GeNIeRate: An Interactive Generator of Diagnostic Bayesian Network Models

Pieter Kraaijeveld

THESIS
Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Knowledge Based Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science

Graduation Committee:
Dr. Drs. L.J.M. Rothkrantz
Dr. A.H.J. Oomes
Dr. K. van der Meer
Dr. Ir. Marek J. Druzdzel

June 13, 2005

Abstract

Constructing diagnostic Bayesian network models is a complex and time consuming task. In this thesis, we propose a methodology to simplify and speed up the design of very large Bayesian network models. The models produced using our methodology are based on two simplifying assumptions: (1) the structure of the model has three layers of variables and (2) the interaction among the variables can be modeled by canonical models such as the Noisy-MAX gate. The methodology is implemented in an application named GeNIeRate, which aims at supporting construction of diagnostic Bayesian network models consisting of hundreds or even thousands of variables. Preliminary qualitative evaluation of GeNIeRate shows great promise. The prediction is that GeNIeRate can reduce the model building time for an inexperienced Bayesian network model builder by 20-30%. We conducted an experiment comparing our approach to traditional techniques for building Bayesian network models by rebuilding a diagnostic Bayesian network model for liver disorders, HEPAR-II. We found that the performance of the model created with GeNIeRate is better than the performance of the original HEPAR-II.

Acknowledgements

I want to thank several people who made it possible for me to do my thesis work at the Decision Systems Laboratory (DSL) of the University of Pittsburgh.

First of all, I want to thank my advisor at the University of Pittsburgh, Dr. Ir. Marek J. Druzdzel. Marek showed me how exciting research can be. During my six months in Pittsburgh I learned a lot of him. Without Marek it would not be possible for me to publish my first paper. Second, I want to thank Dr. Drs. L.J.M. Rothkrantz, who made it possible for me to visit the University of Pittsburgh and who has provided me with guidance and feedback in my work.

Next, I also want to thank several people who helped me with my thesis work: Tomek Loboda, Doug Campbell, Adam Zagorecki, Agnieszka Oniśko, Tomek Sowinski, John Mark Agosta, Thomas Gardos, Mark Voortman, Hanna Wasyluk and last but not least my girlfriend Anna-Gerdien Bruna who supported me during my stay in Pittsburgh and helped making GeNIeRate look better due to her excellent graphical design skills.

Furthermore, my stay at the University of Pittsburgh from October 2004 to March 2005 would not have been possible without the financial support of:

- My parents, Kees & Jetty Kraaijeveld
- Fundatie van de Vrijvrouwe van Renswoude
- Stimuleringsfonds voor Internationale Universitaire Samenwerkingsrelaties (STIR)
- Faculteitsfonds
- Universiteitenfonds Delft

Contents

Chapter 1

Introduction

This thesis describes my research work done at the Decision Systems Laboratory (DSL) of the School of Information Science (SIS) of the University of Pittsburgh. Basically, I developed an interactive application to speed up building large diagnostic Bayesian networks based on a methodology with several simplifying assumptions.

1.1 Context

Diagnosis is the process of finding the root cause of a system failure given a set of system observations: symptoms, sensor readings, error codes, test results, historical findings etc. While diagnosis applied in the medical domain is well known, it is also applied in industry, management and various other domains. Over the years researchers tried to automate diagnosis with various techniques. These techniques help finding the cause(s) of the failure faster and, therefore, minimize the loss caused by that failure. The modeling methods include for example fault trees, rule bases, and probabilistic models. In the last two decades, the probabilistic models found great interest. One prominent tool for modeling diagnosis using probability theory is known as Bayesian networks, which we will use in this thesis as the modeling technique.

Bayesian networks (BN) [Pearl, 1988] are acyclic directed graphs with each node representing a variable and each arc representing typically a causal relation among two variables. Although exact and approximate inference in Bayesian networks are both worst-case NP-hard [Cooper, 1990; Dagum and Luby, 1997], they still perform well for practical diagnostic models consisting of several hundreds or even thousands of variables.

A BN consists of a qualitative and a quantitative part. The qualitative part is an acyclic directed graph reflecting the causal structure of the domain, the quantitative part represents the joint probability distribution over its variables. Every variable has a conditional probability table (CPT) representing the probabilities of each state given the states of the parent variables. If a variable does not have any parent variables in the graph, the CPT represents the prior probability distribution of the variable. A BN is able to calculate the posterior probability of an uncertain variable given some evidence obtained from related variables. This is called *evidence propagation* or *belief updating*. This property and the intuitive way BN model complex relationships among uncertain variables makes it a very suitable technique for building diagnostic models. Diagnosis is quite likely the most successful practical application of BNs.

1.2 Problem domain

While the existing diagnostic BN models perform well, the technique is still not widely used and accepted. One of the main reasons for this is that building a BN model is a laborious and time consuming task. During the model building process, both the qualitative and quantitative parts of the BN have to be designed. This can be done in three different ways: (1) learning both structure and parameters from data without human interaction, (2) consulting a domain expert to design the structure and the parameters, or (3) combining learning from data and expert knowledge. In order to learn successful diagnostic BN models from data one would need a very large data set, which is rarely available for diagnostic models. It is never available for new devices or devices that are designed for high reliability such as, for example airplanes, nuclear reactors, oil refineries, etc. So building a diagnostic BN model will most of the time come down to an interaction of a knowledge engineer and a domain expert. They will consult technical manuals, test procedures, and repair databases to define the variables in that domain, determine the interactions among them, and to elicit the parameters. To give an idea of the time needed to develop a diagnostic BN model: the construction of the HEPAR-II model [Oniśko *et al.*, 2001] used to diagnose liver disorders took more than 300 hours of which roughly 50 hours were spent with domain experts. The model consists of 70 variables and the numerical parameters were learned from a data set of patient cases. In another diagnostic Bayesian network, the *PATHFINDER* system [Heckerman *et al.*, 1992], used for lymph node pathology, 14000 conditional probabilities had

to be assessed by expert pathologists. One could imagine that a lot of errors will be made when eliciting such a big amount of probabilities, this, because of tiring out the expert pathologists.

Software to build Bayesian networks is widely available. However, most of this software is developed for educational use or for users who are BN experts. Thus, for example, if a medical doctor decides to build a BN to diagnose lung cancer, the doctor will need a BN model building expert to build the model using this software. Now, as is discussed above, the BN expert will help the doctor to identify the variables in the domain, creating the interactions among them, and eliciting the probabilities. The elicitation of the probabilities is a very difficult task. The BN expert will help the doctor to quantify the probabilities by asking questions. Most of the time a domain expert does have some feeling about the probabilities in a relative sense (X bigger/smaller than Y) but thinks it is hard to quantify them in an absolute matter. Most BN model building software do not support any visual feed-back of the elicited probabilities, like, for example, thickness of the arcs, to indicate the impact of the elicited probabilities. This means that when a large model is created, which does not perform as expected, it is very hard to debug the model and find the erroneous conditional probabilities.

1.3 Objectives and assignment

This being said, the main objective to be met in this thesis is to develop a methodology for building large diagnostic Bayesian network models in a fast and easy way. The methodology has to be such that a diagnostic Bayesian network model, which is complete and comprehensive for its domain, can be built by users who are not necessarily Bayesian network experts. The methodology has to be implemented in an application and has to meet some goals listed below. The application must:

- be a general purpose application, which means it can be applied to a variety of diagnostic domains.
- have an intuitive graphical user interface which helps a domain expert without knowledge of Bayesian networks to construct a diagnostic model.

To fulfill these requirements, we made the following two simplifying assumptions that will reduce the complexity of the models. The application, therefore, must:

- generate Bayesian networks which have a simple structure, consisting of only a few layers of variables.
- use special models of interaction among variables that (1) minimize the number of numerical parameters that used to be elicited from experts and (2) can be used in specialized belief updating algorithms.

With these simplifications some theoretical modeling power and precision will be sacrificed. However, our study shows that the resulting models will not be significantly less accurate in terms of diagnostic performance, while being much easier and faster to build.

To summarize the objectives into an assignment. The assignment of this thesis is to:

- design a predefined Bayesian network model, which is based on the simplifying assumptions described above, and can be used as a "template" model to build very large diagnostic Bayesian networks.
- design a methodology to build very large diagnostic BN models using the designed "template".
- implement a prototype application, based on the methodology, to support the user, who is not necessarily a BN expert, to build a very large diagnostic BN model.
- test the idea's of the "template", the methodology, and the prototype, and test whether the diagnostic performance of the created models suffered, because of the simplifying assumptions it is based on.

1.4 Overview of the thesis

The remainder of this thesis report is structured as follows. Chapter 2 discusses some background material about Bayesian networks, diagnosis, and specialized canonical interaction models. Chapter 3 discusses related work on this topic. Chapter 4 explains our proposed "template" of diagnostic BN models, and the methodology to build them. Chapter 5 gives a complete description of our prototype application: GeNIeRate. Chapter 6 discusses a qualitative study of GeNIeRate, and a quantitative empirical study comparing our methodology to traditional model building techniques. Finally, Chapter 7 states the conclusions and discusses different ideas for future research.

Chapter 2

Background material

This chapter provides some background material, it is a survey about Bayesian networks (Section 2.1), automating diagnosis (Section 2.2), and canonical interaction models (Section 2.3).

2.1 Bayesian networks

Bayesian networks [Pearl, 1988] are acyclic directed graphs in which nodes represent random variables and arcs represent direct probabilistic dependencies among them. A Bayesian network encodes the joint probability distribution over a set of variables $\{X_1, \ldots, X_n\}$, where n is finite, and decomposes it into a product of conditional probability distributions over each variable given its parents in the graph. In case of nodes with no parents, prior probability is used. The joint probability distribution over $\{X_1, \ldots, X_n\}$ can be obtained by taking the product of all of these prior and conditional probability distributions:

$$\Pr(x_1, \ldots, x_n) = \prod_{i=1}^{n} \Pr(x_i | Pa(x_i)) \; . \tag{2.1}$$

Figure 2.1 shows a highly simplified example Bayesian network modeling causes of a car engine failing to start. The variables in this model are: *Age* of the car (A), dead *Battery* (B), dirty *Connectors* (C), *Engine* does not start (E) and *Radio* does not work (R). For the sake of simplicity, we assumed that each of these variables is binary. For example, R has two outcomes, denoted r and $\overline{r}$, representing "Radio fails" and "Radio works," respectively.

A directed arc between B and E denotes the fact that whether or not the battery is dead will impact the likelihood of the engine failing to start.

Similarly, an arc from A to B denotes that the age of the car influences the likelihood of having a dead battery.

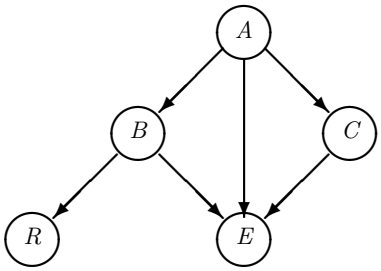


Figure 2.1: An example belief network for engine problem

Lack of directed arcs is also a way of expressing knowledge, notably assertions of (conditional) independence. For instance, lack of a directed arc between A and R encodes the knowledge that the age of the car does not influence the chance whether the radio of the car works or not, only indirectly through the variable *dead battery* B. These causal assertions can be translated into statements of conditional independence: R is independent of A given B. In mathematical notation,

$$\Pr(R|B) = \Pr(R|B, A) \; . \tag{2.2}$$

Similarly, the absence of arc $B \rightarrow C$ means that whether or not the car has a dead battery will not influence the chance of having dirty connectors.

These independence properties imply that:

$$\Pr(a, b, c, r, e) = \Pr(a) \; \Pr(b|a) \; \Pr(c|a) \; \Pr(r|b) \; \Pr(e|a, b, c) \; , \tag{2.3}$$

i.e., that the joint probability distribution over the graph nodes can be factored into the product of the conditional probabilities of each node given its parents in the graph. Please note that this expression is just an instance of Equation 2.1.

The assignment of values to observed variables is usually called *evidence*. The most important type of reasoning in a probabilistic system based on Bayesian networks is known as *belief updating* or *evidence propagation*, which amounts to computing the probability distribution over the variables of interest given the evidence. This evidence propagation makes Bayesian networks very suitable for diagnosis. For example, in the model of Figure 2.1, the variables of interest for diagnosis could be B and C and the focus of computation could be the posterior probability distribution over B and C given

the observed values of A, R, and E, i.e., $\Pr(b, c|a, r, e)$, often approximated in practice as marginal probability distributions, $\Pr(b|a, r, e)$ and $\Pr(c|a, r, e)$. Bayesian network software can be applied to calculate these posterior probabilities. Although exact and approximate inference in Bayesian networks are both worst-case NP-hard [Cooper, 1990; Dagum and Luby, 1997], algorithms embedded in today's software are capable of very fast belief updating in models consisting of hundreds or even thousands of variables. After the belief updating, the software can make a decision or support the user in making a decision what actions to perform given that probability.

2.2 Automating diagnosis

In Chapter 1, a short description of diagnosis is already given. This section will cover a more thorough description of diagnosis and it will give some historical background of techniques to automate the diagnostic process.

2.2.1 Diagnosis

Diagnosis is a common and important problem that is performed daily in many different domains. The medical domain is probably the best known example but diagnosis is also applied in engineering, business, and several other domains. Diagnosis (from the Greek words $\delta\iota\alpha$ = by and $\gamma\nu\sigma\sigma\iota\varsigma$ = knowledge) is the process of identifying the disorders, diseases or malfunctions of an object by considering signs, symptoms, tests, historical facts, various diagnostic procedures, or other facts which caused or are caused by the disorder. De Kleer [1990] described it a little different. He claims that the diagnostic task is to determine why a correctly designed system is not functioning as it was intended. This task comes down to identifying what is wrong in a system given some observations. Both descriptions describe diagnosis as a process of searching the causes of a misbehavior.

After identifying that something is not working correctly in a system, the diagnostic process consists of two different steps, the first step is to acquire as much information as possible explaining the misbehavior of the system by sequentially performing different tests. The second step is to, given all the gathered information, identify which component, or which combination of components of the system, is most probable of having a faulty behavior. After this process the (most probable) faulty components of the system can be repaired. Figure 2.2 shows this sequential process graphically.

