Abstract

Bayesian networks are a successful modeling tool that have become very popular in the last 20 years. Software for constructing models is widely available, but software that combines data and expert knowledge in a principled way to construct networks is rare. We aim on solving this problem by extending GeNIe and SMILE to give the users the possibility to use this feature. To accomplish this we divided the problem into three different parts.

Firstly, we created a case management system in both GeNIe and SMILE that manages the data of the user, we will call this cases. Managing cases involves editing evidence and target nodes in the network, and more. We take a novel approach of storing the cases in the same file the network is stored (usually data is stored in separate files), to easy the user and to keep the cases consistent with the network.

Secondly, we will use these cases to refine a network created by an expert by applying the Expectation-Maximization (EM) learning algorithm. EM is capable of combining expert knowledge and data by using prior probabilities and the possiblility to express how confident the expert is in the priors. Since experts might be more confident in some parts of the network than others, we introduce the concept of hierarchical confidence that experts can use to express how important the data is compared to their constructed networks, on different levels of granularity, e.g., on the network level or CPT level. High confidence will not change the local probability distributions much, but if the confidence is low, the data will change the local probability distributions a lot. Instead of learning complete networks from data, we chose refinement, because usually only limited data is available. And if you realize that the number of parameters that has to be specified in a CPT exponential is in the number of parents, we have to conclude that learning complete networks is only feasible when a lot of data is available.

This leads to the third part of the work that has been done. We will introduce new canonical gates that, based on some assumptions, require a lower number of parameters to be specified and, consequently, less data is needed to learn these gates than was needed in case of CPTs. Another advantage of these gates is that they have the property that inference algorithms can exploit them. Since learning requires a lot of inference, these gates are also suitable for learning from very large datasets.

Acknowledgements

This thesis is the result of my graduation project at the University of Pittsburgh, from November 2004 to June 2005, and a few weeks of writing afterwards. It was a very educational experience and I am proud of the results. The quality of my thesis has been greatly increased under the influence of the following people, to whom I am very thankful:

- Marek Druzdzel. He was my supervisor at the University of Pittsburgh and has his own lab, the Decision Systems Laboratory (DSL). My stay at DSL was pleasant and Marek was always a source of good ideas. Our weekly meetings were always very useful for me and helped me more than once to solve a problem that I encountered.

- Leon Rothkrantz. He was my supervisor at the Delft University of Technology and proposed the idea of going to the University of Pittsburgh. He was also very helpful with writing my thesis.

- Tomasz Sowinski. He is the DSL programmer responsible for GeNIe. For me, he implemented the case management system in GeNIe and helped me with the C++ problems I encountered.

- Adam Zagorecki. He is a Ph.D. student in DSL and I worked together with him on the part about the canonical gates. Also, when I had a problem, he was always able to make time for me, and help me solve the problem.

- Denver Dash. He is an old DSL member now working for Intel, and he helped me with some problems I encountered while implementing the EM algorithm.

- Tsai-Ching Lu. Another old DSL member that gave me valuable feedback about the case management designs that helped me to improve the case management significantly.

Furthermore, my stay at the University of Pittsburgh would not have been possible without the financial support of:

- My parents, Kees and Hennie Voortman.

- Fundatie van de Vrijvrouwe van Renswoude.

- Stimuleringsfonds Internationale Universitaire Samenwerkingsrelaties (STIR), Delft University of Technology.

Contents

Part I

Introduction

Chapter 1

Background

In this chapter we will see in what area the research took place and what the current problems are. We will take a look at Bayesian networks, how to learn them, and the software the Decision Systems Laboratory (DSL) has developed.

1.1 Bayesian Networks

Bayesian networks (BNs) [25] [26] are a successful modeling tool that facilitates a convenient combination of expert knowledge and data. BNs have become a prominent modeling tool for problems involving uncertainty in the last 20 years. Among a wide range of their practical applications are medical diagnosis, hardware troubleshooting, user modeling, intrusion detection, disease outbreak detection, etc.

The BN framework combines strong formal foundations of probability theory with a graphical representation of interactions among variables, providing a formalism that is theoretically sound, yet readily understandable for human knowledge engineers and fairly easy to apply in practice. BNs allow for convenient and flexible fusion of information from various sources, in particular combining expert knowledge with available data. This has proved to be critical in practical applications, where data is typically limited. Elicitation of probabilities from experts is usually not easy, see [8], and we will try to make that easier.

1.2 Learning

Learning is a very useful property of Bayesian networks [13]. Especially the combination of expert knowledge and data is a powerful one: the expert builds a network and that network is refined by learning from the data. Learning from a dataset only is possible when there is a lot of data, but that is usually not the case. [14] is about combining expert knowledge with statistical data, but treats only complete data and does not use hierarchical confidence.

In this thesis we look at the situation where the data is limited. Limited data is a problem in learning, and will always be a problem for learning Bayesian networks. A lot of work has been done to learn Bayesian network from data [13], but usually it is assumed that there is a sufficient amount of data available. This research is different, in the sense that we do not assume that there is a sufficient amount of data, but we do assume that there is a network created by experts. The focus will be more on the combination of expert knowledge and data, where usually only data is used to learn the networks. Networks will be refined by data that comes available. We will use the Expectation-Maximization (EM) algorithm [5] to combine expert knowledge, by making use of hierarchical confidence, with data.

1.3 Canonical Gates

In the line of learning from a few cases, we take a look at a way to replace CPTs with canonical gates that use fewer parameters and are, because of that, capable to learn from a smaller number of cases. An example of a canonical gate is the noisy-OR gate [15], and these gates also have the advantage that they are faster when inference is concerned [12], and this makes them also very suitable for learning from large datasets. We introduce new gates that release some assumptions of causal independence (compared to, for example, noisy-OR), so that they are more flexible and better learners.

1.4 GeNIe and SMILE

The work I did for this thesis has been done in the Decision Systems Laboratory (DSL) at Pittsburgh. The people of DSL created a software package called GeNIe to create Bayesian networks and a lot more. The first version of the GeNIe software, developed by DSL, was released in 1998 and has been steadily developed from then on. Underneath GeNIe there is the SMILE engine, a fully platform independent library of C++ classes. My task was to extend GeNIe and SMILE with features that make it possible to refine Bayesian networks, based on expert knowledge, with data.

Chapter 2

The problem

In this chapter, we state four research questions we are trying to anwser in this thesis. Also, we give our hypotheses about the given questions and we will verify them in chapter 19, the conclusion.

2.1 Research Questions

Although Bayesian network are popular and successful, there are also some limitations and problems:

- The number of parameters to specify a Bayesian network can be very high. CPTs grow exponentially in the number of parents. This complicates knowledge elicitation and learning requires a lot of data.
- Inference in BNs is NP-hard [4]. Learning needs a lot of inference, so this is problematic.

Based on these problems we state the following research questions for which we will try to find an anwser in this thesis:

1. How to learn BNs from a limited amount of data?
2. How can we combine expert knowledge with data in a principled way in SMILE?
3. How can we combine expert knowledge with data in a principled way in GeNIe?
4. How can we speed-up inference?

