3. *Make $\mathcal{G}_{\mathcal{M}}$ undirected by dropping the directions of all the edges.*

An important observation is that in step 1 we create a *copy* of the Bayesian network. From this step we are working on a new graph without destroying the original Bayesian network.

5.2.2 Triangulated graph

An undirected graph is *triangulated* if all cycles of length larger than three has a chord. In the process of triangulation the *elimination of vertices* is a central issue. An *elimination* of a vertex u in an undirected graph, is done by first connecting all of u's neighbors pairwise, and then remove u and all edges connected to it. We say that u is *eliminated*. See figure 5.5 for an example of elimination. When we remove vertex e, all its unconnected neighbors, vertex a, c and d, are connected by adding edges. In this case only the edges $a - c$ and $a - d$ where added, since the edge $c - d$ already existed.
It is guaranteed that a new complete subgraph is created when a vertex is eliminated. We say that a clique is induced from the process of elimination. In figure 5.5, the elimination of vertex e induces the clique $\{a, c, d\}$. A vertex u that can be eliminated without having edges added, can not be part of a chord-less cycle of length greater than three. In general, there are many ways to triangulate an undirected graph. If we have many ways to choose from we would like to get an, in some sense, optimal triangulation. One way to define optimal is to define it as the resulting graph has a minimal sum of the sizes of the potentials' state spaces[1]. This task is NP-complete [Jen94, HD94]. However, there exist several *heuristic* algorithms that produce at least close to optimal triangulation graphs in real-world settings. [Kjæ90] investigate several heuristic methods. One

[1]The size of a potential's state space is equal to the product of the sizes of all state spaces of each variable's distribution.

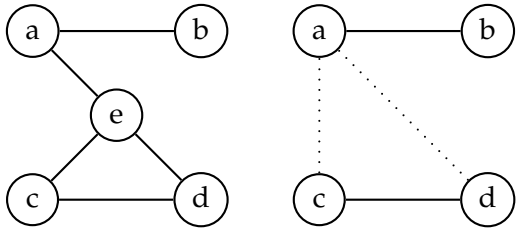


Figure 5.5: Example of elimination of a vertex. Vertex e is eliminated and the dotted edges are the edges added during the process of elimination. The resulting complete subgraph $\{a, c, d\}$ is said to be induced from the process of elimination.

such algorithm is explicitly given by [HD94] and is a modification from [Kjæ90].

The algorithm for triangulating any undirected graph is

ALGORITHM 3 (TRIANGULATION OF AN UNDIRECTED GRAPH)

1. *Create a copy $\mathcal{G}'_{\mathcal{M}}$ of $\mathcal{G}_{\mathcal{M}}$.*
2. *As long as $\mathcal{G}'_{\mathcal{M}}$ has vertices left*
   (a) *Select the vertex u from $\mathcal{G}'_{\mathcal{M}}$ that causes the least number of edges to be added if it is removed. If choice is between two or more vertices, select the one that induces the cluster with the smallest weight.*
   (b) *Connect all vertices in the cluster that is induced by selected vertex u. Add the corresponding edges to $\mathcal{G}_{\mathcal{M}}$.*
   (c) *Save each induced cluster that is not a subset of any previously saved cluster.*
   (d) *Remove u from $\mathcal{G}'_{\mathcal{M}}$.*
3. *$\mathcal{G}_{\mathcal{M}}$, modified by the additional edges introduced in the previous steps, is now triangulated.*

In the criterion found in step 2a the concept of *weight* is used. This is how it is defined:

- The *weight* of a vertex u is defined as the number of states the corresponding variable X_u has. Denote it $w(u)$ or $w(X_u)$.
- The *weight* of a cluster is the product of the weights of its vertices. Denote it $w(A)$ or $w(\mathbf{X}_A)$, it is calculated as $w(A) = \prod_{u \in A} w(u)$.

From THEOREM 6 we get that a triangulated graph is decomposable. The problem is to identify the cliques that decompose the graph, but in step 2c this is what is done. There are two reasons why this works [HD94]; (i) An induced cluster can never be a subset of a subsequently induced cluster. (ii) Each clique in the triangulated graph is an induced cluster from step 2b in the triangulation algorithm.

5.2.3 Junction tree

The step towards the junction tree is about half through now. What we have are the vertices in the junction tree, which are the saved cliques from step 2c. What needs to be done is to connect these cliques so that it satisfies the junction tree definition (see DEFINITION 10). It would also be nice if we could build it in such a way that the calculation of probabilistic inference would be as fast as possible. An approach that fulfills this is once again given by [HD94]. There exists other more sophisticated approaches ([Sha99]) but they will not be discussed further.

ALGORITHM 4 (BUILDING A SEMI-OPTIMAL[2] JUNCTION TREE)

1. *Create a set of n trees, each containing of a single clique. The n cliques are the ones saved in step 2c in the triangulation algorithm. Create an empty set $\mathcal{S}$ to be used for storing sepsets.*
2. *For each distinct pair of cliques A and B:*
   (a) *Create a candidate sepset, S_{AB}, containing the intersection $A \cap B$.*
   (b) *Insert S_{AB} into $\mathcal{S}$. as we will see in chapter 5.*
3. *Repeat until $n-1$ sepsets have been inserted into the forest.*
   (a) *Select and remove the sepset S_{AB} from $\mathcal{S}$ that has the largest mass. If two sepsets have equal mass, select the one with the smallest cost.*
   (b) *If the cliques A and B are on different trees in the forest, then join the two by inserting S_{AB} between them.*
4. *The forest contains of one (cumulated) tree, which is the wanted junction tree.*

In the criterion found in step 3a the concepts of *mass* and *cost* is used. This is how it is defined:

- The *mass* of a sepset S_{AB}, denoted $m(S_{AB})$, is the number of variables it contains.
- The *cost* of a sepset S_{AB}, denoted $c(S_{AB})$, is the weight of A plus the weight of B, i.e. $c(S_{AB}) = w(A) + w(B)$.

Examining the algorithm we see that there will be $n(n-1)/2$ distinct pairs of cliques in step 2. In step 3b an insertion of a sepset will merge the two trees A and B belong to into one larger tree. In other words, each time an insertion is made, we get one less tree. If we not make sure that A and B belongs to different trees a cycle may be formed.

5.3 An example - The Year 2000 risk analysis

An example will show the steps from a Bayesian network into a junction tree. The network used in this example is taken from HUGIN Expert A/S web page [HEA99] and is called "*Year 2000*". Mike Waters[3] made it as part of a class project on "Reasoning under uncertainty", created it.

[3] I could not find his affiliation; I just have his name.

5.3.1 The Bayesian network model

The Bayes net models the year 2000 problems in US stock market, finance, transportation, and utilities industries. It was basically designed to predict the liability of the US stock market conditioned on different scenarios. See figure 5.6. The variables in the network have states like (*working*, *reduced*, *not working*) or (*working*, *moderate*, *severe*, *failure*). The $X_{USStocks}$ variable has the states (*up*, *down*, *crash*).

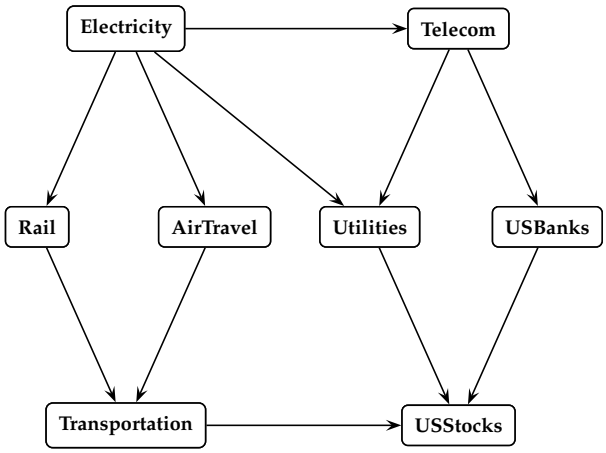


Figure 5.6: The Bayesian network "Year 2000" which is going to be moralized and triangulated, and then the junction tree will be created.

5.3.2 Moralization

The moralization of the "Year 2000" network will add three new edges between mates, namely *Rail-AirTravel*, *Transportation-Utilities*, and *Transportation-USBanks*. In figure 5.7 the moralized graph is shown, where the new inserted edges are dotted. All other edges are made *undirected*. The moralization algorithm (see ALGORITHM 2) is fully deterministic, i.e. there exists a unique solution which is easy to find.

5.3.3 Triangulation

The next step towards the junction tree is the triangulation step. Remind that the triangulation of a graph can be done in many different ways resulting in different solutions. Which solutions you select depend on what you want to optimize for. We select to optimize for size, where we want to have as small clusters as possible in the resulting junction tree. The step is NP-hard. Using ALGORITHM 3 we end up with the graph found in figure 5.8. Added edges are dotted and they are *Electricity-Transportation* and *Electricity-USBanks*. When the vertex *Telecom*were eliminated, all its neighbors were connected. Since all but *Electricity*and *Transportation*already were connected, only the edge *Electricity-Transportation* was added. The edge *Electricity-Transportation* was added when *AirTravel*was eliminated. In table 5.2, the eliminations of all eight vertices is shown.

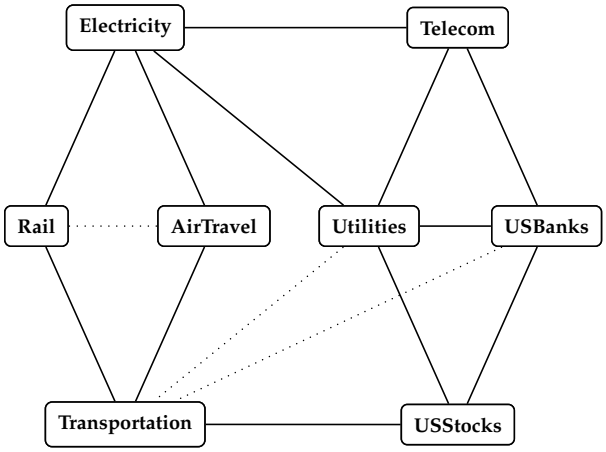


Figure 5.7: The moralized graph of the Bayesian network "Year 2000". Added edges are dotted.

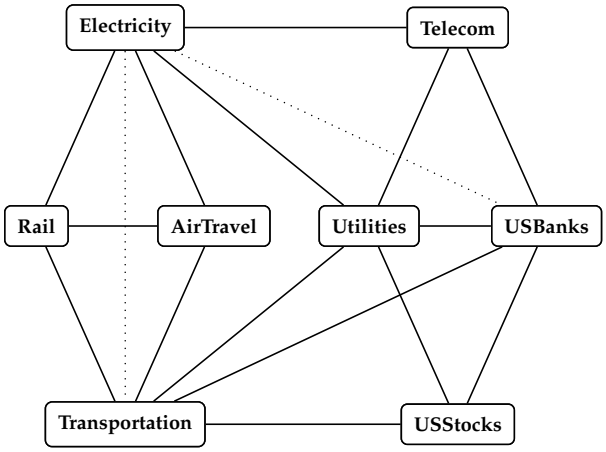


Figure 5.8: The triangulated graph of the Bayesian network "Year 2000". Added edges during triangulation process are dotted.

5.3.4 Building the junction tree

During the process of triangulation (see ALGORITHM 4) we saved the induced cluster (from the process of elimination) that was not a subset of previously saved clusters. See step 2c in ALGORITHM 3. The clusters saved are the one marked by an asterisk (*) in the column of induced clusters that are listed in table 5.2. In the table are also the vertices eliminated and the edges added shown in order or elimination.

Step	Eliminated vertex	Added edges	Induced cluster
1	*USStocks*	-	{*USBanks*, *Utilities*, *Transportation*, *USStocks*}*
2	*Telecom*	{*Electricity-USBanks*}	{*Electricity*, *USBanks*, *Utilities*, *Telecom*}*
3	*AirTravel*	{*Electricity-Transportation*}	{*Electricity*, *Transportation*, *Rail*, *AirTravel*}*
4	*Rail*	-	{*Electricity*, *Transportation*, *Rail*}
5	*Electricity*	-	{*Utilities*, *USBanks*, *Transportation*, *Electricity*}*
6	*Transportation*	-	{*USBanks*, *Utilities*, *Transportation*}
7	*USBanks*	-	{*Utilities*, *USBanks*}
8	*Utilities*	-	{*Utilities*}

Table 5.2: The elimination ordering showing eliminated vertices, added edges and induced clusters during the triangulation process of the network "Year 2000". The saved clusters for the junction tree are the clusters marked by an asterisk (*).

In figure 5.9 is the resulting junction tree shown. The four clusters saved are connected with separator sets according to ALGORITHM 4. The next step is to assign the clusters potential functions. This step is called the initialization step.

5.4 Initializing the network

The structure of the junction tree is now set. Do not forget that we still have the original Bayesian network, i.e. the junction tree is a separate graph. Left to do is the quantitative part of the transformation. The goal of this second half is to have the probability distribution inserted into the tree structure so the global Markov property is conserved.

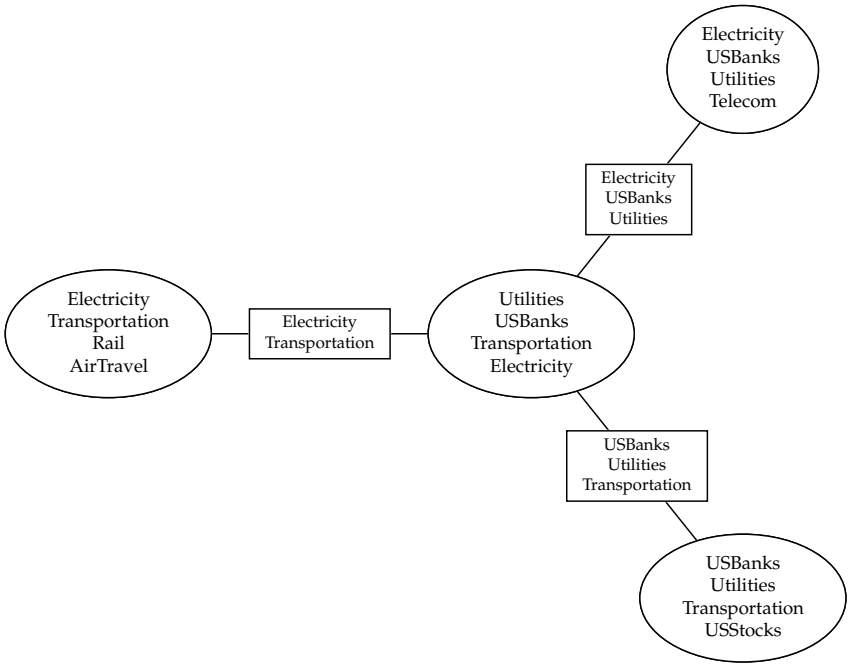


Figure 5.9: The junction tree of the Bayesian network "Year 2000" with cluster vertices (ovals) containing the cliques found during the triangulation process. Between the clusters are edges labeled with sepsets (squares) inserted.

