HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
Laboratory of Computer and Information Science

Janne Toivola

Modular specification of dynamic Bayesian networks for time series analysis

Master's Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.

Espoo, 10th January 2007

Supervisor: Professor Samuel Kaski

Instructor: Jaakko Hollmén, D.Sc.(Tech.)

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE MASTER'S THESIS

Author:	Janne Toivola	
Name of the Thesis:	Modular specification of dynamic Bayesian networks for time series analysis	
Date:	10th Jan 2007	**Number of pages:** xvii + 88
Department:	Department of Computer Science and Engineering	
Professorship:	T-122 Computer and Information Science	
Supervisor:	Prof. Samuel Kaski	
Instructor:	Chief Research Scientist Jaakko Hollmén	

This thesis concentrates on specifying dynamic probabilistic models and their application in the field of discrete time series analysis. At first, the basics of discrete probabilistic models and traditional clique tree propagation inference is summarized to name the concepts used further in this work.

Then, the idea of using a stream of small Bayesian networks for dynamic time series modeling is presented. Previous work on dynamic bayesian networks, like model representation and the interface algorithm used for inference, are discussed. Their implications are subsequently taken into account in the application presented in this work. A way to specify DBNs in practice is introduced and algorithmic use of the models is studied.

An EM algorithm is developed for estimating the quantitative parameters of a specified model when partial data is given. Properties of the learning algorithm are also studied.

In the experimental part of this thesis, two different settings are demonstrated. First, an idealized and controlled laboratory setting is considered by generating artificial data and using it for demonstrating the discussed methods. The another case is a brief study of modeling human DNA copy number amplifications in a real world setting.

Keywords: dynamic Bayesian networks, belief networks, time series analysis, probabilistic inference, latent variables, missing data

TEKNILLINEN KORKEAKOULU

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Janne Toivola	
Työn nimi:	Toistuvarakenteisten Bayes-verkkojen määrittely aikasarja-analyysissä	
Päivämäärä:	10.1.2007	**Sivuja:** xvii + 88
Osasto:	Tietotekniikan osasto	
Professuuri:	T-122 Informaatiotekniikka	
Työn valvoja:	Prof. Samuel Kaski	
Työn ohjaaja:	Johtava tutkija Jaakko Hollmén	

Tämä diplomityö keskittyy dynaamisten probabilististen mallien määrittelyyn ja käyttöön diskreettien aikasarjojen analyysissä. Diskreettien probabilististen mallien ja perinteisen klikkipuun avulla tapahtuvan päättelyn perusteet kerrataan ensin lyhyesti, määritellen samalla myöhemmin käytettävät käsitteet.

Seuraavaksi esitetään idea siitä, kuinka pienistä Bayes-verkoista koostuvaa virtaa voidaan käyttää aikasarjamallinnukseen. Aiempaa dynaamisiin Bayes-verkkoihin liittyvää työtä, kuten mallien esitystapa ja eräs päättelyalgoritmi, käsitellään. Näiden seuraukset otetaan sittemmin huomioon tässä työssä esiteltävässä sovelluksessa. Käytännön tapa määritellä dynaamisia Bayes-verkkoja ja mallien käyttöä algoritmeissa tutkitaan.

EM-algoritmin toteutus kehitetään mallien kvantitatiivisten parametrien määrittämiseksi annetun datan perusteella. Myös kyseisen oppimisalgoritmin ominaisuuksia tutkitaan.

Työn kokeellisessa osuudessa esitetään kaksi esimerkkitilannetta. Ensimmäisenä on ideaalisessa ja kontrolloidussa ympäristössä tehty koe, jossa näytteistetään keinotekoista dataa ja käytetään sitä esitettyjen menetelmien havainnollistamiseen. Toinen havaintoesimerkki on koe, jossa yritetään mallintaa kopiolukumuutoksia ihmisen perimässä kuvaavaa dataa.

Avainsanat: dynaamiset Bayes-verkot, Bayes-verkot, aikasarja-analyysi, tilastolliseen malliin perustuva päättely, piilomuuttujat, puuttuva data

Acknowledgements

This Master's thesis was carried out in the Laboratory of Computer and Information Science at Helsinki University of Technology. The three years spend with the Pattern Discovery group have been wonderful and very educational.

In addition to the Pattern group, I want to thank especially the three people tightly involved with the development of the software for this work. M.Sc. Antti Rasinen implemented the elegant graph manipulation procedures for constructing clique trees. The first part, especially the parsers and static model inference, were created together with M.Sc. Mikko Korpela.

Last but not least, Dr. Jaakko Hollmén has provided invaluable insight and guidance during the whole project. He has been very understanding and patiently provided the funding for the project – despite the constant delays and problems during the software development process. In fact, this whole work fell so much behind the schedule, that I would have quit it without the positive attitude of Dr. Hollmén.

Part of the gratitude goes also to the (new) supervisor Prof. Samuel Kaski for dealing with the paper work and to the whole laboratory for the splendid working environment.

Obviously, I would like to thank my family and relatives, friends, and fellow radio amateurs for making the everyday life worthwhile.

Otaniemi, 10th January 2007

Janne Toivola

Contents

Abbreviations and Notations

BN	Bayesian Network, Belief Network
CGH	Comparative Genomic Hybridization
CPD	Conditional Probability Distribution
CTP	Clique Tree Propagation
DAG	Directed Acyclic Graph
DBN	Dynamic Bayesian Network
1.5DBN	"One-and-a-half" slice Dynamic Bayesian Network
DNA	DeoxyriboNucleic Acid
EM	Expectation Maximization
HMM	Hidden Markov Model
JLO	The inference algorithm described in [17]
JPD	Joint Probability Distribution
NP	Non-deterministic Polynomial time (comp. complexity)
OOBN	Object-Oriented Bayesian Network
OS	Open Source
2TBN	2-slice Temporal Bayesian Network
VE	Variable Elimination (algorithm)
XBN	Bayesian Network in XML
XML	eXtensible Markup Language

t	Time or other similar quantized variable, $t \in [1, T]$
V	Random variable V
V^t	Random variable V at time t
$\boldsymbol{V}$	Set of random variables ($V_i \in \boldsymbol{V}$)
$\boldsymbol{E}^{tmp}$	Temporal edges ($V_i^t \rightarrow V_j^{t+1}$)
v	A value (realization) of variable V
v_i	ith value of variable V
$\boldsymbol{y}^{1:t}$	Sequence of values $\boldsymbol{y}$ (vectors) upto time t
$\boldsymbol{y}^{1:T}$	Entire time series of $\boldsymbol{y}$
$\boldsymbol{y}_{(k)}$	kth (whole) time series of $\boldsymbol{y}$
$\boldsymbol{\Pi}_V$	Parents of variable V
$\boldsymbol{F}_V$	Family of variable V, i.e. $\{V\} \cup \boldsymbol{\Pi}_V$
$\boldsymbol{I}_t^{\rightarrow}$	Outgoing interface (subset of $\boldsymbol{V}^t$)
$P(V)$	Probability distribution for V
$p(V = v_1)$	Probability of event $V = v_1$
$\phi[i]$	Element i of the array ϕ, Note: $i \in [1, N]$
$\phi_{\boldsymbol{V}}$	Potential table (or clique) for the set $\boldsymbol{V}$ of variables
$\phi_{\boldsymbol{Z}} \downarrow \boldsymbol{A}$	Marginalization of $\phi_{\boldsymbol{A}}$ from $\phi_{\boldsymbol{Z}}$ (where $\boldsymbol{A} \subset \boldsymbol{Z}$)
λ_V	Observed likelihood for variable V, i.e. $\lambda_V[i] = p(V = v_i)$
c_V	Cardinality of variable V, i.e. number of possible values
$A \perp B \mid C$	Value of A is conditionally independent of B given C
$\boldsymbol{C}_i$	Set of random variables in clique i
$\boldsymbol{S}_j$	Set of random variables in separator set j
$M \mid \boldsymbol{x}$	Probability mass of clique tree after entering evidence $\boldsymbol{x}$

List of Figures

List of Algorithms

Chapter 1

Introduction

During the past couple of decades, the advance in computer technology has lead to collection of massive datasets. First of all, the storage capacity has grown considerably and people are able to store hundreds of gigabytes of data on their desktop computers today. Also, the sheer speed of access and manipulation of data has grown with the development of communication networks and modern computer architectures. Thus accessing and storing data is made easy in the community of the day.

While most of the gathered data shows us explicit and evident information, some interesting facts may easily be missed. For example, some patterns or rules that govern processes behind datasets are usually not feasible to find by just looking at billions of numbers. There can also be datasets that were collected for another specific purpose, but possibly contain more useful information than apparent at first sight. Data might also miss some values or be statistically skewed by inappropriate sampling.

Consequently, a myriad of data mining methods have been advised in order to take full advantage of the collected data [11]. Some of the most characteristic missions that data mining includes are the following:

- dealing with potentially huge size of data sets,
- coping with partially observed data,
- finding relationships between variables or events, and
- providing understandable summaries of data.

Algorithmic methods belonging to the categories of clustering, classification, regression, and pattern discovery, have been developed to achieve these goals [9, 11].

Among the techniques, probabilistic inference deals with statistical dependencies between quantities of interest and can handle missing data. This is particularly suited for data mining applications since usually the data, which more traditional methods fail to explore, is incomplete or fails to explicitly measure the appropriate quantities in other ways. Reasons for missing values are inability to measure some aspects of the system under study or perhaps the lack of planning before performing the measurements.

Probabilistic analysis is also conveniently fostered by the amount of available data, although the quality of data can be a problem. More importantly, probabilistic approach is a well established way of representing uncertainties that result from incomplete prior knowledge or missing observations [31, 10]. Shortly put, probabilistic inference is based on the Bayes rule familiar from statistics and pattern recognition:

$$P(\boldsymbol{V}|\boldsymbol{E}) = \frac{P(\boldsymbol{E}|\boldsymbol{V}) * P(\boldsymbol{V})}{P(\boldsymbol{E})} \tag{1.1}$$

The rule tells us how to make conclusions about the state of variables $\boldsymbol{V}$ by taking into account the prior knowledge $P(\boldsymbol{V})$ and the model $P(\boldsymbol{E}|\boldsymbol{V})$ that is supposedly behind the evidence $\boldsymbol{E}$. This is straightforward when only single variables V and E are considered.

Things get more complicated when dependencies between larger sets of (discrete) variables V and E are modeled. The fact, that $P(\boldsymbol{E}|\boldsymbol{V})$ is essentially a multidimensional probability table with exponential amount of elements, makes the computation infeasible unless the number of variables can be somehow restricted.

When some independencies between the variables are assumed, the problem can be specified in terms of smaller dependency structures and probabilistic inference is divided into simpler parts. Bayesian networks[1] provide the way to represent dependency structures as graphs and makes the computation more local and even feasible [31, 24, 16].

[1] a.k.a. belief networks, probabilistic independence networks

This master's thesis presents methods for using Bayesian networks in time series analysis[2]. For example, Hidden Markov Models (HMM) are one type of BNs and are widely used for modeling time series [26, 32, 18, 34, 25]. Besides modeling the dependencies between hidden and observed variables (say, V and E respectively), HMMs take also the dependency between consecutive time steps (V^t and V^{t+1}) into account.

Thus, HMM can be seen as a Bayesian network consisting of dynamically created similar pieces of tiny networks. A natural step towards a more general framework is made by allowing arbitrary networks as modules of the repetitive structure. This idea is better known as Dynamic Bayesian Network (DBN), dynamic time-sliced Bayesian network or dynamic probabilistic network [6, 20, 29, 36].

Whereas DBNs in principle allow the dependency structure of time slices vary through time, this work concentrates on time series analysis using modules with static structure. Having constant dependencies between variables and time steps is considered to be a reasonable assumption for many processes behind time series data. Moreover, this restriction makes it possible to precompute some aspects of the model before making probabilistic inference.

In this thesis, the method called Clique Tree Propagation[3] [14, 17, 24, 37] is used as the basis for inference procedures in contrast to the Variable Elimination algorithm seen in more general DBNs [7, 21]. The overall result is a way to specify a dynamic Bayesian network as a series of similar modules. Such a time slice is further transformed into a clique tree forming an inference engine capable of computation localized both in time and space.

The rest of this thesis is organized into following kinds of chapters. Chapter 2 introduces basics behind the Clique Tree Propagation algorithm and explains how to make statistical sampling according to a BN. The chapter presents also the known ideas behind dynamic Bayesian networks [28]. Finally, Chapter 2 explains how dynamic Bayesian networks (with static time slice structure) can be specified modularly. It is also studied how these strips

[2] note that the actual quantifier can be other than time

[3] a.k.a. Probability Propagation in Trees of Clusters, Join Tree Clustering, JLO algorithm etc.

of networks can be transformed into a suitable inference engine using clique tree propagation.

Chapter 3 deals with maximum likelihood parameter estimation with the Expectation Maximization (EM) algorithm [8]. Structures of the belief networks are assumed to be fixed and known: structural learning is beyond the scope of this work.

Chapter 4 presents the experiments done with modularly specified and computed DBNs and the software library implemented for the experiments. First, artificial data was used for testing and getting familiar with the properties of the algorithms. After that, the chapter concentrates on the investigations with real world data consisting of DNA copy number amplification sites in different types of human neoplasias [30].

The final chapter presents a summary of the work and conclusions drawn from the experiments. Additionally, some practical examples of modular model definitions can be found in the appendix.

Chapter 2

Probabilistic modeling with Bayesian networks

2.1 Background

This chapter concentrates on probabilistic models with discrete variables and certain well-known algorithms for using them. The idea of a probabilistic model is to define statistical dependencies among a set of variables. This enables such interesting tasks as probabilistic inference, statistical sampling of a known process, and learning quantitative dependencies.

The methods discussed in this work allow the modeled variables to be nominal (categorical). This means that the values of a variable may be separate labels without a specific order or magnitude. As a consequence, such concepts as mean and variance are mostly useless in describing a probability distribution. Thus, the focus is on multinomial distributions, i.e., having a separate probability for each value of a variable.

Perhaps the most fundamental task in this framework is probabilistic inference. In essence, this means entering available observations into a model and computing the posterior probabilities of hidden variables or other missing data. The traditional *clique tree propagation*[1] (CTP) algorithm [31, 17, 24, 14], handles the case of exact inference in a statically defined discrete Bayesian network. The goal is to eventually use similar method of inference

[1] a.k.a. clique tree inference, join tree clustering, JLO algorithm etc.

in a dynamic manner throughout a repetitive BN, which is useful especially in time series analysis.

There would also be an alternative inference procedure, called variable elimination (VE) [36], also known as bucket elimination algorithm [7]. It can be used with modular networks, such as the object-oriented Bayesian networks (OOBN) [22] or dynamic time-sliced Bayesian networks [21]. But in this work, the belief networks are assumed to have constant repetitive structure which facilitates reuse of data structures when clique tree algorithm is used. Besides, the clique tree inference algorithm has the same time and space complexity as variable elimination [7] in the context of static belief networks and can be more efficient in the case when posteriors for all variables are computed [36] (Sec. 14.4). Therefore it is reasonable to expect only benefits in using CTP instead of VE algorithm in this work.

In the case of time series analysis, probabilistic models tend to have repetitive structure and it would be better if the inference algorithm could concentrate on localized computations in a relatively small time slice at a time. This leads to the idea of dynamic Bayesian networks [6, 36], which extend themselves according to the length of time series.

Some inference procedures for dynamic Bayesian networks have already been invented. For example, one using a dynamic time window with variable elimination approach is presented in [21] and the *interface algorithm* is introduced in [28]. The latter one uses clique tree propagation as the basis for inference.

Probabilistic models can also be used for simulating the behavior of a known process. This is done by exploiting the causal structure implied by the representation as a directed graph. The idea is to simply draw samples for parent variables, to establish cause, and then for the children (i.e., the consequences).

2.2 Probabilistic models with discrete variables

2.2.1 Discrete variables and likelihood

The foundation of the computational methods presented here is the ability to describe dependencies between random variables. This is achieved by *graphs*, *potentials*, and *cliques* presented in the following sections. Shortly, random variables are said to be *children* depending on values of their *parent* variables. A variable V and its parents $\boldsymbol{\Pi}_V$ form a set of variables $\boldsymbol{F}_V$ naturally called the *family* of V [14]. The ways to represent relationships quantitatively are discussed in the following sections.

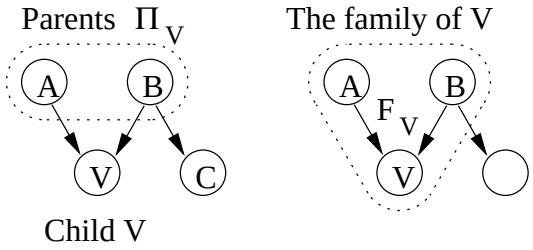

Figure 2.1: Roles of variables in a BN

In this work, each of the variables are assumed to be discrete and have a certain finite number of states: $c_V = |V|$ which is also known as the *cardinality* of the variable V. Thus the probability distribution, i.e., likelihood λ_V of a variable V can be represented as a table of values [14, 37] and the size of the table is c_V. To represent a probability distribution, the sum of the values in the table equals one:

$$\sum_{v=1}^{c_V} \lambda_V[v] = 1. \tag{2.1}$$

2.2.2 Dependencies quantitatively as potentials

As probability distributions of single variables are implemented as tables, probability distributions related to multiple discrete variables can be represented as multidimensional tables. For example, if a variable X depends on a set of other variables $\boldsymbol{Y} = \{Y_1, \ldots, Y_{N-1}\}$, the dependency can be described quantitatively by the conditional probability distribution $P(X|\boldsymbol{Y})$.

Another example could be a joint distribution $P(\boldsymbol{Z})$ of the variables $\boldsymbol{Z} = \{A, B, C\}$.

These kind of distributions can be seen as functions from the N-dimensional space formed by the variables (e.g. $\{X\} \cup \boldsymbol{Y}$, or $\boldsymbol{Z}$) to a positive real number $p \in [0, 1]$. Since the variables have a discrete and finite set of states, the function can be represented as an N-dimensional table $\phi_{\boldsymbol{V}}$, called potential [24] and containing the real numbers corresponding to the probabilities. As an example, the value $p(A = a_1, B = b_3, C = c_5)$ is found in a multidimensional table $\phi_{\boldsymbol{Z}}$ as $\phi_{z=1,3,5}$ or as `P[0][2][4]` in a computer program.

There are two kinds of operations on potentials: marginalization and multiplication [17]. Marginalization is basically for reducing the number of dimensions of a potential by calculating sums over the unnecessary variables. For example, if $\boldsymbol{A} \subseteq \boldsymbol{Z}$, $P(\boldsymbol{A})$ can be calculated from $P(\boldsymbol{Z})$ by summing over $\boldsymbol{Z} \setminus \boldsymbol{A}$ and denoted as

$$\begin{aligned} \phi_{\boldsymbol{A}} &= \sum_{\boldsymbol{Z} \setminus \boldsymbol{A}} \phi_{\boldsymbol{Z}} \\ &= \phi_{\boldsymbol{Z}} \downarrow \boldsymbol{A}. \end{aligned} \tag{2.2}$$

This means that if the set of variables $\boldsymbol{Z} = \{A, B, C\}$, then

$$\begin{aligned} \phi_A &= \sum_{B,C} \phi_{\boldsymbol{Z}} \\ \Leftrightarrow \phi_A[i] &= \sum_{j=1}^{c_B} \sum_{k=1}^{c_C} \phi_{\boldsymbol{Z}}[i][j][k], \forall i \in [1, c_A] \end{aligned} \tag{2.3}$$

The multiplication operation has a same kind of idea: a potential can be weighted with another one by multiplying suitable elements one by one (i.e., pointwise product). Again, if the set $\boldsymbol{A} \subseteq \boldsymbol{Z}$, multiplication would assign

$$\phi_{\boldsymbol{Z}} \leftarrow \phi_{\boldsymbol{A}} * \phi_{\boldsymbol{Z}}. \tag{2.4}$$

As an example, when $\boldsymbol{Z} = \{A, B, C\}$, the assignment

$$\phi_{\boldsymbol{Z}}[i][j][k] \leftarrow \phi_{\boldsymbol{Z}}[i][j][k] * \phi_A[i] \tag{2.5}$$

is done for all combinations of $(i, j, k) \in [1, c_A] \times [1, c_B] \times [1, c_C]$.

These two operations on potentials are the core of inference, because they enable such calculations as

$$\begin{aligned} P(A) &= \sum_B P(A, B) \\ &= \frac{1}{S} \sum_B P(A|B) * P(B), \end{aligned} \tag{2.6}$$

when probabilities are represented as potential tables.