2.2 Hypotheses

Corresponding to the research questions, here are my hypotheses:

1. We think it is impossible to learn complete BNs (structure and parameters) in a reasonable way, when there are only a few cases available. Therefore, my guess is that only network refinement will lead to sufficiently good results in this situation. To do this we will use EM as learning algorithm and hierarchical confidence to express the confidence of the expert.

2. We will have to implement an algorithm that is capable of combining expert knowledge with data. My bet is that the EM algorithm in combination with priors will lead to good results.

3. To combine expert knowledge with data in GeNIe, two things are needed. After the expert has build a network, he should express his confidence. For this we think hierarchical confidence will work and is convenient, since it is possible to express confidence on different levels of granularity. The second thing needed, is a way to manage the data that has to be combined with the network, and for this we introduce a case management system. By trying to make it intuitive and in the GeNIe style, it should be intuitive for the users. The cases that are stored in the case management system will be used to refine the network.

4. We think that canonical gates will improve the accuracy and speed of learning. Because they require fewer parameters than CPTs, they also require less data to learn. Also, because they can be exploited by inference algorithms, they can improve the speed of learning. We will present benchmarks that show this.

It is not reasonable to expect that based on, for example, 10 cases, complete networks are learned. We will focus on networks where the structure is fixed and a prior distribution and the confidence in the prior is assigned by an expert, for all of the local probability distributions (CPTs). Cases can change the CPTs more or less depending on the confidence of the expert. To improve learning with a few cases, we will introduce new gates, that require fewer parameters instead of CPTs, for better learning.

2.3 How is this thesis organized?

In chronological order, the topics this thesis is concerned with, are the following:

- Case Management.
- Learning Bayesian Networks.
- Refinement of BNs Using Cases.
- Exploiting Decomposable Causal Independence.
- Conclusion and Future Work.

The case management system is a system for managing data in a integrated way in GeNIe. Firstly, we will talk about the current status of case management support in available software. Then we will explain how the case management is implemented in SMILE, and the last chapter is about the user interface of the case management in GeNIe.

The following part is completely about the theory of learning Bayesian networks. We will see that we can distinguish four different learning problems. The possible solutions are the subject matter.

In the part about refinement of Bayesian networks using cases, the approach to learning from a low number of cases will become visible. Firstly, we will talk about my selected method for learning, i.e., the EM algorithm. A chapter about priors and hierarchical confidence will emphasize the importance of the use of priors, and the possibility to express confidence in them.

Exploiting Decomposable Causal Independence is about alternatives for CPTs that learn better from small amounts of data. We will explore different ways to make use of the decomposable property.

The last part contains a conclusion and future work.

Before going to the part about the case management system, two chapters that are an introduction to Bayesian networks, and GeNIe and SMILE, will follow now.

Chapter 3

Bayesian networks

This chapter will introduce Bayesian networks, and related concepts like conditional independence and inference.

3.1 Definition

Let $\mathbf{X} = \{X_1, \ldots, X_n\}$ be a set of random variables. A Bayesian network is an annotated directed acyclic graph (DAG) that encodes a joint probability distribution over $\mathbf{X}$, see for an example Figure 3.1. Formally, a Bayesian network for $\mathbf{X}$ is a pair $B =< G, \Theta >$. In this pair, G is a directed acyclic pair whose vertices (nodes) correspond to the random variables $X_1, \ldots, X_n$. G encodes the following set of conditional independence assumptions: each variable X_i is independent of its non-descendents given his parents in G. Θ represents the set of parameters that quantifies the network. It contains a parameter $\theta_{x_i|\Pi_{x_i}} = P(x_i|\Pi_{x_i})$ for each possible value x_i of X_i, and Π_{x_i} of Π_{X_i}, where Π_{X_i} denotes the set of parents of X_i in G. So a Bayesian network defines a complete joint probability distribution over $\mathbf{X}$ given by:

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

BNs are a combination of a sound mathematical foundation in the form of probability theory and an intuitive graphical representation of nodes (variables) and arcs. In probability theory the joint probability distribution of variables

Table 3.1: A CPT that represents the probabilistic part of a Bayesian network. The table should be read like this: $P(Child = State2|Parent = State1) = 0.9$, and the other three probabilities can be interpreted in the same way.

Child\Parent	**State 1**	**State 2**
State 1	0.1	0.7
State 2	0.9	0.3

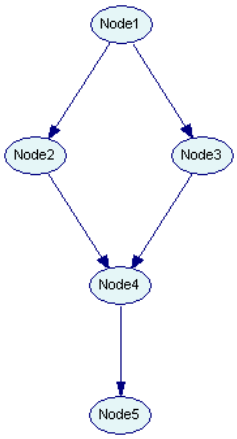

Figure 3.1: Example of the graphical part of a Bayesian network.

$X_1, \ldots, X_n$ is given by $P(X_1, \ldots, X_n)$. This can be calculated using the chain-rule of probability:

$$P(X_1, \ldots, X_n) = P(X_1)P(X_2|X_1) \cdots P(X_n|X_1, X_2, \ldots, X_{n-1}) \; .$$

Based on the structure of the network it is sometimes possible to remove variables in the equation on the right hand side. If a variable A is conditioned on a variable B and C, so $P(A|B,C)$, and A is conditionally independent of B, $P(A|B,C)$ can be reduced to $P(A|C)$. This lowers the number of parameters required to specify the JPD by exploiting independencies among domain variables.

So a BN is basically a compact representation of a joint probability distribution (JPD). The JPD is specified by means of local probability distributions associated with nodes (variables). In case of discrete variables (this is what we will focus on in this thesis), the local probability distributions are encoded in the form of prior probabilities over those nodes that have no parents in the graph, and conditional probability tables (CPTs) for all other nodes, see for an example Figure 3.1. A CPT is a set of conditional probability distributions that define a probability distribution over the child variable given all combinations of values of the parents nodes.

3.2 Conditional independence

Conditional independence is an important concept in Bayesian networks and it can be specified by:

- A node is conditionally independent of its non-descendants, given its parents.

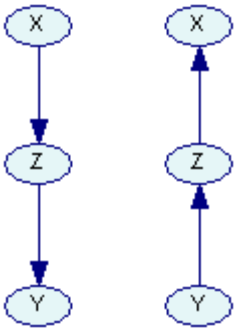

Figure 3.2: Serial connections in both directions.

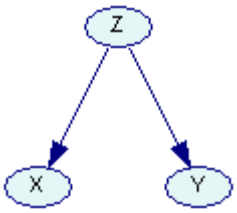

Figure 3.3: A diverging connection.

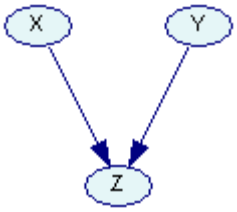

Figure 3.4: A converging connection.

- A node is conditionally independent of all other nodes in the network, given its parents, children, and children's parents. This is called the Markov blanket.