In the diagnostic process the diagnostician has to make a decision when to stop the process and conclude what is wrong. This has to be done as

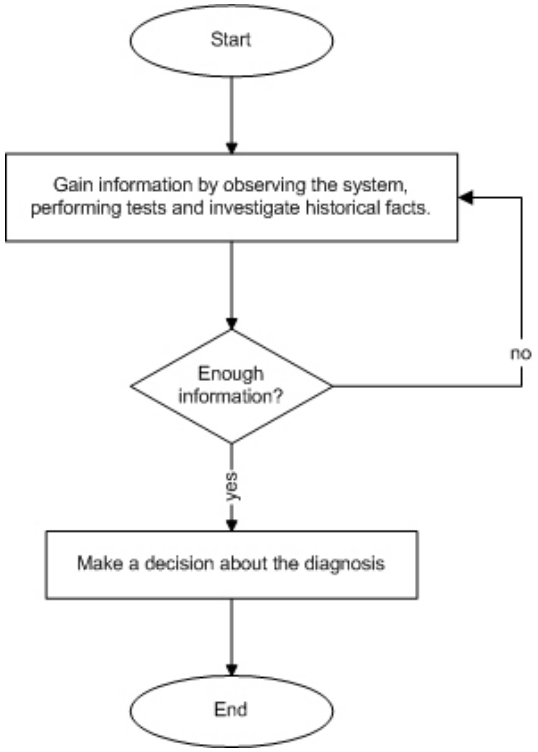


Figure 2.2: The diagnostic process

fast and cheap as possible. Therefore, it becomes an optimization problem which is of great interest of AI research. Over the years, several ways are developed to automate diagnosis and to support the decision making. The next section will give some historical remarks and an overview of some of the main approaches how to automate diagnosis.

2.2.2 Automated diagnosis

Early research to automate the diagnosis process used Bayesian reasoning and decision theory [Ledley and Lusted, 1959] and proposed different techniques to help a medical doctor making a diagnosis. Their pioneering paper was followed by papers written by others describing a huge variety of techniques and methods to automate the (medical) diagnosis process. This interest became less at the end of 1970s because the computers at that time were not powerful enough to compute the complex probabilistic queries within reasonable time.

In the beginning of the 1980's, there was some success to automate diagnosis with expert systems using rule-bases. One of the first of these systems, used to diagnose acute abdominal pain, was developed by De Dombal [1972]. This system was intended for a narrow, well-defined diagnostic problem where the clinician had to decide between a limited number of diagnoses. Another rule-based expert system is the INTERNIST-I system for internal medicine [Miller *et al.*,]. The most valuable product of INTERNIST-I system was its medical knowledge base. This knowledge base was later used as a basis of successor systems. One of these successor systems was the Quick Medical Reference (QMR), which is a commercialized decision support system for internists [Myers, 1987]. While the rule-based systems to automate diagnosis had some successful results they also had some disadvantages. The main disadvantages of rule-based systems are that they have a strong domain dependent character, they are very hard and time consuming to build and they are not practical to use.

Other approaches to automate diagnosis include decision trees [Quinlan, 1986], fault trees [Madden and Nolan, 1999], multi-layer perceptrons, and probability estimation [Stensmo and Sejnowski, 1994]. Almost all of these techniques have one common main disadvantage which is the need of a complete data set when inferring a diagnostic query. It takes a long time before such a data set is created and is therefore almost never available for diagnostic problems.

The development of Bayesian networks (BNs) (Section 2.1) used as probabilistic graphical models encouraged researchers to focus again on the probabilistic techniques to automate diagnosis. One of the first large diagnostic systems that used Bayesian networks was the QMR-DT system [Shwe *et al.*, 1991; Middleton *et al.*, 1991]. This system was a probabilistic reformulation of the Quick Medical Reference (QMR) described above. The QMR-DT system contains approximately 5000 variables. Other examples of diagnostic BN models are the *PATHFINDER* system [Heckerman *et al.*, 1992] used for lymph node pathology, and the *HEPAR-II* system as mentioned already before in Chapter 1. An example of a non medical domain model is the *SACSO* project [Jensen *et al.*, 2001]. This system was built to diagnose printer failures and helps users to solve their printer problems themselves without calling a help desk or sending the printer back to the factory.

Bayesian networks are able to calculate the posterior probability of a variable given some evidence from related variables (i.e., evidence propagation, see Section 2.1). This property and the intuitive way Bayesian networks model complex relationships among uncertain variables makes it a very suitable technique for building diagnostic models. Therefore, we will

use Bayesian networks as the modeling technique to build diagnostic models. The next section will give a more thorough description of Bayesian networks applied in diagnosis.

2.2.3 Diagnostic Bayesian network models

There are various techniques to model the graphical structure (qualitative part) of a Bayesian network when it is applied for diagnosis. The most simple one is based on the assumption that only one *fault* can occur at the same time and that *observations* are conditionally independent. The structure of this model is such that there is only one *fault* variable, which has a separate state for each single fault, with one or more *observation* variables as its children. This structure is called: *Naive-Bayes model* or *Idiot's Bayes model* [Friedman *et al.*, 1997], since the single fault assumption is a somewhat naive assumption. The main advantage of this single fault model is that the number of conditional probabilities is low which makes it attractive from the computational point of view. Figure 2.3 shows an example Naive-Bayes structure, with one fault variable F and three observation variables $O1 \ldots O3$.

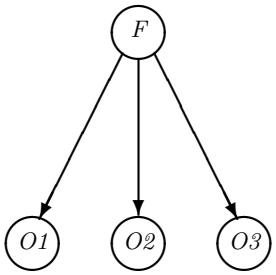


Figure 2.3: Naive-Bayes model

It is also possible to model diagnosis supporting more than one fault. Models like this are called: *Multiple fault models*. Multiple fault models have typically separate *fault* variables for every fault state. These *fault* variables are connected to the *observation* variables which are caused by them. A multiple fault structure gives a more realistic model, however, using this model will result in eliciting a lot more conditional probabilities. For example, if we have a binary *observation* variable which is caused by two binary parent *fault* variables, the CPT already consists of 8 entries. The size

of the CPT of the *observation* variable grows exponentially in the number of parent variables. There are various techniques available to reduce this exponential growth, these techniques are discussed in Section 2.3. Figure 2.4 gives an example of the multiple fault model with two *fault* variables *F1* and *F2* and three *observation* variables *O1*, *O2*, and *O3*.

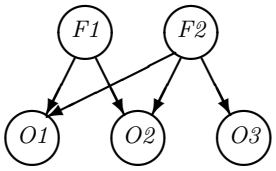


Figure 2.4: Multiple fault model

The multiple fault model in the example above is also based on two independence assumptions. There are no relations among the *fault* variables and also no relations among the *observation* variables. By allowing these relations an extended multiple fault model will be created. This can make the model more accurate but again it will increase the number of parameters that have to be elicited. Figure 2.5 shows an example of an extension of the multiple fault model presented in Figure 2.4.

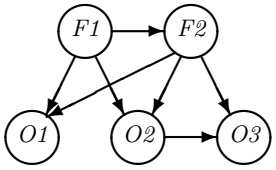
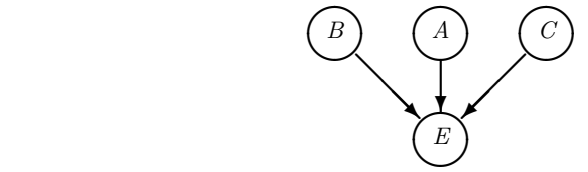


Figure 2.5: Multiple fault model with dependencies among variables of the same type

2.3 Canonical interaction models

In a Bayesian network, every variable contains a Conditional Probability Table (CPT) representing the probabilities of each state given the state of the parent variable. If a variable does not have any parent variables in the graph, the CPT represents the prior probability distribution of the variable. Figure 2.6 shows a simple CPT of the variable E representing "Engine does not start" of the example of Figure 2.1. E has three parent variable: A, C,

and B representing the Age of the car, dirty Connectors, and dead Battery respectively. This CPT now consist of 24 entries representing all possible scenarios among the four variables. The probabilities within the table are either learned from a data set or elicited by a domain expert.



Dead Battery	present					
Dirty Connectors	present			absent		
Age	more_than_...	new_to_ten...	new	more_than_...	new_to_ten...	...etc
present	0.99	0.98	0.96	0.98	0.96	
absent	0.01	0.02	0.04	0.02	0.04	

Figure 2.6: The CPT of the variable: Engine does not start

The size of a CPT grows exponentially in the number of parents of that variable. Thus, in the example above, if the variable E gets one more parent variable, the size of the CPT will grow from 24 to 48 entries. In order to gain speed in the model building process, researchers developed canonical interaction models which approximate the CPTs and require fewer parameters. These canonical interaction models are known as: *gates*.

One type of canonical interaction, widely used in Bayesian networks, is known as the Noisy-OR gate. This gate was first introduced outside the BN domain by [Good, 1961]. Later it was applied in the context of BNs [Pearl, 1986]. It became very popular among BN model builders since it reduces the growth of a CPT of a variable from exponential to linear in the number of parents. The Noisy-OR gate can only be used for binary variables, an extension for multi-valued variables is the Noisy-MAX gate [Henrion, 1989; Díez, 1993]. For the sake of simplicity, we will discuss the Noisy-OR gate in depth in Section 2.3.1 and give a shorter description of the extended Noisy-MAX gate in Section 2.3.2.

2.3.1 The Noisy-OR gate

The Noisy-OR gate models a non-deterministic interaction among n binary parent cause variables X and a binary effect variable Y. Every variable has two states: a distinguished state, which represents that the variable is in its normal working state. Commonly this is *absent* or *false* and a

non-distinguished state: *truth* or *present*. The effect variable Y works as a deterministic OR gate. This means that if all the parent variables are *absent*, the child variable is also *absent*. However, if a parent variable X_i is *present* and all other parent variables are *absent*, it has a probability p_i of causing the effect y. These probabilities p_i address the noisy property of the gate and have to be elicited by the model builder or can be learned from data. Every interaction now only consist of one parameter. The probabilities are fairly easy to understand since they can be represented by questions like: *What is the probability that the effect y will occur, given that only one cause X_i is present and all other causes are absent?* In other words,

$$p_i = Pr(y|\overline{x}_1, \overline{x}_2, \ldots, x_i, \ldots, \overline{x}_{n-1}, \overline{x}_n) \; . \tag{2.4}$$

The probability of an effect y to occur given a subset X of causes which are *present* is now formally given by:

$$p(y|X) = 1 - \prod_{i=1}^{n} (1 - p_i) \; . \tag{2.5}$$

This formula is sufficient to derive the complete CPT of Y conditional on its predecessors $X_1, X_2, \ldots, X_n$.

Henrion [1989] proposed a direct extension of the Noisy-OR gate which models that an effect y can also occur if all the causes are *absent*. He called this extension: *leaky Noisy-OR* gate. This can be modeled by introducing an additional parameter p_0, which is called the *leak probability*. This leak probability is formally given by:

$$p_0 = Pr(y|\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n) \; . \tag{2.6}$$

The leak probability represents the phenomenon that an effect occurs spontaneously, i.e., in absence of any of the causes that are modeled explicitly.

In the leaky Noisy-OR gate, p_i $(i \neq 0)$ no longer represents the probability that X_i causes y given that all other parent variables are absent, but rather the probability that Y is present when X_i is present and every other explicit parent causes (all the X_j's such that $j \neq i$) are absent.

An alternative way of eliciting the parameters of a leaky Noisy-OR gate is given in [Díez, 1993]. Díez defined p_i' as the probability that Y will be *true* if X_i is *present* and every other parent of Y including unmodeled causes (the leak) are *absent*:

$$1 - p_i' = \frac{1 - p_i}{1 - p_0} \; . \tag{2.7}$$

His method amounts essentially to asking the expert for these parameters p'_i.

Converting the parameters p'_i to p_i is straightforward:

$$p_i = p'_i + (1 - p'_i)p_0 \ . \tag{2.8}$$

Now when we extend Equation 2.5, it follows that the probability of y given a subset of parent variables X is given in the leaky Noisy-OR gate by the following formula:

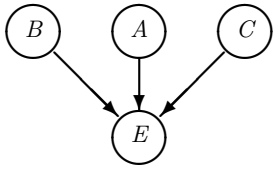
$$p(y|X) = (1 - (1 - p_0)) \prod_{i=1}^{n} \frac{1 - p_i}{1 - p_0} \ . \tag{2.9}$$

The difference between the two proposals (Henrion versus Díez) has to do with the leak variable. While Henrion's parameters p_i assume that the answers of the expert include a combined influence of the parent cause in question and the leak, Díez's parameters p'_i explicitly refer to the mechanism between the parent cause in question and the effect with the leak absent. Conversion between the two parameters is straightforward using Equation 2.8. If the Noisy-OR parameters have to be elicited by a domain expert, Díez's definition is more convenient, since the question to be answered is more intuitive for the expert: *What is the probability that the effect y will occur when you know that all modeled and unmodeled causes X are absent?* It has been found both preferred by experts [Oniśko *et al.*, 2001] and results in higher elicitation accuracy [Zagorecki and Druzdzel, 2004]. Henrion's definition is more convenient if the parameters are learned from data.

2.3.2 The Noisy-MAX gate

The Noisy-OR gate can be generalized to multi-valued variables. These variables are allowed to have more than two states. The difference with the Noisy-OR gate is that the CPT of the multi-valued effect variable Y now is the deterministic MAX function instead of the deterministic OR function. Therefore, it is called Noisy-MAX gate or, if the leak probability is included, leaky Noisy-MAX gate. The variables will, when using the Noisy-MAX gate, still contain one distinguished state.

When applying the Noisy-MAX gate for every interaction in the example above (Figure 2.6), the CPT of the variable E now contains 10 entries instead of 24, see Figure 2.7 (The gray columns represent the distinguished states



Parent	Dead Battery		Dirty Connectors		Age			LEAK
State	present	absent	present	absent	more_tha...	new_to_t...	new	
present	0.95	0	0.2	0	0.25	0.15	0	0.01
absent	0.05	1	0.8	1	0.75	0.85	1	0.99

Figure 2.7: The Noisy-MAX parameters of the variable: Engine does not start

of the parent variables). The user will now only have to elicit 5 parameters since all the columns should add up to one.

To give another example of the reduction of the number of parameters that have to be elicited by the user, we define the number of states of a parent variable X_i as n_{X_i} and the number of states of an effect variable y as n_y. The total number of parameters N that have to be elicited by the model builder using leaky Noisy-MAX becomes:

$$N = \sum_{i=1}^{n} (n_{X_i} - 1)(n_y - 1) + 1 \ , \tag{2.10}$$

compared to the exponential number of parameters using CPT:

$$N = (n_y - 1) \prod_{i=1}^{n} n_{X_i} \ . \tag{2.11}$$

If $n = 10$ and $n_{X_i} = n_y = 3$, we have $2 * 3^{10} = 118,098$ parameters with CPT compared to $10 * 2 * 2 + 1 = 41$ parameters using Noisy-MAX. Every additional parent of y increases this number by multiplying the current number of parameters by 3 in the CPT compared to just adding 4 in case of Noisy-MAX.