From the implementation point of view, one of the most important notions is that an element in a multidimensional table can be referenced by a "flat index" also. For example, if the first index of an array is assumed to be the least significant one and $\boldsymbol{Z} = \{A, B\}$, then

- $\phi_{\boldsymbol{Z}}[1] \triangleq \phi_{\boldsymbol{Z}}[1][1]$,
- $\phi_{\boldsymbol{Z}}[i] \triangleq \phi_{\boldsymbol{Z}}[i][1]$, if $i \leq c_A$,
- $\phi_{\boldsymbol{Z}}[i] \triangleq \phi_{\boldsymbol{Z}}[i - c_A][2]$, if $c_A < i \leq 2c_A$ etc.,
- $\phi_{\boldsymbol{Z}}[c_A + 1] \triangleq \phi_{\boldsymbol{Z}}[1][2]$, and
- $\phi_{\boldsymbol{Z}}[c_A * c_B] \triangleq \phi_{\boldsymbol{Z}}[c_A][c_B]$, which is the last element.

2.2.3 Bayesian networks for model representation

Probabilistic models, and especially the dependencies between variables, are represented visually as directed acyclic graphs (DAG). Each of the nodes in a DAG corresponds to a random variable in the model. The directed edges denote parent-child relations between variables, thus encoding all the probabilistic dependencies. Actually the DAG representation defines all known conditional *independencies*, since missing edges[2] assert that a variable is independent from other variables given its parents (as explained in [31, 37]).

For example, if a model does not have an edge $(A \rightarrow B)$, then $B \perp A | \boldsymbol{\Pi}_B$. In other words, this means that knowing value of A does not provide more

[2]Missing, as compared to a fully connected graph

information about the value of B, **if** the values of the parents of B are known. This is the property known as d-separation [31].

After all the conditional independencies between variables are known, the joint probabilities can be expressed as products of conditional and prior probabilities, but the actual values of these factors are needed. This leads to the fact, that quantitative parameters of the models are represented as conditional probability tables for each child variable and prior probability tables for each variable without parents. This kind of factorization is also the most compact way to specify such a probabilistic model.

2.3 Clique tree propagation

2.3.1 Clique trees for computation

A good introduction to inference in belief networks can be found in [14, 37, 5]. Unlike the plain belief network, a secondary structure called clique tree provides a prearranged coherent structure for inference procedures. Shortly, the graphs described previously in this document are transformed into a set of groups of variables called *cliques*. The cliques are connected to each other by separator sets to form a structure called *clique tree*[3].

To create a clique tree according to a belief network, the DAG is "moralized", i.e., parents of each child variable are connected to each other by undirected edges. After that, the original directed edges are replaced by undirected ones. The resulting graph will therefore have all the families ($\boldsymbol{F}_V$) fully connected and ensure that the clique tree will have at least one clique for each family. An example is shown in Fig. 2.2, where $\{V, A, B\}$ and $\{B, C\}$ are the cliques for F_V and F_C correspondingly and the separator set between them contains B.

The inference also requires that clique trees have a so called running intersection property[4]: if any two cliques in a tree contain the variable V, then every clique on the unique path between the two cliques will contain V also. This property is achieved by triangulation of the graph. Thus, the moralized

[3] Also known as *join tree*, *junction tree*, or *cluster tree*

[4] a.k.a. join tree property

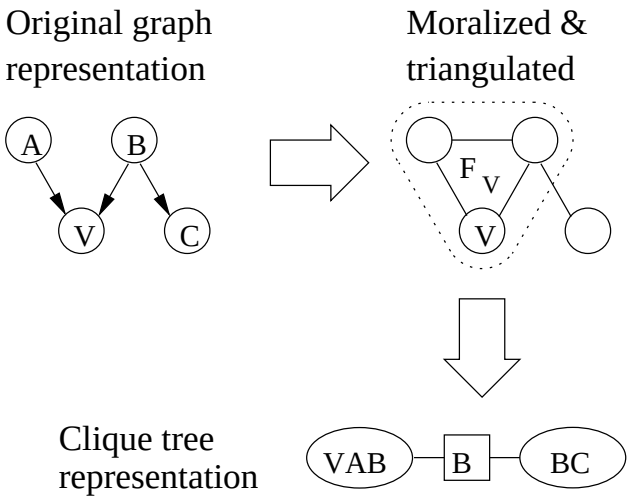

Figure 2.2: Transformation from a graph into a clique tree

graph is supplemented by additional edges that eliminate all chordless cycles in it: every cycle will consist of at most three nodes or have shortcuts due to additional edges.

All the graphs that need the additional edges, can be triangulated in many ways. In order to minimize the computational requirements, number of added edges and sizes of resulting cliques should be minimized. In this work, triangulation is done with heuristic methods [37], because exact minimization is an NP-hard problem [38, 3] and at least some level of scalability is needed.

Traditionally the word clique means a complete subgraph of some graph or just the set of nodes in such a subgraph. In this context, the word clique refers to a structure which contains the variables of a maximal fully connected subgraph of the final modified graph. The cliques[5] have also potential tables for describing the dependencies between its variables. Finally, cliques are connected together to form a clique tree (as in Fig. 3.1 for example).

Cliques are connected together by separator sets, or just shortly sepsets. A sepset contains the intersection of the two neighboring cliques, that is, the

[5]also the name *belief universe* has been used [17, 21] for differentiating it from the graph theoretical counterpart

variables common to both cliques. In the inference engine, a sepset between cliques $\boldsymbol{X}$ and $\boldsymbol{Y}$ contains also two potential tables ϕ_S and ϕ_S^{old} for message passing in addition to the variables $\boldsymbol{S} = \boldsymbol{X} \cap \boldsymbol{Y}$ [14].

2.3.2 Properties of clique trees

The key idea of an inference engine is to keep the clique tree in a consistent state. As the clique and sepset potentials encode dependencies between variables, there has to be so-called **local consistency**, i.e., for each clique $\boldsymbol{C}$ and adjacent sepset $\boldsymbol{S}$:

$$\sum_{C \setminus S} \phi_C = \phi_S, \tag{2.7}$$

which simply means that a sepset potential must describe the same kind of dependency between its variables than the neighboring clique potential. The clique just has some extra variables which the sepset does not care about.

The intention of the clique tree potentials is to encode the joint probability distribution of all the variables in the original belief network. This property is called the **global consistency**:

$$P(\boldsymbol{U}) = \frac{\prod_i \phi_{C_i}}{\prod_j \phi_{S_j}}, \tag{2.8}$$

meaning that the secondary structure expresses all the quantitative information about the belief network.

As stated in [14] (and [17] before that), these consistency requirements guarantee following properties:

$$\phi_C \propto P(\boldsymbol{C}) \tag{2.9}$$

and especially for a single variable V,

$$P(V) \propto \sum_{C \setminus \{V\}} \phi_C. \tag{2.10}$$

As a conclusion, one can compute the probability distribution of any vari-

able or joint probability distribution of its family by marginalizing a suitable clique potential and normalizing the result. The inference is only a matter of inserting evidence into the clique tree and making the potentials consistent.

One of the most important features of a the secondary structure is the join tree property: *if two cliques have a common variable, every clique and sepset on the path between the two cliques have the variable also*. The importance becomes clear in the message passing scheme, presented below, where information about evidence is propagated across the clique tree.

Another important property is that a family of a variable can be found in a single clique: this is due to the moralization of the original network and ensures that conditional probabilities $P(V|\boldsymbol{\Pi}_V)$ can be multiplied into clique potentials.

2.3.3 Initialization of a clique tree

In order to encode the quantitative dependencies of the original belief network, the clique tree must be initialized with the belief potentials [14, 37]. At first, all the potentials are uniform, i.e., $\phi_{\boldsymbol{X}} = 1$, for all cliques and sepsets $\boldsymbol{X}$. The initialization is achieved by multiplying the conditional probability distributions (belief potentials $P(V|\boldsymbol{\Pi}_V)$) and priors (where $\boldsymbol{\Pi}_V = \emptyset$) into certain clique potentials. The clique $\boldsymbol{C}$ to be initialized has to contain the corresponding variable V and its parents $\boldsymbol{\Pi}_V$: if the family of the variable is $\boldsymbol{F}_V = V \cup \boldsymbol{\Pi}_V$ then $\boldsymbol{F}_V \subseteq \boldsymbol{C}$.

$$\phi_{\boldsymbol{C}} \leftarrow \phi_{\boldsymbol{C}} * P(V|\boldsymbol{\Pi}_V), \tag{2.11}$$

as defined previously in Eq. 2.5

2.3.4 Evidence

Available data is usually certain ($V = v_i$), so called *hard* evidence, which means that the likelihood of an observed variable equals one for the observed value and zero for the other values: $p(V = v_i) = 1$, and $p(V \neq v_i) = 0$. On the other hand, uncertainties about data need to be expressed also. As an example, this is the case if one value is observed to be impossible,

$p(V = v_1) = 0$, but all other values are equally probable: $p(V = v_i) = \frac{1}{c_V - 1}$, $\forall i \in [2, c_V]$.

To include observations in the inference, the data is represented as probability distributions for each of the observed variables. This method facilitates both the common hard evidence and any kind of uncertain observations. After all, hard evidence is a special case of the more general notion of *soft* evidence, where probability may be distributed across different possible values of the variable. The likelihood table of each observed variable V is denoted as λ_V.

After the data is encoded as likelihood tables, the inference procedure can be continued by entering the likelihoods into the clique tree. A suitable clique potential ϕ_C, containing the variable, i.e., $V \in \boldsymbol{C}$, is multiplied by the new likelihood [14].

From the implementation point of view, we can reuse the clique tree if we keep record of the entered likelihoods. In the case of dynamic observations, the possible old likelihood of the variable has to be canceled away from the clique potential. If $\lambda_V^{old} \neq 0$ or $\lambda_V = 0$, it can be done with an element-wise division:

$$\phi_C \leftarrow \phi_C * \frac{\lambda_V}{\lambda_V^{old}}. \tag{2.12}$$

Otherwise the clique tree can be re-initialized. The previous likelihood of the variable, denoted by λ_V^{old}, is updated after the observation entry:

$$\lambda_V^{old} \leftarrow \lambda_V. \tag{2.13}$$

2.3.5 Propagation

The initialization and observations will make the clique tree inconsistent. After all, the parameters and data were multiplied into single cliques, so the evidence has to be distributed throughout the tree. One could think that most of the inference will happen during the third phase where the tree is made consistent.

Consistency is achieved with a two-phase procedure, where the effects of evidence are first propagated away from an arbitrary root clique and then towards the same clique in depth first order [17, 14]. The key property of this

message-passing procedure is that each clique passes a message to a neighboring clique after if has received messages from all of its other neighbors (see [14] for examples).

A single message from clique C_1 to its neighbor C_2 is passed as follows:

1. Save the old potential of sepset $S_{1,2}$ between the cliques:

$$\phi_S^{old} \leftarrow \phi_S \tag{2.14}$$

2. Compute the new sepset potential by marginalizing from clique C_1. This is so-called *projection*:

$$\phi_S \leftarrow \sum_{C_1 \setminus S} \phi_{C_1} \tag{2.15}$$

3. Update the potential in clique C_2 by multiplying with the new sepset potential. Old information is canceled by dividing with the old sepset potential. This is also known as *absorption*:

$$\phi_{C_2} \leftarrow \phi_{C_2} * \frac{\phi_S}{\phi_S^{old}} \tag{2.16}$$

Due to the running intersection property, the projection phase will conserve all information about changes needed further down the clique tree.

After the evidence is entered and the clique tree is made consistent, probability distribution of a single variable V can be computed by marginalizing from a suitable clique C (so that $V \in C$):

$$\phi_V = \sum_{C \setminus V} \phi_C. \tag{2.17}$$

Finally, the resulting 1-variable potential may need some normalization to be interpreted as a probability distribution:

$$P(V = v_i) = \frac{\phi_V[i]}{\sum_j \phi_V[j]}. \tag{2.18}$$

2.4 Dynamic Bayesian networks

2.4.1 Introduction

Time series analysis requires models for describing how quantities of interest develop through time. In the case of Bayesian networks this suggests that in addition to describing dependencies between different variables, also the dependencies between time steps should be included.

Perhaps the most simple example of this type of a temporal model is the Hidden Markov Model (HMM) [33, 37, 36]. They can be seen as streams of tiny Bayesian networks where dependency between the observed and hidden variable is modeled as the emission probability distribution. Similarly, the transition probabilities describe how the hidden state changes and the modeled process evolves through time.

The idea can be extended by allowing the time slices to be arbitrary belief networks instead of just two variables. Also additional edges between adjacent time slices can be useful, but because of the causal interpretation of the directed edges, it is considered reasonable to restrict the direction to point only forward in time (i.e., $A^{t_1} \rightarrow B^{t_2}$, where $t_2 > t_1$). The result is a repetitive structure which suggests using only one instance of the repeating elements at a time during the inference procedure.

This has two obvious advantages over a statically defined network: the parameters can be specified in a more compact way and a considerable amount of memory is saved during the inference procedure. As a difference to dynamic Bayesian networks, a statically defined network could have arbitrary variations in its parameters through time and it would be able to model non-stationary processes. This would require defining separate parameters for each time step, and add the linear factor T to memory consumption, as compared to a DBN. Additionally, parameter estimation by learning algorithms would either be cumbersome if some of the parameters were shared or constrained, or need massive amounts of training data and most likely result in overfitting. Yet another issue is, that a dynamic Bayesian network scales automatically according to the length of time series.

In addition to saving parameter space relative to time dimension, belief networks lend themselves to saving space inside a single time slice. It may be hard to create models that are not trivially equivalent to some kind of an HMM, because in most cases the hidden variables can be combined into one giant variable representing the Cartesian product of original variables (as noted in [29] for instance). This explains the fact why the worst case space requirement for a model may be exponential. On the other hand, separate variables with less cardinality allow models to be explained in smaller parts and more independencies included between them. The less dependencies there are, the less computational complexity it has.

Compared to an HMM, having a more elaborate dependency structure of latent variables may make the model more understandable or make it possible to insert some occasionally observed partial data about the variables. In fact, a more general inference engine at the core makes the associated algorithms elegant in the sense that missing or partial data is handled automatically.

It has to be noted, however, that many DBN models don't have sparse enough dependencies to be factorized into smaller parts within one time slice. One example of this is are various factorial HMMs [29] where the observed variable depends on several separate hidden Markov chains as in Fig. 2.3. When the number of hidden chains increases, the family composed of the entire time slice grows and memory requirements increase exponentially with it.

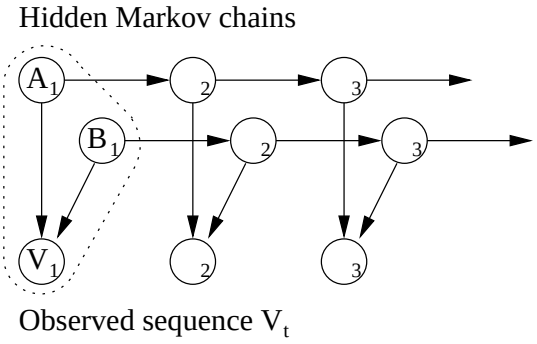

Figure 2.3: A factorial HMM

2.4.2 Alternative inference algorithms

It would be simple to just define a piece of Bayesian network (a time slice) and repeat its structure for T times. Resulting static belief network could then be used for inference with traditional algorithms like the variable elimination or clique tree propagation. Despite being simple, this kind of arrangement would waste computer memory in vain. Since graphical models are used for factoring the inference problem so that more local computations can be used [24, 17], it is more space efficient to provide sufficient context for only one time step at a time.

Various time slice inference schemes have been proposed. One of them uses a dynamic time window [21], another one is called *the frontier algorithm* [40] and the one used in this work is called *the interface algorithm* [28, 29].

Shortly, the dynamic time window approach is based on the idea of expanding the current piece of a model into the direction of inference and reducing away the old parts. The fundamental point is to maintain the Markov property: the future time slices are conditionally independent of the past time slices given the current time window. Dynamic expansion and reduction of the window is achieved by adding time slices to the graph, constructing a new clique tree and disposing the cliques corresponding to old time slices [21]. Although this scheme allows dynamic changes in the time slice structure, it is considered too complex and inefficient approach in the scope of this work. Since the graph structure is assumed to repeat itself, it would be better to create the clique tree once and reuse it for all time slices.

The frontier algorithm would be more suitable for forward inference (a.k.a. filtering) [28]. Frontier is a set of variables $\boldsymbol{Z}^t$ whose posterior, $P(\boldsymbol{Z}^t|\boldsymbol{y}^{1:t})$, summarizes all the evidence $\boldsymbol{y}$ up to a certain point of time: frontier d-separates past from future. The algorithm advances forward in time by adding variables to the frontier in topological order. A variable can be removed from the frontier when all its children have been included. Although an intuitive idea, the frontier algorithm is left aside in this work since the interface algorithm is considered more efficient [29] and uses clique tree propagation framework.

2.4.3 Model representation

Some concepts need to be defined more carefully to explain how the interface algorithm works. The repeating part of the belief network is represented in a certain way to compose a suitable clique tree for inference.

Time slice is defined as a belief network including nodes (variables $\boldsymbol{V}^t$) at a certain moment in time and the dependencies between them (edges $V_i^t \rightarrow V_j^t$). The point here is that a time slice contains all the variables of the repeating model exactly once. Thus, time slice is primarily a visual concept related to the graph representation.

To construct a DBN, the dependencies between consecutive time slices have to be defined also. In the graph representation this is done by so called *temporal arcs* $\boldsymbol{E}^{tmp}$ which extend from a node in a time slice to another node in the next time slice ($V_i^t \rightarrow V_j^{t+1}$). It is assumed that all the temporal arcs point forward in time: this is reasonable from the causality point of view. Temporal arc between nodes of the same variable ($V_i^t \rightarrow V_i^{t+1}$) is called *persistence arc* [29].

It would be simple to incorporate temporal arcs in a partial belief network by including two time slices and all the arcs between them. Such a model is called two-slice temporal Bayes net (2TBN) [28]. In this case, it is assumed that the model is first-order Markov: all the temporal arcs are between adjacent time slices. Higher-order models can be defined by adding time slices to the graph but inference engines derived from n-slice TBNs have certain drawbacks with sampling algorithms as noted in Section 2.6.

It is shown in [28] that a graph with (usually) less nodes than 2TBN is more suitable for defining a DBN and deriving a clique tree from it. The latter time slice (time t) is left intact but unnecessary nodes from the first time slice ($t-1$) are stripped off. The resulting graph is called 1.5DBN [29] because of the incomplete earlier time slice. An example of this is shown in Fig. 2.4.

From the interface algorithm point of view, the first time slice needs only the nodes that d-separate past from future and thus relay enough information from a time step to the next one during inference procedure. Nodes in the first time slice of a 1.5DBN graph comprises a set called *outgoing in-*

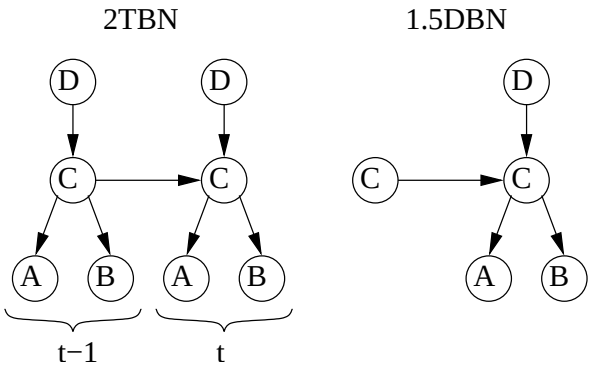

Figure 2.4: A 2TBN and corresponding 1.5DBN

terface [29]. Outgoing interface $\boldsymbol{I}_{t-1}^{\rightarrow}$ is composed of all the nodes that have children in the next time slice

$$\boldsymbol{I}_{t-1}^{\rightarrow} \stackrel{\text{def}}{=} \{A \in \boldsymbol{V}^{t-1} | (A, B) \in \mathbf{E}^{tmp}, B \in \boldsymbol{V}^{t}\} \tag{2.19}$$

When constructing a clique tree according to a 1.5DBN model, one of the cliques has to be identified as the *outgoing clique* ϕ_{out} [29] and it should contain the variables of outgoing interface $\boldsymbol{I}_{t}^{\rightarrow}$. To ensure the existence of such a clique, additional undirected edges between the interface nodes may have to be inserted before triangulation of the graph ($\boldsymbol{I}_{t}^{\rightarrow}$ has to be fully connected). *Incoming clique* ϕ_{in} is defined in similar manner: it contains the interface nodes $\boldsymbol{I}_{t-1}^{\rightarrow}$ of the earlier time slice, i.e., all the nodes V^{t-1} included in 1.5DBN.