These two specifications are equivalent. A more general topological criterion is d-separation. It can be used to decide whether a set of nodes **X** is independent of another set **Y**, given a third set **Z**. It is more complicated than the two previously mentioned methods and it works as follows.

Two nodes, X and Y are d-separated if and only if for every path between them, there is an intermediate variable Z such that:

- The connection is serial or diverging and Z is known. A connection is serial if there is a path from X to Y or Y to X through Z, see also Figure 3.2. A connection is diverging if node Z has an arc to X and an arc to Y, see

also Figure 3.3.

- The connection is converging and neither Z nor any descendant of Z is known. A connection is converging if there is an arc from X and an arc from Y to Z, see also Figure 3.4.

Node X and Y are d-connected by node Z, if they are not d-separated. If two nodes are d-separated they are independent. This is very useful for inference at which we will take a look in the next section.

3.3 Inference

Inference [23] is a basic task in BNs and is used to compute the posterior probability distribution for a query variable $Q \in \mathbf{Q}$, given the set of evidence (observed) variables $\mathbf{E}$. There is a possible third set of (hidden) variables $\mathbf{H}$ that are neither query variables nor observed variables. The complete set of variables of a query is $\mathbf{X} = \{Q\} \cup \mathbf{E} \cup \mathbf{H}$. The posterior probability can be calculated in this way:

$$P(Q|\mathbf{E}) = \frac{P(Q,\mathbf{E})}{P(\mathbf{E})} = \alpha P(Q,\mathbf{E}) = \alpha \sum_{\mathbf{H}} P(Q,\mathbf{E},\mathbf{H}) \ ,$$

where α is a normalizing constant equal to $\frac{1}{P(\mathbf{E})}$. Informally, the equation says that a query can be answered using a Bayesian network by summing out the variables in $\mathbf{H}$ and dividing that by the probability of the evidence.

Speeding up inference is achieved by manipulating the sum in the equation in a clever way. One easy improvement is to move the constant terms out of the sum. A second improvement is to compute repeating values only once. There are values that do not even need to be computed: every variable that is not an ancestor of a query variable or evidence variable is irrelevant to the query. Clustering algorithms (also known as joint tree algorithms) [18] are very useful when someone is not interested in only one query variable, but if, for example, the values of all the variables in the network have to be computed. The idea is to join individual nodes of the network to form cluster nodes in such a way that the resulting network is a polytree. When this network is in polytree form, a special purpose inference algorithm is applied that can compute the values of all the nodes in the network in $O(n)$ time. But the NP-hardness does not disappear: the construction of the polytree requires exponential time and space if the network is difficult enough.

Since inference in BNs is NP-hard, approximate algorithms have been developed to handle large multiply connected networks. There is a variety of approximate algorithms:

- Direct sampling methods.
- Rejection sampling.

- Likelihood weighting [3].
- Markov Chain Monte Carlo (MCMC) [24].

The idea with direct sampling methods is to generate evidence for each variable following a partial order defined by the structure of the graph. The probability distribution from which the value is sampled is conditioned on the evidence of the parents of the node.

Rejection sampling can be used to compute conditional probabilities. Samples are generated by using the prior distribution of the network and samples are removed that are not consistent with the network. Finally, we count the occurences of the state of the query variable to estimate the probability.

Likelihood weighting is an improvement over rejection sampling in the sense that it only generates samples that are consistent with the network.

MCMC does not generate each event from scratch, but jumps from state to state. The next state is generated by randomly sampling a value for one of the non-evidence variables X_i, conditioned on the current values of the variables in the Markov blanket of X_i. The idea is that in the long run the number of visits to each state is exactly proportional to its posterior probability.

3.4 Relevance Reasoning

Relevance reasoning is a method used in inference that computes only the posterior probability of the nodes that are really needed. Before performing inference, all the nodes that are irrelevant for the query are removed. We will not go into technical details, but we are introducing this concept here, because in GeNIe and SMILE there exists the notion of target nodes. If there are target nodes in a network, only the posteriors of those nodes, and the nodes of which the posteriors are needed to compute the posteriors of the target nodes, will be computed. For more information, see [7] and [19].

3.5 Canonical Distributions

The number of parameters that needs to be specified in a BN grows exponentially in the number of parents. If a node has n parents and all the nodes are binary, there are 2^n parameters required to specify the CPT of the child. One way to reduce the number of parameters is to use canonical distributions to describe the child-parents relationship instead of a CPT. A canonical distribution fits some standard pattern that is selected by assuming some characteristics of the relationship between child and parents.

3.5.1 Deterministic Nodes

Deterministic nodes are one of the simplest canonical gates that are available. The value of a deterministic node is exactly specified if the values of his parents are known. Example of deterministic nodes are logical gates like AND and OR.

3.5.2 Noisy-OR and Noisy-MAX Gates

Uncertain relationships can often be characterized by the so-called 'noisy' relationship. The noisy-OR relationship [15] is a generalization of the logical OR. To explain the difference, take a look at the following example. Suppose we have a node that has three causes (parents) that can make the node true. In case of the logical OR the child is true if one of the causes is true. In case of the noisy-OR the probability that the child is true raises, but it does not have to be 1. Noisy-MAX is again a generalization, but then of the noisy-OR gate. See the part V about exploiting decomposable causal independence for more information about noisy-OR and noisy-MAX.

3.6 Bayesian Networks in Practice: Diagnosis

Diagnosis is probably one of the most succesful applications of Bayesian network. Based on symptoms, a BN tries to identify the underlying cause of the symptoms. In medical diagnosis this means that based upon symptoms of the patient, the underlying disease or condition has to be found. GeNIe has a special diagnosis extension that can be turned on. The DSL lab has also developed software to automatically generate diagnostic models, see, for example, [17].

Chapter 4

GeNIe and SMILE

The GeNIe and SMILE software has been developed by the Decision Systems Laboratory (DSL) of the University of Pittsburgh. SMILE is the engine and GeNIe is the graphical user interface on top of SMILE. Both can be downloaded from the GeNIe and SMILE website at `http://www.sis.pitt.edu/~genie`.

4.1 GeNIe

The GeNIe (Graphical Network Interface) software package can be used to create decision theoretic models intuitively using the graphical click-and-drop interface. It is the graphical interface to SMILE, a fully portable Bayesian inference engine developed by the Decision Systems Laboratory and thoroughly tested in the field since 1998. GeNIe 2.0 is the latest version of GeNIe. GeNIe 1.0, released to the community in 1998, has received a wide acceptance within both academia and industry. Users of the programs have shared with us their experiences and their suggestions have led to the development of GeNIe 2.0. GeNIe 2.0 has a refreshingly new modern interface, and is even more intuitive and easier to use than GeNIe 1.0. In addition to aesthetics, GeNIe 2.0 has many features to offer.