Using Noisy-OR and Noisy-MAX gates for some (not all interactions could be approximated by these gates) of the conditional distributions in the HEPAR-II model [Oniśko *et al.*, 2001], not only reduced the number of parameters that had to be elicited from 3,714 to 1,488 but also improved the diagnostic performance of the model. Research shows that many interactions in practical models can be approximated by the Noisy-MAX gates [Zagorecki and Druzdzel, 2005].

Chapter 3

Related work

Software to build Bayesian networks is widely available. Examples are: GeNIe,[1] Hugin,[2] or the Microsoft MSBNx Tool.[3] However, the user must possess knowledge of Bayesian networks when building a diagnostic Bayesian network with one of these tools. Although research addressing the laborious and time consuming task of building large diagnostic Bayesian network models for inexperienced BN model builders is still in its early stages, several applications have already been developed. This chapter evaluates these applications.

3.1 BATS Author

The most prominent work to build a diagnostic BN model for inexperienced domain experts is done by Skaanning [2000]. In the paper Skaanning describes a model building tool for the Systems for Automated Customer Support Operations (SACSO) [Jensen *et al.*, 2001] project. SACSO is a step-by-step troubleshooting tool for Hewlett Packard printers based on Bayesian networks. The Bayesian Automated Troubleshooting System (BATS) Author,[4] developed for the SACSO project, is the knowledge acquisition tool for building diagnostic BN models. To keep model building simple, the models built with BATS Author are based on the simplifying assumption that at most one fault can occur at the same time when diagnosing the system (single-fault assumption). The graphical structure of this single-fault

[1] `http://www.sis.pitt.edu/~genie`
[2] `http://www.hugin.com`
[3] `http://research.microsoft.com/adapt/MSBNx`
[4] `http://www.dezide.com`

Bayesian network model (Naive-Bayes model) is discussed in Section 2.2.3. The elicitation of probabilities in the BATS Author is supported in a natural and intuitive way (Figure 3.2).

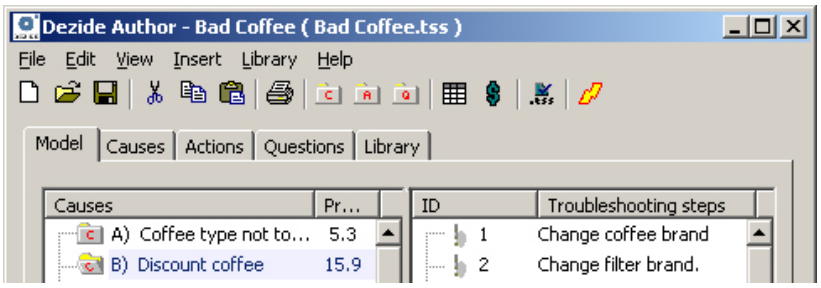


Figure 3.1: The BATS Author tool

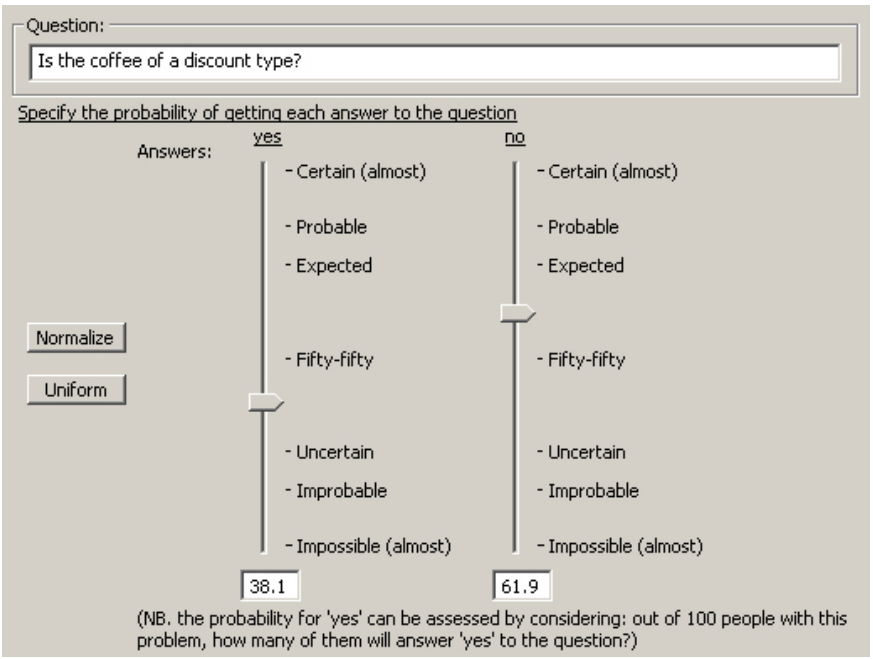


Figure 3.2: An example of the elicitation of probabilities in the BATS Author tool

After building the model with BATS Author other SACSO software can be used which suggests an optimum sequence of troubleshooting steps to arrive at a service action. A major assumption with the SACSO project is

that the agents know which model to use in diagnostics.

We evaluated the BATS Author application with all the members of the Decision Systems Laboratory. We found that, although the elicitation of the probabilities is done in a natural and intuitive way, the application has some drawbacks. First, the single fault assumption may lead to good diagnostic results but is not always very realistic, an observation of an abnormality in the system can be caused by a combination of faults. Second, the BATS Author is designed specifically for the printer domain. We were more interested in a general application which can be applied in any given domain. Furthermore, the BATS Author hides the causality of the models from the user. We found that doing so does not help the model builder in understanding his diagnostic problem as a probabilistic model.

3.2 A user-friendly development tool for medical diagnosis based on Bayesian networks

Another example of a system to construct diagnostic Bayesian networks is discussed by Milho and Fred [2000]. The tool described in the paper supports users in building medical diagnostic Bayesian network models. When the model has been built, it can be used for diagnosis by a web page interface, in this way it supports diagnosis through the internet. One of the goals of the development tool was to offer experts the possibility of designing their particular applications without Bayesian network background knowledge. The models created with the application are based on the simplifying assumption of only having two layers of variables.

Although this system does support diagnosing multiple faults, it has the drawback of being designed specifically for the medical domain. Furthermore, the paper gives a minimal description of the application for the knowledge acquisition. Therefore, it is not possible to understand how the diagnostic BNs are being modeled by the medical expert (the user). Unfortunately, a demo application or some screen-shots of their development tool are not given in the paper and can not be found on the Internet.

3.3 The Design for Serviceability (DFS) Tool

A third diagnostic BN model building tool is described in [Chen, 2003]. Chen describes a "Design for Serviceability" (DFS) Tool for easy and intuitive creation of causality engines for diagnosing complex medical imaging machines based on Bayesian network theory. The tool is used in two phases: first by

design engineers to synthesize knowledge necessary to automatically build a constrained Bayesian model and to assess the serviceability capability of the current system. Second, after the tool is deployed as a web application, field engineers will use the tool to diagnose failures given observations from automated logs of user initiated and manual observations.

Again, this paper lacks a thorough description of the authoring application and how the Bayesian network models are built by the user. The paper does not provide any screen shots of the application and a demo application of this DFS tool is also not available on the web. The paper does describe a good high-level view of the process of authoring, model refinement, and using the model (reporting). The domain for which this tool can be used is very narrow, namely, diagnosing complex medical imaging machines.

3.4 MEDICUS

The next example of a methodology to construct diagnostic Bayesian network models is MEDICUS (Modeling, Explanation, and DIagnostic support for Complex, Uncertain Subject matters) [Schroder *et al.*, 1996]. MEDICUS uses a simplified-natural-language model editor ("linguistic model editor") to build a diagnostic Bayesian network model. After expressing the model

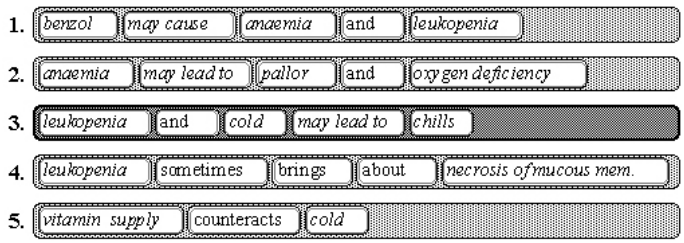


Figure 3.3: Five sentences created in the linguistic model editor of MEDICUS

in this editor, the system can generate an initial graph automatically. Figure 3.3 shows five example sentences created with the linguistic model editor. Alternatively, the user may also create a graph directly in the graphical model editor.

Obviously, MEDICUS only aims at the medical domain. The linguistic model editor helps users without knowledge about Bayesian networks to build an initial model. After an initial graph was created the user can ask the system to propose graph modifications. This means that the system asks the

user to add or remove edges from the graph if the system thinks it is required. The drawbacks of the linguistic model editor is that the sentences of the user can be misinterpreted by the system and second, complex relations among several variables may lead to even more complex sentences. This means that, even though the user does not has to be a BN expert, the user does need some training before starting to model a diagnostic problem.

3.5 Nokia: Knowledge Acquisition Tool (KAT)

The last example of an application is developed at Nokia [Barco *et al.*, 2002]. The authors present an automated troubleshooting tool for cellular networks based on Bayesian networks. Next to this troubleshooting tool, a knowledge acquisition tool (KAT) is presented. KAT converts the knowledge of trou-

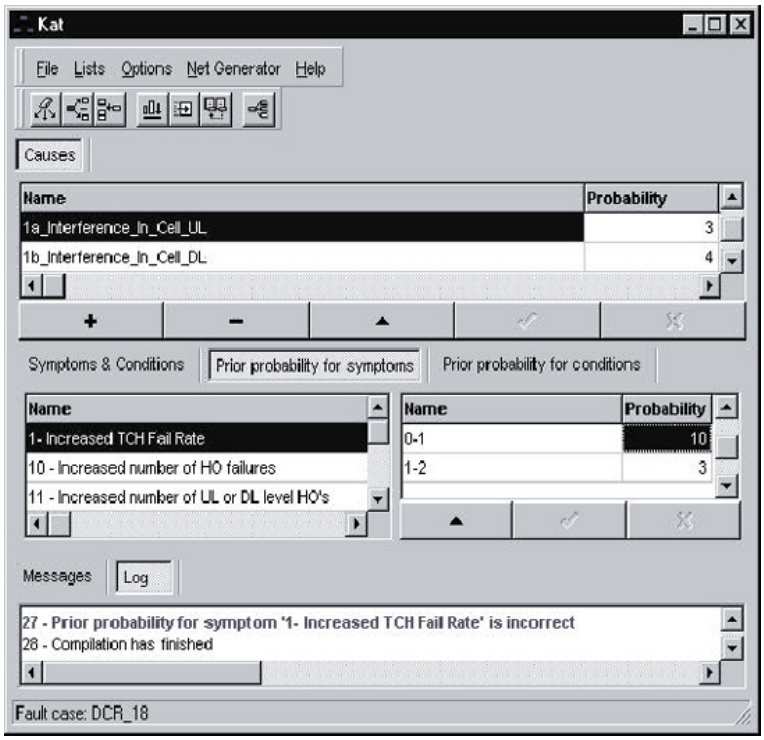


Figure 3.4: KAT main window

bleshooting experts into Bayesian network models by means of a friendly user interface. The application makes use of "Noisy" gates (Section 2.3) to reduce the number of parameters that have to be elicited by the user.

Although a demo application of this KAT tool cannot be found on the internet, the paper does provide one screen-shot of the main window of this application (Figure 3.4).

The domain of this tool is again not general but for a specific field: cellular networks. The screen shot (Figure 3.4) gives a small idea of how KAT acquires knowledge from the domain experts. Unfortunately, the causal structure of the graph is hidden from the user just like in the BATS Author application described earlier in this chapter.

3.6 Evaluation

This chapter presented several applications which are related to the assignment stated in the introduction of this thesis (Chapter 1). Unfortunately, only one demo of an application (BATS Author) can be found on the internet, and two more screen shots of other applications. It is remarkable that all authors of the papers state they had good results using their tool to acquire the knowledge from the domain experts. We think that this is because all these tools were developed for one specific domain or problem and therefore it will fit well in the domain or on the problem it is designed for. Unfortunately, we did not found an example of a methodology or an application that aims at developing diagnostic BN models for any domain, which is one of the goals of this thesis. Table 3.1 compares all the applications presented in this chapter on the following properties:

- The domain of the application.
- Is a demo application available?
- Does the system support single fault or multiple fault diagnosis?
- Does the system also provides software to apply diagnostic reasoning?
- Do the users have to be Bayesian network model building experts?
- Does the application support canonical interaction models ("Noisy" gates) to reduce the number of parameters that have to be elicited by the model builder?

As can be seen, none of the applications is designed specifically for a general domain. This does not mean they cannot be used outside their domain. But since there is a lack of formal tests evaluating this we do not know if this is true. Furthermore, since the one demo application that

Table 3.1: Evaluation of the applications discussed in this chapter

	BATS	Milho	DFS	MEDICUS	KAT
Domain	printer	medical	machines	medical	mob. netw.
Demo available	yes	no	no	no	no
Single/multiple fault	single	multiple	multiple	multiple	multiple
Diagnosis software	yes	yes	yes	yes	yes
Users BN experts	no	no	no	no	no
Noisy gates	no	no	no	no	yes

can be found on the internet does not satisfy our goals, and the software developed at the Decision Systems Laboratory does not contain a tool to build diagnostic BN models for user who are not BN experts, we had to develop a methodology and an application by ourself.

Chapter 4

The BN3M model

One of the main goals of this thesis was to develop a methodology to build large diagnostic Bayesian network models in a fast and intuitive way. In order to accomplish this goal the models that are being produced by the methodology have to lean on some strong simplifying assumptions. This chapter discusses the predefined model that is used as a "template" for all the models build with the methodology.

The predefined model we propose can be seen as an extension of the QMR-DT model described in Section 2.2. The authors made several simplifying assumptions to deal with the complexity of constructing this large network. The system was a multiple fault BN with only two layers of variables representing two different types of variables: *diseases* and *findings*. The structure was such that the *disease* variables influence the outcome of the *findings*. While this structure was simple, its performance was close to the original QMR. But since the independence assumptions made were explicit, the inconsistencies of QMR-DT could easily be explained. Inspired by the graphical structure of the QMR-DT system and its performance, we developed an extension of this model. This model will be called: 3-layer Bayesian network using Noisy-MAX gates (BN3M). Section 4.1 describes the qualitative part of the model (its structure). Section 4.2 discusses the quantitative part of the model, i.e., how the interactions among all the variables are approximated. Finally, Section 4.3 summarizes the motivation of choosing this model and describes the methodology of building the BN3M models.