5.4.1 Initializing the potentials

From chapter 3 we know that

$$P(\mathbf{X}_{\mathcal{V}}) = \prod_{u \in \mathcal{V}} P(X_u | X_{pa(u)})$$

where $P(X_u | X_{pa(u)}) = P(X_u)$ if u is the root, is a potential representation of the distribution $P(\mathbf{X}_{\mathcal{V}})$. Now we can construct the following algorithm:

ALGORITHM 5 (INITIALIZING THE JUNCTION TREE)

1. *For each cluster and sepset A, do the assignment*

$$\phi_A \leftarrow 1$$

2. *For each vertex u, assign to u a cluster A that contains $fa(u)$ ($A \supseteq fa(u)$) and call A the* parent cluster of $fa(u)$. *Then include the conditional probability $P(X_u | \mathbf{X}_{pa(u)})$ (or just $P(X_u)$ if there are no parents) into ϕ_A according to*

$$\phi_A \leftarrow \phi_A \cdot P(X_u | \mathbf{X}_{pa(u)})$$

For future needs it is useful, but not necessary, to remember to which vertex each parent cluster in step 2 was assigned. We will return to this in section 5.6.3 when discussing ALGORITHM 12.

This initialization procedure satisfies equation (5.1);

$$\frac{\prod_{C\in\mathcal{C}}\phi_C}{\prod_{S\in\mathcal{S}}\phi_S} = \frac{\prod_{u\in\mathcal{V}}P(X_u|\mathbf{X}_{pa(u)})}{1} = P(\mathbf{X}_\mathcal{V}).$$

5.4.2 Making the junction tree locally consistent

The junction tree potentials does indeed satisfy the equation (5.1), but it might not be consistent. Currently, comparing cliques they might tell you different stories about the distributions on certain variables. For instance, consider two cliques A and B, which are connected with a sepset S_{AB}, and both contains the variable X_a. Marginalizing on both $\mathbf{X}_A$ and $\mathbf{X}_B$ to get X_a, might not give the same result. The reason for this is that the individual cliques have not yet been affected by the information of other cliques in the tree. The information in the tree has to be distributed "equally" on. The *global propagation* algorithm performs a series of local manipulations that makes the junction tree locally consistent. The local manipulations is referred to as *message passing* which can be seen as an update message sent from clique to another filtered through their common sepset. The global propagation algorithm keeps equation (5.1) valid. The idea behind the propagation is to have all cliques to pass a message to all of its neighbors. The order in which this is done is such that the consistency from previously passed messages will be preserved. After the performance each cluster-sepset pair will be consistent, and therefore also the whole tree.

Message passing

A *single message pass* between two clusters A and B consist of two steps. The first step, called *projection*, passes the message from A to the sepset S_{AB}. The second step, called *absorption*, passes the message from the sepset S_{AB} to B. See figure 5.10.

ALGORITHM 6 (PASS-MESSAGE(A,B))

1. PROJECTION. *Update the potential of the sepset S_{AB}:*

$$\phi_{S_{AB}}^{old} \leftarrow \phi_{S_{AB}}$$
$$\phi_{S_{AB}} \leftarrow \sum_{A\setminus S_{AB}} \phi_A$$

2. ABSORPTION. *Update the potential of the receiving cluster ϕ_B:*

$$\phi_B \leftarrow \phi_B^{old} \cdot \frac{\phi_{S_{AB}}}{\phi_{S_{AB}}^{old}}$$

If $\phi_{S_{AB}}^{old} = 0$ then $\phi_B \leftarrow 0$.

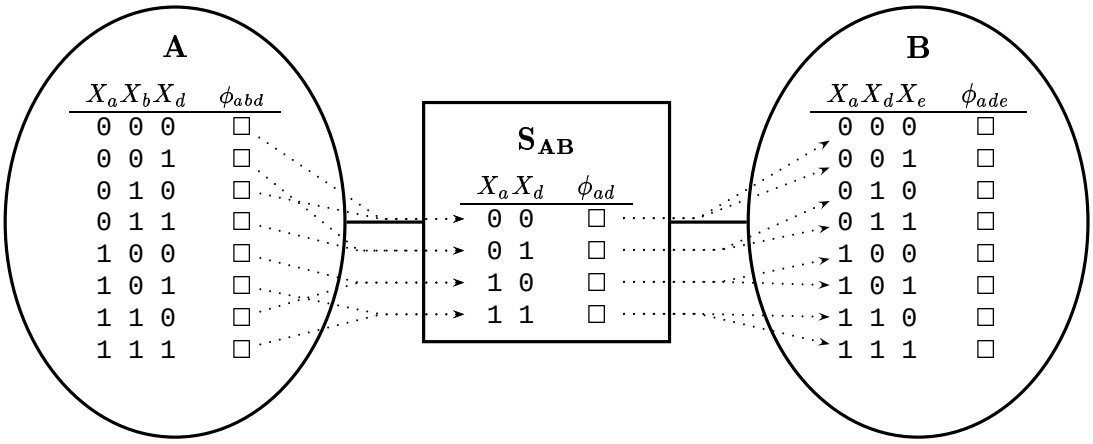


Figure 5.10: A single message pass from cluster $A = \{a, b, d\}$ to cluster $B = \{a, d, e\}$. All variables are binary. The potential ϕ_S of the sepset S_{AB} is updated through a projection from the potential from cluster A. Then the potential of cluster B absorbs the new information from the sepset. These steps are described by ALGORITHM 7.

A division between two identically shaped potentials is performed elementwise. If the division is $0/0$ it is defined to be 0. For $x/0$ where $x \neq 0$ the division is undefined. But, for any instantiation $\mathbf{s}_{AB}$ of $\mathbf{S}_{AB}$, Jensen [HD94, Jen96] shows that whenever $\phi^{old}_{S_{AB}} = 0$ then $\phi_{S_{AB}} = 0$ also holds. This is the reason for the if-condition in ABSORPTION. Because of the way division is defined for potentials, the division in the ABSORPTION-method should be performed *before* the multiplication (with the potential).

In practice, it is not unusual that the values of some elements in a potential are very small. Performing a message pass might then vanish (put to zero) some of the elements because of the limited precision a value can have in a computer[4]. To prevent this from happening we can normalize the potential during the *absorption* step. Here is the revised algorithm (with modification according to [Jen94]):

ALGORITHM 7 (PASS-MESSAGE(A,B) revised)

1. PROJECTION. *Update the potential of the sepset S_{AB}:*

$$\phi^{old}_{S_{AB}} \leftarrow \phi_{S_{AB}}$$

$$\phi_{S_{AB}} \leftarrow \sum_{A \setminus S_{AB}} \phi_A$$

2. ABSORPTION. *Update the potential of the receiving cluster ϕ_B:*

$$\mu \leftarrow \sum \phi_{S_{AB}}$$

$$\phi_B \leftarrow \phi^{old}_B \cdot \frac{\mu^{-1} \cdot \phi_{S_{AB}}}{\phi^{old}_{S_{AB}}}$$

[4]Most programming languages follow the IEEE 754 standard for floating-point arithmetic; the smallest number that can be represented in single precision is $2^{-126} \approx 10^{-38}$ [Jen94].

If $\phi^{old}_{S_{AB}} = 0$ *then* $\phi_B \leftarrow 0$.

Global propagation

The global message passing, called *global propagation*, will make sure that each cluster passes its information, i.e. its belief potential, to all of the other clusters in the junction tree. Paying attention to the collect evidence method we see that a cluster passes a message to a neighbor first when it has received evidence from all of its other neighbors.

ALGORITHM 8 (GLOBAL PROPAGATION)

1. *Choose an arbitrary cluster A:*
   (a) *Unmark all clusters. Call* COLLECT-EVIDENCE(A,NIL).
   (b) *Unmark all clusters. Call* DISTRIBUTE-EVIDENCE(A).

ALGORITHM 9 (COLLECT-EVIDENCE(A,C))

1. *Mark A.*
2. *while there are unmarked* $B_i \in ne(A)$*:*
   Call COLLECT-EVIDENCE(B_i,A) *(recursively)*
3. *Call* PASS-MESSAGE(A,C) *back to the invoking cluster C if* $C \neq$ NIL.

Comment: Note that the cluster C in the COLLECT-EVIDENCE(A,C) is the cluster that invokes COLLECT-EVIDENCE.

ALGORITHM 10 (DISTRIBUTE-EVIDENCE(A))

1. *Mark A.*
2. *while there are unmarked* $B_i \in ne(A)$*:*
   Call PASS-MESSAGE(A,B_i)
3. *while there are unmarked* $B_i \in ne(A)$*:*
   Call DISTRIBUTE-EVIDENCE(B_i) *(recursively)*

5.4.3 Marginalizing

After having converted the Bayesian network into a junction tree and the initialized it, we have a consistent structure from which we can marginalize out any variables. Consider the potential ϕ_{X_B} in table 5.1. Assume that this potential is a result from a global propagation. Then it represents the joint probability distribution over the cluster $B = \{a, b, c\}$. To get the probability distribution for X_c, $P(X_c)$, one has to sum over $\mathbf{X}_{\{a,b\}}$. Using the values of ϕ_B found in table 5.1 the probability distribution for X_c will become

$$P(X_c) = \sum_{\mathbf{X}_B \setminus X_c} \phi_B = (0.53, 0.47).$$

5.5 The Year 2000 example continued

Before we saw how the "Year 2000" network was transformed into a junction tree. In this section it will be shown how its potentials are initialized and made globally consistent with each other.

5.5.1 Initializing the potentials

After have been building a junction tree its quantitative part has to be initialized. In this example the initialization of the clique {*Electricity*, *Transportation*, *Rail*, *AirTravel*} and the sepset {*Electricity*, *Transportation*} will be presented. See figure 5.11 or figure 5.9.

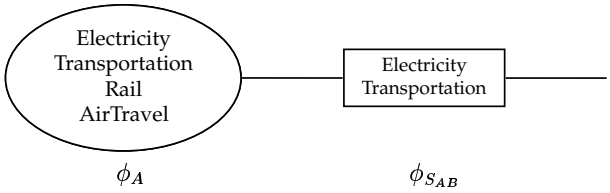


Figure 5.11: The clique {*Electricity*, *Transportation*, *Rail*, *AirTravel*} and its sepset {*Electricity*, *Transportation*} whose potentials ϕ_A and $\phi_{S_{AB}}$, respectively, will be initialized.

The initialization algorithm (see ALGORITHM 5) does first assign 1 to all cliques and sepsets in the junction tree. This is actually all the initialization that is done for the sepsets. The potential of sepset {*Electricity*, *Transportation*} is shown in table 5.3.

$X_{Electricity}$	$X_{Transportation}$	$\phi_{S_{AB}}(\mathbf{x})$
working	working	1
working	moderate	1
working	severe	1
working	failure	1
reduced	working	1
reduced	moderate	1

$X_{Electricity}$	$X_{Transportation}$	$\phi_{S_{AB}}(\mathbf{x})$
reduced	severe	1
reduced	failure	1
not working	working	1
not working	moderate	1
not working	severe	1
not working	failure	1

Table 5.3: The potential $\phi_{S_{AB}}$ of sepset {*Electricity*, *Transportation*} after initialization. It has $3 \times 4 = 12$ states and all states will have the value one.

Next, the algorithm steps through all variables X_u in the Bayesian network and assigns to u a clique A, called the parent cluster, that contains $fa(u)$. These are some of the interesting variables and their family:

$$
\begin{aligned}
fa(Electricity) &= \{Electricity\} \\
&\dots \\
fa(Rail) &= \{Electricity, Rail\} \\
fa(AirTravel) &= \{Electricity, AirTravel\} \\
&\dots \\
fa(Transportation) &= \{Transportation, Rail, AirTravel\} \\
&\dots
\end{aligned}
$$

Sometimes there is more than one clique A such that $A \supseteq fa(u)$. Which clique that is selected as the parent cluster does not matter, the result and the performance will be the same. In this example, we select the parent cluster for the variable $X_{Electricity}$ to be the clique {*Electricity*, *USBanks*, *Utilities*, *Telecom*} (see figure 5.9). The variables X_{Rail}, $X_{AirTravel}$ and $X_{Transportation}$ are all assigned the same parent cluster, namely {*Electricity*, *Transportation*, *Rail*, *AirTravel*} (see figure 5.11 and figure 5.9). For this reason, the potential over clique {*Electricity*, *Transportation*, *Rail*, *AirTravel*}, let us call it clique A and its potential ϕ_A, will be initiated according to:

$$\phi_A \leftarrow \mathbf{1} \cdot P(X_{Transportation}|\mathbf{X}_{pa(Transportation)}) \cdot P(X_{Rail}|\mathbf{X}_{pa(Rail)}) \cdot P(X_{AirTravel}|\mathbf{X}_{pa(AirTravel)})$$

or, explicitly

$$\phi_A \leftarrow \mathbf{1} \cdot P(X_{Transportation}|\mathbf{X}_{\{Rail,AirTravel\}}) \cdot P(X_{Rail}|X_{Electricity}) \cdot P(X_{AirTravel}|X_{Electricity}).$$

Remember that on a computer all factors in the multiplication are multidimensional arrays (see appendix A). Since the variables $X_{Electricity}$, X_{Rail} and $X_{AirTravel}$ all have three states (*working*, *reduced*, *not working*) and variable $X_{Transportation}$ has four states (*working*, *moderate*, *severe*, *failure*), the potential will have a state space containing $3 \times 4 \times 3 \times 3 = 108$ states. In table 5.4 it is shown how some of the states in the potential is initialized. In appendix C.2, the Bayesian-Network model in XBN-format (see appendix B) is shown. There are also the individual values of the different conditional probability distributions that are used in the multiplication, found. $P(X_{AirTravel}|X_{Electricity})$ can be found on lines 100-110 in the XBN file, $P(X_{Rail}|X_{Electricity})$ on lines 112-122, and $P(X_{Transportation}|\mathbf{X}_{\{Rail,AirTravel\}})$ on lines 227-246.

$X_{Electricity}$	$X_{Transportation}$	X_{Rail}	$X_{AirTravel}$	$\phi_A(\mathbf{x})$
working	working	working	working	$1 \cdot 1.0 \cdot 0.7 \cdot 0.6 = 0.420$
working	working	working	reduced	$1 \cdot 0.7 \cdot 0.7 \cdot 0.3 = 0.247$
working	working	working	not working	$1 \cdot 0.5 \cdot 0.7 \cdot 0.1 = 0.035$
working	working	reduced	working	$1 \cdot 0.7 \cdot 0.2 \cdot 0.6 = .0840$
⋮	⋮	⋮	⋮	⋮
reduced	moderate	not working	reduced	$1 \cdot 0.2 \cdot 0.2 \cdot 0.4 = 0.016$
reduced	moderate	not working	not working	$1 \cdot 0.0 \cdot 0.2 \cdot 0.3 = 0.000$
reduced	severe	working	working	$1 \cdot 0.0 \cdot 0.5 \cdot 0.3 = 0.000$
reduced	severe	working	reduced	$1 \cdot 0.0 \cdot 0.5 \cdot 0.4 = 0.000$
⋮	⋮	⋮	⋮	⋮
not working	failure	reduced	not working	$1 \cdot 0.1 \cdot 0.2 \cdot 0.7 = 0.014$
not working	failure	not working	working	$1 \cdot 0.0 \cdot 0.7 \cdot 0.0 = 0.000$
not working	failure	not working	reduced	$1 \cdot 0.1 \cdot 0.7 \cdot 0.3 = 0.021$
not working	failure	not working	not working	$1 \cdot 0.9 \cdot 0.7 \cdot 0.7 = 0.441$

Table 5.4: An excerpt of the table showing the initialization of the potential over clique {*Electricity*, *Transportation*, *Rail*, *AirTravel*}. The whole potential has 108 states and only few of them are shown here.