Primary features:

- Graphical editor to create and modify network models.
- Uses the SMILE engine. You may develop models in GeNIe and create a custom interface for them using SMILE.
- Supports chance nodes with General, Noisy-OR/MAX and Noisy-AND distribution.
- Open multiple networks and cut and paste sections of models between them.
- Complete integration with MS. Excel, cut and paste data into internal spreadsheet view of GeNIe.

- Cross compatibility with other software. Supports all major file types (e.g., Hugin, Netica, Ergo).

- Support for handling observation costs of nodes.

- Support for diagnostic case management.

4.2 SMILE

SMILE (Structural Modeling, Inference, 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 SMILE API (Application Programming Interface), allows to create, edit, save, and load graphical models, and use them for probabilistic reasoning and decision making under uncertainty.

SMILE supports directly the object-oriented methodology. SMILE is implemented in C++ in a platform independent fashion. Individual classes of SMILE are accessible from C++ or (as functions) from the C programming language. Since most implementations of programming languages define a C interface, this makes SMILE accessible from practically any language on any system. SMILE can be embedded in programs that use graphical probabilistic models as their reasoning engines. Models developed in GeNIe can be equipped with a user interface which utilizes SMILE as the backend engine. SMILE is released as a dynamic link library (DLL). There are also several SMILE wrappers, such as SMILE.NET (.NET interface), SMILEX (Active X), jSMILE (Java interface), etc.

Primary features:

- Graphical editor to create and modify network models.

- Platform independent, versions available for Windows, Unix (Solaris), Linux, Mac, Pocket PC, etc.

- SMILE.NET available for use with .NET framework. Compatible with all .NET languages, including C# and VB.NET. May be used to create web-based applications of Bayesian networks.

- Thorough and complete documentation.

- Robust and running successfully in the field since 1997.

- Responsive development team support that will compile SMILE for any platform on demand.

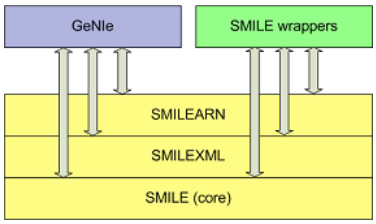

Figure 4.1: The different layers of SMILE.

4.3 Design of SMILE

SMILE is build modularly with the SMILE C++ library as core. SMILE contains only the basic functions like for example to build a network and to perform inference. For persistence, the SMILEXML library is available. This library contains methods to read and write XML files. A module that will be introduced in the future is SMILEARN. This module is a learning API and is in development for the last few months and is not publicly available yet. I designed the new API together with Adam Zagorecki and Tomasz Sowinski, and more details will follow in part IV.

On top of the SMILE modules we have the SMILE wrappers and GeNIe. They just 'talk' to the SMILE modules we just mentioned to perform the requests made. See Figure 4.1 for an overview of all the layers.

Part II

Case Management

Chapter 5

Management of Cases in Existing Systems

This chapter will give an overview of existing case management systems. A case management system is, in our context, a tool to manage data that belongs to a network. When a user sets evidence and targets nodes in a network, he might want to save this. When he saves it, this is considered to be one case, but he can save as many cases as he wants. It is useful to store cases for later use, but it is even more useful to use the cases to refine the network. This is what we will do in part IV.

5.1 What are Cases?

Cases have different meanings in different contexts. In a hospital, for example, a case can be a patient where doctors are trying to diagnose a disease. If a doctor talks about a case, he means the patient and the file that contains the medical history of the patient.

When we talk about a case, it is in the context of Bayesian networks. A case in a Bayesian network consists of the the states of the variables in the model (the state can also be unknown), plus some additional data. Examples of additional data are, for example, unique case names and target nodes in the network.

5.2 Current State of Case Support in Software

Before we are going to talk about the new case management system, lets take a look at the current state of the available software. We also investigated if the cases can be used for learning, something we will do in part IV. An overview of the findings are in Table 5.1. We will go through this table step by step.

For this comparison we focused on four different features:

Table 5.1: Comparison between the different software products.

	Import data	**Edit cases**	**Param. learn.**	**Struct. learn.**
Bayesia	Yes	No	Yes	Yes
GeNIe	No	No	No	No
Hugin	Yes	No	Yes	Yes
Netica	Yes	No	Yes	No
SamIam	Yes	No	Yes	No

- Import data: this means that data can be imported to the program and can be used for learning.
- Edit cases: this means that there are built-in features to edit cases (data), comparable to our case management system.
- Parameter learning.
- Structure learning.

There are some remarkable observations that can be derived from the comparison. One is that GeNIe does not have even one of the features stated and this can be explained by the fact that DSL is not a company and they have only limited resources. But in the short future GeNIe will also support almost all features, because currently there is a lot of development in all the four areas in DSL. Besides this work on the case management and refinement, work is also done to build a learning interface for importing data from different data sources. Some features are already implemented in SMILE, but do not have an interface in GeNIe yet.

Another interesting point is that editing cases is not supported by any software package. Our approach, however, will keep the cases tightly integrated to the network and it is this point what will be the biggest difference with the other software. We will take a look at the advantages and disadvantages of this approach in the next chapter. Most of the software has the capability of learning parameters, and sometimes structure from data, but refinement of the network using cases is usually not supported.

Chapter 6

Design of Case Management in SMILE

In the previous chapter we already explained what cases are, and compared existing facilities to manage cases. The conclusion was that there are usually no built-in facilities to edit cases, but that importing cases (data) to learn a BN is possible. In our approach the case management system is separate of learning. Case management is a stand-alone GeNIe and SMILE feature and case refinement is an extension on this, i.e., it uses the cases in the case management system to refine a network. This chapter will explain the design of the case management system in SMILE.

6.1 Attributes of a Case

The current design of the case management system stores the following attributes of a case (more attributes could be stored in the future):

- The unique case name.
- The description.
- The category.
- The creation date of the case.
- The last modification data of the case.
- The evidence set in the network.
- The target nodes set in the network.

Case management in SMILE basically exists of two classes, i.e., DSL_caseManager and DSL_simpleCase (this name is because there already was a class named

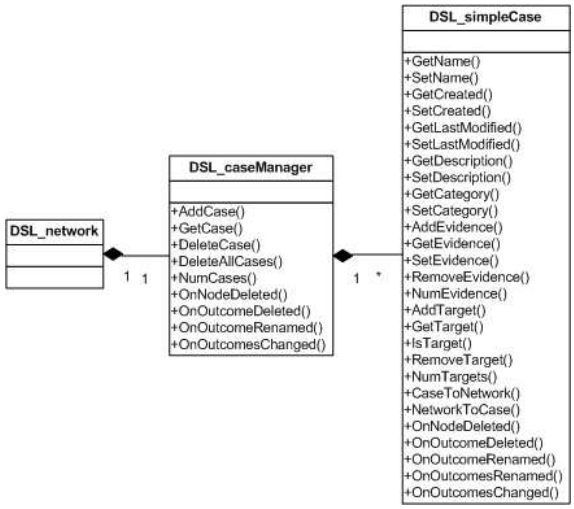

Figure 6.1: UML diagram of the case management system. Unimportant details have been left out.