4.1 Qualitative design

We believe that there are three fundamental types of variables in diagnostic models. The first type (1) are variables representing the failures of the device or a specific part of the device. We will call these failure variables *Faults*. The second type (2) are variables which have an observable effect if the device is in some faulty state. When the effect of the given *fault* can not be observed clearly, a test can be performed which will give information about the state of the device. These observation or test variables will be called: *Evidence* variables. The last type of variables (3) are variables which indicate context properties of the device that may influence the risk of causing a *fault*. These variables can also be seen as enabling conditions of the faults. We will call these variables: *Context* variables. Example *context* variables are the age of the device or the history of failures of the device.

In the QMR-DT model, the authors distinguish only two types of binary variables: *diseases* and *findings*. These variables are graphically structured in two layers where the *disease* variables influence the *findings*. This structure is based on some independence assumptions. First is the *marginal independence of the diseases*, which amounts to no arcs among the disease variables. Second is the *conditional independence of the findings*, which amounts to no arcs among *finding* variables. Figure 4.1 shows an example of the graphical structure of the QMR-DT model.

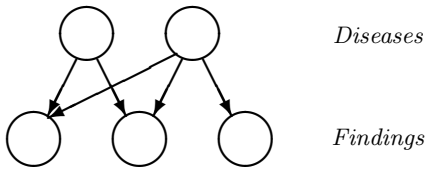


Figure 4.1: The two-level QMR-DT model

The structure of the QMR-DT model was simple, yet the performance of the model was close to that of the original rule-based QMR system. This motivated us to design an extension of this structure to connect our three types of variables in a diagnostic BN model. The authors of the QMR-DT system already mention that assuming a two layered structure is not always very accurate. They state that it would be more accurate to model some of their *findings* representing, for example, historical findings, as parent variables of the *diseases*. We decided to cover this inaccuracy by making our defined *context* variables the parents of the *fault* variables. While keeping

the independence assumptions as in the QMR-DT system it follows that no arcs are allowed among variables within the same layer. This means that the structure becomes a three layered structure with the *context* variables on top, *fault* variables in the middle, and the *evidence* variables at the bottom. We also thought about the possibility of having more layers of variables between the *fault* and *evidence* variables. However, there were two main disadvantages of implementing additional layers in our predefined structure. First, a study of Provan [1995] shows that adding more layers of variables in a BN between the faulty variables and its observations will not improve the diagnostic performance. Second, adding more layers of variables will increase the complexity of building such a model and makes it therefore not useful for the goals of this thesis. An example of the proposed BN3M structure is showed in Figure 4.2. Section 4.2 discusses the use of the Noisy-MAX gates.

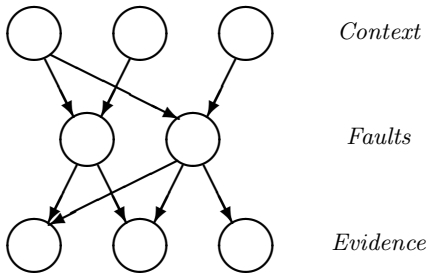


Figure 4.2: The structure of the BN3M model

As mentioned in the introduction, this structure sacrifices some modeling power and precision. For example, the arc between variables A and E of the example given in Figure 2.1 of Chapter 2 can not be created in our methodology. However, a model that is theoretically very precise may turn out to be inferior in practice because elicitation of a huge number of parameters from a human expert may decrease their quality. This, because of tiring out the expert.

4.2 Quantitative design

Next to the graphical extension of the QMR-DT to three layers of variables, another extension is that our model supports multi-valued variables instead of binary variables used in QMR-DT. In order to approximate the interaction of these multi-valued variables using the canonical interaction models

as described in Section 2.3, all interaction among the variables are approximated using (leaky) Noisy-MAX gates. The main reason for doing this is to decrease the number of parameters that have to be elicited by the model builder. Remember that using Noisy gates instead of straightforward CPTs will decrease the growth of the parameters of a variable in the number of parents from exponential to linear (Section 2.3.1). Second, Noisy-MAX parameters are easy to understand even for inexperienced BN model builders. Section 2.3.1 state some example questions which can be answered by a domain expert in order to acquire the parameters of the model.

In the proposed methodology we set the leak probability distribution of an effect variable $p_0(Y)$ by default to 0 for the non-distinguished states $y_i, \ldots, y_{n-1}$ and 1 for the distinguished state $\overline{y}$:

$$p_0(Y) = \begin{cases} 1 & \text{if } Y = \overline{y} \\ 0 & \text{otherwise} \end{cases} . \tag{4.1}$$

In this way the interaction among variables is by default Noisy-MAX and it becomes leaky Noisy-MAX if the model builder defines a specific leak probability distribution for the effect variable.

The disadvantage of using only Noisy-MAX gates for all the interactions of the model is that not all interactions are appropriate to be approximated by this gate. Take for example the binary variable *Sex* with the states: *female*, and *male*. Assigning the distinguished state for this variable is in almost all cases impossible. In this case, approximating it with a Noisy gate is theoretically inaccurate, here it would be better to use a normal CPT or other more specialized canonical gates.

4.3 Theoretical evaluation and methodology

Before describing the methodology of how the predefined BN3M models are build, let us summarize the motivation of choosing this model by giving a theoretical evaluation of the BN3M model:

1. The three layer structure is simple, and the expectation is that the diagnostic performance will be reasonable, this, since a quite similar two layer network also performs reasonably well [Middleton *et al.*, 1991].

2. Using only three types of variables keeps it easy for the user to identify which variables belong to what type. Adding more layers of variables, and therefore, more types of variables, will increase the complexity

of building such a model, and second, a study shows that it will not increase the diagnostic performance of the final model [Provan, 1995].

3. Using Noisy-MAX gates as a canonical interaction model for all interactions among the variables, will decrease the growth of the number of parameters that have to be elicited from exponential to linear in the number of parents. Thus, using the Noisy-MAX gate will speed up the time to build a model since less parameters have to be elicited. Although not all interactions are appropriate to be approximated with the Noisy-MAX gate, different studies show that the Noisy-MAX gate can be used frequently and will not decrease the diagnostic performance of a model [Zagorecki and Druzdzel, 2005; Oniśko *et al.*, 2001].

Since the predefined BN3M model is simple, the methodology of building such a model only consists of three different steps:

1. Add general information of the model, such as, name of the model, description of the model etc.

2. Identify the types of the different variables within the domain and add them to the model.

3. Add relations between the different layers of variables and elicit the prior probabilities for the variables without any parents, and the Noisy-MAX parameters for the variables with parents.

The three steps of the methodology are implemented in a prototype application which is described in the next chapter.

Chapter 5

GeNIeRate

The ideas presented in Chapter 4 are embedded in an interactive environment for generation of diagnostic BN models that we will call GeNIeRate. This chapter will give a complete description of GeNIeRate. The main goal of GeNIeRate is to support a user in building the simplified diagnostic BN3M models in a fast and intuitive way. In Appendix C a complete step-by-step tutorial/help file of GeNIeRate is given to built a simple diagnostic Bayesian network model. GeNIeRate can also be downloaded from: `http://www.sis.pitt.edu/~genie`. GeNIeRate is a member of the family of software developed at the Decision Systems Laboratory (DSL) of the University of Pittsburgh. This software is developed for the purpose of probabilistic modeling with special extensions for diagnostic inference, such as rank ordering, tests, and case management. The most important software developed at DSL are GeNIe and SMILE. Section 5.1 provides some information about GeNIe and SMILE. Section 5.2 discusses how GeNIeRate is integrated in the process of diagnostic BN model building, also the architecture and the design of the application is given in this section. Section 5.3 gives a description of the Graphical User Interface (GUI) of GeNIeRate.

5.1 SMILE and GeNIe

SMILE (Structural Modeling, Reasoning, and Learning Engine) is a fully platform independent library of C++ classes implementing graphical probabilistic and decision-theoretic models, such as Bayesian networks, influence diagrams, and structural equation models. Its individual classes, defined in the SMILE Application Programmer Interface (API), allow to create, edit, save, and load graphical models, and use them for probabilistic reasoning

and decision making under uncertainty. These classes are accessible from C++ or (as functions) from the C programming language. In order to access the SMILE library from other programming languages some "wrappers" are developed: jSMILE for Java, SMILE.NET for a Microsoft .NET environment and pocketSMILE for the Pocket PC. Additional to the SMILE platform and its wrappers is the development of SmileX, an ActiveX Windows component that allows SMILE to be accessed from any Windows programming environment, including World Wide Web pages [Thijssen, 1999]. This makes SMILE accessible from practically any language on any system. Also, SMILE may be embedded in programs that use graphical probabilistic models as their reasoning engines. Furthermore, models developed in SMILE can be equipped with a GUI that suits the user of the resulting application most.

GeNIe is a versatile and user-friendly development environment for building graphical decision models and is developed at the Decision Systems Laboratory. Its name with its uncommon capitalization originates from the name "Graphical Network Interface". The original interface was designed for SMILE which is described above. GeNIe may be seen as an outer shell to SMILE. GeNIe is implemented in Visual C++ and draws heavily on the Microsoft Foundation Classes. Other applications with a GUI using SMILE are: ImaGeNIe which is a general purpose model building interface and GeNIeRate which is discussed in this thesis. Figure 5.1 shows the structure of all the DSL software.

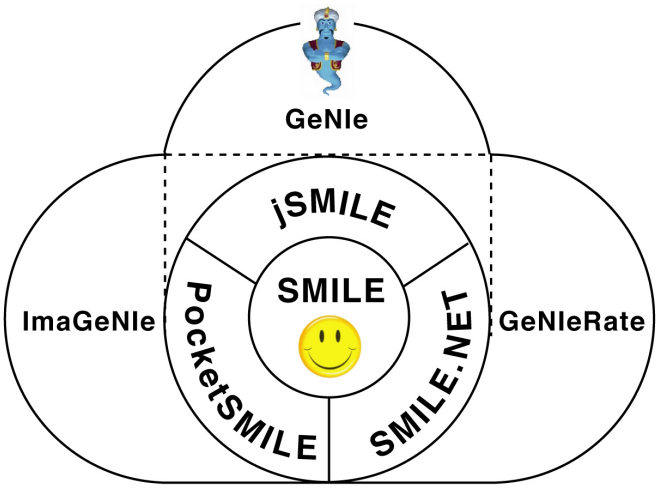


Figure 5.1: The probabilistic modeling software developed at DSL

GeNIe also provides an interface to perform diagnosis. Once a diagnostic model is created with for example GeNIeRate, the model can be loaded in GeNIe and used to apply diagnosis. After setting evidence for some of the *context* and *evidence* variables, GeNIe is able to calculate the most probable *fault(s)* given that evidence. Figure 5.2 shows the GUI of the diagnostic tool of GeNIe applied on the HEPAR-II model.

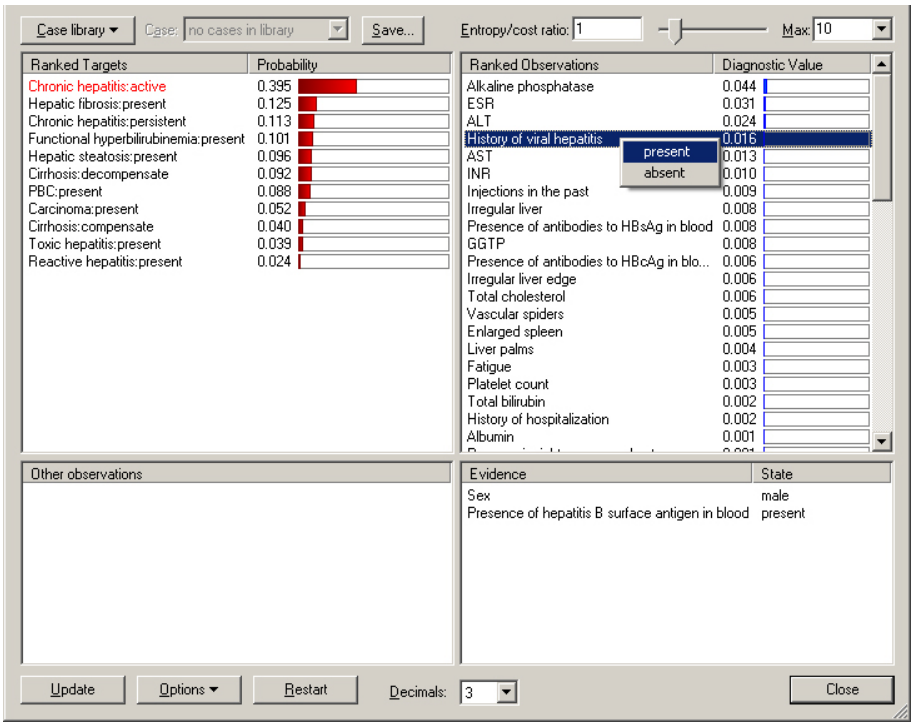


Figure 5.2: Diagnostic tool in GeNIe

Some example applications, built using GeNIe or SMILE, are: battle damage assessment (Rockwell International and U.S. Air Force Rome Laboratory), group decision support models for regional conflict detection (Decision Support Department, U.S. Naval War College), intelligent tutoring systems (Learning and Development Research Center, University of Pittsburgh), medical therapy planning (National University of Singapore), medical diagnosis (Medical Informatics Training Program, University of Pittsburgh and Technical University of Bialystok, Poland). GeNIe and SMILE have also been used in teaching statistics and decision-theory at several universities, including the Technical University of Delft.

5.2 System design

5.2.1 Process

Building and using a diagnostic Bayesian network model consists of three consecutive steps: (1) model building, (2) model refinement, and (3) using the model for diagnosis. GeNIeRate was developed as a model building tool (1). After an initial model has been created, the model can be "fine-tuned" by using for example GeNIe, other BN building software, or user applications using (j)SMILE. At the same time the model can be applied to diagnosis. In Section 5.1, the diagnostic tool of GeNIe is described which can be used as software to apply diagnosis. Next to this tool it is also possible to write software using (j)SMILE to perform diagnosis. Figure 5.3 shows this process of model building and using the model.