2.4.4 The interface algorithm

After defining the interface and creating a suitable clique tree, the inference procedures are relatively straightforward as shown in [28]. Taking one inference step forward in time is explained in Algorithm 1. The clique tree is first initialized with the model parameters and evidence corresponding to the current time step. If there was a previous time slice ($t > 1$), the message α^{t-1} is passed to the current time slice in order to take the history into account.

After making the clique tree consistent, individual posterior probability

distributions $P(V^t|\boldsymbol{y}^{1:t})$ can be marginalized (and normalized) each from a clique containing the corresponding variable V. Finally, the new message α^t is computed by marginalizing non-interface variables away from the outgoing clique ϕ_{out}.

Algorithm 1 FORWARD-STEP ($\boldsymbol{y}^t$, α^{t-1}, DBN)

Input: evidence vector $\boldsymbol{y}^t$, message α^{t-1}, and the model DBN
Output: $P(V^t|\boldsymbol{y}^{1:t})$, $\forall V$, and α^t

```
initialize clique tree (erase old results)
insert evidence y^t (for the latter time slice)
if t > 1 then
    φ_in ← φ_in * α^{t-1}
end if
make clique tree consistent
for each V do
    # optional loop for forward-only inference
    marginalize φ_V from a suitable clique
    normalize P(V^t|y^{1:t}) from φ_V
end for
α^t ← φ_out ↓ I_t^→
```

Forward inference[6] is achieved by applying the FORWARD-STEP algorithm repeatedly for each time step as in Algorithm 2. Note that the first time step ($t = 1$) does not have a message α^0 to receive and priors included in the model parameters are used instead. Forward inference needs to store only one instance of α^t message potential for relaying the information from time slice to the next one. Therefore, online inference requires essentially one time slice worth of space $O(s)$, and has linear time complexity $O(sT)$.

Algorithm 2 FORWARD-INFERENCE($\boldsymbol{y}^{1:T}$, DBN)

Input: evidence $\boldsymbol{y}^{1:T}$ and the model DBN
Output: $P(V^t|\boldsymbol{y}^{1:t})$, $\forall V$ and $\forall t \in [1, T]$

```
FORWARD-STEP (y^1, null, DBN)
for t ← 2 to T do
    FORWARD-STEP (y^t, α^{t-1}, DBN)
end for
```

[6] a.k.a. filtering, or online inference

When running the inference offline, there is opportunity to take the whole time series $\boldsymbol{y}^{1:T}$ into account at each time step. This naturally leads to more accurate results because of the additional data. As a matter of fact, valuable conclusions are achieved only by hindsight in some cases [36]. After computing the forward message potentials α^t for each time step, the effects of later evidence can be propagated back in time by the procedure described in Algorithm 3 (equivalent to the BACK-functions in [29]).

Algorithm 3 BACKWARD-STEP($\boldsymbol{y}^t$, γ^t, α^t, DBN)

Input: evidence vector $\boldsymbol{y}^t$, messages γ^t and α^t, and the model DBN
Output: $P(V^t|\boldsymbol{y}^{1:T})$, $\forall V$, and γ^{t-1}

initialize clique tree (wipe old results)
insert evidence $\boldsymbol{y}^t$
$\phi_{out} \leftarrow \phi_{out} * \frac{\gamma^t}{\alpha^t}$
make clique tree consistent
for each V **do**
    marginalize ϕ_V from a suitable clique
    normalize $P(V^t|\boldsymbol{y}^{1:T})$ from ϕ_V
end for
if $t > 1$ **then**
    $\gamma^{t-1} \leftarrow \phi_{in} \downarrow \boldsymbol{I}_{t-1}^{\rightarrow}$
end if

BACKWARD-STEP starts with a clean clique tree by initializing original model parameters. Then, outgoing clique ϕ_{out} is updated according to the old forward message α^t and the new knowledge γ^t from future. This is analogous to the message passing scheme described in Section 2.3.5. Finally, the clique tree is made consistent, posteriors $P(V^t|\boldsymbol{y}^{1:T})$ can be computed, and a new backward message γ^{t-1} is marginalized.

Full *forward-backward* inference algorithm starts with forward inference and saving the intermediate forward messages α^t as shown in Algorithm 4. (The bother of marginalizing $P(V^t|\boldsymbol{y}^{1:t})$ can be omitted, though.) At the end of the time series ($t = T$), the backward message γ^T becomes equal with forward message α^T because all of the data is taken into account already: results for both the forward and forward-backward inference are the same for the last time step.

The implementation used in this work makes the inference for each time

Algorithm 4 FORWARD-BACKWARD-INFERENCE($\boldsymbol{y}^{1:T}$, DBN)

Input: evidence $\boldsymbol{y}^{1:T}$ and the model DBN
Output: $P(V^t|\boldsymbol{y}^{1:T})$, $\forall V$ and $\forall t \in [1, T]$

```
FORWARD-STEP (y^1, null, DBN)
for t ← 2 to T do
    FORWARD-STEP (y^t, α^{t−1}, DBN)
end for
γ^T ← α^T
for t ← T to 1 do
    BACKWARD-STEP(y^t, γ^t, α^t, DBN)
end for
```

step again from scratch by initializing cliques with the original model parameters in the beginning of each step. Only the forward-phase results α^t between time steps are stored (and later replaced by γ^t) which helps in reducing the time complexity from $O(sT^2)$ to $O(sT)$ ([36] Sec. 15.2) by trade off: memory requirements rise from constant space[7] $O(s)$ to linear space $O(sT)$. There could also be a constant space smoothing algorithm without the trade off by running the forward inference for each time step separately.

Still, one has to remember the fact that the "constant" $O(s)$ is exponential in the number of interface variables [36], i.e., size of the forward message $O(d^{|\boldsymbol{I}_t^{\rightarrow}|})$, even if the structure of a single time slice is sparse.

2.5 Modular specification of DBNs

2.5.1 Networks with repetitive structure

In this work, belief networks were chosen to be defined using Hugin Net language developed by Hugin Expert A/S [15]. Other description formats do exist, such as the one using Extensible Markup Language (XML) [27], but Hugin Net language has some major advantages: better human readability due to minimal redundancy, wide support among other BN software, and adequate extensibility needed for specifying DBN models.

[7] Note: the space required by the 1.5DBN clique tree is assumed constant s (regarding time dimension)

Hugin Net language is a simple format for describing belief networks and influence diagrams [15]. For the purposes of this work, the language offers two elements for defining BNs: *node* and *potential* descriptions. The main functionality of a (discrete) node description is to declare the name of a random variable and enumerate its possible values. Additionally, it may have a label for a verbal description of the random variable, position coordinates for user interfaces displaying the BN, and optional application specific fields.

As a fictional example, if there was data from an opinion poll about favorite fruits, the probabilistic model could contain a variable F, defined in the following manner:

```
node F
{
    label = "Favorite fruit";
    position = (225 75);
    states = ("apple" "banana" "orange" "pear" "lemon");
    MY_field = "42";
}
```

The purpose of a potential description is to define the parents and the table of conditional probabilities for a child variable. As a special case, it describes prior probabilities for a variable without parents. This way, potential descriptions define both the structure and quantitative parameters of a BN without redundancy. All the the graphs are assumed to be directed, i.e., chain graphs which may have undirected edges are beyond this thesis. Consequently, all potentials are either prior probability distributions or conditional probability distributions for a single child variable given the parents.

To continue the example, variable F depends on age A and sex S, which are declared respectively as shown in Appendix A.1. Potential descriptions use the notation familiar from mathematics: the child variable is separated from the list of parents with a vertical bar. In addition to defining the order of accompanying data, the list of variables define the structure of the BN

implicitely to avoid redundancy. Finally, the actual data field contains a list[8] of probabilities. As shown below, the child variable F is used as the least significant index in ordering the data[9]. Parents are in significance order: the first parent variable S is used as the most significant index.

```
potential (F | S A)
{
  %  F = apple banana orange pear lemon
  data = ((( 0.2 0.3 0.1 0.3 0.1 )     %  S=male    A=child
           ( 0.3 0.3 0.1 0.2 0.1 )     %  S=male    A=teen
           ( 0.3 0.2 0.2 0.2 0.1 ))    %  S=male    A=adult
          (( 0.1 0.4 0.1 0.3 0.1 )     %  S=female  A=child
           ( 0.2 0.3 0.2 0.2 0.1 )     %  S=female  A=teen
           ( 0.2 0.2 0.2 0.2 0.2 )));  %  S=female  A=adult
}
```

Only one additional application specific field is needed to define a repetitive structure, such as the DBNs discussed in this work. To honor the principle of minimal redundancy, the model description consists of 1.5DBN, where the outgoing interface variables of the earlier time step have an additional field for naming the corresponding variable in the next time step. In other words, models have ordinary node descriptions V^t for one time slice ("1.0DBN") and all the nodes $V^{t-1} \in \boldsymbol{I}_{t-1}^{\rightarrow}$ included from the previous time slice must name the variable $V^t \in \boldsymbol{I}_t^{\rightarrow}$ they substitute during the current time step. This causes a natural constraint between the two nodes: they must have the same states enumerated in the node description. To simplify the implementation, the state labels also need to be in the same order.

The conventions of Hugin Net language were followed by naming the additional field as `NIP_next`. For instance, the fruit poll example can be extended into a DBN by forming a Markov chain out of the age variable A as shown in Fig. 2.5. To achieve this, the model needs to be supplemented by one additional node A^{t-1}, and instead of specifying a prior for A^t, there has to be a prior for A^{t-1} and potential $P(A^t|A^{t-1})$ defining the transition prob-

[8]The list can also be unstructured, that is, without inner parentheses.
[9]percent symbol % is the comment delimiter

abilities. Potential description doesn't have anything out of the ordinary, but the node A^{t-1} is described as follows:

```
node A0
{
    label = "Previous age";
    position = (50 75);
    states = ("child" "teen" "adult");
    NIP_next = "A";
}
```

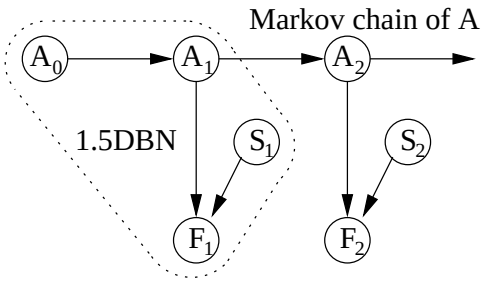

Figure 2.5: Possible DBN for fruit poll example

Second-order Markov models, such as an HMM where the hidden variable depends on two previous time steps, can be defined respectively by including the outgoing interface variables $V^{t-2} \in \boldsymbol{I}_{t-2}^{\rightarrow}$ to the model description. If the model is such that $V^{t-1} \notin \boldsymbol{I}_{t-1}^{\rightarrow}$ and V^{t-1} is therefore missing from the 1.5DBN, the division into time slices has to be redefined so that the successor can be named.

2.5.2 Inference engine

The process of transforming a static BN into a clique tree was dealt with in Section 2.3.1 and the way of specifying a dynamic BN in terms of one time slice was discussed above. To achieve a working inference engine, one more step needs to be taken as visualised in Fig. 2.6.

In principle, one could first create a clique tree according to a static BN and then somehow invent how the tree should be modified to achieve an

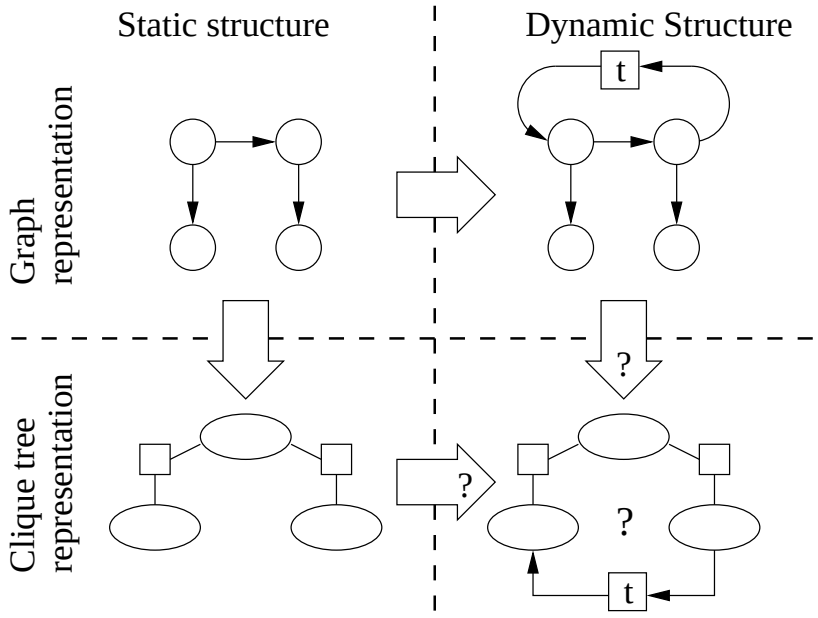

Figure 2.6: The problem with defining a partial clique tree

inference engine for a given DBN. After all, it could be argued that repetitive structure in graph domain implies repetitive structure in clique tree representation. Moreover, there can be only one sepset between two repeating parts of the tree: otherwise there would be a loop. This sepset between repeating elements could pass the message (similar to α^t) from time step to another during inference with a partial clique tree.

In general, the relationship between graph and clique tree representations can be confusing and ambiguous due to the possibility of multiple choices in graph triangulation. Clearly, it is easier to first define how a DBN repeats itself than try to automatically recognize repeating structure in a clique tree when given some strip of static BN. Therefore, the method presented in [28] was chosen to be applied in this work. First the user has the simple task of defining a 1.5DBN and then the software ensures that the corresponding clique tree can make the stepwise inference in a valid way.

The special case of a static network is also included. A model without `NIP_next` variables will perform the ordinary clique tree inference on a single static BN. If the evidence has more time steps than one, the inference is

performed independently for each time step as if the model didn't have any temporal edges.

2.6 Stochastic sampling

2.6.1 Drawing samples for a variable

Belief networks can be seen as models on how causes (parent variables) generate effects (child variables): generative models. Thus, the models discussed in this work can be used for creating time series samples which serve as synthetic data or part of numerical approximations.

In order to generate data according to discrete belief networks, samples need to be drawn from single-variable multinomial probability distributions. Luckily, this is as easy as dividing the unit interval into smaller buckets according to the individual probabilities and sampling a uniform probability distribution within the unit interval. The result is determined by the bucket where the random number generator happened to drop its sample.

More formally expressed, samples from the uniform probability density function (provided by common software libraries)

$$u(x) = \begin{cases} 1, & \text{if } 0 \leq x \leq 1 \\ 0, & \text{else} \end{cases} \tag{2.20}$$

are transformed into samples from multinomial set of probabilities by the obvious fact that

$$\begin{aligned} p(V = v_i) \stackrel{\text{def}}{=} p_i &= \int_{p_i} u(x)\mathrm{d}x \\ &= \int_{p_s}^{p_s+p_i} 1\mathrm{d}x, \forall i \end{aligned} \tag{2.21}$$

where $p_s = \sum_{j=1}^{i-1} p_j$ counts for the probability mass allocated for values before v_i (here the values have some constant order).

2.6.2 Sampling according to a dependency network

The causal process defined by a Bayesian network can be simulated by so called forward sampling. Such a task would require prohibitive amount of memory or processor time in a general large scale and dense BN [2]. In this work though, the models are assumed to consist of small networks loosely chained together[10] to enable exact (forward) inference. Consequently, forward sampling is guaranteed to be possible.

Another aspect is that only sampling as predictive inference is considered. This means that the entire BN is to be sampled without any prior observed evidence (like direct sampling in [36]) and e.g. diagnostic reasoning by sampling methods is beyond the scope of this work. Concentrating on sampling without prior data avoids the possible pitfalls associated with unlikely observations [2] as in rejection sampling.

First task in the sampling process is to order the variables topologically, i.e., parent variables before children. This ensures that parents of each family of variables are sampled before the child variable.

According to the decided topological order, sampling starts from a variable without any parents. Samples for such variables come from the prior probability distributions $P(V)$ defined among the model parameters. Child variables get their samples according to the conditional probabilities $P(V|\boldsymbol{\Pi}_V)$ by first selecting the vector $P(V|\boldsymbol{\Pi}_V = \boldsymbol{\pi}_V)$ corresponding to the previously drawn samples $\boldsymbol{\pi}_V$ of parent variables.

In an inference engine running dynamic Bayesian networks, the sampling process is accomplished naturally by selecting a variable according to a topological order, sampling according to the current likelihood of the variable, setting the sample as hard evidence and making the clique tree consistent to start with the next variable. After all the variables in a time slice get sampled, the gained evidence is transferred to the next time slice as a starting point for the next iteration.

The direct sampling algorithm (Prior-Sample) presented in [36] (Sec. 14.5) can be used with inference engine of a DBN as shown in Algorithm 5. Traversing the model piece by piece does not interfere with the topological

[10] i.e., networks with special topology as referred in [2]

Algorithm 5 DBN-PRIOR-SAMPLE(T, DBN)

Input: $T > 0$, and DBN (defines probability distributions and topology)
Output: consistent samples v^t, $\forall V$ and $\forall t \in [1, T]$

```
Sort variables topologically
for t ← 1 to T do
    if t > 1 then
        φ_in ← φ_in * α^{t-1}
    end if
    for each V in topological order do
        infer P(V) (given π_V)
        v^t ← a random sample from P(V)
        enter v^t as new evidence
        make join tree consistent
    end for
    α^t ← φ_out ↓ I_t^→
end for
```

order because temporal arcs are assumed pointing forward in time. Thus, current time slice does not have any children of the next time slice.

Since interface variables d-separate the earlier half from the latter half of the time slice model, sampling is independent on each side. As a consequence, unnecessary child variables in the earlier $(t-1)$ half should be omitted from the model (1.5DBN) or else they are likely to generate samples that are inconsistent with the ones generated by the latter half.

Chapter 3

Parameter estimation

3.1 Likelihood of evidence

3.1.1 JPD of arbitrary variables

Equation 2.8 tells how to compute the joint probability distribution of *all* variables using the tables stored in a clique tree. In practice, computing such a distribution would be pointless, since the primary motivation for using a clique tree in the first place is to save memory by avoiding huge tables required by probability distributions of many variables. However, some algorithms (e.g. the EM algorithm [8, 23]) need to know the likelihood of the data and therefore at least single elements of a joint probability distribution of arbitrary variables need to be computed.

In case it is needed, joint probability distribution of arbitrary variables $P(\boldsymbol{V})$ can be computed by traversing the (consistent) clique tree in depth-first order and multiplying clique and sepset potentials as shown in Eq. 2.8. Since the variables of interest (set $\boldsymbol{V}$) do not usually include all the variables (i.e., $\boldsymbol{V} \subset \boldsymbol{U}$), some marginalization is needed:

$$P(\boldsymbol{V}) = \sum_{U \setminus V} P(\boldsymbol{U}). \tag{3.1}$$

Computing the entire joint probability distribution $P(\boldsymbol{U})$ first would either use too much memory ($O(d^{|\boldsymbol{U}|})$) or at least make the use of a clique tree

meaningless. In order to avoid pointless memory consumption, some of the marginalizations can be done between the multiplications as in the variable elimination algorithm.