DSL_case in SMILE that is used for diagnosis), and as you can see in Figure 6.1. DSL_caseManager holds a collection of DSL_simpleCases. One other important task is that it forwards method calls that indicates a change in the network, to all the cases that are currently hold in DSL_caseManager.

The main case management logic is implemented in DSL_simpleCase. Most of the methods are for manipulating and quering the evidence and targets in a case. The two key methods are CaseToNetwork and NetworkToCase, where the former method copies all the evidence and targets in a case to the network, and the latter does it the other way around.

The DSL_caseManager is being kept in DSL_network. When a network is created, the DSL_caseManager is also automatically created. If the network and cases are loaded from file, they are of course automatically initialized. DSL_network works as a facade for the DSL_caseManager, because DSL_network contains the same methods as DSL_caseManager, but just forwards the calls to DSL_caseManager. In this way, it is not necessary for people that are using the SMILE API to know about the DSL_caseManager, it is encapsulated.

6.2 Hooking into the Network

Obviously, cases are very tightly related to a certain network, because most, or all, variables match. Cases should be consistent with the network. If evidence in a case cannot be set in the network this will lead to problems. To force that cases are consistent, we update the cases when changes in the network occur. When a node or state is removed, or a statename has been changed in the network, we have to update the cases. Nodes or states that are added are

not anywhere in the cases yet, so nothing has to be changed. Summarizing, we have the following actions:

- Removal of a node. If this node contained evidence, this node will be removed from the evidence list. The same holds for a target node.
- Removal of a state. If there are nodes that have this state as evidence, this will be removed from the evidence list.
- Change of a statename. When this happens we have to update the statename of the corresponding node in the cases too.

Everything we just mentioned is implemented in the SMILE core, since it is closely related to the core SMILE features.

6.3 Storage

Usualy data is stored in separate data files, but we chose the novel approach of storing the cases in the same file as the network is stored. This is basically a consequence of the tight integration of cases with the network. Cases 'belong' to a network, so they can just as well be stored in the same file. In this way, we enforce that the user can not use cases that are inconsistent with the network. Also, it is very convenient to have only one file that contains everything. Exchange of networks including cases becomes a lot easier.

Here is an example of an XDSL file, containing a network and cases, where irrelevant details have been left out:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<smile version="1.0" id="Aa" numsamples="1000">
  <nodes>
    ...
  </nodes>
  <cases>
    <case name="case1" category="Mark">
      <description>The first case</description>
      <evidence node="node_7" state="value1" />
      <evidence node="node_32" state="value1" />
      <evidence node="node_34" state="value2" />
      <evidence node="node_10" state="value0" />
      <evidence node="node_15" state="value0" />
      <evidence node="node_21" state="value2" />
      <evidence node="node_36" state="value1" />
    </case>
    <case name="case2" category="Mark">
      <description>The second case</description>
      <evidence node="node_7" state="value1" />
      <evidence node="node_13" state="value0" />
```

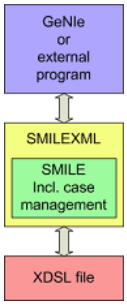

Figure 6.2: Overview of the saving to and loading from the XDSL file.