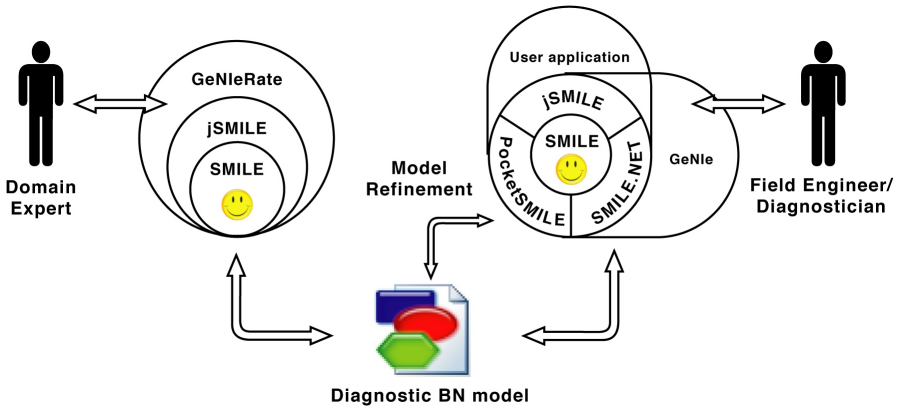


Figure 5.3: Model building with GeNIeRate and using the model with other DSL software

5.2.2 Application architecture

GeNIeRate is developed in Java, therefore it uses jSMILE as its modeling library. GeNIeRate is built upon a Model-View-Controller (MVC) architecture. This architecture applied in GeNIeRate works as follows:

- The *Model* stores the Bayesian Network structure and parameters by interacting with the jSMILE libraries.
- The *View* provides the Graphical User Interface (GUI) of the application representing the different steps of building a diagnostic BN model.

- The *Controller* part provides the control of the application. Dispatching user requests from the view to the model and communicating the responses from the model to the view.

Using the MVC architecture has the advantage of having a separate environment for storing the data (the *Model*) and presenting it to the user (the *View*). Changing the GUI of the application while maintaining the same functionalities becomes easy using this architecture. Furthermore, the architecture of MVC is very clear and easy to understand. Figure 5.4 shows how the MVC architecture is used in GeNIeRate.

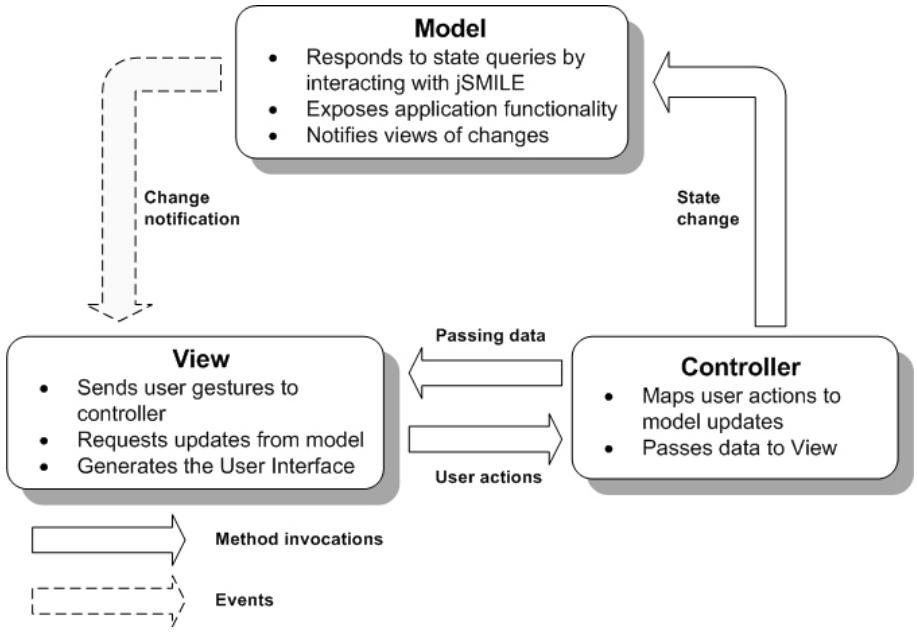


Figure 5.4: Model-View-Controller architecture of GeNIeRate

When looking in more detail, GeNIeRate consists of 65 Java classes. The *Model* is implemented in one big class. The main function of this class is storing the BN structure and parameters by calling the jSMILE functions and passing the returned values to the controllers. Since only one class maintains the data for the BN model being developed by the user, consistency is secured. Next to this large class there are 7 *Controller* classes, 49 *View* classes, and 8 supporting classes. As stated above, the *View* classes represent the GUI of GeNIeRate. This GUI will be discussed in the next section. Figure 5.5 shows a small part of the class diagram of GeNIeRate.

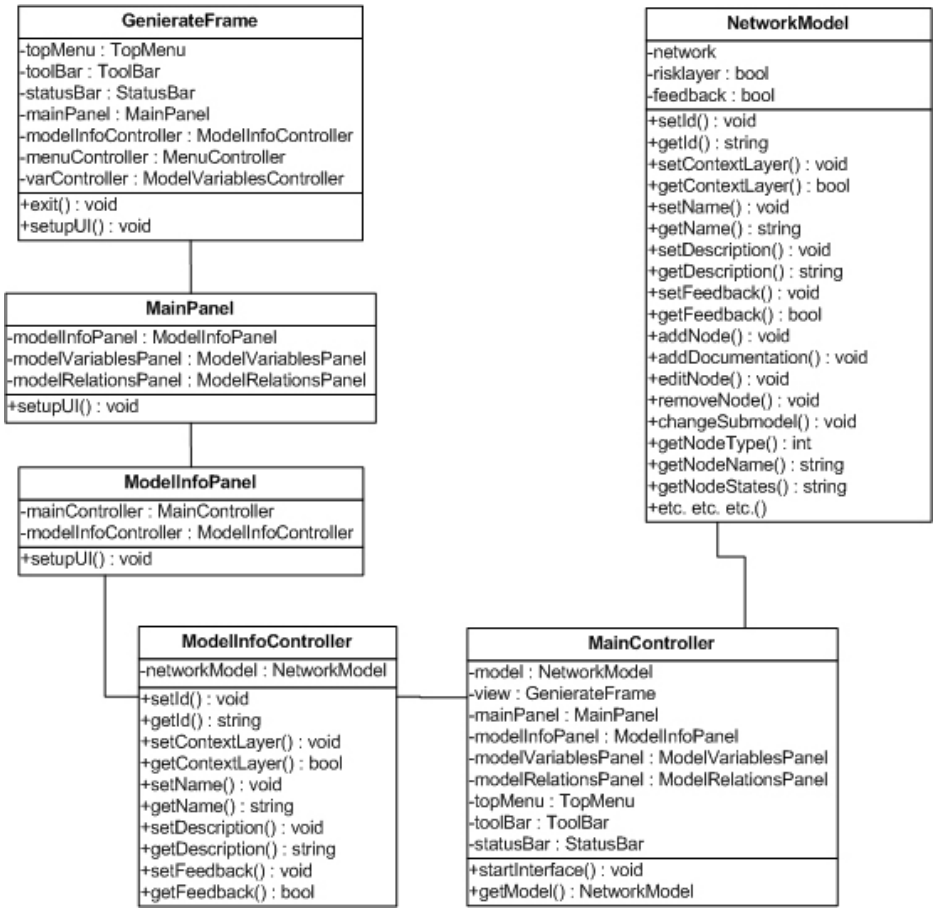


Figure 5.5: A part of the class-diagram of GeNIeRate

5.3 Graphical User Interface

This Section discusses the Graphical User Interface (GUI) of GeNIeRate. Section 5.3.1 describes the design process of the GUI and in Section 5.3.2 a complete description of the final GUI of GeNIeRate is given.

5.3.1 GUI design

When designing the GUI of GeNIeRate Microsoft Visio was useful to support making various designs of possible GUI's. The descriptions of the applications given in the papers presented in Chapter 3 helped in creating different designs. After evaluating the BATS Author software ([Skaanning, 2000],

Chapter 3) with all the members of DSL we decided that our application must provide some visual feed-back presenting the causality of the model. This, by showing the graph or a part of the graph to the model builder. We thought that this would help even an inexperienced BN model builder to understand the diagnostic problem as a causal probabilistic model. After this evaluation, we designed three different interfaces. Again we evaluated with all the members of DSL these interfaces and concluded that one of them was the most promising. This interface is implemented as the final GUI of GeNIeRate and presented in the next section.

5.3.2 GeNIeRate's GUI

In order to keep the model building process simple GeNIeRate contains three different screens, representing three steps of the methodology for building a BN3M model: (1) add general information, (2) add the variables, and (3) add the relations and the probabilities. For every step GeNIeRate has a different screen. This section gives a step-by-step explanation of the GUI. Appendix B provides some full screen shots of GeNIeRate.

General information

In the first tab, the user can define general model information: name, and description of the model. Sometimes a model will not have any *context* variables but only consist of *fault* and *evidence* variables. In this case a two

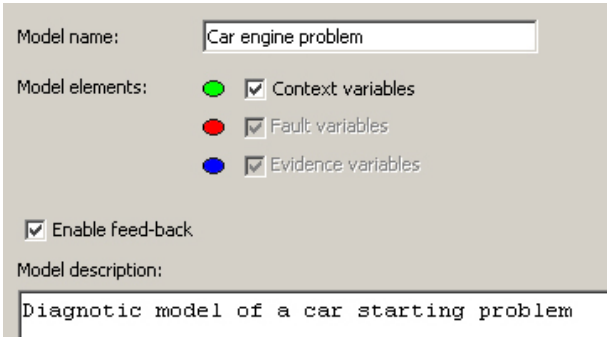


Figure 5.6: Part of the model info panel

level network will be created (BN2M) representing the *fault* and *evidence* variables. For this reason, the *context* variables can be switched on or off in this screen. If the *context* variables are switched off, these variables can not be added in the second screen and no relations can be created from

context to *fault* variables in the third screen. This, to secure the consistency of the model. Furthermore, in this first screen, feedback can be enabled. This feedback amounts to natural language feedback given by GeNIeRate in a dialog for each action of the user. If the user is not a BN expert, this natural language feedback will help to get familiar with this technique and give the user some insight about the causality of the graphical structure of the model. Figure 5.6 shows a part of this first tabbed screen.

Adding variables

In the second tab, the variables of the model can be added, edited, or deleted. For each type of variable (*context*, *fault* and *evidence*) a tree structure represents the variables of that type added to the model. Figure 5.7 shows a part of this screen containing the different tree structures.

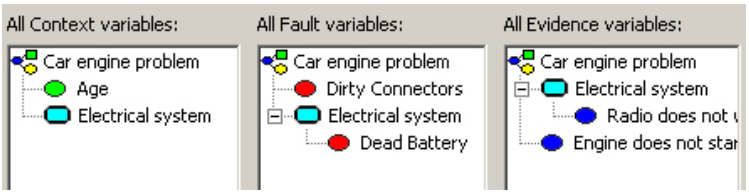


Figure 5.7: Part of the model variables panel

When adding a variable the user can define if the variable is a *mandatory* variable for diagnosis. This means that it is mandatory to give information about the value of that variable when applying final diagnosis with the model. States of the variable can be added or removed. One state can be set as the default state which means that the variable is initially set in that state when performing diagnosis. Furthermore, documentation can be added to the variables and the states of the variables. This documentation can be a treatment for each non-distinguished (Section 2.3.1) state of a *fault* variable or a question for each *context* or *evidence* variable. The documented questions and treatments can later be used when the model is applied in diagnostic software. Answers to these questions can be used as evidence for belief updating (Chapter 2) to calculate the most probable *fault(s)*. After this calculation, the documented treatment of that fault can be presented to help the user make a decision how to fix his problem. Figure 5.8 shows the screen of adding a variable.

GeNIeRate uses the concept of a *system* to divide large diagnostic models into manageable parts. A system is typically a module of a device that, while

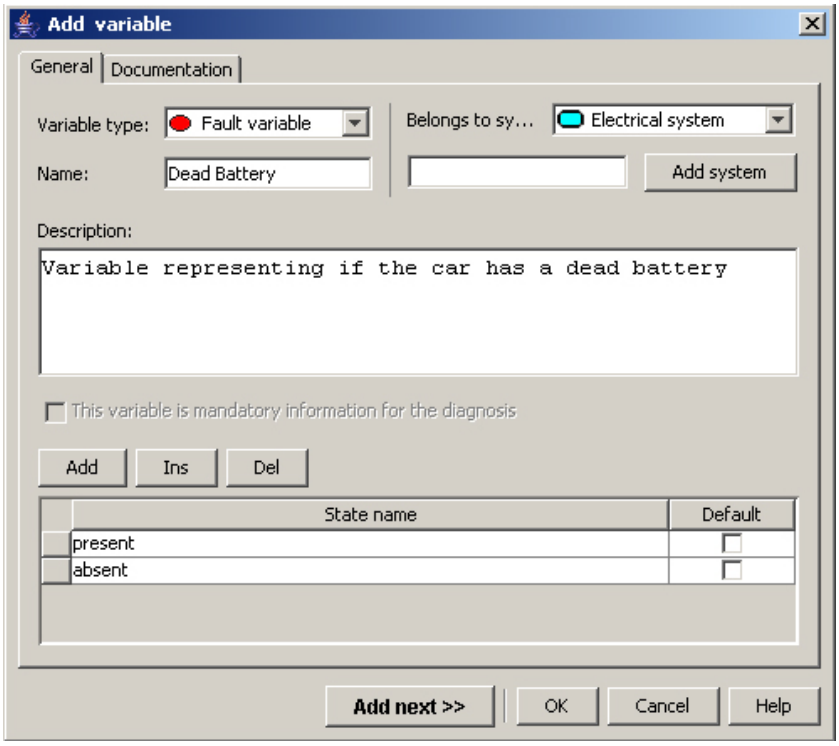


Figure 5.8: Adding a variable

interconnected with other modules, can be thought of in separation from the rest of the device. Typically devices, whether natural or artificial, are composed hierarchically and have clearly identifiable modules, for example electrical system, power train, break, or steering system in an automobile [Simon, 1996]. Systems are the largest entities through which the user can

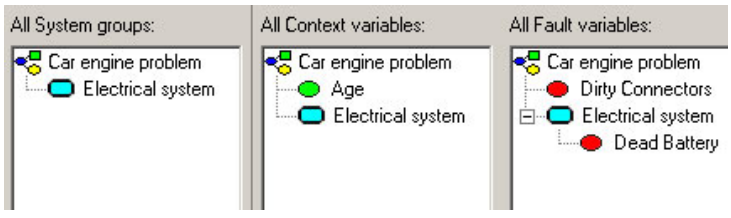


Figure 5.9: Systems in the tree structures

interact with a model, i.e., the users can view a system's variables after selecting that system and add variables to it or remove variables from it.

This supports working on a specific part of the model, ignoring the rest of the model and, since humans can only focus on a small number of things at a time, limiting the amount of information presented to the user [Miller, 1956]. Figure 5.9 shows how GeNIeRate presents the systems in the tree structures.

Adding relations

The last screen supports creating relations among the variables and defining the Noisy-MAX parameters for the interactions. When in this screen, for example, a *fault* variable is selected, *context* variables can be added as parents or *evidence* variables as children of that *fault*. After a relation has been created, the right side of the screen will present the relevant part of the model as a graphical structure (see Figure 5.10).