An example: the variables are $\boldsymbol{U} = \{A, B, C, D, E, F, G, H\}$ and the cliques are $\boldsymbol{C}_1 = \{A, B, C\}$, $\boldsymbol{C}_2 = \{A, B, D\}$, $\boldsymbol{C}_3 = \{A, E\}$, $\boldsymbol{C}_4 = \{B, F\}$, $\boldsymbol{C}_5 = \{C, G\}$, and $\boldsymbol{C}_6 = \{G, H\}$. The joint probability distribution of variables $\boldsymbol{V} = \{C, G, F\}$ can be calculated in the following way:

$$P(\boldsymbol{V}) = \sum_{A,B} \phi_{ABC} * \left[\sum_{D} \left(\frac{\phi_{ABD}}{\phi_{AB}} * (\sum_{\emptyset} \frac{\phi_{BF}}{\phi_B}) * (\sum_{E} \frac{\phi_{AE}}{\phi_A}) \right) \right] * \left[\sum_{\emptyset} \left(\frac{\phi_{CG}}{\phi_C} * (\sum_{H} \frac{\phi_{GH}}{\phi_G}) \right) \right]. \tag{3.2}$$

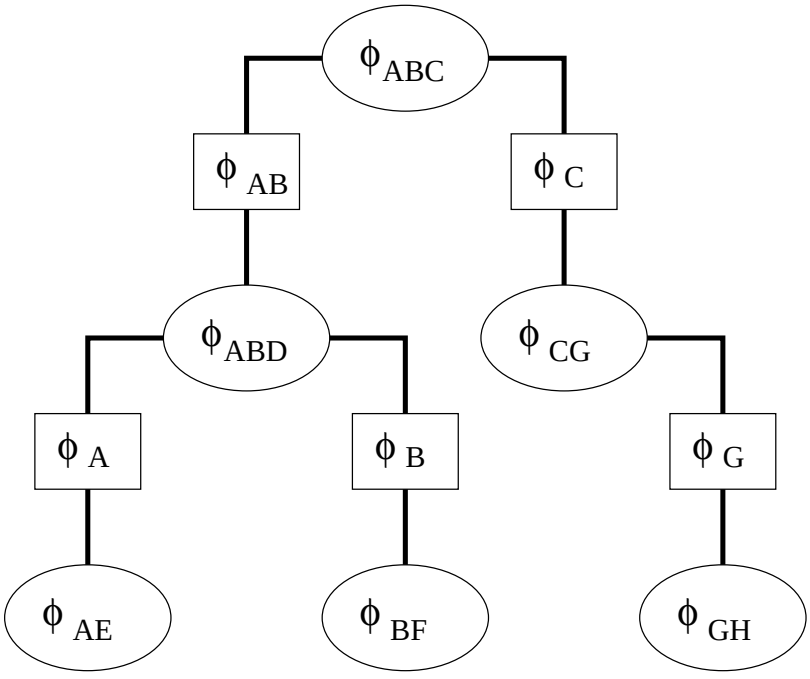

Figure 3.1: An example of a clique tree

As the example shows, the clique tree is traversed in depth-first postorder marginalizing away all the variables which are not included in $\boldsymbol{V}$ or the current parent clique in the tree (see Fig. 3.1). Again, the running intersection

property determines the opportunities for marginalizations in a recursive algorithm as seen in Algorithm 6. A variable Z can be marginalized away from the current result ϕ_U if it either won't be included in the final result ($Z \notin \boldsymbol{V}$) or it doesn't appear in the parent clique ($Z \notin \boldsymbol{I}$). The recursion is started by calling JOINT-DISTRIBUTION($\boldsymbol{V}$, $\emptyset$, ϕ_X) for an arbitrary root clique X.

Algorithm 6 JOINT-DISTRIBUTION($\boldsymbol{V}$, $\boldsymbol{I}$, ϕ_{Root})

Input: desired set of variables $\boldsymbol{V}$,
intersection $\boldsymbol{I}$ of ϕ_{Root} and its parent,
and the clique ϕ_{Root} (root of the subtree)
Output: $P(\boldsymbol{V} \cup \boldsymbol{I})$

mark *Root* clique
$\boldsymbol{U} = \boldsymbol{V} \cup Root$
$\phi_U \leftarrow \mathbf{1} * \phi_{Root}$
for each unmarked neighbor clique ϕ_C **do**
  sepset $\boldsymbol{S} = \boldsymbol{C} \cap Root$
  $\phi_U \leftarrow \frac{\phi_U}{\phi_S}$
  $\boldsymbol{I}_2 = \boldsymbol{S} \setminus \boldsymbol{V}$
  $\phi_U \leftarrow \phi_U *$ JOINT-DISTRIBUTION($\boldsymbol{V}$, $\boldsymbol{I}_2$, ϕ_C))
end for
$P(\boldsymbol{V} \cup \boldsymbol{I}) \propto \phi_U \downarrow (\boldsymbol{V} \cup \boldsymbol{I})$

Still, this is waste of resources from learning point of view, since the intermediate results (like ϕ_U) use possibly more memory than the clique tree and only the single value of evidence likelihood is needed. Besides, the above method is not applicable to soft evidence. Following subsection describes a better solution.

3.1.2 Ratio of probability masses

The method described above is naive in the sense that it may waste huge amounts of memory and processor time while computing intermediate results. Since only the likelihood of given data is needed in the EM algorithm, there exists a much easier way [17, 23, 37] which facilitates likelihood computations as a by-product of forward inference.

The idea is to compute the probability mass contained by the potential

tables before and after entering the evidence. For a clique tree of a static BN, the mass is defined as the sum of clique weights minus the sum of sepset weights (Eq. 3.3).

$$M = \sum_i \sum_c \phi_{C_i}[c] - \sum_j \sum_s \phi_{S_j}[s], \tag{3.3}$$

There are a couple of aspects to note in order to explain how the above expression ends up being proportional to the likelihood of evidence. First of all, it is assumed that the evidence distributions, which are multiplied into clique potentials, are normalized (i.e., the sum of elements equals one).

The elementary case is that all the variables belong to single clique and the evidence is hard. When the evidence is entered into the clique, all the elements corresponding to the observation become multiplied by one and all the other elements in the clique potential become zero. Consequently, adding up over the entire clique is equivalent to selecting the elements corresponding to the observation and summing their weight (as done in [23, 37]). Finally, the sum needs to be normalized by dividing with the original probability mass because potential tables generally don't sum up to one.

When using only hard evidence, most of the terms in Eq. 3.3 become zero and summing over the potential tables is partly redundant, but the case is different when the evidence is soft. The procedure applies also when the clique potential becomes "reweighted" by any normalized evidence distribution. As a second example, suppose the single clique $\boldsymbol{C}$, containing the binary variables $\{X, Y, Z\}$, receives evidence $P(X) = [0.1, 0.9]$. Potential elements corresponding to $P(X = x_1, Y, Z)$ become multiplied by 0.1 and $P(X = x_2, Y, Z)$ is multiplied by 0.9. Now the clique weight is

$$\begin{aligned} M|x &= \sum_{\boldsymbol{C}} \phi_{\boldsymbol{C}} \\ &\propto \sum_{Y,Z} (0.1 \cdot P(X = x_1, Y, Z) + 0.9 \cdot P(X = x_2, Y, Z)) \\ &= 0.1 \cdot P(X = x_1) + 0.9 \cdot P(X = x_2), \end{aligned} \tag{3.4}$$

which is the evidence likelihood since $\phi_{\boldsymbol{C}} \propto P(X, Y, Z)$.

In a more general setting, there are more cliques and the sum of separator set weights is subtracted to eliminate the probability mass included twice in

neighboring cliques. After entering the evidence and making the clique tree consistent, likelihood of given evidence (according to the model) is computed as the probability mass left in the clique tree ($M|\boldsymbol{y}$) normalized with the mass (M_0) that existed before entering the evidence ($\boldsymbol{y}$).

$$\mathcal{L}(\boldsymbol{y}) = \frac{M|\boldsymbol{y}}{M_0} \tag{3.5}$$

Conditional likelihoods can be computed by choosing another point of reference, as shown in Eq. 3.6. This is useful when calculating factors of the likelihood of a time series by conditioning on the previous time step as discussed in the next section.

$$\begin{aligned} \mathcal{L}(\boldsymbol{y}|\boldsymbol{x}) &= \frac{\mathcal{L}(\boldsymbol{y},\boldsymbol{x})}{\mathcal{L}(\boldsymbol{x})} \\ &= \frac{M|\boldsymbol{y},\boldsymbol{x}}{M_0} \div \frac{M|\boldsymbol{x}}{M_0} \\ &= \frac{M|\boldsymbol{y},\boldsymbol{x}}{M|\boldsymbol{x}} \end{aligned} \tag{3.6}$$

3.1.3 Logarithmic likelihood of time series

Dynamic Bayesian networks, such as the HMM, suffer from the fact that the likelihood of a (long) time series is usually too small to be successfully computed with a real world computer. Instead, the logarithm of the likelihood behaves well from the numerical computation point of view. On average, it decreases linearly as the length of time series grows and it can be expressed as a sum of logarithmic quantities for each time step because of the well-known fact that $\log(ab) = \log(a) + \log(b)$.

As a matter of fact, the inference procedure has the same problem: without additional normalization, absolute weights of message-passing potentials (α^t and γ^t) approach zero as t increases. This results in considerable numerical error with long time series, because of the limitations of conventional computer arithmetics. Therefore, message-passing potentials need to be normalized between time steps and it is assumed that a single time slice is small enough to avoid excessive numerical errors.

In order to compute the logarithmic likelihood gradually as a sum of log-

arithmic terms (i.e., in a numerically stable manner), the likelihood of time series has to be represented as a product of factors: one for each time slice. Fortunately, it was already noted that the interface variables d-separate past time slice from future and temporal edges point forward in time. Thus likelihood can be factorized as follows:

$$\begin{aligned} \mathcal{L}(\boldsymbol{y}^{1:T}) &= P(\boldsymbol{y}^1) * \prod_{t=2}^{T} P(\boldsymbol{y}^t | \boldsymbol{I}_{t-1}^{\rightarrow}) \\ \Rightarrow \log(\mathcal{L}(\boldsymbol{y}^{1:T})) &= \log(P(\boldsymbol{y}^1)) + \sum_{t=2}^{T} \log(P(\boldsymbol{y}^t | \boldsymbol{I}_{t-1}^{\rightarrow})) \end{aligned} \tag{3.7}$$

Algorithm 7 shows how the original FORWARD-STEP procedure has to be updated in order to compute likelihood of data as a by-product of forward inference. To compute conditional likelihood of the new data, post-evidence probability mass (M_2) is compared with the one after passing message from previous time slice (M_1).

Algorithm 7 FORWARD-STEP-LL($\boldsymbol{y}^t$, α^{t-1}, DBN)

Input: evidence $\boldsymbol{y}^t$, message α^{t-1}, and the model DBN
Output: $P(V^t|\boldsymbol{y}^{1:t})$, $\forall V$, α^t, $log(\mathcal{L}(\boldsymbol{y}^{1:t}))$

initialize clique tree (erase old results)
if $t > 1$ **then**
    $\phi_{in} \leftarrow \phi_{in} * \alpha^{t-1}$
end if
make clique tree consistent
$M_1 = \sum \phi_C - \sum \phi_S$
insert evidence $\boldsymbol{y}^t$ (for the latter time slice)
make clique tree consistent
$M_2 = \sum \phi_C - \sum \phi_S$
$log(\mathcal{L}(\boldsymbol{y}^t)) = log(\mathcal{L}(\boldsymbol{y}^{t-1})) + (log(M_2) - log(M_1))$
for each V **do**
    # optional loop for forward-only inference
    marginalize ϕ_V from a suitable clique
    normalize $P(V^t|\boldsymbol{y}^{1:t})$ from ϕ_V
end for
$\alpha^t \leftarrow \phi_{out} \downarrow \boldsymbol{I}_t^{\rightarrow}$
normalize α^t

3.2 The EM algorithm

3.2.1 CPD estimation using inference engine

Since the main focus is on probabilistic models with hidden variables and possibly other missing data, a natural choice for learning is the Expectation-Maximization [8, 39, 35, 23, 1, 36] (EM) algorithm. The goal of this iterative algorithm is to find a maximum likelihood estimate for the model parameters: conditional probability distributions (CPD) for child variables and priors for variables without parents. Learning the structure of a BN [4, 36] is beyond the scope of this work.

As noted for example in [29], the CPD of each variable can be estimated independently because the graph is directed. Thus the algorithm is straightforward to build on top of the forward-backward inference. It provides the posterior probability distributions of variable families at each time step given an entire time series, $P(\boldsymbol{F}_{V_i}^t|\boldsymbol{y}^{1:T})$, as shown in Algorithm 8 which is used instead of BACKWARD-STEP defined earlier.

Algorithm 8 BACKWARD-STEP-LL($\boldsymbol{y}^t$, γ^t, α^t, DBN)

Input: evidence vector $\boldsymbol{y}^t$, messages γ^t and α^t, and the model DBN
Output: $P(\boldsymbol{F}_V^t|\boldsymbol{y}^{1:T})$, $\forall V$, γ^{t-1}

```
initialize clique tree (wipe old results)
insert evidence y^t
φ_out ← φ_out * γ^t / α^t
make clique tree consistent
for each V do
    marginalize φ_{F_V} from a suitable clique
    normalize P(F_V^t | y^{1:T}) from φ_{F_V}
    if t = 1 ∨ V ∉ I→_{t-1} then
        N̂(F_V) += P(F_V^t | y^{1:T})
    end if
end for
if t > 1 then
    γ^{t-1} ← φ_in ↓ I→_{t-1}
    normalize γ^{t-1}
end if
```

Algorithm 9 shows the necessary steps for computing expected counts

$\widehat{N}(\boldsymbol{F}_V)$ for each variable family during one time series. At first, forward inference and likelihood calculations are taken care of by FORWARD-STEP-LL procedure and finally the expected counts are accumulated by repeating backward inference in BACKWARD-STEP-LL. Note that it accumulates expected counts, $\widehat{N}(\boldsymbol{F}_V) = \widehat{N}(V)$, for $V \in \boldsymbol{I}_{t-1}^{\rightarrow}$ only for the first time step of a time series. This is due to the fact that the priors of variables in the earlier time slice ($P(V)$, $V \in \boldsymbol{I}_{t-1}^{\rightarrow}$) account only for the beginning of a time series.

Algorithm 9 E-STEP($\boldsymbol{y}^{1:T}$, DBN)

Input: evidence $\boldsymbol{y}^{1:T}$ and the model DBN
Output: $\sum_t \widehat{N}(\boldsymbol{F}_V^t|\boldsymbol{y}^{1:T})$, $\forall V$, and $\log(\mathcal{L}(\boldsymbol{y}^{1:T}))$

```
FORWARD-STEP-LL(y^1, null, DBN)
for t ← 2 to T do
    FORWARD-STEP-LL(y^t, α^{t−1}, DBN)
end for
γ^T ← α^T
for t ← T to 1 do
    BACKWARD-STEP-LL(y^t, γ^t, α^t, DBN)
end for
```

Similarly to the Bayes rule, update of the model parameters is given by the normalized expected counts [23, 36]:

$$P(V|\boldsymbol{\Pi}_V)^{new} = \frac{\sum_t \widehat{N}(\boldsymbol{F}_V)}{\sum_t \widehat{N}(\boldsymbol{\Pi}_V)} \tag{3.8}$$

A pseudocode implementation of the above normalization and parameter update is shown in Algorithm 10. If a potential table containing $\widehat{N}(\boldsymbol{F}_V)$ is arranged so that the child variable V behaves as the "least significant" index, it is only a matter of normalizing each c_V sized block of the table. In other words, selecting any configuration of parent variable values will result in a properly normalized table of probability values for the child variable. The divisor, $\widehat{N}(\boldsymbol{\Pi}_V) = \sum_V \widehat{N}(\boldsymbol{F}_V)$, is implicitly computed during the aforementioned block-wise normalization to make the blocks sum to one.

Algorithm 10 M-STEP($\widehat{N}(\boldsymbol{F}_V)$, DBN)

Input: sum of expected counts $\widehat{N}(\boldsymbol{F}_V)$, $\forall V$, and the model DBN
Output: updated DBN with $P(V|\boldsymbol{\Pi}_V)^{new}$, $\forall V$

```
for each V do
    φ_{F_V} ← N̂(F_V)
    i ← 1
    while i ≤ |φ_{F_V}| do
        normalize block φ_{F_V}[i : (i + c_V − 1)]
        i ← i + c_V
    end while
    update DBN with P(V|Π_V)^new = φ_{F_V}
end for
```

3.2.2 Stopping criterion

The learning process requires also a stopping rule before it deserves to be called an algorithm. Most literature about EM algorithm, like [1, 36], seem to concentrate on describing the parameter update rules but don't elaborate on how to determine when the estimated parameters are good enough. In some cases, there can be an application-specific stopping criterion, like cross-validation with another method or data set, but in this work it is considered important to use as generally applicable stopping rule as possible.

The two most popular choices seem to be monitoring the maximum absolute change in parameters and monitoring the change in logarithmic likelihood of training data (given the model). Although the parameter values are shown to converge during EM training [39, 35], it is possible at least in principle that the labels of hidden variables switch between iterations because the labels don't have fixed identity. Therefore, it may be difficult to compare two sets of parameters for equality. On the other hand, the logarithmic likelihood of evidence provides a single unambiguous monotonic value for monitoring the progress, as was done e.g. in [23].

The procedure for computing the logarithmic likelihood of a single time series, $\log \mathcal{L}(\boldsymbol{y}_{(k)})$, was described in Alg. 7, but the training data may contain time series of different lengths: $|\boldsymbol{y}_{(k)}| = T_k$. Since each time step is taken into account equally during the parameter estimation, also the likelihood values should be treated accordingly. The method used in this work is to sum up

the logarithmic likelihood of each time series and divide it with the total number of time steps, as shown in Eq. 3.9.

$$\log \mathcal{L}_{ave} = \frac{\sum_k \log \mathcal{L}(\boldsymbol{y}_{(k)})}{\sum_k T_k} \tag{3.9}$$

Considering the fact that each of the logarithmic likelihoods $\log \mathcal{L}(\boldsymbol{y}_{(k)})$ is a sum over T_k time steps, the result is one kind of average logarithmic likelihood of one time step. It doesn't take into account the situations where missing data causes some time steps to have higher likelihood than those with more observations. But on the other hand, $\log \mathcal{L}_{ave}$ is still a consistent value for monitoring convergence because the training data doesn't come up with new observations or missing values during EM algorithm.

Finally, the actual EM algorithm used in this work is shown in Alg. 11. It repeats estimation process using Algorithms 9 and 10 defined earlier and compares the change in average logarithmic likelihood to the minimum required change δ given as a parameter.

Algorithm 11 EM-LEARN(δ, $\boldsymbol{y}_{(1:K)}$, DBN)

Input: threshold δ, set of time series data $\boldsymbol{y}_{(1:K)}$, and the model DBN
Output: updated DBN with estimated parameters, and log-likelihood score ($\log \mathcal{L}_{ave}$) from each iteration

```
set random parameters for DBN
T_tot = Σ_k T_k
log L_ave ← −∞
repeat
    s ← 0
    for each k do
        E-STEP(y_(k), DBN)
        s ← s + log L(y_(k))
    end for
    Δ ← s/T_tot − log L_ave
    log L_ave ← s/T_tot
    M-STEP(N̂(F_V), DBN)
until Δ < δ
```

3.2.3 Limitations of the algorithm

There are few shortcomings in using the EM algorithm described above and in probabilistic modeling in general. The first problem is that the events not presented in the training data are considered impossible. Due to the way in which the likelihood is computed, it is obvious that EM algorithm avoids models where probability mass is wasted on unseen combinations. For example, an ordinary HMM will have zero emission probability for the observed variable value $Y = y_3$, if y_3 never occurs in the training data. The same situation applies also for a combination of variable values e.g. in an HMM with two observed variables (two-dimensional observations).

This complication is a well-known trait of machine learning: models with a lot of parameters will need a lot of training data or otherwise they will experience overfitting. An extreme example of this would be a model whose hidden variables are able to enumerate all the time steps in training data. Such a model may just memorize the training samples, i.e., report a positive likelihood when given a time series from training set but consider all other observation sequences impossible.

Some probability distributions for continuous variables, like the Gaussian distribution, don't usually[1] suffer from the problem of assigning zero probabilities to unseen events, which corresponds to log-likelihood of $-\infty$. On the other hand, having separate probability values for discrete events sets upper limits for the likelihood function. This ensures that likelihood doesn't increase without bound as it might do with Gaussian components assigned to single samples [35] (again, with zero variance).

Another problem is that errors in training data are also included in the estimated model. In particular, combinations of observations that should be impossible cause positive likelihoods if included in training data. This is a very common difficulty in machine learning domain since real world data sets may have errors that are challenging to remove. For example, it may be hard to assess what is an interesting unique finding and what is an error in measurement when exploring a previously unknown data set, like genetic data. Alternative option for screening errors from the training data

[1] except in the degenerate case of zero variance

is to extend the probabilistic model to include the process responsible for the errors. In any case, the learning algorithm fits the model "blindly" to the training data and the only other source of assumptions is the structure of the BN.

The third limitation is that the states of hidden variables are ambiguous. This means that the same probabilistic model is equally good in explaining the training data regardless of the names or order of values for hidden variables. The consequence is that there are usually several maximum likelihood solutions due to the "label switching" difficulty [35]. Since the machine can't miraculously invent correct labels, the hidden variables should have some arbitrary names for their values before the EM algorithm. Afterwards, the labels can be changed to something more descriptive if they are identified to correspond to real world events.

The last feature noted here is that the EM algorithm finds *local* maxima of the likelihood function [35]. So, in addition to having several equally good solutions, there may exist a better solution than the first one found by EM algorithm. Therefore, it is worth trying EM algorithm several times with randomized initial model parameters and see if some of the found parameter sets gives higher likelihood for the data.

Chapter 4

Experiments and results

4.1 Software

Some software for belief network modeling exists already. Those include:

- Hugin by Hugin Expert A/S,
- BayesiaLab by Bayesia SA,
- Netica by Norsys Software Corp.,
- Bayesian Network tools in Java (BNJ) by Dr. William H. Hsu and his team at Kansas State University,
- SamIam by the Automated Reasoning Group of Professor Adnan Darwiche at UCLA,
- Bayes Net Toolbox for Matlab by Kevin Murphy and OS community,
- Open Bayes for Python by OS community,

and more, but a new software library was implemented for this thesis. The choice of using C as the programming language made the task slightly tedious compared to using a higher level language or a platform such as Matlab. Despite the effort invested in debugging the software memory management and re-inventing some technical details etc., the project was considered a very educational and an invaluable programming experience, and more importantly, a software development experience.