```
      <evidence node="node_18" state="value0" />
      <evidence node="node_24" state="value1" />
      <evidence node="node_28" state="value1" />
      <evidence node="node_31" state="value3" />
      <evidence node="node_10" state="value1" />
    </case>
    <case name="case3" category="Mark">
      <evidence node="node_5" state="value0" />
      <evidence node="node_0" state="value1" />
      <evidence node="node_13" state="value0" />
      <evidence node="node_26" state="value1" />
      <evidence node="node_32" state="value1" />
      <evidence node="node_36" state="value0" />
    </case>
  </cases>
  <extensions>
    ...
  </extensions>
</smile>
```

Of course there should be a possibility to import and export cases from the main XDSL file. This is done by using XML files where the same tags are used as in the XDSL file. When imported, the cases are checked for consistency with the network. All of this is implemented in SMILEXML.

One problem is that the XDSL file can become very large, and that it can take a long time to load. Storing the network and cases in separate files does not really solve this problem, since you have to load both files then. The only advantage would be if a user only wants to edit the network and does not want to do something with the cases. This is, however, not a very likeli scenario and another thing is, that we think that users of GeNIe usually do not have a really big dataset. In the future, we might decide to store both network and cases in a database, because it offers the big advantage that cases are individually accessible. The main problem is that a database has to be delivered with GeNIe

Table 6.1: Overview of the differences between the cases.

Diagnostic cases	**General cases**
Loosely coupled with a network.	Tightly coupled with a network.
Does not keep cases synchronized with the network.	Keeps cases synchronized with the network.
Separate file for the cases.	Stored in the XDSL file.

and SMILE, or the user has to install one himself, and that makes things more complicated.

To make use of the case management in SMILE we need an user interface for users to work with. We will describe this in the next chapter.

6.4 Comparison with Existing Diagnostic Cases

In SMILE and GeNIe there are already diagnostic cases available. They are not as powerful as general cases, e.g., diagnostic cases can only be used in GeNIe when diagnosis is turned on, where general cases can be used always. It is likely that the diagnostic cases will be replaced by the general cases in the future. It will take time since GeNIe and SMILE have to be backwards compatible, because the diagnostic cases are already used by GeNIe and SMILE users. An overview of the differences between the two types of cases can be found in Table 6.1

Chapter 7

Case Management in GeNIe

This chapter is about the graphical design of the case management system in GeNIe. GeNIe forwards all the user requests in the interface to SMILE, that executes them.

7.1 Case Management

By default the case management system in GeNIe is deactivated. It can be activated in several different ways: using the menu, using a button on the panel or a key combination. Once activated, the case management consists of the basic functions of adding, editing, saving, deleting, selecting and searching cases. We will discuss the features one by one in this chapter.

Tomasz Sowinski implemented the part of the case management system in GeNIe. I implemented the SMILE part and made the designs for the GeNIe interface. The screenshots in this chapter are from the latest case management prototype. Improvements in the near future can be expected, for example, the icons will be replaced by somewhat more profesional ones when they become available.

The main user view is shown in Figure 7.1 where the case management is fixed in the main GeNIe view. It is also possible to have it in a separate panel as you can see in Figure 7.2. One important design decision was to show by default only the columns where for at least one of the cases evidence is available. Typically, evidence is only available for certain variables, and that justifies this approach. This makes the interface for the user a lot easier to manage, certainly in combination with the search functionalities. It is possible switch between showing all the columns or only the columns with evidence by using a button.

7.2 Adding Cases

The use of cases is integrated in the main model view. If the user sets evidence and targets in the network model, he can save it by pressing the save case button.

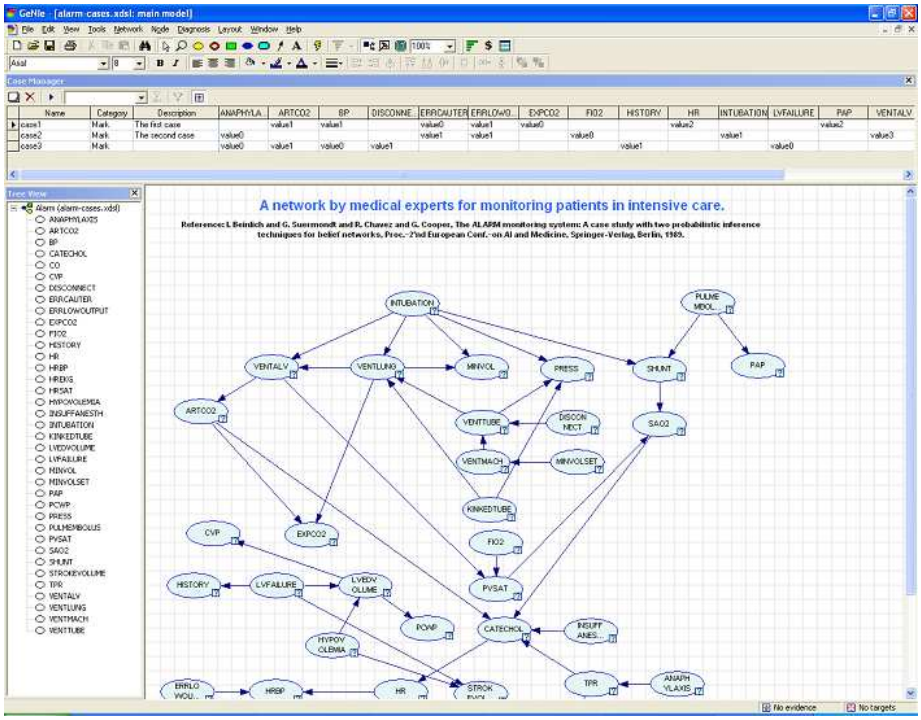

Figure 7.1: The GeNIe interface with the case management system. The text may not be readable, but it shows the different panels. The case management is between the buttons on the top and the network.

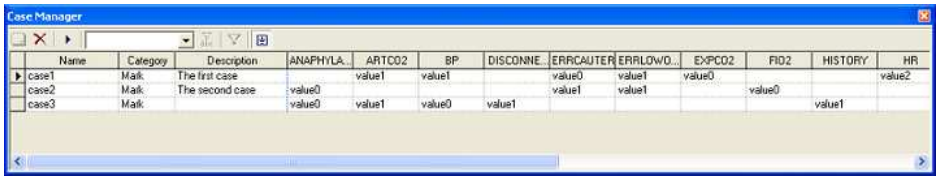

Figure 7.2: Case management in a separate window.

In Figure 7.3 you can see a pop-up displays that asks for a required unique case name, an optional category, and an optional description. After saving a new case the case is still active in the network and can be edited.

7.3 Editing and Saving Cases

Cases can be edited in the same way as adding a case, with the difference that the case already exists. However, there are two places where cases can be edited:

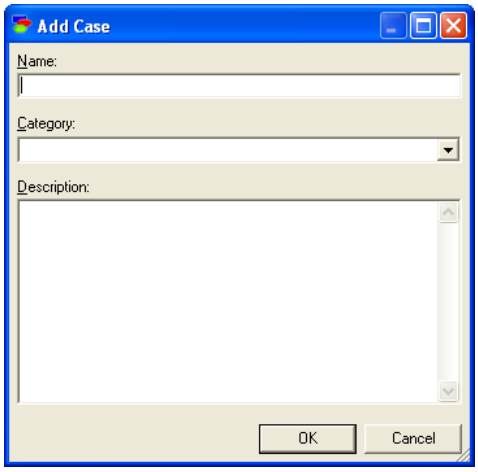

Figure 7.3: The window to add a case.

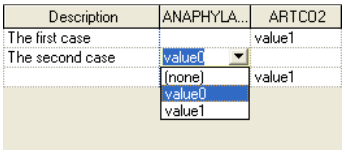

Figure 7.4: Changing a value in the case management screen.

- In the network view of GeNIe in the same way as adding a case. The case that is instantiated in the network can be edited.

- In the case management screen, see Figure 7.4. All the cases can be edited at the same time in the table view.

If the case that is edited is an already saved case, a * sign is appended to the visible case name. When the user is finished, there are two ways to save a case:

- Save. This saves the current case under the same name.

- Save as. A pop-up displays that ask for a required unique case name, an optional category, and an optional description. 'Save as' adds a new case to the case database.

In this situation, by saving we mean saving in memory. The cases are written to file only when the whole network is saved. This is a consequence of the fact that everything is stored in one file. For more information, see the previous chapter.

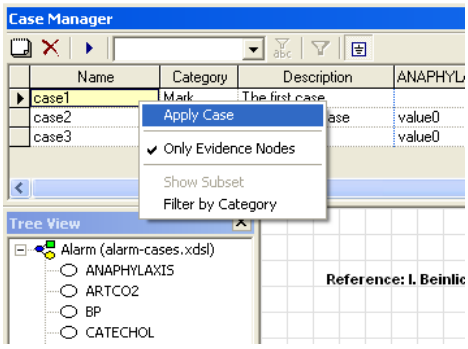

Figure 7.5: Applying a case to the network.

7.4 Selecting Cases

Selecting cases happens by selecting the case and applying it in the network. There is an apply button and also right clicking and selecting 'Apply case' is possible as can be seen in Figure 7.5. The apply button looks like the 'play' sign on video recorders. When a case is applied, the evidence and targets in the case are copied to the network.

7.5 Deleting Cases

A case can be deleted by the delete button. It will delete the currently active case, but first the user will be asked for a confirmation.

7.6 Search

It is possible to search through the cases by entering keywords in the search field and pressing the enter key or search button. The system will search in case names, case categories and case descriptions, see also Figure 7.6. It is possible to switch back and forth between all the cases and the cases that are the result of a search query. Additionally, it is possible to sort all the columns in the case management by clicking on the header. Clicking again reverses the sorting order.

