

Contents

Notation and abbreviations	vii
1 Introduction	1
1.1 Motivation	1
1.2 Assignment	3
1.2.1 Theoretical part	3
1.2.2 Implementation part	4
1.2.3 Application part	4
1.3 Thesis overview	5
I Preliminaries	7
2 Modeling physiological processes in the human body	9
2.1 Glucose-insulin regulation	9
2.2 Cardiovascular system	11
2.2.1 Circulatory system	11
2.2.2 Regulation of the heart rate	11
2.2.3 Mathematical model	12
2.3 Summary	14
3 Bayesian network theory	15
3.1 Bayesian networks	17
3.1.1 Bayesian probability	17
3.1.2 Network structure	18
3.1.3 Conditional probability table	19
3.1.4 Inference	20
3.1.5 Conditional independence and d-separation	21
3.1.6 Canonical models	22
3.1.7 Learning	23
3.2 Dynamic Bayesian networks	28
3.2.1 Network structure	28
3.2.2 Inference	29
3.2.3 Learning	31
3.3 Other temporal reasoning techniques	31
3.3.1 Hidden Markov models	31
3.3.2 Kalman filter models	32
3.4 Summary	33
4 Current software	35
4.1 DSL software	35

4.1.1	SMILE	36
4.1.2	GeNIe	36
4.2	DBN libraries	36
4.2.1	BNT	37
4.2.2	GMTK	37
4.2.3	PNL	37
4.3	DBN modeling tools	38
4.3.1	BayesiaLab	38
4.3.2	Netica	40
4.4	Summary	41
5	Related work	43
5.1	Problem solving in medical applications	43
5.2	Temporal reasoning	44
5.3	Summary	45
II	Model and implementation	47
6	Research architecture	49
7	DBN formalism extensions	53
7.1	K^{th} -order Markov processes	53
7.2	Temporal plate and contemporal nodes	56
7.3	Anchor and terminal nodes	57
7.4	Extended mathematical representation	57
7.5	Empirical study	58
7.5.1	Design	58
7.5.2	Setup	59
7.5.3	Results	59
7.6	Summary	62
8	Temporal reasoning in GeNIe and SMILE	63
8.1	Requirements	64
8.1.1	Actors	64
8.1.2	Use cases	64
8.1.3	Functional requirements	64
8.1.4	Nonfunctional requirements	66
8.1.5	Pseudo requirements	67
8.2	Design	67
8.2.1	Design goals	67
8.2.2	Subsystem structure	68
8.2.3	Class identification	68
8.2.4	Persistent data management	69
8.2.5	Graphical user interface	71
8.3	Implementation	76
8.3.1	DSL_network	77
8.3.2	DSL_nodeEntry	78
8.3.3	DSL_temporalArcs	78
8.3.4	DSL_dbnUnrollImpl	79

8.3.5	DSL_node	79
8.3.6	DSL_nodeDefinition	79
8.3.7	DSL_nodeValue	80
8.3.8	DSL_nodeDefTemporals	81
8.3.9	DSL_nodeValTemporals	81
8.4	Testing	81
8.4.1	Inspecting components	83
8.4.2	Unit testing	83
8.4.3	Integration testing	84
8.4.4	System testing	85
8.4.5	Testing results	86
8.5	Summary	86
9	Modeling with a DBN	87
9.1	DBN structure	87
9.2	DBN parameters	89
9.2.1	Considerations	90
9.2.2	Parameter learning method for the extended DBN formalism	90
9.3	Discretization of continuous data	92
9.3.1	Uniform-sized intervals	93
9.3.2	Uniform-sized binning	94
9.3.3	Single-link clustering	95
9.3.4	Empirical study and results	95
9.4	Summary	97
III	Application and empirical studies	99
10	Glucose-insulin regulation in the human body	101
10.1	Patient simulation	102
10.2	DBN structure	103
10.3	DBN parameters	105
10.4	Empirical study	105
10.4.1	Experimental design	106
10.4.2	Special-purpose software	106
10.4.3	Experiment 1: varying the glucose measurement rate	109
10.4.4	Experiment 2: varying the number of patient cases used for learning	111
10.4.5	Experiment 3: changing the discretization method	112
10.4.6	Conclusions	114
10.5	Summary	114
11	Cardiogenic shock in the human cardiovascular system	115
11.1	Patient simulation	116
11.2	DBN structure	117
11.3	DBN parameters	120
11.4	Empirical study	121
11.4.1	Experimental design	123
11.4.2	Special-purpose software	123
11.4.3	Experiment 1: Comparison of DBN and BN model	124
11.4.4	Experiment 2: Varying the measurement rate	127

11.4.5 Conclusions	129
11.5 Summary	129
IV Results	131
12 Conclusions and future work	133
12.1 Goals and objectives	133
12.1.1 Theoretical part	134
12.1.2 Implementation part	135
12.1.3 Application part	136
12.2 Contributions to the field	137
12.3 Future work	138
Bibliography	141
V Appendices	147
A Temporal reasoning in SMILE tutorial and API	149
A.1 Introduction	149
A.2 Tutorial	149
A.2.1 Creating the DBN	150
A.2.2 Performing inference	152
A.2.3 Unrolling	153
A.3 Public API	154
A.3.1 DSL_network	154
A.3.2 DSL_node	155
A.3.3 DSL_nodeDefinition	156
A.3.4 DSL_nodeDefTemporals	156
A.3.5 DSL_nodeValue	157
A.3.6 DSL_nodeValTemporals	157
B Temporal reasoning in GeNIe tutorial	159
B.1 Creating the DBN	160
B.2 Performing inference	162
B.3 Unrolling	165
C An example of defining and learning the CV DBN	167
C.1 Snippet of dataset	167
C.2 CV DBN definition and learning tool	168

Notation and abbreviations

A summary of the most frequently used notation and abbreviations is given below. The standard convention is adopted that random variables are denoted as capital letters (e.g. X, Y, X_i, Θ), and values of random variables are denoted as lower-case letters (e.g. x, y, x_i, θ). Sets of random variables and values are denoted as bold-face letters (e.g. $\mathbf{X}, \mathbf{Y}, \mathbf{X}_i$ and $\mathbf{x}, \mathbf{y}, \mathbf{x}_i$). Furthermore, *node* and (*random*) *variable* can be used interchangeably, unless explicitly stated otherwise.

α	normalization factor
θ_i	i^{th} parameter of a CPD of a random variable
π	initial state distribution
σ	standard deviation
B_1	BN part of DBN that defines the initial state distribution
B_{\rightarrow}	BN part of DBN that defines the transition model
h_i	i^{th} hypothesis
$I(\theta; \mathbf{x})$	incomplete log-likelihood
$I_c(\theta; \mathbf{x}, \mathbf{z})$	complete log-likelihood
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and variance σ^2
$p(x)$	probability of the assignment of x to X , also used as probability distribution
$Pa(X)$	parent of X
r_i	number of states of i^{th} variable
T	length of time sequence
X_t	random variable at time-slice t
$\mathbf{x}_{1:t}$	values for X for time-sequence 1 to t
X_t^i	i^{th} random variable in time-slice t
($k+1$)TBN	($k+1$)-slice temporal Bayesian network
2TBN	two-slice temporal Bayesian network
API	application program interface
ASR	automatic speech recognition
BN	Bayesian network
BNT	Bayes net toolbox for Matlab
CPD	conditional probability distribution
CPT	conditional probability table
DAG	directed acyclic graph
DBN	dynamic Bayesian network
DSL	Decision Systems Lab
ED	Euclidean distance
EM	expectation-maximization
GeNIe	Graphical Network Interface
GMTK	graphical models toolkit
GUI	graphical user interface
HD	Hellinger distance
HMM	hidden Markov model
ICT	information and communications technology
ICU	intensive care unit

i.i.d.	independent and identically distributed
JPD	joint probability distribution
KFM	Kalman filter model
LPD	local probability distribution
MAP	maximum a posteriori
ME	mean error
MED	mean Euclidean distance
MHD	mean Hellinger distance
ML	maximum likelihood
MSE	mean squared error
ODE	ordinary differential equation
PNL	probabilistic network library
PRNA	Philips Research North America
SDE	stochastic differential equation
SMILE	Structural Modeling, Inference, and Learning Engine

Chapter 1

Introduction

This thesis is about designing probabilistic dynamic models for temporal reasoning, in particular for modeling the physiological processes in a living human being. For centuries, system modeling is a scientists principal tool for studying complex systems. Models exist in all disciplines ranging from Astrophysics to Zoology. Historically, numerical models dominate the modeling practice, for dynamic models the utilization of systems of (ordinary) differential equations is dominant.

A model is a simplified picture of a part of the real world in order to explain and understand the reality. One of the best-known models are *Newton's three laws of motion*. This is a very good model because of its relative simplicity, reliability and predictive power. These three aspects are the reason why models of the motion of physical objects are generally so successful.

Imagine constructing a simple model of the impact of a tile falling from the roof of a building. Newton's laws prescribe exactly what parameters we need, such as the gravitational constant, air resistance, weight of the tile, and height of the building. Furthermore, it prescribes the exact relations between the variables. In this case it is relatively easy to make a good model, because we know the underlying relations and we can measure all parameters directly. The problem that we are interested in is the situation where the relations between variables are not that obvious, and the exact parameters are unknown and cannot be measured. We focus on the medical domain, in particular the physiological processes that form the basis of a major part of the existing medical applications.

1.1 Motivation

Medicine is a field in which much data is generated. Most of this data used to disappear, but in recent years, rapid developments in the field of ICT gave the opportunity to store data in a

maintainable fashion. During the early days of the ICT revolution, most physicians were averse to using ICT in the form of decision support systems or even plain patient databases, and most physicians still are, being afraid that the *craft of healing* will get mechanized. However, the tide is changing. More and more physicians see the benefits that ICT can provide them, and understand that machines will never replace a human's judgment.

Imagine the complex system of physiological processes in a living human being. In this particular case, imagine a subset of this system, the human's glucose-insulin regulation. Recent studies show that the glucose-insulin regulation in critically ill patients malfunctions more often than not. This malfunction is not limited to patients who are known to be diabetic, and thus tight glucose control is needed to decrease postoperative morbidity and mortality.

Tight glucose control involves identifying patients at risk, monitoring blood glucose frequently and using an effective insulin infusion algorithm to control blood glucose within a narrow range. Since the introduction of computers in the medical domain, the majority of models for the glucose-insulin regulation (and other physiological processes) are based on ordinary differential equations (ODEs).

Originally, a system of ODEs is deterministic, meaning that it needs exact values for all its parameters. This poses a problem, because it is impossible to obtain, for instance, the exact insulin production rate of the pancreas for every unique patient at time t as this value changes from patient to patient and from time to time. Another problem is that for more complex physiological processes, the interactions between variables are unknown altogether.

The first problem can be solved by the insight that a satisfying model can be obtained by defining the insulin production rate of the pancreas as a Gaussian distributed variable with a given mean μ and variance σ^2 . Hence, we introduce the concept of *uncertainty* in our model.

Several modeling techniques exist that can handle uncertainty, one of them is extending the system of ODEs with random variables, turning them in stochastic differential equations (SDEs). However, a system of SDEs still does not solve the problem where the interactions between variables are unknown altogether. In this research, we use the dynamic Bayesian network (DBN) formalism, which is a relatively new and promising technique in the field of artificial intelligence that naturally handles uncertainty well and is able to learn the interactions between variables from data.

DBNs are the temporal extension of Bayesian networks (BNs), which are graphical models for probabilistic relationships among sets of variables. A DBN can be used to describe *causal* relations between variables in a probabilistic context. This means that a DBN can provide a good insight in the modeled reality by giving the interaction between variables a meaning, making it robust for erroneous results. The random variables in a DBN do not have to be real numbers, they could just as easily be nominal values (e.g. male/female or true/false), ordinal values (e.g. grade or rank), or intervals (e.g. temperature $\leq 50^\circ\text{C}$, temperature $> 50^\circ\text{C}$).

A DBN model can be obtained by expert knowledge, from a database using a combination of machine-learning techniques, or both. These properties make the DBN formalism very interesting for the medical domain, as this domain has an abundance of both expert knowledge and databases of patient records. Furthermore, many interactions between variables in physiological processes are still unclear. More insight can be gained in these interactions by modeling them with a DBN.

Creating models for temporal reasoning with DBNs sounds promising at first sight, but many questions remain. For instance: *How do we obtain the structure and parameters? What is the performance of inference? How much patient data do we need for learning the parameters? If possible, how much does a DBN outperform a BN when applied to the same problem?* These questions form the motivation for the research presented in this thesis.

1.2 Assignment

This thesis is a result of research done at the Decision Systems Lab in collaboration with Philips Research. The thesis is part of the requirements for my Master of Science degree at the Man-Machine Interaction Group within the Faculty of EEMCS of Delft University of Technology.

The *Decision Systems Lab (DSL)* is a research group within the Department of Information Science and Telecommunications and the Intelligent Systems Program at the University of Pittsburgh. Its mission is maintaining a research and teaching environment for faculty and students interested in the development of techniques and systems that support decision making under uncertainty. The methods include theoretical work, system building, and empirical studies. The DSL utilizes probabilistic, decision-theoretic, and econometric techniques combined with artificial intelligence approaches. The probabilistic reasoning software library SMILE and its graphical user interface GeNIE are a result of DSL research.

Philips Research North America (PRNA) is the North American research laboratory of the well-known multi-national of Dutch origin. This research was done in collaboration with the BioMedical Informatics department of PRNA. The mission of this department is to research, specify, design, and prototype healthcare systems with a focus on medical IT, in particular clinical decision support systems.

The assignment consisted of three main parts:

1. A *theoretical* part that consisted of (1) investigating the existing DBN theory, (2) extending the current DBN formalism, (3) adapting DBN parameter learning, and (4) validating the extensions.
2. An *implementation* part that consisted of (5) investigating current DBN software, (6) designing and implementing DBN functionality in SMILE, and (7) designing an intuitive graphical user interface to DBNs in GeNIE.
3. An *application* part that consisted of (8) applying the extended DBN formalism to physiological processes in the human body, and (9) performing empirical studies on the performance.

An overview of the different smaller parts is given below. These smaller parts can be seen as the *objectives* of this research.

1.2.1 Theoretical part

1. Investigating existing DBN theory A survey needed to be performed on current (D)BN research and its (medical) applications to get a feeling of the field and its challenges. This part of the research is based mainly on research papers published in the last decade [Hul05].

2. Extending DBN formalism Important work on DBNs has been done in [Mur02], making this work currently *the standard* in DBN theory. However, [Mur02] gives a very conservative definition of DBNs using the 2TBN approach. This approach has three main problems: First, only first-order Markov processes can be modeled. Second, when unrolling the network, every node is copied to every time-slice, even if the value of this node is the same for every time-slice. Finally, although it is possible to define a different initial distribution, it is not possible to define a different terminal distribution. This can be useful for inferring variables that are only important at the end of a process, but are not of interest during the process. In the current formalism, these variables need to be inferred every time-slice. We desired to extend the DBN formalism to counter these problems.

3. Adapting DBN parameter learning In most existing literature it is stated that parameter learning with DBNs is just a generalization of parameter learning with BNs. In general, this is true. However, there are some problems that need to be addressed when learning parameters

of DBNs. First, there is the problem of the discretization of time-series data. Using continuous variables in DBNs is problematic, so we need to discretize the incoming continuous signal first. Furthermore, current learning algorithms do not take into account the extended DBN formalism introduced in this thesis.

Another potential problem is learning the initial state distribution of the system. For the initial state distribution, there are two possibilities. In the first possibility, it is just the initial state of the dynamic system. In this case learning is relatively easy, since the initial state distribution can be learned separately from the transition distribution. However, when the initial state distribution represents the stationary distribution of the system, the initial state distribution and the transition distribution become coupled.

We needed to devise a practical method for parameter learning with the extended DBN formalism that accounts for these problems.

4. Validating the extended DBN formalism Extending the DBN formalism is nice, but meaningless if the performance gain is not verified. Experiments needed to be performed to show the performance gain in time and memory for dynamic systems modeled with the current DBN formalism and the extended DBN formalism.

1.2.2 Implementation part

5. Investigating current DBN software Current DBN software needed to be investigated to get an insight in what was possible and what was missing for existing implementations of temporal reasoning. Insight that was gained from this investigation was used in the design and implementation of temporal reasoning in the SMILE and GeNIe software.

6. Designing and implementing the DBN formalism in SMILE The environment that we used for this research is the SMILE library that has been in development since 1997. The SMILE library needed to be extended with temporal reasoning within certain constraints. A design and implementation of temporal reasoning in SMILE that honors these constraints needed to be made and tested.

7. Designing a GUI to DBNs in GeNIe GeNIe is the graphical user interface to SMILE. A GUI for incorporation of the temporal extension in GeNIe needed to be designed, implemented and tested.

1.2.3 Application part

8. Applying the extended DBN formalism to physiological processes in the human body After extending the DBN formalism and implementing it in the DSL software, we wanted to investigate the usability of the extended DBN formalism in the modeling of physiological processes in the human body. In collaboration with Philips Research, the extended formalism was applied to two physiological processes: the glucose-insulin regulation, and the cardiovascular system. The challenge was to convert these dynamic systems to a DBN and learn the parameters in such a way that the resulting DBN represented the dynamic system as close as possible.

9. Performing empirical studies Philips Research is very interested in the possibilities that DBNs can provide in medical diagnosis, prognosis and treatment support systems. For this research, we needed to investigate the following aspects of the usability of DBNs: The performance of inference in general, and as a function of learned data size and missing measurements. Empirical studies were needed to get insight in these aspects. Furthermore, different discretization methods were tested and a comparative study with a BN of the same problem was performed.

1.3 Thesis overview

The thesis is globally separated in five parts:

- Part I contains the physiological processes that we model with DBNs in chapter 2; the theoretical foundation of the DBN formalism in chapter 3; existing software packages for temporal reasoning in chapter 4; and a short overview of related work in the medical field in chapter 5. In this part we describe the preliminaries or framework that this research is performed in.
- Part II contains our research architecture in chapter 6; our extensions of the DBN formalism in chapter 7; our design and implementation of temporal reasoning in the GeNIe and SMILE software in chapter 8; and our methodology of deriving the DBN structure, the DBN parameters, and the discretization process in chapter 9. In this part we extend the framework described in part I. It contains general work on DBN theory and application, which we will apply to specific problems in part III.
- Part III contains our application of the DBN formalism to two physiological processes in the human body: glucose-insulin regulation in chapter 10; and cardiogenic shock in the cardiovascular system in chapter 11. Empirical studies are performed to provide insight in our research questions formulated in section 1.1.
- Part IV contains the conclusions and the possibilities for future work following from this research in chapter 12.
- Finally, part V contains the appendices. Appendix A contains the SMILE tutorial and API that will be included in the SMILE documentation; appendix B contains the GeNIe tutorial that will be included in the GeNIe documentation; and appendix C contains a practical example of defining a DBN structure and learning its parameters from a dataset.

Part I

Preliminaries

*“Everything should be made as simple as possible,
but not simpler.”*
- Albert Einstein.

Chapter 2

Modeling physiological processes in the human body

Traditionally, medicine is a field in which much data is generated. Most of this data used to disappear, but in recent years, rapid developments in the field of ICT gave the opportunity to store data in a maintainable fashion. During the early days of the ICT revolution, most physicians were averse to using ICT in the form of decision support systems or even plain patient databases, and most physicians still are, being afraid that the *craft of healing* will be mechanized. However, the tide is changing. More and more physicians see the benefits that ICT can provide them, and understand that machines will never replace a human's judgment.

In this chapter we give a high-level introduction to two dynamic systems of the physiological processes in the human body: glucose-insulin regulation and the cardiovascular system, mostly because this kind of medical knowledge does not belong to the skill set of the average of computer scientist. Both dynamic systems are modeled using systems of ordinary differential equations (ODEs). We are going to use the dynamic systems to simulate patients. Why simulate patients when there exists so much patient data in medical center databases? The reason for simulation is that medical centers are not eager to share their data, as they are aware that this data can be of inestimable value. Next to that, generally much data processing is needed to extract useful information from raw patient data stored in databases. The research in this thesis focuses on how patient data can be used for decision support systems, not on how to store and retrieve this data.

2.1 Glucose-insulin regulation

Glucose is the vehicle by which energy gets from digested food to the cells of the human body. Insulin is needed to get glucose out of the blood and into the cells of the human body. Blood glucose and insulin form a negative feedback loop, a mechanism designed for maintaining

the human body's dynamic equilibrium (homeostasis). Figure 2.1 shows how the glucose and insulin levels evolve due to the negative feedback loop when insulin is injected in an average patient.

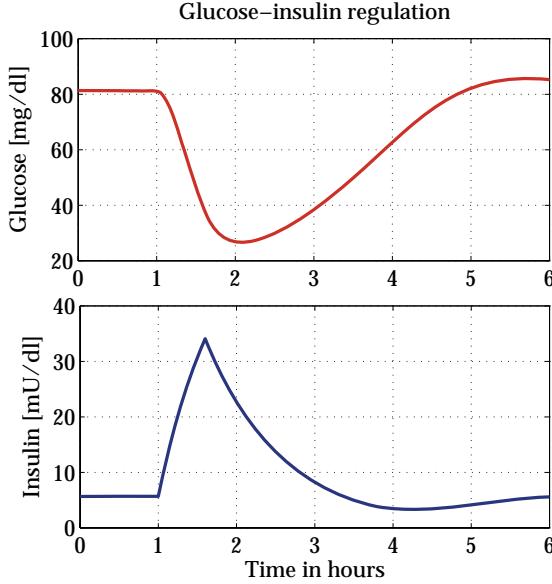


Figure 2.1: Glucose-insulin regulation of an average patient. After one hour, an insulin infusion of 10 [U] is given and the negative glucose-insulin feedback loop gets the patient back to its equilibrium levels.

Glucose-insulin regulation in the human body is mainly administered by the pancreas, therefore a model of the pancreatic function is needed. One of the main functions of the pancreas is to regulate glucose concentration in the blood through the release of the enzyme insulin. The glucose-insulin regulation model presented here is based on Stolwijk and Hardy's version, the parameters are based on average values. The model was modified by adding a term for exogenous insulin infusion [CR04]. The glucose dynamics are governed by

$$C_G \frac{dG}{dt} = U_G + Q_G - \lambda G - \nu GI, \quad G \leq \theta, \quad (2.1)$$

$$C_G \frac{dG}{dt} = U_G + Q_G - \lambda G - \nu GI - \mu(G - \theta), \quad G > \theta, \quad (2.2)$$

and the insulin dynamics are governed by

$$C_I \frac{dI}{dt} = U_I - \alpha I, \quad G \leq \varphi, \quad (2.3)$$

$$C_I \frac{dI}{dt} = U_I - \alpha I + \beta(G - \varphi), \quad G > \varphi, \quad (2.4)$$

where

$G(t)$	instantaneous blood glucose level in [mg/dl] at time [t],
$I(t)$	instantaneous blood insulin level in [mU/dl] at time [t],
$U_G(t)$	exogenous glucose infusion in [mg] at time [t],
$U_I(t)$	exogenous insulin infusion in [mU] at time [t],
C_G	glucose capacitance in the extracellular space,
C_I	insulin capacitance in the extracellular space,
$Q_G(t)$	glucose inflow into blood in [mg/h],
α	insulin destruction rate,
β	insulin production rate by the pancreas,

θ	threshold for renal discharge of glucose in [mg/ml],
λ	tissue usage rate of glucose that is independent of $I(t)$,
μ	glucose rate excreted by the kidneys,
ν	tissue usage rate of glucose that is dependent on $I(t)$,
φ	threshold for pancreatic production of insulin in [mg/ml].

The parameter coefficients have physiological significance, and differ depending on the condition of the patient. In chapter 10, we will see how this model can be used to simulate different patients in an intensive care unit (ICU) situation, and how a dynamic Bayesian network can be build based on patient data. Experiments are performed to investigate the predictive abilities of the resulting dynamic Bayesian network.

2.2 Cardiovascular system

The cardiovascular system is one of the three principal coordinating and integrating systems of the human body. It distributes essential substances and removes by-products of metabolism. It also participates in mechanisms such as the regulation of body temperature. The cardiovascular system consists of a pump, a series of distributing and collecting tubes, and an extensive system of thin-walled vessels. The explanation given here is rather simplified and the interested reader is directed to [BL00] for a detailed explanation.

2.2.1 Circulatory system

The heart consists of two pumps in series. First, there are the right ventricle and atrium which pump blood through the lungs to exchange CO_2 with O_2 . This is known as the small blood circulation or pulmonary circulation. Second, there are the much stronger left ventricle and atrium which pump blood to all other tissues of the body. This is known as the large blood circulation or systemic circulation. The heart has two main phases, the ventricular contraction (systole) where the blood is pumped out and the ventricular relaxation (diastole) where the blood is pumped in the chambers.

In the large blood circulation, the blood is pumped from the left ventricle to the aorta and its arterial branches. In these vessels, the blood contains much O_2 . The branches become narrower and narrower, until they are so thin that the blood can exchange its diffusible substances (and O_2) with the tissue that needs it. These thin-walled vessels are called capillaries. After the exchange, the branches get larger and larger again and keep decreasing in number until the right atrium of the heart is entered via the vena cava. These vessels are called veins and contain blood that is low on O_2 . In the small blood circulation, blood is pumped from the right atrium to the lungs where CO_2 is exchanged with O_2 . After that, the blood enters the left atrium again and the process starts all over again. As a side note, the heart itself receives its O_2 by the coronary arteries, which branch from the aorta. See figure 2.2 for a simplified high-level view of the circulatory system.

2.2.2 Regulation of the heart rate

The heart has a natural excitation, which means that it continues to beat for a while, even when it is completely removed from the body. In the heart, the automatic cells that ordinarily fire at the highest frequency are located in the sinoatrial node or SA node. This node is the natural pacemaker of the heart and is located at the junction of the vena cava and right atrium.

The quantity of blood pumped by the heart each minute is called the cardiac output. The cardiac output is equal to the volume of blood pumped each beat (stroke volume) multiplied by the number of heartbeats per minute (heart rate). Thus the cardiac output can be varied by changing either the heart rate or the stroke volume.

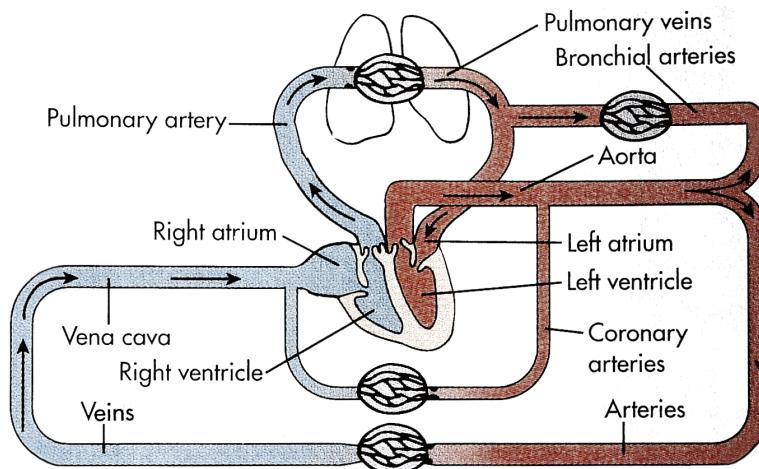


Figure 2.2: Main vessels that make up the circulatory system. Based on similar drawing in [BL00].

Nervous control The heart rate is mainly regulated by the autonomic nervous system. The SA node is usually influenced by both divisions of the autonomic nervous system: (1) Stimulation of the sympathetic system increases the heart rate. The sympathetic system is stimulated by the so-called three f's: flight, fright and fight. Stimulation of the parasympathetic (or vagal) system decreases the heart rate. (2) The parasympathetic system is stimulated during digestion and sleep. Changes in heart rate usually involve a reciprocal action of the two divisions of the autonomic nervous system. The sympathetic system influences the heart rate quickly whereas the parasympathetic system influences the heart rate more gradually. The contractile strength of the heart determines the stroke volume. This is partly regulated by the cardiac nerves, but also by a number of mechanical and humoral factors.

Reflex control is achieved in four different ways: (1) Acute changes in blood pressure reflexly alter the heart rate. Such changes are mediated mainly by the baroreceptors. The baroreceptors are located in the carotid sinuses and the aortic arch. The baroreceptors influence the sympathetic and parasympathetic systems, who in their turn influence the heart rate. (2) The sensory receptors located in the atria of the heart detect changes in blood volume. When blood volume suddenly increases (by for instance an infusion), the heart rate decreases or increases depending on the current heart rate. The sensory receptors send impulses to the certain nuclei in the brain which in their turn send impulses to the SA node that are carried by sympathetic and parasympathetic fibers. Also, stimulating the sensory receptors increase the urine flow. (3) The frequency of respiration has an influence on the heart rate, because neural activity increases in the sympathetic nerve fibers during inspiration and increases in the parasympathetic fibers during expiration. (4) Arterial chemoreceptors influence the heart rate. The way these receptors affect the heart rate is very complex and not interesting for this research, so we will not get into details here.

2.2.3 Mathematical model

The explanation given in the previous section is rather simplified in contrast to the complex system that is functioning in every living human being. However, it is detailed enough to understand the assumptions on which the mathematical model given in [Urs98] is based on.

[Urs98] presents a mathematical model of the interaction between carotid baroregulation and the pulsating heart. It consists of an elastance variable description of the left and right

parts of the heart, the systemic and pulmonary circulations, the afferent¹ carotid baroreceptor pathway, the sympathetic and parasympathetic efferent² activities and the action of several effector mechanisms. The model is used to simulate the interaction among the carotid baroreflex, the pulsating heart, and the effector responses.

Figure 2.3 shows the qualitative model. It consists of eight compartments. Each compartment has a hydraulic resistance (R_j), a compliance (C_j), and an unstressed volume ($V_{u,j}$). Furthermore, L_j denote inertances, P_j the intravascular pressure and F_j the blood flow. Five of the compartments reproduce the systemic circulation: differentiating among the systemic arteries (sa), the splanchnic³ peripheral (sp) and venous (sv) circulations, and the extrasplanchnic⁴ peripheral (ep) and venous (ev) circulations. The other three compartments reproduce the pulmonary circulation: the arterial (pa), peripheral (pp) and venous (pv) circulations. Inertial effects of blood are only taken into account for the larger compartments, since they are not significant in the smaller ones. At the inertances (L_j) preservation of mass and equilibrium of forces are enforced.

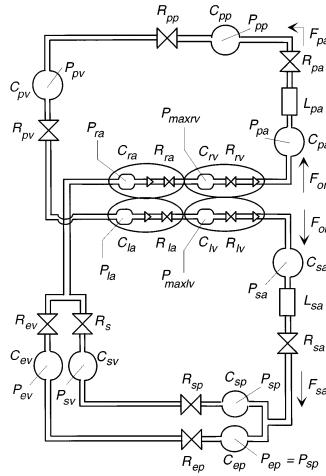


Figure 2.3: Qualitative hydraulic analog of cardiovascular system [Urs98].

The compartments of the heart (left atrium, left ventricle, right atrium, and right ventricle) are denoted by respectively *la*, *lv*, *ra*, and *rv*. The left and right parts of the heart have a similar model, but with different parameters. The heart period varies as a consequence of the baroreflex control action.

In the operation of the carotid baroreflex, a distinction is made among the afferent pathway, the efferent sympathetic and parasympathetic pathways, and the action of several distinct effectors. The distinct effectors are: the response to sympathetic stimulation of peripheral systemic resistances, of systemic venous unstressed volumes, and of heart contractility and the response of heart period to both sympathetic and parasympathetic activities.

This qualitative description, along with the correct equations and their parameters is sufficient to create a satisfactory model of an interesting part of the cardiovascular system. The resulting dynamic system is much more complex than glucose-insulin regulation. In chapter 11, we will show how an extension of this model can be used to simulate different patients in an

¹Afferent is Latin for *going to*. In this context it represents the pulses going from the carotid sinuses to the nucleus of the solitary tract and the nucleus ambiguus in the brain.

²Efferent is Latin for *going from*. In this context it represents the pulses going from the brain to the heart / SA node. This signal is carried by the sympathetic and parasympathetic nervous systems.

³The splanchnic circulation consists of the blood supply to the gastrointestinal tract, liver, spleen, and pancreas.

⁴The extrasplanchnic circulation consists of the blood supply to the other parts of the human body, such as muscle and skin.

intensive care unit (ICU) situation, and we will investigate if the dynamic Bayesian network formalism can provide us with a useful tool to interpret the generated patient data.

2.3 Summary

In this chapter we briefly discussed two physiological processes in the human body. A general understanding of these processes is needed, because they form the basis of the DBN structures that we will derive in part III. Furthermore, the systems of ODEs are going to help us to simulate different patients. This patient data is needed for learning the parameters of the derived DBNs.

Chapter 3

Bayesian network theory

Before we dive into the (dynamic) Bayesian network theory, let us first present an example to get some feeling with the subject. It is a relatively simple example that presents some of the issues involved in modeling with (dynamic) Bayesian networks, without getting into the theory too much. This example will return several times in this chapter for clarification of the theory.

Example (Officer on a navy vessel) Imagine an officer on the bridge of a navy vessel who has the responsibility to watch the radar and alert his superiors in case an incoming flying object, such as a missile, is hostile and poses a thread of impact. Ideally, he does not want to give false alarm, because that causes an unnecessary burden on his superiors. However, if the incoming flying object is hostile and poses an impact threat, he must not hesitate to alarm his superiors, because else the consequences can be fatal. An incoming flying object has a *speed* S , and a *distance* D from the navy vessel. The radar can measure these variables within a certain error range, providing the officer with approximations of the speed and distance: $\tilde{S} = \tilde{s}$ and $\tilde{D} = \tilde{d}$. It is up to the officer to decide on the two measurable variables if the object is *hostile* H and if the object is heading for *impact* I . Based on these variables, he can choose to inform his superiors to take evasive *action* A .

What can we tell about the above problem? First of all, the officer has to deal with seven variables: (1) *speed* S that represents the real speed of the object; (2) its derived \tilde{S} that represents the measured speed; (3) *distance* D that represents the real distance of the object from the navy vessel; (4) its derived \tilde{D} that represents the measured distance; (5) *hostile* H that represents the probability that the object is hostile; (6) *impact* I that represents the probability that the object is heading for impact; and (7) *action* A that represents the probability that evasive action is needed.

These seven variables interact with each other: the variables \tilde{S} and \tilde{D} are a result of the real values S and D (and of the error range of the radar, but we choose not to incorporate

this explicitly); S and D on their turn are a result of whether the object is *hostile*: H can be seen as the *cause* of a certain value for S and D ; S and D influence the probability that the object is heading for *impact* I ; and finally, the combined probabilities of H and I influences the probability of whether to take evasive *action* A . If these interactions are drawn schematically, this results in the graph presented in figure 3.1.

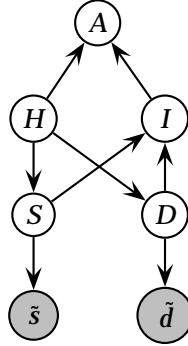


Figure 3.1: The interaction between the variables *speed*, *distance*, *hostile*, *impact*, and *action*, including the noisy measurements.

In this graph, the nodes that are gray can be observed or measured, which means that an outcome can be assigned to the variable that the node represents. In the network shown in figure 3.1, the outcomes are $\tilde{S} = \tilde{s}$ and $\tilde{D} = \tilde{d}$. If a node is white, its value cannot be observed or measured directly, meaning that variable is hidden. In this case, the values of S , D , H , I , and A have to be *inferred* based on the values that can be measured. In fact, the graph presented in figure 3.1 is a Bayesian network structure of the officer's decision problem! We will call this network the *Navy BN* from now on.

When using the BN from figure 3.1, the officer soon finds out that the results are not satisfying. This is because the BN only takes into account the current measured *speed* and *distance* and does not take into account previous measurements. However, a human would take into account previous measurements as well, because they can be extrapolated to obtain better predictions. Thus, S and D should not only rely on the measurements at time t (S_t and D_t), but also on values of S and D at time $t - 1$ (S_{t-1} and D_{t-1}). On their turn, S_{t-1} and D_{t-1} should partly rely on S_{t-2} and D_{t-2} , and so on and so forth. This process is shown in figure 3.2, where the length of the sequence is denoted with $t = T$.

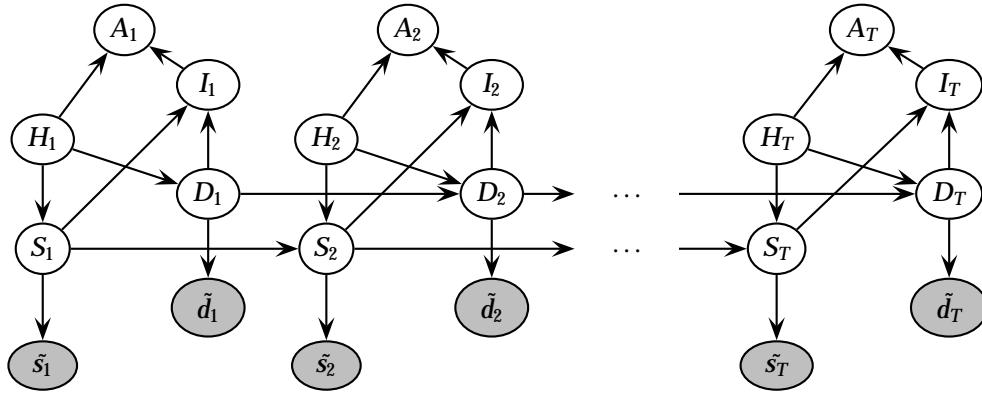


Figure 3.2: Extending the BN with a temporal dimension.

A BN extended with a temporal dimension is called a DBN. In a DBN, every time $t = 1, 2, \dots, T$ represents one time-instant, or *time-slice*. Figure 3.2 presents a possible DBN struc-

ture of the officer's decision problem for a sequence of length T , that we will call the *Navy DBN* from now on. We say possible, because other structures might be better.

The enemy figured out that the navy is using a DBN to detect if an incoming object is hostile and heading for impact. Even worse, the enemy figured out what the DBN structure looks like and decides to anticipate on the fact that the values for S_t and D_t partly rely on S_{t-1} and D_{t-1} . The enemy decides to adapt its missiles in such a way that at every given time t , the distance to the navy vessel and the missile's speed are more or less random. In this case, the arcs between S_t , S_{t-1} and D_t , D_{t-1} are not valid anymore, making the DBN obsolete. After a while, the officer finds out how the enemy adapted its missiles and he decides to remove the arcs between S_t , S_{t-1} and D_t , D_{t-1} and let H_t and I_t partly depend on H_{t-1} and I_{t-1} instead.

Even then, the officer can not be sure that the adapted DBN structure is optimal. For instance, one can argue that the probability of I_t also depends on H_t . If this is the case, why did the officer not draw an arc between I_t and H_t ? There can be several reasons for this, one of them is that I_t and H_t still depend on each other because of a phenomenon called *d-separation*, meaning that I_t and H_t depend on each other through the S_t and D_t variables. How this works exactly will be explained further on in this chapter.

In the above example we tried to introduce some basic terminology from the Bayesian network theory, such as network structure, inference, time-slice, and d-separation, without getting too much into details. Furthermore, it presented some modeling questions that need to be answered when applying the (D)BN formalism to a problem domain, such as: *What is the network structure? Do we need a DBN or is a BN sufficient? and What are the temporal relations?* However, many questions remain. This chapter tries to answer these questions in two sections. In section 3.1 we discuss Bayesian networks. However, Bayesian networks are only useful in static domains, as we have seen in the example above. The temporal extension of a Bayesian network, i.e. a dynamic Bayesian network, is discussed in section 3.2. In the third and final section of this chapter, we discuss some other commonly used temporal probabilistic techniques.

3.1 Bayesian networks

A *Bayesian network* (BN), also known as a *belief network* or a *Bayesian belief network*, is a graphical model for probabilistic relationships among a set of variables [Hec98]. For over a decade, expert systems use BNs in domains where uncertainty plays an important role. Nowadays, BNs appear in a wide range of diagnostic medical systems, fraud detection systems, missile detection systems, etc.

Why are BNs so interesting? BNs have a couple of properties that make them so popular and suitable to use. The five most important properties are, in no particular order, the following: (1) BNs can handle incomplete data sets; (2) BNs allow one to learn about relationships between variables; (3) it is possible to combine expert knowledge and data into a BN; (4) because of the use of Bayesian methods, the overfitting of data during learning can be avoided relatively easy; and (5) BNs are able to model *causal* relationships between variables.

BNs have a qualitative and a quantitative component. The qualitative component is represented by the network structure and the quantitative component is expressed by the assignment of the conditional probability distributions to the nodes. Before we discuss the network structure and conditional probability distributions, first Bayes' Theorem is presented.

3.1.1 Bayesian probability

The main building block of BN theory is *Bayes' Theorem*. This theorem is stated as follows:

$$p(X|Y) = \frac{p(Y|X) \cdot p(X)}{p(Y)}, \quad (3.1)$$

where:

- $p(X|Y)$ is the posterior probability of the hypothesis X , given the data Y ,
- $p(Y|X)$ is the probability of the data Y , given the hypothesis X , or the likelihood of the data,
- $p(X)$ is the prior probability of the hypothesis X , and
- $p(Y)$ is the prior probability of the data Y , or the evidence.

Equation 3.1 states that by observing evidence, the prior probability of the hypothesis changes to a posterior probability.

Two other rules are important in BNs. The first one is the *expansion rule*. Consider the situation where X and Y are random variables with k possible outcomes:

$$\begin{aligned} p(X) &= p(X|y^{k=1}) \cdot p(y^{k=1}) + p(X|y^{k=2}) \cdot p(y^{k=2}) + \cdots + p(X|y^{k=k}) \cdot p(y^{k=k}) \\ &= \sum_Y p(X|Y) \cdot p(Y). \end{aligned} \quad (3.2)$$

Applying the expansion rule means that we can introduce variables on the right-hand side of the equation, as long as we sum over all their possible values. This concept is also known as the *marginal probability* of X , meaning that only the probability of one variable (X) is important and all information about other variables (Y) is ignored.

The second rule is the *chain rule*. This rule is derived by writing Bayes' Theorem in the following form, which is called the *product rule*:

$$\begin{aligned} p(X, Y) &= p(X|Y) \cdot p(Y) \\ &= p(Y|X) \cdot p(X). \end{aligned} \quad (3.3)$$

Successive application of the product rule yields the chain rule:

$$\begin{aligned} p(X_1, \dots, X_n) &= p(X_1, \dots, X_{n-1}) \cdot p(X_n|X_1, \dots, X_{n-1}) \\ &= p(X_1, \dots, X_{n-2}) \cdot p(X_{n-1}|X_1, \dots, X_{n-2}) \cdot p(X_n|X_1, \dots, X_{n-1}) \\ &= p(X_1) \cdot p(X_2|X_1) \cdots p(X_n|X_1, \dots, X_{n-1}) \\ &= p(X_1) \prod_{i=2}^n p(X_i|X_1, \dots, X_{i-1}). \end{aligned} \quad (3.4)$$

For every X_i , there may be a subset of (X_1, \dots, X_{i-1}) such that X_i and the subset are conditionally independent. This means that this subset can be left out of the original set (X_1, \dots, X_{i-1}) . When the subset is empty, X_i is conditionally dependent on all variables in (X_1, \dots, X_{i-1}) .

3.1.2 Network structure

The qualitative part of the BN is represented by its structure. A BN is a directed, acyclic graph (DAG) where the nodes represent the set of random variables and the directed arcs connect pairs of nodes. We have already seen two examples of BN structures in the figures 3.1 and 3.2. The nodes in the BN represent *discrete*¹ random variables. If an arc points from X_1 to X_2 then X_1 is a *parent* of X_2 and X_2 is a *child* of X_1 . A parent directly influences its children. Furthermore, every node has its own local probability distribution. All these components together form the joint probability distribution (JPD) of the BN. The joint probability distribution of \mathbf{X} follows from applying equation 3.4 on the network structure. The parents of X are denoted by $\mathbf{Pa}(X)$:

$$p(X_1, \dots, X_n) = \prod_{i=1}^n p(X_i|\mathbf{Pa}(X_i)). \quad (3.5)$$

¹The theory can be easily extended to handle continuous random variables with arbitrary probability distributions.

$p(X_i|\mathbf{Pa}(X_i))$ is called the *local probability distribution* (LPD). The process of breaking up the joint probability distribution into local probability distributions is called *factorization*, which results in an efficient representation that supports fast inference (subsection 3.1.4). This is a property of BNs that forms a major contribution to its success. To come back to the restriction for a BN to be a DAG, this follows from equation 3.5. To see this, imagine the network in figure 3.3. For this network, the JPD can be calculated by:

$$\begin{aligned} p(X_1, X_2, X_3) &= p(X_1|\mathbf{Pa}(X_1)) \cdot p(X_2|\mathbf{Pa}(X_2)) \cdot p(X_3|\mathbf{Pa}(X_3)) \\ &= p(X_1|X_3) \cdot p(X_2|X_1) \cdot p(X_3|X_2) . \end{aligned} \quad (3.6)$$

It is easy to see that the resulting calculation can never be written back to the original chain rule (equation 3.4), which in its turn was a direct result of successive application of the product rule form of Bayes' Theorem. In short, allowing cycles in the network is not consistent with Bayes' theorem!

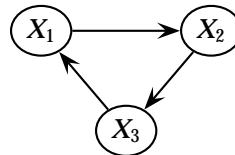


Figure 3.3: A network with a cycle.

3.1.3 Conditional probability table

The quantitative part of the BN is represented by the assignment of the conditional probability distributions to the nodes. Each node in a BN has a *conditional probability table* (CPT) that defines the conditional probability distribution (CPD) of the represented discrete random variable. To get insight in what a CPT looks like, we take a subset of the Navy BN from figure 3.2. The subset, including its CPT parameters are shown in figure 3.4. Note that the CPT entries of the parents of I and H are already marginalized out, according to equation 3.2. The entries in

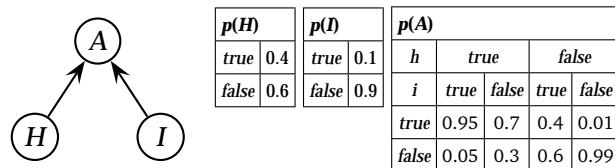


Figure 3.4: A subset of the Navy BN with the hostile, impact, and action variables.

the CPTs tell us what the probabilities are of a hidden node given its parents. For instance, the probability that I is *true* without inserting any evidence is equal to 0.1 (according to figure 3.4).

- || The officer knows for sure that an incoming object is hostile and heading for impact. In this case, the probability of A can be read from the CPT by looking at the entry for $p(A|h = \text{true}, i = \text{true}) = 0.95$.

One could ask why this probability is 0.95 and not just 1.0, because a hostile object (such as a missile) is heading for impact and this can not be a good thing! The answer to this question is that there could be other factors involved that the officer did not explicitly put in the model. For instance, the navy vessel could be navigating within an enemy minefield. In this situation, getting hit by a missile can be a better scenario than getting hit by a mine. The officer could also choose to explicitly insert a variable *minefield* into the BN, but he choose to implicitly model

such external factors through the probabilities. When modeling a problem, the modeler always has to solve this kind of dilemma's.

A potential problem of using CPTs for defining the conditional probability distributions is their size. The size of the CPT of variable $X_i \in \mathbf{X}$ depends on the number of states r_i of the variable and the number of states r_j of each parent $X_j \in \mathbf{Pa}(X_i)$ as follows:

$$\text{size}(\text{CPT})_i = r_i \cdot \prod_{r_j=r(X_j \in \mathbf{Pa}(X_i))} r_j . \quad (3.7)$$

If r_j is equal for all parents of X_i (as is the case in figure 3.4), the equation simplifies to:

$$\text{size}(\text{CPT})_i = r_i \cdot (r_j)^n , \quad (3.8)$$

where n is the number of parents. As can be seen from either equation 3.7 or equation 3.8, the size of the CPTs grows exponentially with the number of parents. Knowing this, it is important to keep the number of nodes that node X_i is dependent on as low as possible.

3.1.4 Inference

Inference is the process of calculating the probability of one or more variables \mathbf{X} given some evidence \mathbf{e} . The evidence is expressed as an instantiation of some variables in the BN. In short: $p(\mathbf{X}|\mathbf{e})$ needs to be calculated. The two rules that are needed for inference are Bayes' theorem (3.1) and the expansion rule (3.2). Inference can be best explained by an example.

- || An observer from the mainland sees the navy vessel making an evasive action, thus A is equal to *true*. The observer now wonders what the probability of the object approaching the vessel being hostile equals to. Thus, $p(H = \text{true})$ needs to be calculated.

H is called a *query variable* and A is called an *evidence variable*. The observer wants to know what the probability is of $H = \text{true}$, given $A = \text{true}$. The calculation comes down to:

$$p(\mathbf{x}|\mathbf{e}) = p(h|a) = \frac{p(a|h) \cdot p(h)}{p(a)} . \quad (3.9)$$

This expression needs some rewriting using equations 3.1 and 3.2 to enable us to insert the probabilities in the CPTs of figure 3.4 directly:

$$\frac{p(a|h) \cdot p(h)}{p(a)} = \frac{\sum_I (p(a|hi) \cdot p(i)) \cdot p(h)}{\sum_H \sum_I p(a|hi) \cdot p(h) \cdot p(i)} , \quad (3.10)$$

$$\frac{(0.95 \cdot 0.1 + 0.7 \cdot 0.9) \cdot 0.4}{0.95 \cdot 0.4 \cdot 0.1 + 0.4 \cdot 0.6 \cdot 0.1 + 0.7 \cdot 0.4 \cdot 0.9 + 0.01 \cdot 0.6 \cdot 0.9} = \frac{0.29}{0.3194} \approx 0.91 . \quad (3.11)$$

The probability of H being *true*, knowing that A is *true* equals to approximately 0.91. The probability that H is *false* given that A is *true* equals to $1 - p(h) \approx 0.09$. As can be seen in equation 3.10, the denominator is complex, but constant no matter the value of the queried variable H . Basically, the denominator is a normalization constant that is needed to make the probabilities of the different values of H sum up to 1. That is why often the normalization factor α is introduced:

$$\alpha(p(a|h) \cdot p(h)) = \alpha\left(\sum_I (p(a|hi) \cdot p(i)) \cdot p(h)\right) , \quad (3.12)$$

changing the calculation into:

$$H = \text{true}: \quad \alpha((0.95 \cdot 0.1 + 0.7 \cdot 0.9) \cdot 0.4) = 0.29 , \quad (3.13)$$

$$H = \text{false}: \quad \alpha((0.4 \cdot 0.1 + 0.01 \cdot 0.9) \cdot 0.6) = 0.0294 . \quad (3.14)$$

$\alpha \langle 0.29, 0.0294 \rangle$ results in the normalized probabilities ($\approx 0.91, \approx 0.09$)! Despite this simplification of the original calculation, exact inference of large, multiply connected BNs becomes very complex. In fact, it is *NP-hard* [Coo90]. Because of this intractability, approximate inference methods are essential. In general, *randomized sampling algorithms*, also called *Monte Carlo algorithms*, are used.

For exact inference, *variable elimination* algorithms or *junction tree* algorithms can be used. Variable elimination uses a smart way to rewrite the inference calculations by choosing a specific elimination order of the variables, making computation more efficient. Junction tree algorithms convert the multiply connected BNs to *junction trees*, after which inference can be performed using variable elimination (Shafer-Shenoy algorithm) [SS88] or belief propagation (Lauritzen-Spiegelhalter algorithm) [LS88]. Of course, this conversion can be very complex as well, so it is not suitable for every BN.

A third technique that can be used to reduce complexity of the inference computations is called *relevance reasoning*. This is a preprocessing step that explores structural and numerical properties of the model to determine what parts of the BN are needed to perform the computation. Relevance reasoning reduces the size and the connectivity of a BN by pruning nodes that are computationally irrelevant to the nodes of interest in the BN [DS94].

3.1.5 Conditional independence and d-separation

In the previous section we stated that exact inference is very complex. In fact, with n variables, inference has a complexity of 2^n . However, the conditional independence of variables is not taken into account yet. Conditional independence can cut the cost of inference calculations drastically, turning the complexity from 2^n to $n \cdot 2^k$ were n is the total number of variables and k the maximum number of variables that a single variable can be dependent on.

Variables with no connection whatsoever are conditionally independent, but variables with an indirect connection can still be conditionally dependent. Again, we look at a subset of the Navy BN from figure 3.1, this time with the variables *hostile*, *impact*, *distance*, and *speed*. This subset is shown in figure 3.5.

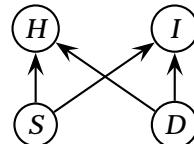


Figure 3.5: A subset of the Navy BN with the *hostile*, *impact*, *distance* and *speed* variables.

The officer on the navy vessel did not model an explicit relation between an object being hostile and heading for impact. This is expressed by the absence of an arc between H and I . However, he soon finds that if he observes $H = \text{true}$, the probability of $I = \text{true}$ increases. On the other hand, if he observes the exact values for D and S , observing H does not influence I .

It seems that the variables H and I can be dependent after all! At least, for specific configurations of the BN. To understand why and for what configurations, we take a closer look at what happens when observing $H = \text{true}$: If $H = \text{true}$, more becomes known about the S and D variables, because hostile objects have a different behavior than friendly objects. But, if more becomes known about S and D , also more becomes known about I , because they have an explicit connection in the form of an arc going from S and D to I . What happens is that the evidence of H propagates through S and D to I . If S and D are observed, the connection between H and I is blocked.

This example introduces the notion of d-separation [Pea88]. The four configurations in figure 3.6 are used to describe the definition of *d-separation*. The variables in a BN are conditionally independent if and only if they are d-separated.

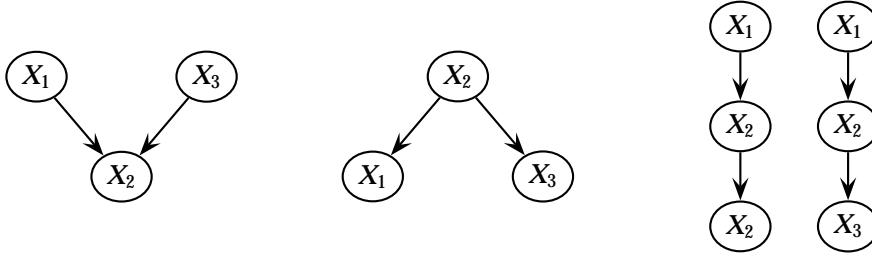


Figure 3.6: From left to right: A converging (or v-structure), a diverging and two serial BNs.

Definition (D-separation)

Two nodes X_1 and X_3 in a BN are d-separated if for all paths between X_1 and X_3 , there is an intermediate node X_2 for which either:

- The connection is serial or diverging and the state of X_2 is known for certain; or
- The connection is converging and neither X_2 (nor any of its descendants) have received any evidence at all.

For large BNs, d-separation of two variables can be solved using the *Bayes ball algorithm*, a reachability algorithm that makes use of the notion of a ball and a set of bouncing rules to determine whether a set of variables can be reached from another set of variables through a third set of variables [Sha98].

3.1.6 Canonical models

Generally, it is necessary to assign each node a set of conditional probabilities that grows exponentially with the number of its parents. This makes inference, training and storage very complex for nodes with many parents. To reduce complexity, causal interaction models, also called *canonical models*, were introduced [Pea88]. In this section, the (leaky) Noisy-OR and -MAX models are discussed. The main advantage of using these models is that the set of conditional probabilities grows linearly (rather than exponentially) with the number of parents.

Noisy-OR The *Noisy-OR* model is based on the deterministic OR gate. Each binary variable X_i produces the binary variable Y independently of the other variables with a certain probability c_i . This probability is modeled using an inhibitor node Z_i . Absence of all \mathbf{X} yields absence of Y . For a certain configuration of \mathbf{x} , the probability of Y is calculated as follows:

$$p(y|\mathbf{x}) = 1 - \prod_i (1 - c_i \cdot x_i). \quad (3.15)$$

Leaky Noisy-OR This model is based on the assumption that next to the causes that are modeled in the Noisy-OR gate, other causes can produce Y as well. In most real-world applications, a common problem is that not all causes are known or can be modeled in the BN. In the *leaky Noisy-OR* model, this problem is solved by introducing a probability c_L that Y is present given the absence of all \mathbf{X} . This extra probability is introduced using an extra node Z_L . Using the leaky Noisy-OR, the probability of Y is calculated as follows:

$$p(y|\mathbf{x}) = 1 - (1 - c_L) \cdot \prod_i (1 - c_i \cdot x_i). \quad (3.16)$$

(Leaky) Noisy-MAX The *Noisy-MAX* and *leaky Noisy-MAX* models are generalizations of the Noisy-OR variants for non-binary variables. The Noisy-MAX model is based on the deterministic max function: $y = \max(\mathbf{x})$. The variables should be ordinal and have an equal number of states. Again, a probability c_L can be introduced to obtain the leaky variant.

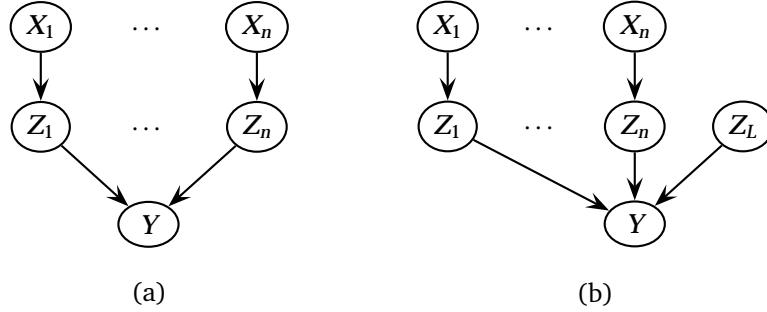


Figure 3.7: (a) A Noisy-OR/MAX model. (b) A leaky Noisy-OR/MAX model.

Other models Similar to Noisy-OR and Noisy-MAX models, also Noisy-AND and Noisy-MIN models exist, which are based on respectively the deterministic AND and the min function. Currently, a great deal of research effort goes into finding useful canonical models.