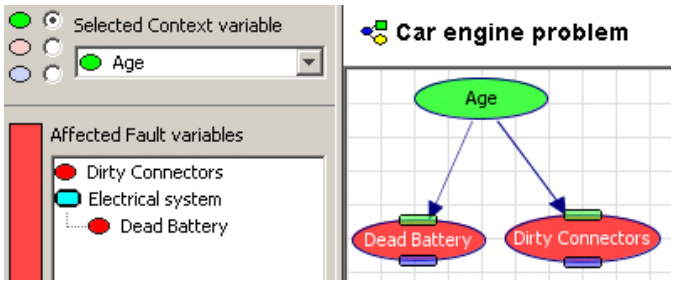


Figure 5.10: Part of the model relations panel

A *system* can be selected to show only the relations of the selected variable with the variables within that system. Navigating the graph can be done by clicking on the icon on top or the bottom of a node, to show the parents or children of that node respectively.

All the parameters can be defined by double-clicking on a node or an arc in the graph. If a node has parents, the leak-probability distribution for that variable can be defined, otherwise the prior probability distribution can be defined. If an arc in the graph is selected, the Noisy-MAX parameters for the relation represented by that arc can be defined (see Figure 5.11).

Thickness of arcs

The thickness of an arc indicates the strength of influence of the relation. This means that if one of the parent states has a big influence of causing a faulty state of the child variable, the arc has to be thick. This property will

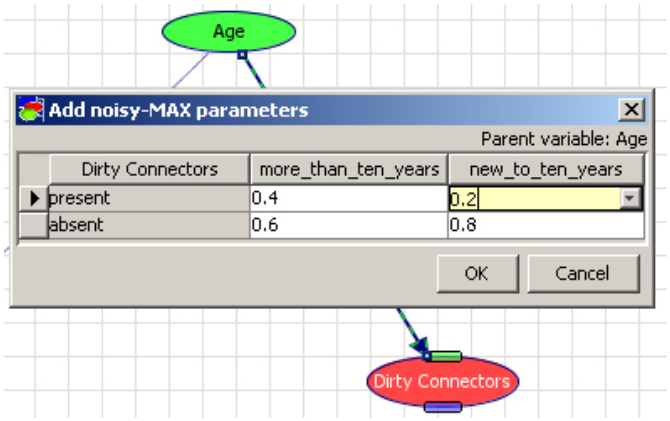
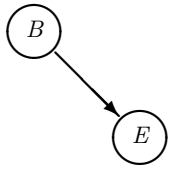


Figure 5.11: Adding Noisy-MAX parameters

make eliciting the Noisy-MAX parameters easier. Human beings are better at comparing quantities relative to each other (x is bigger/smaller than y) than at eliciting them in an absolute way.

To calculate the thickness of the arcs, we first looked at the distance of two probability distributions. The idea was that this measure could say something about the strength of influence between two variables. For example, if we have the two variables *dead Battery (B)* and *Engine does not start (E)* (Chapter 2), where having a dead battery will influence the chance of the engine failing to start, the CPT of the variable E representing the Noisy-MAX parameters will look like Figure 5.12. Now, to calculate the distance



Parent State	Dead Battery: present	Dead Battery: absent	LEAK
present	0.95	0	0
absent	0.05	1	1

Figure 5.12: CPT of variable Engine does not start

among these two variables we need to calculate the distance of the probability distribution of the non-distinguished state(s) of the parent variable and the leak probability distribution of the child variable. In this case, [0.95,

0.05] and [0, 1]. When the parent variable has more than two states, the maximum of all the distances would represent the strength of influence of the arc.

When P and Q represent two probability distributions, and p_i and q_i represent items/states in those distributions respectively, and D represents the distance between the two probability distributions, D can be calculated in various ways:

1. Euclidian distance
$$D = \sqrt{\sum_i (p_i - q_i)^2} \, . \tag{5.1}$$

2. Kullback-Leibler distance
$$D = \sum_i (p_i log_2(\frac{p_i}{q_i})) \, . \tag{5.2}$$

3. Hellinger distance
$$D = \sqrt{\sum_i (\sqrt{p_i} - \sqrt{q_i})^2} \, . \tag{5.3}$$

Although the distance among the two distributions intuitively represents the strength of influence, there were some disadvantages of using one of these measures. The main disadvantage was that all the measures represent fairly complicated solutions to a simple problem, we thought that this could be done in a much easier way. Next to this main disadvantage, all the different measures had their own specific advantages and disadvantages for different situations. We aimed at one measure which could be used in all cases.

After evaluating the different ways to calculate the distances among two probability distributions we found a much easier way to represent the "strength of influence" of an arc. To calculate the strength of influence, we decided to use the cumulative distribution function (CDF). Formally, a CDF is given by:
$$F_x(x) = \Pr[X \leq x] \; \forall x \, , \tag{5.4}$$
where X is a discrete random variable and x is a state of X. The area below this CDF also represents a "distance" among the distribution and an imaginary deterministic distribution consisting of all zeros and a one. This distance becomes bigger when the first states of the variable, representing the faulty non-distinguished states, become larger. The mathematical notation is as follows:
$$D = \sum_x \frac{1}{n} \Pr[X \leq x_i] \, . \tag{5.5}$$

However, when taking the full area below the CDF the last column will some sort of normalize the distance since it always represents a relative big amount of the arced area. For example, the probability distribution of the engine failing to start given that there is a dead battery can be represented by the CDF given in Figure 5.13. As can be seen the last column represents

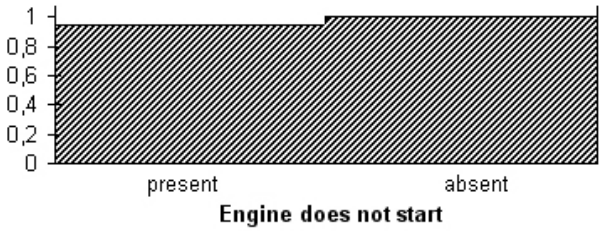


Figure 5.13: The CDF of the variable E given B, the last column "normalizes" the total arced area

a relative large area of the arced part. Therefore, when adjusting the Noisy-MAX parameters the change in line thickness will not be big enough to see any difference.

A fairly simple solution to this was to neglect the area below the last column. In this way only the area below the non-distinguished states is calculated, this measure does react better on changes in the Noisy-MAX parameters and represents exactly what we wanted to achieve. Mathematically Equation 5.5 is changed in:

$$D = \sum_{i=0}^{n-1} \frac{1}{n-1} \Pr[X \leq x_i] \, . \tag{5.6}$$

In Figure 5.14 the same CDF as in Figure 5.13 is shown but now only the area below the non-distinguished state is arced. If the parent variable has more than two states the maximum value of all the columns of the Noisy-MAX parameter table will be used for the thickness of the arc. As can be seen, having a dead battery will have a strong effect on the engine failing to start so the arced area will be large and that will result in a thick arc (Figure 5.15).

The implementation of calculating the line thickness was based on Equation 5.6 and shown in Figure 5.16.

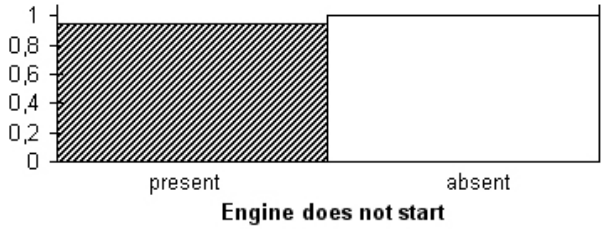


Figure 5.14: The thickness of the arc is represented by the arced area below the non-distinguished state of the CDF

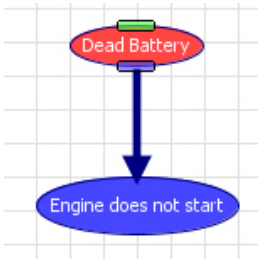


Figure 5.15: GeNIeRate shows a thick line when the influence of the parent is strong

Model consistency

The consistency of the model has to be secured at all times during the model building process. Due to the design of the part of the interface where the relations can be added and deleted, the structure of the model will always be a BN3M model. This, since there are only fixed ways to add relations to variables of different types. For example, only *fault* variables can be added as children of a *context* variable.

Keeping the Noisy-MAX parameters consistent was a little bit more complicated. At all times all columns of a CPT should add up to one, since it represents a probability distribution. When, for example, the user decides to add or remove states of a variable which already is connected to other variables, the columns of the CPT will be changed. Therefore, if something like this happens, the CPT of the relevant variables will be reset, and the user will be warned by the application. Figure 5.17 shows an example of the warning of GeNIeRate. Furthermore, when eliciting probabilities of a certain relation and the probabilities do not count up to one GeNIeRate will also warn the user to change the probabilities (Figure 5.18).

```
public double calculateLineThickness(String childId, String parentId)
{
This function calculates the thickness of the arc between the variable
parentId and the variable childId
        double[] def = getNodeDefinition(childId);
        int startingPos = getParentStartingPosition(childId, parentId);
        int childStates = getStateCount(childId);
        int parentStates = getStateCount(parentId);

        double tmp = 0.0;
        double result = 0.0;
        for(int i=startingPos; i<startingPos+(parentStates*childStates)-childStates; i++)
        {
        Calculate for each column the area below the CDF and remember the maximum
                if(i%childStates==childStates-1)
                {
                        if(tmp>result)
                                result = tmp;
                        tmp = 0.0;
                        continue;
                }
                tmp += ((double)1/(childStates-1))*def[i];
        }
        if(tmp>result)
                result = tmp;
        return result;
}
```

Figure 5.16: Implementation of the thickness of an arc

Figure 5.17: Warning given by GeNIeRate when the number of states of a variable is changed

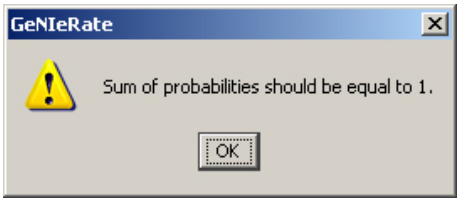


Figure 5.18: Warning given by GeNIeRate when a column in a CPT does not count up to one

Chapter 6

Empirical evaluation

This chapter describes the empirical evaluation of the predefined BN3M model, the methodology and GeNIeRate. Section 6.1 discusses a qualitative evaluation performed by the members of DSL and a professional consultant in building diagnostic BN models. Section 6.2 describes an experiment testing the diagnostic performance of the structure of the BN3M model.

6.1 Qualitative evaluation

6.1.1 DSL members

A first qualitative evaluation of the methodology and GeNIeRate was performed by the members of DSL. They were asked to perform the task of building a diagnostic BN model with GeNIeRate and with GeNIe. After this, they had to answer the following questions:

1. How much time did it take to create your BN with GeNIe?
2. How much time did it take to create your BN with GeNIeRate?
3. On a scale from one to ten how would you value the easiness of creating your network with GeNIe?
4. On a scale from one to ten how would you value the easiness of creating your network with GeNIeRate?
5. On a scale from one to ten how intuitive was it to create your network in GeNIe?

6. On a scale from one to ten how intuitive was it to create your network in GeNIeRate?

7. If you have to build a very large diagnostic model for diagnosis do you think you will use GeNIe or GeNIeRate?

8. List tasks you would run GeNIeRate for, you may propose a task which is not yet there.

Table 6.1 presents a summary of the answers of questions 1 to 6. All members say they will use GeNIeRate for building a large diagnostic BN model (question 7). However, they state they will use other BN model building software to "fine-tune" the model after an initial model has been created. The answers for question 8 all came down to using GeNIeRate as an initial model building tool and after that switch to other BN model building software, like GeNIe, to "fine-tune" the model. It must be said that all members of DSL are experienced BN model builders. Therefore, and since only four members answered the questions, this experiment does not give a very reliable representation of how easy it is to build a model using GeNIeRate.

Table 6.1: Scores for the first 7 questions

	DSL members				
Questions	1	2	3	4	Average
Question 1	n/a	4 min.	3 min.	n/a	3.5 min.
Question 2	n/a	3 min.	5 min.	n/a	4 min.
Question 3	5	7	8	n/a	6.7
Question 4	7	8	8	8.5	7.9
Question 5	5	7	8	n/a	6.7
Question 6	7	8	9	n/a	8

6.1.2 Professional consultant

GeNIeRate was also evaluated by Mr. M. D. Campbell, a professional consultant with 5 years of practical experience in building diagnostic BN models. GeNIeRate was received warmly and Mr. Campbell estimated that this methodology will be of great help especially in building large diagnostic models. He mentioned that a domain expert usually spends 90% of the time in analyzing and understanding his diagnostic problem as a probabilistic

model. However, his opinion was that our approach will reduce the model building time for an inexperienced Bayesian network model builder. He estimated that this model building time could be reduced by 20-30%.

6.2 Quantitative evaluation

Next to the qualitative evaluation we also performed an empirical study evaluating the diagnostic performance of the BN3M structure. The goal of this experiment was to see the influence of the simplifying assumptions made in the BN3M model on the final diagnostic performance of the model. In order to do this, we rebuilt an existing diagnostic BN model using GeNIeRate. After rebuilding, we compared the diagnostic performance of the BN3M model against the original model. The existing model that we rebuilt is the HEPAR-II model [Oniśko *et al.*, 2001]. This section gives a detailed description of this experiment and its results.

6.2.1 The Hepar-II model

"Hepatology is the branch of medicine that is concerned with disorders of the liver, gall bladder and biliary ducts. Although occasionally practiced by specialized hepatologists, it is most often considered the terrain of gastroenterology".[1]

An early statistical approach to automate diagnosis in hepatology is the HEPAR system [Bobrowski, 1992] which is a computer-based tool designed for gathering and processing the clinical data of patients with liver disorders and it aimed at reducing the need for hepatic biopsy. The HEPAR system was integrated with a medical database and it consisted of various numerical tools to process and analyze data. Another example used the rule-based approach to automate diagnosis in hepatology. This rule-based HEPAR [Lucas *et al.*, 1989] aims at assisting clinicians in the appropriate selection of patients who need to be submitted to invasive tests, such as liver biopsy. The system is able to differentiate among nearly 80 disorders of the liver and biliary tract. Later, Bayesian network theory was used to make a probabilistic computer-based diagnostic model of liver disorders [Oniśko *et al.*, 2001], this new probabilistic system was called: the HEPAR-II model and is used in the experiment.

The HEPAR-II model (Figure 6.1) consist of 70 variables of which 10 variables are liver diseases (*faults*), the 60 other variables are test results,

[1] http://en.wikipedia.org/wiki/Hepatology

observations, and risk factors (*context* and *evidence* variables).