5.5.2 Making the junction tree consistent

The next after the initial assignment of potentials in to make the junction tree globally consistent. The global message-passing algorithm (see ALGORITHM 8) does this. Using a arbitrary clique A as a start clique, a series of COLLECT-EVIDENCE and DISTRIBUTE-EVIDENCE calls are performed. Let us select the clique {*USBanks*, *Utilities*, *Transportation*, *USStocks*} as a start clique. The messages passed while performing these methods are shown in figure 5.12. The solid arrows represent the message passes during the collect-evidence phase and dotted arrows represent the message passes during the distribute-evidence phase. They are numbered in the order they are performed.

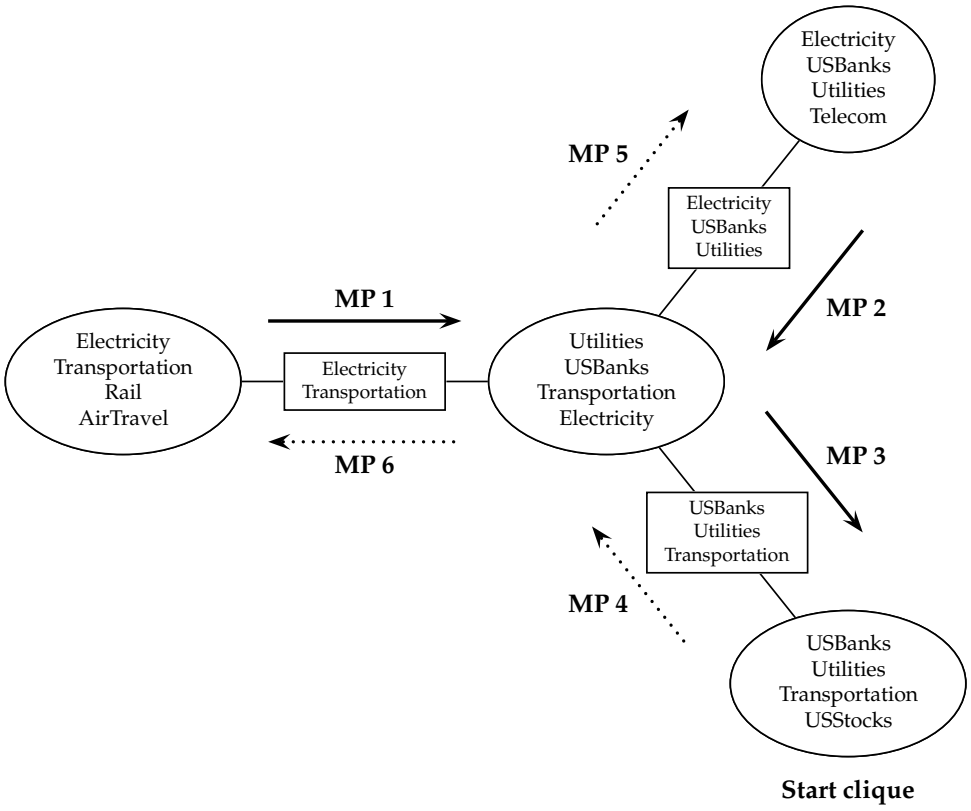


Figure 5.12: All the messages passed during the global propagation. The arrows represent the message passes, PASS-MESSAGE(A,B), in the direction from cluster A to B. They are numbered in the order they are performed. The solid arrows are called during the COLLECT-EVIDENCE and the dotted arrows are called during the DISTRIBUTE-EVIDENCE calls.

To make things more clear, we log all calls to method COLLECT-EVIDENCE and method DISTRIBUTE-EVIDENCE. The result is the log shown in table 5.5.

5.5.3 Calculation the a priori distribution

After the whole transformation step has been performed and the initialization of the potentials has been done, we can through marginalization calculate the a priori distribution for all variables. See figure 5.13. For the result, it does not matter which clique

Step	Method called
1	COLLECT-EVIDENCE($\{USBanks, Utilities, Transportation, USStocks\}$)
2	COLLECT-EVIDENCE($\{Utilities, USBanks, Transportation, Electricity\}$)
3	COLLECT-EVIDENCE($\{Electricity, USBanks, Utilities, Telecom\}$)
4	COLLECT-EVIDENCE($\{Electricity, Transportation, Rail, AirTravel\}$)
5	DISTRIBUTE-EVIDENCE($\{USBanks, Utilities, Transportation, USStocks\}$)
6	DISTRIBUTE-EVIDENCE($\{Utilities, USBanks, Transportation, Electricity\}$)
7	DISTRIBUTE-EVIDENCE($\{Electricity, USBanks, Utilities, Telecom\}$)
8	DISTRIBUTE-EVIDENCE($\{Electricity, Transportation, Rail, AirTravel\}$)

Table 5.5: The log of all calls to COLLECT-EVIDENCE and DISTRIBUTE-EVIDENCE when performing a global propagation with the clique {*USBanks*, *Utilities*, *Transportation*, *USStocks*} as the start clique.

or sepset we choose for getting a distribution as long as it contains the variable of interest. But for best performance, it is advisable to select the sepset or clique that has the smallest state space, i.e. the smallest number of variables.

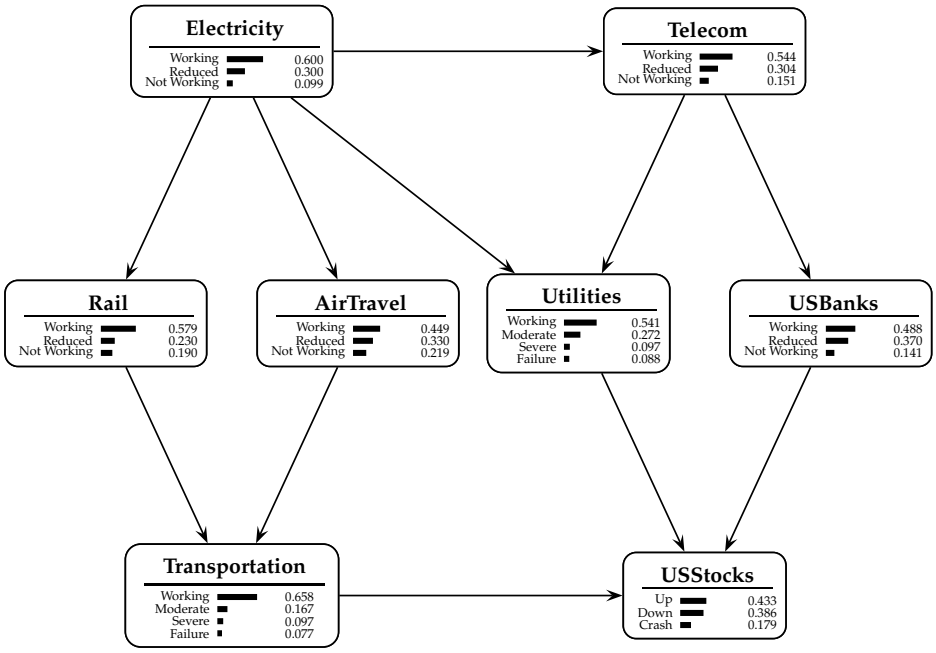


Figure 5.13: The a priori distribution over the variables in the model "Year 2000" after initialization with global propagation.

5.6 Evidence

In the first chapter *evidence* was introduced and was further discussed in section 4.2.1. To summarize and add some comments about what evidence in the Bayesian network

sence is we say that evidence is when we receive new information about a variable that changes our belief about its distribution over possible states. Evidence can be either *soft* or *hard*. *Hard evidence* is when we *instantiate* the variable, i.e. when we know exactly which state the variable is in. A hard evidence is also referred to as an *observation* and is a statement of the form $X_u = x_u$. Remember that we are only talking about discrete variables. *Soft evidence* is all other kinds of evidence, i.e. when we change our belief about the state distribution, but will still do not know exactly which state the variable is in. Soft evidence on a variable X_u can also be when we have observed (hard evidence) one of its children, but not the variable itself. This report will not discuss soft evidence any further, only hard evidence will be used.

5.6.1 Evidence encoded as a likelihood

Mathematically we code evidence as a *likelihood* function over a variable X_u, denoted as $\lambda_{X_u}(x_u)$. The likelihood function for a discrete variable X_u is a potential over the variable. If we have *observed* the variable the likelihood function is:

$$\lambda_{X_u}(x_u) = \begin{cases} 1, & \text{when } x_u \text{ is the observed value of } X_u \\ 0, & \text{otherwise.} \end{cases}$$

When a variable is unobserved, set the likelihood function to a constant; $\lambda_{X_u}(x_u) \leftarrow \frac{1}{n}$ for all x_u where n is the number of states in X_u. When implementing, it is enough to assign *one* to the likelihood function; $\lambda_{X_u}(x_u) \leftarrow 1$. This is because a normalization of all potentials will be performed during the global propagation.

5.6.2 Initialization with observations

We change our initialization algorithm of the junction tree method to include the setup of likelihood functions. There is only one minor modification.

ALGORITHM 11 (INITIALIZING THE JUNCTION TREE revised)

1. *For each cluster and sepset A, do the assignment*

$$\phi_A \leftarrow 1$$

2. *For each vertex u:*

   (a) *Assign to u a cluster A that contains $fa(u)$ and call A the* parent cluster of *$fa(u)$. Then include the conditional probability $P(X_u|\mathbf{X}_{pa(u)})$ (or just $P(X_u)$ if there are no parents) into ϕ_A according to*

$$\phi_A \leftarrow \phi_A \cdot P(X_u|\mathbf{X}_{pa(u)})$$

   (b) *Set each likelihood element to one:*

$$\lambda_{X_u}(x_u) \leftarrow 1$$

5.6.3 Entering observations into the network

When we receive new information in form of observations we change the likelihood function for the variables concerned. If the observed variables are X_a and X_c, we update the likelihood functions $\lambda_{X_a}(x_a)$ and $\lambda_{X_c}(x_c)$. Thereafter the potentials are updated using global propagation, see ALGORITHM 8 in section 5.4.2. The algorithm for entering new observations into the model is:

ALGORITHM 12 (OBSERVATION-ENTRY)

1. *For each observation $X_u = x_u$:*

   (a) *Encode the observation as a likelihood $\lambda_{X_u}^{\text{new}}$.*

   (b) *Identify a cluster A that contains u and update ϕ_A and λ_{X_u} as:*

$$\phi_A \leftarrow \phi_A \lambda_{X_u}^{\text{new}}$$
$$\lambda_{X_u} \leftarrow \lambda_{X_u}^{\text{new}}$$

Note that the cluster referred to as a A in step 1(b) can be chosen in many ways if there are several clusters containing the vertex u. It is suggested to use the *parent cluster* that was chosen in step 2 in ALGORITHM 5.

5.7 The Year 2000 example continued

When the second millennium will set and the new millennium will rise, people have suggested several different scenarios to happen. The most pessimistic ones think that the world will end and the most optimistic ones think that there will be peace all over the World. This report will not state any of those extremes. Instead we will look at what might happen if the electricity and/or the telecommunication would be reduced, go down or work as normal. Let us see what happens in the following three cases:

I. Both the electricity and the telecommunication work.

II. Electricity will be reduced and the telecommunication does not work.

III. Both the electricity and the telecommunication do not work.

5.7.1 Scenario I

I think the most likely scenario is when both the electricity and the telecommunication works as normal. Using the `hb`$^{\texttt{BN}}$ tool with the observations $X_{Electricity}$ = *Working* and $X_{Telecom}$ =*Working* we get that US Stocks will most probably go up, and that the transportation sector, the banks and the utilities will all have a big chance of working. See figure 5.14.

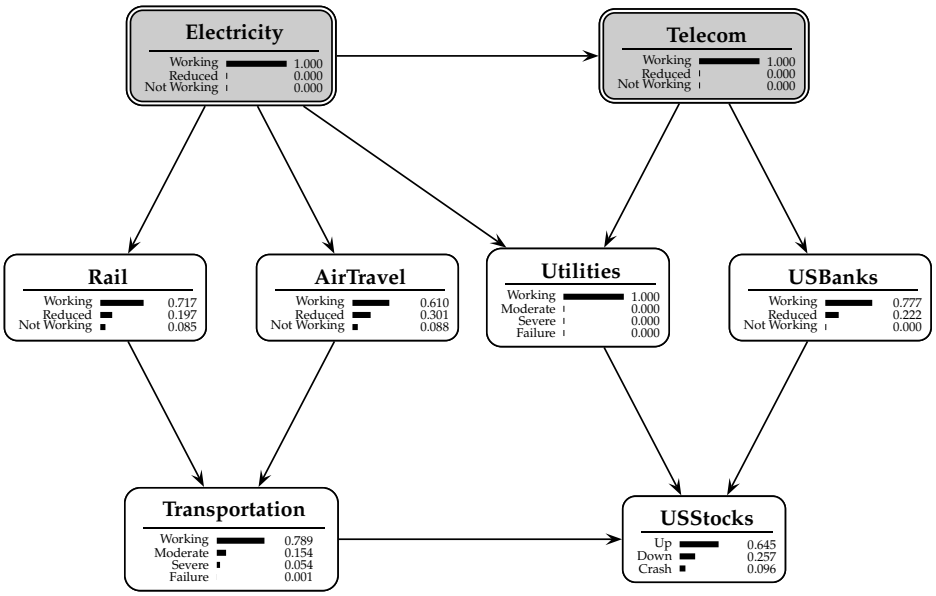


Figure 5.14: Scenario I: Both the electricity and the telecommunication works.

5.7.2 Scenario II

It is also reasonable to believe that there might be some reduction in the power production or the telecommunication services. This is our second scenario and it is illustrated in figure 5.15. The observations $X_{Electricity}$ =*Reduced* and $X_{Telecom}$ =*Not Working* were used. We see that the banks will still work, but the sector might be reduced in functionality. Also the transportation sector, which relies a lot on the railway the air sector, will have problems; it might even failure to work. The utility sectors will also have a lot of problems and the stock market will go down or even crash.

5.7.3 Scenario III

What will happen if both the electricity and the telecommunication go down? In this case we have the observations $X_{Electricity}$ =*Not Working* and $X_{Telecom}$ =*Not Working*. Entering this evidence into `hb`$^{\texttt{BN}}$ we will get the outcome in figure 5.16. In this work scenario, most of the sectors will experience severe damages or even stop working. The risk for a stock market crash is increased.

5.7.4 Conclusions

We see that using a Bayesian network tool like `hb`$^{\texttt{BN}}$, it is easy to try out different scenarios. In this when we model a one time occasion like the millennium shift, we can not learn the structure and/or the parameters from data. Instead, we have to rely on experts and their knowledge. But after having the expert set up a Bayesian network model, it can be used to try out different scenarios.