3.1.7 Learning

One very useful property of BNs is their ability to learn from observations. Learning of BNs can be divided into two types: *parameter learning* and *structure learning*. With parameter learning, the structure of the BN is given and only the CPT parameters are learned. With structure learning, the BN structure itself is learned. Before parameter learning and structure learning are discussed, a short introduction on the concept of *Bayesian learning* is given.

Bayesian learning forms the basis of learning in BNs and calculates the probability of each of the hypotheses given the data. Predictions are made by averaging over all probabilities of the hypotheses. The probability of each hypothesis h_i given data \mathbf{d} is given by:

$$p(h_i|\mathbf{d}) = \alpha \cdot p(\mathbf{d}|h_i) \cdot p(h_i). \quad (3.17)$$

If we want to make a prediction about an unknown quantity X with given data \mathbf{d} , an average over all possible hypotheses h_i is calculated using the expansion rule:

$$\begin{aligned} p(X|\mathbf{d}) &= \sum_i p(X|\mathbf{d}, h_i) \cdot p(h_i|\mathbf{d}) \\ &= \sum_i p(X|h_i) \cdot p(h_i|\mathbf{d}) \\ &= \sum_i \alpha \cdot p(X|h_i) \cdot p(\mathbf{d}|h_i) \cdot p(h_i). \end{aligned} \quad (3.18)$$

Note that the set of hypotheses forms an intermediate variable between the unknown quantity and the data, the unknown quantity does not depend directly on the data.

Example (A biased coin) Imagine we are tossing a coin and predicting the outcome. Since the coin can be unfair, the prediction whether it lands on *heads* or *tails* is based on the hypotheses: h_1 that the coin is fair, h_2 that the coin is biased with a probability of $\frac{3}{4}$ toward *heads* and h_3 that the coin is biased with a probability of $\frac{3}{4}$ toward *tails*. It is interesting to know which hypothesis h_i is *true* given the previous outcomes \mathbf{d} , because that defines the probability of the next outcome *Toss*. To get more insight, a prior probability is assigned to the hypotheses, for example $\langle \frac{1}{3}, \frac{1}{3}, \frac{1}{3} \rangle$, which means that in this case, all hypotheses can occur with the same probability. The posterior probability is updated after each observation in \mathbf{d} using the likelihood $p(\mathbf{d}|h_i)$ of each h_i , because with each observation, more about the unknown hypothesis becomes known. If for instance after the first 5 tosses, the outcomes are all *heads*, then the posterior probabilities of h_i become:

$$p(h_1|\mathbf{d}) = \alpha \cdot p(\mathbf{d}|h_1) \cdot p(h_1) = \alpha \cdot (\frac{1}{2})^5 \cdot (\frac{1}{2})^0 \cdot (\frac{1}{3}) = \alpha \cdot \frac{32}{3072} = \frac{32}{276}, \quad (3.19)$$

$$p(h_2|\mathbf{d}) = \alpha \cdot p(\mathbf{d}|h_2) \cdot p(h_2) = \alpha \cdot (\frac{3}{4})^5 \cdot (\frac{1}{4})^0 \cdot (\frac{1}{3}) = \alpha \cdot \frac{243}{3072} = \frac{243}{276}, \quad (3.20)$$

$$p(h_3|\mathbf{d}) = \alpha \cdot p(\mathbf{d}|h_3) \cdot p(h_3) = \alpha \cdot (\frac{1}{4})^5 \cdot (\frac{3}{4})^0 \cdot (\frac{1}{3}) = \alpha \cdot \frac{1}{3072} = \frac{1}{276}. \quad (3.21)$$

Clearly, h_2 is the most probable hypothesis after these five observations. In fact, no matter what the prior assignment may be, as long as the true hypothesis is not ruled out, it will eventually dominate the Bayesian prediction, since any false hypothesis will disappear as more observation data is collected. To calculate which side of the coin to bet on, we use equation 3.18. The calculation below clearly shows that *heads* is the outcome to bet on.

$$\begin{aligned} p(Toss = \text{heads}|\mathbf{d}) &= \sum_i p(X|h_i) \cdot p(h_i|\mathbf{d}) \\ &= \frac{1}{2} \cdot \frac{32}{276} + \frac{3}{4} \cdot \frac{243}{276} + \frac{1}{4} \cdot \frac{1}{276} = \frac{794}{1104}, \end{aligned} \quad (3.22)$$

$$p(Toss = \text{tails}|\mathbf{d}) = 1 - \frac{794}{1104} = \frac{310}{1104}. \quad (3.23)$$

This example has only a few hypotheses, but many real-life problems have an enormous hypothesis space. In equation 3.18, all hypotheses are used to calculate the probability density of X , even the hypotheses with a very small probability. In real-life problems, the calculation of equation 3.18 becomes very complex. A common approximation is *Maximum A Posteriori* (MAP) estimation. Instead of all hypotheses, only the best hypothesis, which is the h_i that maximizes $p(h_i|\mathbf{d})$, is taken: $p(X|\mathbf{d}) \approx p(X|h_{\text{MAP}})$. Another simplification is assuming an equal prior for all hypotheses, as is the case in the biased coin example. In this case, MAP estimation reduces to choosing the h_i that maximizes $p(\mathbf{d}|h_i)$. This is called *maximum-likelihood* (ML) estimation and provides a good approximation as long as the data set is large enough.

Parameter learning A BN consisting of discrete random variables has a CPT for every variable. Every entry θ in a CPT is unknown at first and must be learned. To get back to the coin example: Suppose nothing is known about how the coin is biased. This means that it can be biased toward *heads* with a probability $0 < \theta < 1$ and toward *tails* with a probability $0 < (1 - \theta) < 1$, as can be seen in figure 3.8. Using ML estimation, only an expression for $p(\mathbf{d}|\theta)$

$p(\text{Toss})$	
heads	θ
tails	$1 - \theta$

 Toss

Figure 3.8: A BN of the biased coin example and its CPT.

and its maximum needs to be found to obtain the optimal value for the parameter θ . In the coin example, this will result in a hypothesis equal to the proportion of tosses that gave *heads* divided by the total number of tosses. The obvious problem with this approach is the large data set needed to get a satisfactory result. Such a large data set is not available most of the time.

The solution to this problem is introducing a hypothesis prior over the possible values of the needed parameter. This prior is updated after each new observation. In the coin example, the parameter θ is unknown. Instead of giving it one prior value, a continuous prior distribution $p(\Theta)$ is assigned to it. A *beta distribution* is used to achieve this. The beta distribution is defined as:

$$p(\theta) = \text{Beta}(\theta|\alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha) \cdot \Gamma(\beta)} \cdot \theta^{(\alpha-1)} \cdot (1 - \theta)^{(\beta-1)}, \quad (3.24)$$

where Γ is the Γ function, defined as:

$$\begin{cases} \Gamma(x+1) &= x \cdot \Gamma(x), \\ \Gamma(1) &= 1. \end{cases} \quad (3.25)$$

Beta distributions have the convenient properties that they are only non-zero over the interval $[0, 1]$ and integrate to one. Furthermore, their *hyperparameters* α and β are the number of (*events* + 1) and (*events* + 1). These are called hyperparameters to distinguish them from the parameter θ that is being learned and they must be greater than zero to be able to normalize the distribution. A different prior can be introduced by setting the hyperparameters before the learning process starts.

The mean value of the beta distribution is $\frac{\alpha}{\alpha+\beta}$, so a proportionally larger number of α means that Θ gets closer to one. Larger values of $\alpha + \beta$ result in a peaked distribution, which means a larger data set gives more certainty about Θ . Some examples of the beta distribution are shown in figure 3.9.

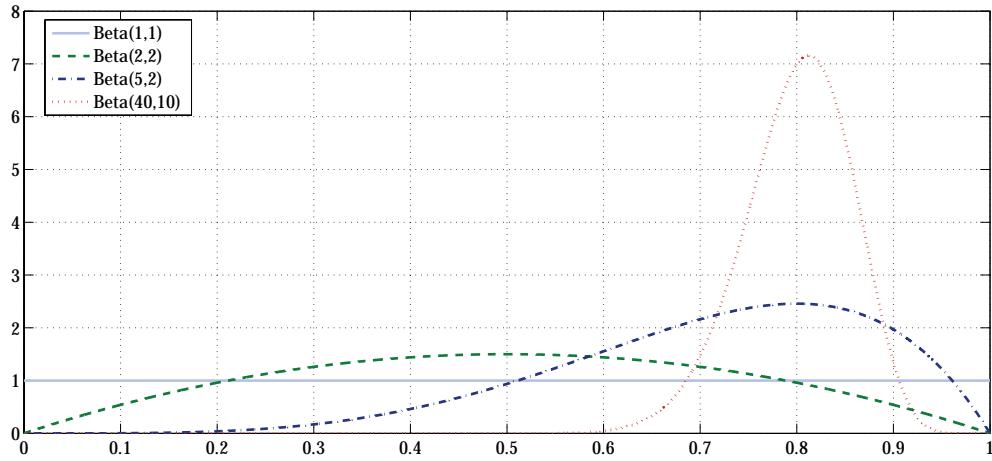


Figure 3.9: Four example beta distributions, as α becomes proportionally larger than β , Θ gets closer to one and as more data is gathered, the distribution becomes more peaked.

The *Dirichlet distribution* is a generalization of the beta distribution for multinomial distributions. In the multinomial case, the observed variable X is discrete and has $r > 2$ possible states. Its parameters are $\theta = \{\theta_2, \dots, \theta_r\}$ (and $\theta_1 = 1 - \sum_{k=2}^r \theta_k$). The hyperparameters are defined as $\alpha = \{\alpha_1, \dots, \alpha_r\}$:

$$p(\theta|\alpha) = \text{Dir}(\theta|\alpha) = \frac{\Gamma(\sum_{k=1}^r \alpha_k)}{\prod_{k=1}^r \Gamma(\alpha_k)} \prod_{k=1}^r \theta_k^{\alpha_k-1}. \quad (3.26)$$

In parameter learning, usually parameter independence is assumed, meaning that every parameter can have its own beta or Dirichlet distribution that is updated separately as data are observed:

$$p(\Theta) = \prod_i p(\Theta_i), \quad (3.27)$$

where each Θ_i denotes a specific combination of the state of the node's parents.

The assumption of parameter independence results in the possibility to incorporate the parameters into a larger BN structure. Figure 3.10 illustrates the Navy BN with the incorporation of the learning parameters Θ . The process of learning BN parameters can be formulated as an inference problem of an extended BN, making it possible to use the same techniques as discussed in section 3.1.4 to solve the learning problem $p(\theta|d)$.

In the case that the data is incomplete or partially observed (i.e. some variables in some cases are not observed), approximation algorithms need to be used. One of the most commonly used algorithms, the EM algorithm, will be discussed next.

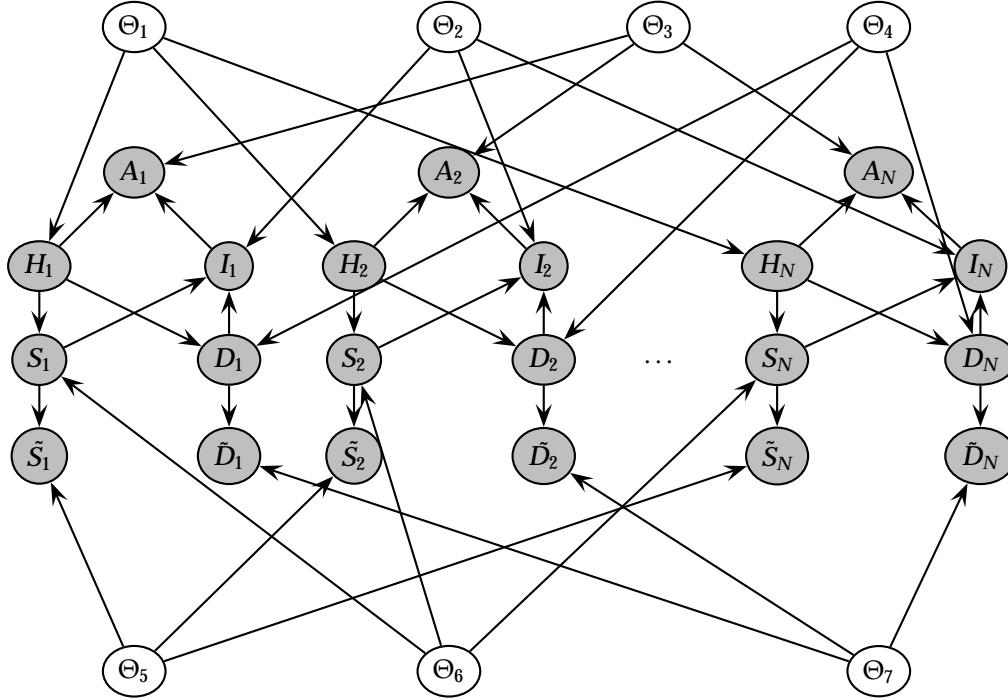


Figure 3.10: The Navy BN with incorporation of the learning parameters Θ and N data samples.

The EM algorithm If the data is only partially observed because of missing data or hidden variables, learning becomes much harder. In this case, one usually has to settle for a locally optimal solution, obtained using an approximation method such as the expectation-maximization algorithm or *EM algorithm*. Much has been written about the EM algorithm. Papers that specifically apply the EM algorithm on BNs include [Lau95, Zha96, BKS97]. This paragraph presents a brief summary of the algorithm.

The EM algorithm is an *iterative hill-climbing algorithm* that consists of an *expectation step* (E step) and a *maximization step* (M step). In the E step, the unobserved values are basically *filled-in*. This means that the probabilities of the unobserved variables are calculated given the observed variables and the current values of the parameters. In short, the expected sufficient statistics are computed. In the M step, the parameters are recomputed using the filled-in values as if they were observed values. This step is no more complex than it would be if the missing values were observed in the first place. The alternating process of filling-in the missing values and updating the parameters is iterated until convergence.

To get a general understanding of the algorithm, let \mathbf{x} denote the observable variables and \mathbf{z} the variables that are hidden. The resulting probability model is $p(\mathbf{x}, \mathbf{z}|\theta)$. If \mathbf{z} could be observed, than the problem reduces to ML estimation maximizing the quantity²:

$$l_c(\theta; \mathbf{x}, \mathbf{z}) \triangleq p(\mathbf{x}, \mathbf{z}|\theta) . \quad (3.28)$$

In the context of the EM algorithm, this is referred to as the *complete log likelihood*. However, \mathbf{z} is not observed and one has to deal with the *incomplete log likelihood*:

$$l(\theta; \mathbf{x}) = \log p(\mathbf{x}|\theta) = \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}|\theta) . \quad (3.29)$$

²Previously, the likelihood instead of the log-likelihood was maximized. The log-likelihood is monotonically related to the likelihood and therefore, maximizing the one is equivalent to maximizing the other. However, since in the log-likelihood products are converted to summations, this form is preferable.

Equation 3.29 shows that because of the summation sign, the separate components of θ can not be written in separate terms outside the logarithm, which means that they cannot be maximized independently. That is why the EM algorithm is needed, to estimate the values \mathbf{z} first and the parameters θ second in an iterative way. The notion that forms the basis of the EM algorithm is the *expected complete log likelihood*, which is defined using the *averaging distribution* $q(\mathbf{z}|\mathbf{x})$:

$$\langle l_c(\theta; \mathbf{x}, \mathbf{z}) \rangle_q \triangleq \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}, \theta) \cdot \log p(\mathbf{x}, \mathbf{z}|\theta) . \quad (3.30)$$

The expected complete log likelihood acts as a surrogate to the complete log likelihood, because the complete log likelihood can not be maximized directly. The averaging distribution $q(\mathbf{z}|\mathbf{x})$ needs to be chosen, after which the surrogate is maximized to obtain new values of θ , which can represent an improvement from the initial values of θ . If so, one can iterate this process and hill-climb. To obtain the final form of the EM algorithm, the averaging distribution $q(\mathbf{z}|\mathbf{x})$ is used to provide a lower bound on the incomplete log likelihood:

$$l(\theta; \mathbf{x}) = \log p(\mathbf{x}|\theta) = \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}|\theta) = \log \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \cdot \frac{p(\mathbf{x}, \mathbf{z}|\theta)}{q(\mathbf{z}|\mathbf{x})} \quad (3.31)$$

$$\geq \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \cdot \log \frac{p(\mathbf{x}, \mathbf{z}|\theta)}{q(\mathbf{z}|\mathbf{x})} \triangleq \mathcal{L}(q, \theta, \mathbf{x}) . \quad (3.32)$$

In equation 3.32, a concept called *Jensen's inequality* is used to show that the function $\mathcal{L}(q, \theta, \mathbf{x})$ is a lower bound for the incomplete log likelihood.

These are all the ingredients required for the EM algorithm. To summarize, the EM algorithm is a *coordinate ascent algorithm* on the function $\mathcal{L}(q, \theta, \mathbf{x})$. This means that hill-climbing is achieved indirectly, with the advantage that the expected log likelihood is maximized instead of the incomplete log likelihood, which is a substantial simplification. At every iteration, first $\mathcal{L}(q, \theta, \mathbf{x})$ is maximized using the θ obtained from the previous step with respect to $q(\mathbf{z}|\mathbf{x})$. After that, $\mathcal{L}(q, \theta, \mathbf{x})$ is maximized with respect to θ to obtain the updated θ :

$$(\text{E step}) \quad q^{t+1} = \arg \max_q \mathcal{L}(q, \theta^t, \mathbf{x}) , \quad (3.33)$$

$$(\text{M step}) \quad \theta^{t+1} = \arg \max_{\theta} \mathcal{L}(q^{t+1}, \theta, \mathbf{x}) . \quad (3.34)$$

There are two final important points to mention. The first point is that the M step using $\mathcal{L}(q, \theta, \mathbf{x})$ is indeed equivalent to maximizing $\langle l_c(\theta; \mathbf{x}, \mathbf{z}) \rangle_q$. The second point is that $\mathcal{L}(q, \theta^t, \mathbf{x})$ needs to be maximized with respect to the averaging distribution $q(\mathbf{z}|\mathbf{x})$. The choice $q^{t+1}(\mathbf{z}|\mathbf{x}) = p(\mathbf{z}|\mathbf{x}, \theta^t)$ yields that maximum.

Structure learning Next to the parameters of the BN, also its structure can be learned from a data set. This is called *structure learning*. There are two types of structure learning: *constraint-based learning* and *score-based learning*. Constraint-based learning tries to find the optimal network structure based on a set of independence constraints. These independence constraints can be learned from data, but that is a statistically difficult task. In score-based learning, a scoring function is defined that consists of one or more search criteria that assign a value to each network structure. A searching algorithm, such as hill-climbing or simulated annealing, is used to find the maximum value of this scoring function.

A network structure does not have to be learned from ground up, but can be derived from expert knowledge. After that, a structure learning technique can be used to optimize the derived network structure. Structure learning is not in the scope of this thesis and interested readers are directed to [Hec98, KF05].

3.2 Dynamic Bayesian networks

A BN is useful for problem domains where the state of the world is static. In such a world, every variable has a single and fixed value. Unfortunately, this assumption of a static world is not always sufficient, as we have seen with the Navy BN in the introduction of this chapter. A *dynamic Bayesian network* (DBN), which is a BN extended with a time dimension, can be used to model dynamic systems [DK88]. In this section we only describe the DBN formalism that is in common use today. Our extension of the formalism and its application is described in parts II and III of this thesis. A more extensive introduction on DBNs can be found in: [Jor03, Mur02].

3.2.1 Network structure

First of all, the dynamic extension does not mean that the network structure or parameters changes dynamically, but that a dynamic system is modeled. A DBN is a directed, a-cyclic graphical model of a stochastic process. It consists of time-slices (or time-steps), with each time-slice containing its own variables. A DBN is defined as the pair (B_1, B_{\rightarrow}) where B_1 is a BN

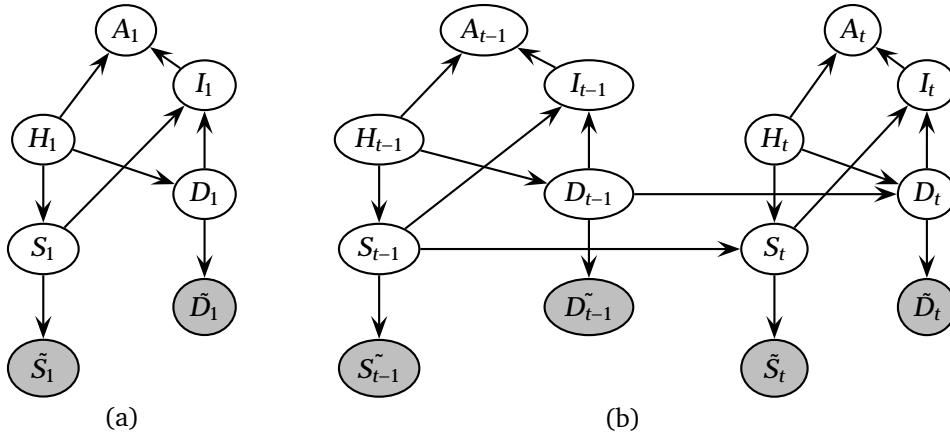


Figure 3.11: (a) The initial network for the Navy DBN, (B_1) . (b) The 2TBN for the Navy DBN, (B_{\rightarrow}) .

that defines the prior or initial state distribution of the state variables $p(\mathbf{Z}_1)$ [Mur02]. Typically, $\mathbf{Z}_t = (\mathbf{U}_t, \mathbf{X}_t, \mathbf{Y}_t)$ represents the input, hidden and output variables of the model. B_{\rightarrow} is a two-slice temporal Bayesian network (2TBN) that defines the transition model $p(\mathbf{Z}_t | \mathbf{Z}_{t-1})$ as follows:

$$p(\mathbf{Z}_t | \mathbf{Z}_{t-1}) = \prod_{i=1}^N p(Z_t^i | \mathbf{Pa}(Z_t^i)) , \quad (3.35)$$

where Z_t^i is the i -th node at time t and could be a component of \mathbf{X}_t , \mathbf{Y}_t or \mathbf{U}_t . $\mathbf{Pa}(Z_t^i)$ are the parents of Z_t^i , which can be in the same or the previous time-slice (in this case, the model is restricted to first-order Markov models). The nodes in the first slice of the 2TBN network do not have parameters associated with them. The nodes in the second slice do have a CPT. The structure repeats and the process is stationary, so the parameters for the slices $t = 2, 3, \dots$ remain the same. This means that the model can be fully described by only giving the first two slices. In this way, an unbounded sequence length can be modeled using a finite number of parameters. The joint probability distribution for a sequence of length T can be obtained by *unrolling* the 2TBN network:

$$p(\mathbf{Z}_{1:T}) = \prod_{t=1}^T \prod_{i=1}^N p(Z_t^i | \mathbf{Pa}(Z_t^i)) . \quad (3.36)$$

We have already seen the Navy DBN in the introduction of this chapter in figure 3.2. The (B_1, B_{\rightarrow}) definition for this DBN is shown in figure 3.11.

3.2.2 Inference

There exist several ways of performing inference on DBNs. The most common types of inference are given in figure 3.12 and discussed next.

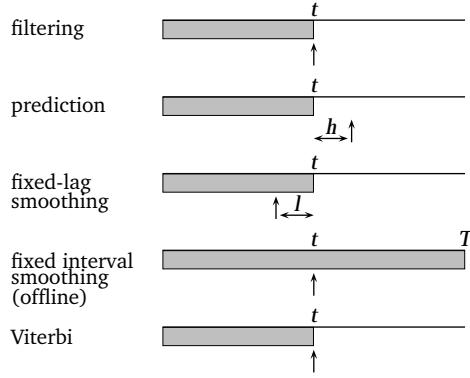


Figure 3.12: The main types of inference. The shaded region is the interval that contains evidence. The arrow represents the time of inference. t is the current time, and T is the sequence length.

- With *filtering* (or *monitoring*), the current belief state is computed given all evidence from the past. To achieve this, $p(\mathbf{X}_t|\mathbf{y}_{1:t})$ needs to be calculated. Filtering is used to keep track of the current state for making rational decisions. It is called *filtering*, because noise is filtered out from the observations.
- With *prediction*, a future belief state is computed given all evidence from the past. This means $p(\mathbf{X}_{t+h}|\mathbf{y}_{1:t})$ needs to be calculated for some $h > 0$. Prediction can be used to evaluate the effect of possible actions on the future state.
- With *smoothing* (or *hind sight*), a belief state in the past is calculated given all evidence up to the present. So, $p(\mathbf{X}_{t-l}|\mathbf{y}_{1:t})$ is calculated for some fixed time-lag $l > 0$. Smoothing is useful to get a better estimate of the past state, because more evidence is available at time t than at time $t - l$. In figure 3.12, two types of smoothing are given: *fixed-lag smoothing*, which is the type of smoothing given above, and *fixed-interval smoothing*, which is the offline case in which $p(\mathbf{X}_t|\mathbf{y}_{1:T})$ is calculated for all $1 < t < T$.
- The final inference method to be described here is *Viterbi decoding* (or *most likely explanation*). This is a different kind of inference, but it is used very often nonetheless. With Viterbi decoding, one wants to compute the most likely sequence of hidden states given a sequence of observations. That is: $\arg \max_{\mathbf{x}_{1:t}} p(\mathbf{x}_{1:t}|\mathbf{y}_{1:t})$ needs to be calculated.

Summarized, the goal of inference in a DBN is to calculate $p(\mathbf{X}_t|\mathbf{y}_{1:\tau})$, where with filtering $\tau = t$, with smoothing $\tau > t$ and with prediction $\tau < t$. Several exact and approximate inference algorithms exist that will be shortly discussed next.

Exact inference The first approach for exact inference in DBNs is based on the notion that an unrolled DBN is in fact the same as a static BN. *Variable elimination* can be used with filtering. This algorithm results in having only two slices of the unrolled DBN in memory at a time. For smoothing, using the *junction tree algorithm* is more efficient.

A second approach is to convert the DBN to a HMM and then apply the *forward-backward algorithm*. Converting a DBN to a HMM is only possible with discrete state DBNs. With N_h hidden variables per slice, each variable having up to M values, the resulting HMM will have maximally $S = M^{N_h}$ values. As long as S is not too large, this is a nice method, because it is

very easy to implement. Unfortunately, most of the times S will be too large and more efficient methods need to be used.

The *frontier algorithm* [Zwe96] is based on the notion that in the forward-backward algorithm, X_t of the HMM d-separates the past from the future. This can be generalized to a DBN by noticing that the hidden nodes in a slice d-separate past from future. This set of nodes is called *the frontier*. This algorithm also uses a forward and a backward pass.

The *interface algorithm* [Mur02] is an optimization of the frontier algorithm because it does not use all hidden nodes in a slice, but only the subset of nodes with outgoing arcs to d-separate the past from the future. This subset is called the forward *interface*. The modified 2TBN is called a *1.5DBN* H_t in this algorithm, because it contains all the nodes from slice 2, but only the nodes with an outgoing arc from slice 1. After this modification, a junction tree is created of each H_t and they are glued together. Finally, inference is performed on each separate tree and messages are passed between them via the interface nodes, first forward and then backward.

Finally, analogous to the conversion from discrete state-space DBN to HMMs, a linear-Gaussian state-space DBN can be converted to a Kalman filter model (KFM). *Kalman filtering* or *Kalman smoothing* can be used for exact inference after this conversion.

Approximate inference Exact inference in discrete-state models is often unacceptably slow. When faster results are needed, approximate inference can be used. Furthermore, it turns out that for most DBNs with continuous or mixed discrete-continuous hidden nodes, exact representations of the belief state do not exist. Generally, two types of approximations are distinguished: *deterministic* and *stochastic* algorithms.

Deterministic algorithms for the discrete-state DBNs include: the *Boyen-Koller algorithm* (BK) [BK98], where the interface distribution is approximated as a product of marginals and the marginals are exactly updated using the junction tree algorithm; the *factored frontier algorithm* (FF) [MW01], where the frontier distribution is approximated as a product of marginals and the marginals are computed directly; and *loopy belief propagation*, which is a generalization of BK and FF.

Deterministic algorithms for linear-Gaussian DBNs include: the *Generalized Pseudo Bayesian* approximation (GPB(n)), where the belief state is always represented using K^{n-1} Gaussians; and the *interacting multiple models* (IMM) approximation, which also collapses the priors.

Deterministic algorithms for mixed discrete-continuous DBNs include: *Viterbi approximation*, in which the discrete values are enumerated in a priori order of probability; *expectation propagation*, which is the generalization of BK for discrete-state DBNs and GPB for linear Gaussian DBNs; and *variational methods*, that decompose the problem into separate discrete and continuous chains which are treated differently.

Deterministic algorithms for non-linear/non-Gaussian models include: the *extended Kalman filter* (EKF) or the *unscented Kalman filter* (UKF) for non-linear models, which calculate a local linearization and then apply a Kalman filter (KF).

Stochastic algorithms can be divided in offline and online methods. Offline methods are often based on *importance sampling* or *Monte Carlo Markov Chain*. Online methods often rely on methods such as *particle filtering*, *sequential Monte Carlo*, the *bootstrap filter*, the *condensation algorithm*, *survival of the fittest*, and others.

Stochastic algorithms have the advantage over deterministic algorithms that they are easier to implement and that they are able to handle arbitrary models. Unfortunately, this comes with a price, because stochastic algorithms are generally slower than the deterministic methods. Both methods can be combined to get the best of two worlds. [Mur02] presents the *Rao-Blackwellised Particle Filtering algorithm*, in which some of the variables are integrated out using exact inference and sampling is applied to the other variables.

3.2.3 Learning

The techniques for learning DBNs are generally the same as the techniques for learning static BNs. The specific methodology used in this research for learning the parameters of a DBN with complete data is presented in chapter 9. The main differences between learning of static and DBNs are discussed below.

Parameter learning can be done both online and offline. For online learning, the parameters are added to the state space after which online inference is applied. For offline learning, the same techniques as for learning BNs can be used. Some points that are specifically applicable to DBNs:

- Parameters must be tied across time-slices, so models of unbounded length can be modeled.
- In the case that the initial parameters π represent the stationary distribution of the system, they become coupled with the transition model. As such, they cannot be estimated independently of the transition matrix [Nik98].
- Linear-Gaussian DBNs sometimes have closed-form solutions to the maximum likelihood estimates.

Structural learning of DBNs consists of learning both inter-slice and intra-slice connections. If only the inter-slice connections are learned, the problem reduces to feature selection (what variables are important for temporal reasoning and what variables are not). Again, learning for static BNs can be adapted.

3.3 Other temporal reasoning techniques

The DBN formalism is not the first development in temporal reasoning under uncertainty. The two most popular techniques still in use nowadays are the hidden Markov model (HMM) and the Kalman filter model (KFM). Their popularity is mostly due to their compact representation, fast learning and fast inference techniques. However, a DBN can have some significant advantages over these two formalisms. For one, HMMs and KFMs are really limited in their *expressive power*. In fact, it is not even correct to call HMMs and KFMs *other techniques*, because the DBN formalism can be seen as a generalization of both HMMs and KFMs. The DBN formalism brings out connections between these models that had previously been considered quite different, because they were developed in very different research areas.

DBNs generalize HMMs by allowing the state space to be represented in factored form, instead of as a single discrete random variable. DBNs generalize KFMs by allowing arbitrary probability distributions, not just conditional linear Gaussian distributions. The basic versions HMMs and KFMs are briefly discussed here because of historical relevance. Extensions exist for both HMMs and KFMs to counter some of their issues, but an extensive description of those is beyond the scope of this thesis.

3.3.1 Hidden Markov models

A HMM models a first-order Markov process where the observation state is a probabilistic function of an underlying stochastic process that produces the sequence of observations. The underlying stochastic process cannot be observed directly, it is *hidden*. Both the hidden and observation states are modeled by discrete random variables [Rab89].

The HMM formalism first appeared in several statistical papers in the mid 1960s, but it took more than 10 years before its usefulness was recognized. Initially, the use of HMMs was a great success, especially in the fields of automatic speech recognition (ASR) and bio-sequence

analysis. Because of its success, the use of HMMs in ASR is still dominant nowadays, despite its undeniable issues [RB03].

A HMM definition consists of three parts: an initial state distribution π_0 , an observation model $p(Y_t|X_t)$ and a transition model $p(X_t|X_{t-1})$. HMMs are generally used to solve one of the following problems: (1) state estimation $p(X_t = x|y_{1:t})$ using the *forward-backward* algorithm, (2) most likely explanation $\arg \max_{x_{1:t}} p(x_{1:t}|y_{1:t})$ using the *Viterbi* algorithm, and/or (3) maximum likelihood HMM $\arg \max_{\lambda} p(y_{1:t}|\lambda)$ (for learning the HMM) using the *EM* algorithm, called the *Baum-Welch* algorithm in this context.

Figure 3.13 shows a HMM modeled as a DBN. In this figure, the HMM is not shown as the usual state transition diagram, but every outcome of \mathbf{x}_t indicates the occupied state at time t and \mathbf{y}_t represents the observation vector. In this representation, time is made explicit, which means that the arcs pointing from X_{t-1} to X_t represent causal influences instead of state transitions.

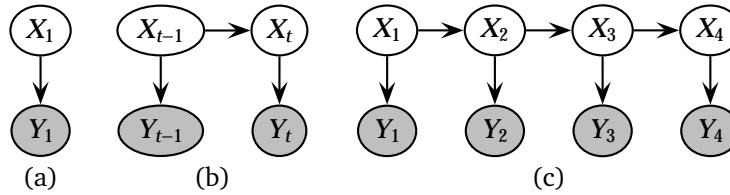


Figure 3.13: (a) The initial network for a HMM, (B_1). (b) The 2TBN for a HMM, (B_{\rightarrow}). (c) The model unrolled for $T = 4$ slices.

Being a HMM, the sequence of observations (\mathbf{Y}_t) depend on discrete hidden states (\mathbf{X}_t). The sequence of this hidden state is defined by a Markov process. This results in the following assignment of \mathbf{Z}_t :

- $Z_t^1 = X_t$,
- $Z_t^2 = Y_t$,
- $\mathbf{Pa}(Z_t^1) = \mathbf{Pa}(X_t) = X_{t-1}$, and
- $\mathbf{Pa}(Z_t^2) = \mathbf{Pa}(Y_t) = X_t$.

Using this assignment in equation 3.36, it becomes:

$$p(\mathbf{X}_{1:T}, \mathbf{Y}_{1:T}) = p(X_1) \cdot p(Y_1|X_1) \prod_{t=2}^T p(X_t|X_{t-1}) \cdot p(Y_t|X_t), \quad (3.37)$$

where you can clearly see the that the equation consists of three parts: the initial state distribution π_0 : $p(X_1) \cdot p(Y_1|X_1)$, the transition model: $p(X_t|X_{t-1})$, and the observation model: $p(Y_t|X_t)$.

One of the main problems of HMMs is the fact that the hidden state is represented by only a single discrete random variable. DBNs are able to decompose the state of a complex system into its constituent variables, taking advantage of the sparseness in the temporal probability model. This can result in exponentially fewer parameters. The effect is that using a DBN can lead to fewer space requirements for the model, less expensive inference and easier learning.

3.3.2 Kalman filter models

Basically, a KFM is a HMM with conditional linear Gaussian distributions [WB04]. It is generally used to solve uncertainty in linear dynamic systems. The KFM formalism first appeared in papers in the 1960s [Kal60], and was successfully used for the first time in NASA's Apollo program. Nowadays, it is still used in a wide range of applications. The KFM formalism assumes the dynamic system is jointly Gaussian. This means the belief state must be unimodal, which is inappropriate for many problems. The main advantage of using a DBN over a KFM is that the DBN can use arbitrary probability distributions instead of a single multivariate Gaussian distribution.

3.4 Summary

This chapter presented the theoretical foundation of temporal reasoning on which the remainder of this thesis relies. It presented a global overview of (D)BN theory and techniques that is vital for a general understanding of the rest of this thesis, such as d-separation, inference, and learning. Furthermore, a short comparison of DBNs to other temporal reasoning techniques was given, and in-depth reference material on different aspects of temporal reasoning was presented. As such, this chapter can serve as a good starting point for readers that are trying to get familiar with (D)BN theory and techniques.

In part II of this thesis, the DBN formalism as presented in this chapter will be extended, implemented and tested, because the existing formalism proved too restrictive for our modeling needs. Furthermore, a practical methodology for parameter learning of the extended DBN formalism will be presented. In part III of this thesis, the extended DBN formalism will be applied to two physiological processes: glucose-insulin regulation and the cardiovascular system.

Chapter 4

Current software

In this chapter we present a general overview of the software developed at the Decision Systems Laboratory (DSL), which did not have support for temporal reasoning before. After that, libraries and modeling tools that did have support for temporal reasoning are discussed.

The software with DBN functionality that is currently on the market can be divided into two classes: software that does have a GUI and software that does not have a GUI. The software that does have a GUI is mostly commercial and is relatively easy to use for a user with only an average knowledge of BNs. The software without a GUI is mostly academic-oriented, which means that it is a result of research in the area of (D)BNs. Academic-oriented software is very flexible, but this can also make its application to specific problems very time-consuming.

In this chapter only the software that could be investigated directly is covered¹. Investigating this software gives insight on which features are useful, which are currently offered and which are currently missing.

4.1 Decision Systems Laboratory software

SMILE and GeNIE are members of the family of software solutions developed at the DSL of the University of Pittsburgh. This software is a direct result of the research interests of the DSL and is developed for the purpose of probabilistic modeling, including special extensions for diagnostic inference. The software, the documentation, and many example applications can be freely downloaded from <http://genie.sis.pitt.edu>.

¹This means that software such as dHugin [Kjæ95] is not covered, since it is not possible to obtain a free, downloadable (test)version.

4.1.1 SMILE

The Structural Modeling, Inference, and Learning Engine is a fully platform independent library of C++ classes that implements graphical probabilistic and decision-theoretic models and is suitable for direct inclusion in intelligent systems [Dru99]. The individual classes defined in the Application Program Interface (API) of SMILE enable the user to create, edit, save and load graphical models, and use them for probabilistic reasoning and decision making under uncertainty. To be able to access the SMILE library from a number of other programming languages, wrappers exist for ActiveX, Java, and .NET. SMILE was first released in 1997 and has been thoroughly used in the field since. Its main features are:

- Platform independent; versions available for Windows, Unix (Solaris), Linux, Mac, PocketPC, etc.
- Wrapper available for use with .NET framework. Compatible with all .NET languages. May be used to create web-based applications of BNs.
- Thorough and complete documentation.
- Robust and running successfully in the field since 1997.
- Responsive development team support.

4.1.2 GeNIe

The Graphical Network Interface is the graphical user interface to the SMILE library. It is implemented in C++ and makes heavy use of the Microsoft Foundation Classes. Its emphasis is on accessibility and friendliness of the user interface, making creation of decision theoretic models intuitive using a graphical click-and-drop interface approach. It is a versatile and user-friendly environment for building graphical decision models and performing diagnosis. Its primary features are:

- Graphical editor to create and modify models.
- Supports nodes with General, Noisy-OR/MAX and Noisy-AND/MIN probability distributions.
- Functionality to cut and paste parts from/to different BNs.
- Complete integration with Microsoft Excel, cut and paste data into internal spreadsheet view of GeNIe.
- Cross compatibility with other software. Supports all major file types (e.g. Hugin, Netica, Ergo).
- Support for handling observation costs of nodes.
- Support for diagnostic case management.

Nowadays, the combination of SMILE and GeNIe has several thousand users worldwide. Applications based on the software range from heavy-duty industrial software to academic research projects [Dru05]. Some examples are: battle damage assessment, group decision support models for regional conflict detection, intelligent tutoring systems, medical diagnosis and therapy planning, diagnosis of diesel locomotives, airplanes, and production processes of integrated circuits. SMILE and GeNIe are also used for teaching statistics and decision-theoretic methods at several universities.

4.2 DBN libraries

Three libraries exist that have DBN functionality and are freely downloadable. All three have a work-in-progress status, which means that much functionality is missing and/or has to be tested and optimized.

4.2.1 The Bayes net toolbox for Matlab

Until the introduction of the Bayes net toolbox for Matlab (BNT) [Mur01], the field lacked a free general purpose software library that was able to handle many different variants of graphical models, inference and learning techniques. The BNT is an attempt to build such a free, open-source, and easy-to-extend library that can be used for research purposes.

The author chose to implement the library in Matlab because of the ease with which it can handle Gaussian random variables. Matlab has various advantages and disadvantages, the main disadvantage being that it is terribly slow.

In the BNT, BNs are represented as a structure containing the graph, the CPDs and a few other pieces of information. A DBN is represented similar to the representation in section 3.2 with a prior and a transitional network, making it only possible to model first-order processes. The BNT offers a variety of inference algorithms, each of which makes different trade offs between accuracy, generality, simplicity, speed, etc. All inference methods have the same API, so they can be easily interchanged. The conditional probabilities of the defined variables can be continuous or discrete. Parameter and structure learning are supported as well.

The toolbox was the first of its kind and set a standard for (D)BN libraries. It is still widely in use today, because the code is relatively easy to extend and well documented and the library has by far the most functionality of all software currently available. However, much functionality still needs to be added to make it a real general purpose tool, such as tree-structured CPDs, Bayesian modeling, online inference and learning, prediction, more approximate inference algorithms, and support for non-DAG graphical models. Also, the scripting part that is needed to implement a model in the BNT can be really cumbersome. Added to that, BNT does not have support for standard BN file formats, which cancels the possibility to export and/or import BN models from/to other packages. The BNT is released as open-source software and can be downloaded from <http://bnt.sourceforge.net>. A GUI is currently in development.

4.2.2 The graphical models toolkit

The graphical models toolkit (GMTK) [BZ02] is a freely-available and open-source toolkit written in C++ that is specialized in developing DBN-based automatic speech recognition (ASR) systems. The GMTK has a number of features that can be used for a large set of statistical models. These features include several inference techniques, continuous observation variables and discrete dependency specifications between discrete variables. The DBN model needs to be specified in a special purpose language. In this language, a DBN is specified as a template that contains several time-slices. The collection of time-slices is unrolled over time to create an unrolled DBN. The time-slices in the template are divided into a set of prologue, repeating and epilogue time-slices and only the repeating frames are copied over time. This approach to modeling DBNs has much expressive power, but it is also a lot of work to specify a DBN model.

The GMTK is a promising library. To become really useful, a couple of disadvantages need to be tackled. Its main disadvantages are that it is not a general purpose toolkit, because it specializes in ASR and it therefore lacks much functionality that is useful for other applications. Furthermore, the documentation is far from complete, making it difficult for a user that is not closely involved in the development of the software to understand the toolkit. Finally, although the author of the toolkit states that it is open-source, the website (<http://ssli.ee.washington.edu/~bilmes/gmtk>) still only offers a binary version.

4.2.3 The probabilistic network library

Intel's research lab in Saint Petersburg, Russia is responsible for the probabilistic network library (PNL) [Int04], which is an open-source library written in C++ that can be used for inference and learning using graphical models. PNL has support for a large variety of graphical models, including DBNs. In fact, PNL can be seen as the C++ implementation of BNT, since its design is

closely based on that library. PNL does not support all functionality provided by BNT yet, but the long-term goal is that it will surpass that functionality.

The influence of BNT is very clear when diving into the API of PNL. A DBN is also defined as a prior and a transitional network and the BN techniques are clearly based on [Mur02] as well. Because the approach is similar, PNL can also only model first-order processes. The API is very well documented, making it a good option if one wants to model DBNs (or other graphical models). However, the implementation is far from complete and still lacks much functionality that BNT offers. The library recently reached its v1.0 status, which can be downloaded from <http://www.intel.com/technology/computing/pnl>. Work is being done to make PNL compatible with the GeNIE GUI.

4.3 DBN modeling tools

Currently, there are only two modeling tools that both support temporal reasoning and have a GUI: BayesiaLAB and Netica. To illustrate the use of DBNs in these programs, the umbrella DBN that also serves as an illustrative example in [RN03] is implemented:

Example (Umbrella) Imagine a security guard at some secret underground installation. The guard wants to know whether it is raining today, but his only access to the outside world occurs each morning when he sees the director coming in, with or without, an umbrella. For each day t , the set of evidence contains a single variable Umbrella_t (observed an umbrella) and the set of unobservable variables contains Rain_t (whether it is raining). If it is raining today depends on if it rained the days before. The model is considered first-order Markov and stationary, so Rain_t depends only on Rain_{t-1} . The resulting initial network, 2TBN and CPTs are shown in figure 4.1

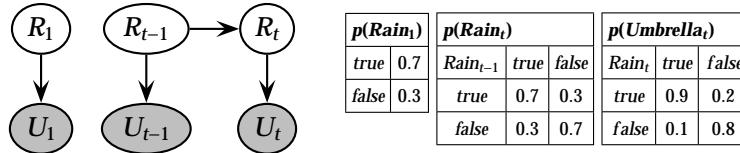


Figure 4.1: The umbrella DBN and its CPTs.

4.3.1 BayesiaLab

BayesiaLab is the BN software toolkit developed by the French company Bayesia (<http://www.bayesia.com>) [Bay04]. BayesiaLAB has two modi: a *modeling mode* for designing the BN (figure 4.2a) and a *validation mode* (figure 4.2b) for inference. In the modeling mode, it is possible to add temporal arcs to a BN to indicate that the parent node of the two nodes is in the previous time-slice. Only first-order Markov models can be designed this way. Temporal arcs have a different (red) color. The user needs to specify the initial state and the temporal probabilities of the nodes. After specification, the validation mode can be selected to follow the changes of the system over time by browsing between time steps (only forward steps are supported) or by setting the number of steps and then plotting a graph of the variables (figure 4.2c). The DBN does not unroll graphically, only the values change. It is possible to set evidence by hand or by importing a data file. Of the inference methods discussed in 3.2.2², only filtering and prediction are possible.

²filtering, prediction, online/offline smoothing and Viterbi

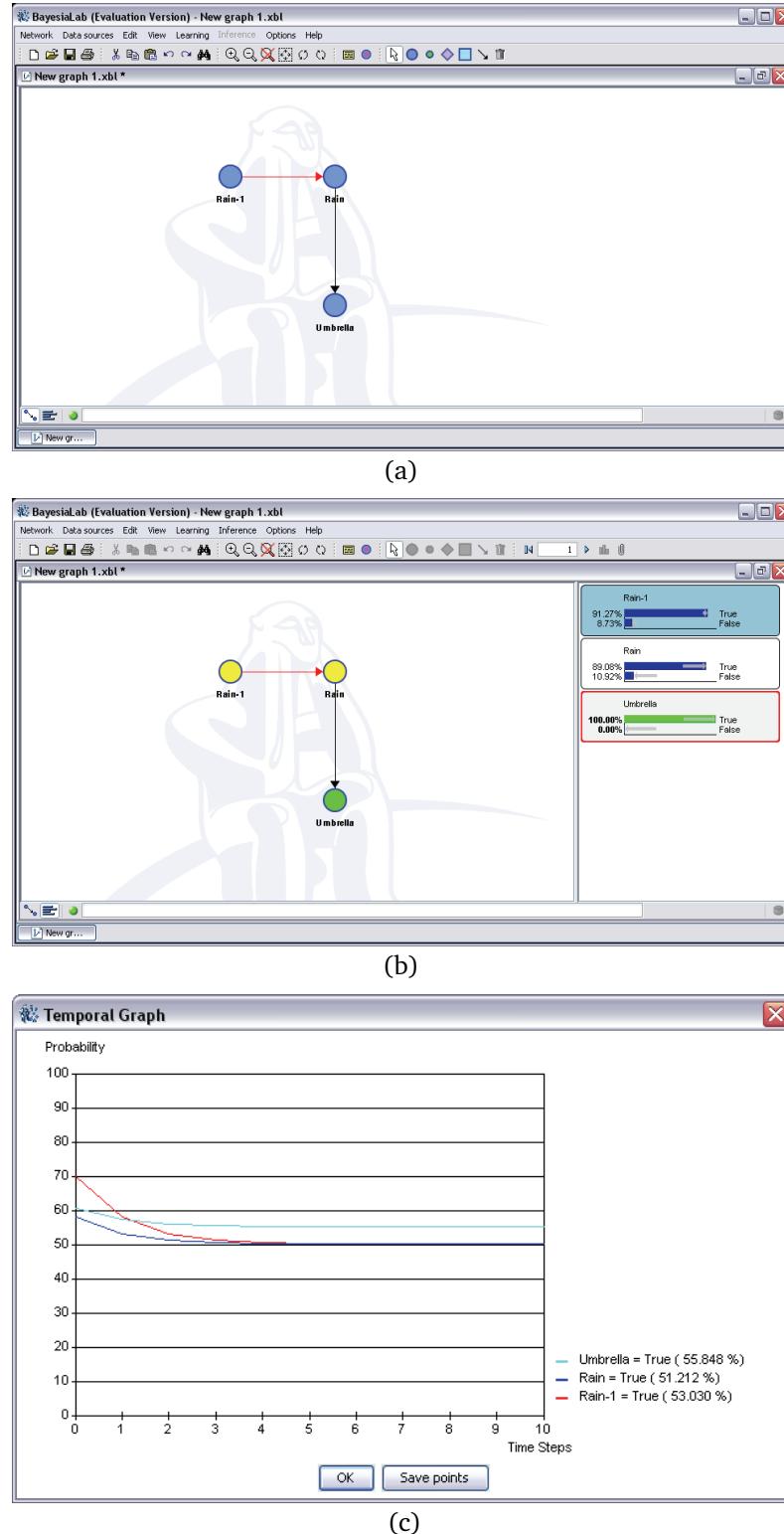


Figure 4.2: The GUI of BayesiaLAB: (a) The design mode where the user designs the (D)BN. (b) The validation mode where it is possible to perform inference on the (D)BN. (c) A time-graph of the variables. Since no evidence is set, the system goes to an equilibrium.

4.3.2 Netica

Netica is a product of Norsys (<http://www.norsys.com>) [Nor97], which is located in Canada. In Netica, the user designs the (D)BN and then compiles it, after which inference is possible. Compiling the (D)BN basically means that it is converted to a junction tree representation. When designing a DBN, temporal arcs can be added between nodes (figure 4.3). The temporal arcs have a red color. The difference with BayesiaLAB is that we explicitly define the temporal relationship (called a time-delay in Netica) of two nodes in one of the nodes instead of the implicit approach of BayesiaLAB. In this way, k -order Markov processes can be modeled. When the DBN has nodes with temporal relations to itself (which is usually the case), the DBN definition contains loops. Of course, this is only valid in the definition of the DBN; the unrolled version should not contain loops. After defining the DBN, it can be unrolled for t slices and compiled. The resulting BN is opened in a new window (figure 4.3). Inference can be performed on this unrolled and compiled BN. All inference methods discussed in 3.2.2³ are supported. We can enter evidence by hand or by importing a data file.

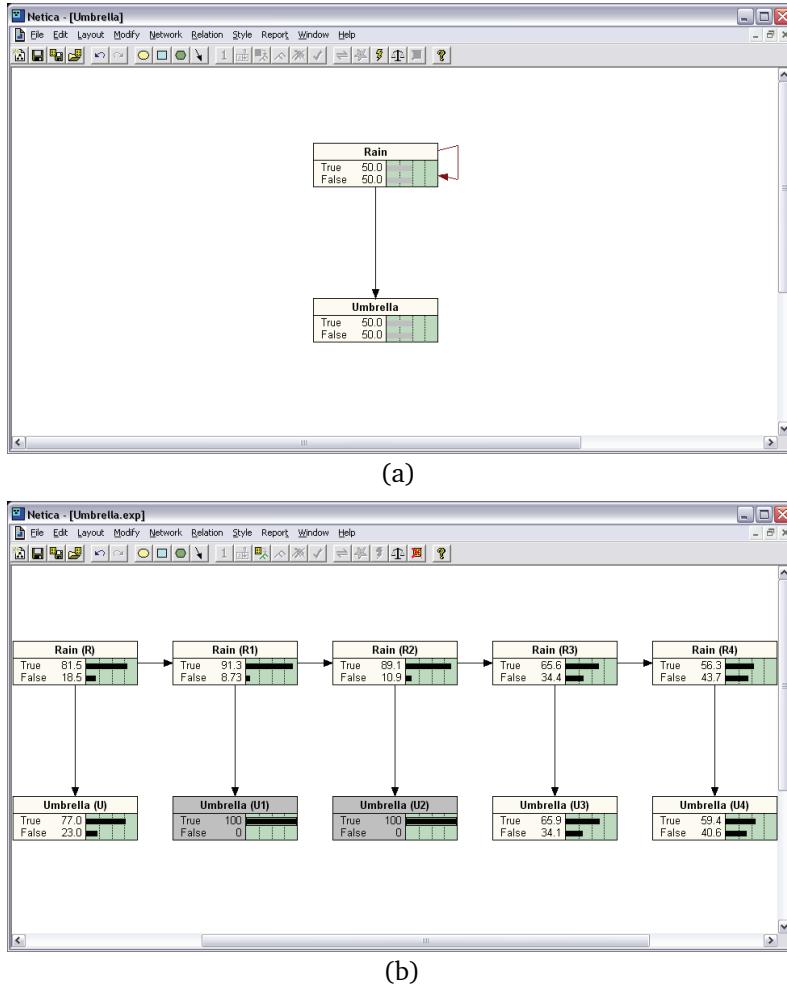


Figure 4.3: The GUI of Netica: (a) The design mode where the user designs the BN. (b) The unrolled and compiled (static) BN on which inference is possible.

³filtering, prediction, online/offline smoothing and Viterbi

4.4 Summary

After looking at and playing with the existing software, we can only conclude that there is no satisfying solution for modeling with DBNs yet. Cumbersome scripting to implement the DBN, conservative DBN definitions, lack of import/export functionality for BN models, and slow implementations are features that apply to one or more of the reviewed packages. The package with the most functionality is BNT, but unfortunately it is slow, lacks a GUI and has a very conservative definition of DBNs. However, the two modeling tools that do have a GUI are not satisfying either, since they also lack much functionality and are quite slow, making them only useful for toy problems.

The nice thing about SMILE and GeNIe is that it combines the best of two worlds: The SMILE API provides a researcher with all the flexibility and functionality he wants and the GeNIe GUI makes the core functionality relatively easy accessible to the public. However, SMILE and GeNIe need to be extended with temporal reasoning before it can be used for modeling with DBNs.

Extending SMILE and GeNIe with temporal reasoning functionality is not trivial. Since no standard approach of representing DBNs exists yet, the extension consists of two parts: designing a modeling approach, and implementing it in the software. The designed modeling approach will be discussed in chapter 7. The design and implementation of temporal reasoning in SMILE and GeNIe will be discussed in 8.

Chapter

5

Related work

BNs are very popular in medical applications where they have been used for over a decade now. This popularity is mostly due to the property of BNs that makes it possible to combine expert knowledge and gathered data for handling the uncertain knowledge involved in establishing diagnoses, selecting optimal treatment and predicting treatment outcomes. Traditionally, medicine is a field that relies heavily on expert knowledge and generates much data. Combining these can provide a great help for physicians.

5.1 Problem solving in medical applications

According to [LvdGAH04], there are four main applications of BNs in biomedicine and health-care support. Three of these applications consider the important stages of patient treatment requiring decision support. The fourth is a more research based application:

1. **Diagnostic reasoning** Diagnosing a patient comes down to constructing a hypothesis about the disease based on observations or symptoms of the patient. However, these observations generally do not unambiguously reveal the condition of the patient. To avoid errors in diagnosing the patient, this uncertainty needs to be taken into account. BNs offer a natural formalism for this type of reasoning, because BNs present the physician with a probability distribution over the diseases that can be present, and provides information on what test to perform to maximize the certainty of a correct diagnose. In fact, diagnostic reasoning was the first real-life medical application of BNs. Well-known examples are the *MUNIN* [AJA⁺89] and *Pathfinder* [HHN92] systems, both can be found in the network repository of <http://genie.sis.pitt.edu>.
2. **Prognostic reasoning** amounts to making a prediction about how the disease evolves over time. Knowledge of the process evolution over time needs to be exploited to be able

to make successful predictions. Although very useful, prognostic BNs are a rather new development in medicine. This is probably because there is little or no experience with integrating traditional ideas about disease evolution into BNs and the DBN formalism is a rather recent development.

3. **Treatment selection** The BN formalism does not provide means for making decisions, since only the joint distribution over the random variables is considered. In decision support systems, the BN formalism is often extended to include knowledge about decisions and preferences. An example of such an extension is the influence diagram formalism [HM84]. This formalism can help an expert to select the best treatment for a patient. When a treatment selection system reasons over time, this application class is also known as *therapy planning*.
4. **Discovering functional interactions** The above three applications account for problem solving with already constructed BNs. However, important new insights can be gained by constructing a BN, especially when learning both the structure and parameters from data. Currently, there is a growing interest in using BNs to unravel uncertain interactions among variables, such as unraveling molecular mechanisms at the cellular level by learning from biological time-series data. Again, the use of the DBN formalism is required.

5.2 Temporal reasoning

The incorporation of temporal reasoning helps to continue the success of decision support systems in medicine. To clarify its usefulness, let us look at the four main applications again: (1) During diagnostic reasoning, it can make a substantial difference to know the temporal order in which some symptoms occurred or for how long they lasted; (2) during prognostic reasoning, the potential evolutions of a disease are conceived as a description of events unfolding in time; (3) during treatment selection, the different steps of treatment must be applied in a precise order, with a given frequency and for a certain time-span in order to be effective [Aug05]; and (4) in discovering functional interactions, research often deals with noisy time-series data.

These phenomena can not be modeled in an ordinary BN, hence the DBN formalism comes into play. Despite the early adaption of BNs in medical applications, the incorporation of temporal reasoning started only a few years ago and practical applications are starting to emerge at this very moment. Some recent examples of applications with temporal reasoning using DBNs in the medical field are:

In [CvdGV⁺05], a DBN is designed for diagnosing ventilator-associated pneumonia in ICU patients with the help of domain experts. A DBN is constructed that explicitly captures the development of the disease through time and the diagnostic performance is compared to an earlier constructed BN of the same disease. The results show that the DBN formalism is suitable for this kind of application. The paper also discusses how probability elicitation from domain experts serves to quantify the dynamics involved and shows how the nature of patient data helps to reduce the computational burden of inference.