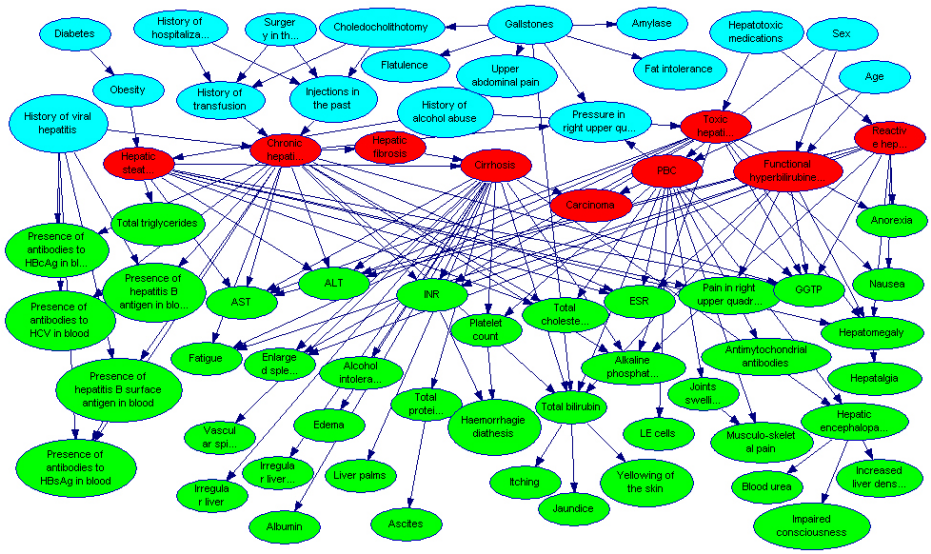


Figure 6.1: The HEPAR-II model

6.2.2 HEPAR-II-BN3M

In collaboration with Agnieszka Oniśko, who developed the HEPAR-II model, we rebuilt the structure of the model from scratch using GeNIeRate. Since Agnieszka had a lot of knowledge about the original HEPAR-II model, she was able to identify which variable belonged to what type, and how these variables should be related in the new BN3M structure. We will refer to this new structure as HEPAR-II-BN3M. It took us approximately 8 hours to build HEPAR-II-BN3M. Building the original HEPAR-II model took 300 hours, but comparing that to the 8 hours we spent is not fair since the original 300 hours spent on building HEPAR-II included learning, interaction with experts and literature study. All this knowledge was already available when rebuilding it in the new BN3M form. We decided to focus especially on the diagnostic performance of the BN3M structure by comparing the performance of the new BN3M model against that of the original model.

When rebuilding the HEPAR-II model, we had to add and remove several arcs since the assumptions of the BN3M model do not allow all interactions among the variables. As can be seen in Figure 6.1 there are several arcs among variables of the same type. We had to decide to remove these arcs or

replace them by direct influences from other variables. Figure 6.2 shows a good example of how we initially converted a part of the original HEPAR-II to the new HEPAR-II-BN3M.

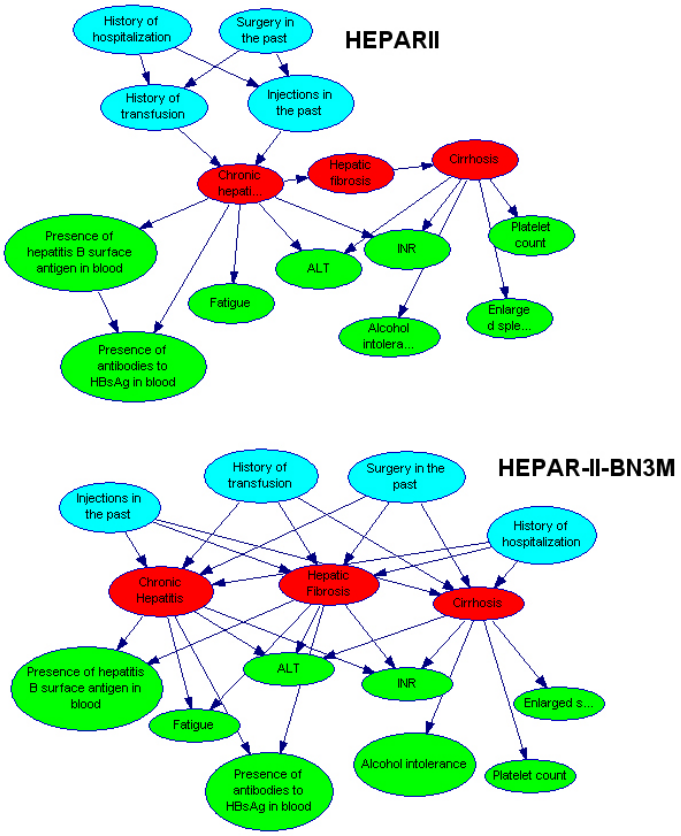


Figure 6.2: Example of some modifications of the original HEPAR-II model (top) and the new HEPAR-II-BN3M model (bottom)

6.2.3 The diagnostic performance

After we rebuilt the structure of the model, we were interested in the diagnostic performance of this new model. The diagnostic performance is whether, after setting some evidence in the model, the most probable diagnosis (*fault*) given by the model is indeed the correct diagnosis. To do this, one would need the parameters to be set in the BN model. Since these parameters were learned from a data set in the original HEPAR-II, we decided

to do this too in our model. To learn the parameters, we used the same data set, created in 1990 and thoroughly maintained since then. This data set contains approximately 700 patient cases and it is still growing.

To get a good idea of the diagnostic performance, we learned the parameters using the "leave-one-out" algorithm. This means that we learned the parameters iteratively 700 times with 699 cases (700 minus one) where in each iteration the case that was not used in the learning algorithm was used for testing. Thus after 700 loops of 700 cases you can see how many times the model states the correct diagnosis as the most probable diagnosis. Next to the most probable diagnosis, we were also interested whether the set of k most probable diagnoses contains the correct diagnosis for small values of k (we chose a "window" of k=1, 2, 3, and 4).

The learning algorithm used to learn the parameters of the original HEPAR-II model was able to learn the CPTs of a model, thus no "Noisy" gates. So, in order to use the same learning algorithm to get a fair comparison, we had to transform the variables of HEPAR-II-BN3M from Noisy-MAX to normal CPTs. This is a trivial operation using GeNIe.

Fortunately, all the algorithms to learn the parameters and to test the diagnostic performance were already available. A description how this learning method works is given in Appendix A and was copied from Section 3.4.2 of [Oniśko, 2002].

6.2.4 Results and Discussion

The results of the first model we build were: 40% (compared to 59% for the original HEPAR-II model), 47% (72%), 55% (79%), and 63% (85%) for a window size of k=1, 2, 3, and 4 respectively, see Table 6.2. Figure 6.3 shows a graphical interpretation of these first results.

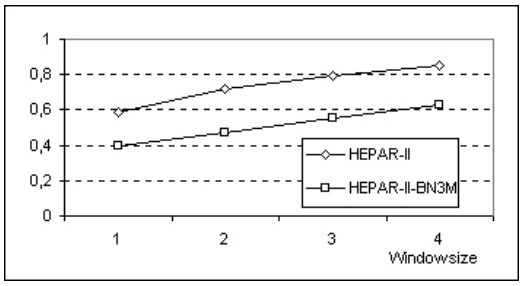


Figure 6.3: Diagnostic accuracy of our first HEPAR-II-BN3M compared to the original HEPAR-II as a function of different window sizes

Table 6.2: Diagnostic accuracy of the initial HEPAR-II-BN3M for all different diseases for four different window sizes

	Window size (k)			
Disease	1	2	3	4
Hepatic steatosis	11.94%	17.91%	23.88%	31.34%
Chronic hepatitis: Active	68.13%	71.43%	73.62%	80.22%
Chronic hepatitis: Persistent	0%	0%	5.56%	27.78%
Hepatic fibrosis	6.25%	6.25%	12.5%	31.25%
Cirrhosis: Compensate	38.71%	70.97%	96.77%	100%
Cirrhosis: Decompensate	0%	32.76%	65.52%	96.55%
Carcinoma	0%	0%	20%	30%
PBC	68.93%	72.5%	73.93%	76.79%
Toxic hepatitis	0%	0%	3.70%	3.70%
Reactive hepatitis	0%	0%	0%	0%
Hyperbilirubinemia	7.14%	8.93%	33.93%	35.71%
Total: HEPAR-II-BN3M (HEPAR-II)	40% (59%)	47% (72%)	55% (79%)	63% (85%)

When looking closely to the table it is clear that these results are not good. Some diseases get a reasonable diagnostic performance for a window size of 4, for example, *Cirrhosis: Compensate* (100%) and *Cirrhosis: Decompensate* (96.55%). However, most of the diseases perform very poorly, one disease even keeps a diagnostic performance of 0% (*Reactive Hepatitis*). We did not expect these bad results (23% off), we expected that due to the simplifying assumptions the diagnostic performance would decrease a little bit (approximately 5% to 10%) but not that much. We discovered that in the CPTs of the new model many columns still had a uniform distribution. This means that those columns were not learned from the data. This could be caused by the fact that our data set of patient cases is too small. In order to get better results we tried to remove some arcs. We used our knowledge of the HEPAR-II model to remove some of the arcs which in our opinion were irrelevant. The result of this was that some variables got fewer parent variables and in that way the size of the CPTs, and thus the number of parameters that had to be learned decreased. After applying this to the model, we obtained its second version, a fragment of which is shown in Figure 6.4.

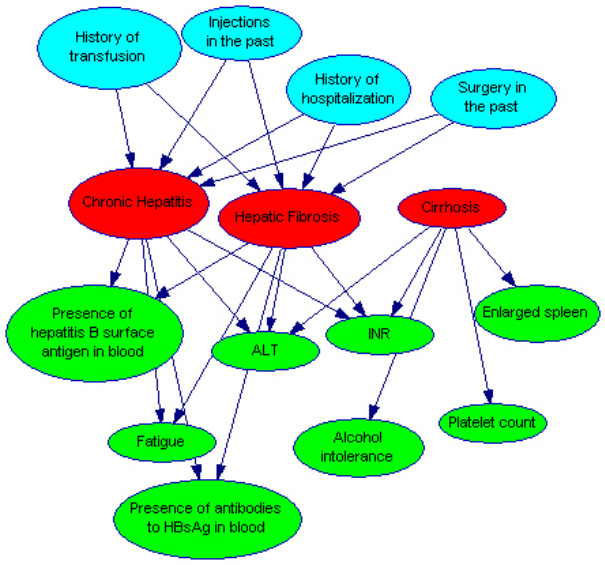


Figure 6.4: Same part of the model in Figure 6.2 after new modifications

The results of this new model were 64% (compared to 59% for the original HEPAR-II model), 75% (72%), 82% (79%), and 87% (85%) for a window size of k=1, 2, 3, and 4 respectively, see Table 6.3. Figure 6.5 shows these results graphically. As can be seen in the table, the difference of the

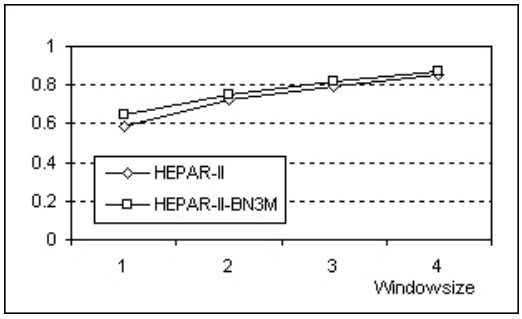


Figure 6.5: Diagnostic accuracy of the modified HEPAR-II-BN3M compared to the original HEPAR-II as a function of different window sizes

diagnostic accuracy between the diseases is in some cases big. When the window size is 1, the best diagnostic performance is achieved by *PBC* with 98.92%, compared to the worst disease *Reactive hepatitis* with 0%. This bad result can be explained by the amount of data that is available for that

Table 6.3: Diagnostic accuracy of the modified HEPAR-II-BN3M for all different diseases for four different window sizes

	Window size			
Disease	1	2	3	4
Hepatic steatosis	31.34%	55.22%	73.13%	86.57%
Chronic hepatitis: Active	50.55%	82.42%	93.41%	94.51%
Chronic hepatitis: Persistent	28.57%	48.57%	62.85%	82.85%
Hepatic fibrosis	6.25%	12.5%	43.75%	50%
Cirrhosis: Compensate	68.97%	75.86%	82.75%	87.93%
Cirrhosis: Decompensate	19.35%	41.94%	48.38%	54.83%
Carcinoma	50%	60%	70%	70%
PBC	98.92%	100%	100%	100%
Toxic hepatitis	11.11%	18.51%	33.33%	55.56%
Reactive hepatitis	0%	0%	11.76%	23.53%
Hyperbilirubinemia	64.29%	69.64%	76.79%	82.14%
Total: HEPAR-II-BN3M (HEPAR-II)	64% (59%)	75% (72%)	82% (79%)	87% (85%)

specific disease. In the data set of 700 patient cases, only 16 were diagnosed with *Reactive hepatitis*, compared to 279 patients having *PBC*. If more data would be available for every specific disease, the diagnostic performance of all the diseases could increase.

However, surprisingly, the results of the new HEPAR-II-BN3M model show, on the average, a better diagnostic performance for every window size than the original HEPAR-II model (2-5%). The most plausible explanation of this surprising result is that the original HEPAR-II model contained erroneous conditional independence assumptions. If, for example, in the situation of Figure 6.6 we have information of all the *context* and *evidence* variables: *History of hospitalization*, *History of transfusion*, *Total bilirubin* and *Itching*. Now, in the situation of the original HEPAR-II model, information of the variables *History of hospitalization* and *Itching* does not influence the *fault* variable *Chronic hepatitis* since the information about *History of transfusion* and *Total bilirubin* eliminates the other information. In the new HEPAR-II-BN3M model we assume that the variables *History of hospitalization* and *Itching* will influence the chance of having *Chronic hepatitis* directly. The difference of independence assumptions like these can be the cause of the better results we found with the new HEPAR-II-BN3M

model. We expect that results like this will not always be observed in other models. However, we believe that the BN3M model and our methodology supported with GeNIeRate should take some credit for facilitating building a better quality model.