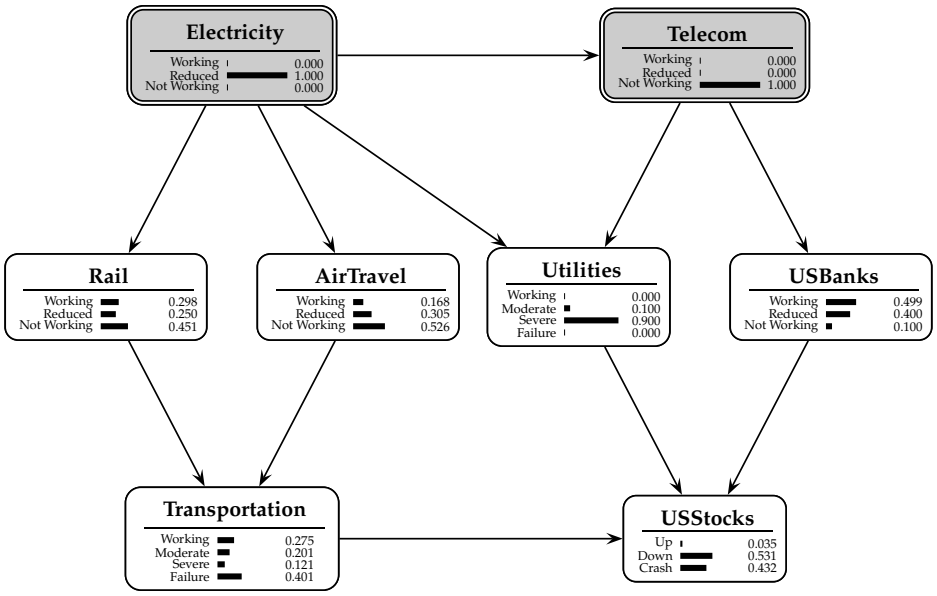


Figure 5.15: Scenario II: Electricity will be reduced and the telecommunication do not work.

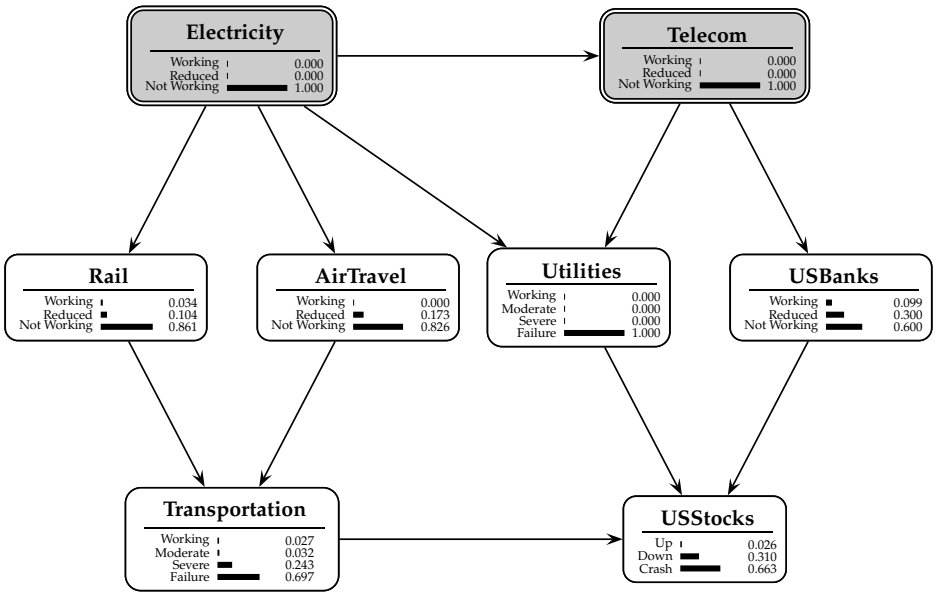


Figure 5.16: Scenario III: Both the electricity and the telecommunication do not works.

Chapter 6

Reasoning and Causation

This chapter is included to get the reader more interested and make her want to study the subject further. It might even convince her that a Bayes net is a powerful tool to model a problem.

6.1 What would have happened if we had not...?

An interesting paper about causality is a paper by Pearl called *Reasoning with Cause and Effect* [Pea99]. In this paper an example of the power of Bayesian network is given. The example is called *"The Firing Squad"* and could unfortunately be describing a real-world scenario. We are not going to discuss the ethical problems in this example, but it is included because it shows an interesting approach how to solve a common problem. The example is about a prisoner who might be executed, depending on the court order. If the court decides upon execution, they tell a captain, who then will order two riflemen to shoot the prisoner. In figure 6.1 the example is illustrated.

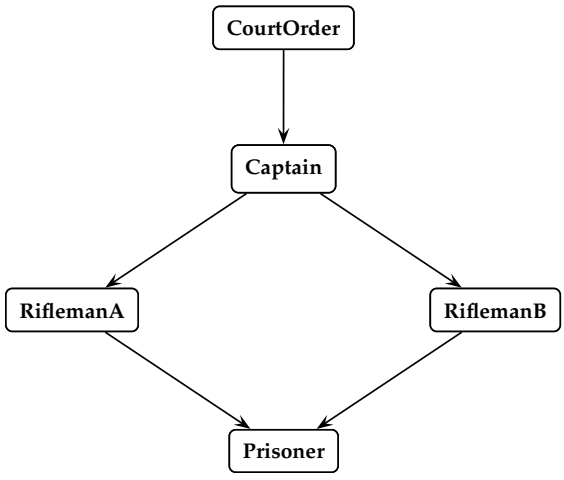


Figure 6.1: The model of the "The Firing Squad".

First Pearl defines the world to be fully deterministic, i.e. if the court orders an exe-

cution, then the captain will order the two riflemen to shoot, who then will simultaneously fire there weapons and then the prisoner will die. This system is easily modeled using predicate logic (see the paper). It is also fair to say that model it as a Bayesian network is too much work. But what happens if we add uncertainty? Let us say that rifleman A is a nervous guy that just as he hear the captain telling them to shoot or not, he might pull the trigger anyway. Assume that the risk that rifleman A pulls the trigger because he is nervous, is one out of ten. We do not know the outcome of the courts decision, but we estimate the probability for it to order an execution to be 70%. Now it is reasonable to use belief network to model the problem. For those who, after been reading this report, still roots for using predicate logic together with probability calculations, Pearl explains how to model it. It is possible, but tedious and hard to understand.

6.1.1 The twin-model approach

To complicate the world for logicians even more, Pearl asks the question:

> *What if we observe that the prisoner is dead and we know that rifleman A has a nervous finger, what is then the probability that the prisoner would be alive if rifleman A would not have been shooting?*

This is a really tricky question, it is hard to answer. However, it is justified. It is often that questions like "What would have happened if we had not...?" are asked. The approach to solve this problem is to think about the world as it contains two joined models. See figure 6.2.

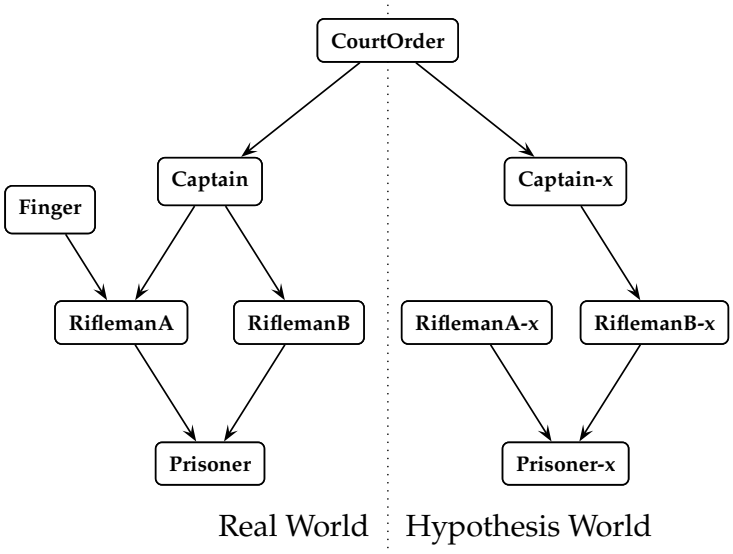


Figure 6.2: The twin model using for answering the question "What is the probability that the prisoner would be alive if rifleman A would not have been firing?" The left branch of the graph represents the current state of the world (as we have observed it). The right branch represents our hypothesis that the rifleman A did not shoot.

The first model is used for modeling the *real world*, i.e. a Bayesian network were we have *observed* the prisoner being dead ($X_{Prisoner}$ =*Dead*). The second model is used for modeling the *hypothesis world*, were the evidence is that the rifleman A does not fire ($X_{RiflemanA\text{-}x} = Don't\ Fire$). The latest statement could also be coded such that there is no influence from the captain to the rifleman A, i.e. there is no edge in the network. These two worlds do communicate through the court order.

We know that the risk for rifleman A to fire because he is nervous is 10%; $P(X_{Finger} = Nervous) = 0.10$. We also know that $P(X_{CourtOrder} = Execution) = 0.70$. Enter this knowledge into our twin-model together with the observation that the prisoner is dead, $P(X_{Prisoner} = Dead) = 1$. Enter also the "evidence" in the hypothesis world that rifleman A do not shoot, $P(X_{RiflemanA\text{-}x} = Don't\ Fire) = 1$, then we get that the prisoner would be alive by a chance of 9%, if rifleman A had not been shooting. See figure 6.3.

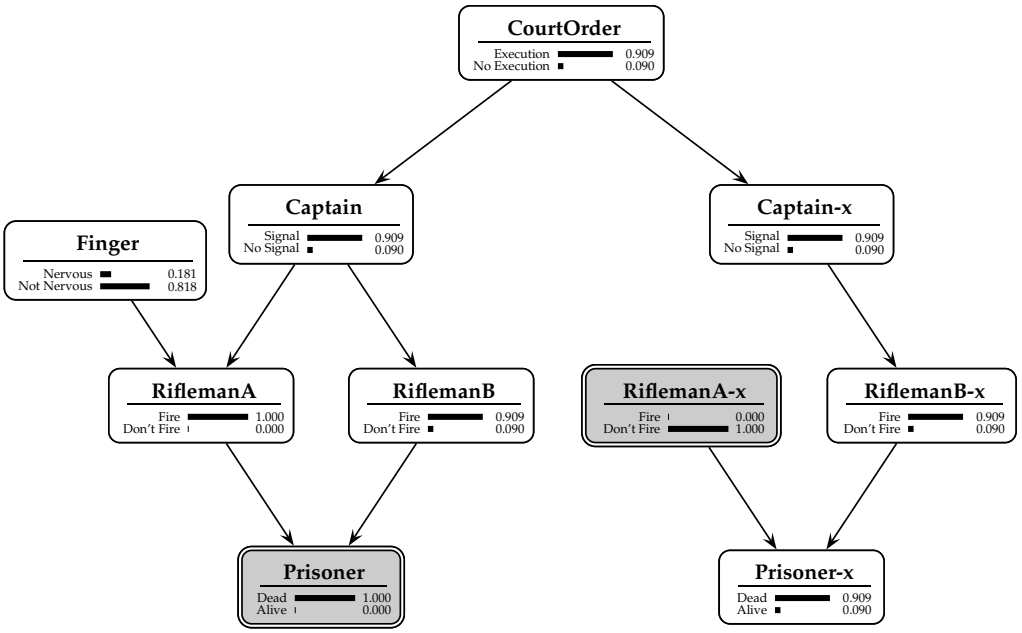


Figure 6.3: We observe that the prisoner is dead, what is the probability that he would be alive if rifleman A had not shot? He would be alive with the probability of 9%.

Chapter 7

Further readings

This report covered the basics of Bayesian networks, what they are and how they can be used for making inference. Also the theory of Markov trees, which carries the fundamental properties for Bayesian networks is described. The secondary structure, into which Bayesian networks are converted by the junction tree algorithm, also relies on Markov trees and Markov properties. Further, the report showed that it is possible to convert any Bayes net into a junction tree which can be used for making inference and belief updates. From the updated and consistent junction tree we can then, with help of marginalization, get the discrete probability distribution for any of the variables in the Bayesian network. All algorithms needed were covered, explicitly explained and exemplified.

However, there were also a lot of things that were not discussed. For readers who wants to learn more about Bayesian networks and their applications, several suggestions of further readings are given below.

7.1 Further readings

This report did only cover Bayesian networks where the variables are *discrete*. The theory for Bayesian networks where the variables are continuous or even mixed is not covered. A simple way around, if one only have a discrete Bayesian network tool to work with, is to discretize the continuous data. If this is not satisfactory it is recommended to read further in Lauritzen's book *Graphical Models* [Lau96].

With help of a Bayesian network one can find the most probable reason for an outcome. This is useful when performing diagnosis and fault detection. It is also possible to let your model undergo sensitivity analysis; "What happens if I change this parameter? How much does it affect the system?". More about this can be found in [Jen96].

In real-life it might happen that we can not observe all variables or that it is very expensive to observe some of the variables. A nice feature of a Bayes net application would then be that it give suggestions of which variable to observe next in order to obtain most information about the system given earlier observation with the constraints of time and cost. This is what you might need to do when you are planning a test. The

value of information is a "theory" that deals with these kinds of problems and it originates from [How66] and [Lin71].

We can also take the Bayesian networks one step further and create something namned *influence diagrams* [HM84]. A influence diagram is a powerful tool for modeling decision problems [NJ99] and is useful for decision making and planning. In the belief network society a lot of work and research are done on the subject (for example [NJ99, XP99, Jen96, Sha99]).

The report did not cover how to build a Bayesian network model. It can be created manually with help of experts who create both the structure of the network and assign the strength of dependency between variables. It is also possible to learn the strength of dependency (conditional probabilities) from databases or experiments. The most exciting is learning of *both* structure and conditional probabilities from data. In fields where it is not known how the variables are related, this can be used for suggesting models, which then can be reconfirmed or refined by experiments. A very interesting and recent research project tries to learn the relations between genes in the yeast cell *saccharomyces cerevisiae* [FNP99][1]. Learning can be of type *batch learning*, where you do the learning from beginning using all of the data, or it can be of type *adaptation* where you change your Bayesian network, qualitative and/or quantitative, when you receive new data. The later is preferred, but it is not always possible. More information about learning can be found in for instance [Hec95], [Jen96], [RS98] and [HGC94].

Dynamic Bayesian networks have not been mentioned at all. Bayesian networks were not designed to model temporal reasoning under uncertainty. The extension of the Bayesian network semantics to include temporal relations is difficult, but some suggestions how to do it have been given [AC96, NB94, Kan91]. Nicholson and Brady's article *Dynamic Bayes nets for Discrete Monitoring* [NB94] are told to be worth reading. There exists though a simple work around that allows us to use a regular Bayesian network to also represent the temporal information. The idea is to discretize the time into periods and use these periods as states in the vertices (variables) in the network. Each variable does also contain one state representing the union of all time periods. The approach is called *Temporal Nodes Bayesian Networks* (TNBN), see for instance [AFS99].