In [KCC00], a comparison is made between stationary and non-stationary temporal models that are designed for the prediction of ICU mortality. A process is stationary if its stochastic properties stay the same when it is shifted in a positive or negative direction by a constant time parameter, and is non-stationary otherwise. The DBNs are trained using data collected from 40 different ICU databases. Overall, the DBNs based on the stationary temporal models outperformed the non-stationary versions, except for one that was trained using a large dataset. The authors suggest that stationary and non-stationary models can be combined to get even better results.

[LdBSH00] proposes a framework for incorporating an expert system based on the DBN formalism into clinical information systems. The development of a probabilistic and decision-theoretic system is presented that can assist clinicians in diagnosing and treating patients with pneumonia in the ICU.

[TVLGH05] describes a DBN for modeling and classifying visual field deterioration and compares it to a number of more common classifiers. The authors recognize the potential of the recent explosion in the amount of data of patients who suffer from visual deterioration that is being stored, including field test data, retinal image data and patient demographic data and took this opportunity to investigate the usefulness of the DBN formalism in this area. Results show that the DBN classifier is comparable to existing models, but also gives insight of the underlying spatial and temporal relationships.

5.3 Summary

The presented papers show that in the last couple of years, some advances have been made in the application of the DBN formalism in medical systems. However, a lot can still be done to make the application of DBNs to medical systems more successful, especially in prognostic reasoning and treatment selection.

Temporal modeling of physiological processes present some new challenges. If a physician starts to monitor a patient at time $t = 0$, he will also take the disease history and other properties such as gender/age/etc. of the patient into account to be able to predict the outcome better. When modeling this kind of features using the DBN formalism, the modeler has to answer many questions, such as: (1) Is a DBN really needed, or is a BN sufficient? (2) Is it even possible to model the medical problem in a DBN? (3) What is the purpose of the DBN? Is the DBN going to be used for monitoring and diagnosis, or does it need input variables for prognostic reasoning and treatment selection? (4) What variables are important for the physician and for the performance of inference? (6) How to model non-numeric properties such as nominal, ordinal, and interval variables? (7) What data is stored in available patient databases, what is the quality of the data and how useful are the stored variables?

Given the fact that nowadays more and more patient data is stored creates better opportunities for DBNs to make a difference in the medical field, but for most of the above questions there is no general answer. However, insight can be gained by reading how these questions are tackled in this research, starting with our extension of the DBN formalism in chapter 7 and finishing with two applications and accompanying empirical studies in chapters 10 and 11.

Part II

Model and implementation

“Do you like our owl?”
“It’s artificial?”
“Of course it is.”
- Blade Runner.

Chapter 6

Research architecture

This part of the thesis is about our extension of the DBN theory and the design and implementation of different research tools that constitute the framework for temporal reasoning in modeling physiological processes. In short, in the chapters in this part of the thesis we discuss the subsystems that form the research architecture. In this chapter we present a short overview of these subsystems and how they relate to each other. Before we discuss the research architecture, let us go back to the original assignment introduced in chapter 1.

Although creating models for temporal reasoning with DBNs sounds promising at first sight, many questions remain to be answered. For instance: *How do we obtain the structure and parameters? What is the performance of inference? How much patient data do we need for learning the parameters? If possible, how much does a DBN outperform a BN when applied to the same problem?* These questions form the motivation for the research presented in this thesis. The assignment consisted of three main parts:

1. A *theoretical* part that consisted of (1) investigating the existing DBN theory, (2) extending the current DBN formalism, (3) adapting DBN parameter learning, and (4) validating the extensions.
2. An *implementation* part that consisted of (5) investigating current DBN software, (6) designing and implementing DBN functionality in SMILE, and (7) designing an intuitive graphical user interface to DBNs in GeNIe.
3. An *application* part that consisted of (8) applying the extended DBN formalism to physiological processes in the human body, and (9) performing empirical studies on the performance.

Most of the above-mentioned items not only form the basis of our research on temporal reasoning, but can be seen as a framework for temporal reasoning in the medical domain, where the following questions generally need to be answered: (1) Is a DBN really needed, or is a BN

sufficient? (2) Is it even possible to model the medical problem in a DBN? (3) What is the purpose of the DBN? Is the DBN going to be used for monitoring and diagnosis, or does it need input variables for prognostic reasoning and treatment selection? (4) What variables are important for the physician and for the performance of inference? (6) How to model non-numeric properties such as nominal, ordinal, and interval variables? (7) What data is stored in available patient databases, what is the quality of the data and how useful are the stored variables?

Figure 6.1 shows how the different subsystems derived from the assignment are related. Each independent subsystem is briefly discussed below.

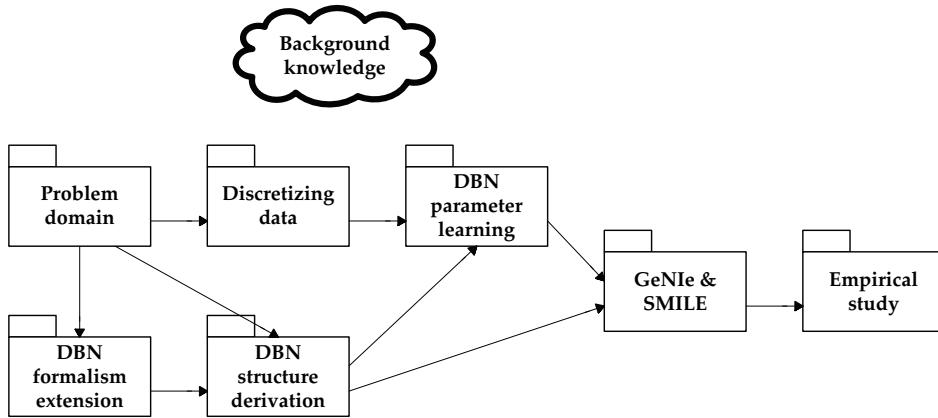


Figure 6.1: Research architecture.

Background knowledge It is not possible to do good research if one does not have a thorough understanding of the field. Background knowledge forms the basis of the research and is connected to every subsystem in the research architecture. It includes knowledge about the medical domain, Bayesian network theory, existing software tools and related work. All these elements are discussed in part I of this thesis.

Problem domain We are going to model the human physiological processes by means of a DBN. This subsystem contains the aspects of the problem that we are going to model, in our case both the glucose-insulin regulation and the cardiovascular system discussed in chapter 2. In this part of the thesis the problem domain will not be made explicit, because the presented research architecture holds for every problem domain. In part III, we will elaborate on how to apply this research architecture to our two physiological processes, where we will simulate patients in an ICU situation by means of systems of ordinary differential equations. The parameters of the systems of ODEs are varied for different patients and for different ICU situations. An ideal situation would be gathering real patient data from a database, but we do not have such a database at our disposal.

DBN formalism extension The definition of a DBN given in [Mur02] is very conservative and does not provide enough modeling power when modeling human physiological processes. The first step was to extend the DBN theory to be able to model the human physiological processes in a concise and efficient manner. The DBN theory extension, including general experimentation on the performance is discussed in chapter 7. In general, one should investigate if the DBN formalism provides enough modeling power for the domain to be modeled and extend it when appropriate.

DBN structure derivation A DBN consists of two parts: a qualitative part that defines the structure of the DBN and a quantitative part that defines the conditional probability distributions of the DBN. We have seen in chapter 3 that we can obtain the DBN structure by machine-learning techniques or by expert elicitation, e.g. interviewing a physician.

In our case, the obtained DBN structures are a combination of expert knowledge from physicians and our knowledge about modeling with DBNs. The obtained DBN models are a close enough approximation of the real world, without being computationally too expensive. We elaborate more on derivation of the DBN structure in general in chapter 9.

DBN parameter learning The quantitative part of the DBN consists of its conditional probability distributions. Again, this can be obtained by machine-learning techniques or by expert elicitation. Because many parameters are involved, obtaining them by means of expert elicitation is infeasible. We will devise a DBN parameter learning method for the extended DBN formalism in chapter 9. This method can be used to learn the DBN parameters from a patient database and an existing BN learning algorithm, i.e. the EM algorithm. A special-purpose C++ tool was written on top of SMILE to perform parameter learning on the glucose-insulin regulation DBN and the cardiovascular system DBN.

Discretizing data Because continuous variables in a DBN is problematic due to complexity issues, the patient data must be discretized. Currently, discretization of continuous time-series data is often performed as an unsupervised preprocessing step. The choice of the method and accompanying parameters are rarely justified [MU05]. However, for field experts it is of great importance that the resulting interval boundaries are meaningful within the domain, and that is why we will discuss some discretization methods and their considerations in chapter 9. Matlab was used to implement and test these discretization methods. We choose Matlab for this purpose, because it provides many mathematical and graphical representation possibilities. These tools will not be released to the public, but similar functionality is available in the current version of SMILE and GeNIE.

GeNIE & SMILE The DSL software needed to be extended with temporal reasoning. For our research, extending SMILE with temporal reasoning was most important, because this functionality was needed for the empirical studies on the glucose-insulin regulation and the cardiovascular system DBNs. GeNIE was extended with a temporal reasoning GUI to provide its users with the added temporal reasoning functionality of SMILE. The temporal extension of both GeNIE and SMILE is presented in chapter 8 and will be released to the public.

Empirical studies The above subsystems are needed to perform empirical studies on modeling the problem domain with a DBN. Special-purpose tools where written in both Matlab and C++ to perform several experiments on the glucose-insulin regulation and the cardiovascular system DBNs. The application of the extended DBN formalism, the experiments and the results are discussed in part III of this thesis.

Chapter 7

DBN formalism extensions

Before implementing the DBN formalism, we need to ask ourselves the question what aspects of a dynamic system we want to be able to model. The DBN theory discussed in section 3.2 is largely based on work done in [Mur02]. This work is currently viewed as *the* standard in DBN theory, but it employs a rather limited definition of a DBN. We decided to extend the formalism to provide us with more modeling power than Murphy's definition allows, and to improve the performance in terms of execution time and memory usage. Extending the DBN formalism enables us to deal with complex dynamic systems in an intuitive way.

Before we extend the DBN formalism, let us review the original definition given in [Mur02]. In the original definition, a DBN is defined as the pair (B_1, B_{\rightarrow}) where B_1 is a BN that defines the prior or initial state distribution of the state variables $p(\mathbf{Z}_1)$. B_{\rightarrow} is a two-slice temporal Bayesian network (2TBN) that defines the transition model $p(\mathbf{Z}_t|\mathbf{Z}_{t-1})$. Only the nodes in the second time-slice of the 2TBN have a conditional probability distribution.

If we take a closer look at this definition, we notice a couple of things. The most obvious one is that it is only possible to model first-order Markov processes. Another one is that when unrolling the network for inference, every node is copied to every time-slice, even if it has a constant value for all time-slices. Finally, although it is possible to introduce a different initial state using the B_1 part of the definition, it is not possible to define a different ending state, which can be useful for modeling variables that are only interesting after the end of the process. These three observations form the basis of our extension of the DBN formalism.

7.1 k^{th} -order Markov processes

Murphy's definition of DBNs is sufficient for modeling first-order Markov processes. However, as we want to be as expressive in modeling temporal processes as possible, we also want to

be able to model k^{th} -order Markov processes¹. This extension can be useful when modeling physiological processes at a small time-scale, for example when devising a DBN model that catches the dynamics of the human cardiovascular system for every 50[ms]. In such temporal models, assuming a first-order process is not sufficient.

We have extended Murphy's definition to the following: B_{\rightarrow} is not defined as a 2TBN, but as a $(k+1)$ TBN that defines the transition model $p(\mathbf{Z}_t|\mathbf{Z}_{t-1}, \mathbf{Z}_{t-2}, \dots, \mathbf{Z}_{t-k})$ for a k^{th} -order Markov process.

$$p(\mathbf{Z}_t|\mathbf{Z}_{t-1}, \mathbf{Z}_{t-2}, \dots, \mathbf{Z}_{t-k}) = \prod_{i=1}^N p(Z_t^i | \mathbf{Pa}(Z_t^i)). \quad (7.1)$$

Not surprisingly, equation 7.1 is essentially the same as equation 3.35 in section 3.2, only the set of parents is not restricted to nodes in the previous or current time-slice, but can also contain nodes in time-slices further in the past. In this definition, the joint distribution for a sequence of length T can be obtained by unrolling the $(k+1)$ TBN and then multiplying all the CPTs, again this equation is similar to equation 3.36 in section 3.2.

$$p(\mathbf{Z}_{1:T}) = \prod_{t=1}^T \prod_{i=1}^N p(Z_t^i | \mathbf{Pa}(Z_t^i)). \quad (7.2)$$

Because graphically representing a k^{th} -order Markov process with a $(k+1)$ TBN can be really cumbersome, we have developed a different visualization of a DBN, where we introduce the concept of a *temporal arc*. This concept is inspired on earlier work in [AC96].

Definition (Temporal arc)

A *temporal arc* is an arc between a parent node and a child node with an index that denotes the temporal order or time-delay $k > 0$. A parent and a child node can be the same node.

An example second-order DBN according to the $(k+1)$ TBN visualization is shown in figure 7.1a. This example DBN is relatively simple, but imagine specifying a DBN model with many variables and different temporal orders per time-slice. With our visualization, complex temporal models can be visualized showing only one time-slice. An example of our visualization is shown in figure 7.1b. Even while we are dealing with a relatively simple DBN, it is immediately clear from our visualization that the DBN has three temporal arcs ($X_{t-2} \rightarrow X_t$, $X_{t-1} \rightarrow X_t$, and $Y_{t-1} \rightarrow Y_t$) and two non-temporal arcs ($U_t \rightarrow X_t$, $X_t \rightarrow Y_t$), where this is less obvious from the $(k+1)$ TBN visualization. Not only the visualization is more comprehensible, also the speci-

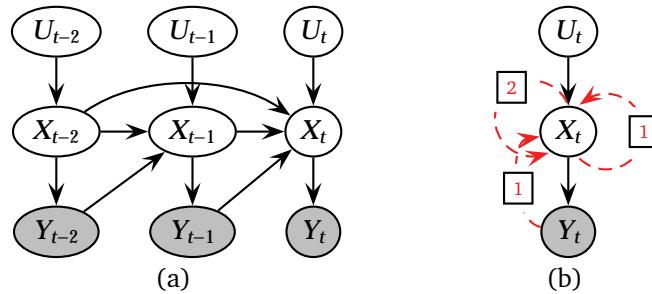


Figure 7.1: (a) The $(k+1)$ TBN visualization of a second-order DBN using $t = 3$ time-slices. (b) The same second-order DBN using our temporal arcs visualization. The initial network is defined implicitly in this example.

ification of a DBN is easier. For the example DBN in figure 7.1, specifying the $(k+1)$ TBN would

¹It is possible to simulate k^{th} -order dependencies by using first order Markov processes and introducing extra state variables in time-slice t that are directly copied from the previous time-slices, but that introduces many unnecessary extra variables.

mean specifying nine nodes and eleven arcs, where for our representation only three nodes and five arcs have to be specified, which results in significantly less code and most probably fewer errors. To see how big the difference is for the example DBN in figure 7.1, two simplified XML listings are shown in listing 7.1. Even with such a simple network, the specification for the 3TBN is already almost two times larger than the specification with temporal arcs.

```

1  <?xml version="1.0"?>
2  <smile version="1.0" id="temporalarcs">
3      <nodes>
4          <cpt id="U" dynamic="plate">
5              <state id="False" />
6              <state id="True" />
7              <probabilities>.</probabilities>
8          </cpt>
9          <cpt id="X" dynamic="plate">
10             <state id="False" />
11             <state id="True" />
12             <parents>U</parents>
13             <probabilities>.</probabilities>
14         </cpt>
15         <cpt id="Y" dynamic="plate">
16             <state id="False" />
17             <state id="True" />
18             <parents>X</parents>
19             <probabilities>.</probabilities>
20         </cpt>
21     </nodes>
22     <dynamic numslices="3">
23         <node id="X">
24             <parent id="X" order="1" />
25             <parent id="Y" order="1" />
26             <parent id="X" order="2" />
27             <cpt order="1">.</cpt>
28             <cpt order="2">.</cpt>
29         </node>
30     </dynamic>
31 </smile>
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56

```

```

<?xml version="1.0"?>
<smile version="1.0" id="3TBN">
    <nodes>
        <cpt id="U">
            <state id="False" />
            <state id="True" />
            <probabilities>.</probabilities>
        </cpt>
        <cpt id="X">
            <state id="False" />
            <state id="True" />
            <parents>U</parents>
            <probabilities>.</probabilities>
        </cpt>
        <cpt id="Y">
            <state id="False" />
            <state id="True" />
            <parents>X</parents>
            <probabilities>.</probabilities>
        </cpt>
        <cpt id="U_1">
            <state id="False" />
            <state id="True" />
            <probabilities>.</probabilities>
        </cpt>
        <cpt id="X_1">
            <state id="False" />
            <state id="True" />
            <parents>U_1 X Y</parents>
            <probabilities>.</probabilities>
        </cpt>
        <cpt id="Y_1">
            <state id="False" />
            <state id="True" />
            <parents>X_1</parents>
            <probabilities>.</probabilities>
        </cpt>
        <cpt id="U_2">
            <state id="False" />
            <state id="True" />
            <probabilities>.</probabilities>
        </cpt>
        <cpt id="X_2">
            <state id="False" />
            <state id="True" />
            <parents>U_2 X_1 Y_1 X</parents>
            <probabilities>.</probabilities>
        </cpt>
        <cpt id="Y_2">
            <state id="False" />
            <state id="True" />
            <parents>X_2</parents>
            <probabilities>.</probabilities>
        </cpt>
    </nodes>
</smile>

```

Listing 7.1: A simplified XML representation for the temporal arcs visualization (left) and the 3TBN (right) of the DBN shown in figure 7.1.

7.2 Temporal plate and contemporaneous nodes

The second problem of Murphy's definition is that every node is copied to every time-slice when unrolling the DBN for inference. For nodes that have a constant value at every time-slice, this is a real waste of memory and computational power. To represent the notion of nodes that are constant for every time-slice graphically, we were inspired by the notion of a *plate* from graphical model theory that was originally developed independently by both Spiegelhalter and Buntine [Bun94]. In the original definition, the plate stands for N independent and identically distributed (i.i.d.) replicas of the enclosed model. We decided to use it in a similar way for DBNs. However, since the plate was originally introduced without the notion of temporal arcs, we have extended the definition to include temporal arcs, introducing the concept of a *temporal plate*.

Definition (Temporal plate)

The temporal plate is the area of the DBN definition that holds the temporal information of the network. It contains the variables that develop over time (and are going to be unrolled for inference) and it has an index that denotes the sequence length T of the process.

The part of the DBN that is going to be unrolled is represented within the temporal plate, so all variables with temporal arcs need to be inside the temporal plate. Variables outside the temporal plate cannot have temporal arcs! The length T of the temporal process is denoted as an index. All nodes outside the temporal plate are not copied during unrolling, but their arcs are. The temporal arcs are set to the appropriate nodes in future time-slices, according to their temporal order. The temporal plate has the effect that no matter how many time-slices the DBN is unrolled to, the nodes outside the temporal plate are unique. During unrolling, the unique node itself is not copied, but its outgoing arcs are copied to the corresponding nodes in all time-slices. This is useful for modeling variables in the network that remain constant over time, for instance the gender of a patient. We will call these nodes *contemporaneous nodes*.

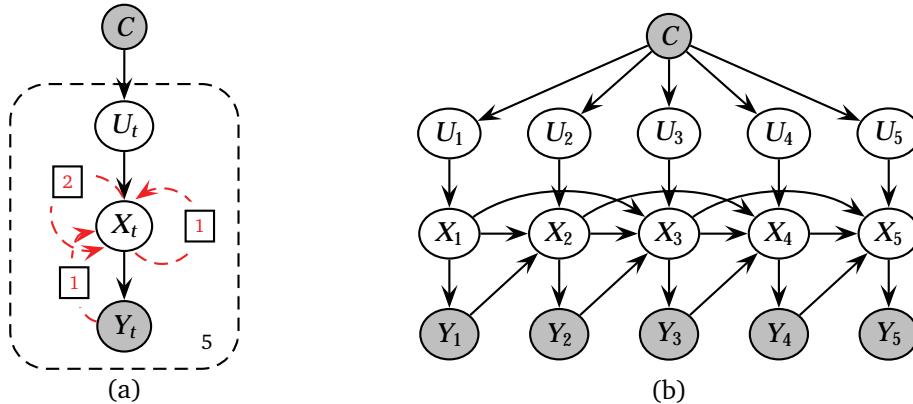


Figure 7.2: (a) The visualization of a second-order DBN with one contemporaneous variable. The temporal plate holds the part of the network that is unrolled for inference, the contemporaneous node is stored outside the temporal plate. The index of the temporal plate denotes that the DBN is going to be unrolled for $t = 5$ time-slices. (b) The unrolled DBN. The initial network is defined implicitly in this example.

Definition (Contemporaneous node)

A contemporaneous node is a node outside the temporal plate whose value remains constant over time. It can have children inside the temporal plate, but no parents. When a DBN is unrolled for inference, a child variable X_t with a contemporaneous parent node C has C as parent for every unrolled node $X_{1:T}$.

An important consequence of the extension of the plate definition is that the replicated time-slices are not i.i.d. anymore because of their temporal relationships. However, if no temporal arcs and no contemporaneous nodes with temporal children exist, then the time-slices are still i.i.d.,

so our definition of the temporal plate is backwards compatible with the original definition of a plate. The graphical visualization of the use of a temporal plate for a DBN definition is shown in figure 7.2.

7.3 Anchor and terminal nodes

Our final extension of the original DBN formalism consists of introducing nodes that are only connected to the first or the last time-slice of the unrolled DBN. This can be useful for introducing extra variables before the start or after the end of the process that need not to be copied every time-slice and are not connected to every time-slice like contemporaneous variables. Typical applications would be situations where we only want to infer variables after or before the process, such as an end-of-sequence observation in the case of ASR [Zwe98]. This also enables us to give an explicit definition of the initial network of the model, just like the definition of Murphy². Graphically, these nodes are stored outside the temporal plate. However, to distinguish them as being of a special kind, a third arc is introduced. In figure 7.3, the *anchor*³ and *terminal* nodes are denoted by a blue dotted arc.

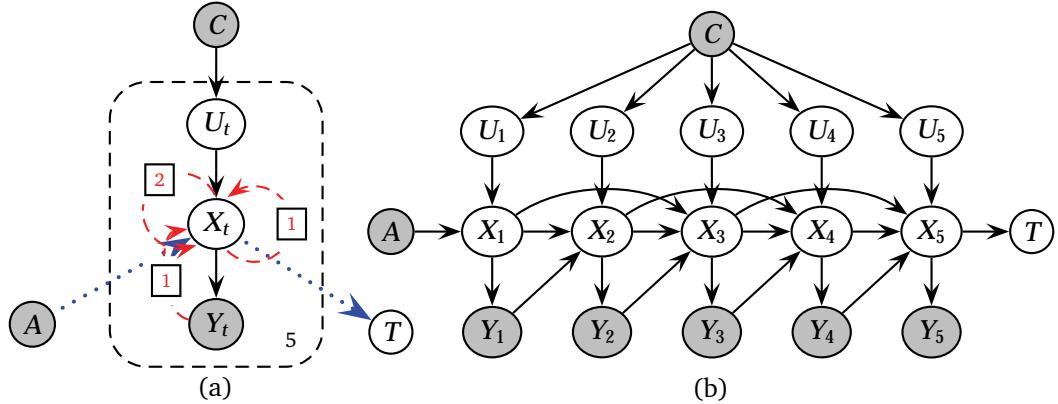


Figure 7.3: (a) The visualization of a second-order DBN. The temporal plate holds the part of the network that is unrolled. The anchor and terminal nodes are stored outside the temporal plate, next to the nodes that are contemporaneous. The index of the temporal plate denotes that the DBN is going to be unrolled for $t = 5$ time-slices.
(b) The unrolled DBN. It is possible to explicitly model a different initial network using the anchor nodes.

Definition (Anchor node)

An *anchor node* is a node outside the temporal plate that has one or more children inside the temporal plate. If the DBN is unrolled for inference, the anchor node is only connected to its children in the first time-slice of the unrolled network.

Definition (Terminal node)

A *terminal node* is a node outside the temporal plate that has one or more parents inside the temporal plate. If the DBN is unrolled for inference, the terminal node is only connected to its parents in the last time-slice of the unrolled network.

7.4 Extended mathematical representation

Introducing a temporal plate with contemporaneous variables (**C**), anchor (**A**) and terminal (**T**) variables also changes the mathematical representation of DBNs. We have already adapted

²Using the B_1 part of the (B_1, B_{\rightarrow}) pair.

³We call them *anchor* nodes to distinguish them from the (implicit) initial network of a DBN (B_1 of Murphy's definition).

Murphy's original equations to k^{th} -order Markov processes earlier in this section, resulting in equations 7.1 and 7.2. Now, we also need to include the \mathbf{C} , \mathbf{A} and \mathbf{T} variables. Adapting the transition model (equation 7.1) results in equation 7.3.

$$p(\mathbf{Z}_t | \mathbf{Z}_{t-1}, \mathbf{Z}_{t-2}, \dots, \mathbf{Z}_{t-k}, \mathbf{C}) = \prod_{i=1}^N p(Z_t^i | \mathbf{Pa}(Z_t^i), \mathbf{C}^i). \quad (7.3)$$

where $\mathbf{Pa}(Z_t^i)$ denote the parents of Z_t^i inside the temporal plate and \mathbf{C}^i denote the contemporaneous variables that are a parent of Z_t^i .

The joint distribution for a sequence of length T can be obtained by unrolling the $(k+1)$ TBN and then multiplying all the CPTs, including those of the \mathbf{A} , \mathbf{C} and \mathbf{T} variables. To achieve this, equation 7.2 needs to change to equation 7.4.

$$p(\mathbf{A}, \mathbf{C}, \mathbf{Z}_{1:T}, \mathbf{T}) = p(\mathbf{C}) \cdot p(\mathbf{A}|\mathbf{C}) \prod_{i=1}^N p(Z_1^i | \mathbf{Pa}(Z_1^i), \mathbf{A}^i, \mathbf{C}^i) \prod_{t=2}^T \prod_{i=1}^N p(Z_t^i | \mathbf{Pa}(Z_t^i), \mathbf{C}^i) \prod_{i=1}^N p(\mathbf{T}^i | \mathbf{Z}_T^i, \mathbf{C}^i). \quad (7.4)$$

\mathbf{A}^i denote the anchor variables that are a parent of Z_1^i and \mathbf{T}^i denote the terminal variables that are a child of Z_T^i . To conclude, in order to completely specify the extended DBN, we need to define four sets of parameters:

1. **Initial parameters:** $p(\mathbf{A}|\mathbf{C})$, that specify the probability distribution of the anchor variables and $p(\mathbf{Z}_1 | \mathbf{Pa}(\mathbf{Z}_1), \mathbf{A}, \mathbf{C})$, that specify the probability distribution of the initial time-slice in the temporal plate.
2. **Temporal parameters:** $p(\mathbf{Z}_t | \mathbf{Pa}(\mathbf{Z}_t), \mathbf{C})$, that specify both the transition distribution between two or more consecutive time-slices and the distribution within a time-slice.
3. **Contemporaneous parameters:** $p(\mathbf{C})$, that specify the probability distribution of the contemporaneous variables.
4. **Terminal parameters:** $p(\mathbf{T} | \mathbf{Z}_T, \mathbf{C})$, that specify the distribution of the terminal variables.

Again, the parameters can be obtained by expert elicitation or learned from a dataset. Our parameter learning methodology for the extended DBN formalism is discussed in chapter 9.

7.5 Empirical study

To get an insight of the performance gain in terms of execution time and memory usage of the extended DBN formalism over Murphy's DBN formalism, an empirical study needed. The following experiments give that insight, next to some insight on the overall performance of the SMILE library. Our claimed gain in modeling power partly depends on the performance gain and partly depends on the modeled domain. We will show how to use the extended DBN formalism in part III of this thesis, where DBN models of two physiological processes are presented.

7.5.1 Design

The experiments measured duration and memory usage of inference on four basic DBNs with varying sequence lengths (T was set consecutively to 500, 1000, 2000, 4000, and 8000 time-slices). These sequence lengths were chosen to get insight in how the inference complexity grows for the two DBN formalisms when inferring longer sequences. The basic DBNs exploit the three properties of the extended DBN formalism that can have a large impact on the performance:

1. The introduction of contemporaneous nodes,
2. The possibility of k^{th} -order Markov processes, and

3. The introduction of terminal nodes.

The DBN topologies modeled using the extended formalism and Murphy's formalism are shown in table 7.1. This table also includes a *baseline* DBN. This is a minimal DBN that is used to enable us to compare the different extensions. The left column of the table shows the DBN modeled with the extended formalism, the middle column shows the DBN converted to a DBN according to Murphy's formalism, and the right column gives a short explanation on how to get to that conversion. From the DBN inference techniques discussed in section 3.2, we choose to stick with unrolling the DBN and applying a BN inference algorithm. Because exact inference is not feasible for (D)BNs with many nodes per time-slice and many time-slices, the state-of-the-art EPIS sampling algorithm [YD03] was used for inference.

7.5.2 Setup

The experiments were conducted on a 1.5Ghz Pentium-M processor with 512Mb RAM as a real-time process (priority 24) under Windows XP. Because it is difficult to get a reliable estimate based on a single measurement, inference on every DBN topology is repeated 100 times after which the average is calculated. The CPT parameters are randomized at every iteration. Measurement of memory usage was done using Sysinternals' Process Explorer (<http://www.sysinternals.com>).

7.5.3 Results

An overview of the experimental results is given in table 7.2. Before we consider the results, it must be stated that the performance gain measured from these experiments cannot be generalized just like that. A great deal of the performance of a DBN depends on the overall complexity of the topology and its observable nodes, not only on whether or not we use the features provided by the extended formalism. However, it can be concluded that when one wants to model a process that is suitable for incorporation of contemporaneous nodes, has a temporal order larger than 1, and/or has terminal nodes, the performance gain in speed and memory is significant.

The experimental results can be divided into two parts: (1) performance gain in time and (2) performance gain in memory. We can see clearly from table 7.2 that both the performance gain in time and memory is significant for every extension of the formalism. The biggest improvement is due to the addition of the terminal nodes, because adding terminal nodes to the last time-slice does not influence drastically the performance of the *baseline* DBN. The two experiments with temporal arcs of different temporal orders also show promising results. However, we argue that the incorporation of the contemporaneous nodes in the extended formalism will be the most useful. As can be seen in table 7.2, inference time and memory costs are always an order of magnitude larger with Murphy's formalism compared to the extended formalism with the addition of contemporaneous nodes. Furthermore, feedback from the field suggests that contemporaneous nodes can be used in a wide variety of domains as index variables, whereas higher-order temporal processes are less commonly used.

One final thing worth mentioning is the space complexity of inference for DBNs. Currently, SMILE naively unrolls the DBN for the required number of time-slices after which inference is performed. For Murphy's formalism, special-purpose algorithms exist that exploit the repetitive structure of a DBN, so only a couple of time-slices have to be kept in memory at every time instant. This decreases memory usage at the cost of extra running time. Examples of these algorithms are discussed in the following papers: [Dar01, ZP00, BMR97]. However, these existing algorithms are incompatible with the extended formalism and thus need to be adapted, something we will not do in the research presented in this thesis. For now, we will just stick to performing inference by the unrolling the DBN and applying static inference algorithms.

Table 7.1: The four basic DBNs and the baseline DBN modeled using the extended DBN formalism and Murphy's DBN formalism. Both variants model the same process.

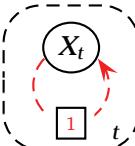
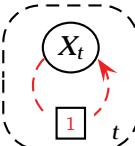
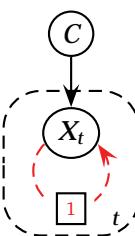
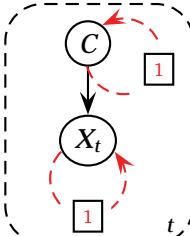
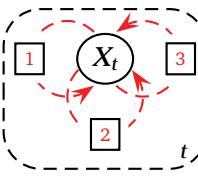
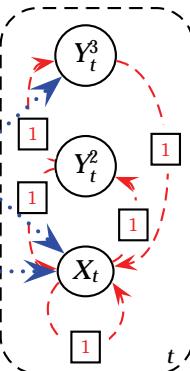
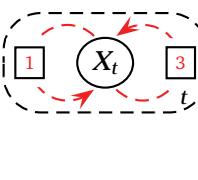
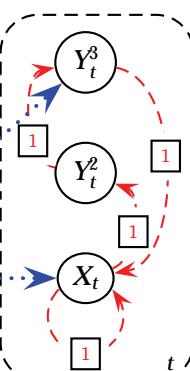
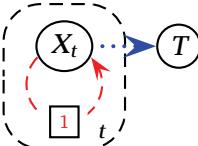
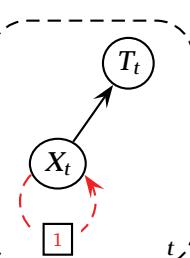
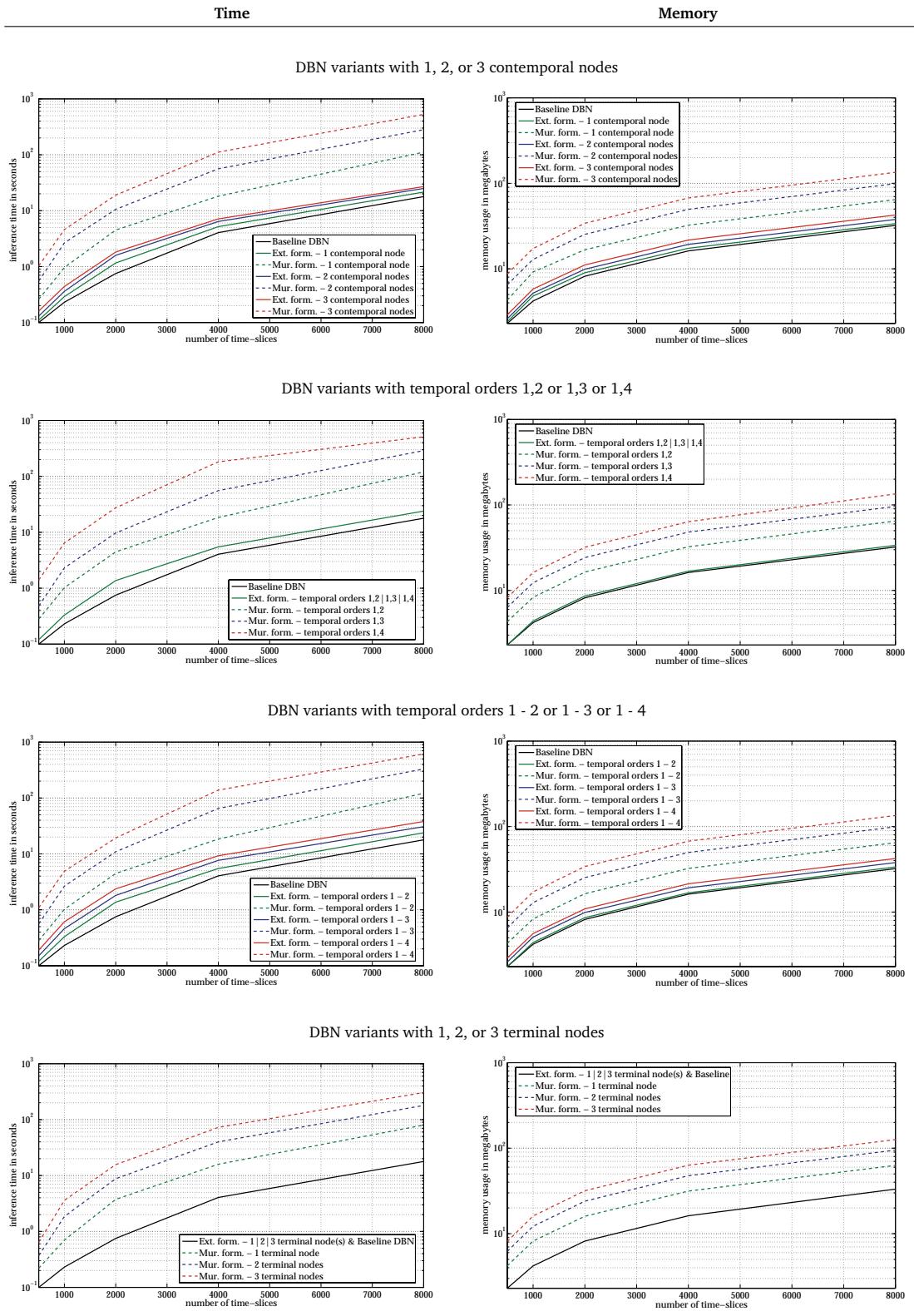
Extended formalism	Murphy's formalism	Variants	Conversion explanation
		None.	The baseline DBN. This DBN is the same for the extended formalism and Murphy's formalism and has only one node and an arc of temporal order 1.
			To incorporate contemporaneous nodes in Murphy's formalism, the contemporaneous nodes must be copied to every time-slice and the temporal CPT must be defined as a deterministic one. The initial CPT of the contemporaneous nodes is taken from the extended formalism.
			To model k^{th} -order Markov processes using a first-order DBN, transfer nodes need to be introduced. These nodes transfer the value of a node at time t to consecutive time-slices. The CPT parameters of the transfer nodes are fully deterministic. Two kinds of k^{th} -order Markov processes are used in the experiment: One with all temporal arcs up to and including k^{th} , and one with only first and k^{th} -order temporal arcs. This is the former variant.
			To model k^{th} -order Markov processes using a first-order DBN, transfer nodes need to be introduced. These nodes transfer the value of a node at time t to consecutive time-slices. The CPT parameters of the transfer nodes are fully deterministic. Two kinds of k^{th} -order Markov processes are used in the experiment: One with all temporal arcs up to and including k^{th} , and one with only first and k^{th} -order temporal arcs. This is the latter variant.
			The modeling of terminal nodes using Murphy's formalism is a matter of simply copying the terminal nodes to every time-slice. This implies that although only interesting at the end of the process, the value is inferred for every time-slice.

Table 7.2: Performance of inference in time and memory for the four DBN variants and the baseline DBN.

7.6 Summary

To conclude this chapter, let us briefly review our extension of the DBN formalism. First of all, the our extension offers far more modeling power, such as the introduction of k^{th} -order temporal arcs for modeling of complex time-delays, and the introduction of contemporaneous nodes for modeling the *context* of variables in the temporal plate. This modeling power enables the modeler to apply our DBN formalism to a wide range of complex temporal problems. Second, variables that do not have to be copied every time-slice are not copied. This feature delivers both a considerable performance gain and a more intuitive way of modeling. Third, by introducing the notion of a temporal plate and temporal arcs, the visualization of complex models is more insightful, making a DBN model easier to understand for field experts, and specification of any DBN can be as concise as possible. In part III, we will use our extended DBN formalism on two physiological processes and show how its properties can be used in practice.

Chapter 8

Temporal reasoning in GeNIe and SMILE

In chapter 3 we presented the DBN formalism that is reasonably common since [Mur02]. We argued that the original formalism is too limited for a wide range of temporal reasoning applications and as a result, the formalism was extended in chapter 7, after which it was validated to get insight in its advantages. An overview of existing software was given in chapter 4, where we concluded that no satisfying software solution exists yet.

From the chapters mentioned above, it became clear that there was a need for reliable and fast temporal reasoning software of industrial quality with academic capabilities. This chapter presents the requirements, design, implementation, and testing of such software. The overall goal was to extend the already widely used high quality software SMILE and GeNIe with temporal reasoning. The users of this software are generally field experts with a basic knowledge about (temporal) reasoning, but not about the underlying algorithms. Because this is a *reengineering* project, the current design and implementation choices of SMILE and GeNIe needed to be respected, which resulted in some extra challenges not applicable when building software from the ground up.

In this chapter, a general overview of the design and implementation trajectory is given; these are loosely based on methods described in [BD00]. Because the focus of this thesis is not on the software engineering part, only the most important aspects are discussed in detail. One final thing worth mentioning is the fact that SMILE and GeNIe are in fact two different products. SMILE is the underlying library and GeNIe is the shell that provides the GUI. This separation makes that it is sometimes wise to treat them differently and sometimes to treat them as a whole. In the following sections, it will become clear from the text whether they are treated as a whole or separately. The author of this thesis was fully responsible for the design and implementation of temporal reasoning into SMILE and for the design of the GUI in GeNIe. The implementation of the graphical user interface in GeNIe and the extension of SMILEXML was done by T. Sowinski.

8.1 Requirements

8.1.1 Actors

GeNIE has three types of actors: **network modelers**, **decision makers**, and **domain experts**. SMILE has two types of actors: **application builders** and **network modelers**. The possibility exists that one or more of these actors are represented by one user, but in larger projects, the different responsibilities will most probably lie with different users.

An **application builder** is an actor that builds an application on top of the SMILE library, in a similar way as GeNIE is built on top of SMILE. In many domains, the general user-interface of GeNIE does either not suffice or is too extensive. In these type of applications, a specialized application can be built that gives the **decision maker** only access to a subset of the functionality of SMILE. The **application builder** only interacts with the API of SMILE by writing a specialized tool on top of it.

The **network modeler** is the actor that elicits knowledge from a **domain expert** to model the problem domain in a DBN. Knowledge elicitation can refer to elicitation of the DBN structure and/or DBN parameters. The **network modeler** interacts with the software by building a DBN from the elicited knowledge.

The **decision maker** is the actor that actually uses the DBN, either by using a specialized tool or GeNIE. The **decision maker** inserts evidence in the network, performs inference and makes decisions based on the posterior beliefs. He does not change the DBN structure or parameters. The **decision maker** interacts with the software to get a better insight in the situation and to make (hopefully) better decisions.

Finally, the **domain expert** is the actor that has extensive knowledge of the problem domain. He does not have to be a direct user of the system, but it is still an actor, as the system needs a way to provide intuitive input of domain knowledge. GeNIE already has means to provide such intuitive input, so this actor is not taken into account in the current temporal reasoning design.

8.1.2 Use cases

A common way of obtaining use cases is *scenario identification*, where developers observe users and develop a set of detailed scenarios for typical functionality provided by the future software. However, because of time and resource constraints, we choose to obtain the most important use cases at brainstorm sessions within the DSL, sessions with users from Intel Research and Philips Research, and feedback from individual users. This resulted in the use cases presented in table 8.1. We have to keep in mind that:

- Table 8.1 only contains the added use cases, i.e. use cases that are introduced because of the extension of the software with temporal reasoning.
- The use cases for GeNIE and SMILE are a little different, because GeNIE hides some specific implementation details from the user. However, the functionality is basically the same. Table 8.1 shows the use cases for GeNIE, as they demonstrate the general functionality in a concise manner.
- An important constraint in the design of new use cases is keeping them consistent with the already existing underlying structure of the software.

8.1.3 Functional requirements

The GeNIE and SMILE software is extended to incorporate temporal reasoning using the extended DBN formalism as previously discussed. The extension provides support for nodes with the four temporal types: (1) anchor, (2) contemporaneous, (3) plate, and (4) terminal. Nodes in

Table 8.1: Use cases for the temporal reasoning extension.

Use case name	TemporalType (Set, Edit)	Use case name	TemporalArc (Add, Remove, Edit)
<i>Participating actor</i>	Executed by the network modeler.	<i>Participating actor</i>	Executed by the network modeler.
<i>Entry condition</i>	1. The network modeler selects the node that needs its temporal type changed.	<i>Entry condition</i>	1. The network modeler selects the arc drawing tool.
<i>Flow of events</i>	2. The network modeler drags the node to one of the following parts of the DBN definition: anchor, contemporaneous, temporal plate, or terminal.	<i>Flow of events</i>	2. The network modeler draws an arc from a parent to a child (in the temporal plate). 3. The network modeler sets the temporal order of the arc.
<i>Exit condition</i>	3. The node has its temporal type changed.	<i>Exit condition</i>	4. The network has a new temporal dependency.
Use case name	TemporalProbabilities (Set, Edit)	Use case name	NumberOfSlices (Set, Edit)
<i>Participating actor</i>	Executed by the network modeler.	<i>Participating actor</i>	Executed by the decision maker.
<i>Entry condition</i>	1. The network modeler selects the node that needs its temporal probabilities set.	<i>Entry condition</i>	1. The decision maker opens the global temporal network settings.
<i>Flow of events</i>	2. The network modeler selects the temporal CPT(s) of that node. 3. The network modeler specifies the parameters of the temporal CPT(s).	<i>Flow of events</i>	2. The decision maker sets the number of time-slices in the appropriate option.
<i>Exit condition</i>	4. The node has its temporal probabilities set.	<i>Exit condition</i>	3. The network has the number of time-slices set for which inference must be performed.
Use case name	TemporalEvidence (Insert, Edit, Remove)	Use case name	TemporalValue
<i>Participating actor</i>	Executed by the decision maker.	<i>Participating actor</i>	Executed by the decision maker.
<i>Entry condition</i>	1. The decision maker selects the node for which there is temporal evidence.	<i>Entry condition</i>	1. The decision maker has set the number of time-slices. 2. The decision maker performs inference on the temporal network. 3. The decision maker selects the node for which he wants to know the temporal posterior beliefs.
<i>Flow of events</i>	2. The decision maker inserts temporal evidence at the appropriate time-slice(s).	<i>Flow of events</i>	4. The decision maker views the temporal beliefs in one of the dedicated representation windows.
<i>Exit condition</i>	3. The node has some or all of its temporal evidence set.	<i>Exit condition</i>	None.
Use case name	UnrollNetwork	Use case name	LocateNode
<i>Participating actor</i>	Executed by the decision maker.	<i>Participating actor</i>	Executed by the decision maker.
<i>Entry condition</i>	1. The decision maker has set the number of time-slices.	<i>Entry condition</i>	1. The decision maker has unrolled the temporal network. 2. The decision maker selects an unrolled node.
<i>Flow of events</i>	2. The decision maker opens the global temporal network settings. 3. The decision maker selects the unroll option.	<i>Flow of events</i>	3. The decision maker selects the locate in original DBN option.
<i>Exit condition</i>	4. A window is opened with the temporal network unrolled for the given number of time-slices.	<i>Exit condition</i>	4. The temporal network is opened and the located node is highlighted.

the temporal plate can have temporal arcs with arbitrary temporal orders. Users are able to define temporal probability distributions and the number of time-slices of the temporal process. Users can set temporal evidence and perform inference on the temporal network to obtain the posterior temporal values. The posterior temporal values can be visualized in several ways to give the user insight in the development of the different variables over time. A temporal network can be unrolled to a static network for a given number of time-slices. This static network can be used for debugging purposes. Finally, the software is able to restore and save temporal networks.

8.1.4 Nonfunctional requirements

The nonfunctional requirements relevant for the temporal extension are presented in this section. Above all, the design and implementation of the temporal extension must be consistent with the current design and implementation of the software.

User interface and human factors GeNIe and SMILE have a wide variety of users, varying from users from the industry (Boeing, Intel, Philips) to academic users from a wide range of universities all over the world. Most of these users have a basic knowledge of BNs, but are not experts on more complex issues, such as the exact workings of the different inference techniques. The interface of the software needs to hide as much of the more complex functionality as possible in a consistent way, so these users do not get confused. However, expert users are able to make use of this advanced functionality as well. An important property that all the users share is that they want as much freedom as possible when modeling networks.

Documentation An advantage of the DSL software over its competitors is the availability of documentation, reference manuals, tutorials and examples on all the provided functionality. An on-line documentation, reference manual and tutorial also needs to be provided for the temporal extension of both GeNIe and SMILE¹. Example networks need to be made available in the network repository of <http://genie.sis.pitt.edu>.

Platform considerations SMILE is compiled for a wide variety of platforms: Windows, Unix (*BSD, Solaris, Mac OS X), Linux, and PocketPC. This means that for the implementation only the standard C++ library can be used, as it is fully cross-platform. GeNIe is only available for Windows, because it relies heavily on the Microsoft Foundation Classes. There are no hardware considerations, as long as the hardware is able to run one of the before-mentioned platforms.

Performance characteristics Inference techniques rely on already implemented algorithms, thus the achieved response time and accuracy are determined by the efficiency of the existing implementation. The implementation of the temporal extension is light-weight, so users of static networks do not notice a performance drop when upgrading to a version that has support for temporal reasoning.

Error handling and extreme conditions A current design choice that must be respected is that network changes that result in inconsistencies are always blocked. Because inference can use a lot of memory resources, the software should be able to handle situations where no more memory can be allocated.

¹See appendices A and B.

Quality issues SMILE and GeNIe are robust in the sense that they do not crash when doing what they are designed to do. When something goes wrong, the software needs to give a proper warning and the network needs to stay consistent.

System modifications Future changes of the software are done by different people at the DSL. Because many different people develop the software, good documentation is a must.

8.1.5 Pseudo requirements

The SMILE library and thus the temporal extension needs to be written in C++. GeNIe relies heavily on the Microsoft Foundation Classes and will its incorporation of the temporal extension. Finally, networks are saved to a proprietary XML file format (.xds1). This file format needs to be extended to enable users to save and load temporal networks.

8.2 Design

In the previous section, a global overview of the system requirements was presented. After an analysis step, a system design can be made from these requirements. We purposely skip the analysis model, as this model only shows the interaction between the actors and the software, but does not contain information about its internal structure. Because the temporal extension cannot be seen separately from the DSL software, this section directly presents the resulting class diagrams of the temporal extension integrated in the DSL software.

8.2.1 Design goals

After requirements elicitation, we could derive the most important design goals. The design goals identify the qualities that the temporal extension focuses on. Note that the design partly follows from the research assignment presented in section 1.2, where we stated that temporal reasoning has to be implemented in SMILE instead of making stand-alone version. Evaluating the requirements from the previous section resulted in four main design goals:

1. Since the start of the implementation of the SMILE library in 1997, many different people have added many different features. As a result, the SMILE library gained a lot of momentum. A returning challenge for every extension is to keep the SMILE functionality consistent. For the extension with temporal reasoning this is even more important, as it adds functionality to some of the core classes of SMILE. So, the first design goal is *consistency of the temporal extension with the current SMILE library*.
2. An important property of SMILE over its competitors is its performance. The temporal extension must not have an effect on the performance of existing static networks. From this fact follows the second design goal: *the temporal extension must be light-weight*.
3. Currently, SMILE has many users in both the industrial and the academic world. More often than not these users have written dedicated applications that use the SMILE library as underlying engine. The third design goal follows from the fact that these users are willing to try temporal reasoning, but not if this results in a total rewrite of the dedicated applications. The temporal extension of SMILE must provide *easy access to temporal reasoning in existing networks and dedicated applications*.
4. The final design goal follows from the fact that many different people work on SMILE. Next to a good documentation, the design should be *Maintainable and expandable*.

These four goals need to be satisfied for a successful implementation of temporal reasoning in SMILE. How these goals are satisfied will become clear from the design choices made in the remainder of this chapter. On a global level, the first goal is satisfied by transferring much

of the static functionality to the temporal domain. The second goal is satisfied by making a design that only initializes the temporal classes when needed. The third design goal is satisfied by integrating the temporal extension in the core classes of SMILE and keeping the extended SMILE library backwards compatible. Note that it can seem that design goal two excludes design goal three, because integrating the temporal extension in the core classes of SMILE can easily decrease the current performance. However, in the current approach, this is not the case, as we will show in the final design. Design goal four is satisfied by adopting the current object-oriented design approach.

8.2.2 Subsystem structure

The SMILE library is divided into four subsystems that provide different parts of the BN functionality. In most cases, the subsystems are maintained by people that are dedicated to that particular subsystem. Users of SMILE can choose to use load different subsystems for extra functionality or choose to stick with the basic SMILE functionality. From the design goals follows that the temporal extension should be added to the main SMILE library and not be designed as a subsystem.

1. **SMILE** This is the subsystem that provides all the basic functionality for the design of and inference on Bayesian networks and influence diagrams. It has support for several CPDs, such as CPTs, canonical gates, and in the near future also arbitrary continuous distributions. The temporal extension is part of this subsystem.
2. **SMILearn** This subsystem extends SMILE with learning and data mining functionality. It provides a set of specialized classes that implement learning algorithms and other useful tools for automated building graphical models from a dataset. Learning algorithms include methods for learning both structure and parameters of BNs.
3. **SMILEXML** This subsystem provides the functionality to save and restore networks to a XML file in the .xds1 format. This file format needed to be extended to be able to save and restore networks with temporal relations.
4. **SMILEMech** A relatively new research topic in BNs is the incorporation of structural equations. This subsystem provides functionality for this incorporation and is currently under development at the DSL.
5. **Wrappers** To enable users of programming languages other than C++ to interface with the SMILE functionality, different wrappers are provided. Currently, Java programmers can use the jSMILE wrapper, .NET programmers can use the SMILE.NET wrapper, and VB6/VBA programmers can use the SMILEX ActiveX wrapper. Functionality that is added to SMILE generally has a thorough testing period, after which the extra functionality becomes available through the wrappers.
6. **GeNIe** This subsystem provides an easy-to-use GUI to SMILE for users of Microsoft Windows. The GUI needs to be extended to provide access to the temporal reasoning functionality.

The subsystem structure of the DSL software is shown in figure 8.1. To summarize, the temporal extension was incorporated into the core classes of the SMILE subsystem. After that, SMILEXML was adapted to enable users to save and restore temporal networks. Finally, GeNIe was adapted to provide access to the temporal functionality. When the development of temporal reasoning in SMILE has stabilized, the functionality will also be made accessible through the different wrappers. However, that will not be part of this thesis.

8.2.3 Class identification

To be able to incorporate temporal reasoning in SMILE, several new classes and types were designed. A global description is given here. How these classes fit into the current design and how they relate to each other is discussed in section 8.3.

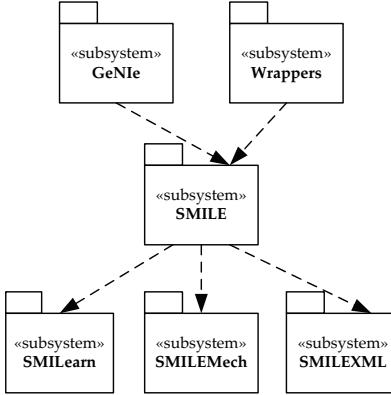


Figure 8.1: Subsystem structure of DSL software.

Temporal arcs A temporal arc is defined as an arc from a parent to a child with a certain temporal order. The `DSL_temporalArcs` class keeps track of the temporal parents and children of a node.

Temporal node types Because we have extended the DBN formalism with the notion of a temporal plate, anchor and terminal nodes, a property of type `dsl_temporalType` is introduced that can be set to one of the following values: `dsl_normalNode`, `dsl_anchorNode`, `dsl_terminalNode`, or `dsl_plateNode`.

Temporal definitions Every node needs to keep track of an extra CPD for every incoming temporal arc with a different temporal order. The superclass `DSL_nodeDefTemporals` is introduced that contains virtual methods to administer temporal CPDs. Several subclasses implement these virtual methods: the `DSL_defTemporalCpts` class for the administration of the temporal CPTs; the `DSL_ciDefTemporals` class that adds functionality for canonical gates and the `DSL_noisy-MaxTemporals` that inherits from `DSL_ciDefTemporals` to implement the Noisy-MAX gate.

Temporal evidence and beliefs Every node in the temporal network is responsible for the administration of its evidence for every time-slice. After inference, each node contains the updated beliefs for every time-slice. The `DSL_nodeValTemporals` superclass contains virtual methods for this administration. Again, every type of CPD has its own subclass. Because the posterior beliefs of CPTs and canonical gates can be expressed in the same way, only one subclass is introduced that implements the virtual methods: `DSL_valTemporalBeliefVectors`.

Inference For inference in a temporal network, the `DSL_dbnUnrollImpl` class is introduced that takes a temporal network, unrolls it for a given number of time-slices, performs inference and copies the posterior beliefs back to the temporal network.

8.2.4 Persistent data management

GeNIE and SMILE save networks to the flat files in the XML-based `.xds1` format. XML is a text-based general-purpose markup language for creating special-purpose markup languages, such as the `.xds1` format. Languages based on XML are defined in a formal way, allowing programs to modify and validate documents in these languages without prior knowledge of their form. The XML schema for `.xds1` files can be downloaded from <http://genie.sis.pitt.edu>.

The main advantage of using XML is that it is both a human- and machine-readable format. Another important advantage is that it is platform-independent. A disadvantage can be the

storage-size of files saved in an XML format. XML files tend to get very large, because the syntax is fairly verbose and partially redundant. This is also the case for larger networks stored in the .xds1 file format. However, because generally these networks do not have to be sent over a network and because storage space is cheap, this does not pose a problem.

The .xds1 file format needed to be extended to hold the temporal information of a network. Furthermore, SMILEXML needed to be extended to deal with the extensions of the .xds1 file format. A challenge of incorporating the temporal extension into the .xds1 file format was that at every moment during parsing, the intermediate network must be consistent. For instance, it is not possible to add temporal arcs to a node if that node is not yet in the temporal plate. Basically, the incorporation of temporal networks in the .xds1 file format comes down to:

1. Introducing the attribute *dynamic* to the node tag that can take one of the following values: *anchor*, *plate*, or *terminal*. Nodes for which this attribute is not set are contemporaneous by default. The use of this attribute is not forced to stay compatible with existing networks.
2. Introducing the *dynamic* tag with a *numsllices* attribute. This tag contains the temporal information of the network. It contains the nodes in the temporal plate with temporal parents of a certain order through the *node* tag and the *parent* tag that has an *order* attribute. The temporal CPDs are also saved in this part in corresponding CPD tags, such as the *cpt* tag for CPTs. These tags also get an extra *order* attribute.