7.7 Importing and Exporting Cases

For importing and exporting cases we use a prescribed XML file format as was mentioned in the previous chapter. A dialog window opens to select or create a file.

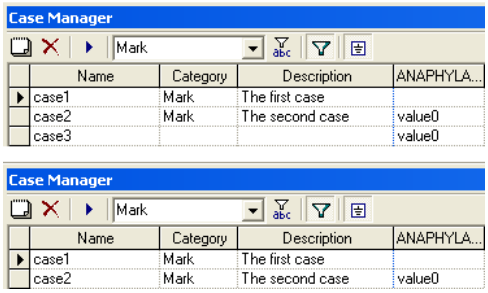

Figure 7.6: A simple search example, the upper one is before and the lower one after searching.

7.8 Case Management and Learning

There will be a button 'learn' that uses the cases to update the parameters in the network. No feedback is required, because the priors in the network and the hierarchical confidence contains all the information needed. How this exactly works will be described in Part IV about refinement of BNs using cases.

Part III

Learning Bayesian Networks

Chapter 8

Statistical and Bayesian Learning

Learning is the process of fitting a Bayesian network to the knowledge of an expert, and/or a dataset. The better the network fits, the better the network will perform, as long as it does not overfit the data. Learning a Bayesian network involves learning the structure of a network (graph) and learning the probabilities in a network (CPTs). There are four cases in learning:

1 Known structure, complete data.

2 Known structure, incomplete data.

3 Unknown structure, complete data.

4 Unknown structure, incomplete data.

In the next chapters we will take a look at these four cases, but first we will say something about the difference between statistical learning and Bayesian learning.

8.1 Statistical Learning vs Bayesian Learning

In learning with Bayesian networks there are essentially two fundamentally different approaches that can be used. One is statistical learning (the classical approach), usually in the form of Maximum Likelihood Estimation (MLE), the other is Bayesian learning. Lets take the example of a coin toss sequence and try to estimate the parameter θ, i.e., the probability of heads. In the classical approach θ is fixed, and unknown, and each dataset D will occur with a certain probability, i.e., $P(D|\theta)$. Eventually, we choose the parameters that most likely generated the actually observed data.

In contrast, in the Bayesian approach D is fixed and we imagine all possible values of θ from which the set could have been generated. θ is treated as a

Table 8.1: An overview of the differences and similarities of statistical and Bayesian learning.

Statistical	**Bayesian**
Fixed parameters	Fixed dataset
No priors	Priors
Bad with small dataset	Can handle small datasets
Asymptotically consistent	Asymptotically consistent

random variable, so when we want to compute the exact expectation we have to average over all possible values of θ:

$$E_{P(\theta|D)}(\theta) = \int \theta P(\theta|D) d\theta \; .$$

The question which approach is better has lead to endless debates. It is not possible to say which one is best, the approaches are just different, but self-consistent, and they should be applied in the right situation. An overview of the differences and similarities are given in Table 8.1.

Chapter 9

Learning Parameters of a Known Structure with Complete Data

The first of the four different cases in learning, and the easiest, is learning parameters of a structure that is known, and the data is complete. We will explain how to the two different types of learning, maximum likelihood estimation and Bayesian learning, in this scenario.

9.1 Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) is the principle of finding values of parameters that fit the data best. It is one of the most commonly used estimators in statistics. To be more formal, MLE finds parameters in such a way that they maximize the likelihood function, given by:

$$L(\theta : D) = P(D|\theta) = \sum_m P(x[m]|\theta) \ ,$$

where $x[m]$ denotes sample (or case) m. A basic assumption of this approach is that we assume the samples to be independently and identically distributed (i.i.d. for short).

Suppose we have data of a coin toss: HTHHT (H=heads, T=tails). We can give the likelihood function as (where θ is the probability of heads):

$$L(\theta : D) = \theta \cdot (1-\theta) \cdot \theta \cdot \theta \cdot (1-\theta) \ .$$

To compute the likelihood of this coin toss, we only need the number of heads N_H and tails N_T in the experiment. The likelihood function becomes:

$$L(\theta : D) = \theta^{N_H} \cdot (1-\theta)^{N_T}$$

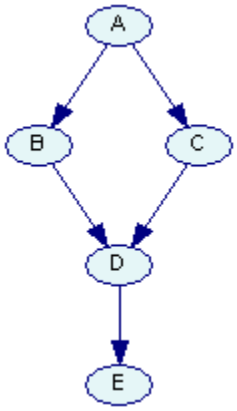

Figure 9.1: Bayesian network used as a more complex example for maximum likelihood learning.

where N_H and N_T are said to be sufficient statistics for this binomial distribution, because only these numbers are needed to compute the likelihood function. In general, sufficient statistics can be obtained by counting occurences in the dataset.

To compute the optimal likelihood parameters, we take the derivative and set it equal to 0. Usually the log likelihood is used, because it is more convenient to take the derivative.

$$\log L(\theta : D) = \ell(\theta : D) = N_H \ln \theta + N_T \ln(1 - \theta) \; .$$

Taking the derivative (we replaced θ by $\hat{\theta}$ to indicate that we are calculating the optimal θ), we obtain:

$$\ell'(\theta : D) = 0 = \frac{N_H}{\hat{\theta}} + \frac{N_T}{1 - \hat{\theta}} \; ,$$

which leads to:

$$\hat{\theta} = \frac{N_H}{N_H + N_T} \; .$$

Lets apply this principle to a more complex dataset based on the Bayesian network in Figure 9.1:

$$D = \begin{bmatrix} A[1] & B[1] & C[1] & D[1] & E[1] \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ A[M] & B[M] & C[M] & D[M] & E[M] \end{bmatrix} .$$

All the parameters are binomially distributed, just like in the coin toss example.

For the likelihood function we get:

$$\begin{aligned}
L(\theta : D) &= \prod_m P(A[m], B[m], C[m], D[m], E[m] : \theta) \\
&= \prod_m P(A[m]|\theta_A) \cdot P(B[m]|A[m] : \theta_{B|A}) \cdot P(C[m]|A[m] : \theta_{C|A}) \cdot \\
&\quad P(D[m]|B[m], C[m] : \theta_{D|B,C}) \cdot P(E[m]|D[m] : \theta_{E|D}) \\
&= \prod_m P(A[m]|\theta_A) \cdot \prod_m P(B[m]|A[m] : \theta_{B|A}) \cdot \prod_m P(C[m]|A[m] : \theta_{C|A}) \cdot \\
&\quad \prod_m P(D[m]|B[m], C[m] : \theta_{D|B,C}) \cdot \prod_m P(E[m]|D[m] : \theta_{E|D}) \; .
\end{aligned}$$

The first rewrite is because of the network layout. It introduces the conditional dependencies of the variables. The second rewrite is because the terms are independent. What we are left with are 4 independent estimation problems. Lets generalize our findings to all Bayesian networks with binomial variables:

$$\begin{aligned}
L(\theta : D) &= \prod_m P(x_1[m], \ldots, x_n[m] : \theta) \\
&= \prod_m \prod_i P(x_i[m]|Pa_i[m] : \theta_i) \\
&= \prod_i \prod_m P(x_i[m]|Pa_i[m] : \theta_i) \\
&= \prod_i L_i(\theta_i : D) \; .
\end{aligned}$$

In the rewrite above we see that the likelihood function can be decomposed into seperate likelihood functions, according to the structure of the network. This is a very desirable property, because it makes the likelihood function locally computable.

We can also generalize the likelihood function to multinomial variables:

$$L(\theta : D) = \prod_{k=1}^{K} \theta_k^{N_k} \; .$$

Again, the optimal parameters can be computed in closed form:

$$\hat{\theta}_k = \frac{N_h}{\sum_l N_l} \; .$$

9.2 Learning Parameters by Bayesian Inference

The Bayesian approach to parameter learning needs only one tool: inference. Parameters are treated as random variables themselves. In this section we take a look at an example that uses this approach.