The *Uncertainty in Artificial Intelligence* (UAI) society put a lot of effort in exploring Bayesian networks and its possibilities. Each year a UAI conference is hold and this year (1999) the conference was in Stockholm, Sweden. A dominant part of papers presented was about Bayesian networks [2]. They also held a one-day course introducing the subject for the less experience audience. This conference can be recommended for anybody who wants to know more about the Bayesian networks and reasoning under uncertainty.

Even though I tried to cover a lot of the fields on which today's research is focusing, and some applications that have been successful, there is much much more to be explored.

[1] There is a web page about the project *Using Bayesian Networks to Analyze Whole-Genome Expression Data*. Its URL is `http://www.cs.huji.ac.il/labs/pmai2/expression` and there will soon be a paper published presenting the biological result more deeply.

[2] See URL `http://uai99.iet.com`.

Appendix A

How to represent potentials and distributions on a computer

A.1 Background

In many situations when designing statistical software, the number of variables a discrete distribution will have is not known in advance (at compilation time). There exists a need to generate discrete multidimensional distributions on the fly. In figure A.1, several examples of probability representations are given. To the upper left there a variable X_a with a one-point distribution. To the upper right X_a has a four-point distribution. The variable $\mathbf{X} = (X_a, X_b)$ found in the lower left corner have a 2×3-point distribution. Finally, found in the lower right corner, there is a $4 \times 3 \times 3$-point distribution.

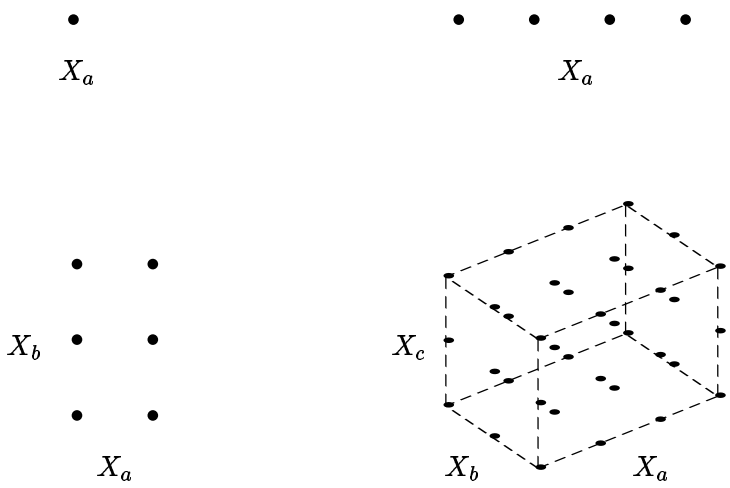


Figure A.1: Top left: A one-point distribution. Top right: A one-dimensional distribution with four possible values. Bottom left: A joint distribution with two variables, having two and three possible values, respectively. Bottom right: A joint distribution with rank three, each variable having four, three and three possible values, respectively.

A.2 Multi-way arrays

One obvious way to store these multidimensional probabilities is to put them in a *multi-way array*. Let us say that we want to create a one-dimensional array with four elements of type double using Java. This is easily done as

```
double[] X = new double[4];
```

To *declare and create* a 2x3-matrix and a 4x3x3-dimensional array with elements of type double one can write

```
double[][]   X = new double[2][3];
double[][][] Y = new double[4][3][3];
```

In Java, which has zero-based indices on arrays, `Y[1][0][2]` refers to the element with at position $(2, 1, 3)$ (one-based) in the multi-way array $\mathbf{Y}$.
This is no problem at all, but these examples deals with situations when the *rank* (number of dimensions) of an array is known *at compilation time*. But what if one wants to create an array with rank N at run-time? In programming languages such as Java, C and Pascal, there is no easy way to do it. Just think about it and consider the following desperate attempt

```
double[][]...[] X = new double[2][4]...[6];
```

A.2.1 The vec-operator

An easy way around this problem is to use the *vec-operator*. The vec-operator is an operator that takes a multi-way array, e.g. a matrix, and puts the elements in a column vector. In the case of a matrix the columns are stacked on top of each other creating a column vector. Here is an example

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \Longleftrightarrow vec(A) = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{12} \\ a_{22} \end{bmatrix} = \begin{bmatrix} a'_1 \\ a'_2 \\ a'_3 \\ a'_4 \end{bmatrix}$$

This idea is extended to higher dimensions. For a three-dimensional array we will get

$$A = \left[\begin{bmatrix} a_{112} & a_{122} \\ a_{111} & a_{121} \end{bmatrix} \begin{bmatrix} a_{212} & a_{222} \\ a_{211} & a_{221} \end{bmatrix} \right]$$

$$\Longleftrightarrow$$

$$vec(A) = \begin{bmatrix} vec(A_1) \\ vec(A_2) \end{bmatrix} = \begin{bmatrix} vec(A_{11}) \\ vec(A_{21}) \\ vec(A_{12}) \\ vec(A_{22}) \end{bmatrix} = \begin{bmatrix} a_{111} \\ a_{211} \\ a_{121} \\ a_{221} \\ a_{112} \\ a_{212} \\ a_{122} \\ a_{222} \end{bmatrix} = \begin{bmatrix} a'_1 \\ a'_2 \\ a'_3 \\ a'_4 \\ a'_5 \\ a'_6 \\ a'_7 \\ a'_8 \end{bmatrix}$$

To handle this representation we have to be able to convert between the two indices; the index $(i_1, i_2, \ldots, i_k)$ identifying the elements in the k-dimensional array and the index p identifying the elements in the corresponding vector representation. In the three-dimensional example given above, we have for instance that $a_{212} = a'_6$, i.e. $(2,1,2) \mapsto 6$ and vice versa.

A.2.2 Mapping between the indices in the multi-way array and the vec-array

The general mapping from the k-dimensional array to the vectored one and vice versa is a one-to-one mapping. These mapping functions will now be described using the same notation as in the article [Hol85]. Some ideas below are also from personal discussions with the author [Hol99]. Denote the index of an element in the k-dimensional array as $\mathbf{i}_k = (i_1, i_2, \ldots, i_k)$ which is a sequence of positive integers. The *shape* of the array is described by the number of elements along each dimension and is denoted $\mathbf{n}_k = (n_1, n_2, \ldots, n_k)$. Obviously, we have that $1 \leq i_r \leq n_r;\ r = 1, \ldots, k$. We use the notation $\pi_j(\mathbf{n}_k)$ for $n_0 n_1 \cdots n_j$ with $n_0 = 1$. As a comment, $\pi_k(\mathbf{n}_k)$ is equal to the number of elements in the array, called the *size* of the array. We define the vector $\Pi_k(\mathbf{n}_k)$ as $(\pi_0(\mathbf{n}_k), \pi_1(\mathbf{n}_k), \ldots, \pi_{k-1}(\mathbf{n}_k))$. A scalar product $\langle \mathbf{a}_k | \mathbf{b}_k \rangle$ is defined as $a_1 b_1 + \ldots + a_k b_k$ where $\mathbf{a}_k = (a_1, \ldots, a_k)$ and $\mathbf{b}_k = (b_1, \ldots, b_k)$. The mapping from the k-dimensional array to the stacked vector is then given by

$$p(\mathbf{i}_k, \mathbf{n}_k) = 1 + \langle \mathbf{i}_k - \mathbf{1}_k | \Pi_k(\mathbf{n}_k) \rangle, \qquad \mathbf{i}_k \in \mathbf{n}_k. \tag{A.1}$$

One could also write

$$\begin{aligned} &p\left((i_1, i_2, \ldots, i_k), (n_1, n_2, \ldots, n_k)\right) = \\ &= 1 + (i_1 - 1) + (i_2 - 1) n_1 + (i_3 - 1) n_1 n_2 + \ldots + (i_k - 1) n_1 n_2 \cdots n_{k-1}. \end{aligned}$$

In the other way around, the mapping goes from a scalar to a k-dimensional integer. Denote with $\lfloor x \rfloor$ the integer part of x. The inverse mapping is then given by

$$i_r = 1 + \left\lfloor \frac{(p(\mathbf{i}_k, \mathbf{n}_k) - 1) \mod \pi_r(\mathbf{n}_k)}{\pi_{r-1}(\mathbf{n}_k)} \right\rfloor. \tag{A.2}$$

Considering the 4x3x3-dimensional array in figure A.1 we have that

$$\begin{aligned} k &= 3 \quad \text{(rank)} \\ \mathbf{n}_3 &= (4,3,3) \quad \text{(shape)} \\ \Pi_3(\mathbf{n}_3) &= (1,4,12) = (1,4,4 \cdot 3). \quad \text{(base)} \end{aligned}$$

Then, for instance, the index $\mathbf{i}_3 = (1,3,2)$ is mapped to the following scalar-index in the vec-array

$$p(\mathbf{i}_3, \mathbf{n}_3) = 1 + (1-1) + (3-1) \cdot 4 + (2-1) \cdot 12 = 1 + 0 + 8 + 12 = 21.$$

Back again we get

$$
\begin{aligned}
i_1 &= 1 + \left\lfloor \frac{(21-1) \mod 4}{1} \right\rfloor = 1 + \left\lfloor \frac{0}{1} \right\rfloor = 1 \\
i_2 &= 1 + \left\lfloor \frac{(21-1) \mod 12}{4} \right\rfloor = 1 + \left\lfloor \frac{8}{4} \right\rfloor = 3 \\
i_3 &= 1 + \left\lfloor \frac{(21-1) \mod 36}{12} \right\rfloor = 1 + \left\lfloor \frac{20}{12} \right\rfloor = 2.
\end{aligned}
$$

Implementation notes: In the case of Java and C, arrays start with index *zero*, and because of this one can skip the adding and subtraction of one that is found both of equation (A.1) and equation (A.2). The ones are there since the indices start counting from one.

A.2.3 Fast iteration along dimensions

A lot of algorithms are often iterating from element to element along one dimension in the multi-way array. An example of this is the summation of the elements in one direction where the results are saved in a new multi-way array with the same rank as before minus one. In figure A.2 an example where we sum along the third index in the multi-way array $\mathbf{X} = (X_b, X_a, X_c)$ are shown. We see that the result is a new multi-way array $\mathbf{X}' = (X_b, X_a)$ with $\mathit{rank}(X') = \mathit{rank}(X) - 1$. Note that the variables must not be ordered in a special way as long as we keep track of the order.

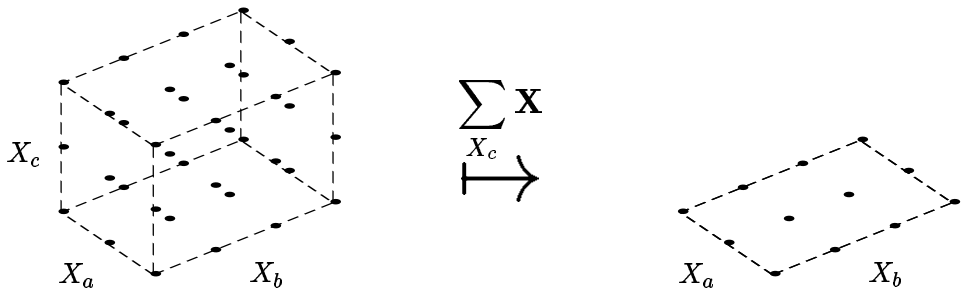


Figure A.2: To the left is a multi-way array $\mathbf{X} = (X_b, X_a, X_c)$ with dimension $4 \times 3 \times 3$. Summing along the third dimension we get a new multi-way array with dimension 4×3, which is seen to the right.

The summing function in the previous example motivates the need for a fast method to iterate along a given dimension. Consider another three-dimensional array with a huge number of elements. Starting from element $\mathbf{i} = (i_1, i_2, i_3) = (0, 0, 0)$ and summing along the second dimension (i_2) it should be intractable to use the mapping function (eq. (A.1)) each time we increase the index i_2. Looking at the mapping function or using common sense, we see that by increasing i_2 by one the corresponding index in the vector representation is increased by a constant (the base for that dimension). This

is always the case. More generally we have that

$$i_j \leftarrow i_j + 1 \Longleftrightarrow p(\mathbf{i}_k, \mathbf{n}_k) \leftarrow p(\mathbf{i}_k, \mathbf{n}_k) + \pi_j(\mathbf{n}_k). \tag{A.3}$$

A.2.4 Object oriented design of a multi-way array

Reading the section above, we find that a multi-way array has some attributes like *rank*, *shape* etc. For a multi-way array and its stacked vector representation, there is also some mapping-functions between their indices. Putting everything together it is reasonable to design a class called DOUBLEARRAY to be used for representing a multi-way array with elements of type double. Here is an UML-notation for the class

DoubleArray	
public	double[] **value**;
protected	int[] **shape**;
protected	int **rank**;
protected	int[] **base**;
public	**DoubleArray**();
public	**DoubleArray**(int[] shape);
public	double **get**(int[] index);
public	double **set**(int[] index, double value);
public	int **rank**();
public	int[] **shape**();
public	int **size**();
public	int **size**(int dimension);
public	int **getIndex**(int[] index);
public	int[] **getIndex**(int index);
public	int **iteratorStep**(int dimension);
public	double **sum**();
public	DoubleArray **sum**(int[] dimension);

A.3 Probability distributions and potentials

A.3.1 Discrete probability distributions

In the previous section we designed the basic functionality for representation of discrete distributions. For one-dimensional and multi-dimensional distributions we can use the DoubleArray design basically as it is. There is though a slight problem. How do we keep track of what dimension is representing what variable? We need some kind of mapping function between variables and dimensions. In table A.1 a mapping function, represented by a lookup-table, is given. It is denoted $\mathbf{d} = (X_b, X_a, X_c)$. This is the mapping function for the three variables X_a, X_b and X_c that are given in figure A.2. The notation ① is used to distinguish between dimensions (①,②,③, ...) and indices (1,2,3, ...).

Implementation notes: Assume that each variable is implemented as a class called `Variable`. We can then use a single array of type `Variable[]` to implement a binary one-to-one map. For now, lets call it `varMap`. The length of the array is equal the number of variables in the distribution. The mapping function from dimension index to the

dimension	variable
①	X_b
②	X_a
③	X_c

Table A.1: The lookup-function that maps the dimension indices ①, ②, and ③, to the three variables X_b, X_a and X_c, respectively.

Variable is just a simple `varpMap[`i_j`-1]` where i_j is the index of the dimension[1]. To find the index corresponding to a certain variable we have to do a linear search.

A.3.2 Discrete conditional probability distributions

To represent discrete *conditional* distributions we can also use a multi-way array, where the first dimension is reserved for distribution of the dependant variable and all other dimensions represent the conditioning variables. In figure A.3 the probability function $P(Y|X_a, X_b)$ is represented as a three-dimensional multi-way array.

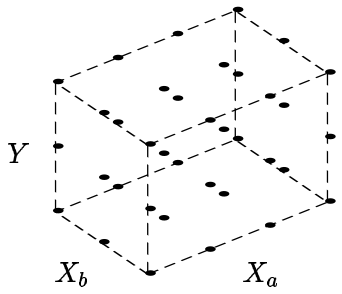


Figure A.3: The discrete conditional probability function $P(Y|X_a, X_b)$ is represented as a multi-way array. X_a has four states and X_b and Y has both three states.