When a network is loaded from an .xds1 file, first the nodes and non-temporal arcs and non-temporal CPDs are set, together with the temporal type of these nodes. After that, the temporal arcs and temporal CPDs are added to the nodes in the temporal plate. As a result, .xds1 files can be parsed without getting intermediate networks that are inconsistent. Figure 8.2 shows the extended umbrella network that is also used in the tutorial (appendices A and B) and listing 8.1 shows a snippet of the corresponding .xds1 file.

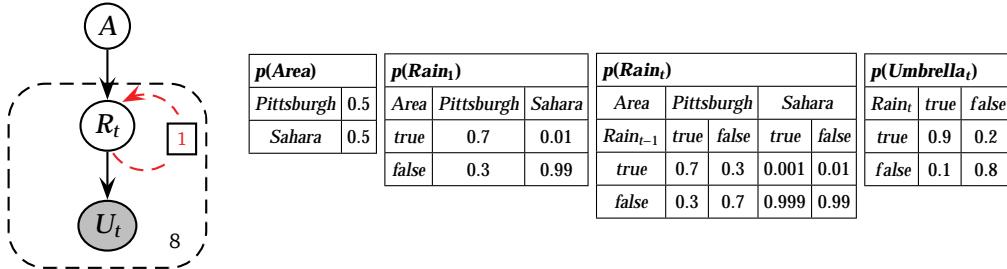


Figure 8.2: The extended umbrella DBN and its CPTs.

```

1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <smile version="1.0" id="UmbrellaNetwork" numsamples="1000">
3   <nodes>
4     <cpt id="Area">
5       <state id="Pittsburgh" />
6       <state id="Sahara" />
7       <probabilities>0.5 0.5</probabilities>
8     </cpt>
9     <cpt id="Rain" dynamic="plate">
10      <state id="True" />
11      <state id="False" />
12      <parents>Area</parents>
13      <probabilities>0.7 0.3 0.01 0.99</probabilities>
14    </cpt>
15    <cpt id="Umbrella" dynamic="plate">
16      <state id="True" />
17      <state id="False" />
18      <parents>Rain</parents>

```

```

19      <probabilities>0.9 0.1 0.2 0.8</probabilities>
20    </cpt>
21  </nodes>
22  <dynamic numsllices="8">
23    <node id="Rain">
24      <parent id="Rain" order="1" />
25      <cpt order="1">0.7 0.3 0.3 0.7 0.001 0.999 0.01 0.99</cpt>
26    </node>
27  </dynamic>
28  <extensions>
29    ...
30  </extensions>
31 </smile>

```

Listing 8.1: The extended umbrella network stored as an .xds1 file.

8.2.5 Graphical user interface

The design of a GUI can be a time-consuming process for both designer and user. In large software projects, a common approach for GUI design is *prototyping*. In this approach, designing the GUI is an iterative process where the designer proposes a design that is presented to the user for evaluation. Based on feedback from the user, the designer modifies the design, presents the modified design, and so on and so forth.

However, because this process is so time-consuming, we decided to follow a different approach. First, we take a look at the target user group that we distinguished in section 8.1:

GeNIE and SMILE have a wide variety of users, varying from users from the industry (Boeing, Intel, Philips) to academic users from a wide range of universities all over the world. Most of these users have a basic knowledge of BNs, but are no experts on more complex issues, such as the exact workings of the different inference techniques.

What extra GUI design criteria follow from the specification of the user group? First of all, our design needs to focus on visualizing temporal reasoning, not on the techniques behind it. Second, because most users use GeNIE as a research tool, debugging of temporal networks must be provided. Of course, the design goals stated earlier in this section also hold: the GUI design for temporal reasoning must be consistent with the current GUI and easy access to temporal reasoning in existing networks must be provided.

Based on this, brainstorm sessions within the DSL were held to design an initial GUI with the help of existing packages. The resulting GUI design was globally evaluated in sessions with potential users from Intel and Philips using screen mock-ups of the design of a DBN and of inference results. After this revision, the global design was finished and during its implementation, individual feedback was used to further refine the GUI. This resulted in the current GUI. One thing to keep in mind is that the process of GUI refinement is never finished. Development of GeNIE is largely driven by feedback from users all over the world and several good suggestions are being implemented at this very moment. The GUI design process resulted in a design of temporal reasoning in GeNIE that can be separated in several subparts. These subparts were incorporated into existing parts of GeNIE to maintain consistency.

Temporal types Remember that in the DBN formalism, a node can be one of the four temporal types: *anchor*, *contemporaneous*, *plate*, and *terminal*. In a static network, the concept of temporal types does not exist. In the GUI design for temporal networks, we needed to incorporate the concept of temporal types without changing the GUI that already existed. This is achieved in the following way: A new network is static by default. When the network modeler decides to add temporal relations, he clicks on *Network → Enable Temporal Plate* in the menu bar as shown in figure 8.3. This results in the network area being divided into four parts, according to the four temporal types defined in chapter 7:

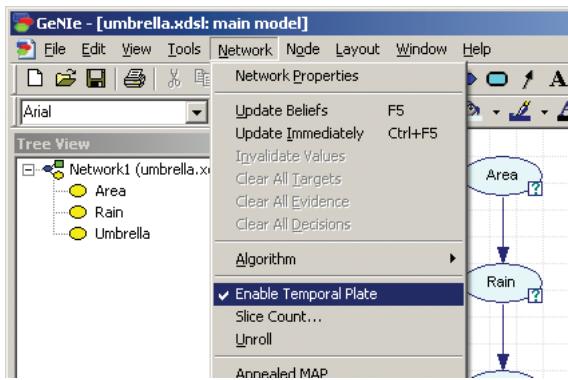


Figure 8.3: Temporal options in the menu bar of GeNIE under Network.

1. **Contemporals** This is the part of the network area where the static nodes are stored by default.
2. **Initial conditions** This is the part of the network area where the anchor nodes are stored.
3. **Temporal plate** This is the part of the network area where the nodes in the temporal plate are stored. Nodes in the temporal plate are the only nodes that are allowed to have temporal arcs. This area also shows the number of time-slices for which inference is performed.
4. **Terminal conditions** This is the part of the network area where the terminal nodes are stored.

The boundaries of the different areas can be changed by dragging them. These boundaries are also used to layout the network when it is unrolled explicitly. Temporal types of nodes can be changed by dragging them to one of the three other areas. A general overview of the network area environment is shown in figure 8.4.

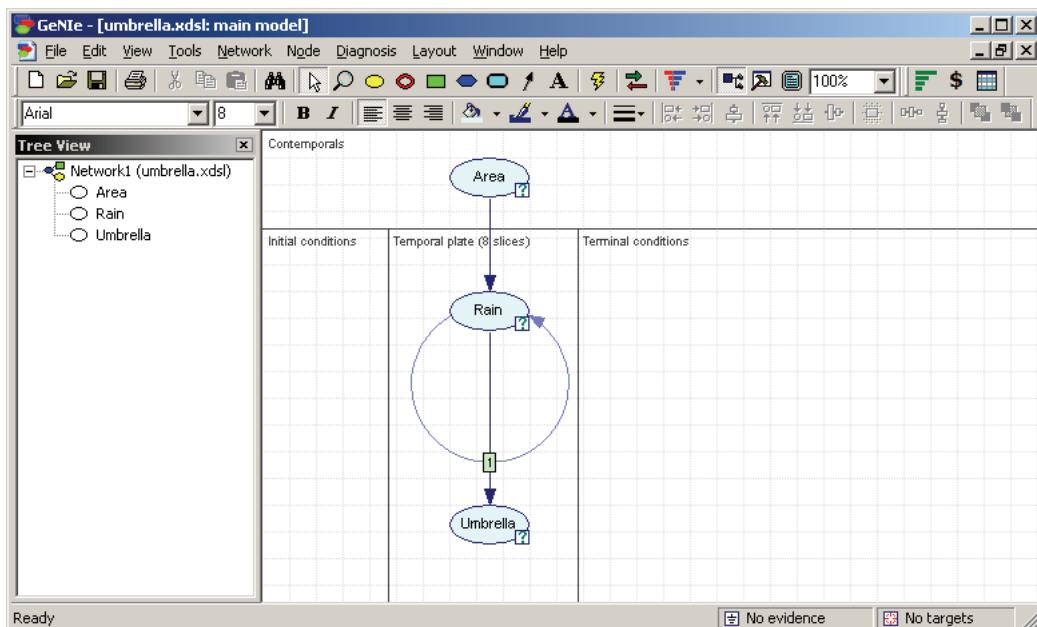


Figure 8.4: General overview of the network area environment in GeNIE.

Temporal arcs After a network modeler drags a node to the temporal plate, he can add temporal arcs. A temporal arc consists of a parent, a child and a temporal order. The network modeler can add a temporal arc by clicking on the arc button and drawing an arc from a parent to a child². When the network modeler has drawn the arc and releases the mouse button, a context menu appears to enable the user to set the temporal order of the arc. Every temporal arc has a small label with its temporal order. Figure 8.5 demonstrates the addition of a temporal arc to a network.

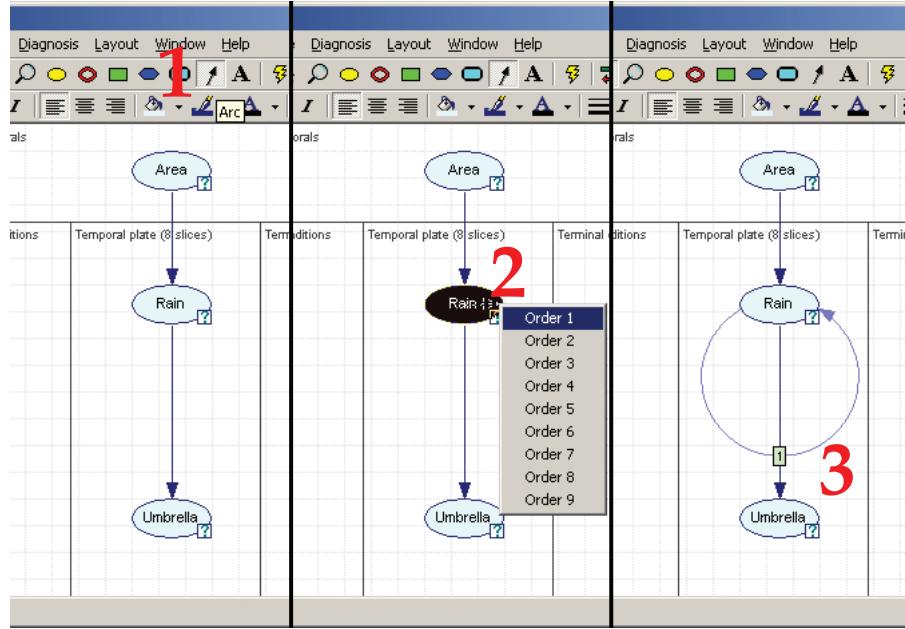


Figure 8.5: Adding a temporal arc in GeNIE.

Temporal definition Every node in the temporal plate needs a CPD for every incoming temporal arc with a different temporal order. Just like setting the CPD for ordinary nodes, the network modeler needs to double-click on the node in the network area and go to the definition tab to set the temporal CPDs. When a node has incoming temporal arcs, the network modeler can select the appropriate temporal CPD from a list and set its parameters. Figure 8.6 demonstrates this process.

Temporal evidence When a temporal network is finished, the decision maker can use it for decision support. Before the decision maker can add observations or evidence to the temporal network, he needs to set the number of time-slices of the temporal network by clicking on *Network → Slice Count....* The number of time-slices denote the time-period the decision maker is interested in. After setting the number of time-slices, he can add evidence to nodes in the temporal network by right-clicking on a node and selecting *Evidence* from the context menu. For non-temporal nodes, the decision maker can set the evidence directly. For temporal nodes, a form appears where the decision maker can add evidence for every time-slice. Figure 8.7 demonstrates the addition of evidence to a temporal node.

²This can be the same node.

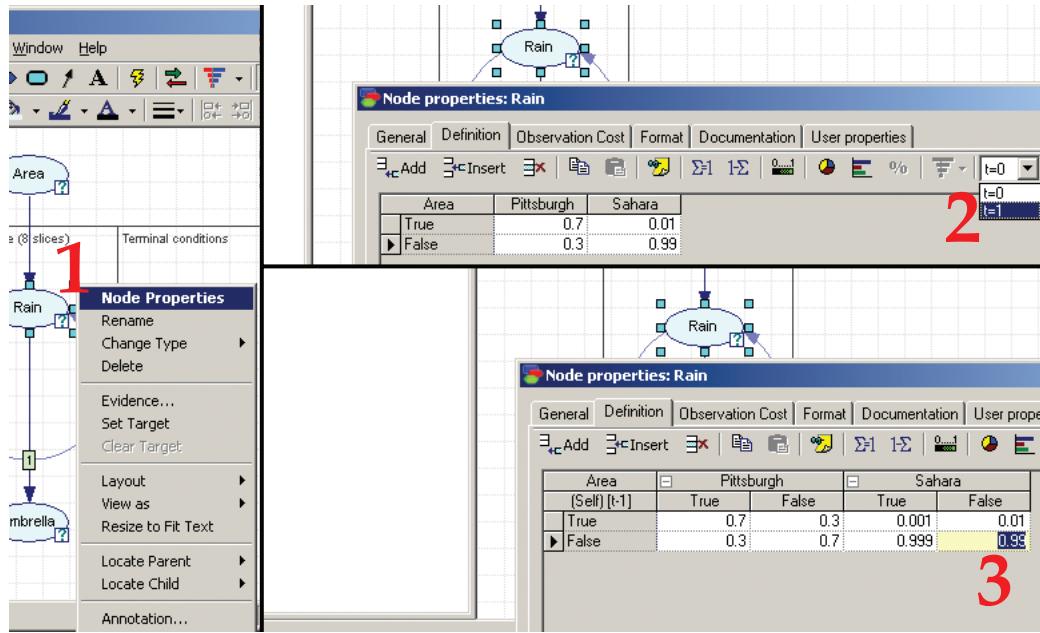


Figure 8.6: Setting the temporal definition of a node in GeNIE.

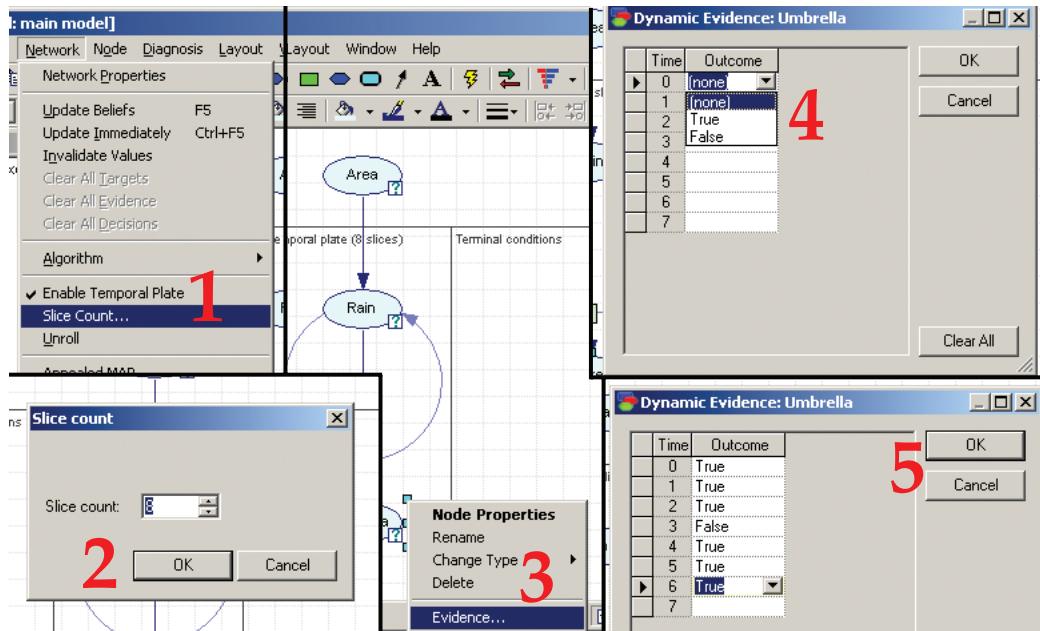


Figure 8.7: Setting temporal evidence in GeNIE.

Temporal inference Calling inference on a temporal network is no different than calling inference on a non-temporal network from a GUI perspective. The decision maker can set the inference algorithm by clicking on *Network* → *Algorithm...* and selecting the appropriate algorithm. The algorithm is called by right-clicking on the network area and selecting *Update Beliefs* from the context menu or by pressing F5.

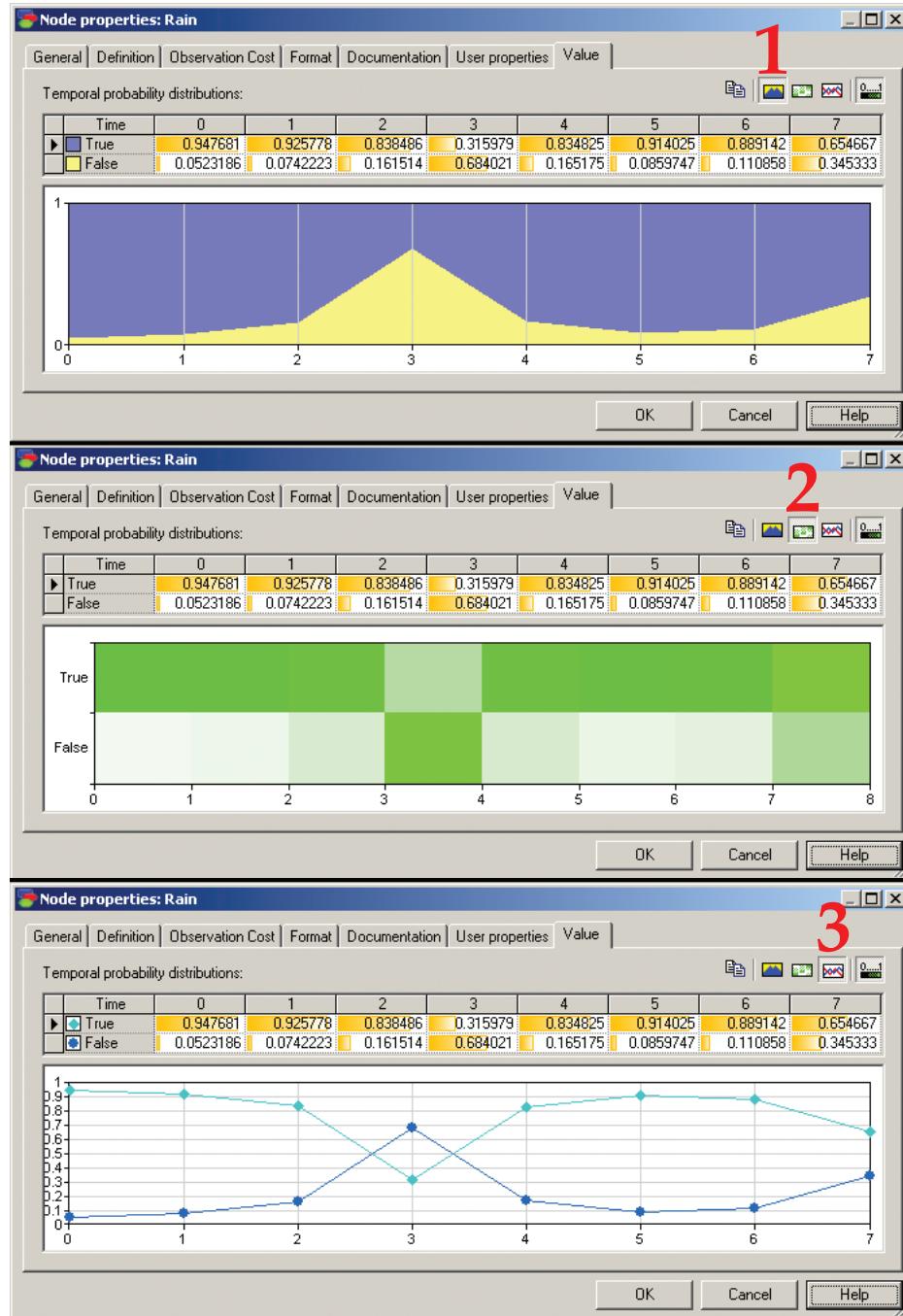


Figure 8.8: Showing the different representations of the temporal posterior beliefs in GeNIE.

Temporal posterior beliefs After inference is called, the temporal network has its beliefs updated. A clear presentation of these updated beliefs is very important for the decision maker. The updated beliefs for a temporal node can be obtained by double-clicking on the node and selecting the *Value* tab. This tab contains the updated beliefs for all time-slices of that node not only in floating point numbers, but also as: an area chart, contour plot, and time-series plot. Both the area chart and time-series plot can be found in other packages as well, but the contour plot is a representation of temporal beliefs that is a unique feature of

GeNIE to the best of our knowledge. The contour plot represents a three dimensional space with the dimensions: time (x-axis), state (y-axis), and probability of an outcome at every point in time (the color). It is possible for the decision maker to cut-off the color range to get more insight into the development and the uncertainty of the temporal process. This representation is especially useful for temporal nodes that have many states, where the other representations are more useful for temporal nodes with not too many states. Figure 8.8 shows the different representations of the temporal posterior beliefs.

Unrolling It can be useful for debugging purposes to explicitly unroll a temporal network for a given number of time-slices. GeNIE provides this possibility through the *Network* → *Unroll* option. When clicking this option, GeNIE opens a new network that has the temporal network unrolled for the given number of time-slices. It is possible to locate a node in the temporal network from the unrolled network by right-clicking on the node in the unrolled network and select *Locate Original in DBN* from the context-menu. The unrolled network that is a result from unrolling the temporal network is cleared from any temporal information whatsoever. It can be edited, saved and restored just like any other static network. Figure 8.9 shows the unrolled representation of a temporal network.

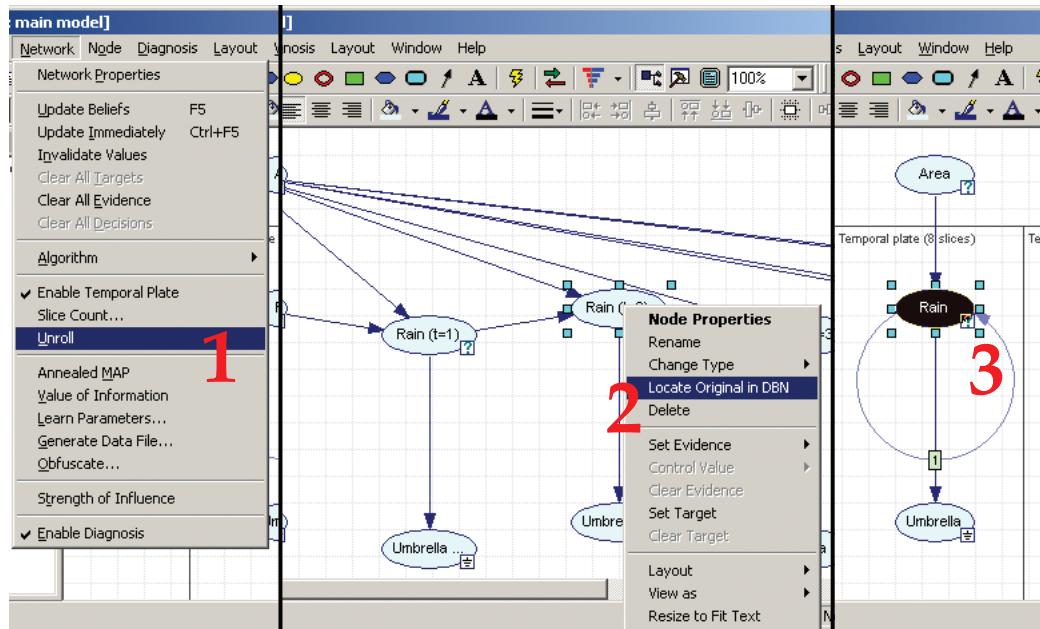


Figure 8.9: Unrolling a temporal network and locating a node in GeNIE.

8.3 Implementation

This section discusses the actual implementation of temporal reasoning in SMILE. It presents how the classes of SMILE are re-engineered to incorporate the classes and data types identified in section 8.2. The implementation of the GUI for temporal reasoning in GeNIE is not discussed, because those changes were not implemented by the author of this thesis. The same holds for the implementation of the .xds1 file format extension in SMILEXML.

A general overview of the introduced and adapted classes and data types is given in figure 8.10. In this class diagram, the classes of SMILE that are not adapted for the temporal reasoning extension are omitted. The specific implementation details for every class are described next in a top-down order. Whenever needed, the interaction between the classes is discussed

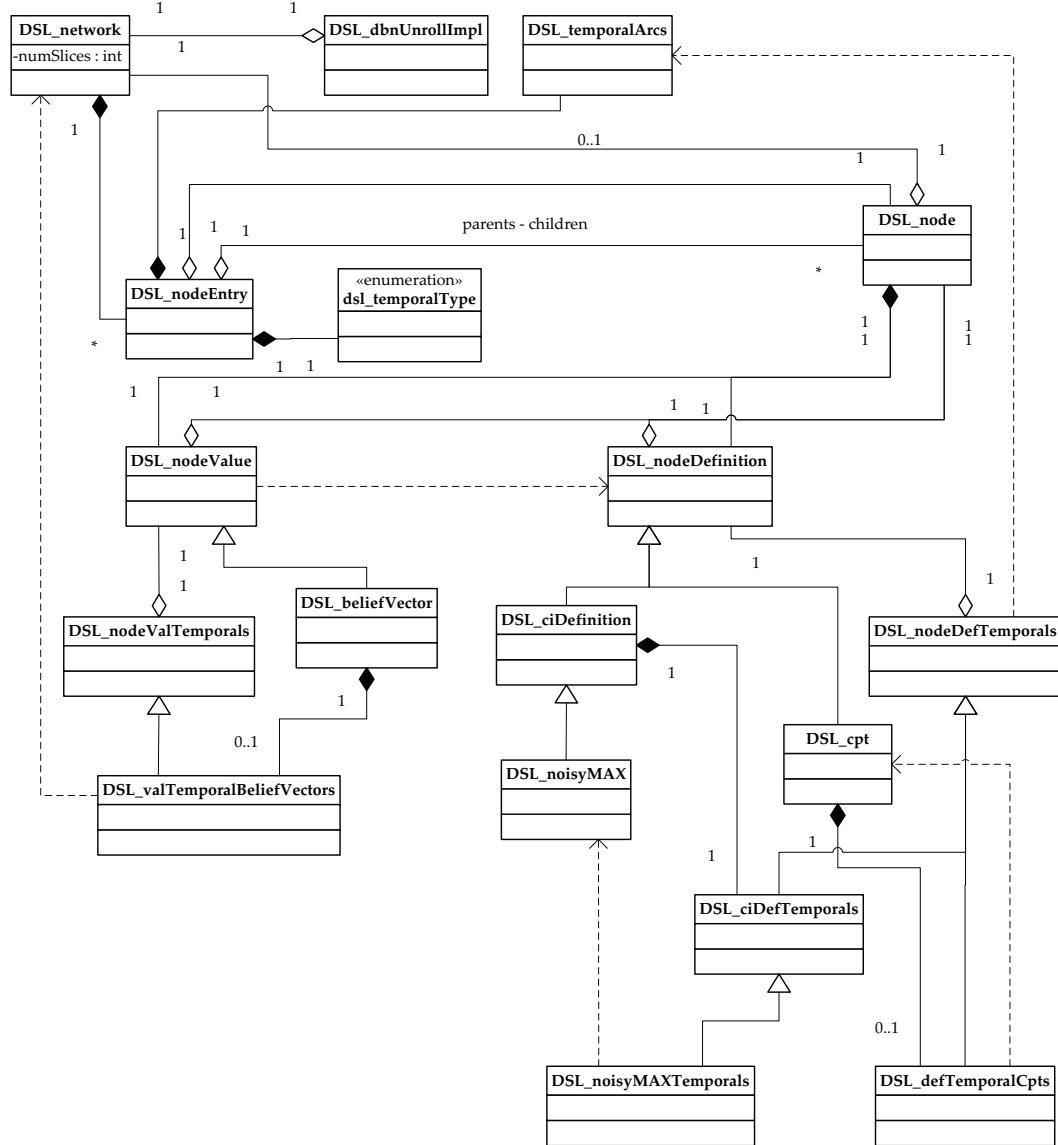


Figure 8.10: General overview of the temporal extension in SMILE.

as well. Unfortunately, not too many implementation details can be given, because SMILE is closed-source. The implementation of the temporal reasoning extension forms a significant part of the current SMILE library.

8.3.1 DSL_network

This is the most important class of the SMILE library. Most functionality can be reached by calling methods from this class. To extend SMILE with temporal reasoning, a property `numSlices` was introduced that holds the number of time-slices. Furthermore, some methods were added for the administration of a temporal network.

DSL_network
-numSlices : int
+AddTemporalArc(in parent : int, in child : int, in order : int) : int
+GetMaximumOrder() : int
+GetNumberOfSlices() : int
+GetTemporalChildren(in node : int) : vector<pair<int, int>>
+GetTemporalParents(in node : int) : vector<pair<int, int>>
+GetTemporalType(in node : int) : dsl_temporalType
+RemoveAllTemporalInfo() : int
+RemoveTemporalArc(in parent : int, in child : int, in order : int) : int
+SetNumberOfSlices(in slices : int) : int
+SetTemporalType(in node : int, in type : dsl_temporalType) : int
+TemporalRelationExists(in parent : int, in child : int, in order : int) : bool
+UnrollNetwork(out unrolled : DSL_network) : int

Figure 8.11: New methods and properties of the `DSL_network` class.

8.3.2 `DSL_nodeEntry`

This is the class that holds all the information of a node. It is invisible for the user, but all communication between the network and the nodes and between nodes themselves is managed through an array of `DSL_nodeEntry` objects. The `DSL_nodeEntry` class is extended with a `dsl_temporalType` property and a `DSL_temporalArcs` property. The `DSL_temporalArcs` is NULL by default. These properties are set through methods in the `DSL_network` class.

DSL_nodeEntry
+temporalArcs : DSL_temporalArcs
+temporalType : dsl_temporalType

Figure 8.12: New methods and properties of the `DSL_nodeEntry` class.

8.3.3 `DSL_temporalArcs`

A temporal arc is defined as an arc from a parent to a child with a certain temporal order larger than zero. The new `DSL_temporalArcs` class is a property of `DSL_nodeEntry` that keeps track of the temporal relations. `DSL_nodeEntry` is the most logical place to hold this property, because it already keeps track of the static relations between parent and child nodes. The `DSL_temporalArcs` property is NULL by default to reduce overhead for static networks and nodes. The `DSL_network` class is responsible for the consistency of the temporal arcs of the `DSL_temporalArcs` property.

DSL_temporalArcs
-children : vector<pair<int, int>>
-parents : vector<pair<int, int>>
+AddChild(in child : int, in order : int)
+AddParent(in child : int, in order : int)
+CleanUp()
+FindPositionOfChild(in child : int, in order : int) : int
+FindPositionOfParent(in parent : int, in order : int) : int
+GetChildren() : vector<pair<int, int>>
+GetParents() : vector<pair<int, int>>
+GetMaximumOrderOfChild() : pair<int, int>
+GetMaximumOrderOfParents() : pair<int, int>
+HasChild(in child : int, in order : int) : bool
+HasParent(in parent : int, in order : int) : bool
+IsEmpty() : bool
+RemoveChild(in child : int, in order : int) : int
+RemoveParent(in parent : int, in order : int) : int

Figure 8.13: The new `DSL_temporalArcs` class.

8.3.4 DSL_dbnUnrollImpl

When a decision maker performs inference on a temporal network, the `DSL_network` class creates a temporary object of the `DSL_dbnUnrollImpl` class. This class takes the temporal network, unrolls it, copies evidence, performs inference, and copies the posterior beliefs back to the temporal network. The decision maker is not aware of this process, the `DSL_dbnUnrollImpl` is completely hidden from the outside. The purpose of this class is to provide an object-oriented way to several temporal inference techniques, because in the future, more temporal inference techniques will be implemented.

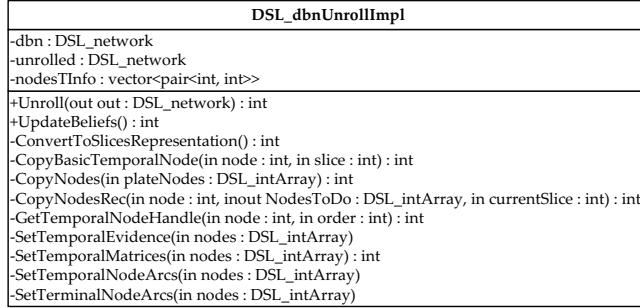


Figure 8.14: The new `DSL_dbnUnrollImpl` class.

8.3.5 DSL_node

The `DSL_node` class is extended with two methods to provide easy access to the temporal definition and the temporal values.

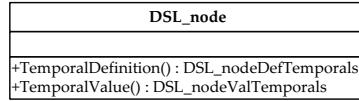


Figure 8.15: New methods of the `DSL_node` class.

8.3.6 DSL_nodeDefinition

This is a class with mainly virtual methods that are implemented by its subclasses. Each subclass represents a certain conditional probability distribution. Every node definition gets a property that holds its temporal definition. This property is NULL by default and is instantiated by calling the (virtual) `EnsureTemporalsExist()` method and can be accessed by the (virtual) `GetTemporals()` method. The `DSL_nodeDefinition` class implements three methods that synchronize children of nodes that change their definition. These methods, recognized by the `Common` prefix, needed to be adapted to provide for synchronization of the temporal definition as well. The subclasses that inherit from `DSL_nodeDefinition` are discussed below.

DSL_cpt This class is extended with the `DSL_temporalCpts` property and according methods. The most important changes within this class are a result of the need to keep the static definition in synchronization with the temporal definition. Several methods needed to be extended to provide this.

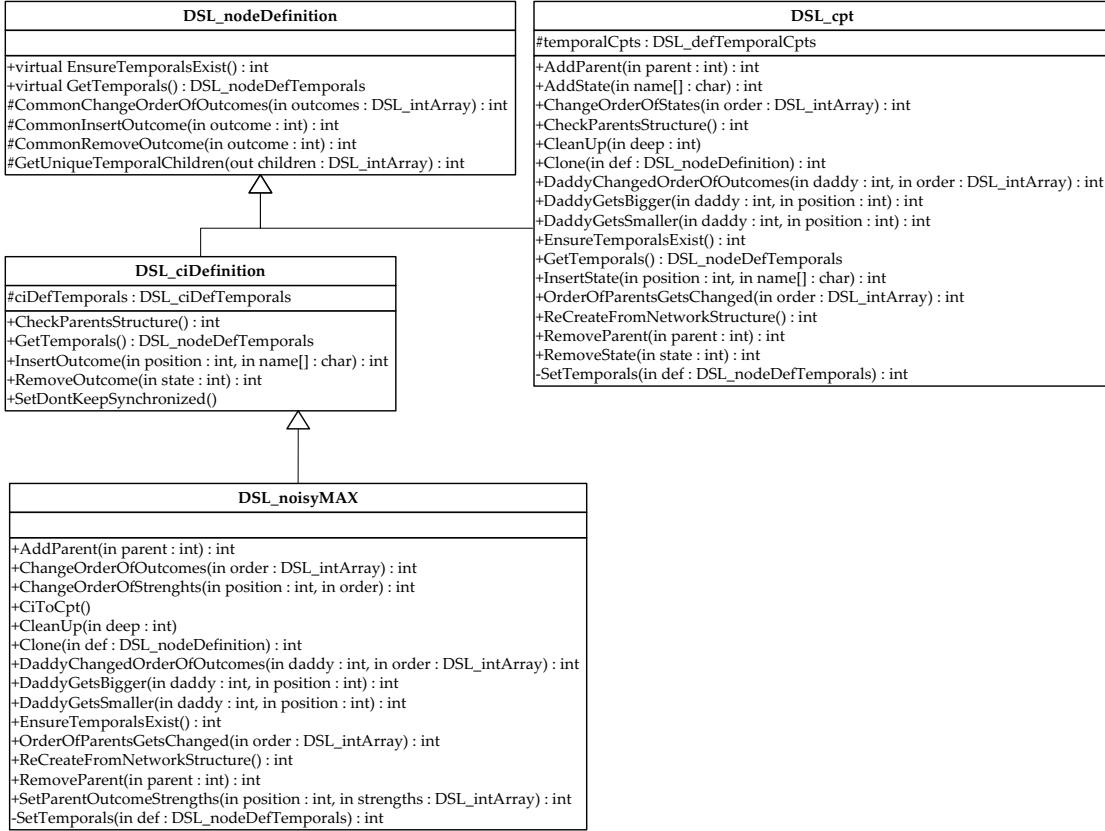


Figure 8.16: New and altered methods and properties of the *DSL_nodeDefinition*, *DSL_cpt*, *DSL_ciDefinition*, and *DSL_noisyMAX* classes.

DSL_ciDefinition This is the superclass for canonical gate definitions. It implements some of the virtual methods of **DSL_nodeDefinition**, but most functionality is implemented by its subclasses. The **DSL_ciDefinition** class is extended with the **DSL_ciDefTemporals** property and the appropriate methods. Some methods needed to be adapted to keep the **DSL_ciDefTemporals** property synchronized.

DSL_noisyMAX This class implements the canonical gate based on the NoisyMAX representation. An important extra data structure that needed to be taken into account is the vector that holds the causal order of the parent strengths. Again, several methods needed to be extended to account for synchronization with the temporal definition.

8.3.7 **DSL_nodeValue**

Just like **DSL_nodeDefinition**, this class contains mainly virtual methods that are implemented by its subclasses. Each subclass represents a certain posterior belief representation. Every node value implementation gets a property that holds its temporal values. This property is NULL by default and is instantiated by calling the (virtual) **SetTemporals()** method. This call is realized in the case where we are dealing with a temporal network and the number of time-slices is set in the **DSL_network** class. The temporal values can be accessed by the (virtual) **GetTemporals()** method. The **DSL_beliefVector** subclass that inherits from **DSL_nodeValue** is discussed below.

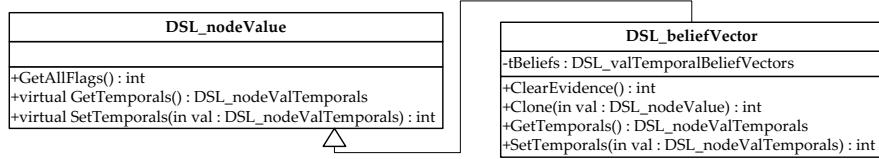


Figure 8.17: New and altered methods and properties of the `DSL_nodeValue` and `DSL_beliefVector` classes.

DSL_beliefVector This class is extended with the `DSL_valTemporalBeliefVectors` property and according methods. The most important changes within this class are a result of the need to keep the static values in synchronization with the temporal values. A couple of methods needed to be extended to provide this.

8.3.8 `DSL_nodeDefTemporals`

The `DSL_nodeDefTemporals` class is the superclass for the temporal part of the node definition. It contains mainly virtual methods that are implemented in its subclasses. Unfortunately, because of historical reasons in the development of SMILE, some methods are redundant. For instance, `State` and `Outcome` refer to the same functionality. To keep the methods in the temporal definition consistent with the methods in the node definition, we decided to deliberately copy this redundancy, but we know this goes against every software engineering principle. A large part of the methods in the `DSL_nodeDefTemporals` class is needed to keep the node definition in synchronization with the temporal definition, only a subset of the methods is needed for the management of the temporal definition itself.

DSL_defTemporalCpts This class implements the virtual methods that are declared in `DSL_nodeDefTemporals` and gets a property that stores the temporal CPTs.

DSL_ciDefTemporals This class implements some of the virtual methods that are declared in `DSL_nodeDefTemporals` and adds general functionality for temporal canonical gates.

DSL_noisyMAXTemporals This class implements the functionality for the temporal Noisy-MAX gate.

8.3.9 `DSL_nodeValTemporals`

The `DSL_nodeValTemporals` class is the superclass for storing the temporal evidence and posterior beliefs. It contains methods for administration of the temporal flags for every time-slice. It also declares virtual methods that are implemented by its subclasses. At the moment, only the `DSL_valTemporalBeliefVectors` subclass is implemented, because this class is compatible with the CPT and Noisy-MAX representations.

DSL_valTemporalBeliefVectors This class implements the virtual methods that are declared in `DSL_nodeValTemporals`. It has one property that contains the temporal evidence or beliefs for every time-slice of the temporal network.

8.4 Testing

Quality assurance is a very important aspect of extending GeNIe and SMILE with new functionality. Both products are of industrial quality regarding robustness and performance. This means

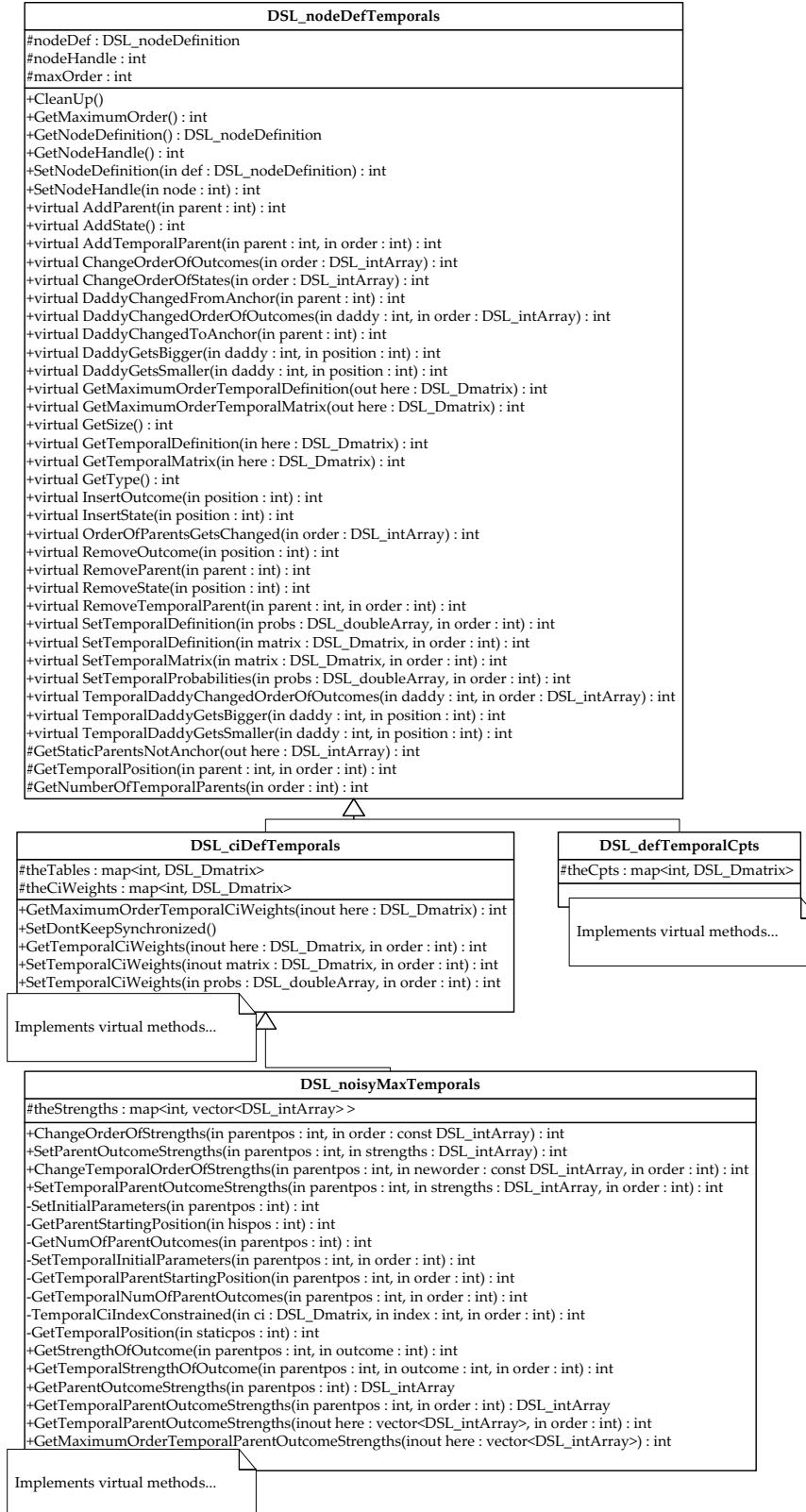


Figure 8.18: The new `DSL_nodeDefTemporals`, `DSL_defTemporalCpts`, `DSL_ciDefTemporals`, and `DSL_noisyMAXTemporals` classes.

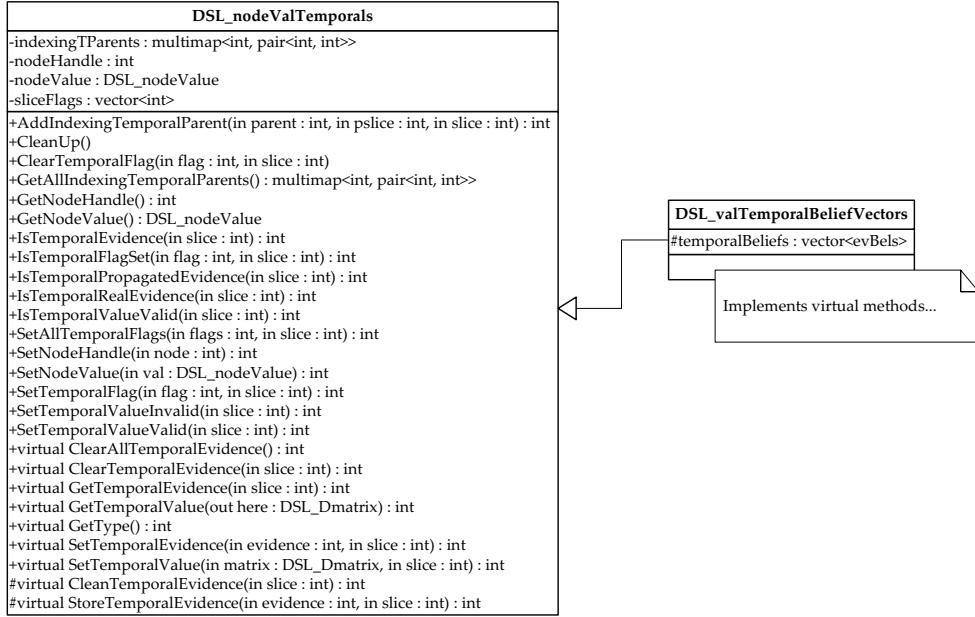


Figure 8.19: The new `DSL_nodeValTemporals` and `DSL_valTemporalBeliefVectors` classes.

that errors discovered by end users need to be minimized. An important aspect of quality assurance is a well-balanced testing methodology. Thorough testing is essential for the temporal reasoning extension before a public version gets released. This section elaborates on the testing methodology followed while implementing temporal reasoning.

8.4.1 Inspecting components

During implementation of temporal reasoning in SMILE, the source code of the implemented classes was reviewed to find potential errors. These inspections were performed manually and took place in turns with unit testing. Next to finding errors, an important goal of this exercise was to maximize quality and efficiency of the delivered source code. Also, the returned error codes were reviewed and the use of assertions was encouraged to enlarge the maintainability of the source code.

8.4.2 Unit testing

Unit testing focuses on the building blocks of the software system. Because of time constraints, not every class is tested individually, but several classes were grouped together to form coherent blocks that could be tested individually. For SMILE, this resulted in the seven subsystems shown in figure 8.20. A library of small test programs was written to test the majority of methods in the subsystems. Several methods were not tested directly. If a method that is composed of several sub-methods behaved correctly, it was assumed that the sub-methods behaved correctly as well. The small test programs implemented the following three test methods:

1. **Boundary testing** Examples of boundary testing include: testing temporal networks with zero time-slices, testing temporal arcs of order zero, retrieving posterior beliefs for non-existing time-slices, etc.
2. **Path testing** Object-oriented languages pose some difficulties for path testing in comparison to imperative languages, because of polymorphism of objects, interconnectivity of objects, and many small methods instead of several large methods. However, path testing

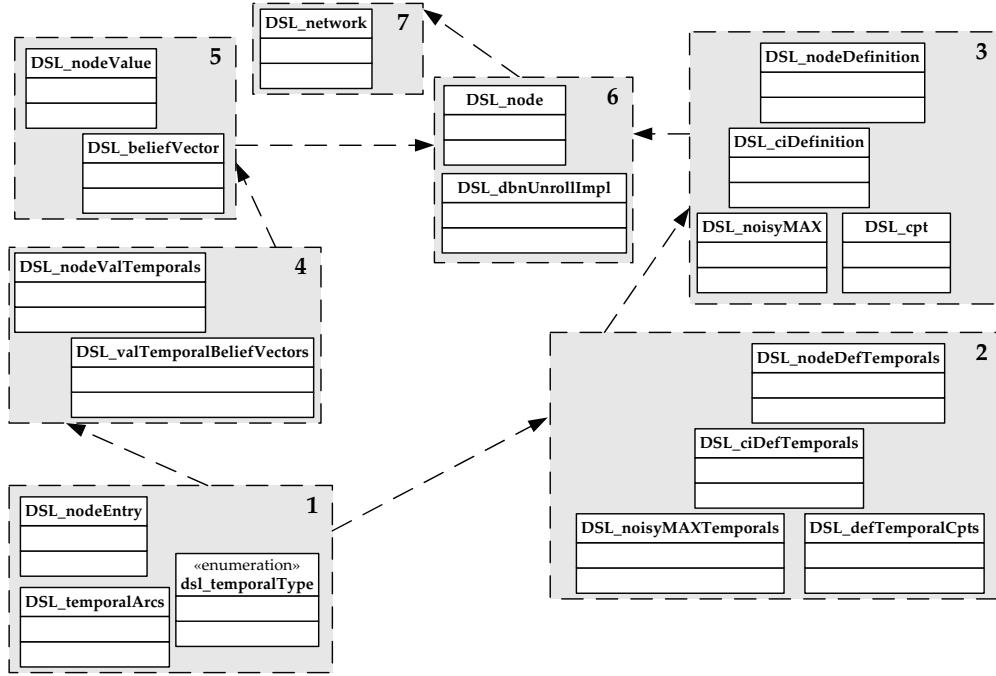


Figure 8.20: The implementation of temporal reasoning in SMILE decomposed into seven subsystems. Each subsystem was tested individually during unit testing. The arrows depict the order in which the subsystems were connected during the integration testing. All subsystems are connected during integration testing when testing subsystem 7 `DSL_network`.

can still be useful. We performed path tests for large methods in the `DSL_network` and `DSL_dbnUnrollImpl` classes, such as setting the temporal type of a node, updating the beliefs, or adding temporal arcs. Several of these tests were repeated during integration testing, because of the relation between the subsystems.

3. **State-based testing** Throughout the testing phase, specialized debug methods printed the state of the accessed objects to the standard output after each step. These methods gave insight into the states of tested objects and were used to test whether state-transitions of those objects behaved correctly.

Figure 8.20 shows the seven subsystems that were tested during unit testing. The arrows depict the order in which the subsystems were connected during the integration testing, which is described next.

8.4.3 Integration testing

During integration testing, components are integrated into larger components and tested together. During this phase, errors are detected that could not be detected during unit testing, such as faults associated with calling the interfaces. Figure 8.20 shows how the seven subsystems are integrated in several steps to ultimately form the SMILE subsystem with temporal reasoning. This is a form of bottom-up testing where the subsystems were cumulatively tested in the following order:

- **Cumulative subsystem I:** double test {1, 2}, triple test {1, 2, 3},
- **Cumulative subsystem II:** double test {1, 4}, triple test {1, 4, 5},
- **Cumulative subsystem III:** double test {6, I}, double test {6, II}, and
- **SMILE subsystem:** double test {III, 7}.

Performing these tests results in the fully tested SMILE subsystem. This subsystem in its turn needed to be integration tested with the two other subsystems that were implemented: SMILEXML and GeNIE. Figure 8.21 shows how the three top layer subsystems are integrated and tested.

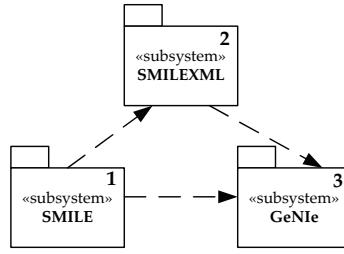


Figure 8.21: Integration testing for SMILE, SMILEXML, and GeNIE.

8.4.4 System testing

System testing focuses on the complete system. It ensures that the complete system complies with the functional and non-functional requirements stated in section 8.1. We decided to test the use cases, performance and robustness of the implementation of temporal reasoning in SMILE and GeNIE.

Application testing Throughout this research the implementation in SMILE and GeNIE was tested by applying it to the two problems in the medical domain that we modeled.

Functional testing The functional testing was completely based on the use cases derived in section 8.1. Testing all possible valid and invalid inputs for all use cases is not feasible and thus a subset of inputs was identified that were likely to cause failures. The tests with this subset of inputs was performed on both GeNIE and SMILE.

Performance testing Performance testing was conducted to investigate the software's behavior with large networks (either because the temporal network had many variables, many time-slices, or many states per node), temporal networks with a high in-degree, and temporal networks that had many d-connected³ variables. The system was tested for both memory- and time-constraints.

Pilot testing GeNIE was tested by a selected set of users. For this test, several DSL members that did not contribute to the development of temporal reasoning in GeNIE were invited to play with the extension and give their feedback. This was also the first time that a working GUI was presented to users from Philips. Pilot testing turned out to be very useful, because the pilot users where using the program without any theoretical background whatsoever. Several potential errors that we did not think of could be caught because of this.

³Two nodes are d-connected if and only if they are not d-separated.

8.4.5 Testing results

The most important errors where discovered during inspection of the components and unit testing. During these testing activities several errors due to wrongly saved CPTs, bad calculation of the size of evidence and/or belief vectors, and copying of evidence to non-existing time-slices were corrected.

Furthermore, the inference results were corrected by turning off relevance reasoning during the unrolling phase before the actual inference and during the transfer of beliefs from the unrolled network to the temporal network after the actual inference. During the actual inference, relevance reasoning was switched on. Switching off relevance reasoning also slightly improved the performance.

Another important test result is that the addition of temporal reasoning does not significantly influence the performance of the functionality that already existed. Next to that, it seems that the separation of the temporal functionality and the existing functionality worked out fine, because none of the errors that were found affected the existing functionality.

8.5 Summary

This chapter presented the requirements, design, implementation, and testing of temporal reasoning in SMILE and GeNIE. The result of this effort can be downloaded from <http://genie.sis.pitt.edu>. Designing, implementing, and testing temporal reasoning in SMILE and GeNIE proved to be a time-consuming process. However, it was required to be able to use and test the extended DBN formalism, because no other software package for temporal reasoning exists that provides this functionality. To the best of our knowledge, our implementation of temporal reasoning in SMILE and GeNIE is the first that provides this much freedom in temporal modeling, is easy to use, and introduces the concept of canonical models in temporal networks. In part III of this thesis, the extended DBN formalism and its implementation will be applied to two problems in the medical domain: glucose-insulin regulation and the cardiovascular system, but first, deriving the DBN structure and parameters with the extended DBN formalism will be discussed in the next chapter.

Chapter 9

Modeling with a DBN

A DBN consists of a qualitative part (the structure) and a quantitative part (the parameters). Both can be obtained by machine-learning techniques, expert elicitation or a combination. In our approach, the DBN structure is obtained by a combination of expert knowledge in the form of interviewing domain experts and by looking at existing models (if they exist). The DBN parameters are learned from a data set. In this chapter our approach for obtaining the DBN structure and learning the DBN parameters of the extended DBN formalism is discussed. Furthermore, some issues related to the discretization of continuous data are presented.

9.1 DBN structure

There exist many methods for obtaining the DBN structure, including various machine-learning and expert elicitation techniques. We choose to obtain the DBN structure by a combination of expert elicitation in the form of domain expert interviews and by investigation of existing models, such as systems of ODEs presented in chapter 2. Note that for many physiological processes, the exact interactions between variables is not known and satisfying models do not exist yet. In this case, the network modeler has to rely on expert knowledge and/or machine-learning techniques only.

For several physiological processes, systems of ODEs exist that describe the interactions between variables. This means that we can get an idea of the underlying reality if we investigate these interactions. However, investigating these interactions are not enough to obtain knowledge about a complete model of the real world. For instance, a system of ODEs does not provide a natural support for modeling different classes of patients (or the *context*). Also, the interactions between variables of a system of ODEs are based on a certain time-granularity. It is not so obvious what remains of the interactions between its variables if the time-granularity is

changed several orders of magnitude. However, when modeling physiological processes with a DBN, the modeled time-granularity will often be a result of the needs of the field expert and not of the time-granularity prescribed by existing models. For instance, a physician is typically not interested in what happens every couple of milliseconds, but it is interested in what will happen the next minute.

An example of the difference in DBN structures as a result of changing the time-granularity is shown in figures 9.1 and 9.2, where two different DBN models for the cardiovascular system are shown, one with a time-granularity of 50[ms] and one with a time-granularity of 3[s]. Note that when enlarging the time-granularity, it might not even be appropriate to devise a DBN model, maybe a non-temporal BN is sufficient. Also note that by changing the time-granularity, different properties of the modeled variables can become important. Figures 9.1 and 9.2 show that when modeling the pressure of the systemic arteries (P_{sa}) for a time-granularity of 50 [ms], the course of the variable is sufficient. However, when modeling P_{sa} for a time-granularity of 3 [s], a separation is needed between the systolic and diastolic pressure. The meaning of the majority of the variables in these models is already explained in chapter 2 and will be repeated in chapter 11, where the full DBN model for a time-granularity of 3[s] will be explained.