The software library consists of about 14000 lines of ANSI C [19] and uses techniques described earlier in this work to implement the following features. First of all, there is a parser for reading discrete time sliced DBNs from Hugin Net language files. Also ordinary discrete static BNs can be used as a special case. After reading a model, the software forms an inference engine by constructing a clique tree suitable for time sliced inference. Respectively, there is a function for writing a DBN into a file in Hugin Net format.

There are also methods for reading and writing time series data files. File formats use simple comma-separated text fields and one line for each time step (record). Hard evidence is stored in a file so that the first non-empty line has the names of the variables corresponding to each of the comma-separated columns. Subsequent non-empty lines contain the data and must have equal number of fields: missing observations are denoted with the reserved word `null`. Consequently, node declarations in the Net files should avoid using `null` as a label – unless missing piece of data needs to be considered as an observation itself. On the other hand, empty lines are used for separating time series so that several time series can be written into a single file.

The file format for soft evidence is slightly different. Since the data requires one dimension more, it was decided that a single file contains data only for one variable V. Instead of naming different variables, the first non-empty line names the values v_i of the variable. Respectively, each subsequent record contains likelihoods $\lambda_V[i] \in [0, 1]$ for each of the values.

Of course, the software is modular enough to be interfaced with any kind of database with a proper programming library for the interaction. Once the evidence is transformed into a suitable data structure used internally by the software, the inference engine can be used for time sliced forward or forward-backward inference. The software library has different layers of abstraction, so that one can use a ready-made inference procedure which transforms a set of time series data into a set of posterior probabilities, *or* one can create custom inference steps.

An inference engine can also be used for prior sampling as discussed in Section 2.6: all one needs to specify is the time series length to the library procedure. Also the likelihood computations and the EM learning algorithm was implemented as described in Section 3.2. Additionally, there are simple command line programs for actually using each of the library features.

The software library was tested successfully against a test oracle (the Hugin Lite by Hugin Expert A/S) with small static BNs during development. For instance, a hierarchical regime-switching model [12] with four time steps and randomly chosen parameters gave identical inference and log-likelihood results regardless of using:

- a static network with Hugin Lite,
- the same static network, or
- an equivalent DBN with the software written for this work.

4.2 Artificial data

4.2.1 Original model and data

In the first experiment, the objective was to demonstrate the capabilities of the presented methods and software and show use cases in the field of time series analysis. Another goal was to test the sanity of software implementation by having an idealized setting where artificial data is sampled from a known model.

Perhaps the main use of probabilistic graphical models is to represent real world processes at some level of detail. Unfortunately, the phenomena behind the data are seldom completely understood (quantitatively) or become too complex to model precisely, providing the reason for using the EM learning algorithm in the first place. Therefore, it was necessary to imitate a real world setting with a known model and complete data sampled from it (as in Fig. 4.1) in order to test and demonstrate the learning algorithm (Chapter 3).

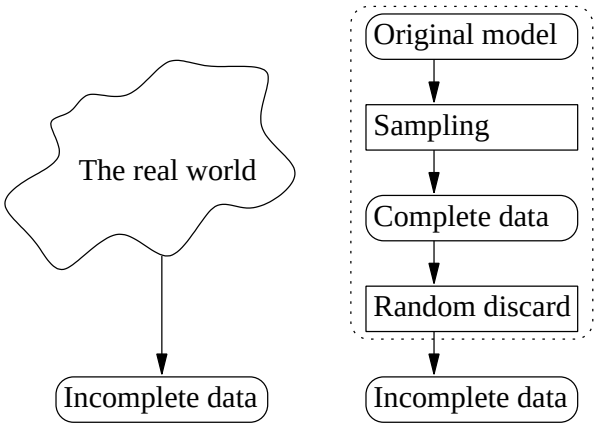

Figure 4.1: Way to imitate real world data

To make the setting more interesting, the DBN used as an example should not be trivially equivalent to a Hidden Markov Model (HMM). This is the case especially when some factors of the hidden state are partially observed. The first model, called "original model" below, is shown in Fig. 4.2 and it was used for generating synthetic data. As can be seen from the specification in Appendix A.2, variable A has three states, persistence variable C has six states, and both B and D are binary. The conditional probabilities $P(C^t|D^t, C^{t-1})$ show that the state of C usually stays constant and changes to the next one every now and then, but the state change is forced when $D^t = 1$.

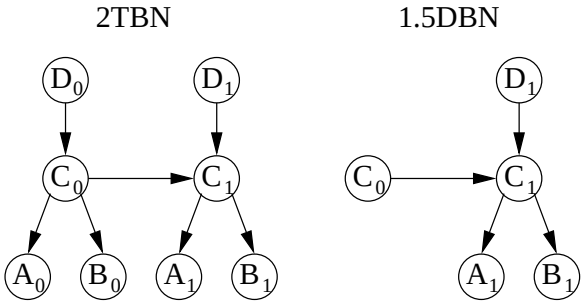

Figure 4.2: The original model for sampling

Some of the reasons for choosing such a model are as follows:

- A and B are used for observed multivariate data
- some of the observations are made partially incomplete by omitting values of B
- cardinalities of A and B are kept low for clarity and to limit the number of parameters
- C summarizes the hidden state and provides persistence over time
- cardinality of C provides enough explanations for different combinations of observed A and B
- D is a transient effect causing changes in the hidden state
- D can be occasionally observed to provide partial information about the otherwise hidden process behind the (A, B) data

The model was sampled for 4000 time series with 80 time steps each and it took about 2 minutes 20 seconds on an average PC workstation of the day. After that, part of the produced complete data was randomly discarded and values of C were completely omitted. 50% of B and 20% of D were left intact: i.e. B is missing at random half of the time and D is missing at random most of the time. Values of different variables and time steps are missing independently of each other which ensures that nothing particular could be inferred from lack of sample at certain point.

2000 time series of the resulting incomplete data was subsequently used for EM learning and the other 2000 series for inference demonstrations. All the original complete data was kept for testing. One of the complete time series is shown in Fig. 4.3. Some of the properties of the model discussed above can be seen in the figure: the event $D = 1$ forces a state change in C, and different states of C cause different distributions in observations of $A \times B$. The state of C also tends to "rotate" as expected.

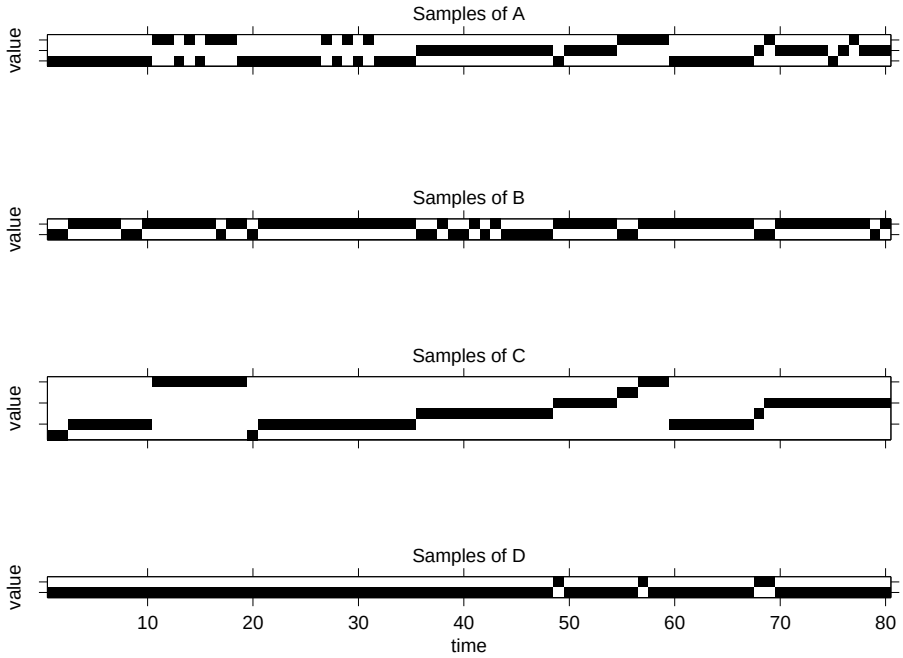

Figure 4.3: One of the complete artificial time series.

4.2.2 Alternative expert model

Usually, the processes behind new data are not well known. Therefore it was considered interesting to create a second experimental setting where the structure of the original model is considered unknown and another expert model[1] is used for parameter estimation and analysis instead. The alternative model (Fig. 4.4) was designed to be slightly different than the original one. The model resembles the variation of HMM used in [13], but instead of deriving and implementing custom inference rules, it is only required to specify a suitable DBN and use the framework described in this work.

One of the most notable differences to the original model is that the variable D is omitted. In a real life situation, this might be due to the fact that D is rarely observed or it is considered irrelevant to the phenomenon under study.

[1]The name *expert* model refers to the fact that specifying an independence structure requires professional insight of a domain expert

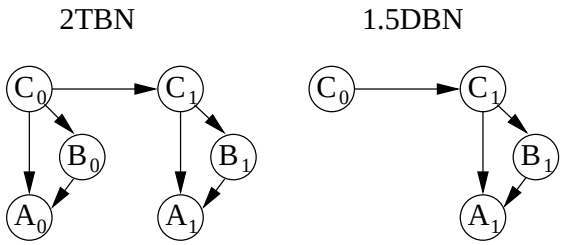

Figure 4.4: Alternative model for analysis

Another significant difference is that the hidden persistence variable C has less states ($c_C = 5$) than in the original model. This tries to demonstrate how well a model with insufficient amount of hidden states can explain the observed data. On the other hand, the alternative model makes one independence assumption less by asserting that A^t depends directly on B^t.

4.2.3 Parameter estimation

As discussed in Section 3.2, the EM learning algorithm receives a known model structure as input and uses incomplete data to produce estimated model parameters and a learning curve as a result. Thus, the experimental setting for demonstrating parameter learning is as shown in Fig. 4.5. Half of the generated artificial data was used for learning with two different model structures: the original and the alternative model. The final estimated models are found in Appendix A.3 and A.4 respectively.

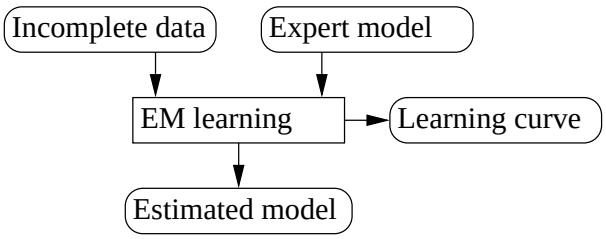

Figure 4.5: Common setting for EM learning

The learning curves in Fig. 4.6 show the average logarithmic likelihood of evidence during parameter estimation, as defined in Section 3.1.3 and in Eq. 3.9. The model with the original structure took about 3 hours 40

minutes (220 iterations) to estimate while the threshold value was set to 1/1000000. The alternative model was estimated faster: it took 22 minutes (48 iterations), but the threshold value was set to 1/100000.

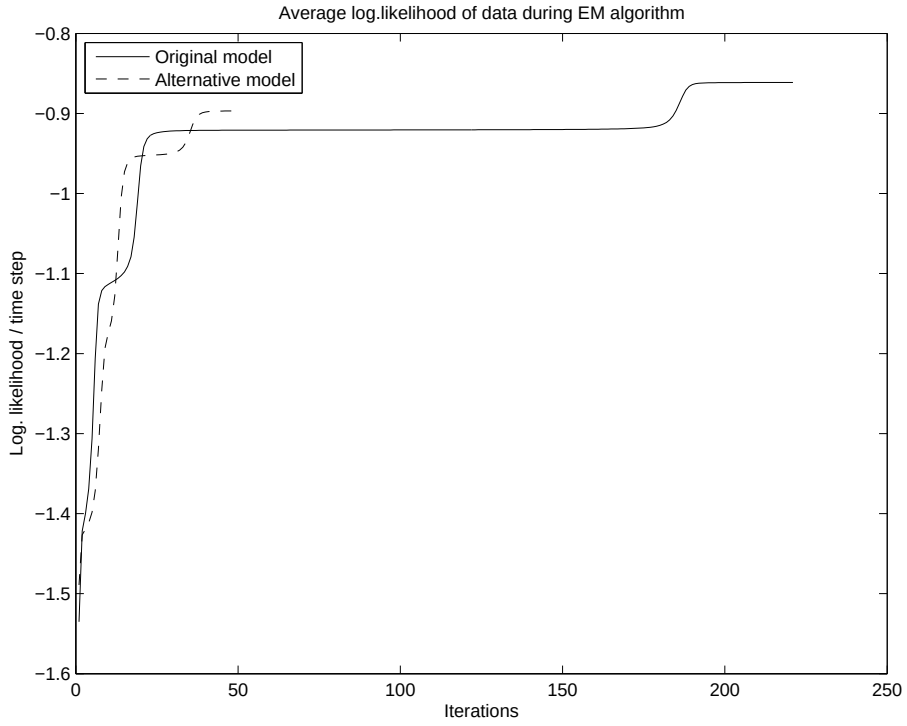

Figure 4.6: Average log-likelihood during EM algorithm

Each level of likelihood in the learning curve represents how well the hidden states of a model can be utilized to explain the observed data at each iteration of the learning procedure. In this case, and presumably in many other cases as well, the learning curves look as if the likelihood was "quantized" into steps. One reason for this could be the property, that the HMM-like models used here act like one kind of "soft" and context sensitive clustering: EM algorithm tries to assign a different distribution of observations for each of the hidden states while also considering that the hidden state shouldn't change arbitrarily between time steps.

What happens with the model when the likelihood suddenly jumps one step higher in a few iterations? Probably two or more hidden states have explained the same kind of data together at first but then the states become

separated and specialize to explain "separate clusters". Anyhow, it is not certain whether there is an even better set of parameters at this point of the experiment. It could be so that the threshold values were too large and there could be one step more in the likelihood. On the other hand, limited amount of data, numerical resolution, and execution time make it unreasonable to set an arbitrarily small threshold value.

As can be seen in Fig. 4.6, the original model reached higher average logarithmic likelihood (-0.86128) than the alternative model (-0.89686). There are several potential reasons for this. First, the structure of the alternative model is different from the original and therefore may not be able to fit the data as well. Secondly, the lack of hidden states provides weaker abilities to explain the data. A third reason might be that the observations about variable D aid the original model to adjust the parameters related to the hidden state and its changes.

It should be noted, that the likelihood values may not be directly comparable between models of different structure. As an extreme example, if all the data was made missing, computed likelihood would be one (and log-likelihood zero). Now that the data about variable D was ignored in the alternative model, it didn't have to waste probability mass on wrong predictions about its values, but then again, it missed the information contained in the data.

4.2.4 Inference results

The other half of the generated data was used for inference demonstrations to avoid misleading results due to possible overfitting. The forward-backward procedure, described in Section 2.4, took about 54 seconds to compute posterior probability distributions of all the variables in the original model and for all 2000×80 time steps in the data. The inference procedure was run with three models: the original model with the original parameters, the original model with the estimated parameters, and the alternative model with its estimated parameters. The original model, which actually produced the data, was used for this "clean room" experiment to see how the estimated models perform compared to the optimal one.

The average logarithmic likelihood scores were computed as a useful side-effect of the inference procedures. The original model with original parameters (i.e. optimal model) scored -0.831959 which is clearly higher than the score reached by the corresponding model during EM algorithm (-0.86128). This reveals that the model estimated above could possibly reach even better parameters if the threshold value was smaller, EM algorithm had continued, and numerical precision had allowed. The learning processes in Fig. 4.6 might have also encountered a local maximum which is slightly less than the optimal.

The original model with the estimated parameters scored average logarithmic likelihood of -0.854189 for the testing data set. This suggests that the model was not overfitted, because the score is approximately the same as the final score for training set (-0.86128). It seems that the numbers have about one or two significant digits, due to the stochastic nature of models and data sets, since the score for testing set is higher than for training set.

The estimated alternative model achieved log-likelihood score -0.890992 for the testing data and it is also very close to the level reached during EM algorithm (-0.89686). In case of overfitting, the likelihood of test set would usually turn out to be less than the likelihood of the data used for parameter estimation.

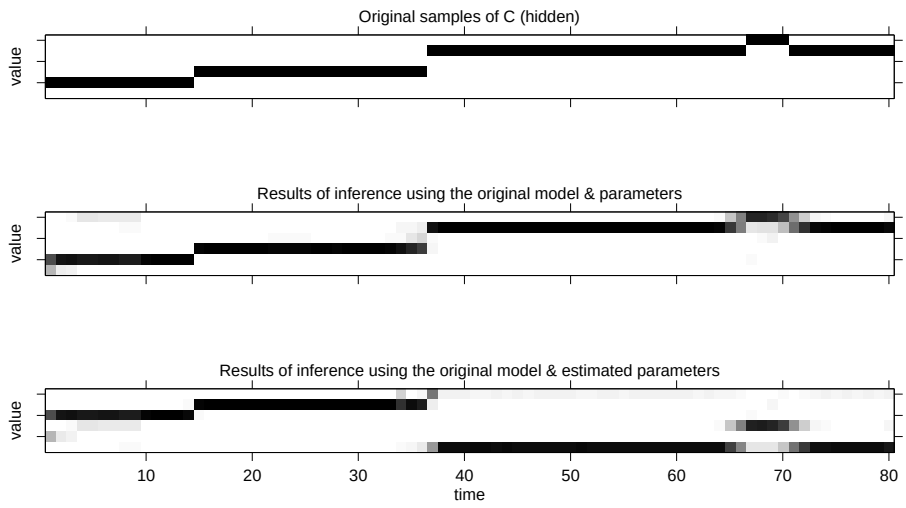

Figure 4.7: Sample of inference results for C^t by the original model

Figure 4.7 shows samples of hidden variable C from one of the time series in the testing set. The figure also displays the inference results by the original model with the original parameters (i.e. optimal inference results) and inference results by the original model with estimated parameters.

The most prominent feature is that the states of the hidden variable have switched due to the lack of identifiability discussed in Section 3.2.3. The figure suggests that the labels have switched as follows: $c_1 \rightarrow c_2$, $c_2 \rightarrow c_4$, $c_3 \rightarrow c_5$, $c_4 \rightarrow c_6$, $c_5 \rightarrow c_1$, and $c_6 \rightarrow c_3$. Otherwise the EM algorithm seems to have found almost optimal parameters since there is only small amount of additional uncertainty compared to the optimal results.

Figures 4.8 and 4.9 display similar set of inference results for variables B and D respectively (during the same time series as in Fig.4.7). This time, there is a reference image showing the partial data that survived the random discard process and was seen by the inference procedure.

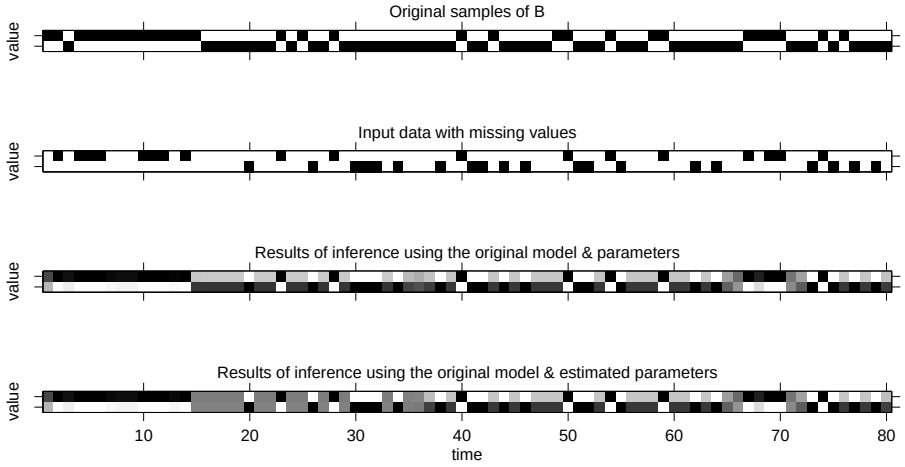

Figure 4.8: Sample of inference results for B^t by the original model

The results for original and estimated model in Fig. 4.8 look very similar, but the estimated model is very uncertain about the missing values of B during the first half of the time series. It seems that the parameters $P(B^t|C^t)$, for some values of C^t, are the ones that could have been learned better during the EM algorithm. After all, the state of C seemed to be accurately inferred at the particular point of time.