A.3.3 Discrete potentials

Potentials can also be represented as multi-way arrays. Consider two potentials ϕ_A and ϕ_B on the variable sets $\mathbf{X}_A = (X_a, X_b, X_c)$ and $\mathbf{X}_B = (X_c, X_b)$, respectively. X_a, X_b and X_c have 2, 3 and 3 states, respectively. See figure A.4. Potentials also need a mapping function between variables and dimension indices in the multi-way array. We use the same design as before.

A.3.4 Multiplication of potentials and probabilities

The *multiplication* of ϕ_A and ϕ_B is a potential ϕ_C, where $C = A \cup B$. It is denoted $\phi_C = \phi_A \phi_B$. Each $\phi_C(\mathbf{x}_C)$ is computed by first identifying the instantiation of $\mathbf{x}_A$ and $\mathbf{x}_B$ that are consistent with $\mathbf{x}_C$. The $\phi_C(\mathbf{x}_C)$ is then assigned the product $\phi_A(\mathbf{x}_A)\phi_B(\mathbf{x}_B)$.

[1]The subtraction of one is because we assume Java or C arrays which have zero-based indices.

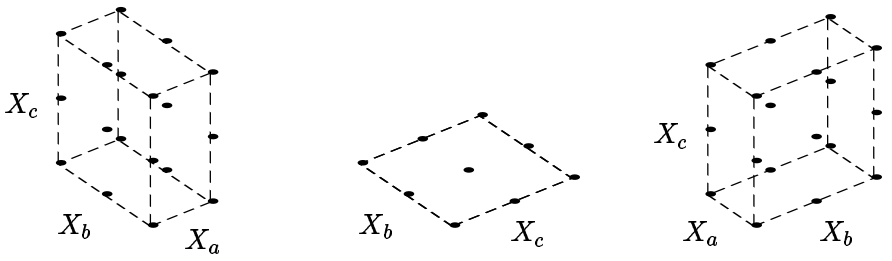


Figure A.4: Left: The potential ϕ_A over the variable set $\mathbf{X}_A = (X_a, X_b, X_c)$ is represented by a $2 \times 3 \times 3$-dimensional array. Middle: The potential ϕ_B over the variable set $\mathbf{X}_B = (X_c, X_b)$ is represented by a 3×3-dimensional array. Right: The potential ϕ_C over the variable set $\mathbf{X}_C = (X_b, X_a, X_c)$ is represented by a $3 \times 2 \times 3$-dimensional array.

d_A	Variable	$\sigma_{A \to C}$
①	X_a	②
②	X_b	①
③	X_c	③

d_B	Variable	$\sigma_{B \to C}$
①	X_c	③
②	X_b	①

Table A.2: Left: The mapping function $\sigma_{A \to C}$ between the dimensions in potential ϕ_A and potential ϕ_C. Right: The mapping function $\sigma_{B \to C}$ between the dimensions in potential ϕ_B and potential ϕ_C.

Since the setup of variable set are made during run-time we can not make any assumptions about the order of the variables in the set[2]. In our example, let us say that (after creating a junction tree for instance) we get $\mathbf{X}_C = (X_b, X_a, X_c)$ and a therefore a potential ϕ_C with dimensions $3 \times 2 \times 3$. See figure A.4. To speed up the process of finding consistent values during the multiplication, we can create two mapping functions between the indices in ϕ_A and ϕ_C, and between the indices in ϕ_B and ϕ_C. We denote these indices $\mathbf{i}_A$, $\mathbf{i}_B$, and $\mathbf{i}_C$, respectively. So, we need two functions $\rho_{C \to A} : \mathbf{i}_C \mapsto \mathbf{i}_A$ and $\rho_{C \to B} : \mathbf{i}_C \mapsto \mathbf{i}_B$. In order to do this, we first have to find how the dimensions in each multi-way array maps to each other. It is only necessary to have the mapping from A to C and from B to C, respectively. The reason for this will come clear later. Lets denote these mapping functions as $\sigma_{A \to C} : \mathbf{d}_A \mapsto \mathbf{d}_C$ and $\sigma_{B \to C} : \mathbf{d}_B \mapsto \mathbf{d}_C$. We see that the first dimension in $\mathbf{X}_A$ is the one corresponding to variable X_a. This variable is found on the second dimension in $\mathbf{X}_C$. The second dimension in $\mathbf{X}_A$ (X_b) is found on dimension ① and the third dimension (X_c) is found on dimension ③ in $\mathbf{X}_C$. This gives us the dimension map $\sigma_{A \to C}(\mathbf{d}_A) = (②, ①, ③)$. Similar we get that $\sigma_{B \to C}(\mathbf{d}_B) = (③, ①)$. See table A.2. The mapping functions can be implemented as arrays of integers (`int[]`) with length equal to the number of variables in cluster A and cluster B, respectively.

When multiplying we step through ϕ_C element by element. After a while we reach, lets say, the element with index $\mathbf{i}_\mathbf{C} = (3, 1, 2)$. This element correspond to state a $\mathbf{x}_C = (x_{b,3}, x_{a,1}, x_{c,2})$, and denote the corresponding potential $\phi_C(\mathbf{i}_\mathbf{C})$. Also, let $\mathbf{i}_\mathbf{C}(①)$ denote the first element in $\mathbf{i}_\mathbf{C}$ ($= 3$), $\mathbf{i}_\mathbf{C}(②)$ denote the second element in $\mathbf{i}_\mathbf{C}$ ($= 1$),

[2]One should never make assumption about the order of elements in *sets*. If ordered sets are needed, the concept of *lists* or *sequences* should be used.

and so on. Now, since the dimension map $\sigma_{A\to C}(\mathbf{d}_A)$ is equal to $(②, ①, ③)$, we find the consistent state $\mathbf{x}_A = (x_{a,1}, x_{b,3}, x_{c,2})$ as the element with index $\mathbf{i_A} = \rho_{C\to A}(\mathbf{i}_C) = (\mathbf{i_C}(②), \mathbf{i_C}(①), \mathbf{i_C}(③)) = (1, 3, 2)$ in ϕ_A. In the same way do we get that the element with index $\mathbf{i_B} = \rho_{C\to B}(\mathbf{i}_C) = (\mathbf{i_C}(③), \mathbf{i_C}(①)) = (2, 3)$ corresponds to the consistent state $\mathbf{x}_B = (x_{b,3}, x_{c,2})$. Now we can perform the multiplication $\phi_A(\mathbf{i_A})\phi_B(\mathbf{i_B})$ and assign the result to the element at $\mathbf{i_C} = (3, 1, 2)$ in ϕ_C.

In pseudo-code the algorithm for multiplication between two potentials can be written as

ALGORITHM 13 (MULTIPLICATION OF POTENTIALS ϕ_A AND ϕ_B)

1. *Calculate the intersection $C = A \cup B$.*
2. *Create the mapping functions $\sigma_{\mathbf{A}\to\mathbf{C}}$ and $\sigma_{\mathbf{B}\to\mathbf{C}}$.*
3. *Create a new potential ϕ_C to store the result.*
4. *For each element in ϕ_C with index $\mathbf{i_C}$;*
   (a) *Get the index $\mathbf{i_A}$ of the state element in ϕ_A that is consistent with ϕ_C using map $\sigma_{\mathbf{A}\to\mathbf{C}}$.*

   FOR $k = 1$ TO rank(A)
   Assign to index element k in $\mathbf{i_A}$ the $\sigma_{A\to C}(k)$:th value of $\mathbf{i_C}$:

   $$\mathbf{i_A}(k) \leftarrow \mathbf{i_C}\left(\sigma_{A\to C}(k)\right)$$

   (b) *Get the index $\mathbf{i_B}$ of the state element in ϕ_B that is consistent with ϕ_C using map $\sigma_{\mathbf{B}\to\mathbf{C}}$.*

   FOR $k = 1$ TO rank(B)
   Assign to index element k in $\mathbf{i_B}$ the $\sigma_{B\to C}(k)$:th value of $\mathbf{i_C}$:

   $$\mathbf{i_B}(k) \leftarrow \mathbf{i_C}\left(\sigma_{B\to C}(k)\right)$$

   (c) *Multiply the two elements in ϕ_A and ϕ_B with index $\mathbf{i_A}$ and $\mathbf{i_B}$, respectively, and save the result in ϕ_C at element with index $\mathbf{i_C}$;*

   $$\phi_C(\mathbf{i_C}) \leftarrow \phi_A(\mathbf{i_A})\phi_B(\mathbf{i_B}).$$

The *multiplication* of the potential ϕ_A and the probability distribution $P(\mathbf{X}_B)$ is a potential ϕ_C, where $C = A \cup B$. It is denoted $\phi_C = \phi_A P(\mathbf{X}_B)$. Each $\phi_C(\mathbf{x}_C)$ is computed by first identifying the instantiation of $\mathbf{x}_A$ and $\mathbf{x}_B$ that are consistent with $\mathbf{x}_C$. The $\phi_C(\mathbf{x}_C)$ is then assigned the product $\phi_A(\mathbf{x}_A)P(\mathbf{x}_B)$.

Not surprisingly, the multiplication between a potential and a probability function is defined the same way as the multiplication between two potentials. Similarly, is the multiplication between two probability functions.

Appendix B

XML Belief Network File Format

B.1 Background

During some years now, one has in the Uncertainty and Artificial Intelligence (UAI) community[1] discussed the need of an interchange format for Bayesian networks. Several formats have been designed, most by individual developers. Examples are the *Net Language* by the HUGIN Expert A/S and the *Bayesian Network Interchange Format* (BNIF) by Microsoft Research.

During the 1998 Conference on Uncertainty in Artificial Intelligence (UAI '98) the community decided to develop a language based on XML (see below), which was by then becoming a widely used language for the Web. At the time of writing, there is not one standardized XML belief network file format, but a small number of well-documented XML formats from different developers. This report making use of the XBN-format [Dec99], a draft developed and posted by the Decision Theory & Adaptive Systems Group at Microsoft Research[2] (DTAS).

In the next section, the XML-language will be introduced. A brief overview of the XBN-format will then follow.

B.2 XML - Extensible Markup Language

Extensible Markup Language (XML^{TM}) is a data format for structured document interchange, initially designed for the Internet, but now also to be used in intranets and on file systems. It is *not* a single, predefined markup language, but a *meta-language*. A meta-language is a language for describing other languages. XML is a "extremely simple dialect" of the international standard meta-language for markup, SGML[3]. Using XML we can define our own markup language, one for each class of document[4] there

[1] For more information about the Association for Uncertainty in Artificial Intelligence see http://www.auai.org

[2] The home page of the Decision Theory & Adaptive Systems Group is http://www.research.microsoft.com/research/dtg/bnformat

[3] Standard Generalized Markup Language (ISO 8879).

[4] The definition of the term *document* is flexible. It is not necessary that it should be visually presented. It can also be used for describing a data structure, such as a Bayesian network, which is read by an expert system.

are.

Responsible for the specification is the XML Working Group of the World Wide Web Consortium (W3C)[5]. The language is not a proprietary development of any company. Instead it is a public format. The v1.0 specification [TNMoIaC98] was accepted by the W3C as Recommendation in February 1998.

There are several commercial and non-commercial software packages in almost any language parses XML-documents. Currently, `hb`$^{\texttt{BN}}$ are using the *XML Parser for Java* from AlphaWorks at IBM. It is a free parser that can be downloaded at

http://www.alphaworks.ibm.com/formula/xml

In the future, it is probably very easy to exchange this parser for another one, if one would like to.

B.3 XBN - XML Belief Network File Format

Remember that, even though the XML-language is a standardize language, the XBN-format described here, which is based on XML, is *not* standardized. It is developed by Microsoft Research.

The XBN-format is fully defined in the document-type description file `xbn.dtd` provided by Microsoft Research and is based on XML version 1.0. The content of the file is printed in B.3.1 and there is also a description how to download the file from Internet. This file contains the language definition and is written in a meta-language. The only thing needed is to declare the document type on the first line of the XBN-file:

```
<!DOCTYPE ANALYSISNOTEBOOK SYSTEM "xbn.dtd">
```

An XBN-file contains exactly one top-level document called **ANALYSISNOTEBOOK**, which in itself can contain one or more Bayes nets documents (**BNMODEL**)[6]. In figure B.1, the structure (hierarchy) of the elements in an XBN-file is displayed. The actual XBN-language is defined in B.3.1. For an example of an XBN-file see one of the Bayesian networks listed in appendix C.
For more detailed information about the XBN format see [Dec99].

[5] For more information about the W3C see http://www.w3.org

[6] According to DTAS the analysis book "will be extended over time to encompass other types of objects; for example, collections of data, database connection and query information, and so on".

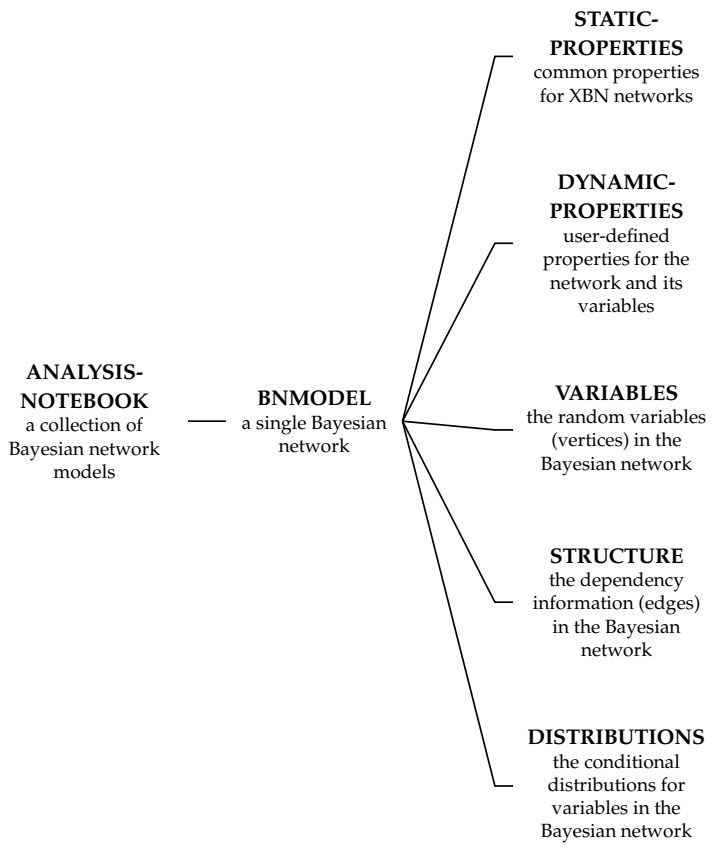


Figure B.1: A simplified graph explaining the element hierarchy of an XBN-file.