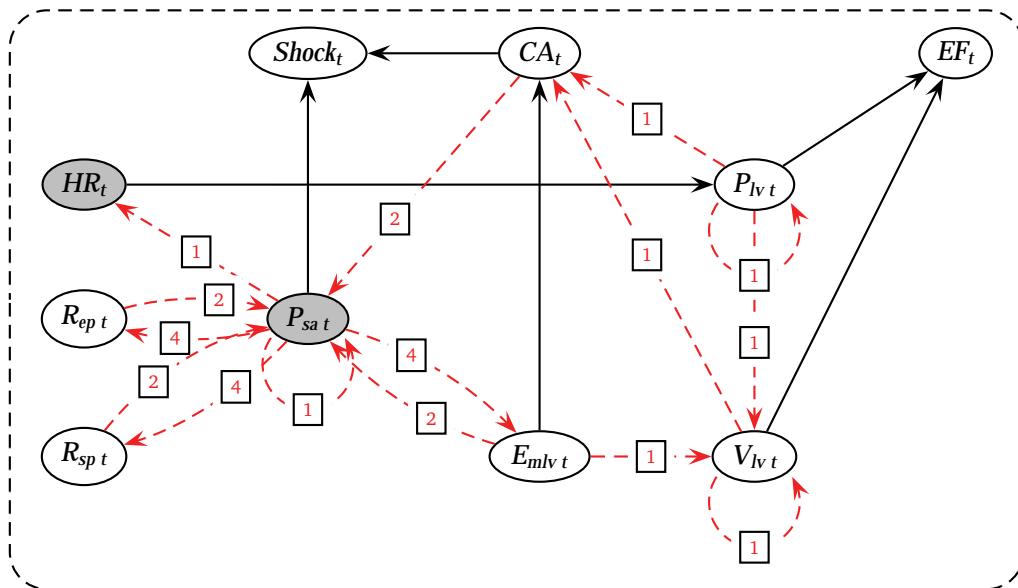


Figure 9.1: The DBN model of the cardiovascular system for a time-granularity of 50[ms].

Another consideration of using existing models is that when modeling physiological processes with a DBN, in most cases only a subset of the variables in existing models are interesting. Investigating the interactions between variables in existing models becomes difficult in these cases, as many interactions will change due to omitting certain variables.

Despite the above-mentioned considerations, a part of the DBN structure can be obtained from existing models, but this has to be done with caution and with the underlying reality in mind. For instance, some dependencies in a DBN can be obtained by investigating the terms and factors that ODEs consist of. If the equation for dY/dt contains terms and/or factors with X and Y , we have to take into account the possibility of temporal arcs between $Y_{t-1} \rightarrow Y_t$ and $X_{t-1} \rightarrow Y_t$.

Furthermore, one can choose to purposely leave out some interactions between variables in the DBN to reduce computational complexity by investigating the parameters of existing models. This can be useful in situations where many variables and dependencies need to be modeled, but certain dependencies do not have a large influence. The modeler can choose to leave these dependencies out of the DBN, because their influence on the final result is minimal. The possibility of this approximation varies from application to application.

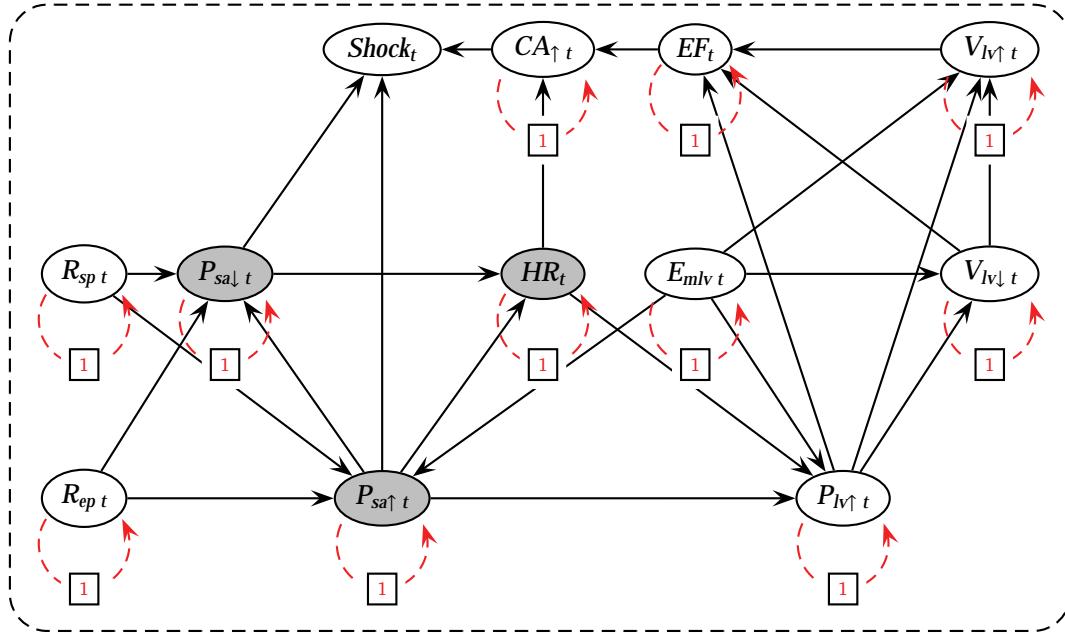


Figure 9.2: The DBN model of the cardiovascular system for a time-granularity of 3[s].

When modeling a physiological process with a DBN, one has to keep in mind that the resulting DBN model must not be limited by the assumptions of existing models. The DBN formalism is very powerful, providing natural ways of modeling phenomena that other techniques might lack. An example of this is our extension of the DBN formalism with contemporaneous nodes that can be introduced to model the *context* of a variable. These contemporaneous nodes can serve as an index of a child in the temporal plate to distinguish between different contexts. Consider the glucose-insulin regulation that was introduced in chapter 2. In this case, we have three classes of patients: diabetic type-1, diabetic type-2 and normal. To be able to create a DBN model, we could use the system of ODEs as a tool to simulate patients by solving the ODEs with different parameters. (In fact, this is exactly what we will do in chapter 10.) Every simulated patient gets slightly different ODE parameters compared to patients in the same class, but the ODE parameters of patients in different classes are very different. The contemporaneous node can serve as a classifier when inferring a patient whose class is unknown, or can reduce the error of the inferred values when the patient class is known. Note that this contemporaneous node does not follow from the system of ODEs as such, only from the underlying reality where the system of ODEs is a result of.

A great deal has been written about the connection between graphical models and other modeling techniques, such as systems of ODEs. Interested readers are directed to: [Das03, DDN00, LPKB00]. In part III of this thesis we will derive the DBN structures for two physiological processes and discuss the modeling issues involved.

9.2 DBN parameters

In section 3.2 we stated that learning parameters of DBNs is generally the same as learning parameters of BNs with a few small differences. Because we want to perform parameter learning for our extension of the DBN formalism, we need to investigate what specific differences hold.

9.2.1 Considerations

Parameters must be tied across time-slices, so models of unbounded length can be modeled. This is easily achieved by pooling the expected sufficient statistics for all variables that share the same parameters. However, parameter tying is not needed with the method that we employ for learning with complete data, because we do not unroll the DBN explicitly and thus do not have to tie the CPDs in every time-slice.

In the case that the initial parameters π represent the stationary distribution of the system, they become coupled with the transition model. As such, they cannot be estimated independently of the transition matrix. The parameters π for $p(\mathbf{X}_1)$ can be taken to represent the initial state of the dynamic system. If such an initial state distribution is known, π can be learned independently from the transition distribution $p(\mathbf{X}_t|\mathbf{X}_{t-1})$. However, if this is not the case, [Mur02] proposes that we should follow the algorithm presented in [Nik98]. This algorithm starts with an initial guess for the priors in the first time-slice, does belief updating, and reads off the values of the state variables in the second time-slice. These values are transferred back to the first time-slice, and the process repeats until the distributions in the two time-slices converge to the same value. Note that this algorithm only works for first-order DBNs.

Linear-Gaussian DBNs sometimes have closed-form solutions to the maximum likelihood estimates. Since SMILE does not support continuous-valued variables yet, this does not need to be considered in this research.

9.2.2 Parameter learning method for the extended DBN formalism

Current DBN parameter learning algorithms for Murphys's formalism unroll a DBN for $t = T$ time-slices, where T is the sequence length of the process, after which the learning data is inserted in the unrolled network and the parameters are learned. Remember that for Murphys's formalism, two sets of parameters need to be specified, the initial parameters in B_1 and the transition parameters in the second slice of B_{\rightarrow} . An example for learning a DBN model of a HMM is shown schematically in figure 9.3. In this example, the parameters for B_1 are given by $\{\Theta_{X_1}, \Theta_{Y_1}\}$ and the parameters for B_{\rightarrow} are given by $\{\Theta_{X_t}, \Theta_{Y_t}\}$. When learning the parameters, the DBN is unrolled for T time-slices, the parameters are tied and learned by insertion of the learning data. However, in case the initial parameters represent the stationary distribution of

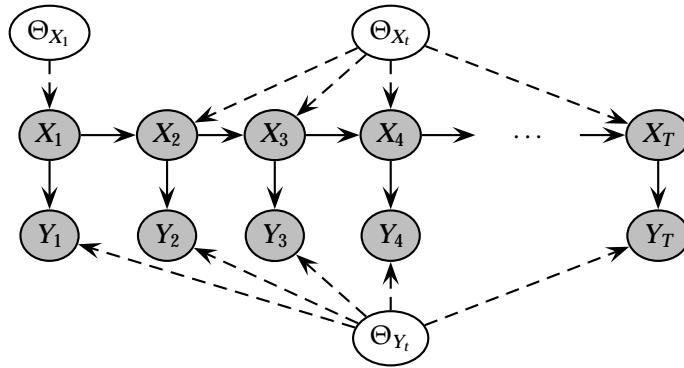


Figure 9.3: Schematic representation of unrolling the DBN for learning of its parameters with decoupled initial and transition parameters.

the system, they become coupled with the transition parameters. This is shown schematically in figure 9.4. On a modeling level, decoupled initial and transition parameters mean that the modeled process has a specific beginning, i.e. the first time-slice of the DBN always models the

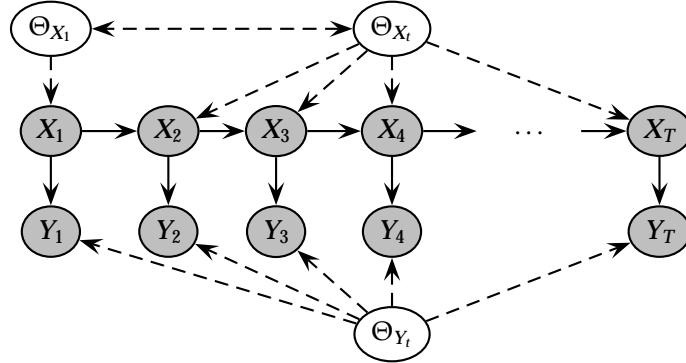


Figure 9.4: Schematic representation of unrolling the DBN for learning of its parameters with coupled initial and transition parameters.

same time instant of the process. The initial and transition parameters become coupled when the modeled process does not have a specific beginning, i.e. the first time-slice of the DBN starts somewhere in the modeled process, but where is unclear. When modeling physiological processes, we are dealing with the latter, because a patient does not have a distinctive initial state. As we mentioned already in the previous section, learning DBN parameters in case the initial parameters represent the stationary distribution of the system can be problematic. For first-order DBNs, an algorithm is presented in [Nik98], but this algorithm cannot be used with our extension of the DBN formalism. Furthermore, unrolling the DBN at every learning iteration seems unnecessary, and that's why we decided to develop our own method for learning the DBN parameters.

We have developed a straightforward method that enables us to learn the parameters of the extended DBN formalism with a given structure using standard BN learning algorithms. Our method is based on the fact that a DBN only has a limited number of CPDs that need to be learned and that it is possible to decompose the DBN definition into several BNs that can be learned independently. With our method it is possible to learn the parameters for DBNs where the initial and transition parameters are either coupled or decoupled.

Remember that from equation 7.4 follows that we need to specify four sets of parameters for the extended DBN formalism: the initial parameters $\{p(\mathbf{A}|\mathbf{C}), p(\mathbf{Z}_1|\mathbf{Pa}(\mathbf{Z}_1), \mathbf{A}, \mathbf{C})\}$, the temporal parameters $p(\mathbf{Z}_t|\mathbf{Pa}(\mathbf{Z}_t), \mathbf{C})$, the contemporaneous parameters $p(\mathbf{C})$, and the terminal parameters $p(\mathbf{T}|\mathbf{Z}_T, \mathbf{C})$. Our method makes use of the fact that these four sets can be learned independent of each other and that the resulting decomposition of the DBN definition can be learned with existing BN learning algorithms. Note that it is often possible to merge one or more of these parameter sets. The general outline of our method is as follows:

1. The DBN definition is unrolled for $k + 1$ time-slices where k denotes the temporal order of the Markov process. The DBN needs to be unrolled for this number of time-slices, because in the resulting unrolled network every CPD that needs to be learned will be represented with at least one copy.
2. The unrolled DBN is decomposed into four smaller BNs. The resulting BNs jointly contain exactly one copy of every CPD that needs to be learned. If an already learned variable is needed to index a variable that is not learned yet, its parameters are fixed. The decomposition consists of:
 - A BN that defines the initial parameters $p(\mathbf{A}|\mathbf{C})$ and $p(\mathbf{Z}_1|\mathbf{A}, \mathbf{C})$.
 - If the initial distribution of the DBN represents the initial state of the dynamic system, the variables in the first time-slice need to be defined as anchor variables. In this case, only a network with $p(\mathbf{A}|\mathbf{C})$ needs to be learned.
 - If the initial distribution of the DBN represents the stationary distribution of the dynamic system, the coupling of the initial distribution and the transition

distribution is obtained by defining the variables as the first time-slice in the plate. In this situation, the BN contains both $p(\mathbf{A}|\mathbf{C})$ and $p(\mathbf{Z}_1|\mathbf{Pa}(\mathbf{Z}_1), \mathbf{A}, \mathbf{C})$.

- A BN that defines the (temporal) parameters within the plate $p(\mathbf{Z}_t|\mathbf{Pa}(\mathbf{Z}_t), \mathbf{C})$.
 - In this BN, all the variables with tied parameters are deleted except for the ones that are not learned yet or needed to provide an index for the variables that still need to be learned.
 - A BN that defines the contemporaneous parameters $p(\mathbf{C})$.
 - A BN that defines the terminal parameters $p(\mathbf{T}|\mathbf{Z}_T, \mathbf{C})$.
3. (Optional) Discretize the time-series data using a proper discretization method, more on this in section 9.3.
 4. Load the (discretized) time-series data and learn the parameters of the BNs by setting prior distributions and running a BN learning algorithm accordingly.

The presented learning method enables us to learn the parameters of the extended DBN formalism without completely unrolling the network. Also, expert knowledge can be easily incorporated by setting the prior distributions and an equivalent sample size (to denote the confidence of the expert knowledge) before learning the parameters from a dataset.

An example of the decomposition of a DBN model is shown in figure 9.5. The definition of the DBN is shown in figure 9.5a, where we can see that we are dealing with a second-order Markov process. Remember that each node has a different CPD for every incoming arc with a different temporal order. For this DBN, we need to specify nine CPDs: $\{A, C, T, U_1, X_1, Y_1, X_{t+1}, Y_{t+1}, X_{t+2}\}$. To obtain at least one copy of each CPD that needs to be learned, we unroll the DBN to three time-slices. The unrolled version is shown in figure 9.5b. The unrolled version is decomposed into three smaller BNs that can be learned independently (9.5c, 9.5d, and 9.5e). Because of the decomposition, each CPD is learned only once. In this example, the initial and contemporaneous BNs are merged to obtain the initial network in figure 9.5c. Furthermore, during learning of the transition network in figure 9.5d, the CPDs need entries from CPDs that were already learned in the initial network. These CPDs are gray, which in this case means that their parameters are fixed and that they are only used as an index for the CPDs that are being learned.

9.3 Discretization of continuous data

Using continuous variables in a DBN is often problematic due to complexity issues. In addition, SMILE does not support the use of continuous variables yet. Because of this, discretization of continuous data is needed before the DBN parameters can be obtained using machine-learning techniques. Discretization can be defined as follows [Har01]:

Definition (Discretization)

A *discretization* of a real-valued vector $\mathbf{v} = (v_1, \dots, v_n)$ is an integer-valued vector $\mathbf{d} = (d_1, \dots, d_n)$ with the following properties:

1. Each element of \mathbf{d} is in the set $\{0, 1, \dots, k - 1\}$ for some (usually small) positive integer k , called the degree of the discretization.
2. For all $1 \leq i, j \leq n$, we have $d_i \leq d_j$ if $v_i \leq v_j$.

In current research, discretization of continuous time-series is often performed as an unsupervised preprocessing step. The choice of the method and accompanying parameters are rarely justified [MU05]. However, for field experts it is of great importance that the resulting interval boundaries are meaningful within the domain, and that is why we will discuss some discretization methods and their considerations.

There are three different dimensions by which discretization methods can be classified: *global* vs. *local*, *supervised* vs. *unsupervised* and *static* vs. *dynamic*. Local methods produce

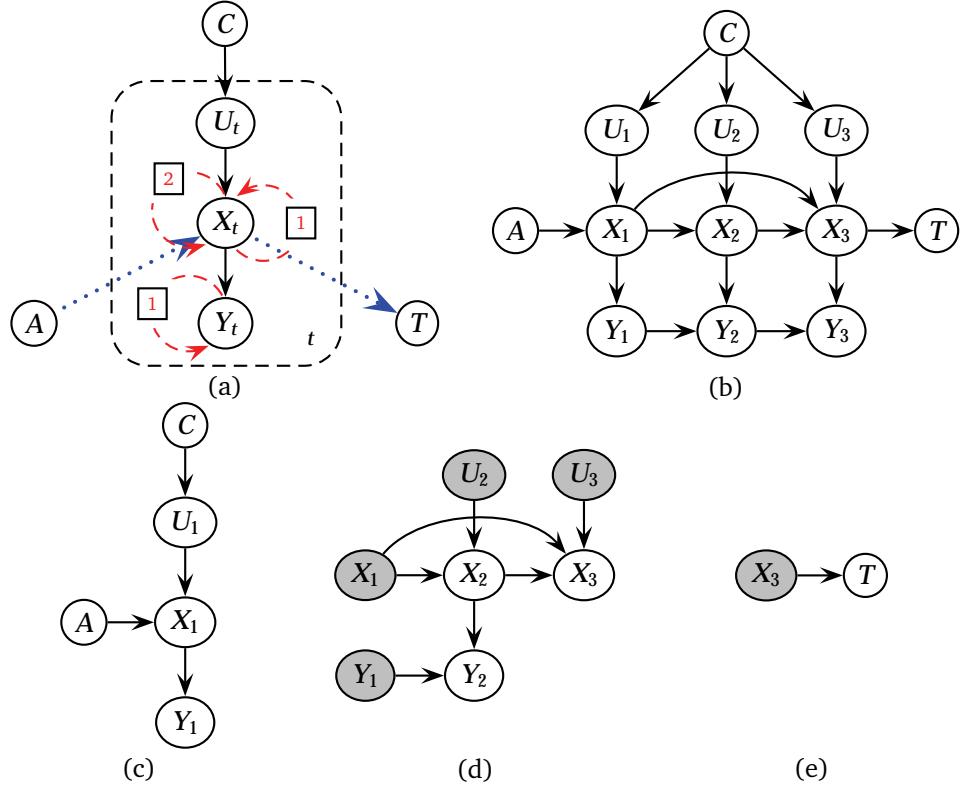


Figure 9.5: (a) A DBN definition. (b) The DBN definition unrolled for $k+1$ order time-slices. (c), (d), and (e) The decomposed network. Note that in this specific example, the initial and contemporaneous network are merged to obtain network (c).

partitions that are applied to localized regions of the instance space. Global methods produce a *mesh* over the entire continuous instance space, where each variable is partitioned independent of the other variables. Unsupervised methods do not make use of instance labels in the discretization process, where supervised methods do. Many discretization methods require a given k as an indication of the maximum number of intervals to produce. These methods are called static, in contrast to dynamic methods where such a parameter does not have to be specified [DKS95].

We found that currently a great deal of research on discretization methods is going on, but also found that three discretization methods are most common for discretization of time-series data: uniform-sized intervals, uniform-sized binning, and single-link clustering. We decided to investigate the performance of these three discretization methods by implementing them in Matlab and applying them to the continuous-valued glucose and insulin variables from the glucose-insulin regulation model. But first, the three discretization methods are presented.

9.3.1 Uniform-sized intervals

The simplest discretization method is *uniform-sized intervals*. This is an unsupervised, global, and static discretization method that merely divides the range of the data set \mathbf{v} into k equal sized intervals, the interval width ω is obtained by:

$$\omega = \frac{\max(\mathbf{v}) - \min(\mathbf{v})}{k} , \quad (9.1)$$

and the discretization thresholds τ are obtained by:

$$\tau_j = \min(\mathbf{v}) + j \cdot \omega , \quad (9.2)$$

where $j = 1, \dots, k - 1$ and k is provided by the user. In most cases, the selection of a suitable k is an ad-hoc event, because its ideal value is a combination of the range of the variable, the complexity of the model and the required precision. A downside of this method is that a general understanding of the range is needed, else the method is vulnerable to outliers. Furthermore, skewed data is not handled well. An advantage of this approach is that it is very easy to implement and use if a general understanding of the range is available.

9.3.2 Uniform-sized binning

A discretization method that is slightly more complex is *uniform-sized binning*. This is also an unsupervised, global, and static discretization method that divides the data set \mathbf{v} into k intervals, each interval containing approximately the same number of samples. In case of a data set \mathbf{v} of n samples, the size of the sample sets $s = s_1 = \dots = s_k$ for k intervals can be obtained by:

$$s_1 \approx \dots \approx s_k \approx \frac{n}{k}, \quad (9.3)$$

and the discretization thresholds τ are obtained by ordering the samples in \mathbf{v} from low values to high values and obtaining the threshold by:

$$\tau_j = v_0 + v_{j \cdot n/k}. \quad (9.4)$$

Finding the correct thresholds is not as straightforward as it may seem, because the boundaries obtained by naively dividing \mathbf{v} into uniform-sized bins can have the effect that two or more intervals overlap. For our research, we implemented an algorithm that first sorts the data values in \mathbf{v} from low values to high values and calculates $\frac{n}{k}$, then creates a bin for every distinct value and then iteratively loops through these bins while adding them together until an optimal distribution of the samples over the bins is reached.

An optimal distribution is a solution where the sizes of all bins approximate the mean-size $\frac{n}{k}$ as closely as possible. That this is an optimal solution can be proven by showing that Shannon's entropy (the measure of how much information can be carried with a certain discretization) is maximized when splitting the range of observed values into uniform-sized bins. Shannon's entropy is given by:

$$H = - \sum_{i=1}^k p_i \log_2 p_i, \quad (9.5)$$

where k is the set of possible events and p_i denotes an individual probability of such an event. In the case of discretization of data set \mathbf{v} into k bins, Shannon's entropy is given by:

$$H = \sum_{i=1}^k \frac{s_i}{n} \log_2 \frac{n}{s_i}. \quad (9.6)$$

From equation 9.6 follows that an increase in the number of bins implies an increase in entropy, with an upper bound of $\log_2 n$. However, we do not want to maximize the entropy by enlarging the number of bins, but we want to maximize the entropy when the number of bins is given and fixed. To see that the entropy is maximized in the case of uniform-sized bins, imagine that we divide the data set into two bins, with one bin containing m samples and the other $n - m$ samples. In this case, Shannon's entropy can be calculated by:

$$H = \sum_{i=1}^2 \frac{s_i}{n} \log_2 \frac{n}{s_i} \quad (9.7)$$

$$= \frac{s_1}{n} \log_2 \frac{n}{s_1} + \frac{s_2}{n} \log_2 \frac{n}{s_2} \quad (9.8)$$

$$= \frac{m}{n} \log_2 \frac{n}{m} + \frac{n-m}{n} \log_2 \frac{n}{n-m}. \quad (9.9)$$

We can verify that the entropy reaches its maximum value over $0 < m < n$ at $m = \frac{n}{2}$. This result can be easily generalized for $k > 2$ bins and thus the maximum entropy is reached when we split the data set into uniform-sized bins.

9.3.3 Single-link clustering

The final method we will present here is a variant of single-link clustering (SLC). Normally, this is an unsupervised, global, and static discretization method that divides the range of the data set \mathbf{v} into k intervals, where k has to be specified by the user. However, we will discuss the variant presented in [DML05], where the ideal value for k is calculated using several criteria. Therefore, the discretization method is dynamic.

SLC is a divisive (top-down) hierarchical clustering that defines the distance between two clusters as the minimal Euclidean distance of any two real-valued entries belonging to different clusters. The algorithm starts from the entire data set and iteratively splits it until either a certain threshold is reached or every cluster only contains one element. SLC has one major advantage: very little starting information is needed, only distances between points. This may result in a discretization where most of the points are clustered into a single bin if they happen to be relatively close to one another. However, this disadvantage is countered in the variant of the algorithm described in [DML05]. This algorithm employs graph theory as a tool to produce a clustering of the data and provides a termination criterion that finds a trade-off between the number of bins and the amount of information, so k does not have to be specified by the user. An outline of the algorithm is given below:

Extended SLC discretization algorithm [DML05] The algorithm has a real-valued vector $\mathbf{v} = (v_1, \dots, v_n)$ as input and an integer-valued vector $\mathbf{d} = (d_1, \dots, d_n)$ as output. It consists of the following steps:

1. Construct a complete weighted graph G where each vertex represents a distinct v_i and the weight of each edge is the Euclidean distance between the incident vertices.
2. Remove the edge(s) of the highest weight.
3. Repeat step (2) until G gets disconnected into clusters C_j , $j = 1, \dots, k$.
4. For each cluster C_j , recursively disconnect further if one of the following criteria holds:
 - (a) $\text{avg.weight}(C_j) > \frac{\text{avg.weight}(G)}{2}$.
 - (b) $|\max(v) \in C_j - \min(v) \in C_j| \geq \frac{|\max(v) \in G - \min(v) \in G|}{2}$.
 - (c) $\min.\text{vertex.degree}(C_j) < \#\text{vertices}(C_j) - 1$.
5. Apply Shannon's information measure criterion to the largest C_j to check whether further disconnection is needed. If so, go to (6). Else, go to (7).
6. Sort the values of C_j and split them into two equally sized sets.
7. Sort the clusters C_j , $j = 1, \dots, k$ and enumerate them $0, \dots, k - 1$, where k is the number of components into which G got disconnected. For each $i = 1, \dots, n$, d_i is equal to the label of the cluster in which v_i is a vertex. The discretization thresholds can be obtained by finding the minimal and maximum vertex in every cluster.

9.3.4 Empirical study and results

We tested the performance of the three discussed discretization methods by implementing them in Matlab and applying them to the continuous-valued glucose and insulin levels from the glucose-insulin regulation model. The datasets for the glucose and insulin variables consisted of 333,000 records each. In figure 9.6, the two histograms of the variables are shown to gain insight into the range and distribution of their values. We decided to discretize the variables with the uniform-binning and -interval methods starting with $k = 10$ bins, and adding $k = 5$ bins until

$k = 100$ bins was reached. Remember that in most cases, the selection of a suitable k is an ad-hoc event, because its ideal value is a combination of the range of the variable, the complexity of the model and the necessary precision. We will see in chapter 10 that for the glucose-insulin regulation model, a discretization of $k = 30$ bins for the glucose and insulin variables provided a sufficient precision for our purposes. We decided to measure the performance using three

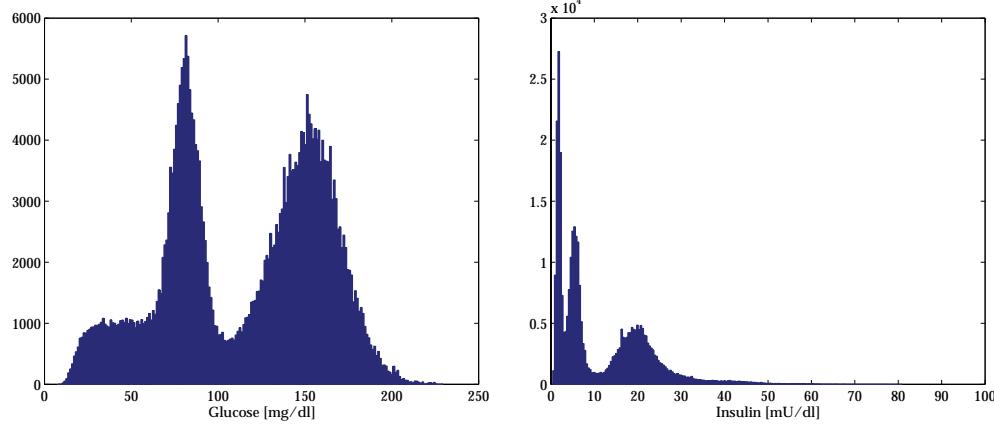


Figure 9.6: The histograms of the glucose and insulin variables.

different measures. The first measure is Shannon's information measure or *entropy*, which we discussed already and is given in equation 9.5. This measure is useful because it gives insight into how much information can be encoded in the discretization. The higher the entropy, the more information can be encoded in the discretization. The second measure we use is the *mean error* (ME):

Mean error The mean error or ME of an estimated vector \tilde{x} and the true vector x can be calculated as follows:

$$\text{ME} = \frac{1}{n} \sum_{i=1}^n |x_i - \tilde{x}_i| . \quad (9.10)$$

The mean error is an intuitive measure for gaining insight into the precision of the discretization. For this measure it is evident that the smaller the outcome, the better the results. The final measure we use is the *mean squared error* (MSE):

Mean squared error The mean squared error or MSE of an estimated vector \tilde{x} and the true vector x can be calculated as follows:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (x_i - \tilde{x}_i)^2 . \quad (9.11)$$

The MSE can also be used to gain insight into the precision of the discretization, but it is less intuitive, because the MSE has the property that larger errors give a proportionally larger MSE than smaller errors. For this measure also holds that the smaller the outcome, the better the results.

During the experiment we found that the SLC-based algorithm was way too slow for the amount of data that we needed to discretize. The SLC-based algorithm is ideal in situations where we do not have a large dataset and do not have insight into the range of the data. However, for our applications, we do have insight into the range of the data. That is why we decided to drop the SLC-based algorithm altogether and do not show its results in figure 9.7.

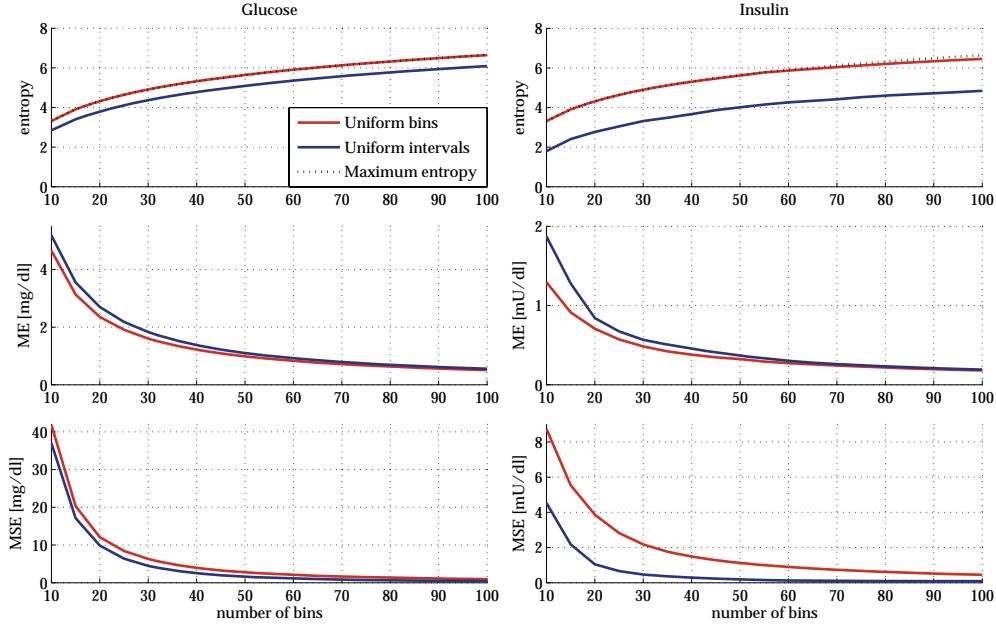


Figure 9.7: ME, MSE, and entropy of discretization of glucose and insulin variables.

From figure 9.7, we can notice a couple of things. First of all, the entropy of the uniform-binning method comes really close to its optimal value, where the entropy of the uniform-interval method stays behind. Note that for the calculation of entropy, the distribution and range of the data does not influence the outcome, only the number of bins and the number of samples in those bins are important. That is why the entropy for glucose and insulin are more or less the same for the uniform-binning method.

If we compare the ME and the MSE of each variable, we see that the uniform-binning method gives better results for the ME and the uniform-interval method gives better results for the MSE. The difference can be translated back to the difference between the ME and MSE. Apparently, the uniform-binning method has a smaller overall error, but for some values, the error is really large. This is especially the case for the insulin variable. If we look at the histogram of insulin in figure 9.6, we see that in the range between 50 [mU/dl] and 80 [mU/dl] we have only a couple of data points. This means that the uniform-binning method only assigns a couple of bins to a significant part of the total range. The result is that the data-points in this range have a potentially large error. If this poses a problem is a modeling decision. In the case for the insulin variable, the modeler will probably consider the data points larger than 50 [mU/dl] as outliers and only consider the data points up to 50 [mU/dl].

Overall, we can conclude that it depends on the application whether uniform-binning or uniform-interval discretization is a better choice. How both methods work out when applied to a physiological process will be discussed in chapter 10, where we compare two versions of the glucose-insulin regulation model that only differ in discretization method.

9.4 Summary

This chapter discussed important issues for modeling with the extended DBN formalism, such as obtaining the structure, learning the parameters, and choosing a discretization. Overall, we are now ready to apply the presented theory, tools, methods and algorithms to actual physiological processes. We therefore conclude this part of the thesis and go on to part III, where we elaborate on the glucose-insulin regulation and cardiovascular system DBN models.

Part III

Application and empirical studies

*“In the computer field,
the moment of truth is a running program;
all else is prophecy.”*
- Herbert Simon.

Chapter 10

Glucose-insulin regulation in the human body

Recent studies show that hyperglycemia¹ and/or hypoglycemia² are common in critically ill patients. This malfunction is not limited to patients who are known to be diabetic, and thus tight glucose control is needed to decrease postoperative morbidity and mortality. Tight glucose control involves identifying patients at risk, monitoring blood glucose frequently and using an effective insulin infusion algorithm to control blood glucose within a narrow range. In this chapter, we will present a DBN model that can be used to infer insulin and glucose levels based on noisy glucose measurements. This is a first step toward an application of the extended DBN formalism in an expert system that can help to govern the glucose-insulin administration of a patient in an intensive care unit (ICU).

The goal of this application was to obtain an initial insight into the applicability and usability of the extended DBN formalism in modeling physiological processes. In the previous part of this thesis, we showed how we extended Murphy's DBN formalism (chapter 7), how we implemented the extension in GeNIE and SMILE (chapter 8), and how we can obtain the DBN structure and parameters of the DBN model (chapter 9). In this chapter, a study is performed to answer our research questions presented in section 1.1: *How do we obtain the structure and parameters? What is the performance of inference? How much patient data do we need for learning the parameters? If possible, how much does a DBN outperform a BN when applied to the same problem?*

The chapter first discusses the simulated patients and the resulting patient data. After that, the derivation of the DBN structure and the learning of the DBN parameters is discussed. Finally, the setup of the empirical study and its results are presented. Special-purpose tools

¹A condition in which an excessive amount of glucose circulates in the blood plasma.

²A condition in which too little glucose circulates in the blood plasma.

were programmed in C++ and/or Matlab for patient simulation, data discretization, parameter learning, and the empirical study. A general overview of the steps for modeling glucose-insulin regulation with a DBN is given in figure 10.1, including some questions and considerations regarding these steps. This figure is derived from the global research architecture presented in chapter 6.

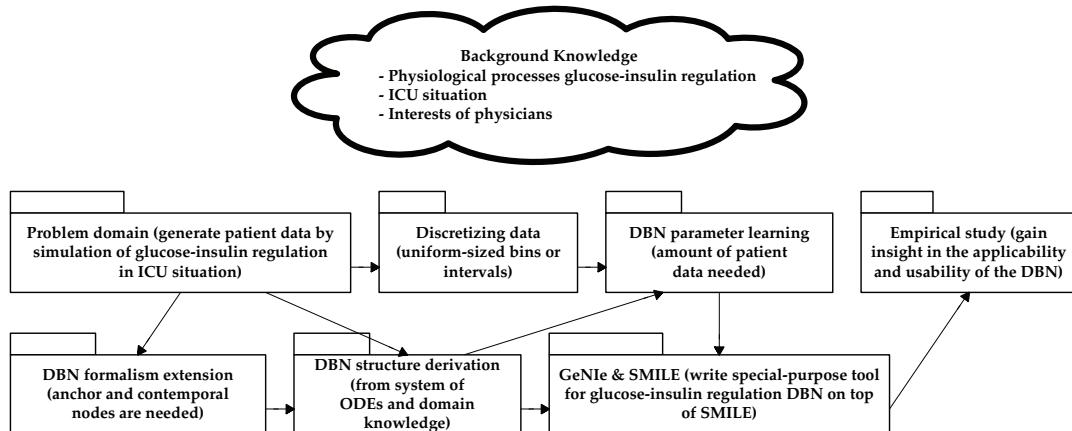


Figure 10.1: Overview of steps for modeling glucose-insulin regulation with a DBN.

10.1 Patient simulation

Chapter 2 presented a dynamic system that is able to model the glucose-insulin regulation of patients based on whether a patient is diabetic type-1³, diabetic type-2⁴, or normal. We are going to use this dynamic system to simulate patients. Remember that the presented dynamic system was based on average values for its parameters. We will give these parameters a Gaussian distribution, where we will take the average values as mean and the standard deviation will be set to $(0.13 \cdot \text{average value})$. The 0.13 is based on expert knowledge from Philips Research. The parameters of the dynamic system that are Gaussian distributed are: C_G , C_I , $Q_G(t)$, α , β , λ , μ , ν , ϕ , and θ .

We use the dynamic system to model the situation where an arbitrary patient in an ICU gets an insulin infusion on an empty stomach of 0, 5, 10 or 15 [U/h]. The glucose and insulin levels of every patient are monitored for 9 hours after the insulin infusion and every 15 minutes a measurement of the glucose level is taken. This measurement is simulated by taking the glucose level $G(t)$ and adding Gaussian noise with a standard deviation σ of 5.5 [mg/ml]. The Gaussian noise is added for two reasons. First, in a real-life situation, measuring the glucose level is noisy by itself, and second, it is not improbable that physicians measure the glucose levels several minutes too early or too late. Because the patient has an empty stomach before the insulin infusion, his glucose and insulin levels are considered to be in the dynamic system's steady-state (or *homeostasis* in this context). Figure 10.2 shows the course of the glucose and insulin levels of three patients, including the noisy observations of the glucose levels. (It is not possible to measure the insulin levels directly.)

Decision space One might wonder what the decision space is going to be of the resulting DBN model. To give some insight in the decision space, we have plotted the glucose-insulin

³A patient lacks the capacity to produce adequate amounts of insulin.

⁴A patient is unable to utilize circulating insulin effectively due to peripheral tissue resistance to its effects.

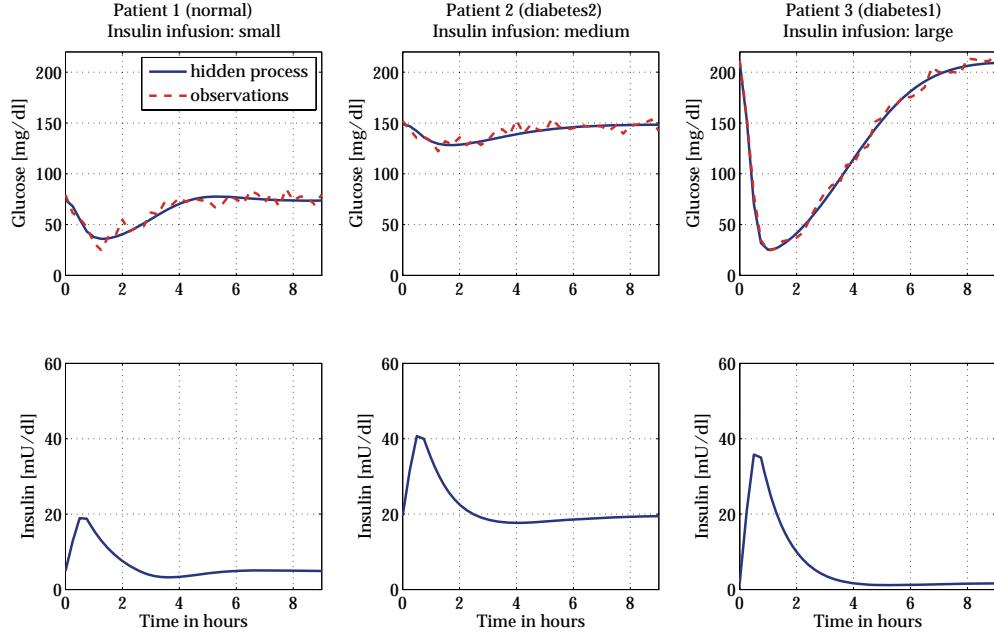


Figure 10.2: The glucose-insulin regulation of three randomly generated patients.

levels over a period of 9 hours of 1,000 patients in one graph per variable (figure 10.3). The patients are: normal (blue), diabetic type-1 (red), or diabetic type-2 (green). As one can see, the graphs that result from plotting 1,000 patients are rather complex. The task of the DBN is to smooth and/or predict the glucose and insulin levels (top and bottom graphs) based on the noisy glucose measurements (center graph).

10.2 DBN structure

The DBN has to be able to predict the course of the glucose and insulin levels before and after the infusion and/or identify diabetes based on noisy glucose measurements. Before we go to the actual DBN structure, let us recall the ODEs our dynamic system is based on. The glucose dynamics are governed by

$$C_G \frac{dG}{dt} = U_G + Q_G - \lambda G - \nu GI, \quad G \leq \theta, \quad (10.1)$$

$$C_G \frac{dG}{dt} = U_G + Q_G - \lambda G - \nu GI - \mu(G - \theta), \quad G > \theta, \quad (10.2)$$

and the insulin dynamics are governed by

$$C_I \frac{dI}{dt} = U_I - \alpha I, \quad G \leq \varphi, \quad (10.3)$$

$$C_I \frac{dI}{dt} = U_I - \alpha I + \beta(G - \varphi), \quad G > \varphi, \quad (10.4)$$

What can we learn from these equations? Well, we can use these equations to obtain a part of our DBN structure. Remember that we discussed in chapter 9 what considerations we have to keep in mind when obtaining the DBN structure. We will now apply these considerations to the glucose-insulin regulation model, starting with the fact that the system of ODEs and our DBN model have a time-granularity in the same order of magnitude, enabling us to give the interactions in the system of ODEs some more attention. From equations 10.1 and 10.2, it

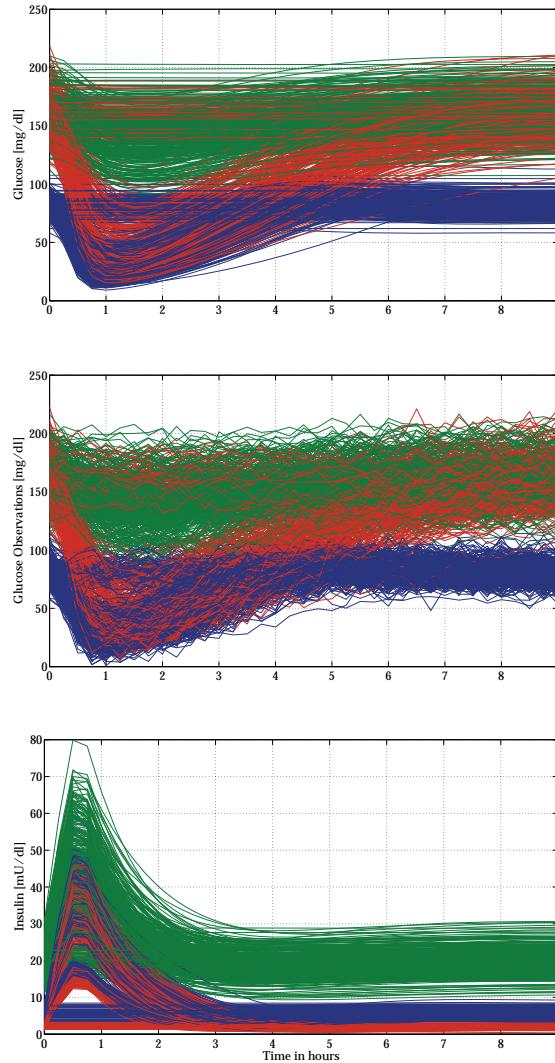


Figure 10.3: The glucose levels, noisy glucose measurements, and insulin levels of 1000 randomly generated patients. Normal patients are blue, diabetic type-1 patients are red, and diabetic type-2 patients are green.

follows that the glucose level at time t depends on the glucose and insulin levels at time $t - 1$. Furthermore, from equations 10.3 and 10.4, it follows that the insulin level at time t depends on the glucose and insulin levels at time $t - 1$ and the size of the insulin infusion. Another important fact is that the parameters of different patient types (diabetic type-1, diabetic type-2 and normal) are Gaussian distributed with different means and variances for every patient type. So, we can add a contemporaneous variable that serves as an index variable to the glucose and insulin level variables that switches between the probability distributions of a diabetic type-1, diabetic type-2, or normal patient. Finally, we learn that at the beginning of each modeled patient process, the patient has an empty stomach, meaning that the dynamic system is in a steady-state (homeostasis). In this case, the dependency of the insulin and glucose levels interact with each other within the same time-step. Hence, the initial distribution (or the anchor nodes) also contains an arc between the glucose and insulin variables. The resulting DBN definition is shown in figure 10.4. Included are the two networks that result from the decomposition of the DBN definition according to the method described in section 9.2.2.

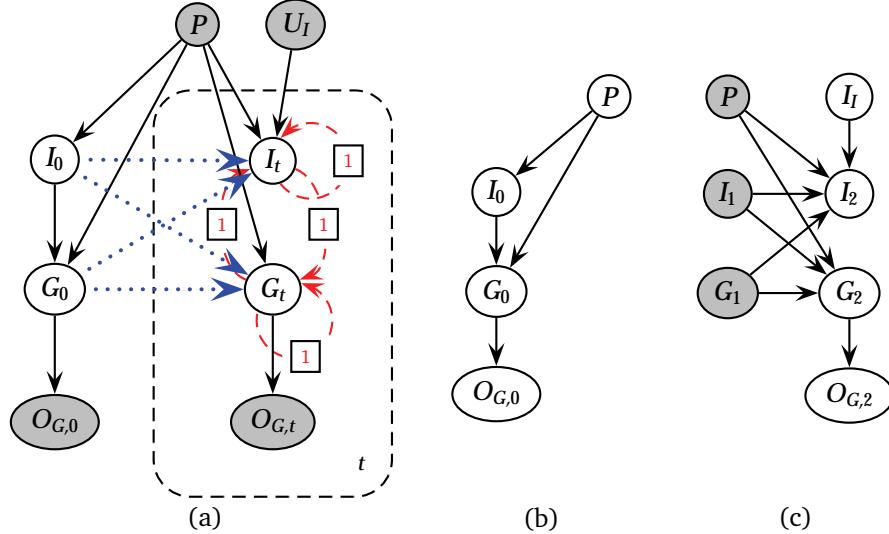


Figure 10.4: (a) DBN definition for glucose-insulin administration, shaded variables can be observed. (b) The initial network for learning the parameters. (c) The transition network for learning the parameters, variables that are only used for indexing are shaded.

10.3 DBN parameters

Figures 10.4b and 10.4c show the networks that result from the decomposition of the glucose-insulin regulation DBN. A part of the patient data that was generated during the simulation is shown in tables 10.1a and 10.1b. Before this data can be used to learn the DBN parameters, it needs to be processed. This preprocessing consists of two steps: Sampling (decimation) and discretization. In this particular case, sampling is fixed to 15 minutes between each sample. The variables are discretized into 30 bins, which provides a sufficient precision for our purposes. Section 10.4 discusses the experimental results of two discretization approaches of the patient data.

Table 10.1: Part of the simulated patient data that can be used to learn the glucose-insulin regulation DBN.

P	I_0	G_0	$O_{G,0}$
diabetes1	2.0	166.2	172.5
diabetes2	21.8	154.9	164.7
diabetes2	21.4	129.5	132.7
diabetes1	2.0	179.7	176.3
normal	5.8	85.5	85.1
:	:	:	:

(a)

P	I_I	I_1	I_2	G_1	G_2	$O_{G,2}$
diabetes1	large	2.0	23.7	166.2	136.3	143.0
diabetes1	large	23.7	40.0	136.3	82.2	86.5
diabetes1	large	40.0	38.5	82.2	45.3	50.2
diabetes1	large	38.5	29.4	45.3	32.8	32.3
diabetes1	large	29.4	22.4	32.8	29.4	32.2
:	:	:	:	:	:	:

(b)

10.4 Empirical study

Tight glucose control for critically ill patients is needed to decrease postoperative morbidity and mortality. Currently, the glucose control algorithms used by physicians do not take into account past glucose measurements to predict the future. Often, this results in the physician overriding a glucose control algorithm, because it is not effective. In this case, a DBN can help to obtain a better glucose control algorithm, because it can make predictions of glucose and insulin levels

based on glucose observations from both the past and the present. This section will show how well the DBN formalism is suited for this application.

10.4.1 Experimental design

For the glucose-insulin regulation DBN, we are interested in three research questions: *How many observations do we typically need for reliable inference results and when do we need to do the observations? How many patient cases do we need for parameter learning? What is the effect of uniform-sized intervals versus uniform-sized binning in the discretization step on the performance of the DBN?*

We will try to answer these questions in three separate experiments. We have many degrees of freedom in the design of these experiments, so we need to make some choices. The DBN parameters are learned with patient simulations of 9 hours. The patient data is sampled every 15 minutes, thus the time between two consecutive time-slices is 15 minutes as well. The derivation of the sampling-rate can be an experiment on its own, but in this case, expert knowledge from physicians stated that 15 minutes is optimal. This seems like a long time between two consecutive time-slices, but in real-life, glucose levels are only measured a couple of times per day at most.

In real ICU situations, only the first couple of hours after the infusion are interesting for a physician and thus we will concentrate on the first 4 hours after the infusion during the experiments. This means that the DBN is going to be unrolled for a sequence length of $T = 16$ time-slices. Including the anchor variables, this results in a DBN of 17 time-slices. The chosen time-period has the largest variation of the glucose and insulin levels, as can be seen in figure 10.3. All experiments assume that the present time t is at 2 hours after the infusion and that the size of the insulin infusion and the patient type are known. Glucose observations are collected for 0-2 hours. This means that from 0-2 hours the glucose and insulin levels are smoothed, at 2 hours after the infusion the levels are filtered and from 2-4 hours the levels are predicted, as clarified in figure 10.5. The baseline DBN used for the experiments is trained by

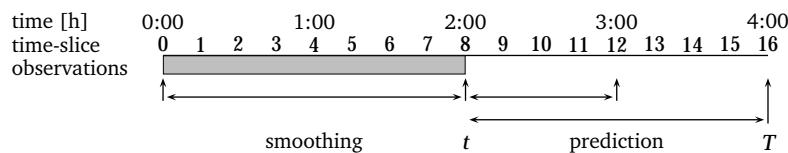


Figure 10.5: Inference in the glucose-insulin regulation DBN. The shaded region is the interval that contains observations. t is the current time, and T is the sequence length.

taking $90\% \text{ of } 10,000 = 9,000$ simulated patients. The remaining 1,000 patients are used as test data. The glucose levels, insulin levels, and glucose measurements are discretized using the uniform-interval method. These variables contain 30 distinctive states each.

10.4.2 Special-purpose software

Special-purpose software was written to process the data during the different steps of the experiments. Every experiment consisted of four main steps: (1) Simulating patients in Matlab, (2) discretizing and sampling the patient data, (3) learning the DBN parameters and performing inference with C++/SMILE, and (4) calculating the performance measures in Matlab. Figure 10.6 shows the experimental design of every experiment. In the first step, a number of patients is simulated using the Matlab implementation of the glucose-insulin regulation model presented in chapter 2 and section 10.1. The generated patient data is saved to a .csv⁵ file, after which

⁵Flat file format with each value separated by a comma.

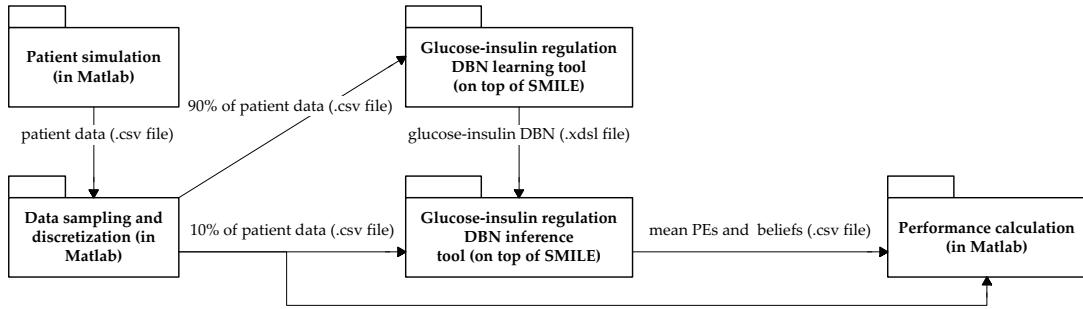


Figure 10.6: The experimental design of every experiment.

it is discretized and sampled in the second step. 90% of the resulting patient data is used for learning the DBN parameters and the remaining 10% is used as test data. The third step loads the .csv file with patient test data and the learned DBN from a .xds1 file into the glucose-insulin regulation DBN tool. This tool performs inference and writes the posterior beliefs and the mean point estimates to another .csv file. The fourth and final step loads this file and the patient test data file into a Matlab program that calculates the inference performance measures. The quality of the posterior beliefs and the mean point estimates can be compared using the following measures, two of which we have already seen in chapter 9, but will be repeated here:

Mean squared error The mean squared error or MSE of an estimated vector \tilde{x} and the true vector x can be calculated as follows:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (x_i - \tilde{x}_i)^2 , 0 \leq \text{MSE} \leq \infty . \quad (10.5)$$

The mean squared error has the property that larger errors give a proportionally larger MSE than smaller errors. This is a property that we are not interested in, and thus we will use the more intuitive mean error or ME for the experiments:

$$\text{ME} = \frac{1}{n} \sum_{i=1}^n |x_i - \tilde{x}_i| , 0 \leq \text{ME} \leq \infty . \quad (10.6)$$

Because we are not only dealing with an error because of inference, but also with an error because of the discretization step, the ME will be taken of the inferred state minus the original state, and of the value that the inferred state represents minus the original value to see how the effect of the discretization translates to the performance of the DBN. Note that the ME will be taken over a point estimate (the mean) of an inferred state vector. With this approach, it is not possible to measure the level of certainty of an inferred state vector. For instance, the state vector {0.01, 0.01, 0.96, 0.01, 0.01} will give the same mean point estimate as the state vector {0.20, 0.20, 0.20, 0.20, 0.20}, but we can clearly see that the level of certainty of the former state vector is much higher than the latter. Therefore two other common measures can be used to get insight into the level of certainty: the Euclidean distance and the Hellinger distance.

Euclidean distance The Euclidean distance or ED of an estimated vector \tilde{x} and the true vector x can be calculated as follows:

$$\text{ED} = \sqrt{\sum_{i=1}^n (x_i - \tilde{x}_i)^2} , 0 \leq \text{ED} \leq \sqrt{2} . \quad (10.7)$$

An important difference between the ED and the ME is that the ED will be taken between state vectors of the original and inferred states, where the ME will be taken over the original state and

the mean point estimate of the inferred state! Because we are interested in the mean Euclidean distance over all the state vectors, we will calculate:

$$\text{MED} = \frac{1}{k} \sum_{j=1}^k \sqrt{\sum_{i=1}^n (x_i - \tilde{x}_i)^2}, \quad 0 \leq \text{MED} \leq \sqrt{2}, \quad (10.8)$$

where k is the number of state vectors. With this approach, it is possible to measure the level of certainty of a state.

The Euclidean distance has one major disadvantage and that is that it treats the inferred state vectors $\{0.01, 0.01, 0.96, 0.01, 0.01\}$ and $\{0.00, 0.02, 0.96, 0.02, 0.00\}$ different (compared to an original state vector $\{0, 0, 1, 0, 0\}$). It is debatable whether the former or the latter distribution gives more certainty, but the former gives definitely not more certainty than the latter. However, the Euclidean distance implies this. For this reason, we will use the Hellinger distance [KN01] in the experiments, which is introduced next.

Hellinger distance The Hellinger distance or HD of an estimated vector \tilde{x} and the true vector x is defined as follows:

$$\text{HD} = \sqrt{\sum_{i=1}^n \left(\sqrt{\frac{x_i}{\sum_{i=1}^n x_i}} - \sqrt{\frac{\tilde{x}_i}{\sum_{i=1}^n \tilde{x}_i}} \right)^2}, \quad 0 \leq \text{HD} \leq \sqrt{2}. \quad (10.9)$$

Because the values of the states within a state vector always sum up to 1, the equation becomes:

$$\text{HD} = \sqrt{\sum_{i=1}^n (\sqrt{x_i} - \sqrt{\tilde{x}_i})^2}, \quad 0 \leq \text{HD} \leq \sqrt{2}. \quad (10.10)$$

Again, we are interested in the mean Hellinger distance over all the state vectors, so we will calculate:

$$\text{MHD} = \frac{1}{k} \sum_{j=1}^k \sqrt{\sum_{i=1}^n (\sqrt{x_i} - \sqrt{\tilde{x}_i})^2}, \quad 0 \leq \text{MHD} \leq \sqrt{2}, \quad (10.11)$$

where k is the number of state vectors. With this approach, it is possible to measure the level of certainty of a state in a better way than the Euclidean distance.