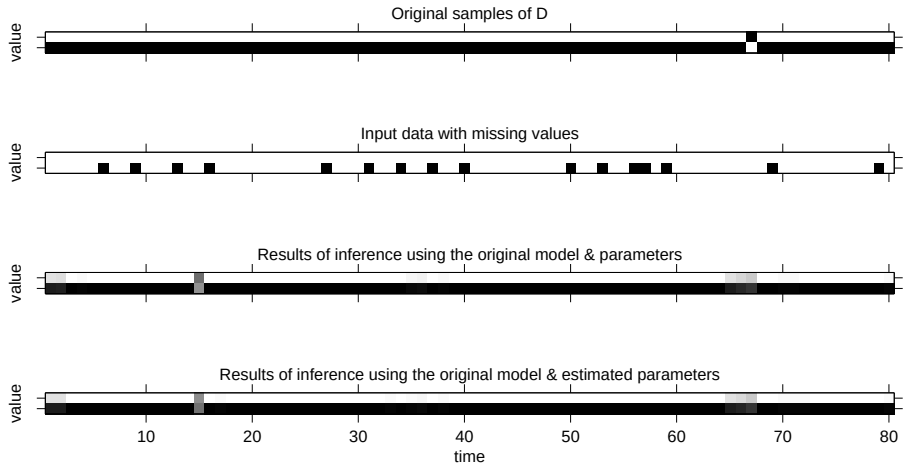

Figure 4.9: Sample of inference results for D^t by the original model

Figure 4.9 shows at least two things about inference results with variable D. First of all, the highly asymmetric prior $P(D^t)$ allows the model to "guess" that the value is most likely zero and therefore make correct conclusions about 97% of the time. On the other hand, even the optimal model is able to "blame" the variable D for state changes of variable C whenever D is unobserved. E.g. at time $t = 15$, both models come to conclusion, that the event $D^{15} = 1$ *might* be the reason behind the inferred fact that $C^{14} \neq C^{15}$, although it isn't. At time $t = 67$, the models have only slight uncertainty towards the fact that $D^{67} = 1$ and has caused a state change: possibly because there's also uncertainty about the exact time of the state change.

A sample of the inference results with the estimated alternative model is shown in Figures 4.10 and 4.11. The first one shows conclusions drawn about the state of variable C during the same time series as with the other model above. At least two features of the model can be seen in the figure. First, data from single hidden state (c_3) of the original model is co-explained by two hidden states (c_2 and c_3) in the alternative model during $t \in [15, 36]$. This phenomenon can be explained by the learning process being stopped too early[2]. Probably, the two hidden states weren't specialized enough to explain two different distributions of $A \times B$ when the threshold value was

[2]and deliberately left as an example by not tightening the threshold value

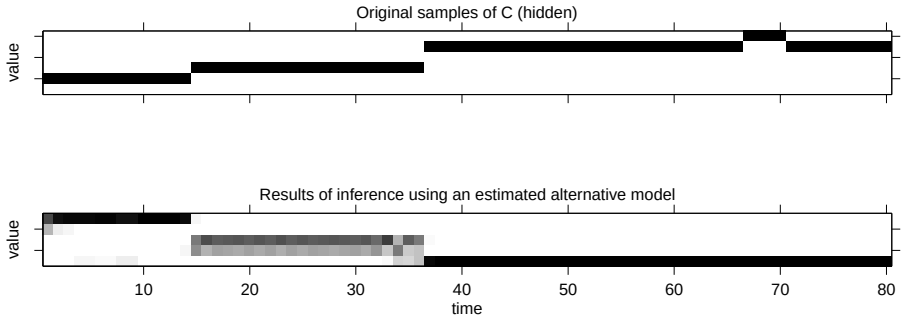

Figure 4.10: Sample of inference results for C^t by the alternative model

reached in EM algorithm.

Secondly, two hidden states of the original model (c_5 and c_6) have merged into one state (c_1) as can be seen during the interval $t \in [37, 80]$. This feature could be explained by both interrupted learning process and the fact that the alternative model has less hidden states than the original model. While two hidden states were wasted on explaining one kind of data, one state had to explain the data from two different distributions.

Due to the problems in modeling the hidden Markov chain, the estimated alternative model is not particularly good at inferring the missing values of variable B either. Figure 4.11 shows that the inference results have high uncertainty about the missing values.

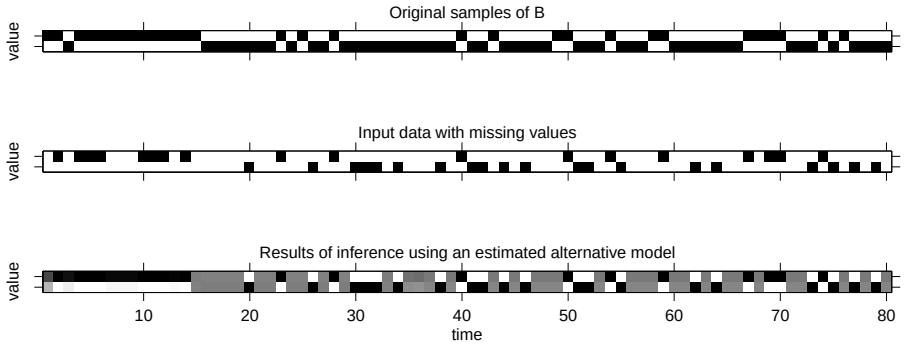

Figure 4.11: Sample of inference results for B^t by the alternative model

Some error statistics need to be defined, to describe the results of inference quantitatively. One way to measure the errors made in inference is to

compute average absolute error in posterior probability (compared to hard evidence) over all values of a given variable and over each time step when the variable is unobserved:

$$E_{abs}(V) = \frac{1}{\sum_t h(V^t)} \sum_t \left(h(V^t) \sum_{i=1}^{c_V} |\delta(V^t = v_i) - p(V^t = v_i|\mathbf{y}_1^T)| \right), \quad (4.1)$$

where $h(V^t) = 1$ when V^t is hidden (and zero otherwise), similarly $\delta(V^t = v_i) = 1$ when the true value of V^t is v_i, and $p(V^t = v_i|\mathbf{y}_1^T)$ is the posterior probability computed by the inference engine. Minimum E_{abs} is zero, which would mean that the inference engine deduced the correct value always without any uncertainty (i.e. model and data were deterministic). Maximum E_{abs} is two, which would mean that the inference engine was always certain that $V^t \neq v^t$.

Another way to measure the errors is to compute the root of the mean square error in the inferred posterior probabilities. More formally:

$$E_{RMS}(V) = \sqrt{\frac{1}{\sum_t h(V^t)} \sum_t \left(h(V^t) \sum_{i=1}^{c_V} (\delta(V^t = v_i) - p(V^t = v_i|\mathbf{y}_1^T))^2 \right)}, \quad (4.2)$$

using the same notation as in Eq. 4.1. The meaning of the error score is similar to E_{abs}, but this one de-emphasizes small errors.

A more practical way of comparing errors is to count how many times the Maximum A Posteriori (MAP) estimate was wrong (when the variable was not observed). That is, if the inference engine assigned probability one for the most probable value and zero for others, how often would the result be wrong.

All the three error statistics were computed for all the three models. Table 4.1 shows the error scores when considering the inference of variable B. Note that variable B is binary and therefore a completely random guess would achieve about 50% error rate.

Table 4.2 shows the similar error scores for inference of variable D, which was not included in the alternative model. Note that the prior $p(D = 0) = 0.97$ and $p(D = 1) = 0.03$. Thus, a model that guesses always $D = 0$ will

achieve about 3% error rate. Then again, such a model should somehow be learned in the first place.

Model	E_{abs}	E_{RMS}	**MAP error %**
Optimal	0.5350	0.5163	17.7
Original	0.6273	0.5586	25.7
Alternative	0.7335	0.6040	30.8

Table 4.1: Error statistics for inference of B^t

Model	E_{abs}	E_{RMS}	**MAP error %**
Optimal	0.09235	0.2151	2.94
Original	0.09449	0.2191	2.99

Table 4.2: Error statistics for inference of D^t

4.3 DNA copy number amplification data

4.3.1 Data

Use of the methods and software was briefly studied with real world data. The data set is the same as was used for amplification profiling of human neoplasias in [30]. Shortly, duplicated strings of DNA cause higher expression level for some genes and is supposedly one factor inducing tumors and even cancer. The expression level is measured using a method called Comparative Genomic Hybridization (CGH) and the data is consequently analyzed for purposes of medical therapy, diagnostics, and prognostics [30].

The original data was collected from 838 published CGH research articles and preprocessed by the authors of [30]. The result is a binary data matrix of DNA copy number amplification flags at chromosome sub-band resolution (393 bands/genome) for 4590 cases. Additionally, the type of tumor was known for each case. Figure 4.12 demonstrates 100 tumor cases from the data matrix with chromosome regions on the horizontal axis: black denotes amplification and chromosome labels point the corresponding centromeres.

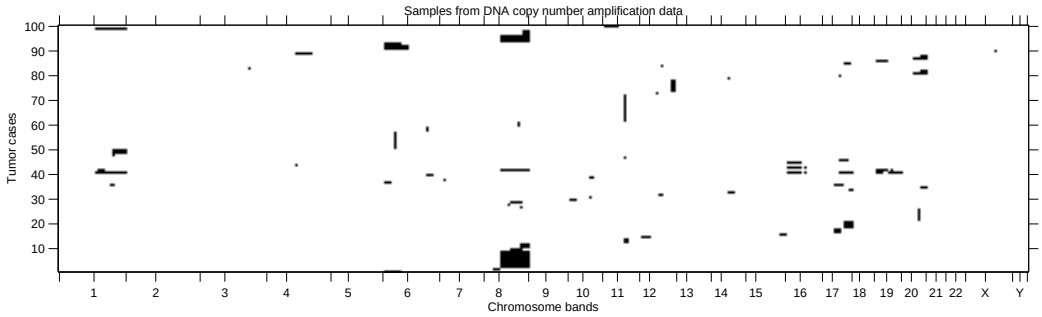

Figure 4.12: Some tumor cases with DNA copy number amplifications

As demonstrated in the figure, the copy number amplifications occur very sparsely. It was also noted that none of the 73 different types of neoplasms had even a single consistent band of amplification. An element-wise AND-operation, over all cases of one type of tumor, results as a zero vector. This shows that tasks like cancer classification and amplification profiling require more sophisticated methods.

To limit the scope of this study, only the chromosome 8 was selected for further investigations and the 73 different tumor labels were replaced by 13 category labels according to the tumor location in a human body. Chromosome 8 contains 18 bands. About 75% of the data (3442 cases) were randomly selected for parameter estimation purposes and the rest (1148 cases) were used for testing.

4.3.2 Model

Restricting the scope to a single chromosome is justified by the assumption that there is no significant dependency between amplifications across a chromosome boundary. The models considered in this work operate with very limited context and assume dependency between two adjacent time steps. Chromosomes, on the other hand, are physically separate entities and therefore it was considered unreasonable to model dependencies between bands $1q44$ and $2p25$ etc.

Creating an intuitive DBN model for a chromosome was still regarded as a difficult task, since the amplification measurements from a chromosome

may not be a time series: merely a sequence. There was no prior knowledge that any kind of Markov assumption holds nor that there would be some causal relationship between adjacent chromosome bands. Additionally, the type or category of tumor is a global property of a cell, i.e., it is not something that varies across the DNA.

While specifying the model, one rule of thumb was to keep the number of parameters small compared to the amount of data used for parameter estimation in order to avoid overfitting. Using 13 tumor categories, instead of 73 more specific tumor types, helped in reducing the number of model parameters, as discussed below.

The chosen model is shown in Figure 4.13. The model has the three variables observed in the data: binary variable D for DNA copy number amplification, 13-state variable C for tumor category, and 18-state variable B for chromosome band. There is also a hidden persistence variable, E, which provides contextual dependency across chromosome bands.

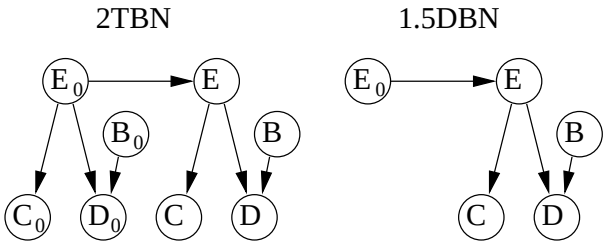

Figure 4.13: Model for amplification analysis

The model has several unusual properties. First of all, it could be interpreted so that a tumor is not directly caused by amplifications, but both tumors and amplifications are effects of the latent variable E (as in *environment*). Note that there actually exists dependency between variables C and D due to the fact that E is hidden: measured data and diagnosed tumor type would be independent of each other only if the assumed "environment factor" could be accurately observed somehow.

Secondly, the behavior in different chromosome bands is modeled explicitly by including variable B as another parent of D. Most time series models do not include time as one of the random variables, but in this case, different parts of the genome are assumed to have unique effects on the organism.

The model is able to assign band-specific amplification probabilities for each of the hidden states.

The third feature is, that the tumor category C is not actually an observation sequence but a strip of copies of the case specific value. This may seem like a misuse of a probabilistic model, but the objective was to take all the given data into account and enforce a tumor specific clustering by utilizing the latent variable E.

If the model had a (degenerate) Markov chain of C (i.e. temporal edge $C^{t-1} \rightarrow C^t$), it could learn that the tumor type is a global value and doesn't change during the sequence. But the size of potential $P(C^t|E^t, C^{t-1})$ would be $c_C^2 \times c_E$ and most of these parameters would be zero ($C^t \neq C^{t-1}$): a better model for this would be a static BN where C is a common child of the 18 instances of E. On the other hand, the static BN model would have a clique of 19 variables (C and E^t, $t \in [1, 18]$) and require memory for $c_C \times c_E^{18}$ elements (e.g. 13^{19} is unrealizable). Using a sequence C^t of copied tumor category values is an experimental compromise made to force the data into a form compatible with a DBN model.

The last chosen feature was the cardinality of E. In principle, c_E could be anything in the range $[2, 73]$, because there are 73 different types of tumor and having more explaining hidden states, than the maximum number of expected "clusters", would result in overfitting. Considering that $c_B = 18$, $c_C = 13$, and $c_D = 2$, the number of parameters in the model is

$$\begin{aligned} n &= c_E^2 + c_E c_C + c_E c_D c_B + c_E + c_B \\ &= c_E^2 + 50 c_E + 18. \end{aligned} \tag{4.3}$$

Finally, $c_E = 13$ was chosen and it corresponds to $n = 837$. That was thought reasonable compared to the amount of 18×3442 amplification samples used for parameter estimation. Now, the model has one hidden state for each tumor category, which in turn makes the method more like tumor clustering than fully-fledged time series analysis.

4.3.3 Parameter estimation

The EM algorithm was run with the 3442 tumor cases selected for parameter estimation while the threshold parameter was set to 1/100000. The last attempt took 2 hours of computing (41 iterations) and the learning curve is shown in Fig. 4.14. The curve reaches average log-likelihood score of -0.49582 and the final parameters are found in Appendix A.5.

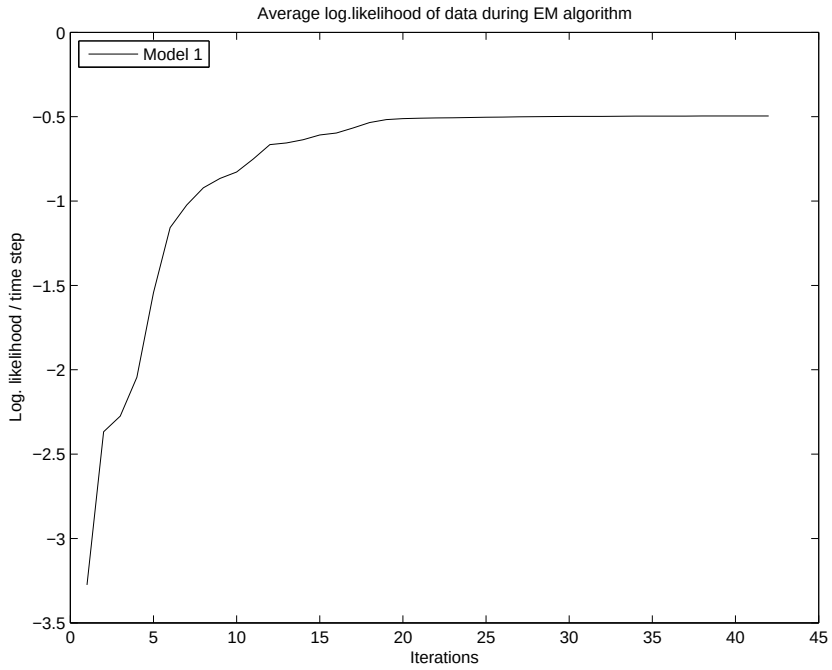

Figure 4.14: Learning curve for the amplification model

4.3.4 Results

Once estimated from the amplification data, the probabilistic model might provide various opportunities for data analysis. If the inference engine is given a sequence of observed DNA copy number amplifications, it may work as a tumor classifier by computing posterior probabilities for tumor categories.

To assess the estimated model, 1148 tumor cases (not included in the training set) were used for testing. First, the inference procedure was run

with the complete data including both tumor categories and amplifications. The average log-likelihood score for the test set turned out to be -0.504938, which is near the score achieved during the EM algorithm. This suggests that the model did not experience much overfitting. At least none of the test samples were found "impossible" by the model.

The second step was to run the inference procedure on the amplification data without the tumor category information to see the performance as a tumor classifier. In a real world situation, one might have measured new set of amplification samples D^t and use the model estimated from prior data to infer probabilities for tumor category of the new case. As an example, figure 4.15 shows the sequence of amplification data from a digestive tract tumor case in the testing data.

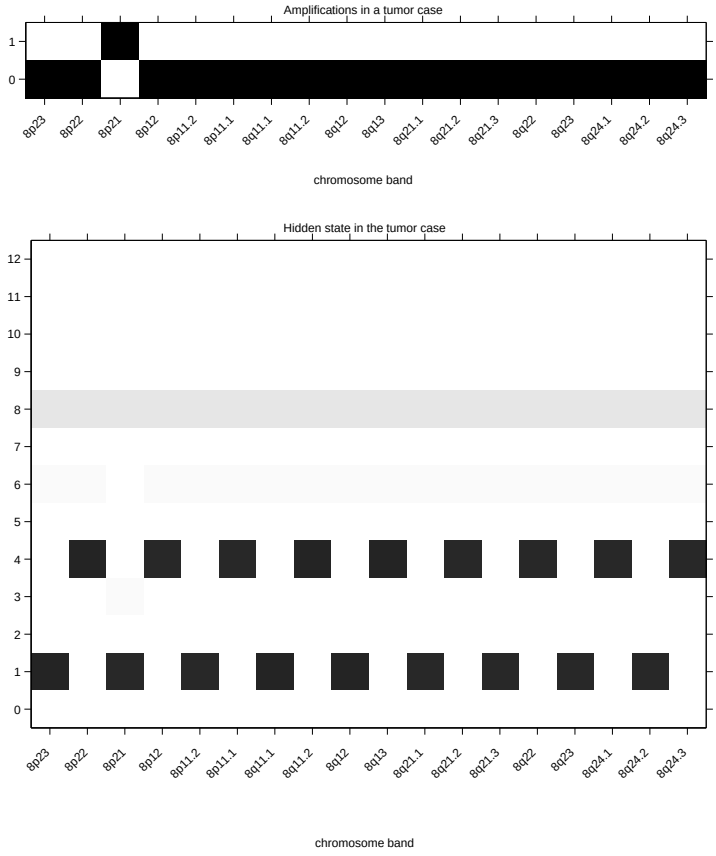

Figure 4.15: D^t and inferred states of E^t in one tumor case

Figure 4.15 displays also the posterior probabilities of the latent variable E during the sequence. It reveals how the estimated model is able to explain both band-specific amplifications and constant tumor category labels. The hidden state is likely to jump between two states (1 and 4 with 84% certainty), while there is a slight (10%) suspicion of staying in the state 8: note that $p(E^t = 8|E^{t-1} = 8) = 1$. The constant "component" probably models the tumor category observations independently from D because of the indifference to the amplification at $8p21$. Similarly, states $D = 0$ and $D = 1$ map[3] to $E = 6$ and $E = 3$ correspondingly with a 2% constant probability component.