B.3.1 The Document Type Description File - `xbn.dtd`

The file `xbn.dtd` printed in this section can be downloaded at

http://www.research.microsoft.com/research/dtg/bnformat/xbn_dtd.html

`xbn.dtd`

```
<?xml encoding="US-ASCII"?>

<!-- DTD for sets of belief network models -->
<!ELEMENT ANALYSISNOTEBOOK (BNMODEL)+>
  <!ATTLIST ANALYSISNOTEBOOK
    NAME ID #REQUIRED
    ROOT IDREF #IMPLIED
    FILENAME CDATA #IMPLIED>

<!-- a single belief network -->
<!ELEMENT BNMODEL (  STATICPROPERTIES
                   | DYNAMICPROPERTIES
                   | VARIABLES
                   | STRUCTURE
                   | DISTRIBUTIONS
                  )+>

  <!ATTLIST BNMODEL NAME ID #REQUIRED>

<!-- comment element declarations -->
<!ELEMENT COMMENT (#PCDATA)>
<!ELEMENT PROPVALUE (#PCDATA)>
<!ELEMENT STATENAME (#PCDATA)>
<!ELEMENT PROPERTY (PROPVALUE)+>
   <!ATTLIST PROPERTY NAME NMTOKEN #REQUIRED>
<!ELEMENT PROPXML (#PCDATA)>
   <!ATTLIST PROPXML NAME NMTOKEN #REQUIRED>

<!-- static header declaration section -->
<!ELEMENT STATICPROPERTIES (#PCDATA | FORMAT | VERSION | CREATOR )*>
  <!ELEMENT FORMAT EMPTY>
    <!ATTLIST FORMAT VALUE CDATA "MSR DTAS XML">
  <!ELEMENT VERSION EMPTY>
    <!ATTLIST VERSION VALUE CDATA #REQUIRED>
  <!ELEMENT CREATOR EMPTY>
    <!ATTLIST CREATOR VALUE CDATA #IMPLIED>

<!-- dynamic properties declaration section -->
<!ELEMENT DYNAMICPROPERTIES (PROPERTYTYPE|PROPERTY|PROPXML)+>
  <!ELEMENT PROPERTYTYPE (COMMENT)?>
    <!ATTLIST PROPERTYTYPE
    NAME NMTOKEN #REQUIRED
    ENUMSET NMTOKENS #IMPLIED
    TYPE (real | string | realarray | stringarray | enumeration) "string">

<!-- random variables declaration section -->
<!ELEMENT VARIABLES (VAR)+>
  <!ELEMENT VAR ( STATENAME | PROPERTY | PROPXML | DESCRIPTION )+>
    <!ATTLIST VAR
      TYPE (discrete | continuous) "discrete"
      NAME NMTOKEN #REQUIRED
      XPOS CDATA #IMPLIED
      YPOS CDATA #IMPLIED>
    <!ELEMENT DESCRIPTION (#PCDATA)>

<!-- topological dependency structure information -->
<!ELEMENT STRUCTURE (ARC|MEMBER)*>
  <!-- specify dependency arc -->
  <!ELEMENT ARC EMPTY>
```

```
    <!ATTLIST ARC
      PARENT NMTOKEN #REQUIRED
      CHILD NMTOKEN #REQUIRED>
  <!-- specify set inclusion for parentless variables -->
  <!ELEMENT MEMBER EMPTY>
    <!ATTLIST MEMBER NAME NMTOKEN #REQUIRED>

<!-- distributions -->

<!ELEMENT DISTRIBUTIONS (DIST)*>
  <!ELEMENT DIST ( (CONDSET)?, ( ( (PRIVATE|SHARED), DPIS) | REFERENCE ) )*>
    <!ATTLIST DIST
      TYPE (discrete|ci) "discrete"
      FUNCTYPE (max|plus) #IMPLIED>

    <!-- conditioning set declaration -->
    <!ELEMENT CONDSET (CONDELEM)*>
      <!ELEMENT CONDELEM EMPTY>
        <!ATTLIST CONDELEM
          NAME NMTOKEN #REQUIRED
          STATES CDATA #IMPLIED>

    <!-- private/shared declarations -->
    <!ELEMENT PRIVATE EMPTY>
      <!ATTLIST PRIVATE NAME NMTOKEN #REQUIRED>
    <!ELEMENT SHARED EMPTY>
      <!ATTLIST SHARED
        NAME NMTOKEN #REQUIRED
        STATES CDATA #IMPLIED>

    <!-- discrete parent instantiation probability vectors -->
    <!ELEMENT DPIS (DPI)*>
      <!ELEMENT DPI  (#PCDATA)>
        <!ATTLIST DPI INDEXES NMTOKENS #IMPLIED>

    <!-- distribution reference (binding) declaration -->
    <!ELEMENT REFERENCE EMPTY>
      <!ATTLIST REFERENCE
        VAR NMTOKEN #REQUIRED
        SHAREDDIST NMTOKEN #REQUIRED>
```

Appendix C

Some of the networks in XBN-format

C.1 "Icy Roads"

IcyRoads.xml

```
<!DOCTYPE ANALYSISNOTEBOOK SYSTEM "xbn.dtd">

<ANALYSISNOTEBOOK NAME="Notebook.IcyRoads" ROOT="IcyRoads">
   <BNMODEL NAME="IcyRoads">
      <STATICPROPERTIES>
         <FORMAT VALUE="MSR DTAS XML"/>
         <VERSION VALUE="0.2"/>
         <CREATOR VALUE="Henrik Bengtsson, hb@maths.lth.se"/>
      </STATICPROPERTIES>
      <VARIABLES>
         <VAR NAME="Watson" TYPE="discrete" XPOS="340" YPOS="0">
            <DESCRIPTION>Watson</DESCRIPTION>
            <STATENAME>yes</STATENAME>
            <STATENAME>no</STATENAME>
         </VAR>
         <VAR NAME="Holmes" TYPE="discrete" XPOS="0" YPOS="10">
            <DESCRIPTION>Holmes</DESCRIPTION>
            <STATENAME>yes</STATENAME>
            <STATENAME>no</STATENAME>
         </VAR>
         <VAR NAME="Icy" TYPE="discrete" XPOS="170" YPOS="110">
            <DESCRIPTION>Icy</DESCRIPTION>
            <STATENAME>yes</STATENAME>
            <STATENAME>no</STATENAME>
         </VAR>
      </VARIABLES>

      <STRUCTURE>
         <ARC PARENT="Icy" CHILD="Watson"/>
         <ARC PARENT="Icy" CHILD="Holmes"/>
      </STRUCTURE>

      <DISTRIBUTIONS>
         <DIST TYPE="discrete">
            <PRIVATE NAME="Icy"/>
            <DPIS>
               <DPI> 0.7 0.3</DPI>
            </DPIS>
         </DIST>

         <DIST TYPE="discrete">
```

```
            <CONDSET>
               <CONDELEM NAME="Icy"/>
            </CONDSET>
            <PRIVATE NAME="Holmes"/>
            <DPIS>
               <DPI INDEXES=" 0 "> 0.8 0.2</DPI>
               <DPI INDEXES=" 1 "> 0.1 0.9</DPI>
            </DPIS>
         </DIST>

         <DIST TYPE="discrete">
            <CONDSET>
               <CONDELEM NAME="Icy"/>
            </CONDSET>
            <PRIVATE NAME="Watson"/>
            <DPIS>
               <DPI INDEXES=" 0 "> 0.8 0.2</DPI>
               <DPI INDEXES=" 1 "> 0.1 0.9</DPI>
            </DPIS>
         </DIST>
      </DISTRIBUTIONS>
   </BNMODEL>
</ANALYSISNOTEBOOK>
```

C.2 "Year2000"

Year2000.xml

```
<!DOCTYPE ANALYSISNOTEBOOK SYSTEM "xbn.dtd">

<ANALYSISNOTEBOOK NAME="Notebook.Year2000" ROOT="Year2000">
   <BNMODEL NAME="Year2000">
      <STATICPROPERTIES>
         <FORMAT VALUE="MSR DTAS XML"/>
         <VERSION VALUE="0.2"/>
         <CREATOR VALUE="Henrik Bengtsson, hb@maths.lth.se"/>
      </STATICPROPERTIES>
      <VARIABLES>

         <VAR NAME="Electricity" TYPE="discrete" XPOS="130" YPOS="180">
            <DESCRIPTION>Electricity</DESCRIPTION>
            <STATENAME>Working</STATENAME>
            <STATENAME>Reduced</STATENAME>
            <STATENAME>Not Working</STATENAME>
         </VAR>
         <VAR NAME="Telecom" TYPE="discrete" XPOS="280" YPOS="180">
            <DESCRIPTION>Telecom</DESCRIPTION>
            <STATENAME>Working</STATENAME>
            <STATENAME>Reduced</STATENAME>
            <STATENAME>Not Working</STATENAME>
         </VAR>
         <VAR NAME="Air_Travel" TYPE="discrete" XPOS="170" YPOS="90">
            <DESCRIPTION>Air Travel</DESCRIPTION>
            <STATENAME>Working</STATENAME>
            <STATENAME>Reduced</STATENAME>
            <STATENAME>Not Working</STATENAME>
         </VAR>
         <VAR NAME="Rail" TYPE="discrete" XPOS="90" YPOS="90">
            <DESCRIPTION>Rail</DESCRIPTION>
            <STATENAME>Working</STATENAME>
            <STATENAME>Reduced</STATENAME>
            <STATENAME>Not Working</STATENAME>
         </VAR>
         <VAR NAME="US_Banks" TYPE="discrete" XPOS="320" YPOS="90">
            <DESCRIPTION>US Banks</DESCRIPTION>
```

```
            <STATENAME>Working</STATENAME>
            <STATENAME>Reduced</STATENAME>
            <STATENAME>Not Working</STATENAME>
         </VAR>
         <VAR NAME="US_Stocks" TYPE="discrete" XPOS="280" YPOS="0">
            <DESCRIPTION>US Stocks</DESCRIPTION>
            <STATENAME>Up</STATENAME>
            <STATENAME>Down</STATENAME>
            <STATENAME>Crash</STATENAME>
         </VAR>
         <VAR NAME="Utilities" TYPE="discrete" XPOS="240" YPOS="90">
            <DESCRIPTION>Utilities</DESCRIPTION>
            <STATENAME>Working</STATENAME>
            <STATENAME>Moderate</STATENAME>
            <STATENAME>Severe</STATENAME>
            <STATENAME>Failure</STATENAME>
         </VAR>
         <VAR NAME="Transportation" TYPE="discrete" XPOS="130" YPOS="0">
            <DESCRIPTION>Transportation</DESCRIPTION>
            <STATENAME>Working</STATENAME>
            <STATENAME>Moderate</STATENAME>
            <STATENAME>Severe</STATENAME>
            <STATENAME>Failure</STATENAME>
         </VAR>
      </VARIABLES>

      <STRUCTURE>
         <ARC PARENT="Electricity" CHILD="Telecom"/>
         <ARC PARENT="Electricity" CHILD="Air_Travel"/>
         <ARC PARENT="Electricity" CHILD="Rail"/>
         <ARC PARENT="Telecom" CHILD="US_Banks"/>
         <ARC PARENT="US_Banks" CHILD="US_Stocks"/>
         <ARC PARENT="Utilities" CHILD="US_Stocks"/>
         <ARC PARENT="Transportation" CHILD="US_Stocks"/>
         <ARC PARENT="Electricity" CHILD="Utilities"/>
         <ARC PARENT="Telecom" CHILD="Utilities"/>
         <ARC PARENT="Rail" CHILD="Transportation"/>
         <ARC PARENT="Air_Travel" CHILD="Transportation"/>
      </STRUCTURE>



      <DISTRIBUTIONS>
         <DIST TYPE="discrete">
            <PRIVATE NAME="Electricity"/>
            <DPIS>
               <DPI>0.6 0.3 0.1</DPI>
            </DPIS>
         </DIST>

         <DIST TYPE="discrete">
            <CONDSET>
               <CONDELEM NAME="Electricity"/>
            </CONDSET>
            <PRIVATE NAME="Telecom"/>
            <DPIS>
               <DPI INDEXES="0">0.8  0.15 0.05</DPI>
               <DPI INDEXES="1">0.6  0.3  0.1 </DPI>
               <DPI INDEXES="2">0.1  0.3  0.6 </DPI>
            </DPIS>
         </DIST>

         <DIST TYPE="discrete">
            <CONDSET>
               <CONDELEM NAME="Electricity"/>
            </CONDSET>
            <PRIVATE NAME="Air_Travel"/>
            <DPIS>
```

```
            <DPI INDEXES="0">0.6  0.3  0.1 </DPI>
            <DPI INDEXES="1">0.3  0.4  0.3 </DPI>
            <DPI INDEXES="2">0.0  0.3  0.7 </DPI>
         </DPIS>
      </DIST>

      <DIST TYPE="discrete">
         <CONDSET>
            <CONDELEM NAME="Electricity"/>
         </CONDSET>
         <PRIVATE NAME="Rail"/>
         <DPIS>
            <DPI INDEXES="0">0.7  0.2  0.1 </DPI>
            <DPI INDEXES="1">0.5  0.3  0.2 </DPI>
            <DPI INDEXES="2">0.1  0.2  0.7 </DPI>
         </DPIS>
      </DIST>

      <DIST TYPE="discrete">
         <CONDSET>
            <CONDELEM NAME="Telecom"/>
         </CONDSET>
         <PRIVATE NAME="US_Banks"/>
         <DPIS>
            <DPI INDEXES="0">0.7  0.2  0.1 </DPI>
            <DPI INDEXES="1">0.5  0.3  0.2 </DPI>
            <DPI INDEXES="2">0.1  0.3  0.6 </DPI>
         </DPIS>
      </DIST>

      <DIST TYPE="discrete">
         <CONDSET>
            <CONDELEM NAME="US_Banks"/>
            <CONDELEM NAME="Utilities"/>
            <CONDELEM NAME="Transportation"/>
         </CONDSET>
         <PRIVATE NAME="US_Stocks"/>
         <DPIS>
            <DPI INDEXES="0 0 0">1    0    0   </DPI>
            <DPI INDEXES="0 0 1">0.8  0.2  0   </DPI>
            <DPI INDEXES="0 0 2">0.5  0.5  0   </DPI>
            <DPI INDEXES="0 0 3">0.1  0.9  0   </DPI>

            <DPI INDEXES="0 1 0">0.8  0.2  0   </DPI>
            <DPI INDEXES="0 1 1">0.5  0.5  0   </DPI>
            <DPI INDEXES="0 1 2">0.1  0.9  0   </DPI>
            <DPI INDEXES="0 1 3">0    0.8  0.2 </DPI>

            <DPI INDEXES="0 2 0">0    1    0   </DPI>
            <DPI INDEXES="0 2 1">0    0.99 0.01</DPI>
            <DPI INDEXES="0 2 2">0    0.95 0.05</DPI>
            <DPI INDEXES="0 2 3">0    0.9  0.1 </DPI>

            <DPI INDEXES="0 3 0">0    0.9  0.1 </DPI>
            <DPI INDEXES="0 3 1">0    0.85 0.15</DPI>
            <DPI INDEXES="0 3 2">0    0.5  0.5 </DPI>
            <DPI INDEXES="0 3 3">0    0.7  0.3 </DPI>

            <DPI INDEXES="1 0 0">0    1    0   </DPI>
            <DPI INDEXES="1 0 1">0    0.99 0.01</DPI>
            <DPI INDEXES="1 0 2">0    0.9  0.1 </DPI>
            <DPI INDEXES="1 0 3">0    0.85 0.15</DPI>

            <DPI INDEXES="1 1 0">0    0.85 0.15</DPI>
            <DPI INDEXES="1 1 1">0    0.8  0.2 </DPI>
            <DPI INDEXES="1 1 2">0    0.75 0.25</DPI>
            <DPI INDEXES="1 1 3">0    0.7  0.3 </DPI>
```

```
            <DPI INDEXES="1 2 0">0    0.75 0.25</DPI>
            <DPI INDEXES="1 2 1">0    0.7  0.3 </DPI>
            <DPI INDEXES="1 2 2">0    0.6  0.4 </DPI>
            <DPI INDEXES="1 2 3">0    0.5  0.5 </DPI>

            <DPI INDEXES="1 3 0">0    0.7  0.3 </DPI>
            <DPI INDEXES="1 3 1">0    0.65 0.35</DPI>
            <DPI INDEXES="1 3 2">0    0.6  0.4 </DPI>
            <DPI INDEXES="1 3 3">0    0.5  0.5 </DPI>

            <DPI INDEXES="2 0 0">0    0.6  0.4 </DPI>
            <DPI INDEXES="2 0 1">0    0.5  0.5 </DPI>
            <DPI INDEXES="2 0 2">0    0.4  0.6 </DPI>
            <DPI INDEXES="2 0 3">0    0.3  0.7 </DPI>

            <DPI INDEXES="2 1 0">0    0.5  0.5 </DPI>
            <DPI INDEXES="2 1 1">0    0.4  0.6 </DPI>
            <DPI INDEXES="2 1 2">0    0.3  0.7 </DPI>
            <DPI INDEXES="2 1 3">0    0.2  0.8 </DPI>

            <DPI INDEXES="2 2 0">0    0.4  0.6 </DPI>
            <DPI INDEXES="2 2 1">0    0.3  0.7 </DPI>
            <DPI INDEXES="2 2 2">0    0.2  0.8 </DPI>
            <DPI INDEXES="2 2 3">0    0.1  0.9 </DPI>

            <DPI INDEXES="2 3 0">0    0.3  0.7 </DPI>
            <DPI INDEXES="2 3 1">0    0.2  0.8 </DPI>
            <DPI INDEXES="2 3 2">0    0.1  0.9 </DPI>
            <DPI INDEXES="2 3 3">0    0    1   </DPI>
         </DPIS>
      </DIST>

      <DIST TYPE="discrete">
         <CONDSET>
            <CONDELEM NAME="Telecom"/>
            <CONDELEM NAME="Electricity"/>
         </CONDSET>
         <PRIVATE NAME="Utilities"/>
         <DPIS>
            <DPI INDEXES="0 0">1    0    0    0   </DPI>
            <DPI INDEXES="0 1">0.7  0.3  0    0   </DPI>
            <DPI INDEXES="0 2">0    0.6  0.35 0.05</DPI>

            <DPI INDEXES="1 0">0.7  0.2  0.1  0   </DPI>
            <DPI INDEXES="1 1">0.4  0.3  0.2  0.1 </DPI>
            <DPI INDEXES="1 2">0    0.3  0.5  0.2 </DPI>

            <DPI INDEXES="2 0">0    0.4  0.5  0.1 </DPI>
            <DPI INDEXES="2 1">0    0    0.4  0.6 </DPI>
            <DPI INDEXES="2 2">0    0    0    1   </DPI>
         </DPIS>
      </DIST>

      <DIST TYPE="discrete">
         <CONDSET>
            <CONDELEM NAME="Rail"/>
            <CONDELEM NAME="Air_Travel"/>
         </CONDSET>
         <PRIVATE NAME="Transportation"/>
         <DPIS>
            <DPI INDEXES="0 0">1    0    0    0   </DPI>
            <DPI INDEXES="0 1">0.7  0.3  0    0   </DPI>
            <DPI INDEXES="0 2">0.5  0.3  0.2  0   </DPI>

            <DPI INDEXES="1 0">0.7  0.2  0.1  0   </DPI>
            <DPI INDEXES="1 1">0.5  0.3  0.2  0   </DPI>
            <DPI INDEXES="1 2">0.4  0.3  0.2  0.1 </DPI>