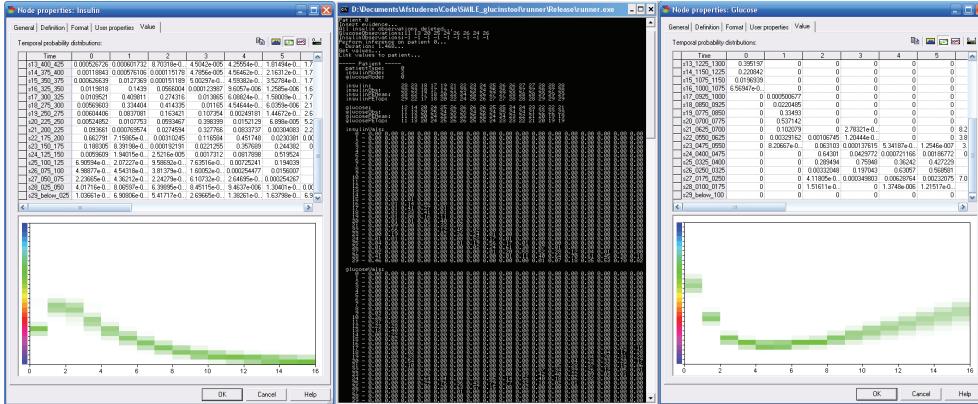


Figure 10.7: The output of the special-purpose glucose-insulin regulation DBN tool in comparison to the output of the same network in GeNIe. The center screenshot shows the special-purpose tool, the left and right screenshots are from GeNIe.

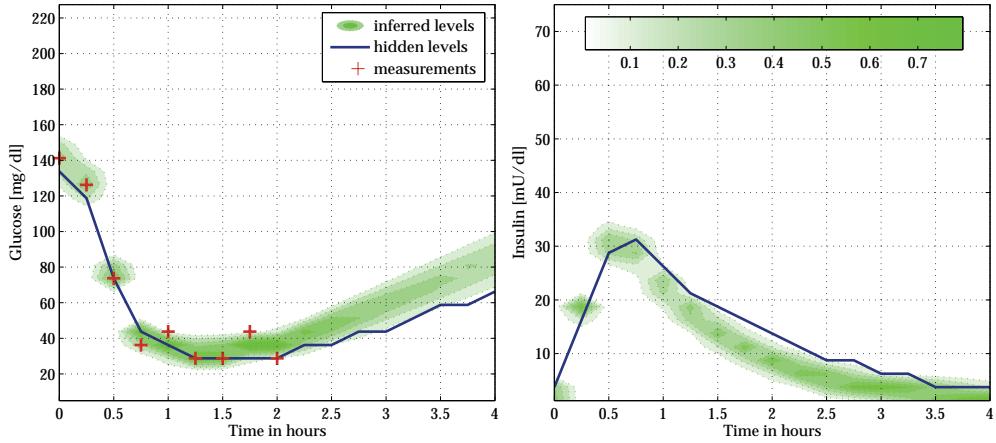


Figure 10.8: The output of the glucose-insulin regulation DBN in Matlab. Shown are the discretized measurements and actual glucose and insulin levels of the patient, and the inferred beliefs.

As a side note, the special-purpose tool that performs inference on the glucose-insulin regulation DBN can also be used as a standalone tool for a decision maker. However, the GUI that GeNIE provides is much more user friendly. Figure 10.7 shows the tool in comparison to the GeNIE GUI. Figure 10.8 shows the same patient using Matlab's visualization tools. The advantage of using Matlab for visualization is the possibility to include the actual glucose and insulin levels.

10.4.3 Experiment 1: varying the glucose measurement rate

The first experiment varies the observation rate of the glucose levels in the first 2 hours (or 9 time-slices) of the total sequence. For 1,000 simulated patients, glucose measurements are done every 15, 30, or 60 minutes until 2 hours after the infusion. Inference is performed to obtain the smoothed, filtered and predicted beliefs of the glucose and insulin levels.

Results We are interested in the performance of the baseline DBN⁶ for time-slices 0-16. Because evidence is inserted until 2 hours after the infusion, the DBN smooths the glucose and insulin levels for time-slices 0-7, filters the levels for time-slice 8, and predicts the levels for time-slices 13-16. Table 10.2 presents the mean errors and Hellinger distances for time-slices 0-16. How the errors and distances develop over time is presented in figure 10.9.

Discussion In the following discussion, we will only consider the performance of inference of the *states* of the glucose and insulin levels (first row of figure 10.9). The actual *values* (second row of figure 10.9) are included in the graph to get an insight on how the inference error translates to actual values of the glucose and insulin levels. The original discretization error after discretization of the test data is included to get an idea of the smallest error that can be achieved. It is theoretically not possible for the DBN to get its inferred values below the discretization error.

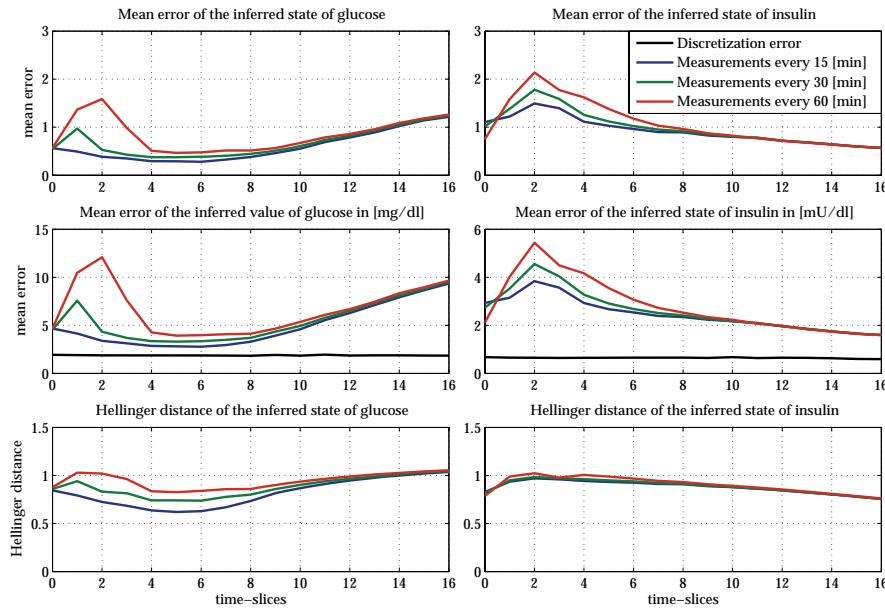
A number of things can be concluded from figure 10.9. Starting with the glucose graphs:

- As expected, the error and the uncertainty of glucose grows as the prediction time gets larger, no matter how many observations are done. This can be seen in the graphs for the ME and MHD of glucose.

⁶The DBN parameters are learned using the simulated data of 9,000 patients. The variables are discretized using uniform-sized intervals and have 30 distinctive states each. The time between two consecutive time-slices is 15 minutes.

Table 10.2: Overview of results for experiment 1: varying the observation rate.

data-rate	Glucose				Insulin			
	ME state	ME value	ME disc.	MHD	ME state	ME value	ME disc.	MHD
15 minutes	0.60	4.91	1.88	0.82	0.93	2.45	0.65	0.88
30 minutes	0.68	5.45	1.88	0.88	0.98	2.59	0.65	0.89
60 minutes	0.84	6.61	1.88	0.94	1.06	2.80	0.65	0.90

**Figure 10.9:** Mean errors and Hellinger distances of glucose and insulin levels over time for experiment 1.

- Smoothing really works! The error and the uncertainty of glucose get smaller when more glucose measurements are gathered over time. The ME even gets smaller than the ME introduced by doing the noisy glucose measurements (not shown in the graph), which is surprising because the DBN also has to deal with the error due to the discretization.
- Not surprisingly, the error and the uncertainty get larger as less glucose measurements are done and thus a glucose measurement every 15 minutes is best. However, further investigation gives the insight that the course of the glucose and insulin levels are especially difficult to predict just after the infusion, as can be seen from the graphs where a peak occurs at time-slices 1 and 2 (15-30 minutes after the infusion) and the error is not significantly different for the measurement rates from time-slice 4 and onwards. This observation suggests that an optimal scheme is taking measurements every 15 minutes for the first 30 minutes and every 60 minutes afterward.

The insulin graphs show very different results. A global explanation can be that the insulin levels have to be inferred with the glucose levels as an intermediate variable between the insulin levels and the glucose measurements. The DBN seems to have some problems with this. A solution would be to find method to measure the insulin level directly. Qualitative experimentation shows that not many insulin measurements are needed to improve the results significantly. However, it is not easy to accomplish insulin measurements on real patients. An evaluation of the insulin graphs:

- Again, the error of the insulin infusion is largest just after the infusion. The DBN finds it difficult to predict how the patient will utilize the infusion based on the glucose measurements.

- An unexpected result is that the error and the uncertainty get smaller over time, no matter what the glucose measurement rate is. This can be explained partly by observing from figure 10.3 that the insulin level goes back to its steady-state relatively fast. The insulin levels in the DBN also goes to this state as time moves forward and the patient gets no new insulin infusion, no matter what the glucose measurement reads. Somehow, the DBN incorporates this background knowledge during the learning phase.

Overall we can conclude from this experiment that a variable glucose measurement rate scheme needs to be devised for optimal smoothing and prediction results. An optimal scheme would be taking measurements every 15 minutes for the first 30 minutes and every 60 minutes afterward. Furthermore, although not shown in figure 10.9, it appears that including some extra insulin measurements improves the performance of the DBN significantly.

10.4.4 Experiment 2: varying the number of patient cases used for learning

The second experiment varies the number of patient cases used for learning. Because we can generate an unlimited amount of data, we decided to vary the number of patients from 100 to 10,000 patients. In every learning iteration, 90% of the data is used for training and the remaining 10% is used for testing. We did not consider techniques such as cross-validation that can be used for learning with a small dataset.

Results In the experiment, the measurement rate of the glucose levels is every 15 minutes in the first 2 hours (or 9 time-slices) of the total sequence. Inference is performed to obtain the smoothed, filtered and predicted beliefs of the glucose and insulin levels for DBNs that are trained with a different number of patients. Table 10.3 shows a global overview of the results. Figure 10.10 shows an overview of the results over time. We decided to only plot the cases where a DBN is learned for 100, 1,000, or 10,000 patients.

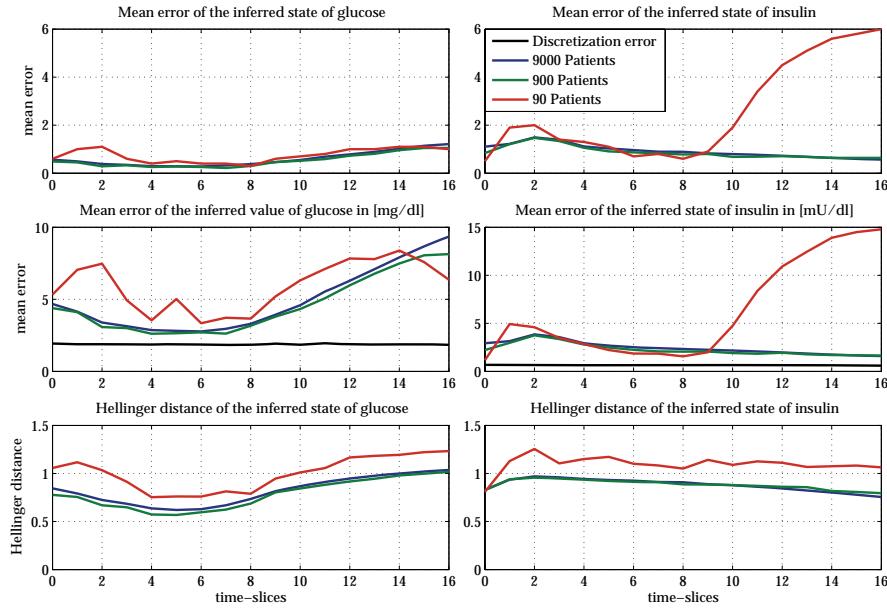
Discussion The experimental results presented in table 10.3 and figure 10.10 give the following insights:

- For a data set ranging from 1,000 to 10,000 patients the inference results do not differ significantly. The results for 1,000 patients are even a little better than the results for 10,000 patients, but this is due to the properties of the test data. The results for experiments with DBNs that are learned with a number of patients that lies in the interval [1,000 – 10,000] are not shown in the graph, but give similar results.
- A patient data set of 100 patients is definitely not enough to learn the DBN. Strangely enough, smoothing still performs quite well, but the error explodes with the prediction part. This is especially the case for the insulin variable. Apparently a patient data set of 100 patients is not enough to let the information of the glucose measurements *get through* to the insulin variable with the glucose variable as intermediate variable. Furthermore, the insulin variable has many more parameters to be learned than the glucose variable, because it has one extra parent.
- Again, the DBN finds it difficult to infer the result of the insulin infusion just after the infusion, but we have seen this already in experiment 1.

Overall, the optimal number of patients for learning the DBN lies somewhere between the interval of [800 – 1,200] patients, depending on the quality of the patient data. This is less than expected according to the number of CPT parameters that need to be learned. For insulin the number of CPT parameters is $30 \cdot 3^1 \cdot 4^1 \cdot 30^2 = 324,000$ and for glucose the number of CPT parameters is $30 \cdot 3^1 \cdot 30^2 = 81,000$ (equation 3.7). This is a huge number of parameters that need to be learned, but after investigation of the insulin and glucose CPTs, it appears

Table 10.3: Overview of results for experiment 2: varying the number of patient cases used for learning.

number of patients	Glucose				Insulin			
	ME state	ME value	ME disc.	MHD	ME state	ME value	ME disc.	MHD
9000 patients	0.60	4.91	1.88	0.82	0.93	2.45	0.65	0.88
900 patients	0.53	4.59	1.88	0.78	0.87	2.27	0.65	0.88
90 patients	0.74	5.92	1.88	1.00	2.56	6.25	0.65	1.09

**Figure 10.10:** Mean errors and Hellinger distances of glucose and insulin levels over time for experiment 2.

that many parameters are not set after learning the DBN, because they still have their initial distribution (uniform). This suggests that a different canonical model can be appropriate for both the glucose and insulin variables. However, applying the Noisy-MAX canonical model did not give satisfying results, so a different canonical model needs to be devised. This is currently a very active research topic [ZD06, ZVD06], but we will not go into details here, because it is beyond the scope of this thesis.

10.4.5 Experiment 3: changing the discretization method

We presented several discretization methods in section 9.3. We already stated that the SLC-based algorithm does not apply to us, but it is interesting to compare the performance of the other two discretization methods. In the following experiment, we are going to compare the baseline DBN that is learned with 9,000 patients and has its glucose and insulin variables discretized using 30 uniform-intervals to a DBN that is also learned with 9,000 patients, but has its variables discretized using 30 uniform-bins.

Results In the experiment, the measurement rate of the glucose levels is every 15 minutes in the first 2 hours (or 9 time-slices) of the total sequence. Inference is performed to obtain the smoothed, filtered and predicted beliefs of the glucose and insulin levels for DBNs that are trained with a different number of patients. Table 10.4 shows a global overview of the results. Figure 10.11 shows an overview of the results over time. In contrast to the two earlier experiments, the difference between the ME of the inferred *states* (first row of figure 10.11) and

values (second row of figure 10.11) is important, because we are varying the mapping between them.

Table 10.4: Overview of results for experiment 3: changing the discretization method.

discretization method	Glucose				Insulin			
	ME state	ME value	ME disc.	MHD	ME state	ME value	ME disc.	MHD
uniform-intervals	0.60	4.91	1.88	0.82	0.93	2.45	0.65	0.88
uniform-bins	0.97	5.44	2.18	0.87	1.75	2.39	0.80	1.05

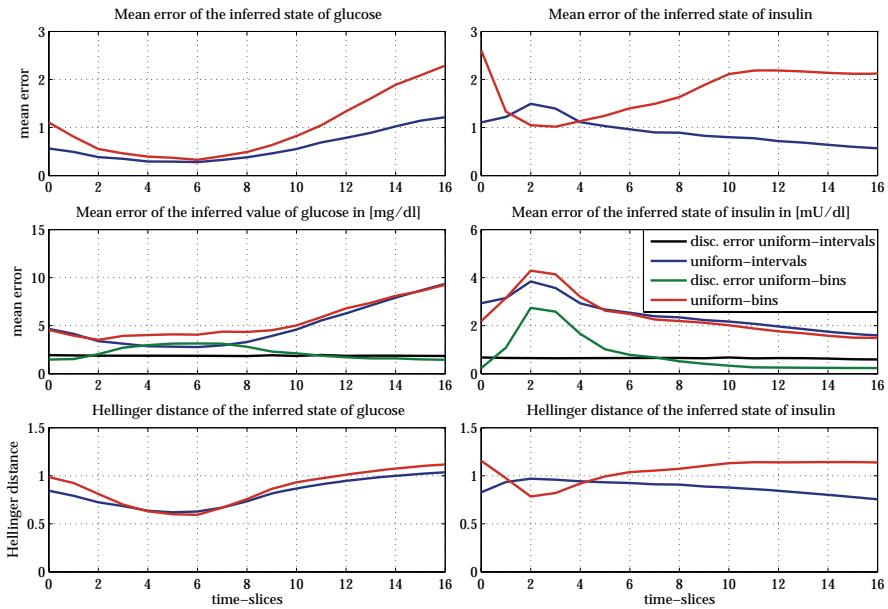


Figure 10.11: Mean errors and Hellinger distances of glucose and insulin levels over time for experiment 3.

Discussion The experimental results presented in table 10.4 and figure 10.11 give the following insights:

- Uniform-interval discretization outperforms uniform-binning discretization, which is a bit surprising because uniform-bin discretization takes the properties of the data into account, and uniform-interval discretization does not. When looking into the results of the uniform-bin discretization algorithm, it appeared that the glucose range from ≈ 182.5 [mg/dl] to ≈ 227.5 [mg/dl] is represented with only 1 interval, where this is 6 intervals when using the uniform-interval discretization algorithm. The insulin discretization is even worse, here the range from ≈ 35 [mU/dl] to ≈ 75 [mU/dl] is represented with only 1 interval, where this is 16 intervals with uniform-interval discretization. This explains why the mean (discretization) error of the glucose and insulin values between respectively time-slices 2-8 and 1-5 are so much larger with uniform-bin discretization than uniform-interval discretization.
- It partly follows from the previous insight, but for lower values of glucose and insulin, uniform-bin discretization gives a lower discretization error than uniform-interval discretization. However, this does not affect the results of inference of the insulin and glucose levels. This can be clearly seen from the graph where the ME of the insulin *value* is shown over time (right-most graph in second row of figure 10.11). In this graph, the discretization error gets less over time for uniform-binning discretization, but the errors of the values stay similar.

The results of this experiment show that both methods have their advantages and disadvantages, although in this particular case, uniform-interval discretization outperforms uniform-binning discretization. Therefore, we will just stick with uniform-interval discretization in this research. From observing the inference performance results and the results of the unsupervised uniform-bin discretization, it seems that an optimal discretization can be found by making the discretization part of the learning algorithm instead of two separate steps. In such an algorithm, the dependencies among variables for different discretization intervals can be exploited. This might result in better inference performance and a smaller overall discretization error. However, devising and implementing such an algorithm is beyond the scope of this thesis. Another option is to use knowledge about the domain to discretize the variables. This is what we will do in chapter 11, where we will devise a DBN model for the human cardiovascular system.

10.4.6 Conclusions

Although several of the insights gained from the experiments apply solely to modeling the glucose-insulin regulation, much experience was gained for modeling a physiological process with a DBN. The three most important insights can be generalized and applied to other DBN models. These insights are:

1. When learning the DBN parameters, less data is needed for an acceptable performance than one might expect from the number of parameters that need to be learned. How much data is needed also depends on the purpose of the DBN, i.e. for smoothing and filtering significantly less data is needed than for prediction.
2. No matter the amount of data used to learn the DBN, many CPT parameters of the discretized variables remain untouched after learning. This suggests that a different parametric representation is suitable for these variables. One example might be applying the Noisy-MAX gate, but this did not give satisfying results. Research needs to be done to find better canonical representations for these kind of variables.
3. Discretization can have a major influence on the performance of the DBN. As such, it is not wise to use an unsupervised discretization method without considering the domain, the process and the DBN structure. Ideally, an algorithm could be devised that incorporates the discretization into the learning algorithm, but we choose to rely on expert knowledge about the domain.

10.5 Summary

In this chapter we devised a DBN model for the human glucose-insulin regulation and performed an empirical study to get insight into its performance. During the empirical study, it turned out that the glucose-insulin DBN performs quite well. For most experiments, the error stays within reasonable bounds, and the computation time for inference was only in the order of seconds. Based on the results from the experiments, we gained many insights into the practical issues involved when modeling a physiological process with a DBN. These insights will be used in the next chapter, where we will devise a DBN model for the much more complex cardiovascular system.

Chapter 11

Cardiogenic shock in the human cardiovascular system

Cardiovascular diseases pose a great problem in the industrialized world, where it has been the leading cause of death for over a century. According to the World Health Organization (WHO), the gap between the cardiovascular cause of death and others will widen as more and more people in the industrialized world are getting overweight because of bad eating and drinking habits and too little exercise.

Unfortunately, the development for treatment of cardiovascular diseases tends to stay behind. One of the main causes might be the lack of an effective and flexible model of the cardiovascular system. Philips Research is currently working on a model based on a system of ordinary differential equations (ODEs) extended with non-linearities that is able to simulate patients with several cardiac disorders. An overview of a part of this system and the cardiovascular system in general was presented in chapter 2. The problem of modeling a physiological process with a system of ODEs is that the exact interactions between variables need to be known, something that is not the case for the complete cardiovascular system. The DBN formalism provides means of countering this problem by enabling the modeler to use machine-learning techniques to obtain the interactions from data. Furthermore, the DBN formalism provides natural support for uncertainty and for non-numeric variables, such as nominal, ordinal, or interval variables.

The cardiovascular system DBN that we present in this chapter is a result of combining our DBN knowledge gained by the extension of Murhpy's DBN formalism, development of the temporal reasoning software, and insights from modeling the glucose-insulin regulation with knowledge from medical experts on the cardiovascular system. The resulting DBN model is able to infer the values of important physiological variables on the basis of measuring non-evasive variables such as the heart-rate and the systolic and diastolic arterial blood pressure for every 3 seconds. Furthermore, it is able to detect whether the patient is having a cardiogenic shock. In the following sections, we will discuss the variables of the cardiovascular system and how they relate to each other, the properties of a cardiogenic shock, and the derivation of the DBN

structure and its parameters. After this, we will present the results of an empirical study on the performance of the DBN model. Just like modeling the glucose-insulin regulation in chapter 10, special-purpose tools were programmed in C++ and/or Matlab for data manipulation and discretization, parameter learning, and the empirical study. A general overview of the steps for modeling the cardiovascular system with a DBN is given in figure 11.1. This figure is derived from the global research architecture presented in chapter 6.

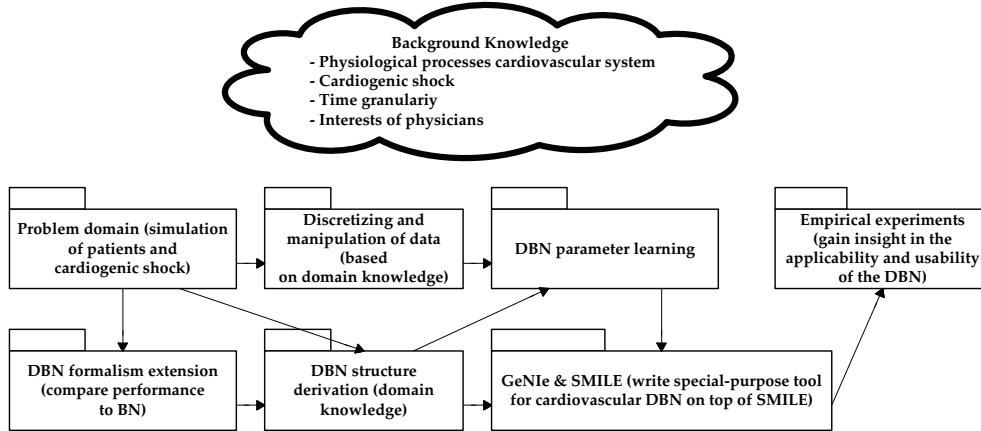


Figure 11.1: Overview of steps for modeling the cardiovascular system with a DBN.

11.1 Patient simulation

A global introduction to the cardiovascular system was presented in chapter 2. Philips Research is developing a model based on a system of ODEs extended with non-linearities that is able to simulate the cardiovascular system of patients with varying properties and cardiac disorders. The cardiovascular system consists of many interacting variables, but only a subset of those is important for a physician and will be incorporated into the DBN model. Note that we will only consider variables that are related to the left-ventricle of the heart, because this is the ventricle that pumps blood through your body. (The right ventricle supplies blood to the lungs.) The nine variables of interest are the following:

- Cardiac output (*CA*) in [ml],
- Ejection fraction (*EF*) in [%],
- Maximum elastance of the left ventricle (*E_{mlv}*) in [mmHg/ml],
- Heart-rate (*HR*) in [bpm],
- Pressure of the left ventricle (*P_{lv}*) in [mmHg],
- Pressure of the systemic arteries (*P_{sa}*) in [mmHg],
- Resistance of the extrasplanchnic peripheral circulation (*R_{ep}*) in [mmHgs/ml],
- Resistance of the splanchnic peripheral circulation (*R_{sp}*) in [mmHgs/ml], and
- Volume of the left ventricle (*V_{lv}*) in [ml].

Most of these variables speak for themselves, except maybe for the cardiac output, which is the volume of blood pumped out of the left ventricle; the ejection fraction, which is the percentage of blood pumped out at each heart beat; and the maximum elastance of the left ventricle, which determines the strength of the cardiac muscle.

For the DBN model, we added a binary cardiogenic shock variable (*Shock*) that gives meaning to the levels of the variables by assigning each moment in time with a belief of the patient having a cardiogenic shock or not. A cardiogenic shock is a result of a heart failure and causes

an insufficient amount of blood reaching the brain. In our dataset, the heart failure is simulated by lowering the maximum elastance of the left ventricle.

The dataset provided by Philips Research consisted of 2933 patient simulations of 100 [s] per patient, sampled every 50 [ms]. During this period, the maximum elastance of the left ventricle was lowered by a variable amount, consequently simulating the possibility of a cardiogenic shock. Furthermore, high-frequency fluctuations of several variables were incorporated into the simulation to get a better approximation of the real cardiovascular system. An example of the course of the variables of an arbitrary simulated patient is given in the figure 11.2, where the blue line represents the data set as provided by Philips Research. The purpose of the red and green lines will be explained in section 11.3. The shown patient clearly suffers from a cardiogenic shock caused by lowering the maximum elastance of the left ventricle at time $t = 30[\text{s}]$. Some patients are able to compensate the drop of the maximum elastance (and thus the drop of the systemic pressure and the cardiac output) by a combination of a higher heart-rate and vasoconstriction (narrowing of the vessels), enabling them to avoid getting into a cardiogenic shock.

11.2 DBN structure

The structure of the DBN for the cardiovascular system was devised by the combined efforts of our knowledge about modeling with DBNs and expert knowledge from physicians. The DBN model had to represent the cardiovascular system for a time-granularity of 3 [s] and be able to infer values for all variables solely based on the easily measurable heart-rate and pressure of the systemic arteries. We chose a time-step of 3 [s] because this is a period that is short enough to capture interesting dynamics of the cardiovascular system, but long enough to capture mean values of the variables that are meaningful. Because we are modeling time-steps of 3 [s], several physiological variables are split into their systolic (upper) and diastolic (lower) counterparts. The presented model is certainly not perfect, but it provides us with enough precision without being too computationally expensive. What follows are the assumptions that the resulting DBN model is based upon.

The strength of the cardiac muscle (E_{mlv}) influences the volume of the left ventricle (V_{lv}). When the cardiac muscle contracts, the volume gets smaller and when it releases the volume gets larger again. The amount of decrease and increase in volume is determined by E_{mlv} . V_{lv} is separated into a systolic ($V_{lv\uparrow}$) and diastolic ($V_{lv\downarrow}$) variable, thus we have to draw two arcs: $E_{mlv} \rightarrow V_{lv\uparrow}$ and $E_{mlv} \rightarrow V_{lv\downarrow}$. These variables also depend on each other and from the insight that the systolic pressure follows from the diastolic pressure, the following arc is added: $V_{lv\downarrow} \rightarrow V_{lv\uparrow}$.

The cardiac muscle also influences the pressure in the left ventricle (P_{lv}), by contracting the pressure increases, thus the higher its strength, the higher the potential pressure. Because the diastolic pressure in the left ventricle is more or less constant for every patient, we decided to only consider the systolic pressure of the left ventricle ($P_{lv\uparrow}$) and we introduce the following arc: $E_{mlv} \rightarrow P_{lv\uparrow}$.

P_{lv} also influences V_{lv} , because more blood will flow into the aorta when the pressure is higher. Thus, we introduce $P_{lv\uparrow} \rightarrow V_{lv\uparrow}$ and $P_{lv\uparrow} \rightarrow V_{lv\downarrow}$.

The interaction between P_{lv} and V_{lv} also depends on the arterial blood pressure P_{sa} . When P_{sa} is high, the heart needs to build up a very high pressure to pump enough blood into the aorta, so V_{lv} will decrease less at higher values for P_{sa} given a certain value for P_{lv} ¹. Again, P_{sa} is divided into a systolic $P_{sa\uparrow}$ and diastolic $P_{sa\downarrow}$ variable. Based on the previous reasoning, we add the following arc: $P_{sa\uparrow} \rightarrow P_{lv\uparrow}$.

P_{sa} depends on three variables: on the amount of blood circulating with a certain speed or force because of E_{mlv} , and on the splanchnic and extra-splanchnic resistance of the vessels R_{sp}

¹This also means that the cardiac muscle needs more strength for people with a blood pressure that is (too) high.

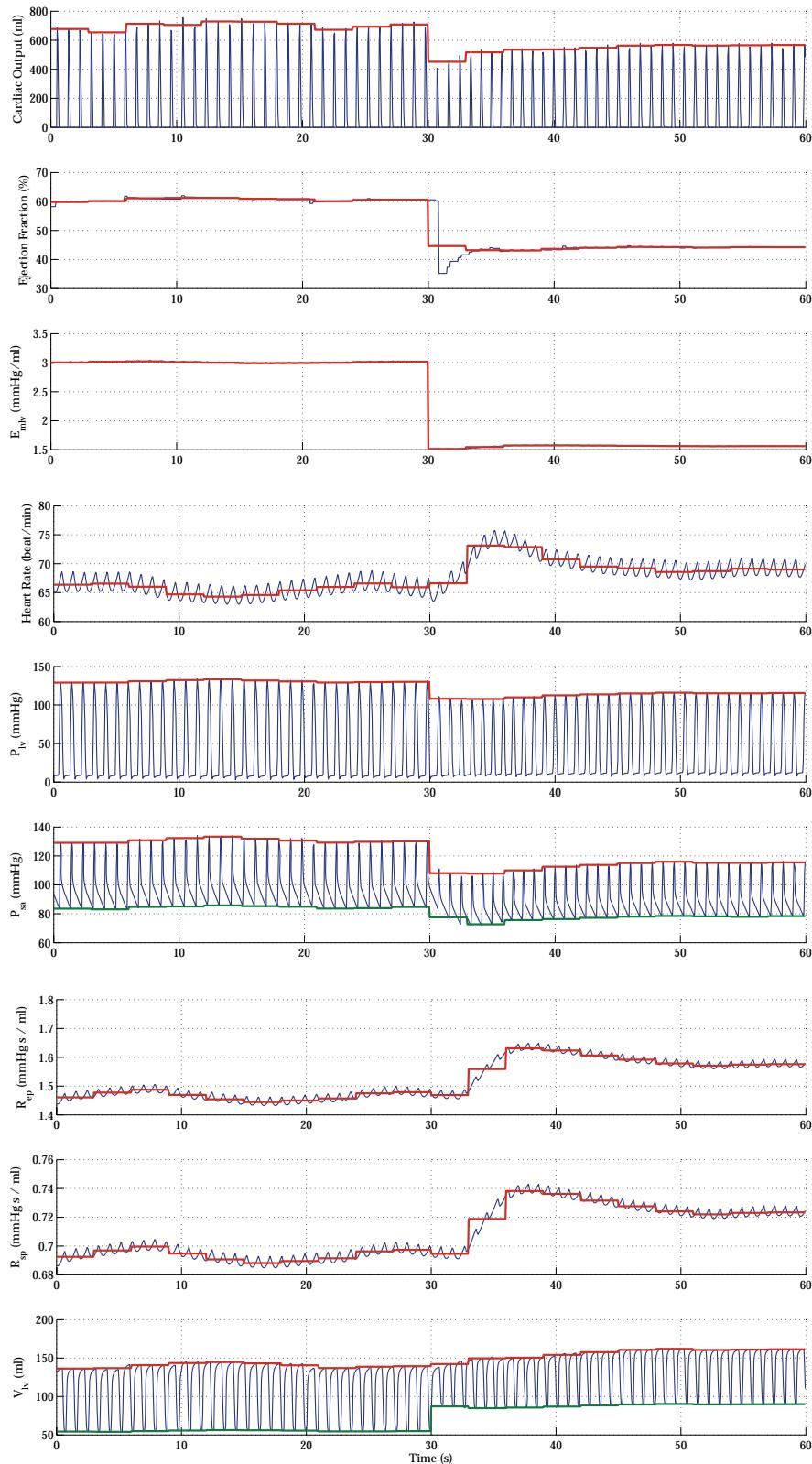


Figure 11.2: The development of the variables in the cardiovascular system of an arbitrary patient measured for a period of 60 [s]. At $t = 30$ [s], the patient suffers from a heart failure.

and R_{ep} . The following arcs are introduced: $E_{mlv} \rightarrow P_{sa\uparrow}$, $R_{ep} \rightarrow P_{sa\uparrow}$, $R_{ep} \rightarrow P_{sa\downarrow}$, $R_{sp} \rightarrow P_{sa\uparrow}$, $R_{sp} \rightarrow P_{sa\downarrow}$, and $P_{sa\uparrow} \rightarrow P_{sa\downarrow}$. The arc from E_{mlv} to $P_{sa\downarrow}$ is omitted, because its influence is minimal. Furthermore, although from a causal perspective it would be better to have the arc going from $P_{sa\downarrow}$ to $P_{sa\uparrow}$, from a computational perspective it is better to add the arc the other way around, because this saves a significant number of CPT parameters to be specified.

Most vessels have circulatory muscles, which gives them the ability to contract. If a vessel contracts, its resistance increases (so R_{ep} and/or R_{sp} increase). When P_{sa} decreases, vessels will contract reflexively trying to stabilize the blood pressure. When you use specific parts of your body, its vessels dilate to provide enough blood. This mechanism regulates the amount of blood going to particular parts of the body. For example, when you exercise, your peripheral muscles will have a larger need for blood, while when you just had some food, the vessels to your stomach and intestines will dilate, so those parts of the body will get more blood. That is why exercising right after having a meal is not a good idea.

Vessels also react to reflexes on temperature. In case you are hot, your vessels will dilate trying to loose the heat and P_{sa} will go down. This is why some people feel dizzy after standing up from a lying position in a hot environment, because P_{sa} dropped so much that for a short period the brain does not get enough blood. Fortunately, this is compensated for very quickly.

In the vessels the baroreceptors are allocated, which sense the value of P_{sa} . They initiate a reflex that influences the heart rate HR and thus we introduce $P_{sa\uparrow} \rightarrow HR$ and $P_{sa\downarrow} \rightarrow HR$. HR also influences P_{sa} , i.e. the higher the heart rate, the higher the blood pressure. This is only true as long as the heart has enough time to fill with blood. If the HR gets too high, P_{sa} will go down because there is only a little amount of blood that leaves the heart at every stroke. We model this by introducing $HR \rightarrow P_{lv\uparrow}$.

The ejection fraction EF is determined by the volume and pressure of the left ventricle, thus we add the following arcs: $V_{lv\uparrow} \rightarrow EF$, $V_{lv\downarrow} \rightarrow EF$, and $P_{lv\uparrow} \rightarrow EF$. The cardiac output CA_\uparrow is related to the heart-rate and the volume of blood leaving the heart. This introduces the followings arcs: $EF \rightarrow CA_\uparrow$ and $HR \rightarrow CA_\uparrow$.

Shock (*Shock*) means that the patient does not get enough blood (oxygen) into his brain. Whether this occurs or not depends on the blood pressure and the cardiac output. Shock can have many different causes, e.g. a bleeding (circulating volume of blood decreases too much) or an allergic reaction (all the vessels dilate, so the volume of vessels increases a lot. This leads to a relative lack of blood volume). Also a failure of the heart can lead to a shock. This is called a cardiogenic shock and is the phenomenon that is modeled in this DBN. To determine whether the patient is having a shock or not we introduce the following arcs: $P_{sa\uparrow} \rightarrow Shock$, $P_{sa\downarrow} \rightarrow Shock$ and $CA_\uparrow \rightarrow Shock$.

The above model describes the causal relation between the variables as closely as possible. However, we did not yet introduce temporal reasoning into the presented model. First of all, we decided not to insert temporal relations between different variables, because most of such relations are short-term (i.e. during a heart-beat) and the DBN model is relatively long-term. Furthermore, we wanted to be able to compare the performance of a BN model with the performance of a DBN model for the same domain. Hence, the model we have defined until so far is the BN model of the cardiovascular system.

The BN model is extended by the observation that the variables in the model not only depend on other variables, but also on their previous values. We decided to stick with a first-order temporal model and we added a temporal arc with a temporal order of one to almost every variable in the model. The only variable that does not have such a temporal arc is the *Shock* variable. This is because a cardiogenic shock only depends on current values of CA and P_{sa} and not on whether the patient was in shock in the previous time-step. The resulting DBN model is shown in figure 11.3. The three variables that are of major importance to physicians are the CA , EF , and *Shock* variables.

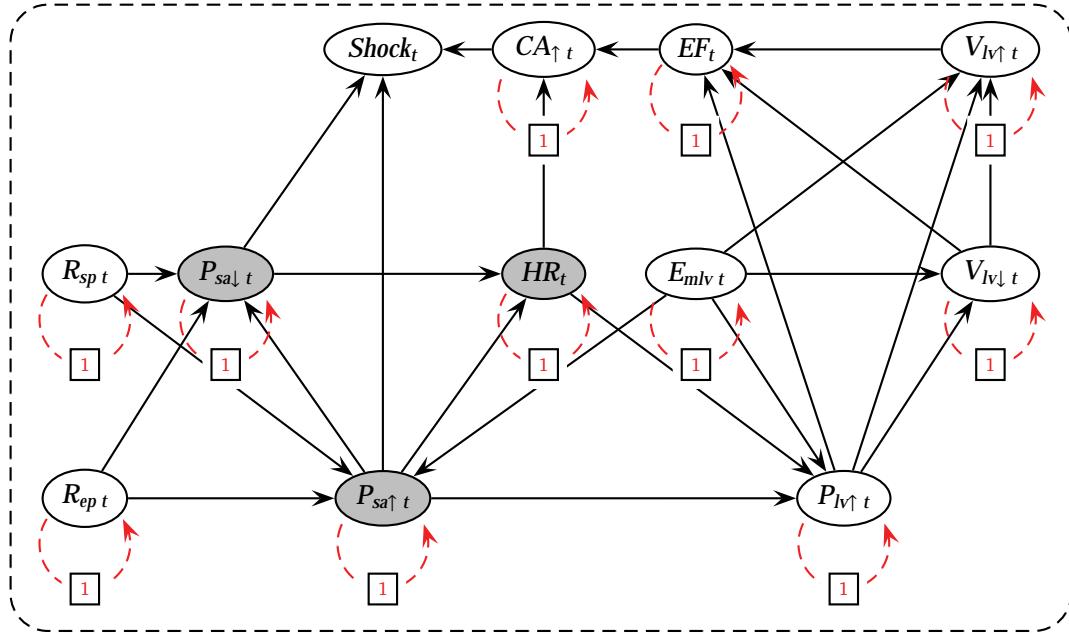


Figure 11.3: The DBN model of the cardiovascular system for a time-granularity of 3 [s].

11.3 DBN parameters

After obtaining the DBN structure, the parameters needed to be learned from the dataset provided by Philips Research. As previously stated, the data set consisted of 2933 patient simulations of 100 [s] per patient, sampled every 50 [ms]. The variables of the patient simulations were stored in Matlab .mat files of about 200 patients each. Before we could learn the DBN parameters, the data needed to undergo several preprocessing steps:

- Merge data and cut out individual patients,
- Manipulate and sample variables,
- Divide data into learn and test set, and
- Discretize variables.

An example of the course of the variables of an arbitrary simulated patient was already given in the figure 11.2, where the blue line represents the data set as provided by Philips Research. After cutting out individual patient simulations of 60 [s] each, we needed to sample the data to 3 [s] between each time step². Ordinary decimation would give erroneous results due to aliasing, so before the actual sampling step, we needed to extract the useful properties of the signal. The red and green lines represent the useful properties that we extracted from the original data. For instance, if we need a measure of P_{sa} over the last 3 [s], it is better to split this measure into two separate values: the mean systolic and the mean diastolic pressure. This manipulation is done for all variables of the cardiovascular system. Note that it might be better for CA to have taken the integral over the signal instead of its peaks. However, taking the peaks will do for now. Besides, the ratio between the peaks and the surface under the signal is pretty constant anyway.

After sampling, we divided the dataset into two sets, one for learning and one for testing. Machine-learning has a rule of thumb for the sizes of these sets (90% for learning, 10% for

²A simulation of 60 [s] was chosen to keep the amount of data before and after the heart failure of equal size. This has the effect that the prior distribution for a heart failure is equally distributed.

testing) that we choose to obey. This resulted in a learning dataset of 2640 patients and a testing dataset of 293 patients. Another rule in machine-learning is to never use any properties of the testing dataset for learning, including the assumptions that you base your model on. In this light, we needed to discretize the variables based on properties of the learning dataset. Remember that from the experiments in chapter 10, we learned that naively applying an unsupervised discretization algorithm is not optimal. That is why we choose to discretize the variables in ten uniform intervals each, based on domain knowledge and the properties of the variables.

To give some insight into the ranges, distributions and course of the variables over time, we present two figures: figure 11.4 shows the ranges and distribution of the variables as a whole and figure 11.5 shows the ranges and distribution of the variables over time, where each simulated patient suffers from a heart failure at time $t = 30$ [s]. After discretization, we

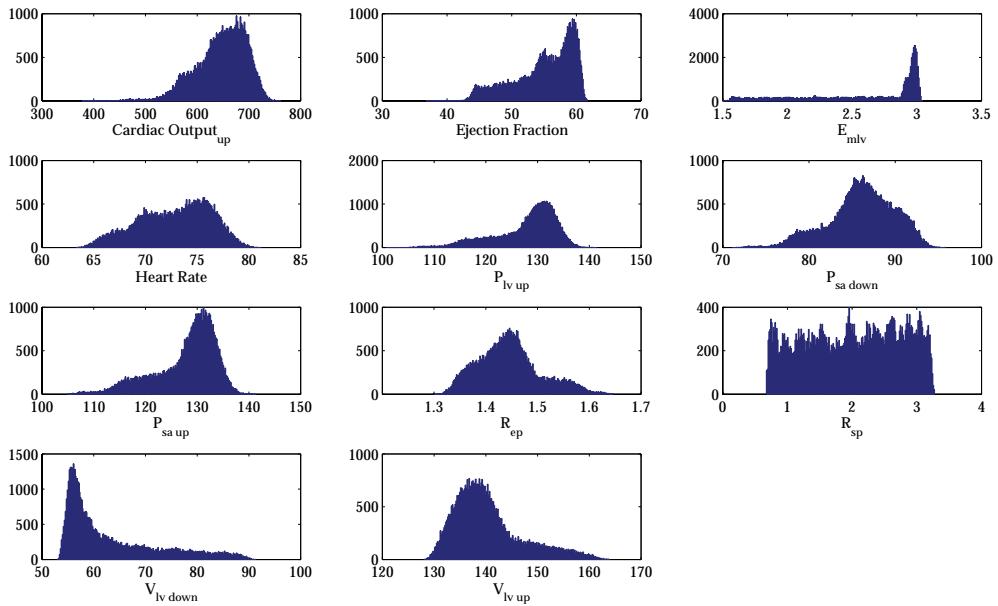


Figure 11.4: Histograms for all continuous variables in the cardiovascular system.

added the binary *Shock* variable and exported the data set to a .csv file that we can import in SMILE for learning the DBN parameters using the methodology presented in chapter 9. It might be interesting for the reader to see how this practically works out. For this reason we added appendix C that contains a snippet of the .csv file and the C++ tool that was written to define the cardiovascular system DBN structure and learn its parameters using the SMILE API.

11.4 Empirical study

After obtaining the cardiovascular DBN, we performed an empirical study on its performance. This section presents the results of two experiments. The first experiment compares the performance of the DBN with a BN of the cardiovascular system. The second experiment measures the performance of the DBN for a varying measurement rate. Note that the second experiment is not possible for the BN, because the DBN explicitly uses evidence from other time-steps to infer the values for time-steps for which no evidence is inserted.

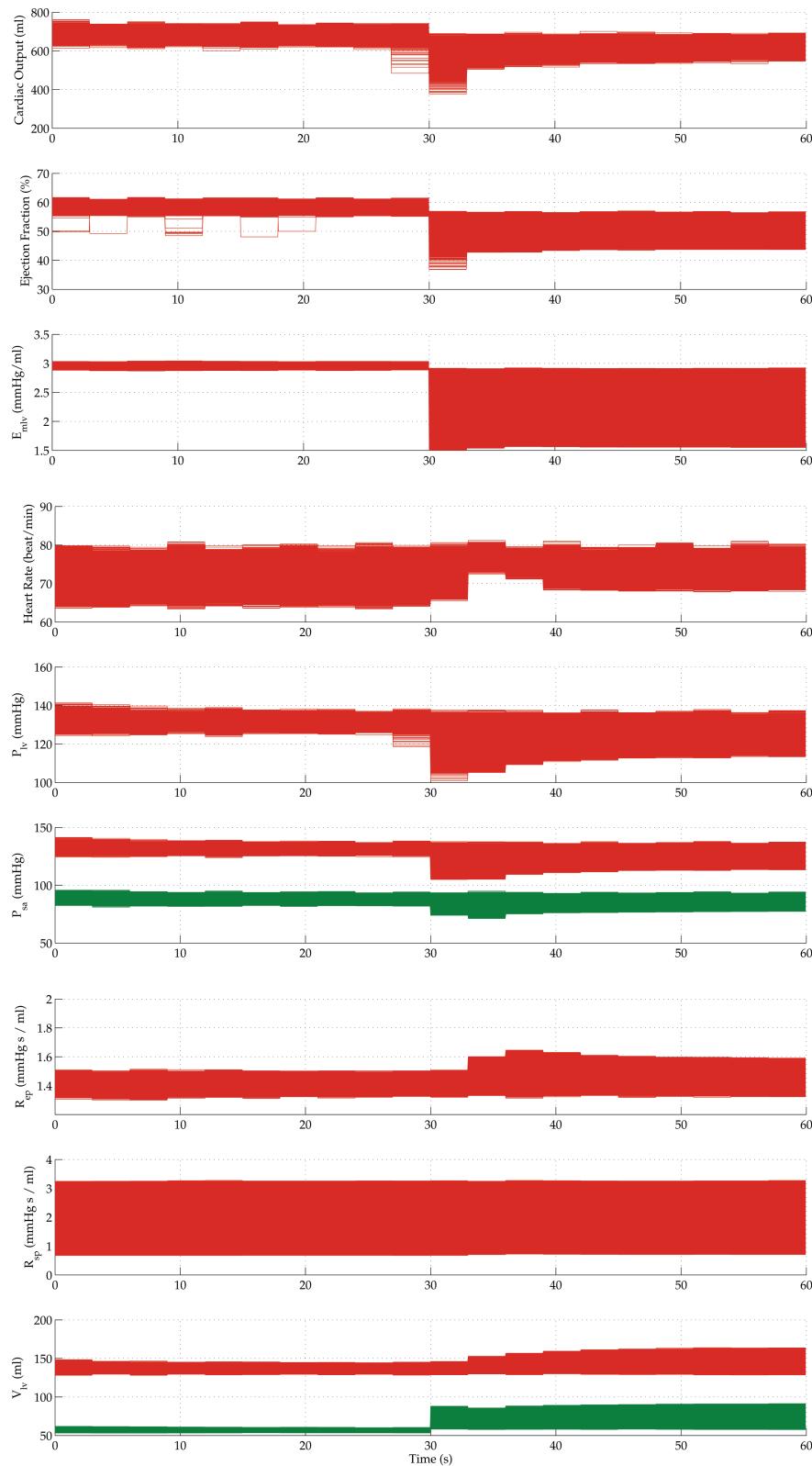


Figure 11.5: The range and distribution of the course of all continuous variables in the cardiovascular system.

11.4.1 Experimental design

For the empirical study on the cardiovascular system DBN, we are interested in two research questions: *How well does the DBN perform when compared to a BN?* and *How well does the DBN perform for a varying measurement rate?* We use the testing dataset of 293 patient simulations of 60[s], where from $t = 30$ [s] onwards, the patients suffer from a heart failure. Each time-slice in the DBN represents 3 [s] of the patient simulation, so the DBN will be unrolled for 20 time-slices during inference.

Because the provided dataset does not show a trend before a heart failure occurs (a heart failure is simulated by an abrupt change for the E_{mlv} variable), it cannot be used to predict when the heart failure occurs. However, it can be used to infer values for all variables in the cardiovascular system based solely on fluctuations in heart-rate (HR) and arterial blood pressure ($P_{sa\uparrow}$ and $P_{sa\downarrow}$). Figure 11.6 shows the time-period involved in the experiments. Note that we do not have a 0th time-slice here, because the inserted observations do not consist of values at time t , but of mean values for a period of 3 [s] of HR , $P_{sa\uparrow}$ and $P_{sa\downarrow}$ before time t . It is not physically possible to have a mean value of the previous 3 [s] at time $t = 0$ [s].

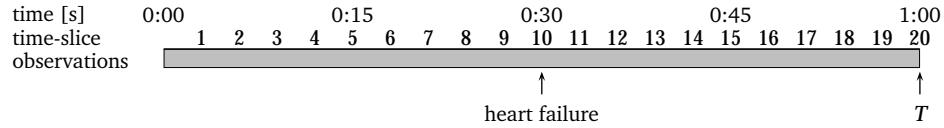


Figure 11.6: Inference in the cardiovascular system DBN. The shaded region represents the observations consisting of mean values over 3 [s] for HR , $P_{sa\uparrow}$ and $P_{sa\downarrow}$. The observation rate is varied in the second experiment.

11.4.2 Special-purpose software

In section 10.4 we briefly discussed the special-purpose software needed to perform the glucose-insulin regulation DBN experiments. To perform the experiments for the cardiovascular system DBN, these software tools were adapted to incorporate the properties of the cardiovascular system variables. Figure 11.7 shows the experimental design for the experiments. Again, we use the mean error (ME) and mean Hellinger distance (MHD) as performance measures. For an explanation on the performance measures, the reader is directed to section 10.4.

Because the cardiovascular system DBN turned out to be too complex for exact inference (clustering algorithm), we used the state-of-the-art EPIS sampling algorithm [YD03] as an approximation. This algorithm uses loopy belief propagation to compute an estimate of the posterior probability over all nodes of the network and then uses importance sampling to refine this estimate. After some experimentation with the EPIS sampling parameters to minimize the errors due to sampling, we decided to configure the EPIS sampling algorithm using the parameters shown in table 11.1. Finally, figure 11.8 shows the Matlab representation of the inferred values for the arbitrary patient that was presented in figure 11.2.

Table 11.1: EPIS sampling algorithm parameters

parameter	value	parameter	value
propagation length	60		
samples	100000	ϵ otherwise	5e-005
# states small	5	ϵ small	0.006
# states medium	8	ϵ medium	0.0001
# states large	20	ϵ large	5e-005

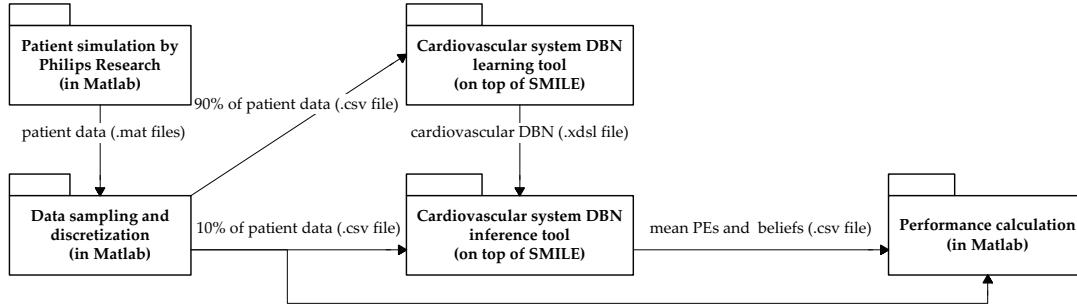


Figure 11.7: The experimental design of every experiment.

11.4.3 Experiment 1: Comparison of DBN and BN model

The first experiment compares the performance of the DBN and the BN of the cardiovascular system. Two inference types were applied to the DBN: filtering and smoothing. Remember that filtering only uses evidence up to time t , where smoothing also uses evidence after time t if available. The HR , P_{sal} , and $P_{sa\uparrow}$ variables of the testing dataset of 293 patient simulations served as evidence for the smoothed DBN, filtered DBN and BN. All other variables needed to be inferred.

Results Although for a physician, only the CA_{\uparrow} , EF , and $Shock$ variables are essential, we decided to show the performance for all inferred variables. Table 11.2 shows the overall performance of the three networks. Figure 11.9 shows the performance of the three networks over time.

Table 11.2: Overview of results for experiment 1: Comparison smoothed and filtered DBNs and BN model.

	CA_{\uparrow}				EF				E_{mlv}			
method	ME state	ME value	ME disc.	MHD	ME state	ME value	ME disc.	MHD	ME state	ME value	ME disc.	MHD
DBN smoothing	0.42	17.06	8.75	0.89	0.42	0.95	0.52	0.85	0.22	0.07	0.04	0.42
DBN filtering	0.47	18.67	8.75	0.91	0.52	1.13	0.52	0.88	0.36	0.09	0.04	0.55
BN	0.56	21.90	8.75	0.95	0.95	1.97	0.52	0.99	0.99	0.20	0.04	0.81
	$P_{I\downarrow\uparrow}$				R_{ep}				R_{sp}			
method	ME state	ME value	ME disc.	MHD	ME state	ME value	ME disc.	MHD	ME state	ME value	ME disc.	MHD
DBN smoothing	0.01	1.01	0.99	0.14	0.38	0.01	0.01	0.75	0.21	0.10	0.08	0.51
DBN filtering	0.03	1.10	0.99	0.26	0.43	0.02	0.01	0.81	0.42	0.15	0.08	0.72
BN	0.05	1.14	0.99	0.47	0.94	0.03	0.01	1.00	0.91	0.28	0.08	1.01
	$Shock$				$V_{I\downarrow\downarrow}$				$V_{I\uparrow\uparrow}$			
method	ME state	ME value	ME disc.	MHD	ME state	ME value	ME disc.	MHD	ME state	ME value	ME disc.	MHD
DBN smoothing	0.10	N/A	N/A	0.46	0.25	1.64	1.09	0.61	0.60	2.55	1.00	0.92
DBN filtering	0.11	N/A	N/A	0.48	0.31	1.87	1.09	0.67	0.60	2.56	1.00	0.94
BN	0.19	N/A	N/A	0.53	0.76	3.80	1.09	0.85	0.99	4.07	1.00	1.02

Discussion It should not come as a surprise that the smoothed DBN outperforms the filtered DBN, which in its turn outperforms the BN. Furthermore, the largest error occurs at the moment of the heart failure, because it takes a while for the DBNs to adapt. The BN is not able to adapt to the situation after the heart failure at all and thus the high error remains.

What is interesting to see is that the BN does not perform much worse than a DBN for a stable cardiovascular system (= the cardiovascular system before the heart failure). However,

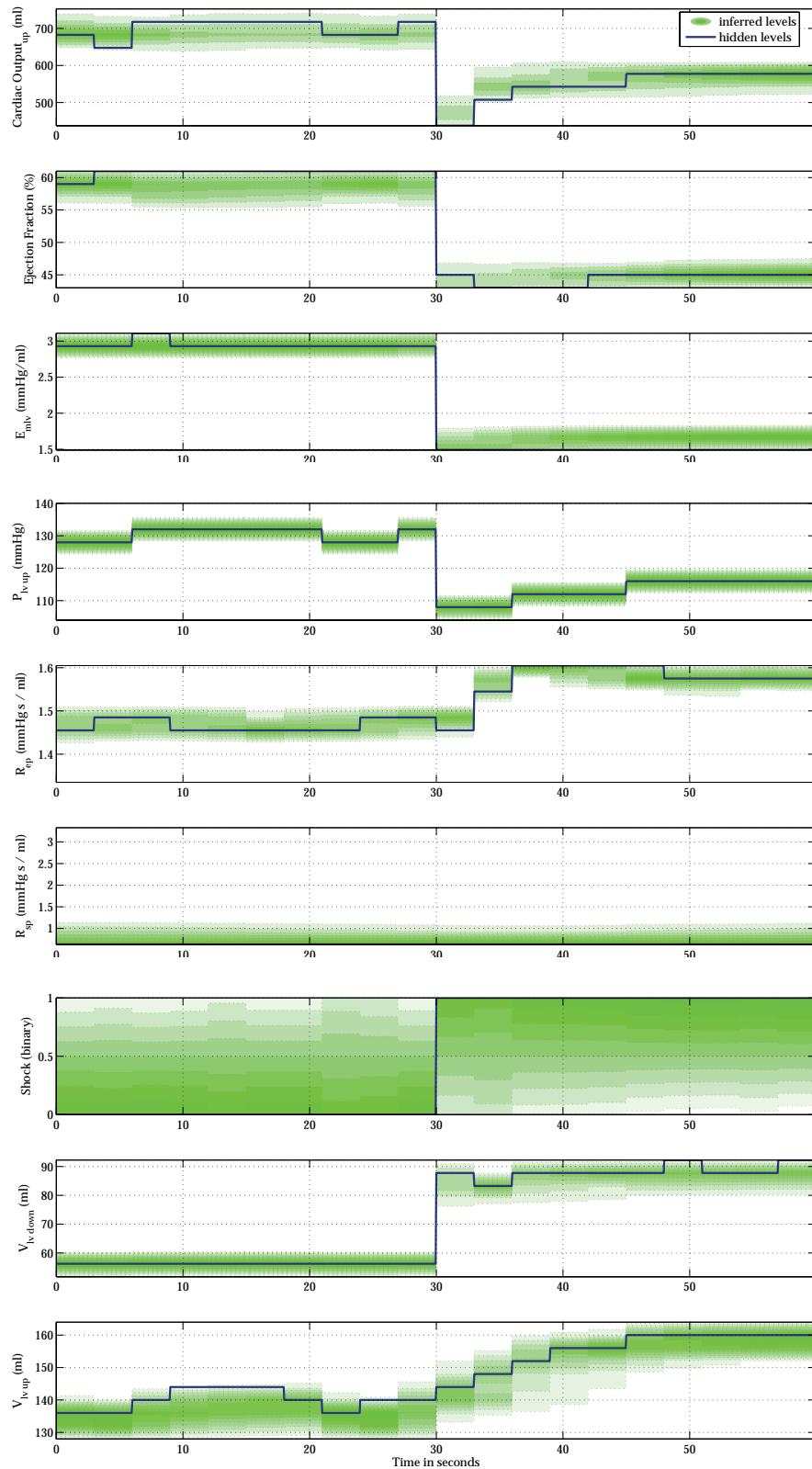


Figure 11.8: The output of the cardiovascular system DBN in Matlab for the arbitrary patient in figure 11.2.