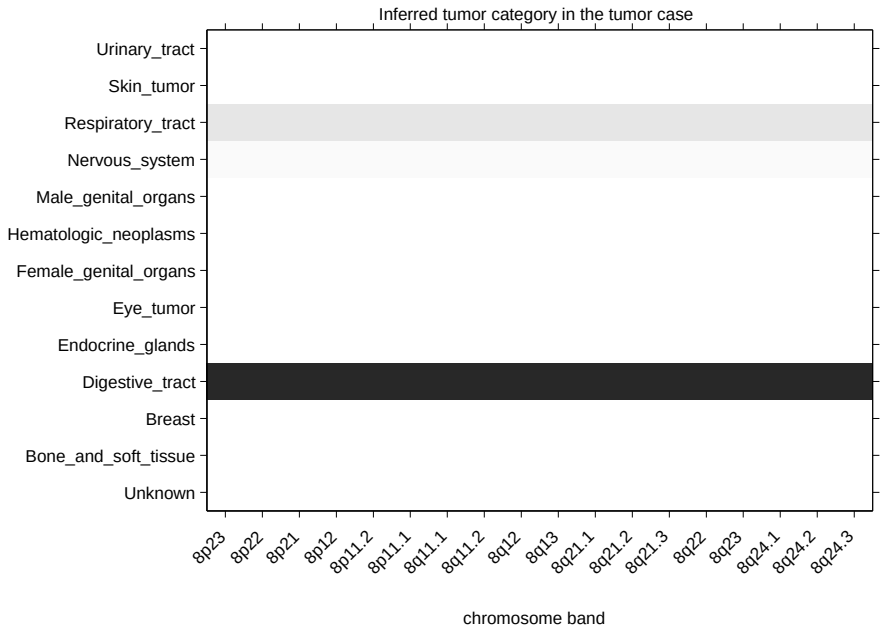

Figure 4.16: Inferred tumor category C^t in the tumor case

Figure 4.16 shows the inferred posterior probabilities of tumor categories for the same tumor case. In this fortunate case, the maximum of about 84% certainty falls to the correct category of digestive tract tumors. Also respiratory tract tumor is suspected with 10% probability, but the value is band-independent, and therefore, possibly due to beliefs about $E = 8$. A nervous system tumor is suspected with less than 2% probability.

[3] $p(D = 0|B = 8p21; E = 6) = 1$ and $p(D = 1|B = 8p21; E = 3) = 1$

Despite the success in classifying one tumor case, the classification accuracy in general is not that impressive. Table 4.3 summarizes the same kind of performace statistics as used earlier with artificial data. The maximum probability is assigned to the correct tumor category in only about 19% of the test cases. A random classifier would guess correctly in $1/13 \approx 7.7\%$ of the cases on average. The weak performance is partly explained by the fact that some tumors don't have copy number amplifications in the chromosome under study, i.e., the data is not even supposed to contain all the relevant information.

E_{abs}	E_{RMS}	**MAP error %**
1.730	0.9385	80.7

Table 4.3: Error statistics for inference of C^t

The other aspect for data analysis is DNA copy number amplification profiling. This can be achieved by feeding the inference engine with one tumor category value along with the deterministic sequence of band labels. The result is a sequence of posterior probabilities for amplifications at different chromosome bands as shown in Fig. 4.17.

In effect, the figure is a compact summary of the data in the training set: what kind of amplifications are expected in each of the tumor types given the model. The general light shade of the figure tells that there were no definite amplification sites that would be both common to a tumor category and separate it from other tumor categories.

For comparison, Figure 4.18 shows normalized amplification averages. Average amplification vector was computed for each known tumor category over the training set, but the image had to be heavily normalized to make it visible since the maximum value (black) is 0.0171. The latter image is also a summary and resembles the figure achieved with the probabilistic model, but shows only the most obvious facts about the data. Probabilistic modeling provides systematic means for specifying additional assumptions about the process behind the data, estimating quantitative parameters, and inferring posterior beliefs about new data.

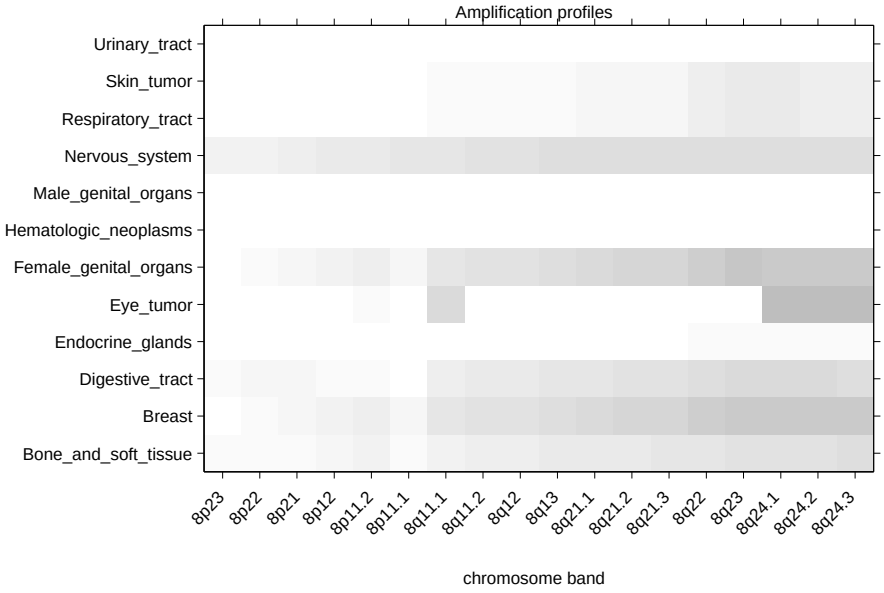

Figure 4.17: Amplification profiles $P(D^t|c, b^t)$

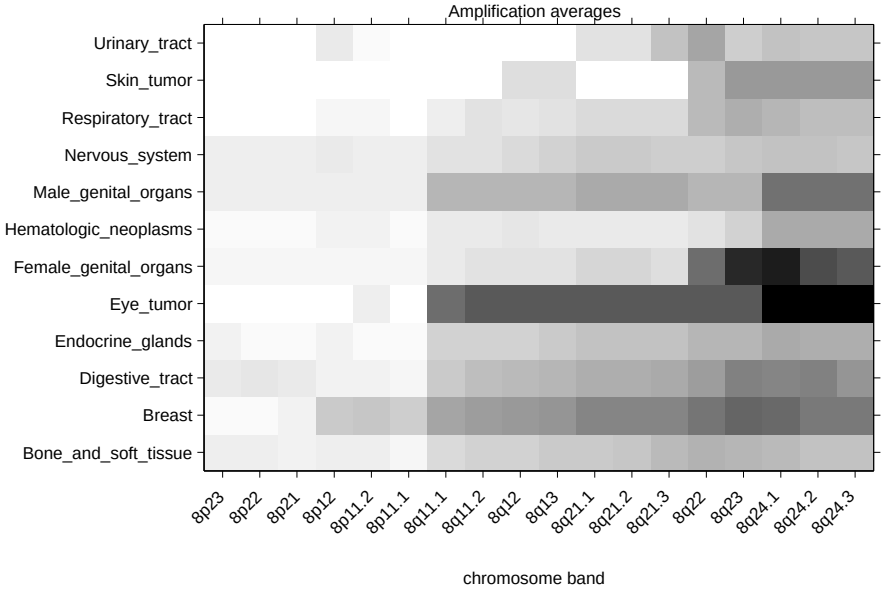

Figure 4.18: Normalized amplification averages

Chapter 5

Summary and conclusions

5.1 Methods and experiments

This thesis focused on modeling multidimensional time series of discrete variables by using exact Bayesian inference. In Chapter 2, it was explained how BNs model discrete valued phenomena, starting on how clique tree propagation is traditionally used for inference in static BNs. Beliefs about the state of the variables are encoded as probability tables and quantitative dependencies between variables are represented as conditional probability tables. Subsequently, a clique tree structure is formed and used as an inference engine.

Section 2.4 described the idea of using a stream of small, identical pieces of networks as a time series model: a dynamic Bayesian network. It was also elaborated how to extend the CTP inference for DBNs by utilizing the interface algorithm. Based on that, Section 2.5 presented a way to specify DBNs in a manner compatible to existing tools. The small addition to the BN description language was taken into use in a software library implemented for this work. Additionally, Section 2.6 dealt with the straightforward task of using the new inference framework for prior sampling of DBNs.

Chapter 3 contained a method for computing arbitrary joint probability distributions given a clique tree (Sec. 3.1.1). However, there exists a better way to compute the likelihood of data, which was later needed to monitor the learning algorithm. In addition to computing logarithmic likelihood

of time series as a side-effect of forward inference, Section 3.1.3 presented message normalization as a way to avoid excessive numerical problems. Finally, the EM algorithm for parameter estimation was constructed on top of forward-backward inference procedure as shown in Section 3.2.

Chapter 4 showed experiments made with the software library and the discussed methods. First, Section 4.2 demonstrated time series modeling with DBNs in an idealized laboratory setting. Artificial data was sampled according to a known model. To mimic a real world situation, part of the data was randomly discarded. The resulting partial observations were used for learning and probabilistic inference.

The experiments with artificial data demonstrated various effects encountered in time series analysis using DBNs. Choosing a suitable threshold value for EM learning was discovered to be a task that requires insight and experimentation. Despite the lack of identifiability related to latent variables, the EM algorithm proved to be a useful method when the structure of the model is known. Experiments with the alternative model structure were less encouraging, but demonstrated the effects of specifying too few hidden states to explain the data.

Brief experiments with DNA copy number amplification data were presented in Section 4.3. The scope of the experiments was limited to the 8th chromosome and broad tumor category labels. Considering the lack of better knowledge about the domain, a simple probabilistic model was specified and its parameters were estimated with the EM algorithm.

Subsequently, the model was used as a tumor classifier and as an amplification profiler. Only about 19% of the tumors were correctly classified, but the results were clearly better than pure guessing. It was also noted, that the data from a single chromosome does not contain enough information for accurate classification. The amplification profiling, on the other hand, proved to be an interesting method to be further investigated.

As a conclusion, it seems that DBNs are not suitable for exploratory data mining *without* sufficient domain knowledge or an automated structural learning method. There has to be a known reason for each missing edge in the graph. Model specification requires solid arguments to achieve clear results: without the correct assumptions, accurate models can't be estimated.

5.2 Future work

There are many potential directions for additional development. The first task might be to investigate more thoroughly the applicability of the methods in real data analysis projects. For example, the DNA copy number amplification profiling could be done for the whole genome to achieve more accurate results. Another types of models could also be used for searching new insights about the data – at least if more prior knowledge can be found to support the specification process. It would be interesting to see whether any Markov assumptions really hold for the data. Is a context sensitive model really needed?

The second area for development could be structural learning [4] methods. These kind of methods would be necessary for more automated exploration of the data.

Third aspect is to allow continuous variables in the models. This is not straightforward, since the representation of state distributions becomes unwieldy in a general case, as noted in [36]. Some limitations, like having continuous variables only as leaf nodes in the graph, might make the modeling feasible. At least Kalman filters have proven to be useful in many applications [36].

Bibliography

[1] J. A. Bilmes. A gentle tutorial of the EM algorithm and its application to parameter estimation for gaussian mixture and hidden Markov models. Technical Report TR-97-021, International Computer Science Institute (ICSI), 1998.

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

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

[4] G. F. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992.

[5] R. Cowell. Introduction to inference for bayesian networks. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 9–26. MIT Press, 1999.

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

[7] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[8] A. P. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.

[9] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, second edition, 2001.

[10] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis*. Chapman & Hall/CRC, second edition, 2004.

[11] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, MA, USA, 2001.

[12] J. Hollmén and V. Tresp. Call-based fraud detection in mobile communications networks using a hierarchical regime-switching model. In M. Kearns, S. Solla, and D. Cohn, editors, *Advances in Neural Information Processing Systems 11: Proceedings of the 1998 Conference (NIPS'11)*, pages 889–895. MIT Press, 1999.

[13] J. Hollmén and V. Tresp. Hidden Markov model for metric and event-based data. In *Proceedings of EUSIPCO 2000 — X European Signal Processing Conference*, volume II, pages 737–740, 2000.

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

[15] Hugin Expert A/S. The Net Language. Internet http://developer.hugin.com/documentation/net/, 2004. referenced October 24, 2006.

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

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

[18] B. Juang and L. Rabiner. Hidden Markov models for speech recognition. *Technometrics*, 33(3):251–272, 1991.

[19] B. W. Kernighan. *The C Programming Language*. Prentice Hall Professional Technical Reference, second edition, 1988.

[20] U. Kjærulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence (UAI-92)*, pages 121–129, 1992.

[21] U. Kjærulff. dHugin: A computational system for dynamic time-sliced Bayesian networks. *International Journal of Forecasting, Special Issue on Probability Forecasting*, 11:89–111, 1995.

[22] D. Koller and A. Pfeffer. Object-oriented bayesian networks. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 302–313, 1997.

[23] S. L. Lauritzen. EM algorithm for graphical association models with missing data. *Computational Statistics & Data Analysis*, 19:191–201, 1995.

[24] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society series B*, 50(2):157–224, 1988.

[25] S. Levinson, L. Rabiner, and M. Sondhi. An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. *The Bell System Technical Journal*, 62(4):1035–1074, 1983.

[26] I. L. MacDonald and W. Zucchini. *Hidden Markov and other models for discrete-valued time-series*. Number 70 in Monographs on Statistics and Applied Probability. Chapman & Hall, 1997.

[27] Microsoft Research DTAS group. XML Belief Network File Format. Internet http://research.microsoft.com/dtas/bnformat/, 1999. referenced May 29th, 2006.

[28] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California at Berkeley, July 2002.

[29] K. Murphy. *Probabilistic Graphical Models, Michael Jordan*, chapter Dynamic Bayesian Networks (draft). To appear. http://www.cs.ubc.ca/~murphyk/Papers/dbnchapter.pdf.

[30] S. Myllykangas, J. Himberg, T. Böhling, B. Nagy, J. Hollmén, and S. Knuutila. DNA copy number amplification profiling of human neoplasms. *Oncogene*, 25(55):7324–7332, 2006.

[31] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[32] A. B. Poritz. Hidden Markov models: A guided tour. In *Proceedings of the IEEE International conference of Acoustics, Speech and Signal Processing (ICASSP-88)*, pages 7–13, 1988.

[33] L. Rabiner and B. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, 3(1):4–16, 1986.

[34] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[35] R. Redner and H. Walker. Mixture densities, maximum likelihood and the EM algorithm. *SIAM Review*, 26(2):195–234, 1984.

[36] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, Inc., (international) second edition, 2003.

[37] P. Smyth, D. Heckerman, and M. I. Jordan. Probabilistic independence networks for hidden Markov probability models. *Neural Computation*, 9(2):227–269, 1997.

[38] W. X. Wen. Optimal decomposition of belief networks. In *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence (UAI-90)*, pages 209–224, New York, NY, USA, 1991. Elsevier Science Inc.

[39] C. F. J. Wu. On the convergence properties of the EM algorithm. *The Annals of Statistics*, 11(1):95–103, 1983.

[40] G. Zweig. A forward-backward algorithm for inference in Bayesian networks and an empirical comparison with HMMs. Master's thesis, University of California at Berkeley, 1996.

Appendix A

Model specifications

Models were specified with the Hugin Net language. The documentation about syntax was found at [15].

A.1 Example model

This one was used for demonstrating Net language.

```
net
{
    node_size = (100 40);
}

node F
{
    label = "Favorite fruit";
    position = (225 75);
    states = ("apple" "banana" "orange" "pear" "lemon");
    MY_field = "42";
}

node A
{
    label = "Age";
    position = (225 175);
    states = ("child" "teen" "adult");
}

node S
{
    label = "Sex";
    position = (300 125);
    states = ("male" "female");
}
```

```
potential (F | S A)
{
  data = ((( 0.2 0.3 0.1 0.3 0.1 )     %  S=male  A=child
           ( 0.3 0.3 0.1 0.2 0.1 )     %  S=male  A=teen
           ( 0.3 0.2 0.2 0.2 0.1 ))    %  S=male  A=adult
          (( 0.1 0.4 0.1 0.3 0.1 )     %  S=female  A=child
           ( 0.2 0.3 0.2 0.2 0.1 )     %  S=female  A=teen
           ( 0.2 0.2 0.2 0.2 0.2 )));  %  S=female  A=adult
}

potential (A)
{
    data = ( 0.2 0.2 0.6 );
}

potential (S)
{
    data = ( 0.5 0.5 );
}
```

A.2 Model for synthetic data

This one was used in the experiments for generating synthetic data.

```
net {
    node_size = (80 40);
}

node A1 {
    label = "A(t)";
    position = (225 50);
    states = ("0" "1" "2");
}

node B1 {
    label = "B(t)";
    position = (375 50);
    states = ("0" "1");
}

node C0 {
    label = "C(t-1)";
    position = (50 175);
    states = ("0" "1" "2" "3" "4" "5");
    NIP_next = "C1";
}

node C1 {
    label = "C(t)";
    position = (300 175);
    states = ("0" "1" "2" "3" "4" "5");
}

node D1 {
    label = "D(t)";
    position = (300 300);
    states = ("0" "1");
}

potential (A1 | C1) {
    %         A=0   A=1   A=2
    data = (( 0.95  0.04  0.01 )   %  C=0
            ( 0.90  0.03  0.07 )   %  C=1
            ( 0.01  0.95  0.04 )   %  C=2
            ( 0.10  0.85  0.05 )   %  C=3
            ( 0.01  0.04  0.95 )   %  C=4
            ( 0.15  0.15  0.70 )); %  C=5
}   % 18 parameters

potential (B1 | C1) {
    %         B=0   B=1
    data = (( 0.90  0.10 )   %  C=0
            ( 0.03  0.97 )   %  C=1
            ( 0.80  0.20 )   %  C=2
            ( 0.15  0.85 )   %  C=3
            ( 0.78  0.22 )   %  C=4
            ( 0.05  0.95 )); %  C=5
}   % 12 parameters

potential (C0) {
    %        C0=0  C0=1  C0=2  C0=3  C0=4  C0=5
    data = ( 0.8   0.1   0.05  0.02  0.02  0.01 );
```

```
}   % 6 parameters

potential (D1) {
    %        D1=0  D1=1
    data = ( 0.97  0.03 );
}   % 2 parameters

potential (C1 | D1 C0) {
    %          C1=0  C1=1  C1=2  C1=3  C1=4  C1=5
    data = ((( 0.94  0.02  0.01  0.01  0.01  0.01 )    %  D1=0 C0=0
             ( 0.01  0.94  0.02  0.01  0.01  0.01 )    %  D1=0 C0=1
             ( 0.01  0.01  0.94  0.02  0.01  0.01 )    %  D1=0 C0=2
             ( 0.01  0.01  0.01  0.94  0.02  0.01 )    %  D1=0 C0=3
             ( 0.01  0.01  0.01  0.01  0.94  0.02 )    %  D1=0 C0=4
             ( 0.02  0.01  0.01  0.01  0.01  0.94 ))   %  D1=0 C0=5
            (( 0.05  0.91  0.01  0.01  0.01  0.01 )    %  D1=1 C0=0
             ( 0.01  0.05  0.91  0.01  0.01  0.01 )    %  D1=1 C0=1
             ( 0.01  0.01  0.05  0.91  0.01  0.01 )    %  D1=1 C0=2
             ( 0.01  0.01  0.01  0.05  0.91  0.01 )    %  D1=1 C0=3
             ( 0.01  0.01  0.01  0.01  0.05  0.91 )    %  D1=1 C0=4
             ( 0.91  0.01  0.01  0.01  0.01  0.05 ))); %  D1=1 C0=5
}   % 60 parameters
```

A.3 First model learned from synthetic data

These parameters were learned from the synthetic data when the original graph structure was known.

```
net
{
    node_size = (80 40);
}

node A1
{
    label = "A(t)";
    position = (225 50);
    states = ( "0"
               "1"
               "2" );
}

node B1
{
    label = "B(t)";
    position = (375 50);
    states = ( "0"
               "1" );
}

node C0
{
    label = "C(t-1)";
    position = (50 175);
    states = ( "0"
               "1"
               "2"
               "3"
               "4"
               "5" );
    NIP_next = "C1";
}

node C1
{
    label = "C(t)";
    position = (300 175);
    states = ( "0"
               "1"
               "2"
               "3"
               "4"
               "5" );
}

node D1
{
    label = "D(t)";
    position = (300 300);
    states = ( "0"
               "1" );
}

potential (C0)
{
```

```
    data = ( 0.003316  0.797307  0.000000  0.102955  0.071876  0.024545 );
}

potential (D1)
{
    data = ( 0.970784  0.029216 );
}

potential (A1 | C1)
{
    data = ( 0.010167  0.037458  0.952375
             0.950211  0.040121  0.009667
             0.152366  0.152904  0.694730
             0.898616  0.030703  0.070681
             0.051024  0.909064  0.039912
             0.013436  0.049169  0.937395 );
}

potential (B1 | C1)
{
    data = ( 0.778390  0.221610
             0.894569  0.105431
             0.044505  0.955495
             0.030125  0.969875
             0.493309  0.506691
             0.760150  0.239850 );
}

potential (C1 | D1 C0)
{
    data = ( 0.886708  0.012282  0.023045  0.012060  0.002094  0.063811
             0.009702  0.940389  0.010011  0.021389  0.018491  0.000018
             0.009464  0.018803  0.939347  0.010482  0.021889  0.000015
             0.007867  0.008980  0.010530  0.939953  0.031901  0.000768
             0.005507  0.008026  0.012155  0.010997  0.948680  0.014634
             0.672799  0.000319  0.001820  0.000001  0.316485  0.008576
             0.027282  0.002698  0.920666  0.014489  0.000013  0.034852
             0.014311  0.070950  0.006280  0.878971  0.029476  0.000011
             0.026457  0.941200  0.014947  0.003406  0.013989  0.000000
             0.025622  0.000001  0.000000  0.025029  0.949140  0.000207
             0.414970  0.028208  0.017170  0.018318  0.492325  0.029009
             0.000203  0.001847  0.821060  0.014465  0.157692  0.004733 );
}
```