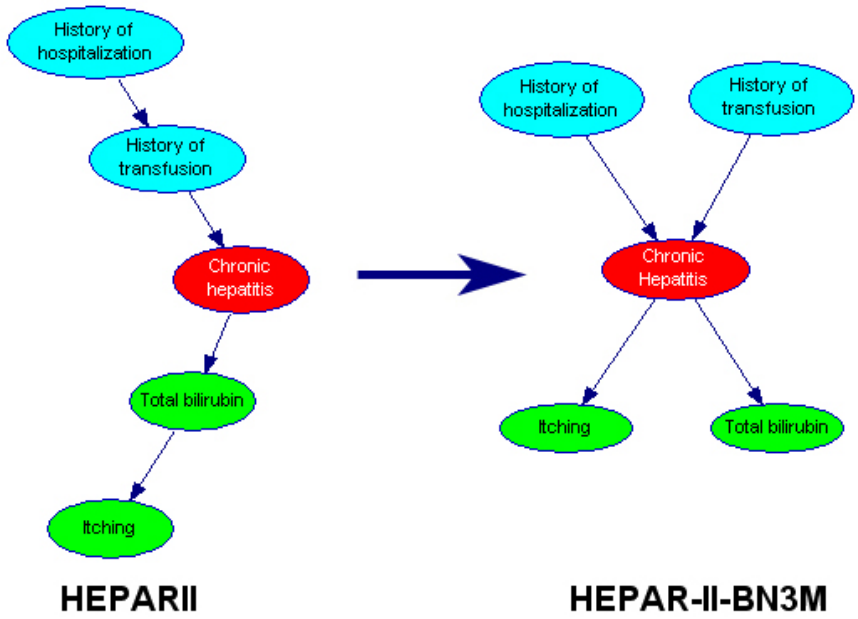


Figure 6.6: Possible erroneous independence assumption in the original HEPAR-II model

Chapter 7

Conclusions and Future Research

7.1 Conclusions

This thesis has presented a methodology implemented in a prototype application for building large diagnostic Bayesian networks. In the introduction of this thesis (Chapter 1) the assignment was given which is repeated below:

1. design a predefined Bayesian network model, based on some simplifying assumptions, that can be used as a "template" model to build very large diagnostic Bayesian networks.
2. design a methodology to build very large diagnostic BN models using the designed "template".
3. implement a prototype application, based on the methodology, to support the user, who is not necessarily a BN expert, to build a very large diagnostic BN model.
4. test the idea's of the "template", the methodology, and the prototype, and test whether the diagnostic performance of the created models suffered, due to the simplifying assumptions it is based on.

The predefined model was presented in Chapter 4 called: the Bayesian Network of three layers of variables using Noisy-MAX gates (BN3M). The BN3M model has two main simplifying assumptions. First, the structural part of the model only consist of three layers of variables representing three different types of variables: *context*, *fault*, and *evidence* variables. Interactions are only possible between two consecutive layers. Second, all the

interactions among the variables are approximated by Noisy-MAX gates to reduce the number of parameters that have to be elicited, and therefore, to reduce the time needed to build a large diagnostic model.

The methodology to build the BN3M models consist of three different steps:

1. Add general information of the model, such as, name of the model, description of the model etc.

2. Identify the types of the different variables within the domain and add them to the model.

3. Add relations between the different layers of variables and elicit the prior probabilities for the variables without any parents, and the Noisy-MAX parameters for the variables with parents.

The methodology is implemented in a prototype application called: GeNIeRate. Chapter 5 discussed a complete description of the design of this application. Next to the implementation of the methodology, GeNIeRate contains different features to make the model building process easier: an intuitive Graphical User Interface, natural language feed-back for the actions of the user, and visual feed-back for the elicitation of probabilities. Furthermore, GeNIeRate assures at all times that the model being build is consistent.

Finally, the ideas of the BN3M model, the methodology, and GeNIeRate were tested in Chapter 6. GeNIeRate was received well by a professional consultant in building diagnostic Bayesian networks. His estimate is that using this methodology for building large diagnostic BN models will reduce the model building time for an inexperienced BN model builder with approximately 20-30%. Next to Mr. Campbell, the members of DSL also state that GeNIeRate will help developing an initial diagnostic BN model for inexperienced BN model builders in an easy way. An empirical study comparing the BN3M structure to a traditional model shows that the diagnostic performance of the model created by traditional model building techniques is close to the performance of the same model created by our methodology. It seems that a model that can be build significantly faster does not necessarily suffer in terms of its diagnostic performance.

7.2 Future Research

For future research the following items must be recommended:

- Extend the experiments with the new HEPAR-II-BN3M model. For example, learn the parameters with the initial Noisy-MAX assumptions and compare the diagnostic performance to the original HEPAR-II model and the HEPAR-II-BN3M with the learned CPTs.
- Perform more experiments to test GeNIeRate by applying it in various domains and by letting inexperienced BN model builders (engineers, medical doctors etc.) built a diagnostic BN model for their own domain. These experiments can be designed such that model building with GeNIeRate is compared with general BN model building software like GeNIe. There are two fundamental ways of doing this: (1) within-subject design and (2) between-subject design. The first means that a person will built a model with GeNIeRate and GeNIe and will compare those two models afterwards on the following features: time, model building process, diagnostic performance. The latter means that there will be a randomization of experts building a model with GeNIeRate and experts building the same model with GeNIe. These models will be compared afterwards on the same features mentioned above. A disadvantage of the between-subject design is that there will be a variance in the level of experience and the expertise of the different experts who build the models.
- Support GeNIeRate with a connection to a database for fast generation of all the variables and to learn the parameters from data if a data set is available.
- Support more specialized gates for the interaction among the variables. Using the Noisy-MAX gate for all the interactions is not always very accurate.

Appendix A

Learning CPT parameters from a data set

Learning CPTs amounts essentially to counting data records for different conditions encoded in the network. Roughly speaking, prior probability distributions are obtained from relative counts of various outcomes for each of the nodes without predecessors. Conditional probability distributions are obtained from relative counts of various outcomes in those data records that fulfill the conditions described by a given combination of the outcomes of the predecessors.

Suppose we have a set of cases: $D = \{d_1, d_2, \ldots, d_n\}$ and a graphical structure of a Bayesian network M. Let X_i be one of the variables of the model M, and Π_i the set of parents of X_i in the graph M. To estimate the conditional probabilities for X_i from D, we apply the following equation:

$$\Theta_{ijk} = \Pr(X_i = x_{ik}|\pi_{ij}) \; . \tag{A.1}$$

Let us indicate $X_i = x_{ik}$ by x_{ik}, where k represents the *k-th* state of X_i. With respect to term π_{ij}, j indicates the *j-th* combination of the parents states of X_i. Then for each frequency the following estimate is computed:

$$\Theta_{ijk} = \frac{n(x_{ik}|\pi_{ij})}{n(\pi_{ij})} \; , \tag{A.2}$$

where: $n(x_{ik}|\pi_{ij})$ is the frequency of (x_{ik}, π_{ij}), and $n(\pi_{ij}) = \sum_k n(x_{ik}|\pi_{ij})$ is the frequency of π_{ij}.

Considering a set of Dirichlet prior parameters α_{ijk}, Equation A.3 takes the form:

$$\Theta_{ijk} = \frac{n(x_{ik}|\pi_{ij}) + \alpha_{ijk}}{n(\pi_{ij}) + \alpha_{ij}} \ , \tag{A.3}$$

where: $\alpha_{ij} = \sum_k \alpha_{ijk}$.

Appendix B

Screen shots of GeNIeRate

Appendix C

GeNIeRate tutorial

Appendix D

Paper version

This appendix contains the paper which was submitted to the "21st Conference on Uncertainty in Artificial Intelligence". Unfortunately, this paper was rejected. Another paper that we wrote was submitted to the "16th International Workshop on Principles of Diagnosis (DX-05)". This paper did get accepted but was actually an older version than the UAI paper, it did not contain the results of the empirical study, since we did not have those experiments yet. Therefore the UAI paper was selected for this appendix.

Bibliography

[Barco *et al.*, 2002] Raquel Barco, Rafael Guerrero, Gustavo Hylander, Lars Nielsen, Martti Partanen, and Sagar Patel. Automated troubleshooting of mobile networks using Bayesian networks. In *Proceedings of the IASTED International Conference on Communications Systems and Networks*, pages 105–110. ACTA Press, 2002.

[Bobrowski, 1992] Leon Bobrowski. HEPAR: Computer system for diagnosis support and data analysis. Prace IBIB 31, Institute of Biocybernetics and Biomedical Engineering, Polish Academy of Sciences, Warsaw, Poland, 1992.

[Chen, 2003] Cathy Chen. Bayesian serviceability tool for diagnosing complex medical imaging machines. *GE Global Research, Technical Information Series*, JAN 2003.

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

[Dagum and Luby, 1997] Paul Dagum and Michael Luby. An optimal approximation algorithm for Bayesian inference. *Artificial Intelligence*, 93:1–27, 1997.

[de Dombal *et al.*, 1972] F.T. de Dombal, D.J. Leaper, J.R. Staniland, A.P. McCann, and Jane C. Horrocks. Computer-aided diagnosis of acute abdominal pain. *British Medical Journal*, 2:9–13, APR 1972.

[de Kleer *et al.*, 1990] J. de Kleer, A. K. Mackworth, and R. Reiter. Characterizing diagnoses. In G. Gottlob and W. Nejdl, editors, *Expert Systems in Engineering: Principles and Applications: Proc. of the International Workshop, Vienna, Austria*, pages 1–15. Springer, Berlin, Heidelberg, 1990.

[Díez, 1993] F. J. Díez. Parameter adjustement in Bayes networks. The generalized noisy OR–gate. In *Proceedings of the 9th Conference on Uncertainty in Artificial Intelligence*, pages 99–105, Washington D.C., 1993. Morgan Kaufmann, San Mateo, CA.

[Friedman *et al.*, 1997] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163, 1997.

[Good, 1961] I. Good. A causal calculus (I). *British Journal of Philosophy of Science*, 11:305–318, 1961.

[Heckerman *et al.*, 1992] David Heckerman, E.J. Horvitz, and B.N. Nathwani. Toward normative expert systems: Part I. The Pathfinder Project. *Methods of Information in Medicine*, 31, 1992.

[Henrion, 1989] M. Henrion. Some practical issues in constructing belief networks. In L. N. Kanal, T. S. Levitt, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 3*, pages 161–173. North-Holland, Amsterdam, 1989.

[Jensen *et al.*, 2001] Finn V. Jensen, Uffe Kjærulff, Brian Kristiansen, Helge Langseth, Claus Skaanning, Jirĺ Vomlel, and Marta Vomlelová. The SACSO methodology for troubleshooting complex systems. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM)*, 2001. To Appear in a Special Issue on AI in Equipment Service.

[Ledley and Lusted, 1959] R. Ledley and L. B. Lusted. Reasoning foundations of medical diagnosis. *Science*, 130:9–21, 1959.

[Lucas *et al.*, 1989] P. J. F. Lucas, R. W. Segaar, and A. R. Janssens. HEPAR: an expert system for diagnosis of disorders of the liver and biliary tract. *Liver*, 9:266–275, 1989.

[Madden and Nolan, 1999] Michael G. Madden and Paul J. Nolan. Monitoring and diagnosis of multiple incipient faults using fault tree induction. In *IEEE proceedings Control Theory and Applications*, 1999.

[Middleton *et al.*, 1991] B. Middleton, M.A. Shwe, D.E. Heckerman, M. Henrion, E.J. Horvitz, H.P. Lehmann, and G.F. Cooper. Probabilistic diagnosis using a reformulation of the INTERNIST–1/QMR knowledge base: II. Evaluation of Diagnostic Performance. *Methods of Information in Medicine*, 30(4):256–267, 1991.

[Milho and Fred, 2000] Isabel Milho and Ana Fred. A user-frienly development tool for medical diagnosis based on Bayesian networks. In *Proceedings of the Second International Conference on Enterprise Information Systems*, pages 176–180, 2000.

[Miller *et al.*,] Randolph A. Miller, Harry E. Pople, Jr., and Jack D. Myers. Internist–1, an experimental computer-based diagnostic consultant for general internal medicine. *New England Journal of Medicine*, 307, NUMBER =.

[Miller, 1956] George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81–97, 1956.

[Myers, 1987] Jerome L. Myers. The background of Internist-I and QMR. In *Proceedings of ACM conference on history of medical informatics*, pages 195–197. ACM Press, 1987.

[Oniśko *et al.*, 2001] Agnieszka Oniśko, Marek J. Druzdzel, and Hanna Wasyluk. Learning Bayesian network parameters from small data sets: Application of Noisy-OR gates. *International Journal of Approximate Reasoning*, 27(2):165–182, 2001.

[Oniśko, 2002] Agnieszka Oniśko. *Probabilistic Causal Models in Medicine: Application to Diagnosis of Liver Disorders*. PhD thesis, Department of Computer Science, Bialystok University of Technology, Wiejska 45-A, 15-351, Bialystok, Poland, FEB 2002.

[Pearl, 1986] Judea Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29(3):241–288, 1986.

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

[Provan, 1995] Gregory Provan. Abstraction in belief networks: The role of intermediate states in diagnostic reasoning. In *Proceedings of the 11th Annual Conference on Uncertainty in Artificial Intelligence (UAI-95)*, pages 464–471, San Francisco, CA, 1995. Morgan Kaufmann Publishers.

[Quinlan, 1986] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Schroder *et al.*, 1996] Olaf Schroder, Claus Mobus, Jorg Folckers, and Heinz-Jurgen Thole. Supporting the construction of explanation models and diagnostic reasoning in probabilistic domains. In *Proceedings of the ICLS 96*, pages 60–67, Northwestern University, Evanston, IL, USA, 1996. Charlottesville, VA: Association for the Advancement of Computing in Education (AACE).

[Shwe *et al.*, 1991] M.A. Shwe, B. Middleton, D.E. Heckerman, M. Henrion, E.J. Horvitz, H.P. Lehmann, and G.F. Cooper. Probabilistic diagnosis using a reformulation of the INTERNIST–1/QMR knowledge base: I. The probabilistic model and inference algorithms. *Methods of Information in Medicine*, 30(4):241–255, 1991.

[Simon, 1996] Herbert A. Simon. *The Sciences of the Artificial - 3rd Edition.* The MIT Press, October 1996.

[Skaanning, 2000] Claus Skaanning. A knowledge acquisition tool for Bayesian-network troubleshooters. In *Uncertainty in Artificial Intelligence: Proceedings of the Sixteenth Conference (UAI-2000)*, pages 549–557, San Francisco, CA, 2000. Morgan Kaufmann Publishers.

[Stensmo and Sejnowski, 1994] Magnus Stensmo and Terrence Sejnowski. A mixture model diagnosis system, 1994.

[Thijssen, 1999] C. R. T. Thijssen. SmileX: An ActiveX decision-analytic reasoning engine and its application to evaluation of credit applicants. Master's thesis, Delft University of Technology, 1999.

[Zagorecki and Druzdzel, 2004] Adam Zagorecki and Marek Druzdzel. An empirical study of probability elicitation under Noisy-OR assumption. In *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Symposium (FLAIRS) Conference, Miami Beach, Florida, USA*, pages 880–885. AAAI Press, 2004.

[Zagorecki and Druzdzel, 2005] Adam Zagorecki and Marek Druzdzel. Knowledge engineering for building Bayesian networks: How common are Noisy-MAX distributions in practice?, 2005.