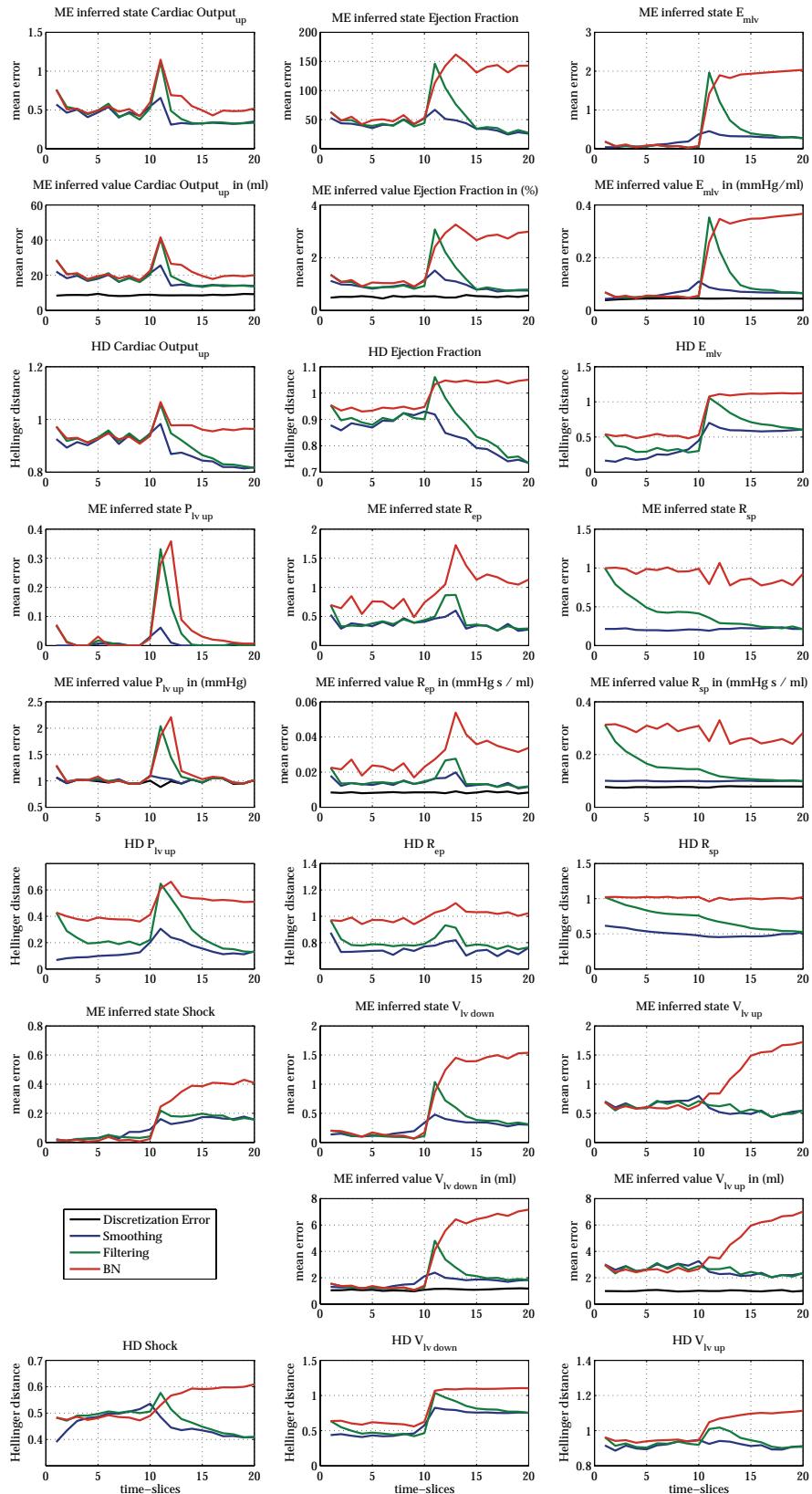


Figure 11.9: The mean errors and Hellinger distances of the variables over time for experiment 1.

after the heart failure the BN is not able to fully compensate for the change. We can partly explain this behavior by looking at the distributions of the dataset in the previous section. Before the heart failure, all patients expose more or less the same behavior, but every patient responds very different to a heart failure. This behavior can be caught in a DBN, but not in a BN.

There is one exception to this observation: the performance for inferring R_{sp} is too difficult for the BN in any case. This can be explained by looking at the dataset, where we see that R_{sp} varies much between patients, even before the heart failure.

During our investigation of individual inferred patients, we noticed one more thing. Basically, when we manipulated the dataset during the preprocessing step, we naively labeled the time-period before $t = 30$ [s] with *Shock = false* and after $t = 30$ [s] with *Shock = true*. However, this is not 100% correct, because some patients are able to compensate for the heart failure and do not get into a shock. Remarkably, the inference results for *Shock* showed that the DBN was somehow able to recover from the erroneous labeling and for some patients, the inferred values were closer to reality than our original labeling. We did not incorporate this observation into the overall results, but it shows us two things: because a DBN is based on statistics, it can be quite robust when learning from datasets with incorrectly labeled entries, and a DBN can indeed give us more insight into the real properties of the modeled problem.

Overall, we can conclude that for changes in complex systems such as the cardiovascular system, values of hidden variables can only be inferred within an acceptable error range when taking temporal dependencies into account. As long as the complex systems is stable, a BN can be sufficient.

11.4.4 Experiment 2: Varying the measurement rate

The second experiment compares the performance of the DBN for varying measurement rates to test its robustness for missing measurements. Measurements of the HR , $P_{sa\downarrow}$, and $P_{sa\uparrow}$ were inserted every time-slice, once every two time-slices, or once every four time-slices. Smoothing was used as inference type.

Results Again, we decided to show the performance for all inferred variables. Table 11.3 shows the overall performance for varying measurement rates. Figure 11.10 shows the performance over time.

Table 11.3: Overview of results for experiment 2: Varying the measurement rate.

	CA_{\uparrow}				EF				E_{mlv}			
data-rate	ME state	ME value	ME disc.	MHD	ME state	ME value	ME disc.	MHD	ME state	ME value	ME disc.	MHD
every slice	0.42	17.06	8.75	0.89	0.42	0.95	0.52	0.85	0.22	0.07	0.04	0.42
1of2 slices	0.50	19.40	8.75	0.92	0.51	1.12	0.52	0.89	0.28	0.08	0.04	0.50
1of4 slices	0.62	23.45	8.75	0.99	0.89	1.85	0.52	0.99	0.65	0.14	0.04	0.67
	$P_{lv\uparrow}$				R_{ep}				R_{sp}			
data-rate	ME state	ME value	ME disc.	MHD	ME state	ME value	ME disc.	MHD	ME state	ME value	ME disc.	MHD
every slice	0.01	1.01	0.99	0.14	0.38	0.01	0.01	0.75	0.21	0.10	0.08	0.51
1of2 slices	0.18	1.35	0.99	0.48	0.42	0.02	0.01	0.81	0.28	0.11	0.08	0.60
1of4 slices	0.42	2.09	0.99	0.76	0.60	0.02	0.01	0.91	0.47	0.16	0.08	0.81
	$Shock$				$V_{lv\downarrow}$				$V_{lv\uparrow}$			
data-rate	ME state	ME value	ME disc.	MHD	ME state	ME value	ME disc.	MHD	ME state	ME value	ME disc.	MHD
every slice	0.10	N/A	N/A	0.46	0.25	1.64	1.09	0.61	0.60	2.55	1.00	0.92
1of2 slices	0.12	N/A	N/A	0.50	0.32	1.88	1.09	0.68	0.65	2.76	1.00	0.95
1of4 slices	0.16	N/A	N/A	0.60	0.59	3.04	1.09	0.80	0.87	3.62	1.00	1.02

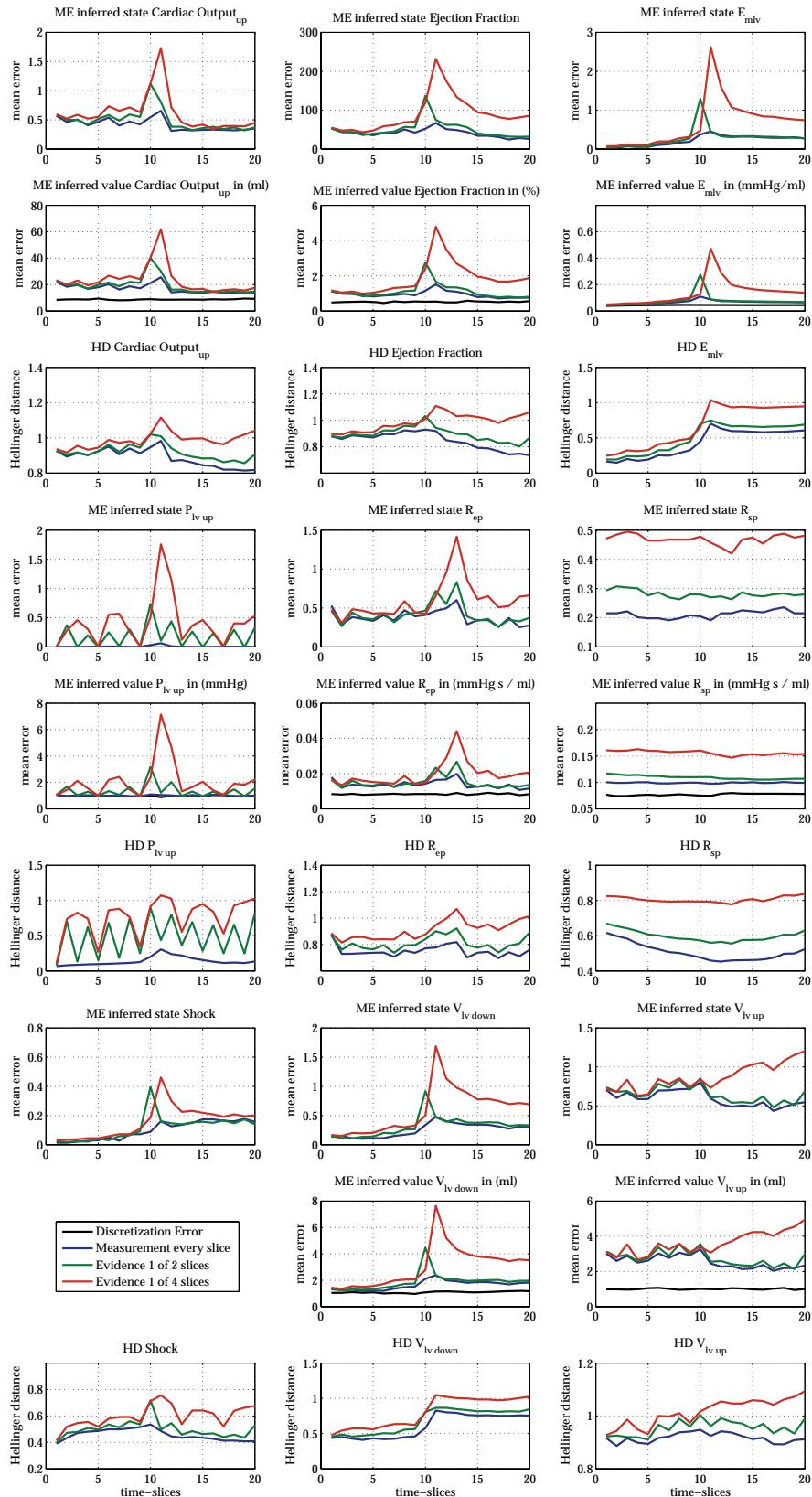


Figure 11.10: The mean errors and Hellinger distances of the variables over time for experiment 2.

Discussion We can clearly see that the performance does not change much when we do measurements every time-slice or once every two time-slices. From this observation, we can conclude that the DBN is quite robust for occasional missing measurements. However, when we do measurements once every four time-slices, we see a drastic performance drop.

The inference performance for each variable shows globally the same trend, except for the $P_{M\uparrow}$ variable, where we see some kind of *saw tooth* pattern occur, with minimal errors when a measurement is inserted. After inspection of the $P_{M\uparrow}$ CPTs, we see that $P_{M\uparrow}$ is very strongly correlated with $P_{sa\uparrow}$. In fact, for certain values it is a fully deterministic relation! As such, leaving out $P_{sa\uparrow}$ measurements has a very strong impact on the inference performance of $P_{M\uparrow}$, hence the *saw tooth* pattern.

Overall, we can conclude that the cardiovascular system DBN is robust for occasional missing measurements, as long as they stay occasional. Occasional in this context means not more than two successive missing measurements before a new measurement is done.

11.4.5 Conclusions

The empirical study gave us more insight into applying the DBN formalism to a complex physiological process such as the cardiovascular system. Overall, we are quite content with the performance of the DBN, especially when compared to a BN, and when taking into account that the DBN presented in this chapter is way more complex than the glucose-insulin regulation DBN in chapter 10. The main insights are:

1. We gained this insight before when investigating the glucose-insulin regulation DBN, but we will repeat it here: investigation of the learned CPTs suggested that different parametric representations are suitable, because many parameters remain untouched during the learning phase.
2. A BN can model a complex system such as the cardiovascular system only when no changes over time are involved. Otherwise, a DBN is needed.
3. A DBN is able to recover from wrongly labeled values for variables during learning. As such, it can be used to get more insight in the actual properties of the modeled domain.
4. A DBN is robust for missing measurements, as long as there are not too many missing measurements. The number of missing measurements differs from application to application. In the cardiovascular system DBN, this means not more than two successive missing measurements before a new measurement is done.

11.5 Summary

In this chapter, we presented the first step toward a cardiovascular system DBN that is able to infer the hidden values of the physiological process based solely on fluctuations in heart-rate (HR) and arterial blood pressure ($P_{sa\uparrow}$ and $P_{sa\downarrow}$). An empirical study gave insight into the performance of the DBN when compared to a BN and when missing measurements are involved. Furthermore, it showed us that a DBN is able to recover from wrongly labeled values for variables during learning and that it can be used to get more insight in the actual properties of the modeled domain.

Part IV

Results

*"The most exciting phrase to hear in science,
the one that heralds the most discoveries,
is not Eureka! but That's funny..."*
- Isaac Asimov.

Chapter 12

Conclusions and future work

The observant reader probably noticed the quote of Isaac Asimov as introduction of this part of the thesis. Although we do not want to claim that our exclamations of *That's funny!* herald great discoveries, many interesting results were due to the fact that the empirical studies on our extended DBN formalism, discretization of continuous variables, the glucose-insulin regulation DBN and the cardiovascular system DBN did not yield the expected results. And that was indeed funny. And useful.

In this chapter we review our initial goals and objectives and show to what extent they were satisfied. Furthermore, our contribution to the field is presented. Finally, we present five opportunities for future work.

12.1 Goals and objectives

In the introduction of this thesis, we proposed the following research questions:

1. *How do we obtain the DBN structure and parameters?*
2. *What is the performance of inference?*
3. *How much patient data do we need for learning the DBN parameters?*
4. *If possible, how much does a DBN outperform a BN when applied to the same problem?*

Based on these questions, we formulated our assignment. The original assignment consisted of three parts: a *theoretical* part, an *implementation* part, and an *application* part. In their turn, these three parts consisted of our goals and objectives of the research presented in this thesis. What follows is a one-by-one discussion of the satisfaction of these goals and objectives.

12.1.1 Theoretical part

1. Investigating existing DBN theory We performed a survey on current (D)BN research and its applications and presented the results of this investigation in chapters 3 and 5. We tried to present our findings as clearly as possible and included many references to in-depth material, so it can serve as a starting point for readers that are trying to get familiar with DBN theory and techniques. Our investigation also resulted in a proposal for the extension of Murphy's DBN formalism (chapter 7) and a description of a framework for applying the DBN formalism to physiological processes (chapter 6).

2. Extending DBN formalism Based on our findings from objective 1, we decided to extend Murphy's DBN formalism with several new concepts because it has several unnecessary limitations due to its restrictive definition. The most obvious limitation of Murphy's DBN formalism is the possibility to model first-order Markov processes only. A second limitation is that when unrolling the network for inference, every node is copied to every time-slice, even if it has a constant value for all time-slices. The final limitation is that although it is possible to introduce a different initial state, it is not possible to define a different ending state, which can be useful for modeling variables that are only interesting after the end of the process.

To overcome these limitations, we introduced several extensions of Murphy's DBN formalism in chapter 7. First, we introduced k^{th} -order temporal arcs to enable the modeler to model processes with complex time-delays. Second, because we found the original definition of DBNs using the (B_1, B_{\rightarrow}) pair too cumbersome for representation of $(k + 1)$ TBNs, we introduced the concept of a *temporal plate*. We argued that the temporal plate representation is both easier to understand and more intuitive to model with. Furthermore, it simplifies the implementation of the model significantly by decreasing the amount of code (and thus the number of potential errors) needed to specify the model. The introduction of a temporal plate also made way to handle nodes within the temporal plate differently from nodes outside the temporal plate. This lead to our introduction of *anchor*, *contemporar*, and *terminal* nodes. Anchor nodes are only connected to the first time-slice in the temporal model and provide the modeler with means of adding nodes to the temporal model that only affect the initial phase of the modeled process. Similarly, terminal nodes provide the modeler with means of adding nodes to the temporal model that are only interesting at the end of the modeled process. With Murphys formalism, these nodes would have to be copied (and inferred) every time-slice. The introduction of contemporar nodes enables the modeler to model the *context* of a process by providing nodes that are connected to every time-slice in the temporal plate, but are not copied themselves during unrolling. Such *context* nodes have two main purposes. First, they can serve as evidence nodes that are used to index the nodes in the temporal plate for switching between different conditional probability distributions. Second, if the state of the context nodes is unknown, they can serve as classifiers based on temporal observations from nodes in the temporal plate.

Finally, we showed in chapter 7 that for our extension of the DBN formalism, next to the obvious advantages modeling wise, also the performance in terms of execution time and memory usage increases significantly.

3. Adapting DBN parameter learning In chapter 9 we discussed issues concerning the DBN structure, the DBN parameters, and discretization of continuous variables. Although we were to obtain the DBN structures of our applications by using domain knowledge, the parameters needed to be learned from data. During our study of existing DBN theory, we found that most literature states that DBN parameter learning is essentially the same as BN parameter learning, with only a few small differences. Existing DBN parameter learning algorithms unroll the DBN for the to-be-learned sequence length after which the parameters of similar CPDs are tied and the DBN is learned using an adapted BN learning algorithm. We proposed a methodology that decomposes the DBN to several BNs (initial parameters, temporal parameters, contemporar parameters, and terminal parameters) instead and use an (unaltered) BN parameter learning

algorithm for several reasons. First of all, by decomposing the DBN, we do not have to deal with unrolling the DBN and tying the parameters for the unrolled DBN. This saves a lot of overhead. Secondly, in the case where the initial parameters represent the stationary distribution of the system (and thus become coupled with the transition model), they cannot be learned independently of the transition parameters. Existing DBN parameter learning algorithms need to be adapted for this situation, but for our method it does not matter whether we learn the parameters for a DBN with coupled or decoupled initial and transition parameters (because of our introduction of anchor nodes). Finally, our methodology is more practical to apply. While there exist many fast and reliable implementations of BN parameter learning, this is not the case for DBN parameter learning; most likely because it is difficult to generalize DBN parameter learning for all possible network topologies.

We found that an important, but generally overlooked preprocessing step for learning the parameters of continuous variables with DBNs is their discretization. We presented several methods for discretization of continuous variables and showed their differences. We showed that different methods have different advantages (chapter 9), but also showed that a wrongly chosen discretization can have a major impact on the performance of the DBN (chapter 10). We decided to apply a discretization of continuous variables based on expert knowledge, but development of an automated optimal discretization method clearly poses a window of opportunity for future work.

4. Validating the extended DBN formalism We support our claims on the relevance of our extension of the DBN formalism, application of our DBN parameter learning methodology, and discretization of continuous variables by means of several smaller and larger empirical studies performed throughout this thesis. Next to a validation of our approach, the empirical studies presented general insight into the overall performance of different methodologies and provided knowledge on how to apply the DBN formalism for modeling the temporal domain in general.

Chapter 7 presents an empirical study that validates our extension of the DBN formalism and gives insight into the performance gain in terms of memory and time. Our parameter learning methodology was validated by applying it to the glucose-insulin regulation and cardiovascular system DBNs in chapter 10 and 11. Empirical studies on the discretization of continuous variables were presented in chapters 9 and 10.

12.1.2 Implementation part

5. Investigating current DBN software To get insight into the functionality of current DBN software, several packages were downloaded, installed, and tested. The results of this study were presented in chapter 4, where we concluded that no satisfying solution existed yet. Current solutions lack one or more essential features, such as different inference types (filtering, prediction, smoothing, Viterbi), different node representations (CPT, Noisy-MAX, continuous distribution), different inference algorithms (exact inference, approximate inference), and/or importing/exporting DBN models. Furthermore, they are generally inflexible, too slow and do not have a GUI. Based on these insights, we saw a need for the development of a flexible, fast, reliable and easy-to-use implementation of temporal reasoning. As such, the DSL software solutions SMILE and GeNIe were extended with temporal reasoning using our extension of the DBN formalism as a guideline.

6. Designing and implementing the DBN formalism in SMILE Chapter 8 presents the design, implementation, and testing of temporal reasoning in SMILE using our extension of the DBN formalism as a guideline. We showed that our extension of SMILE with temporal reasoning is designed and implemented in such a way that it does not affect the performance of existing functionality, the extended API is consistent with the existing API, access to temporal reasoning functionality is straightforward and easy, and the temporal extension is easy maintainable and expandable.

The implementation enabled us to use SMILE for temporal reasoning purposes, and is to the best of our knowledge the first implementation of temporal reasoning that provides this much freedom and ease in temporal modeling by supporting k^{th} -order temporal arcs, a temporal plate, anchor, contemporaneous and terminal nodes, and different canonical representations. Our implementation enabled us to use SMILE for modeling physiological processes, but it can be used to model temporal problems in arbitrary domains.

7. Designing a GUI to DBNs in GeNIE After the implementation in SMILE, a GUI was designed for temporal reasoning in GeNIE based on brainstorm sessions within the DSL and feedback from users in the field. The design of the extension of GeNIE with temporal reasoning was merged into the current design of GeNIE, so existing users do not get confused due to the newly introduced functionality. The GUI design includes our introduction of the concepts of k^{th} -order temporal arcs, a temporal plate, and anchor, contemporaneous and terminal nodes. Furthermore, we designed a straightforward way to define temporal conditional probability distributions, set temporal evidence, perform inference, and visualize the temporal posterior beliefs in several ways. Initial response from the field is enthusiastic, and yields some useful suggestions for future extensions of both functionality and visualization (more on this in section 12.3). However, the current implementation is fully functional and both GeNIE and SMILE with inclusion of temporal reasoning functionality can be downloaded from <http://genie.sis.pitt.edu>.

12.1.3 Application part

8. Applying the extended DBN formalism to physiological processes in the human body After extending the DBN formalism, developing a DBN parameter learning methodology, and investigating several discretization algorithms, it was time to apply the gained knowledge by modeling two clinically relevant physiological processes: glucose-insulin regulation and the cardiovascular system described in chapter 2, and try to answer the remaining research questions by obtaining a dataset, discretizing the continuous variables, implementing the DBN models in SMILE, learning the DBN parameters, and performing empirical studies on the obtained DBNs.

Our first real application of the DBN formalism was modeling the glucose-insulin regulation in the human body (chapter 10). For derivation of the DBN structure we could rely on an existing mathematical model that we extended with nominal-valued nodes to model the patient type and the size of the insulin infusion. Furthermore, we could use the mathematical model to simulate different patient cases for learning the DBN parameters. Based on the size of a given insulin infusion and (noisy) glucose measurements, the DBN was able to infer the patient type and/or predict future glucose and insulin levels. The obtained glucose-insulin regulation DBN can be used by physicians to improve the glucose control algorithms currently in use in the ICU. However, our main goal for modeling the glucose-insulin regulation using a DBN was to serve as a primer for modeling the more complex cardiovascular system.

In contrast to the glucose-insulin regulation, there does not yet exist an effective and flexible model for the cardiovascular system. However, such a model can be of great help for the development of treatment of cardiovascular diseases, the leading cause of death in the industrialized world for over a century. Based on expert knowledge from physicians, simulated patient cases (provided by Philips Research), and our DBN knowledge, we were able to develop a new causal temporal model of the cardiovascular system that is able to infer values for important physiological variables on the basis of measuring non-evasive variables, such as heart-rate, systolic and diastolic arterial blood pressure (chapter 11). Furthermore, it is able to infer whether a patient has a cardiogenic shock, a property that can be of great clinical significance.

9. Performing empirical studies Empirical studies were performed on the glucose-insulin regulation and cardiovascular system DBNs in chapter 10 and 11. Although the glucose-insulin regulation DBN served as a primer for the cardiovascular system DBN, the results of the empirical study were useful nonetheless. First of all, it proved that our newly introduced DBN

concepts and their implementation in SMILE and GeNIe worked, next to our DBN parameter learning methodology. Furthermore, it proved that the chosen discretization of continuous variables can have a major influence on the performance of the DBN, just like we expected. Finally, it showed that not that many patient cases are needed to learn the DBN parameters as one might expect from the number of CPT parameters to be learned. Inspection of the CPTs showed that after learning, many CPT parameters kept their initial distribution, suggesting that a different canonical gate might be appropriate. This observation presents another opportunity for future work.

The empirical study on the cardiovascular system DBN showed that a DBN clearly outperforms a BN for complex systems where change is involved. The BN was able to infer the physiological variables before the heart failure, but after the heart failure, the BN is not able to adapt to the new situation. Furthermore, it showed that due to its statistical properties, a DBN is able to recover from erroneous labeled inputs, providing us (and physicians) with more insight about the actual properties of the modeled domain. Finally, the DBN formalism appears robust to missing measurements.

All in all, the empirical studies showed that our extension of the DBN formalism is well suited for application to physiological processes and can be of great help for both research and practical applications.

12.2 Contributions to the field

The previous section presented our work in terms of the satisfaction of our earlier defined goals and objectives. However, the question remains what parts of our work are a contribution to the field.

As a whole, although we developed a framework with the application of the extended DBN formalism to physiological processes in mind, we want to stress that the framework presented in this thesis can be used in other domains, such as automatic speech recognition, missile detection systems, user-modeling, etc. for classification, filtering, smoothing, Viterbi, and prediction purposes. The issues that we discussed concerning the modeling of physiological processes can be easily generalized to other domains, because every domain has to deal with modeling issues such as complex time-delays, context specific probability distributions, and discretization of continuous variables sooner or later. The research in this thesis shows how to use these concepts and what issues are involved when building temporal models. Furthermore, it clarifies the use of these concepts by applying them to relevant applications, and shows how the concepts work out in practice.

On a modeling level, we made contributions to the field by extending the DBN formalism with the notion of k^{th} -order temporal arcs, a temporal plate, and anchor, contemporaneous, and terminal nodes to counter the limitations of Murphy's DBN formalism. We developed a new approach for representation and implementation of temporal networks and showed how to utilize our extensions for modeling purposes. We claim that our extension is both more intuitive and provides more modeling power than Murphy's DBN formalism. Furthermore, implementation of temporal models using our representation results in less code and less potential errors. Empirical study showed that the performance of our extension in terms of execution time and memory usage is also significantly better than Murphy's DBN formalism.

For learning the DBN parameters, we developed a straightforward method that decomposes the DBN definition into several BNs and uses an existing BN parameter learning algorithm instead of adapting an existing BN parameter learning algorithm to a DBN version. Our methodology counters the disadvantages of existing DBN parameter learning algorithms that have to unroll the DBN for the to-be-learned sequence length and tie the parameters, and that do not provide straightforward means for parameter learning when the initial parameters and the transition parameters are coupled.

Furthermore, we shed light on the issues involved when discretizing continuous variables,

an important preprocessing step that is generally overlooked in applications using the DBN formalism, but can have a major influence on the performance of the DBN, as we have shown in an empirical study as well.

We found that although the DBN formalism is very promising, not much has happened regarding real and practical applications. We think that this is mostly due to the fact that no satisfying software solution exists yet for temporal reasoning. A very concrete contribution of this research is the extension of SMILE and GeNIe with temporal reasoning. This software is to the best of our knowledge the first implementation of temporal reasoning that provides support for k^{th} -order temporal arcs, a temporal plate, and anchor, contemporaneous and terminal variables, next to support for using different canonical representations in temporal models, such as the Noisy-MAX gate. Next to that, the software provides an intuitive modeling approach using our temporal plate representation, a straightforward way to define temporal conditional probability distributions, set temporal evidence, perform inference, and visualize the temporal posterior beliefs in several ways. During our design and implementation of the temporal reasoning extension, we took great care to provide means for future extensions, such as new temporal inference algorithms, which opens another window of opportunities for future work. Initial users of the temporal reasoning extension in GeNIe and SMILE are enthusiastic.

A contribution with clinical significance is our temporal model of the cardiovascular system. As we have stated before, cardiovascular diseases are leading cause of death in the industrialized world for over a century, but a good model of the cardiovascular system does not yet exist. We showed that the extended DBN formalism is well-suited to model (parts of) the cardiovascular system, and is even able to infer relevant physiological and clinical variables within a sufficient error range. With the help of our model, physicians are able to gain a better insight into the cardiovascular system, something that will hopefully help in treatment of cardiovascular diseases.

Finally, although some conclusions from the empirical studies performed in chapters 10 and 11 apply to specific applications, many insights are generalizable, such as: the insight that less data is needed for learning the DBN parameters than expected from the number of CPT parameters to be learned; the insight that the discretization method has a major influence on the performance of the DBN; the insight that DBNs are quite robust for missing evidence; and the insight that DBNs can recover from incorrectly labeled learning data, providing us with more knowledge about the domain. The last insight is particularly important for research in the medical domain.

12.3 Future work

Research on modeling with DBNs is currently a very active field. As such, many opportunities for future research exist. Based on the research reported in this thesis, we will give five opportunities for future research on modeling with DBNs. Some of these opportunities are mentioned briefly in previous sections, but we will give a short explanation here nonetheless.

First of all, for this research we relied on existing BN inference algorithms when performing inference on the DBNs. For these algorithms to work, the DBN needs to be unrolled for the appropriate number of time-slices. However, this approach is very inefficient regarding space-complexity. For Murphy's DBN formalism, space-efficient algorithms exist that keep only a number of time-slices in memory at every instant of the algorithm, but these algorithms only work because of the restrictive nature of Murphy's DBN formalism. An opportunity for future work can be devising and implementing a space-efficient algorithm for our extension of the DBN formalism. A difficulty that needs to be overcome when devising such an algorithm is how to efficiently deal with the large cliques that can appear during the moralization and triangulation steps of exact inference in the case of unobserved contemporaneous nodes. A more general difficulty arises when considering the DBN definition in figure 12.1a. Initially, this DBN looks rather simple. In fact, it does not even have non-temporal arcs. However, during inference the DBN

definition needs to be unrolled (figure 12.1b). Here we can clearly see that although the non-temporal arc $X_4^1 \rightarrow X_4^4$ does not exist, X_4^4 is still dependent on X_4^1 because of the path through X_1^1 ! For this reason, exact inference for DBNs is intractable most of the times. Future work could go into devising an approximate inference algorithm for the extended DBN formalism based on the EPIS sampling algorithm that makes use of the special properties of a DBN, such as its repetitive structure.

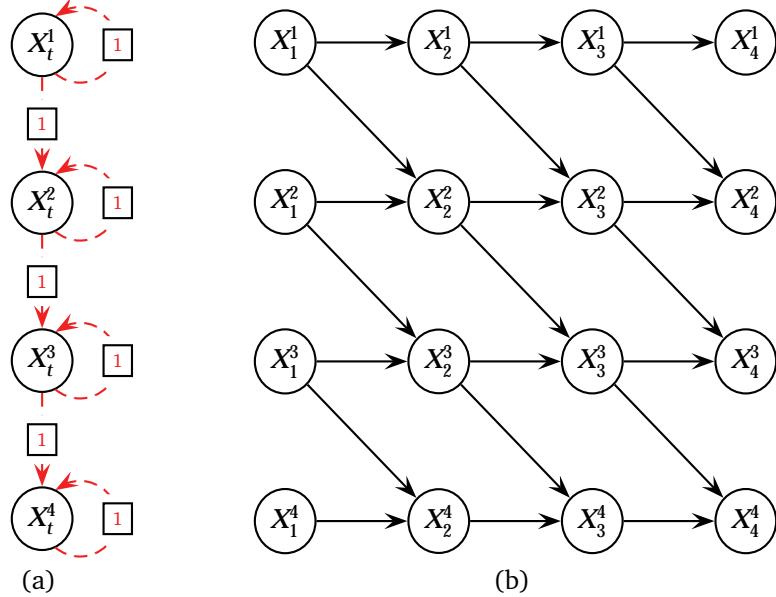


Figure 12.1: (a) DBN definition. (b) Unrolled DBN. X_4^4 becomes dependent on X_4^1 through X_1^1 .

A second opportunity for future work arises when modeling continuous variables with discrete nodes. We have shown that the discretization step can have a major impact on the performance of the DBN. However, most current work overlooks the issues involved when discretizing continuous variables. We argue that a research opportunity lies in devising a learning algorithm that includes the discretization of continuous variables as inherent part of the learning process. In this way, the learning algorithm can search for an optimal tuning between the learned CPT parameters and the chosen discretization intervals.

A third opportunity for future work is to extend the DSL software to allow for continuous variables in the DBN. Allowing for continuous variables is not without problems. For instance, unless for some cases where the continuous variables are limited to a representation as nodes with Gaussian distributions, exact inference is generally impossible and one needs to resort to approximate inference techniques. However, because (D)BNs are often used to model continuous variables, this can be a very interesting topic. A part of the research done at the DSL focuses on developing mathematical and computational solutions to the representation and reasoning of hybrid Bayesian networks consisting of a mixture of discrete and continuous variables with arbitrary distributions or equations. Results of these research efforts are currently built into SMILE and can be extended to the temporal domain in future releases.

A fourth opportunity for future work presented itself during inspection of the learned CPTs of nodes in both the glucose-insulin regulation DBN and the cardiovascular system DBN. We found that many CPT parameters remained untouched after the learning phase. This suggests that different parametric representations are suitable for these variables. However, experimenting with the Noisy-MAX representation did not yield satisfying results, so different parametric representations need to be devised. Research effort could go into finding and implementing new canonical representations specifically for discretized variables.

A fifth opportunity for future work can be found in the medical domain, where on a more practical level our cardiovascular system DBN can be extended with the *context* of patients based on patient properties, such as gender, age, smoker/non-smoker, etc. Unfortunately, this was not possible with the current dataset, because the patients were not labeled as such. Furthermore, the DBN model can be extended with other types of shock (hypovolemic, septic, etc.) to learn how well the DBN performs in classifying different types of shock.

Bibliography

- [AC96] C. F. Aliferis and G. F. Cooper. A structurally and temporally extended Bayesian belief network model: definitions, properties, and modeling techniques. In *Proceedings of the 12th conference on uncertainty in artificial intelligence (UAI-96)*, pages 28–39, Portland, OR, USA, August 1996. Morgan Kaufmann publishers.
- [AJA⁺89] S. Andreassen, F. V. Jensen, S. K. Andersen, B. Falck, U. Kjærulff, M. Woldbye, A. R. Sørensen, and A. Rosenfalck. MUNIN - An expert EMG assistant. In J. E. Desmedt, editor, *Computer-aided electromyography and expert systems*, chapter 21. Elsevier Science publishers, Amsterdam, The Netherlands, 1989.
- [Aug05] J. C. Augusto. Temporal reasoning for decision support in medicine. *Artificial intelligence in medicine*, 33:1–24, 2005.
- [Bay04] Bayesia SA. *BayesiaLAB 3.0 tutorial*, 2004.
- [BD00] B. Bruegge and A. H. Dutoit. *Object-oriented software engineering - conquering complex and changing systems*. Prentice-Hall International, Inc., Upper Saddle River, NJ, USA, 2000.
- [BK98] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proceedings of the 14th conference on uncertainty in artificial intelligence (UAI-98)*, pages 33–42, University of Wisconsin, Madison, WI, USA, July 1998.
- [BKS97] E. Bauer, D. Koller, and Y. Singer. Update rules for parameter estimation in Bayesian networks. In *Proceedings of the 13th conference on uncertainty in artificial intelligence (UAI-97)*, pages 3–13, Brown University, Providence, RI, USA, August 1997.
- [BL00] R. M. Berne and M. N. Levy. *Principles of physiology*. Mosby, third edition, 2000.
- [BMR97] J. Binder, K. P. Murphy, and S. J. Russell. Space-efficient inference in dynamic probabilistic networks. In *Proceedings of the 15th international joint conference on artificial intelligence (IJCAI-97)*, pages 1292–1296, Nagoya, Aichi, Japan, August 1997.
- [Bun94] W. L. Buntine. Operations for learning with graphical models. *Journal of artificial intelligence research*, 2:159–225, December 1994.
- [BZ02] J. A. Bilmes and G. G. Zweig. The graphical models toolkit: An open source software system for speech and time-series processing. In *Proceedings of the 2002 IEEE international conference on acoustics, speech, and signal processing (ICASSP02)*, volume 4, pages 3916–3919, May 2002.
- [Coo90] G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial intelligence*, 42:393–405, 1990.
- [CR04] N. W. Chbat and T. K. Roy. Glycemic control in critically ill patients - effect of delay in insulin administration. Mayo Clinic, Rochester, MN, USA, 2004.

- [CvdGV⁺05] T. Charitos, L. C. van der Gaag, S. Visscher, K. Schurink, and P.J.F. Lucas. A dynamic Bayesian network for diagnosing ventilator-associated pneumonia in ICU patients. In J. H. Holmes and N. Peek, editors, *Proceedings of the 10th intelligent data analysis in medicine and pharmacology workshop*, pages 32–37, Aberdeen, Scotland, 2005.
- [Dar01] A. Darwiche. Constant space reasoning in dynamic Bayesian networks. *International journal of approximate reasoning*, 26:161–178, 2001.
- [Das03] D. Dash. *Caveats for causal reasoning with equilibrium models*. PhD thesis, Intelligent systems, University of Pittsburgh, Pittsburgh, PA, USA, April 2003.
- [DDN00] R. Deventer, J. Denzler, and H. Niemann. Control of dynamic systems using Bayesian networks. In *Proceedings of the IBERAMIA/SBIA 2000 workshops*, pages 33–39, Atibaia, Sao Paulo, Brazil, 2000.
- [DK88] T. Dean and K. Kanazawa. Probabilistic temporal reasoning. In *Proceedings of the 7th national conference on artificial intelligence (AAAI-88)*, pages 524–529, St. Paul, MN, USA, August 1988. MIT Press.
- [DKS95] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In A. Prieditis and S. Russell, editors, *Proceedings of the 12th international conference on machine learning*, pages 456–463, San Francisco, CA, USA, 1995. Morgan Kaufmann publishers.
- [DML05] E. S. Dimitrova, J. J. McGee, and R. C. Laubenbacher. Discretization of time series data. <http://polymath.vbi.vt.edu/discretization>, August 2005. C++ implementation of algorithm available on website.
- [Dru99] M. J. Druzdzel. SMILE: Structural modeling, inference, and learning engine and GeNIE: A development environment for graphical decision-theoretic models. In *Proceedings of the 16th national conference on artifical intelligence (AAAI-99)*, pages 342–343, July 1999.
- [Dru05] M. J. Druzdzel. Intelligent decision support systems based on SMILE. *Software developer's journal*, 2:26–29, February 2005.
- [DS94] M. J. Druzdzel and H.J. Suermondt. Relevance in probabilistic models: "backyards" in a "small world". In *Working notes of the AAAI-1994 fall symposium series: relevance*, pages 60–63, New Orleans, LA, USA, November 1994.
- [Har01] A. Hartemink. *Principled computational methods for the validation and discovery of genetic regulatory networks*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2001.
- [Hec98] D. E. Heckerman. A tutorial on learning with Bayesian networks. In *Proceedings of the NATO advanced study institute on learning in graphical models*, pages 301–354, Norwell, MA, USA, 1998. Kluwer Academic publishers.
- [HHN92] D. E. Heckerman, E. J. Horvitz, and B. N. Nathwani. Toward normative expert systems: part I - the pathfinder project. *Methods of information in medicine*, 31:90–105, 1992.
- [HM84] R. A. Howard and J. E. Matheson. Influence diagrams. In *Readings on the principles and applications of decision analysis*, volume 2, pages 721–762, Strategic decision group, Menlo Park, CA, USA, 1984.

- [Hul05] J. Hulst. *Research Assignment: Dynamic Bayesian networks and speech recognition*. Man-machine interaction group, Faculty of EEMCS, Delft University of Technology, Delft, The Netherlands, August 2005.
- [Int04] Intel Corporation. *Probabilistic Network Library - User guide and reference manual*, March 2004.
- [Jor03] M. I. Jordan. *An introduction to probabilistic graphical models*. To appear, June 2003.
- [Kal60] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME - journal of basic engineering*, 83:35–45, 1960.
- [KCC00] M. M. Kayaalp, G. F. Cooper, and G. Clermont. Predicting ICU mortality: A comparison of stationary and non-stationary temporal models. In *Proceedings of the 2000 AMIA annual symposium*, pages 418–422, 2000.
- [KF05] D. Koller and N. Friedman. *Bayesian networks and beyond*. To appear, 2005.
- [Kjæ95] U. Kjærulff. dHugin: A computational system for dynamic time-sliced Bayesian networks. *International journal of forecasting*, 11:89–111, 1995.
- [KN01] G. Kokolakis and P. H. Nanopoulos. Bayesian multivariate micro-aggregation under the Hellinger’s distance criterion. *Research in official statistics*, 4(1):117–126, 2001.
- [Lau95] S. L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational statistics and data analysis*, 19:191–201, 1995.
- [LdBSH00] P. J. F. Lucas, N. de Brujin, K. Schurink, and A. Hoepelman. A probabilistic and decision-theoretic approach to the management of infectious disease at the ICU. *Artificial intelligence in medicine*, 19(3):251–279, 2000.
- [LPKB00] U. Lerner, R. Parr, D. Koller, and G. Biswas. Bayesian fault detection and diagnosis in dynamic systems. In *AAAI/IAAI*, pages 531–537, Austin, TX, USA, 2000.
- [LS88] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of the royal statistical society, series B (methodological)*, 50(2):157–224, 1988.
- [LvdGAH04] P. J. F. Lucas, L. C. van der Gaag, and A. Abu-Hanna. Bayesian networks in biomedicine and health-care. *Artificial intelligence in medicine*, 30:201–214, 2004.
- [MU05] F. Morchen and A. Ultsch. Optimizing time-series discretization for knowledge discovery. In *The 11th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 660–665, Chicago, IL, USA, August 2005.
- [Mur01] K. P. Murphy. The Bayes net toolbox for Matlab. *Computing science and statistics*, 33:331–350, Oktober 2001.
- [Mur02] K. P. Murphy. *Dynamic Bayesian networks: representation, inference and learning*. PhD thesis, Computer science division, University of California, Berkeley, CA, USA, Fall 2002.
- [MW01] K. P. Murphy and Y. Weiss. The factored frontier algorithm for approximate inference in DBNs. In *Proceedings of the 17th conference on uncertainty in artificial intelligence (UAI-01)*, pages 378–385, University of Washington, Seattle, WA, USA, August 2001.

- [Nik98] D. Nikovski. Learning stationary temporal probabilistic networks. In *Conference on automated learning and discovery*, Pittsburgh, PA, USA, June 1998. Carnegie Mellon University.
- [Nor97] Norsys Software Corporation. *Netica - Application for belief networks and influence diagrams - User guide*, March 1997.
- [Pea88] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann publishers, San Mateo, CA, USA, 1988.
- [Rab89] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77 issue 2, pages 257–286, February 1989.
- [RB03] M. J. Russell and J. A. Bilmes. Introduction to the special issue on new computational paradigms for acoustic modeling in speech recognition. *Computer speech and language*, 17:107–112, April 2003.
- [RN03] S. J. Russell and P. Norvig. *Artificial intelligence: A modern approach - Second edition*. Prentice-Hall international, international edition, 2003.
- [Sha98] R. Shachter. Bayes-Ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *Proceedings of the 14th conference on uncertainty in artificial intelligence (UAI-98)*, pages 480–487, San Francisco, CA, USA, 1998. Morgan Kaufmann publishers.
- [SS88] G. Shafer and P. Shenoy. Local computation in hypertrees. Business school, University of Kansas. Working paper No. 201, 1988.
- [TVLGH05] A. Tucker, V. Vinciotti, X. Liu, and D. Garway-Heath. A spatio-temporal Bayesian network classifier for understanding visual field deterioration. *Artificial intelligence in medicine*, 34:163–177, 2005.
- [Urs98] M. Ursino. Interaction between carotid baroregulation and the pulsating heart: a mathematical model. *American physiological society*, 275(5):H1733–H1747, November 1998.
- [WB04] G. Welch and G. Bishop. An introduction to the Kalman filter. Technical Report 95-041, University of North Carolina at Chapel Hill, Department of computer science, Chapel Hill, NC, USA, April 2004.
- [YD03] C. Yuan and M. J. Druzdzel. An importance sampling algorithm based on evidence pre-propagation. In *Proceedings of the 19th conference on uncertainty in artificial intelligence (UAI-03)*, pages 624–631, San Francisco, CA, USA, 2003. Morgan Kaufmann publishers.
- [ZD06] A. Zagorecki and M. J. Druzdzel. Knowledge engineering for Bayesian networks: How common are Noisy-MAX distributions in practice? In *The 17th European conference on artificial intelligence*, pages 361–365, Aug 2006.
- [Zha96] N. L. Zhang. Irrelevance and parameter learning in Bayesian networks. *Artificial intelligence journal*, 88:359–373, 1996.
- [ZP00] G. G. Zweig and M. Padmanabhan. Exact alpha-beta computation in logarithmic space with application to MAP word graph construction. In *Proceedings of the 6th international conference on spoken language processing (ICSLP-00)*, volume 2, pages 855–858, Beijing, China, October 2000.

- [ZVD06] A. Zagorecki, M. Voortman, and M. J. Druzdzel. Decomposing local probability distributions in Bayesian networks for improved inference and parameter learning. In *The 19th international Florida artificial intelligence research society conference (FLAIRS)*, Melbourne Beach, FL, USA, May 2006.
- [Zwe96] G. G. Zweig. A forward-backward algorithm for inference in Bayesian networks and an empirical comparison with HMMs. Master's thesis, Department of computer science, University of California, Berkeley, CA, USA, 1996.
- [Zwe98] G. G. Zweig. *Speech recognition with dynamic Bayesian networks*. PhD thesis, University of California, Berkeley, CA, USA, Spring 1998.

Part V

Appendices

*"Begin at the beginning
and go on till you come
to the end: then stop."
- The King,
Alice in Wonderland.*



Appendix

A

Temporal reasoning in SMILE tutorial and API

A.1 Introduction

A BN is useful for problem domains where the state of the world is static. In such a world, every variable has a single and fixed value. Unfortunately, this assumption of a static world does not always hold, as many domains exist where variables are dynamic and reasoning over time is necessary, such as dynamic systems. A *dynamic Bayesian network* (DBN) is a BN extended with a temporal dimension to enable us to model dynamic systems [DK88]. This tutorial assumes a basic knowledge about the DBN formalism. If this is not the case, the reader is directed to: [Jor03, KF05, Mur02].

A.2 Tutorial

For this tutorial, we have extended the umbrella DBN that also serves as an illustrative example in [RN03] with a random variable that models the area where the secret underground installation is based:

Example (Extended umbrella network) Suppose you are a security guard at some secret underground installation. You have a shift of seven days and you want to know whether it is raining on the day of your return to the outside world. Your only access to the outside world occurs each morning when you see the director coming in, with or without, an umbrella. Furthermore, you know that the government has two secret underground installations: one in Pittsburgh and one in the Sahara, but you do not know which one you are guarding.

For each day t , the set of evidence contains a single variable $Umbrella_t$ (observed an umbrella) and the set of unobservable variables contains $Rain_t$ (whether it is raining) and $Area$ (Pittsburgh or Sahara). If it is raining today depends on if it rained the day before and the geographical location. The resulting DBN and CPTs are shown in figure A.1.

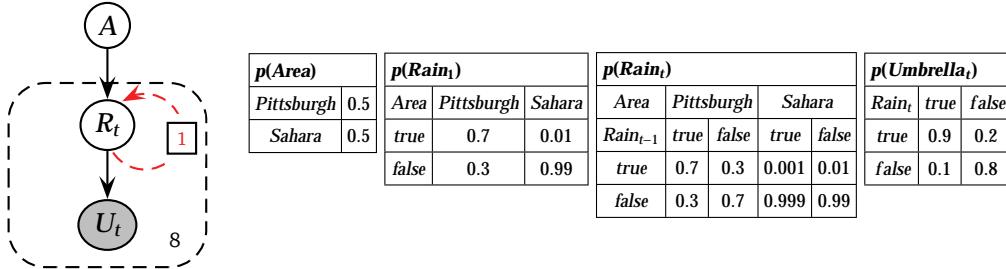


Figure A.1: The extended umbrella DBN and its CPTs.

Before we can start with the tutorial, we need to construct the default program presented in listing A.1. The methods `CreateDBN()`, `InferenceDBN()`, and `UnrollDBN()` will be implemented during the tutorial.

```

1 // Default program.
2 #include "smile.h"
3
4 using namespace std;
5
6 void CreateDBN(void);
7 void InferenceDBN(void);
8 void UnrollDBN(void);
9
10 int main(){
11     EnableXdsfFormat();
12     CreateDBN();
13     InferenceDBN();
14     UnrollDBN();
15     return DSL_OKAY;
16 }
```

Listing A.1: Main program

A.2.1 Creating the DBN

Creating a DBN is essentially the same as creating a BN. First, we initialize the network, the nodes and their states (listing A.2).

```

1 void CreateDBN(void){
2     // Initialize network and nodes.
3     DSL_network theDBN;
4     int rain      = theDBN.AddNode(DSL_CPT, "Rain");
5     int umbrella = theDBN.AddNode(DSL_CPT, "Umbrella");
6     int area      = theDBN.AddNode(DSL_CPT, "Area");
7
8     // Create and add statenames to [rain], [umbrella].
9     DSL_idArray stateNames;
10    stateNames.Add("True");
11    stateNames.Add("False");
12    theDBN.GetNode(rain)->Definition()->SetNumberOfOutcomes(stateNames);
13    theDBN.GetNode(umbrella)->Definition()->SetNumberOfOutcomes(stateNames);
14
15    // Create and add statenames to [area].
```

```

16     stateNames.CleanUp();
17     stateNames.Add("Pittsburgh");
18     stateNames.Add("Sahara");
19     theDBN.GetNode(area)->Definition()->SetNumberOfOutcomes(stateNames);
20

```

Listing A.2: Creating the DBN - Initialize

After that, we set the non-temporal probabilities and arcs, just like we would do with a plain BN (listing A.3).

```

21 // Add non-temporal arcs.
22 theDBN.AddArc(area, rain);
23 theDBN.AddArc(rain, umbrella);
24
25 // Add non-temporal probabilities to nodes.
26 DSL_doubleArray theProbs;
27
28 // Add probabilities to [area].
29 theProbs.SetSize(2);
30 theProbs[0] = 0.5;
31 theProbs[1] = 0.5;
32 theDBN.GetNode(area)->Definition()->SetDefinition(theProbs);
33
34 // Add probabilities to the initial CPT of [rain].
35 theProbs.SetSize(4);
36 theProbs[0] = 0.7;
37 theProbs[1] = 0.3;
38 theProbs[2] = 0.01;
39 theProbs[3] = 0.99;
40 theDBN.GetNode(rain)->Definition()->SetDefinition(theProbs);
41
42 // Add probabilities to [umbrella].
43 theProbs[0] = 0.9;
44 theProbs[1] = 0.1;
45 theProbs[2] = 0.2;
46 theProbs[3] = 0.8;
47 theDBN.GetNode(umbrella)->Definition()->SetDefinition(theProbs);
48

```

Listing A.3: Creating the DBN - Non-temporal

The above code should look familiar, as it just defines a standard BN. Next, we are going to add temporal dependencies. First, the temporal types of the variables need to be set. The temporal type is `dsl_normalNode` by default, but it is only possible to add temporal arcs between nodes in the plate. Listing A.4 shows the code that moves the nodes `Rain` and `Umbrella` to the plate.

```

49 // Set temporal types.
50 theDBN.SetTemporalType(umbrella, dsl_plateNode);
51 theDBN.SetTemporalType(rain, dsl_plateNode);
52

```

Listing A.4: Creating the DBN - Temporal type

We add a temporal arc from $Rain_t$ to $Rain_{t+1}$ by calling the method `AddTemporalArc(int parent, int child, int order)` (listing A.5).

```

53 // Add temporal arc.
54 theDBN.AddTemporalArc(rain, rain, 1);
55

```

Listing A.5: Creating the DBN - Temporal arc

Finally, the parameters of the temporal CPT are set and the DBN is saved to a file (listing A.6). We now have obtained the Umbrella DBN that is going to help us solve the problem of the guard in the secret underground base!

```

56 // Add temporal probabilities to the first-order CPT of [rain].
57 theProbs.SetSize(8);
58 theProbs[0] = 0.7;
59 theProbs[1] = 0.3;
60 theProbs[2] = 0.3;
61 theProbs[3] = 0.7;
62 theProbs[4] = 0.001;
63 theProbs[5] = 0.999;
64 theProbs[6] = 0.01;
65 theProbs[7] = 0.99;
66 theDBN.GetNode(rain)->TemporalDefinition()->SetTemporalProbabilities(theProbs, 1);
67
68 // Write the DBN to a file.
69 theDBN.WriteFile("dbn.xdsl");
70 }
```

Listing A.6: Creating the DBN - Temporal CPT

A.2.2 Performing inference

Performing inference on a DBN is very easy. Just like performing inference on a BN, it is a matter of setting evidence, calling `UpdateBeliefs()`, and obtaining the updated values. This tutorial assumes that the Umbrella DBN is saved to `dbn.xdsl`. Before we can add evidence, the network needs to be loaded and the nodes identified (listing A.7).

```

1 void InferenceDBN(void){
2   DSL_network theDBN;
3   theDBN.ReadFile("dbn.xdsl");
4
5   // Obtain the node handles.
6   int rain    = theDBN.FindNode("Rain");
7   int umbrella = theDBN.FindNode("Umbrella");
8   int area    = theDBN.FindNode("Area");
9 }
```

Listing A.7: Inference - Load network

Because we want to predict the weather conditions on the eight day, the DBN needs to be unrolled for eight time-slices. This number is set through the `SetNumberOfSlices(int slices)` method (listing A.8).

```

10 // Perform inference over a period of 8 days.
11 theDBN.SetNumberOfSlices(8);
12
```

Listing A.8: Inference - Set time-slices

During his shift, the guard only observed an umbrella on the fourth day. In a DBN, evidence is set by calling the `SetTemporalEvidence(int state, int slice)` method. In this case, our evidence is the following: $\mathbf{u}_{0:6} = \{\text{false}, \text{false}, \text{false}, \text{true}, \text{false}, \text{false}\}$ (listing A.9).

```

13 // Set the evidence of the DBN.
14 theDBN.GetNode(umbrella)->TemporalValue()->SetTemporalEvidence(1, 0);
15 theDBN.GetNode(umbrella)->TemporalValue()->SetTemporalEvidence(1, 1);
16 theDBN.GetNode(umbrella)->TemporalValue()->SetTemporalEvidence(1, 2);
17 theDBN.GetNode(umbrella)->TemporalValue()->SetTemporalEvidence(0, 3);
```

```

18     theDBN.GetNode(umbrella)->TemporalValue()->SetTemporalEvidence(1, 4);
19     theDBN.GetNode(umbrella)->TemporalValue()->SetTemporalEvidence(1, 5);
20     theDBN.GetNode(umbrella)->TemporalValue()->SetTemporalEvidence(1, 6);
21

```

Listing A.9: Inference - Set evidence

Inference is performed by calling the method `UpdateBeliefs()`, after which the updated temporal values can be retrieved by calling the method `GetTemporalValue(DSL_Dmatrix &here, int order)` (listing A.10).

```

22 // Do inference.
23 theDBN.UpdateBeliefs();
24
25 // Get beliefs.
26 DSL_Dmatrix abeliefs;
27 DSL_Dmatrix rbeliefs;
28 abeliefs = *theDBN.GetNode(area)->Value()->GetMatrix();
29 theDBN.GetNode(rain)->TemporalValue()->GetTemporalValue(rbeliefs, 7);
30

```

Listing A.10: Inference - Update and retrieve values

Finally, the updated values for *Area* and *Rain* on the eight day are printed to the console (listing A.11). The output should be $\langle \text{Pittsburgh} = 0.088, \text{Sahara} = 0.912 \rangle$ for *A* and $\langle \text{True} = 0.038, \text{False} = 0.962 \rangle$ for R_8 .

```

31 // Print beliefs.
32 DSL_idArray *stateNames;
33
34 stateNames = theDBN.GetNode(area)->Definition()->GetOutcomesNames();
35 cout << "Beliefs_of_[area]" << endl;
36 cout << " " << (*stateNames)[0] << "\t" << abeliefs[0] << endl;
37 cout << " " << (*stateNames)[1] << "\t" << abeliefs[1] << endl;
38 cout << endl;
39
40 stateNames = theDBN.GetNode(rain)->Definition()->GetOutcomesNames();
41 cout << "Beliefs_of_[rain]_tomorrow" << endl;
42 cout << " " << (*stateNames)[0] << "\t\t" << rbeliefs[0] << endl;
43 cout << " " << (*stateNames)[1] << "\t\t" << rbeliefs[1] << endl;
44 cout << endl;
45 }

```

Listing A.11: Inference - Print beliefs

A.2.3 Unrolling

It can be useful to obtain an unrolled version of the DBN. The `UnrollNetwork(DSL_network &here)` method can be used for this purpose (listing A.12).

```

1 void UnrollDBN(void){
2     DSL_network theDBN;
3     theDBN.ReadFile("dbn.xdsl");
4
5     // Unroll DBN for a period of 8 days.
6     theDBN.SetNumberOfSlices(8);
7
8     // Save unrolled DBN to a file.
9     DSL_network unrolled;
10    theDBN.UnrollNetwork(unrolled);
11    unrolled.WriteFile("dbn_unrolled_8.xdsl");
12 }

```

Listing A.12: Unroll

A.3 Public API

A.3.1 DSL_network

The **DSL_network** class contains all the methods that are needed for the administration of nodes, normal arcs and temporal arcs, importing and exporting network definitions from/to files, unrolling the DBN, and performing inference. In the future, specialized inference methods can be added by extending the `UpdateBeliefs()` method, but currently the standard inference techniques of SMILE are used.

int AddTemporalArc(int parent, int child, int order) Adds a temporal arc from the parent to the child with given temporal order. For this method to succeed, the following conditions must hold: the parent and child need to exist as nodes in the plate; the given temporal order needs to be greater than 0; for a given temporal order greater than 1, a temporal arc with a temporal order of 1 must already exist; and the temporal arc must not exist already.