```

```
                <DPI INDEXES="2 0">0.5  0.2  0.3  0   </DPI>
                <DPI INDEXES="2 1">0.4  0.2  0.3  0.1 </DPI>
                <DPI INDEXES="2 2">0    0    0.1  0.9 </DPI>
             </DPIS>
          </DIST>

       </DISTRIBUTIONS>
    </BNMODEL>
 </ANALYSISNOTEBOOK>
```

Appendix D

Simple script language for the hb^{BN}-tool

D.1 XBNScript

D.1.1 Some of the scripts used in this project

D.1.2 IcyRoads.script.xml

IcyRoads.script.xml

```
<?xml version="1.0"?>
<!DOCTYPE XBNSCRIPT SYSTEM "xbnscript.dtd">

<XBNSCRIPT>
 <ECHO>*****************************************</ECHO>
 <ECHO> Icy Roads</ECHO>
 <ECHO/>
 <ECHO> This example is taken from</ECHO>
 <ECHO> "An Introduction to Bayesian Networks"</ECHO>
 <ECHO> Finn V. Jensen, 1996</ECHO>
 <ECHO>*****************************************</ECHO>
 <ECHO>Loading network...</ECHO>
 <LOAD FILENAME="IcyRoads.xml"/>
 <ECHO>OK</ECHO>
 <ECHO/>

 <SAVE FILENAME="IcyRoads.tex" WHAT="BN" FORMAT="LaTeX">
   <PARAMETER NAME="CommandName">IcyRoads</PARAMETER>
   <PARAMETER NAME="Details">0</PARAMETER>
 </SAVE>
 <SAVE FILENAME="IcyRoads.tex" WHAT="BN" FORMAT="LaTeX">
   <PARAMETER NAME="Append">yes</PARAMETER>
   <PARAMETER NAME="CommandName">IcyRoadsPriori</PARAMETER>
   <PARAMETER NAME="Details">3</PARAMETER>
 </SAVE>
 <SAVE FILENAME="IcyRoads.tex" WHAT="JoinTree" FORMAT="LaTeX">
   <PARAMETER NAME="Append">yes</PARAMETER>
   <PARAMETER NAME="CommandName">IcyRoadsJoinTree</PARAMETER>
 </SAVE>

 <!-- Display the distribution for all variables -->
 <DISPLAY VAR="*"/>

 <!-- Observations -->
 <ECHO>************************************************</ECHO>
 <ECHO>* "The information that Watson has crashed</ECHO>
 <ECHO>*  updates the probability that the roads are</ECHO>
```

```
  <ECHO>*  icy and that Holmes also has crashed."</ECHO>
  <ECHO>**************************************************</ECHO>
  <OBSERVE VAR="Watson" STATENAME="yes"/>
  <!-- Global propagation -->
  <UPDATE/>
  <DISPLAY VAR="Icy"/>
  <DISPLAY VAR="Holmes"/>
  <SAVE FILENAME="IcyRoads.tex" WHAT="BN" FORMAT="LaTeX">
    <PARAMETER NAME="Append">yes</PARAMETER>
    <PARAMETER NAME="CommandName">IcyRoadsW</PARAMETER>
    <PARAMETER NAME="Details">3</PARAMETER>
  </SAVE>

  <ECHO>**************************************************</ECHO>
  <ECHO>* "At last, when Inspector is convinced that the</ECHO>
  <ECHO>*  roads are not icy, then P(H|I=n)=(0.1,0.9)."</ECHO>
  <ECHO>**************************************************</ECHO>
  <OBSERVE VAR="Icy" STATENAME="no"/>
  <!-- Global propagation -->
  <UPDATE/>
  <DISPLAY VAR="Holmes"/>
  <SAVE FILENAME="IcyRoads.tex" WHAT="BN" FORMAT="LaTeX">
    <PARAMETER NAME="Append">yes</PARAMETER>
    <PARAMETER NAME="CommandName">IcyRoadsWI</PARAMETER>
    <PARAMETER NAME="Details">3</PARAMETER>
  </SAVE>


  <!-- Remove all observations -->
  <ECHO>******************************************</ECHO>
  <ECHO>* Resetting network</ECHO>
  <ECHO>******************************************</ECHO>
  <RESET/>
  <DISPLAY VAR="*"/>
 </XBNSCRIPT>
```

References

[AC96] Constantin F. Aliferis and Gregory F. Cooper. A Structurally and Temporally Extended Bayesian Belief Network Model: Definitions, Properties, and Modeling Techniques. In Eric Horvitz and Finn Jensen, editors, *Uncertainty in Artificial Intelligence - Proceedings of the 12th Conference*, pages 28–39. Morgan Kaufman Publishers, Inc., August 1996.

[AFS99] Gustavo Arroyo-Figueroa and Luis Enrique Sucar. A Temporal Bayesian Network for Diagnosis and Prediction. In Kathryn B. Laskey and Henri Prade, editors, *Uncertainty in Artificial Intelligence - Proceedings of the 15th Conference*, pages 13–20. Morgan Kaufman Publishers, Inc., 1999.

[Cli90] Peter Clifford. Markov random fields in statistics. In G. R. Grimmett and D. J. A. Welsh, editors, *Disorder in Physical Systems. A Volume in Honour of John M. Hammersley*, pages 19–32. Clarendon Press, 1990.

[Coo90] Gregory F. Cooper. The computational complexity of probabalistic inference using Bayesian belief networks. *Artifical Intelligence*, 42:393–405, 1990.

[Dec99] Decision Theory & Adaptive Systems Group at Microsoft Research, Advanced Technology Division, Microsoft Corporation. *Belief Network File Format for XML*, February 1999. http://www.research.microsoft.com/research/dtg/bnformat.

[FNP99] Nir Friedman, Iftach Nachman, and Dana Peér. Learning Bayesian Network Structure from Massive Datasets: The "Sparse Candidate" Algorithm. In Kathryn B. Laskey and Henri Prade, editors, *Uncertainty in Artificial Intelligence - Proceedings of the 15th Conference*, pages 206–215. Morgan Kaufman Publishers, Inc., 1999.

[HD94] Cecil Huang and Adnan Darwiche. Inference in Belief Networks: A Procedual Guide. Technical report, Section on Medical Informatics, Stanford University School of Medicine, 1994.

[HEA99] Denmark Hugin Expert A/S. Hugin Expert A/S - Home page. http://www.hugin.dk, 1999.

[Hec95] David Heckerman. A Tutorial on Learning With Bayesian Networks. Technical Report MSR-TR-95-06, Microsoft Research, Advanced Technology Division, Microsoft Corporation, March 1995.

[HGC94] David Heckerman, Dan Geiger, and David M. Chickering. Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. Technical Report MSR-TR-94-09, Microsoft Research, Advanced Technology Division, Microsoft Corporation, March 1994.

[HM84] Ronald A. Howard and James E. Matheson, editors. *Influence Diagrams in The Principles and Applications of Decision Analysis*, volume Volumes I and II. Strategic Decisions Group, Menlo Park, California, 1984.

[Hol85] Björn Holmquist. The Direct Product Permuting Matrices. *Linear and Multilinear Algebra*, 17:117–141, 1985.

[Hol99] Björn Holmquist. Personal discussion with Björn Holmquist, May 1999.

[How66] Ronald A. Howard. Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, SSC-2:22–26, 1966.

[Jen94] Frank Jensen. Implementation aspects of various propagation algorithms in HUGIN. Research Report R-94-2014, Department of Mathematics and Computer Science, Aalborg University, Denmark, 1994.

[Jen96] Finn V. Jensen. *An introduction to Bayesian Networks*. UCL Press Ltd., London, 1996.

[Kan91] Keiji Kanazawa. A Logic and Time Nets for Probabilistic Inference. In Kathleen Dean, Thomas L.; McKeown, editor, *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 360–365. MIT Press, July 1991.

[Kjæ90] Uffe Kjærulff. Triangulation of graphs - algorithms giving small total state space. Technical Report R-90-09, Department of Computer Science, Aalborg University, Denmark, 1990.

[Lau96] Steffen L. Lauritzen. *Graphical Models*. Clarendon Press, Oxford, 1996.

[Lin71] D. V. Lindley. *Making Decisions*. Wiley Interscience, New York, 1971.

[Mat92] F. Matúš. On equivalence of Markov properties over undirected graphs. *Journal of Applied Probability*, 29:745–749, 1992.

[NB94] A.E. Nicholson and J.M. Brady. Dynamic Belief Networks for Discrete Monitoring. *IEEE Transactions on Systems, Man and Cybernetics*, Volume 34:1593–1610, 1994.

[NJ99] Thomas D. Nielsen and Finn V. Jensen. Welldefined Decision Scenarios. In Kathryn B. Laskey and Henri Prade, editors, *Uncertainty in Artificial Intelligence - Proceedings of the 15th Conference*, pages 502–511. Morgan Kaufman Publishers, Inc., 1999.

[Pea88] Judea Pearl. *Probabilistic Inference in Intelligent Systems. Networks of Plausible Inference.* Morgan Kaufman Publishers, Inc., San Mateo, CA, 1988.

[Pea97] Judea Pearl. Bayesian Networks. Technical Report R-246 (Rev. II), Cognitive System Laboratory, Computer Science Department, University of California, 1997. In *MIT Encyclopedia of the Cognitive Sciences*.

[Pea99] Judea Pearl. Reasoning with Cause and Effect. Cognitive System Laboratory, Computer Science Department, University of California, April 1999.

[Rip96] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, 1996.

[RS97] Marco Ramoni and Paola Sebastiani. Learning Bayesian Networks from Incomplete Databases. Technical Report KMI-TR-43, Knowledge Media Institute, The Open University; Dept. of Actuarial Science and Statistics, City University, February 1997.

[RS98] Marco Ramoni and Paola Sebastiani. Learning Conditional Probabilities from Incomplete Data: An Experimental Comparison. Technical Report KMI-TR-64, Knowledge Media Institute, The Open University; Dept. of Actuarial Science and Statistics, City University, July 1998.

[SAS94] Ross D. Shachter, Stig K. Andersen, and Peter Szolovits. Global Conditioning for Probabilistic Inference in Belief Networks. In Ramon Lopez de Mantaras and David Poole, editors, *Uncertainty in Artificial Intelligence - Proceedings of the 10th Conference*, pages 514–522, San Francisco, CA, USA, July 1994. Morgan Kaufmann Publishers.

[Sha99] Ross D. Shachter. Efficient Value of Information Computation. In Kathryn B. Laskey and Henri Prade, editors, *Uncertainty in Artificial Intelligence - Proceedings of the 15th Conference*, pages 594–601. Morgan Kaufman Publishers, Inc., 1999.

[TNMoIaC98] Tim Bray (Textuality, Netscape), Jean Paoli (Microsoft), and C. M. Sperberg-McQueen (University of Illinois at Chicago). *Extensible Markup Language (XML) 1.0*. W3C - World Wide Web Consortium, February 1998. http://www.w3.org/TR/REC-xml.

[XP99] Yanping Xiang and Kim-Leng Poh. Time-Critical Dynamic Decision Making. In Kathryn B. Laskey and Henri Prade, editors, *Uncertainty in Artificial Intelligence - Proceedings of the 15th Conference*, pages 688–695. Morgan Kaufman Publishers, Inc., 1999.

Index