A.4 Second model learned from synthetic data

These parameters were estimated for the alternative expert model with a different graph structure.

```
net
{
    node_size = (80 40);
}

node A1
{
    label = "A(t)";
    position = (300 50);
    states = ( "0"
               "1"
               "2" );
}

node B1
{
    label = "B(t)";
    position = (425 175);
    states = ( "0"
               "1" );
}

node C0
{
    label = "C(t-1)";
    position = (50 300);
    states = ( "0"
               "1"
               "2"
               "3"
               "4" );
    NIP_next = "C1";
}

node C1
{
    label = "C(t)";
    position = (300 300);
    states = ( "0"
               "1"
               "2"
               "3"
               "4" );
}

potential (C0)
{
    data = ( 0.027956  0.066295  0.011191  0.800037  0.094521 );
}

potential (A1 | B1 C1)
{
    data = ( 0.019504  0.042015  0.938481
             0.034264  0.895994  0.069741
             0.020318  0.955017  0.024665
             0.950555  0.040239  0.009206
             0.906388  0.052674  0.040938
```

```
            0.109386  0.129077  0.761537
            0.149455  0.809385  0.041160
            0.040427  0.910194  0.049378
            0.947281  0.041345  0.011374
            0.890184  0.031312  0.078504 );
}

potential (B1 | C1)
{
    data = ( 0.419387  0.580613
            0.509740  0.490260
            0.475351  0.524649
            0.892583  0.107417
            0.027865  0.972135 );
}

potential (C1 | C0)
{
    data = ( 0.935220  0.009634  0.012956  0.028105  0.014085
            0.032245  0.075552  0.851081  0.012760  0.028362
            0.046526  0.488260  0.457297  0.007381  0.000536
            0.018634  0.014178  0.005899  0.914444  0.046845
            0.020139  0.029101  0.029608  0.010404  0.910749 );
}
```

A.5 The model learned from DNA data

These parameters were estimated from the DNA copy number amplification data.

```
net
{
    node_size = (60 120);
}

node C
{
    label = "Tumor category";
    position = (200 100);
    states = ( "0"
               "Bone_and_soft_tissue"
               "Breast"
               "Digestive_tract"
               "Endocrine_glands"
               "Eye_tumor"
               "Female_genital_organs"
               "Hematologic_neoplasms"
               "Male_genital_organs"
               "Nervous_system"
               "Respiratory_tract"
               "Skin_tumor"
               "Urinary_tract" );
}

node B
{
    label = "Chromosome band";
    position = (300 300);
    states = ( "8p23"   "8p22"   "8p21" "8p12"   "8p11.2" "8p11.1"
               "8q11.1" "8q11.2" "8q12" "8q13"   "8q21.1" "8q21.2"
               "8q21.3" "8q22"   "8q23" "8q24.1" "8q24.2" "8q24.3" );
}

node D
{
    label = "DNA copy number amplification";
    position = (250 200);
    states = ( "0" "1" );
}

node E
{
    label = "Environment(t)";
    position = (200 400);
    states = ( "0" "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12" );
}

node E0
{
    label = "Environment(t-1)";
    position = (100 400);
    states = ( "0" "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12" );
    NIP_next = "E";
}

potential (B)
{
```

```
    data = ( 0.055556  0.055556  0.055556  0.055556  0.055556  0.055556  0.055556
             0.055556  0.055556  0.055556  0.055556  0.055556  0.055556  0.055556
             0.055556  0.055556  0.055556  0.055556  );
}

potential (E0)
{
    data = ( 0.000510  0.002380  0.001571  0.001935  0.202885  0.108077  0.181099
             0.124986  0.173155  0.015537  0.157639  0.000000  0.030227  );
}

potential (C | E)
{
    data = ( 0.000000  0.000000  0.000000  0.000000  0.298062  0.000000  0.000000
             0.379413  0.322525  0.000000  0.000000  0.000000  0.000000  % E=0
             0.000000  0.000000  0.000000  0.994389  0.000000  0.005611  0.000000
             0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  % E=1
             0.000000  1.000000  0.000000  0.000000  0.000000  0.000000  0.000000
             0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  % E=2
             0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
             0.000000  0.000000  0.850186  0.000000  0.000000  0.149814  % E=3
             0.000000  0.000000  0.000000  0.994316  0.000000  0.005684  0.000000
             0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  % E=4
             0.000000  0.000000  0.552788  0.000000  0.000000  0.000000  0.447212
             0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  % E=5
             0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
             0.000000  0.000000  0.773475  0.000000  0.000000  0.226525  % E=6
             0.000000  0.000000  0.000000  0.000000  0.339780  0.000000  0.000000
             0.660220  0.000000  0.000000  0.000000  0.000000  0.000000  % E=7
             0.021812  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
             0.000000  0.000000  0.000000  0.929530  0.048658  0.000000  % E=8
             0.000000  0.000000  0.000000  0.000000  0.000000  1.000000  0.000000
             0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  % E=9
             0.000000  1.000000  0.000000  0.000000  0.000000  0.000000  0.000000
             0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  % E=10
             0.000000  0.000000  0.549411  0.000000  0.000000  0.000000  0.450589
             0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  % E=11
             0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
             0.000000  1.000000  0.000000  0.000000  0.000000  0.000000 );
}

potential (D | B E)
{
    data = ( 0.000000  1.000000  % B=8p23 E=0
             0.996856  0.003144  % B=8p23 E=1
             1.000000  0.000000  % B=8p23 E=2
             0.000000  1.000000  % B=8p23 E=3
             0.000000  1.000000  % B=8p23 E=4
             0.002377  0.997623  % B=8p23 E=5
             1.000000  0.000000  % B=8p23 E=6
             0.997741  0.002259  % B=8p23 E=7
             0.998322  0.001678  % B=8p23 E=8
             1.000000  0.000000  % B=8p23 E=9
             0.000000  1.000000  % B=8p23 E=10
             1.000000  0.000000  % B=8p23 E=11
             1.000000  0.000000  % B=8p23 E=12
             0.000000  1.000000  % B=8p22 E=0
             0.000000  1.000000  % B=8p22 E=1
             0.000000  1.000000  % B=8p22 E=2
             0.000000  1.000000  % B=8p22 E=3
             0.996626  0.003374  % B=8p22 E=4
             1.000000  0.000000  % B=8p22 E=5
             1.000000  0.000000  % B=8p22 E=6
             1.000000  0.000000  % B=8p22 E=7
```

```
0.998322  0.001678  % B=8p22 E=8
1.000000  0.000000  % B=8p22 E=9
1.000000  0.000000  % B=8p22 E=10
0.321058  0.678942  % B=8p22 E=11
1.000000  0.000000  % B=8p22 E=12
0.000000  1.000000  % B=8p21 E=0
0.992685  0.007315  % B=8p21 E=1
1.000000  0.000000  % B=8p21 E=2
0.000000  1.000000  % B=8p21 E=3
0.081832  0.918168  % B=8p21 E=4
0.195414  0.804586  % B=8p21 E=5
1.000000  0.000000  % B=8p21 E=6
1.000000  0.000000  % B=8p21 E=7
0.998322  0.001678  % B=8p21 E=8
1.000000  0.000000  % B=8p21 E=9
0.160115  0.839885  % B=8p21 E=10
1.000000  0.000000  % B=8p21 E=11
1.000000  0.000000  % B=8p21 E=12
0.000000  1.000000  % B=8p12 E=0
0.436724  0.563276  % B=8p12 E=1
0.000000  1.000000  % B=8p12 E=2
0.000000  1.000000  % B=8p12 E=3
0.997224  0.002776  % B=8p12 E=4
1.000000  0.000000  % B=8p12 E=5
0.995309  0.004691  % B=8p12 E=6
0.997673  0.002327  % B=8p12 E=7
0.991611  0.008389  % B=8p12 E=8
1.000000  0.000000  % B=8p12 E=9
1.000000  0.000000  % B=8p12 E=10
0.000000  1.000000  % B=8p12 E=11
1.000000  0.000000  % B=8p12 E=12
0.000000  1.000000  % B=8p11.2 E=0
0.998650  0.001350  % B=8p11.2 E=1
0.998227  0.001773  % B=8p11.2 E=2
0.000000  1.000000  % B=8p11.2 E=3
0.483180  0.516820  % B=8p11.2 E=4
0.000000  1.000000  % B=8p11.2 E=5
1.000000  0.000000  % B=8p11.2 E=6
1.000000  0.000000  % B=8p11.2 E=7
0.991611  0.008389  % B=8p11.2 E=8
0.981481  0.018519  % B=8p11.2 E=9
0.000587  0.999413  % B=8p11.2 E=10
0.997407  0.002593  % B=8p11.2 E=11
1.000000  0.000000  % B=8p11.2 E=12
0.514251  0.485749  % B=8p11.1 E=0
0.786500  0.213500  % B=8p11.1 E=1
0.570901  0.429099  % B=8p11.1 E=2
0.002583  0.997417  % B=8p11.1 E=3
1.000000  0.000000  % B=8p11.1 E=4
1.000000  0.000000  % B=8p11.1 E=5
1.000000  0.000000  % B=8p11.1 E=6
1.000000  0.000000  % B=8p11.1 E=7
0.996644  0.003356  % B=8p11.1 E=8
1.000000  0.000000  % B=8p11.1 E=9
1.000000  0.000000  % B=8p11.1 E=10
0.468100  0.531900  % B=8p11.1 E=11
1.000000  0.000000  % B=8p11.1 E=12
0.000000  1.000000  % B=8q11.1 E=0
1.000000  0.000000  % B=8q11.1 E=1
1.000000  0.000000  % B=8q11.1 E=2
0.000000  1.000000  % B=8q11.1 E=3
0.067745  0.932255  % B=8q11.1 E=4
0.000009  0.999991  % B=8q11.1 E=5
1.000000  0.000000  % B=8q11.1 E=6
```

```
1.000000  0.000000  % B=8q11.1 E=7
0.981544  0.018456  % B=8q11.1 E=8
0.851872  0.148128  % B=8q11.1 E=9
0.065241  0.934759  % B=8q11.1 E=10
1.000000  0.000000  % B=8q11.1 E=11
1.000000  0.000000  % B=8q11.1 E=12
0.000000  1.000000  % B=8q11.2 E=0
0.000000  1.000000  % B=8q11.2 E=1
0.000000  1.000000  % B=8q11.2 E=2
0.000000  1.000000  % B=8q11.2 E=3
1.000000  0.000000  % B=8q11.2 E=4
1.000000  0.000000  % B=8q11.2 E=5
1.000000  0.000000  % B=8q11.2 E=6
1.000000  0.000000  % B=8q11.2 E=7
0.973154  0.026846  % B=8q11.2 E=8
1.000000  0.000000  % B=8q11.2 E=9
0.998160  0.001840  % B=8q11.2 E=10
0.000000  1.000000  % B=8q11.2 E=11
1.000000  0.000000  % B=8q11.2 E=12
0.000000  1.000000  % B=8q12 E=0
1.000000  0.000000  % B=8q12 E=1
1.000000  0.000000  % B=8q12 E=2
0.000000  1.000000  % B=8q12 E=3
0.000000  1.000000  % B=8q12 E=4
0.000000  1.000000  % B=8q12 E=5
1.000000  0.000000  % B=8q12 E=6
0.997634  0.002366  % B=8q12 E=7
0.973154  0.026846  % B=8q12 E=8
1.000000  0.000000  % B=8q12 E=9
0.000000  1.000000  % B=8q12 E=10
1.000000  0.000000  % B=8q12 E=11
1.000000  0.000000  % B=8q12 E=12
0.000000  1.000000  % B=8q13 E=0
0.000000  1.000000  % B=8q13 E=1
0.000000  1.000000  % B=8q13 E=2
0.000000  1.000000  % B=8q13 E=3
1.000000  0.000000  % B=8q13 E=4
1.000000  0.000000  % B=8q13 E=5
0.998569  0.001431  % B=8q13 E=6
1.000000  0.000000  % B=8q13 E=7
0.971477  0.028523  % B=8q13 E=8
1.000000  0.000000  % B=8q13 E=9
0.998138  0.001862  % B=8q13 E=10
0.000053  0.999947  % B=8q13 E=11
1.000000  0.000000  % B=8q13 E=12
0.000000  1.000000  % B=8q21.1 E=0
0.998513  0.001487  % B=8q21.1 E=1
1.000000  0.000000  % B=8q21.1 E=2
0.000000  1.000000  % B=8q21.1 E=3
0.000000  1.000000  % B=8q21.1 E=4
0.000000  1.000000  % B=8q21.1 E=5
1.000000  0.000000  % B=8q21.1 E=6
1.000000  0.000000  % B=8q21.1 E=7
0.964765  0.035235  % B=8q21.1 E=8
1.000000  0.000000  % B=8q21.1 E=9
0.000000  1.000000  % B=8q21.1 E=10
1.000000  0.000000  % B=8q21.1 E=11
1.000000  0.000000  % B=8q21.1 E=12
0.000000  1.000000  % B=8q21.2 E=0
0.000000  1.000000  % B=8q21.2 E=1
0.036953  0.963047  % B=8q21.2 E=2
0.000000  1.000000  % B=8q21.2 E=3
1.000000  0.000000  % B=8q21.2 E=4
1.000000  0.000000  % B=8q21.2 E=5
```

```
1.000000  0.000000  % B=8q21.2 E=6
1.000000  0.000000  % B=8q21.2 E=7
0.964765  0.035235  % B=8q21.2 E=8
1.000000  0.000000  % B=8q21.2 E=9
1.000000  0.000000  % B=8q21.2 E=10
0.000000  1.000000  % B=8q21.2 E=11
1.000000  0.000000  % B=8q21.2 E=12
0.000000  1.000000  % B=8q21.3 E=0
1.000000  0.000000  % B=8q21.3 E=1
1.000000  0.000000  % B=8q21.3 E=2
0.000000  1.000000  % B=8q21.3 E=3
0.000000  1.000000  % B=8q21.3 E=4
0.000000  1.000000  % B=8q21.3 E=5
0.998134  0.001866  % B=8q21.3 E=6
1.000000  0.000000  % B=8q21.3 E=7
0.964765  0.035235  % B=8q21.3 E=8
1.000000  0.000000  % B=8q21.3 E=9
0.000000  1.000000  % B=8q21.3 E=10
1.000000  0.000000  % B=8q21.3 E=11
1.000000  0.000000  % B=8q21.3 E=12
0.000000  1.000000  % B=8q22 E=0
0.000000  1.000000  % B=8q22 E=1
0.000000  1.000000  % B=8q22 E=2
0.000000  1.000000  % B=8q22 E=3
1.000000  0.000000  % B=8q22 E=4
0.988614  0.011386  % B=8q22 E=5
0.995833  0.004167  % B=8q22 E=6
1.000000  0.000000  % B=8q22 E=7
0.927852  0.072148  % B=8q22 E=8
1.000000  0.000000  % B=8q22 E=9
1.000000  0.000000  % B=8q22 E=10
0.000001  0.999999  % B=8q22 E=11
1.000000  0.000000  % B=8q22 E=12
0.000000  1.000000  % B=8q23 E=0
0.975776  0.024224  % B=8q23 E=1
0.997700  0.002300  % B=8q23 E=2
0.000000  1.000000  % B=8q23 E=3
0.000000  1.000000  % B=8q23 E=4
0.000000  1.000000  % B=8q23 E=5
0.996244  0.003756  % B=8q23 E=6
0.997532  0.002468  % B=8q23 E=7
0.917785  0.082215  % B=8q23 E=8
1.000000  0.000000  % B=8q23 E=9
0.000000  1.000000  % B=8q23 E=10
0.959699  0.040301  % B=8q23 E=11
1.000000  0.000000  % B=8q23 E=12
0.000000  1.000000  % B=8q24.1 E=0
0.000000  1.000000  % B=8q24.1 E=1
0.000000  1.000000  % B=8q24.1 E=2
0.000000  1.000000  % B=8q24.1 E=3
0.995159  0.004841  % B=8q24.1 E=4
0.981813  0.018187  % B=8q24.1 E=5
1.000000  0.000000  % B=8q24.1 E=6
1.000000  0.000000  % B=8q24.1 E=7
0.919463  0.080537  % B=8q24.1 E=8
0.741515  0.258485  % B=8q24.1 E=9
0.997769  0.002231  % B=8q24.1 E=10
0.000000  1.000000  % B=8q24.1 E=11
1.000000  0.000000  % B=8q24.1 E=12
0.000000  1.000000  % B=8q24.2 E=0
0.992865  0.007135  % B=8q24.2 E=1
1.000000  0.000000  % B=8q24.2 E=2
0.000000  1.000000  % B=8q24.2 E=3
0.000000  1.000000  % B=8q24.2 E=4
```

```
            0.000000  1.000000  % B=8q24.2 E=5
            1.000000  0.000000  % B=8q24.2 E=6
            1.000000  0.000000  % B=8q24.2 E=7
            0.924497  0.075503  % B=8q24.2 E=8
            0.741515  0.258485  % B=8q24.2 E=9
            0.000000  1.000000  % B=8q24.2 E=10
            1.000000  0.000000  % B=8q24.2 E=11
            1.000000  0.000000  % B=8q24.2 E=12
            0.000000  1.000000  % B=8q24.3 E=0
            0.112860  0.887140  % B=8q24.3 E=1
            0.000000  1.000000  % B=8q24.3 E=2
            0.000090  0.999910  % B=8q24.3 E=3
            1.000000  0.000000  % B=8q24.3 E=4
            1.000000  0.000000  % B=8q24.3 E=5
            1.000000  0.000000  % B=8q24.3 E=6
            1.000000  0.000000  % B=8q24.3 E=7
            0.924497  0.075503  % B=8q24.3 E=8
            0.741494  0.258506  % B=8q24.3 E=9
            1.000000  0.000000  % B=8q24.3 E=10
            0.029102  0.970898  % B=8q24.3 E=11
            1.000000  0.000000 );
}

potential (E | E0)
{
    data = ( 0.980339  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
            0.008555  0.000000  0.000000  0.000000  0.000000  0.011106  % E0=0
            0.000000  0.010561  0.000000  0.000000  0.989439  0.000000  0.000000
            0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  % E0=1
            0.000000  0.000000  0.008663  0.000000  0.000000  0.000000  0.000000
            0.000000  0.000000  0.000000  0.991337  0.000000  0.000000  % E0=2
            0.000000  0.000000  0.000000  0.953516  0.000000  0.000000  0.046484
            0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  % E0=3
            0.000000  0.990250  0.000000  0.000000  0.008187  0.000000  0.000000
            0.000000  0.000000  0.001563  0.000000  0.000000  0.000000  % E0=4
            0.000000  0.000000  0.000000  0.000000  0.000000  0.010004  0.000000
            0.000000  0.000000  0.000000  0.000000  0.989996  0.000000  % E0=5
            0.000000  0.000000  0.000000  0.004252  0.000000  0.000000  0.995748
            0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  % E0=6
            0.005230  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
            0.994770  0.000000  0.000000  0.000000  0.000000  0.000000  % E0=7
            0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
            0.000000  1.000000  0.000000  0.000000  0.000000  0.000000  % E0=8
            0.000000  0.010644  0.000000  0.000000  0.000000  0.000000  0.000000
            0.000000  0.000000  0.989356  0.000000  0.000000  0.000000  % E0=9
            0.000000  0.000000  0.993290  0.000000  0.000000  0.000000  0.000000
            0.000000  0.000000  0.000000  0.006710  0.000000  0.000000  % E0=10
            0.000000  0.000000  0.000000  0.000000  0.000000  0.978542  0.000000
            0.000000  0.000000  0.000000  0.000000  0.021458  0.000000  % E0=11
            0.010199  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
            0.000000  0.000000  0.000000  0.000000  0.000000  0.989801 );
}
```