Error return values: `DSL_OUT_OF_RANGE` (if any of the above mentioned conditions is not true).

vector< pair<int, int> >& GetTemporalChildren(int node) Returns a vector of pairs of `<node handle, temporal order>` values that are temporal children of the node. If the node does not exist, or does not have temporal children, the vector is empty.

int GetMaximumOrder() Returns the maximum temporal order of the network.

int GetNumberOfSlices() Returns the number of time-slices of the plate.

vector< pair<int, int> >& GetTemporalParents(int node) Returns a vector of pairs of `<temporal order, node handle>` values that are temporal parents of the node. If the node does not exist, or does not have temporal parents, the vector is empty.

dsl_temporalType GetTemporalType(int node) Returns the temporal type of the node. Temporal type can be one of the following values of the enum `dsl_temporalType`:

- `dsl_anchorNode`,
- `dsl_normalNode`,
- `dsl_plateNode`,
- `dsl_terminalNode`.

int RemoveAllTemporalInfo() Removes all temporal information from the network: it sets the number of time-slices to 0; it deletes all temporal arcs; and it sets all temporal types to `dsl_normalNode`.

int RemoveTemporalArc(int parent, int child, int order) Removes the temporal arc from the parent to the child with the given order. For this method to succeed, the following conditions must hold: the parent and child need to exist; the given temporal order needs to be greater than 0; the temporal arc needs to exist; and the temporal arc may not be the last temporal arc with a temporal order of 1 if there are still temporal arcs with a temporal order greater than 1.

Error return values: `DSL_OUT_OF_RANGE` (if any of the above mentioned conditions is not true).

int SetNumberOfSlices(int slices) Sets the number of time-slices of the nodes in the plate. For this method to succeed, the following conditions must hold: the given number of time-slices must be not less than 0; and the maximum temporal order of the network must be greater than 1 if the given number of time-slices is greater than 0.

Error return values: DSL_OUT_OF_RANGE (if any of the above mentioned conditions is not true).

int SetTemporalType(int node, dsl_temporalType type) Sets the temporal type node. Temporal type can be one of the following values of the enum `dsl_temporalType`:

- `dsl_anchorNode`,
- `dsl_normalNode`,
- `dsl_plateNode`,
- `dsl_terminalNode`.

For this method to succeed, the following conditions must hold: the node must exist; and none of the checks in table A.1 fails.

Table A.1: Implemented checks for temporal type transitions. The conditions in the table block the transition.

FROM	TO			
	Anchor	Normal	Plate	Terminal
Anchor		No conditions.	Children out of plate that are not terminal nodes.	Children in the plate.
Normal	No conditions.		Children out of plate that are not terminal nodes.	Children in the plate.
Plate	Parents in the plate or temporal arcs.	Parents in the plate or temporal arcs.		Children in the plate or temporal arcs.
Terminal	Parents in the plate.	Parents in the plate.	No conditions ^a	

^a It is not allowed for terminal nodes to have static arcs to anchor nodes.

Error return values: DSL_OUT_OF_RANGE (if any of above mentioned the conditions is not true).

bool TemporalRelationExists(int parent, int child, int order) Returns true if the temporal arc from the parent node to the child node exists with this temporal order and false otherwise.

int UnrollNetwork(DSL_network &here) Unrolls the plate of the network to the number of time-slices defined by calling the `SetNumberOfSlices(int theSlices)` method and saves it to `DSL_network here`. For this method to succeed, the following conditions must hold: the maximum temporal order of the network must be greater than 1, the DBN definition must be consistent.

Error return values: DSL_OUT_OF_RANGE (if any of above mentioned the conditions is not true), DSL_OUT_OF_MEMORY.

A.3.2 DSL_node

The `DSL_node` class is extended with two methods that make it more convenient to obtain the temporal definitions and values of the node.

DSL_nodeDefTemporals* TemporalDefinition() Returns a pointer to the temporal definition of the node or NULL if the node does not have one. Currently, only nodes of type `DSL_CPT` are supported, the method will return NULL if this is not the case.

DSL_nodeValTemporals* TemporalValue() Returns a pointer to the temporal posteriors of the node or NULL if the node does not have one. Currently, only nodes of type DSL_CPT are supported, the method will return NULL if this is not the case. A user should always use `DSL_nodeValTemporals` when retrieving temporal posterior beliefs for the nodes in the plate.

A.3.3 DSL_nodeDefinition

The `DSL_nodeDefinition` class has a property `DSL_nodeDefTemporals` that is only instantiated if the node has temporal arcs. It is NULL otherwise. Currently, only node definitions of type `DSL_CPT` can have a temporal definition.

DSL_nodeDefTemporals* GetTemporals() Returns a pointer to the temporal definition of the node definition or NULL if the node definition does not have one. Currently, only node definitions of type `DSL_CPT` are supported, the method will return NULL if this is not the case.

A.3.4 DSL_nodeDefTemporals

The `DSL_nodeDefTemporals` class is responsible for the administration of the temporal probability distributions of a node. Currently, only `DSL_nodeDefTemporals` of type `DSL_CPT` are supported.

int GetMaximumOrder() Returns the maximum temporal order of this temporal definition.

int GetMaximumOrderTemporalMatrix(DSL_Dmatrix &here) Saves the temporal probability distribution with the maximum temporal order to `DSL_Dmatrix` `here`.

Error return values: `DSL_WRONG_NODE_TYPE` (if `DSL_nodeDefTemporals` is not of type `DSL_CPT`).

DSL_nodeDefinition* GetNodeDefinition() Returns the pointer of the `DSL_nodeDefinition` that this `DSL_nodeDefTemporals` belongs to.

int GetNodeHandle() Returns the handle of the node that this `DSL_nodeDefTemporals` belongs to.

int GetTemporalMatrix(DSL_Dmatrix &here, int order) Saves the temporal probability distribution with the maximum temporal order to `DSL_Dmatrix` `here`.

Error return values: `DSL_WRONG_NODE_TYPE` (if `DSL_nodeDefTemporals` is not of type `DSL_CPT`).

int GetType() Returns the type of the temporal definition.

Error return values: `DSL_WRONG_NODE_TYPE` (if `DSL_nodeDefTemporals` is not of type `DSL_CPT`).

int SetTemporalMatrix(DSL_Dmatrix &matrix, int order) Sets the temporal matrix with this temporal order. For this method to succeed, the given temporal order must be greater than 0.

Error return values: `DSL_WRONG_NODE_TYPE` (if `DSL_nodeDefTemporals` is not of type `DSL_CPT`), `DSL_OUT_OF_RANGE` (if temporal order smaller than 1).

int SetTemporalProbabilities(DSL_doubleArray &probs, int order) Set the probabilities of the temporal matrix with this temporal order. For this method to succeed, the following conditions must hold: the given temporal order must be greater than 0; the temporal matrix with this temporal order must exist; and the array of probabilities must have the correct size.

Error return values: DSL_WRONG_NODE_TYPE (if `DSL_nodeDefTemporals` is not of type `DSL_CPT`), `DSL_OUT_OF_RANGE` (if any of the above mentioned conditions is not true).

A.3.5 `DSL_nodeValue`

The `DSL_nodeValue` class has a property `DSL_nodeValTemporals` that is only instantiated for nodes in the plate if the number of time-slices is set to be greater than 0. It is `NULL` otherwise. Currently, only node values of type `DSL_BELIEFVECTOR` can have a temporal value.

DSL_nodeValTemporals* GetTemporals() Returns a pointer to the temporal value of the node value or `NULL` if the node value does not have one. Currently, only node values of type `DSL_BELIEFVECTOR` are supported, the method will return `NULL` if this is not the case.

A.3.6 `DSL_nodeValTemporals`

The `DSL_nodeValTemporals` class is responsible for the administration of the temporal posterior beliefs of a node. Currently, only `DSL_nodeValTemporals` of type `DSL_BELIEFVECTOR` are supported.

int ClearAllTemporalEvidence() Clears the observed states for all time-slices of this node.
Error return values: DSL_WRONG_NODE_TYPE (if `DSL_nodeValTemporals` is not of type `DSL_BELIEFVECTOR`).

int ClearTemporalEvidence(int slice) Clears the observed state for the given time-slice of this node. Time-slices range from 0 to (number of slices - 1). Note that this does not guarantee that after this call this time-slice will not be evidence anymore, as evidence can be propagated from other nodes into this one (there must be some deterministic relationship between nodes for this to happen). For this method to succeed, the following conditions must hold: the time-slice must be valid; and the evidence must exist.

Error return values: DSL_WRONG_NODE_TYPE (if `DSL_nodeValTemporals` is not of type `DSL_BELIEFVECTOR`), `DSL_OUT_OF_RANGE` (if any of the above mentioned conditions is not true).

int GetNodeHandle() Returns the handle of the node that this `DSL_nodeValTemporals` belongs to.

DSL_nodeValue* GetnodeValue() Returns the pointer of the `DSL_nodeValue` that this `DSL_nodeValTemporals` belongs to.

int GetTemporalEvidence(int slice) Returns the observed state for the given time-slice of this node. Time-slices range from 0 to (number of slices - 1). The observation may come directly from the user or can be deduced from other observations in other nodes of the network. For this method to succeed, the following conditions must hold: the state must be observed; and the time-slice must be valid.

Error return values: DSL_WRONG_NODE_TYPE (if `DSL_nodeValTemporals` is not of type `DSL_BELIEFVECTOR`), `DSL_OUT_OF_RANGE` (if any of the above mentioned conditions is not true).

int GetTemporalValue(DSL_Dmatrix &here, int slice) Saves the temporal posterior beliefs for the given time-slice of this node to `DSL_Dmatrix here`. Time-slices range from 0 to (number of slices - 1). A user should always use `DSL_nodeValTemporals` when retrieving temporal posterior beliefs for the nodes in the plate.

Error return values: `DSL_WRONG_NODE_TYPE` (if `DSL_nodeValTemporals` is not of type `DSL_BELIEFVECTOR`), `DSL_OUT_OF_RANGE` (if the time-slice is not valid).

int GetType() Returns the type of the temporal value.

Error return values: `DSL_WRONG_NODE_TYPE` (if `DSL_nodeValTemporals` is not of type `DSL_BELIEFVECTOR`).

int IsTemporalEvidence(int slice) Returns a non-zero value if the state for the given time-slice of the node has been directly observed or has been deduced from the evidence in other nodes.

Error return values: `DSL_OUT_OF_RANGE` (if the given time-slice is not valid).

int IsTemporalValueValid(int slice) Returns a non-zero value if the value for this time-slice of the node is valid.

Error return values: `DSL_OUT_OF_RANGE` (if the given time-slice is not valid).

int SetTemporalEvidence(int evidence, int slice) Sets the observed state for the given time-slice of this node to the specified evidence. Time-slices range from 0 to (number of slices - 1). For this method to succeed, the following conditions must hold: the state must be valid; and the time-slice must be valid.

Error return values: `DSL_WRONG_NODE_TYPE` (if `DSL_nodeValTemporals` is not of type `DSL_BELIEFVECTOR`), `DSL_OUT_OF_RANGE` (if any of the above mentioned conditions is not true).

int SetTemporalValue(DSL_Dmatrix &matrix, int slice) Sets the posterior beliefs for the given time-slice of this node to the specified beliefs. Be careful with this function, it does not check if the posterior beliefs are still consistent.

Error return values: `DSL_WRONG_NODE_TYPE` (if `DSL_nodeValTemporals` is not of type `DSL_BELIEFVECTOR`), `DSL_OUT_OF_RANGE` (if the given time-slice is not valid).



Appendix B

Temporal reasoning in GeNIE tutorial

A BN is useful for problem domains where the state of the world is static. In such a world, every variable has a single and fixed value. Unfortunately, this assumption of a static world does not always hold, as many domains exist where variables are dynamic and reasoning over time is necessary, such as dynamic systems. A *dynamic Bayesian network* (DBN) is a BN extended with a temporal dimension to enable us to model dynamic systems [DK88]. This tutorial assumes a basic knowledge about the DBN formalism and GeNIE. If this is not the case, the reader is directed to: [Jor03, KF05, Mur02] and to previous GeNIE tutorials.

For this tutorial, we have extended the umbrella DBN that also serves as an illustrative example in [RN03] with a random variable that models the area where the secret underground installation is based:

Example (Extended umbrella network) Suppose you are a security guard at some secret underground installation. You have a shift of seven days and you want to know whether it is raining on the day of your return to the outside world. Your only access to the outside world occurs each morning when you see the director coming in, with or without, an umbrella. Furthermore, you know that the government has two secret underground installations: one in Pittsburgh and one in the Sahara, but you do not know which one you are guarding.

For each day t , the set of evidence contains a single variable Umbrella_t (observed an umbrella) and the set of unobservable variables contains Rain_t (whether it is raining) and Area (Pittsburgh or Sahara). If it is raining today depends on if it rained the day before and the geographical location. The resulting DBN and CPTs are shown in figure B.1.

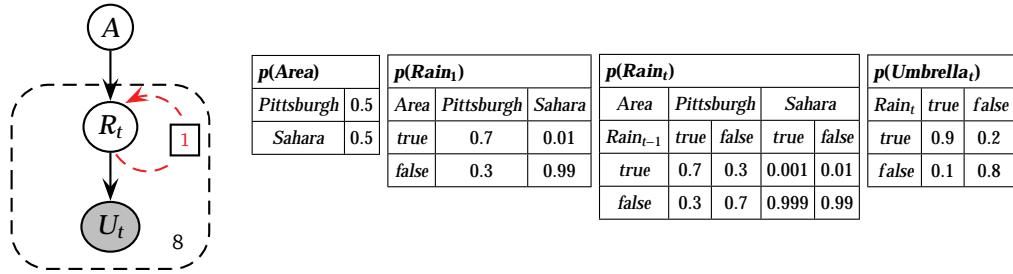


Figure B.1: The extended umbrella DBN and its CPTs.

B.1 Creating the DBN

Creating a DBN is essentially the same as creating a BN. First, we create a static network containing three nodes: *Area*, *Rain*, and *Umbrella* and two arcs going from *Area* to *Rain* and from *Rain* to *Umbrella*, just as we would do normally. The result of this first step is shown in figure B.2.

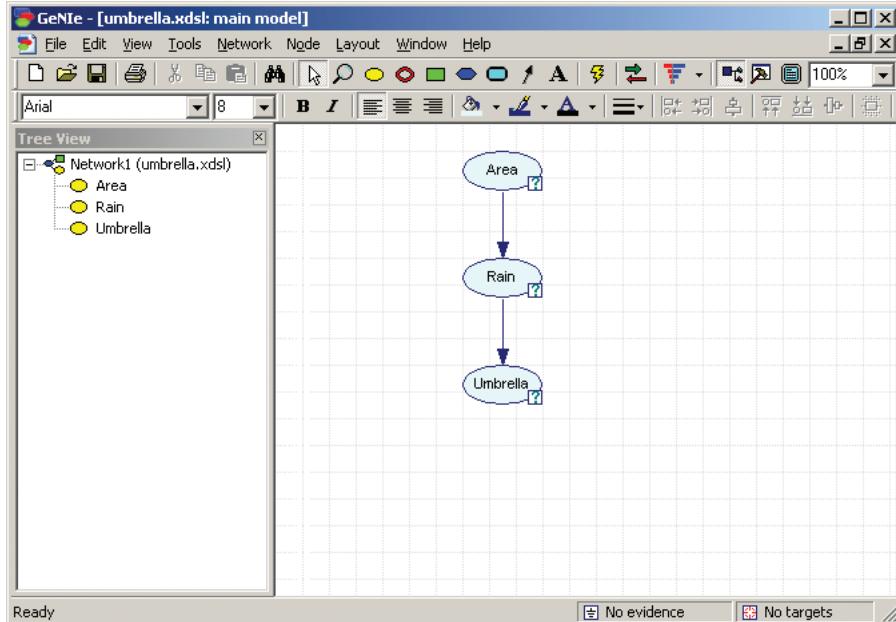


Figure B.2: Creating the static network in GeNIe.

Activating the plate Because we are dealing with a temporal network, we need to add a temporal arc from Rain_t to Rain_{t+1} . Temporal arcs can only be added to nodes in the temporal plate. The temporal plate is the part of the temporal network that contains the temporal nodes, which are *Rain* and *Umbrella* in this example. Before we can add nodes to the plate, we need to activate it first. This can be done by clicking on *Network* → *Enable Temporal Plate* in the menu bar as shown in figure B.3. This results in the network area being divided into four parts:

1. **Contemporals** This is the part of the network area where the static nodes are stored by default.
2. **Initial conditions** This is the part of the network area where the anchor nodes are stored.

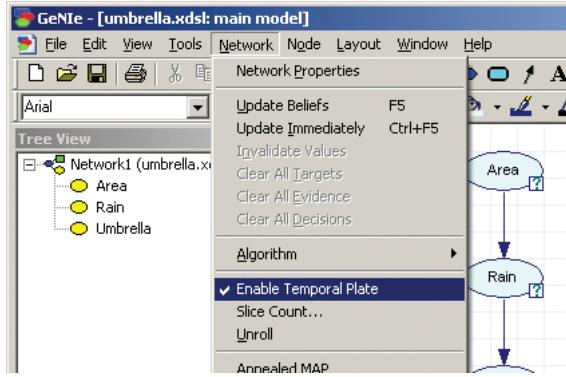


Figure B.3: Temporal options in the menu bar of GeNIe under Network.

3. **Temporal plate** This is the part of the network area where the nodes in the plate are stored. Nodes in the plate are the only nodes that are allowed to have temporal arcs. This area also shows the number of time-slices for which inference is performed.
4. **Terminal conditions** This is the part of the network area where the terminal nodes are stored.

The boundaries of the different areas can be changed by dragging them. These boundaries are also used to layout the network when it is unrolled explicitly. Temporal types of nodes can be changed by dragging them to one of the three other areas. After enabling the temporal plate, we drag first the *Umbrella* node and second the *Rain* node (in this order!) to the temporal plate. This results in the network shown in figure B.4.

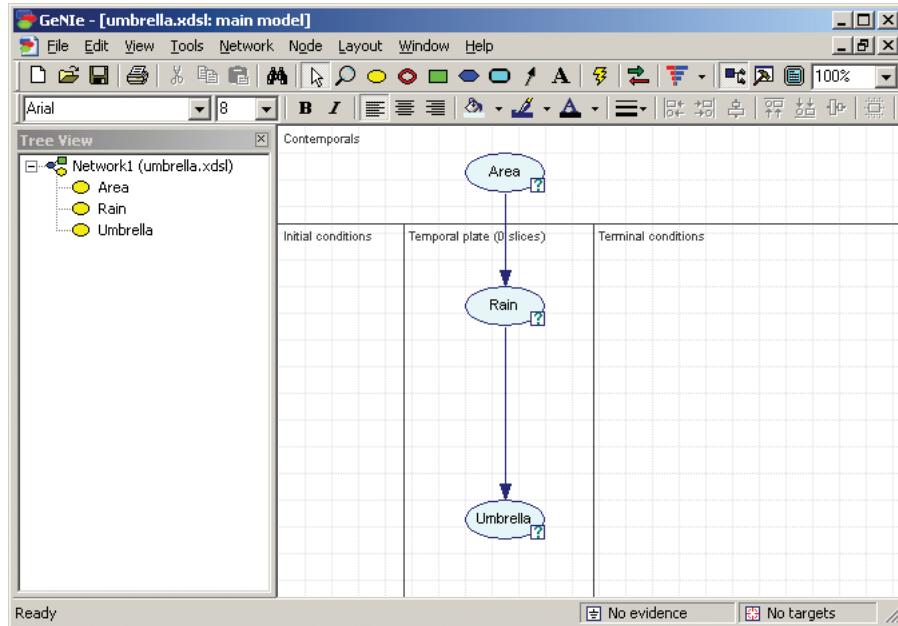


Figure B.4: The network area of GeNIe with the temporal plate enabled

Adding the temporal arc We now have a network with one contemporaneous node and two temporal nodes. The temporal arc can be added by clicking on the arc button and drawing an arc from the parent (*Rain*) to the child (*Rain*). When the mouse button is released, a context menu

appears to enable us to set the temporal order of the arc. We create a temporal arc with order 1. Figure B.5 demonstrates the addition of a temporal arc to a network.

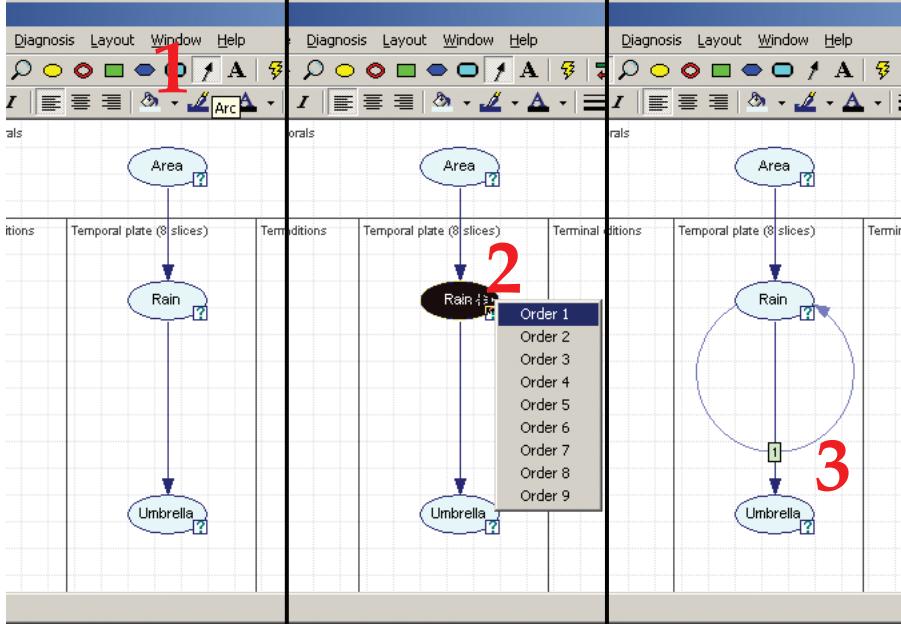


Figure B.5: Adding a temporal arc in GeNIE.

Setting the temporal probabilities The next step is to add the static and temporal probabilities from the CPTs in figure B.1. How to set the static probabilities should be known by now, but setting the temporal probabilities is something new. In a temporal network, every node in the plate needs a CPT for every incoming temporal arc with a different temporal order. Just like setting the CPT for static nodes, the CPTs for temporal nodes can be set by double-clicking on the node in the network area and go to the definition tab. When a node has incoming temporal arcs, the appropriate temporal CPT can be selected from a list and the CPT parameters can be set. Figure B.6 demonstrates the process of adding the temporal probabilities from figure B.1.

Now our first temporal network is finished! We can save it to a file and load it again as many times as we want, but more interesting is to see how we can reason with this network.

B.2 Performing inference

Performing inference on a DBN is very easy. Just like performing inference on a BN, it is just a matter of setting evidence and updating the posterior beliefs by right-clicking the network area, clicking *Network* → *Update Beliefs*, or pressing F5. However, with temporal networks, some extra steps need to be taken. Imagine the following situation:

During his shift, the guard only observed an umbrella on the fourth day. This means that the evidence vector for the *Umbrella* node is the following: $\text{Umbrella}_{0:6} = \{\text{false}, \text{false}, \text{false}, \text{true}, \text{false}, \text{false}\}$.

Setting temporal evidence We have collected evidence for the temporal network and now we want to add it. Before the addition of evidence, the number of time-slices of the temporal network needs to be set by clicking on *Network* → *Slice Count*.... The number of time-slices denote the time-period of interest, in our case we set it to 8.

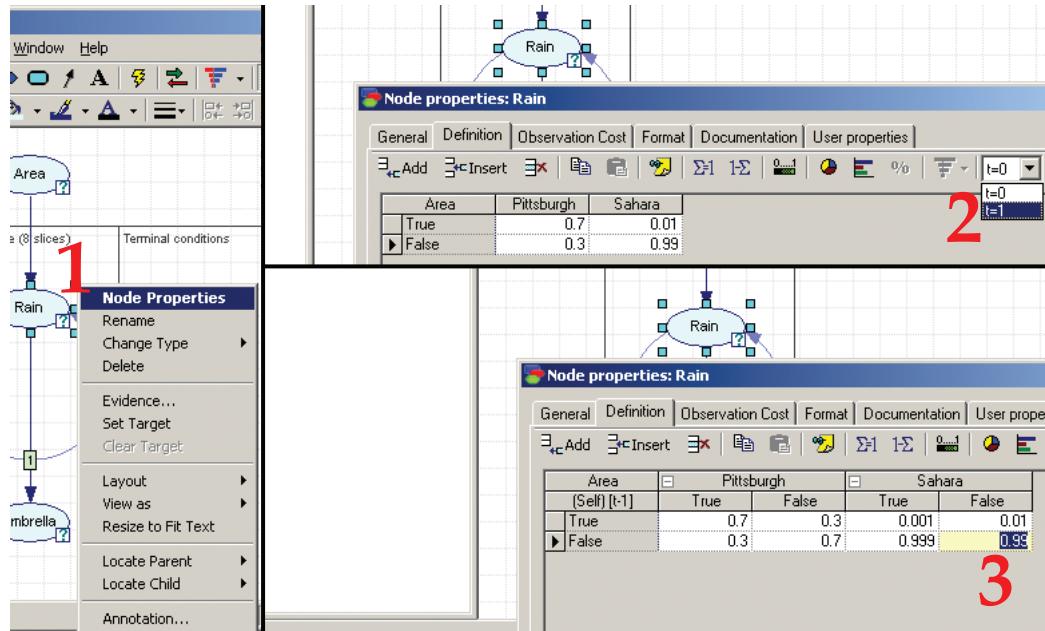


Figure B.6: Setting the temporal CPTs of a node in GeNIe.

After setting the number of time-slices, we can add the evidence by right-clicking *Umbrella* and selecting *Evidence* from the context menu. Because this is a temporal node, a form appears where evidence can be added for every time-slice. Figure B.7 demonstrates the addition of evidence (including the step where we set the number of time-slices).

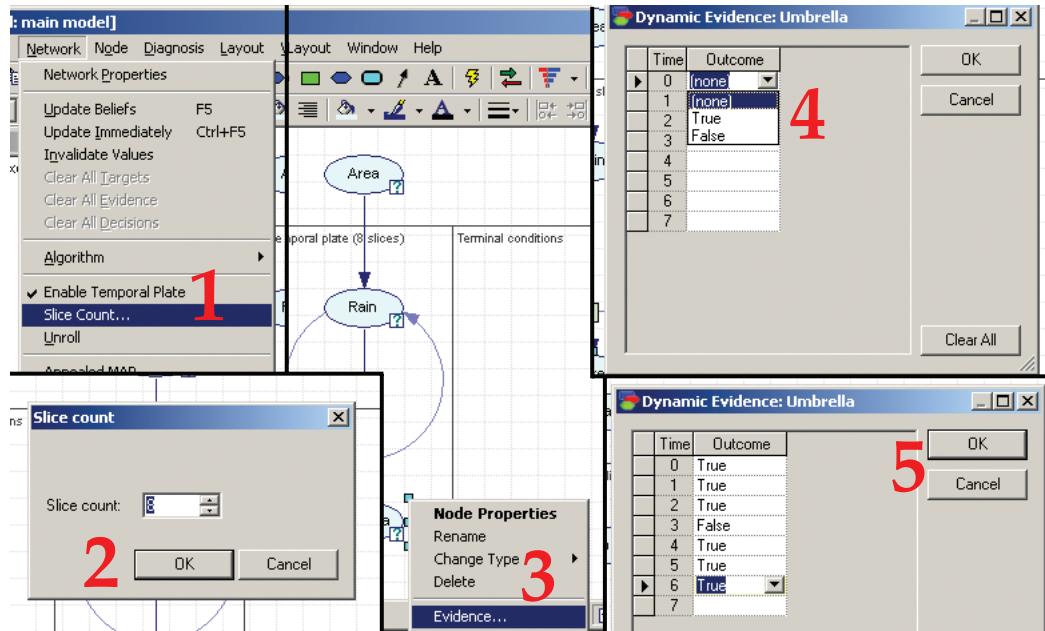


Figure B.7: Setting temporal evidence in GeNIe.

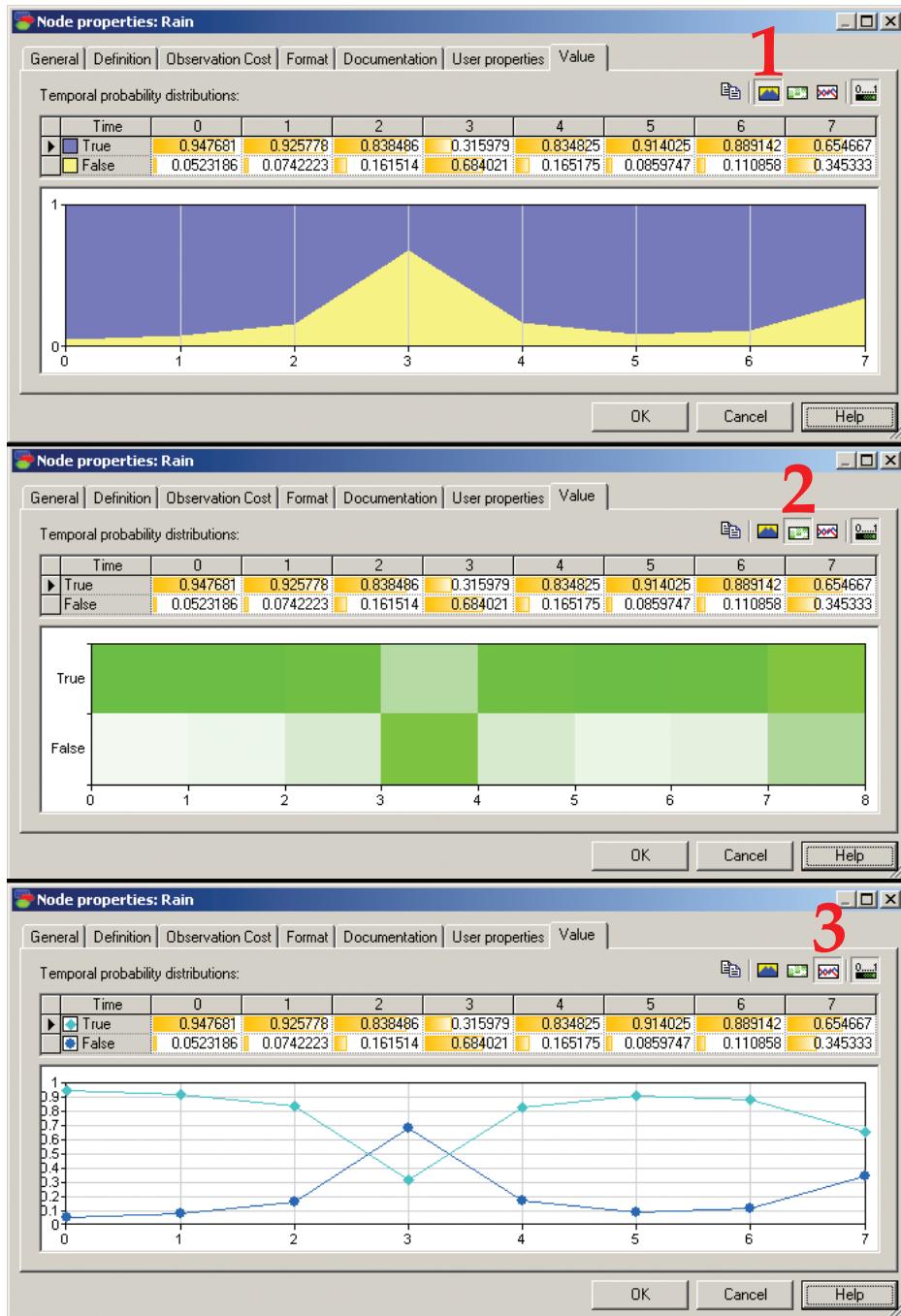


Figure B.8: Showing the different representations of the temporal posterior beliefs in GeNIE.

Temporal posterior beliefs After we call inference, the temporal network has its beliefs updated. The updated beliefs for a temporal node can be obtained by double-clicking on the node and selecting the *Value* tab that appears after inference. This tab contains the updated beliefs for all time-slices of that node not only in floating point numbers, but also as: an area chart, contour plot, and time-series plot. The area and time-series plots speak for themselves. The contour plot represents a three dimensional space with the dimensions: time (x-axis), state (y-axis), and probability of an outcome at every point in time (the color). This representation is

especially useful for temporal nodes that have many states, where the other two representations are more useful for temporal nodes with not too many states. Figure B.8 shows the different representations of the temporal posterior beliefs for the *Rain* node after setting the temporal evidence in *Umbrella* and calling inference.

B.3 Unrolling

It can be useful for debugging purposes to explicitly unroll a temporal network for a given number of time-slices. GeNIE provides this possibility through the *Network* → *Unroll* option. When clicking this option, GeNIE opens a new network that has the temporal network unrolled for the given number of time-slices. It is possible to locate a node in the temporal network from the unrolled network by right-clicking on the node in the unrolled network and select *Locate Original in DBN* from the context-menu. The unrolled network that is a result from unrolling the temporal network is cleared from any temporal information whatsoever. It can be edited, saved and restored just like any other static network. Figure B.9 shows the unrolled representation of a temporal network.

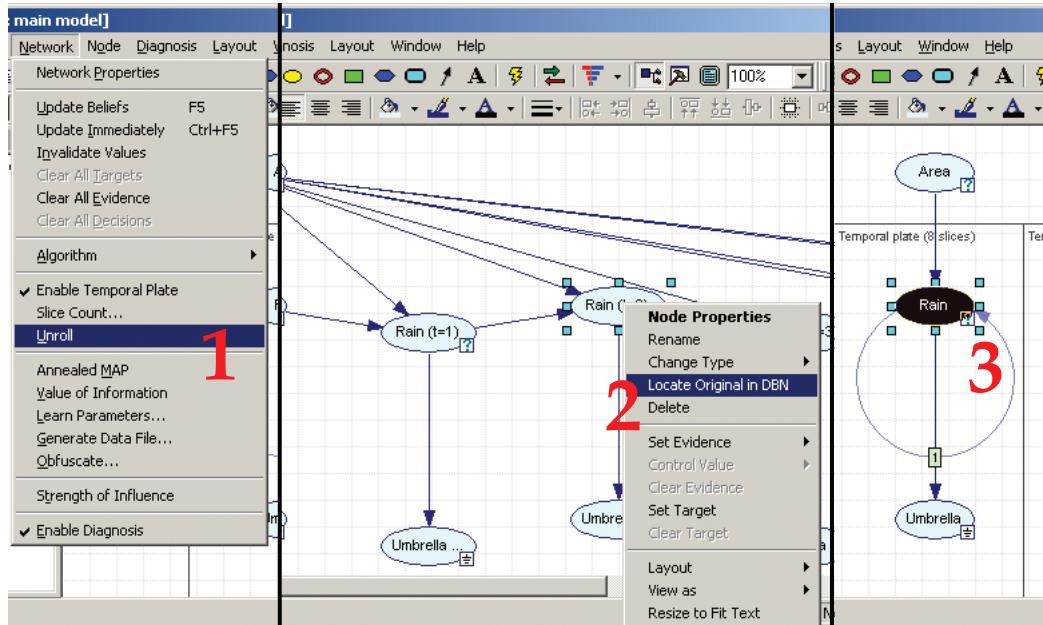


Figure B.9: Unrolling a temporal network and locating a node in GeNIE.

Appendix C

An example of defining and learning the cardiovascular system DBN

This appendix contains a snippet of the dataset in .csv format and the C++ program that was written to define the structure and learn the parameters of the cardiovascular system DBN using the SMILE API. It is included to provide a reference example for using the SMILE API.

C.1 Snippet of dataset

```
1 CardOut_up,EF_mean,EmaxLV_mean,HR_mean,Plv_up,Psa_down,Psa_up,Rep_mean,Rsp_mean,Shock
,VLV_down,VLV_up,CardOut_up_1,EF_mean_1,EmaxLV_mean_1,HR_mean_1,Plv_up_1,
Psa_down_1,Psa_up_1,Rep_mean_1,Rsp_mean_1,Shock_1,VLV_down_1,VLV_up_1
2 s03_630p00_665p00,s02_0p56_0p58,s01_2p84_3p02,s03_74p00_76p00,s02_130p00_134p00,
s01_90p50_93p00,s02_130p00_134p00,s09_below_1p35,s01_2p88_3p18,false,
s07_58p50_63p00,s08_130p00_134p00,s03_630p00_665p00,s02_0p56_0p58,s01_2p84_3p02,
s02_76p00_78p00,s02_130p00_134p00,s01_90p50_93p00,s02_130p00_134p00,
s09_below_1p35,s01_2p88_3p18,false,s08_54p00_58p50,s07_134p00_138p00
3 s03_630p00_665p00,s02_0p56_0p58,s01_2p84_3p02,s02_76p00_78p00,s02_130p00_134p00,
s01_90p50_93p00,s02_130p00_134p00,s09_below_1p35,s01_2p88_3p18,false,
s08_54p00_58p50,s07_134p00_138p00,s03_630p00_665p00,s02_0p56_0p58,s01_2p84_3p02,
s02_76p00_78p00,s01_134p00_138p00,s01_90p50_93p00,s01_134p00_138p00,s08_1p35_1p38
,s01_2p88_3p18,false,s07_58p50_63p00,s07_134p00_138p00
4 s03_630p00_665p00,s02_0p56_0p58,s01_2p84_3p02,s02_76p00_78p00,s01_134p00_138p00,
s01_90p50_93p00,s01_134p00_138p00,s08_1p35_1p38,s01_2p88_3p18,false,
s07_58p50_63p00,s07_134p00_138p00,s03_630p00_665p00,s02_0p56_0p58,s01_2p84_3p02,
s02_76p00_78p00,s02_130p00_134p00,s01_90p50_93p00,s02_130p00_134p00,s08_1p35_1p38
,s01_2p88_3p18,false,s07_58p50_63p00,s08_130p00_134p00
```

Listing C.1: cvdata.csv

C.2 Cardiovascular system DBN definition and learning tool

```

1 #include <string>
2 #include "smile.h"
3 #include "smilearn.h"
4
5 using namespace std;
6
7 class CardioVas{
8     public:
9         CardioVas(void){};
10        ~CardioVas(void){};
11
12        void CopyCPT(int from, int to, int order);
13        void CopyDBNParameters(void);
14        void CreateDBNStructure(void);
15        void CreateTransitionNetStructure(void);
16        void LearnTransitionNet(void);
17        void LoadDataFile(string datafile);
18        void Run(string datafile);
19        void SaveDBN(string dynNetStr);
20
21        DSL_network dynNet;
22        DSL_network tNet;
23        DSL_dataset dataset;
24    };

```

Listing C.2: cvdbn.h

```

1 #include "cardiovash.h"
2 #include <iostream>
3
4 //-----
5
6 void CardioVas::CopyCPT(int from, int to, int order){
7     DSL_Dmatrix probs;
8     probs = *(tNet.GetNode(from)->Definition()->GetMatrix());
9     if (order == 0){
10         dynNet.GetNode(to)->Definition()->SetDefinition(probs);
11     } else {
12         dynNet.GetNode(to)->TemporalDefinition()->SetTemporalMatrix(probs, order);
13     }
14 }
15
16 //-----
17
18 void CardioVas::CopyDBNParameters(void){
19     // Get the nodes from the dbn.
20     int CardOut_up_t = dynNet.FindNode("CardOut_up");
21     int EF_mean_t = dynNet.FindNode("EF_mean");
22     int EmaxLV_mean_t = dynNet.FindNode("EmaxLV_mean");
23     int HR_mean_t = dynNet.FindNode("HR_mean");
24     int Plv_up_t = dynNet.FindNode("Plv_up");
25     int Psa_down_t = dynNet.FindNode("Psa_down");
26     int Psa_up_t = dynNet.FindNode("Psa_up");
27     int Rep_mean_t = dynNet.FindNode("Rep_mean");
28     int Rsp_mean_t = dynNet.FindNode("Rsp_mean");
29     int Shock_t = dynNet.FindNode("Shock");
30     int Vlv_down_t = dynNet.FindNode("Vlv_down");
31     int Vlv_up_t = dynNet.FindNode("Vlv_up");
32
33     // Get the nodes from the trained network - Order 0.
34     int CardOut_up_tNet_0 = tNet.FindNode("CardOut_up");
35     int EF_mean_tNet_0 = tNet.FindNode("EF_mean");

```

```

36     int EmaxLV_mean_tNet_0 = tNet.FindNode("EmaxLV_mean");
37     int HR_mean_tNet_0      = tNet.FindNode("HR_mean");
38     int Plv_up_tNet_0       = tNet.FindNode("Plv_up");
39     int Psa_down_tNet_0     = tNet.FindNode("Psa_down");
40     int Psa_up_tNet_0       = tNet.FindNode("Psa_up");
41     int Rep_mean_tNet_0     = tNet.FindNode("Rep_mean");
42     int Rsp_mean_tNet_0     = tNet.FindNode("Rsp_mean");
43     int Shock_tNet_0        = tNet.FindNode("Shock");
44     int Vlv_down_tNet_0     = tNet.FindNode("Vlv_down");
45     int Vlv_up_tNet_0       = tNet.FindNode("Vlv_up");
46
47 // Order 1.
48     int CardOut_up_tNet_1   = tNet.FindNode("CardOut_up_1");
49     int EF_mean_tNet_1      = tNet.FindNode("EF_mean_1");
50     int EmaxLV_mean_tNet_1  = tNet.FindNode("EmaxLV_mean_1");
51     int HR_mean_tNet_1      = tNet.FindNode("HR_mean_1");
52     int Plv_up_tNet_1       = tNet.FindNode("Plv_up_1");
53     int Psa_down_tNet_1     = tNet.FindNode("Psa_down_1");
54     int Psa_up_tNet_1       = tNet.FindNode("Psa_up_1");
55     int Rep_mean_tNet_1     = tNet.FindNode("Rep_mean_1");
56     int Rsp_mean_tNet_1     = tNet.FindNode("Rsp_mean_1");
57     int Vlv_down_tNet_1     = tNet.FindNode("Vlv_down_1");
58     int Vlv_up_tNet_1       = tNet.FindNode("Vlv_up_1");
59
60 // Copy the CPTs from the trained network to the dynamic network - Order 0.
61 CopyCPT(CardOut_up_tNet_0, CardOut_up_t, 0);
62 CopyCPT(EF_mean_tNet_0, EF_mean_t, 0);
63 CopyCPT(EmaxLV_mean_tNet_0, EmaxLV_mean_t, 0);
64 CopyCPT(HR_mean_tNet_0, HR_mean_t, 0);
65 CopyCPT(Plv_up_tNet_0, Plv_up_t, 0);
66 CopyCPT(Psa_down_tNet_0, Psa_down_t, 0);
67 CopyCPT(Psa_up_tNet_0, Psa_up_t, 0);
68 CopyCPT(Rep_mean_tNet_0, Rep_mean_t, 0);
69 CopyCPT(Rsp_mean_tNet_0, Rsp_mean_t, 0);
70 CopyCPT(Shock_tNet_0, Shock_t, 0);
71 CopyCPT(Vlv_down_tNet_0, Vlv_down_t, 0);
72 CopyCPT(Vlv_up_tNet_0, Vlv_up_t, 0);
73
74 // Order 1.
75 CopyCPT(CardOut_up_tNet_1, CardOut_up_t, 1);
76 CopyCPT(EF_mean_tNet_1, EF_mean_t, 1);
77 CopyCPT(EmaxLV_mean_tNet_1, EmaxLV_mean_t, 1);
78 CopyCPT(HR_mean_tNet_1, HR_mean_t, 1);
79 CopyCPT(Plv_up_tNet_1, Plv_up_t, 1);
80 CopyCPT(Psa_down_tNet_1, Psa_down_t, 1);
81 CopyCPT(Psa_up_tNet_1, Psa_up_t, 1);
82 CopyCPT(Rep_mean_tNet_1, Rep_mean_t, 1);
83 CopyCPT(Rsp_mean_tNet_1, Rsp_mean_t, 1);
84 CopyCPT(Shock_tNet_0, Shock_t, 1);           // Order "1".
85 CopyCPT(Vlv_down_tNet_1, Vlv_down_t, 1);
86 CopyCPT(Vlv_up_tNet_1, Vlv_up_t, 1);
87 }
88
89 //-----
90
91 void CardioVas::CreateDBNStructure(void){
92     // Initialize nodes for the dynamic network.
93     int CardOut_up_t   = dynNet.AddNode(DSL_CPT, "CardOut_up");
94     int EF_mean_t      = dynNet.AddNode(DSL_CPT, "EF_mean");
95     int EmaxLV_mean_t  = dynNet.AddNode(DSL_CPT, "EmaxLV_mean");
96     int HR_mean_t      = dynNet.AddNode(DSL_CPT, "HR_mean");
97     int Plv_up_t       = dynNet.AddNode(DSL_CPT, "Plv_up");
98     int Psa_down_t     = dynNet.AddNode(DSL_CPT, "Psa_down");
99     int Psa_up_t       = dynNet.AddNode(DSL_CPT, "Psa_up");
100    int Rep_mean_t     = dynNet.AddNode(DSL_CPT, "Rep_mean");
101    int Rsp_mean_t     = dynNet.AddNode(DSL_CPT, "Rsp_mean");
102    int Shock_t         = dynNet.AddNode(DSL_CPT, "Shock");

```

```

103 int Vlv_down_t      = dynNet.AddNode(DSL_CPT, "Vlv_down");
104 int Vlv_up_t        = dynNet.AddNode(DSL_CPT, "Vlv_up");
105
106 // Set the outcomenames.
107 int handle = dynNet.GetFirstNode();
108 while (handle != DSL_OUT_OF_RANGE){
109     // Map variable in dataset to dynNet.
110     string id(dynNet.GetNode(handle)->Info().Header().GetId());
111     int v = dataset.FindVariable(id);
112
113     // Get the stateNames.
114     vector<string> stateNamesStr = dataset.GetStateNames(v);
115     DSL_stringArray stateNames;
116     vector<string>::iterator iter = stateNamesStr.begin();
117     for(iter; iter < stateNamesStr.end(); iter++){
118         stateNames.Add(const_cast<char*>(iter->c_str()));
119     }
120     dynNet.GetNode(handle)->Definition()->SetNumberOfOutcomes(stateNames);
121
122     // Go to next node in dynNet.
123     handle = dynNet.GetNextNode(handle);
124 }
125
126 // Set temporal types.
127 dynNet.SetTemporalType(CardOut_up_t, dsl_plateNode);
128 dynNet.SetTemporalType(EF_mean_t, dsl_plateNode);
129 dynNet.SetTemporalType(EmaxLV_mean_t, dsl_plateNode);
130 dynNet.SetTemporalType(HR_mean_t, dsl_plateNode);
131 dynNet.SetTemporalType(Plv_up_t, dsl_plateNode);
132 dynNet.SetTemporalType(Psa_down_t, dsl_plateNode);
133 dynNet.SetTemporalType(Psa_up_t, dsl_plateNode);
134 dynNet.SetTemporalType(Rep_mean_t, dsl_plateNode);
135 dynNet.SetTemporalType(Rsp_mean_t, dsl_plateNode);
136 dynNet.SetTemporalType(Shock_t, dsl_plateNode);
137 dynNet.SetTemporalType(Vlv_down_t, dsl_plateNode);
138 dynNet.SetTemporalType(Vlv_up_t, dsl_plateNode);
139
140 // Add arcs - Order 0.
141 dynNet.AddArc(EF_mean_t, CardOut_up_t);
142 dynNet.AddArc(HR_mean_t, CardOut_up_t);
143 dynNet.AddArc(Plv_up_t, EF_mean_t);
144 dynNet.AddArc(Vlv_down_t, EF_mean_t);
145 dynNet.AddArc(Vlv_up_t, EF_mean_t);
146 dynNet.AddArc(Psa_down_t, HR_mean_t);
147 dynNet.AddArc(Psa_up_t, HR_mean_t);
148 dynNet.AddArc(EmaxLV_mean_t, Plv_up_t);
149 dynNet.AddArc(HR_mean_t, Plv_up_t);
150 dynNet.AddArc(Psa_up_t, Plv_up_t);
151 dynNet.AddArc(Psa_up_t, Psa_down_t);
152 dynNet.AddArc(Rep_mean_t, Psa_down_t);
153 dynNet.AddArc(Rsp_mean_t, Psa_down_t);
154 dynNet.AddArc(EmaxLV_mean_t, Psa_up_t);
155 dynNet.AddArc(Rep_mean_t, Psa_up_t);
156 dynNet.AddArc(Rsp_mean_t, Psa_up_t);
157 dynNet.AddArc(CardOut_up_t, Shock_t);
158 dynNet.AddArc(Psa_down_t, Shock_t);
159 dynNet.AddArc(Psa_up_t, Shock_t);
160 dynNet.AddArc(EmaxLV_mean_t, Vlv_down_t);
161 dynNet.AddArc(Plv_up_t, Vlv_down_t);
162 dynNet.AddArc(EmaxLV_mean_t, Vlv_up_t);
163 dynNet.AddArc(Plv_up_t, Vlv_up_t);
164 dynNet.AddArc(Vlv_down_t, Vlv_up_t);
165
166 // Order 1.
167 dynNet.AddTemporalArc(CardOut_up_t, CardOut_up_t, 1);
168 dynNet.AddTemporalArc(EF_mean_t, EF_mean_t, 1);
169 dynNet.AddTemporalArc(EmaxLV_mean_t, EmaxLV_mean_t, 1);

```

```

170     dynNet.AddTemporalArc(HR_mean_t, HR_mean_t, 1);
171     dynNet.AddTemporalArc(Plv_up_t, Plv_up_t, 1);
172     dynNet.AddTemporalArc(Psa_down_t, Psa_down_t, 1);
173     dynNet.AddTemporalArc(Psa_up_t, Psa_up_t, 1);
174     dynNet.AddTemporalArc(Rep_mean_t, Rep_mean_t, 1);
175     dynNet.AddTemporalArc(Rsp_mean_t, Rsp_mean_t, 1);
176     dynNet.AddTemporalArc(Vlv_down_t, Vlv_down_t, 1);
177     dynNet.AddTemporalArc(Vlv_up_t, Vlv_up_t, 1);
178
179     // Set number of time-slices.
180     dynNet.SetNumberOfSlices(2);
181 }
182
183 //-----
184
185 void CardioVas::CreateTransitionNetStructure(void){
186     // Initialize.
187     int prevSlices = dynNet.GetNumberOfSlices();
188     dynNet.SetNumberOfSlices(2);
189     dynNet.UnrollNetwork(tNet);
190     dynNet.SetNumberOfSlices(prevSlices);
191
192     // Set [DSL_NOLEARN = ALL] userproperty.
193     DSL_node* temp = tNet.GetNode(tNet.FindNode("Shock_1"));
194     temp->Info().UserProperties().AddProperty("DSL_NOLEARN", "ALL");
195 }
196
197 //-----
198
199 void CardioVas::LearnTransitionNet(void){
200
201     // Map network to dataset.
202     for (int v = 0; v < dataset.NumVariables(); v++){
203         dataset.SetHandle(v, tNet.FindNode(dataset.GetId(v).c_str()));
204     }
205
206     // Learn the TransitionNet.
207     DSL_em learnParams;
208     learnParams.SetRandomizeParameters(false);
209     learnParams.Learn(dataset, tNet);
210 }
211
212 //-----
213
214 void CardioVas::LoadDataFile(string datafile){
215     // Initialize.
216     DSL_textParser parser;
217
218     // Parse the data file.
219     parser.SetUseHeader(true);
220     parser.SetTypesSpecified(false);
221     int result = parser.Parse(datafile.c_str());
222     if (result != DSL_OKAY) return result;
223
224     // Obtain the data.
225     dataset = parser.GetDataset();
226 }
227
228 //-----
229
230 void CardioVas::Run(string datafile){
231     cout << "Load_data_file..." << endl;
232     LoadDataFile(datafile);
233
234     cout << "Create_DBN_structure..." << endl;
235     CreateDBNStructure();
236

```

```

237 cout << "Create_tNet_structure..." << endl;
238 CreateTransitionNetStructure();
239
240 cout << "Learn_tNet_structure..." << endl;
241 LearnTransitionNet();
242
243 cout << "Copy_parameters_to_DBN..." << endl;
244 CopyDBNParameters();
245 SaveDBN("cv_steady_dbn_learned.xdsl");
246 cout << "Done..." << endl;
247 }
248
249 //-----
250
251 void CardioVas::SaveDBN(string dynNetStr){
252     return dynNet.WriteFile(dynNetStr.c_str());
253 }
254
255 //-----

```

Listing C.3: *cvdbn.cpp*

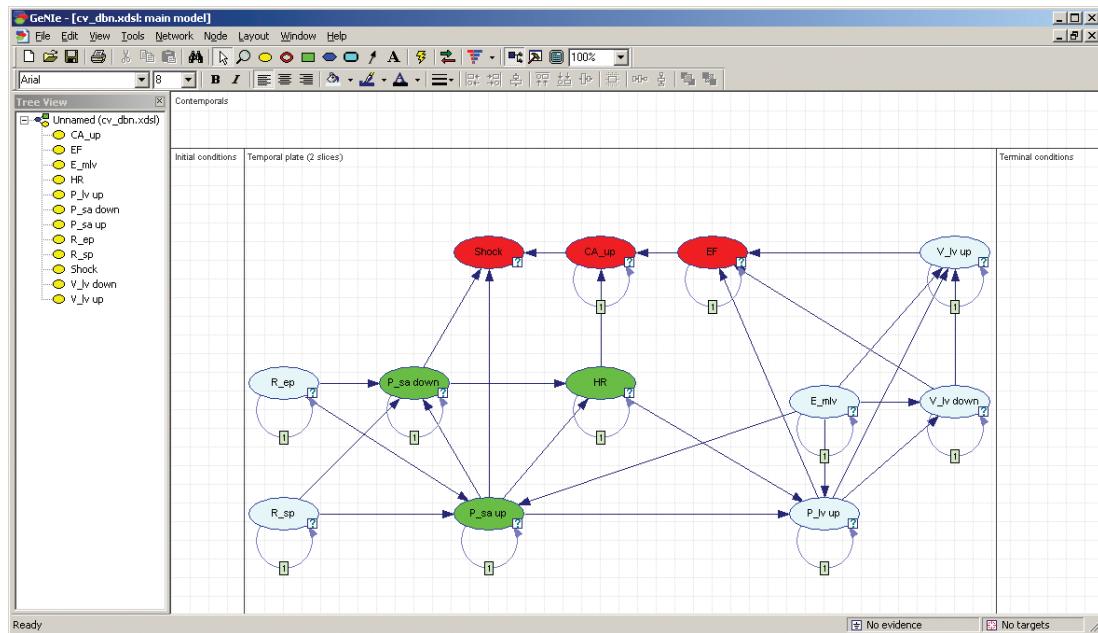


Figure C.1: Resulting cardiovascular system DBN in GeNIe.