Bayesian learning starts with making hypotheses about the data that is given. These hypotheses are expressed by the prior distributions of the parameters that are in the dataset. Examples of frequently used prior distributions are the Beta distribution and the Dirichlet family. They have the wonderful property that the posteriori distribution is of the same type of distribution as the apriori distribution.

This is the *beta* distribution, used for binomial distributions:

$$beta[a,b](\theta) = \alpha\theta^{a-1}(1-\theta)^{b-1} \ .$$

The Dirichlet distribution is used for multinomial distributions. The Dirichlet prior is given by:

$$P(\Theta) \propto \prod_{k=1}^{K} \theta_k^{\alpha_k - 1} \ ,$$

for legal $\theta_1, \ldots, \theta_K$.

The posterior has the same form, with hyperparameters $\alpha_1 + N_1, \ldots, \alpha_k + N_k$:

$$P(\Theta|D) \propto P(\Theta)P(D|\Theta) \propto \prod_{k=1}^{K} \theta_k^{\alpha_k - 1} \prod_{k=1}^{K} \theta_k^{N_k} = \prod_{k=1}^{K} \theta_k^{\alpha_k + N_k - 1} \ .$$

The hyperparameters $\alpha_1, \ldots, \alpha_k$ can be thought of as imaginary counts from our prior experience. The sample size is equal to $\alpha_1 + \cdots + \alpha_k$. The larger the sample size is, the more confident we are in our prior.

It is possible to compute the prediction of a new event in closed form:

$$P(X[1] = k) = \int \theta_k \cdot P(\Theta) d\Theta = \frac{\alpha_k}{\sum_l \alpha_l} \ .$$

Since the posterior is also Dirichlet, we get

$$P(X[M+1] = k|D) = \int \theta_k \cdot P(\Theta|D) d\Theta = \frac{\alpha_k + N_k}{\sum_l (\alpha_l + N_l)} \ .$$

The Dirichlet prior is said to be a conjugate family for the multinomial likelihood. This is because of the property that the posterior follows the same parametric form as the prior distribution.

Chapter 10

Learning Parameters of a Known Structure with Incomplete Data

This chapter is similar to the previous chapter, but with the difference that we have to learn from incomplete data now.

10.1 Missing Values and Hidden Variables

Data can be incomplete in two different ways:

- There are missing values.

- There are hidden variables.

The difference is that missing values can occur everywhere in the dataset. In case of hidden variables, all their values are missing.

In case of missing values we usually make the assumption that the values are missing at random (MAR): The probability that the value of X_i is missing is independent of its actual value given other observed values.

10.2 Problems with Incomplete Data

Incomplete data introduces problems that we did not have with complete data. Where learning with complete data resulted in independent posteriors, they can be inderdependent when learning from incomplete data. The consequences are that the maximum likelihood parameters can not be computed separately for each multinomial.

10.3 Maximum Likelihood Estimation

With incomplete data, the likelihood function can have multiple global maxima. Finding the MLE parameters becomes a nonlinear optimization problem. One option to find the optimal parameters is to use the method of the Gradient Ascent (GA), where the idea is to follow the gradient of the likelihood with reference to the parameters:

$$\frac{\partial \log P(D|\Theta)}{\partial \theta_{x_i,pa_i}} = \frac{1}{\theta_{x_i,pa_i}} \sum_m P(x_i, pa_i|o[m], \Theta) \ .$$

This requires the computation of $P(x_i, pa_i|o[m], \Theta)$ for all i, m. The advantages of gradient ascent are:

- Flexible.
- Closely related to methods in neural network training.

And the disadvantage:

- To get reasonable convergence we need to combine it with 'smart' optimization techniques.
- We have to specify a step size.

The second possibility to find the optimal parameters is the Expectation-Maximization algorithm (EM). EM is a general method to learn from incomplete data and is much simpler than gradient ascent, it uses inference as a subroutine. The idea is the following: If we had access to sufficient statistics, then we can estimate the parameters. However, the missing values prevent us from computing the sufficient statistics. Hence, we complete the sufficient statistics using the current parameter assignment. Then we compute the new parameters and iterate again. More about EM in part IV.

10.4 Bayesian Inference

Remember that with complete data there is a closed form solution for the integral:

$$P(x[M+1]|D) = \int P(x[M+1]|\theta) \cdot P(\theta|D) d\theta$$

In case of incomplete data there is no such solution. Incomplete data means:

- No sufficient statistics.
- Posterior does not compose.
- No closed form.

We need to use approximations. The simplest approximation method is using MAP parameters. MAP stands for Maximum A Posteriori assignment:

$$P(x[M+1]|D) \approx P(x[M+1]|\tilde{\theta}) \ ,$$

where $\tilde{\theta} = \arg\max_\theta P(\theta|D)$. We make the assumption that the posterior mass is dominated by MAP parameters.

We can use the same techniques to find the MAP parameters as finding the maximum likelihood (ML) parameters. But instead of maximizing $L(\theta : D)$, we have to maximize $P(\theta|D)$.

Another type of approximation is the stochastic approximation. First sample $\theta_1, \ldots, \theta_k$ from $P(\Theta|D)$, then:

$$P(x[M+1]|D) \approx \frac{1}{k} \sum_i P(x[M+1]|\theta_i) \ .$$

The question that remains is how we sample from $P(\Theta|D)$. For this, we can use Markov Chain Monte Carlo (MCMC) methods. We will not go into more detail, since this method is not of our main interest.

Chapter 11

Learning Parameters and Structure with Complete Data

There are essentially two forms of learning a structure from data:

1. Constraint based. Try to find a graph that satisfies all the constraints implied by the empirical conditional dependencies measured in the data.
2. Score based (Bayesian). Search through the space of graphs and use some scoring metric to evaluate them, and return the graph with the highest scoring function.

Both are consistent: with a sufficient amount of data and computation, they learn the correct structure.

11.1 Constraint Based

Constraint based learning aims at finding independencies between variables in a dataset. These independencies are used to infer causal links, and the target is to find the set of structures that is consistent with the found independencies. The constraint based approach is older than the score based approach, and is not used as much anymore. An example of an algorithm is the PC algorithm [26]. It takes as input a set of conditional independencies and outputs a 'pattern' that represents a Markov equivalence class of causally sufficient models.

11.2 Score Based

The score based approach is the more Bayesian approach of the two. It searches over the space of models and scores each model using the posterior probability of the model given the data.

11.2.1 Maximum Likelihood Estimation

To score the structures we use the log likelihood score for structures:

$$\begin{aligned} \ell(G:D) &= \log L(G:D) \\ &= M \sum_i (I(X_i; Pa_i^G) - H(X_i)) \ , \end{aligned}$$

where M is the number of samples (or cases), $H(X)$ is the entropy of X and $I(X_i; Pa_i^G)$ is the mutual information between X_i and Pa_i^G. The mutual information tells how much information the parents Pa_i^G give about child X_i. Some properties of mutual information:

- $I(X;Y) \geq 0$.

- $I(X;Y) = 0$ iff X and Y are independent.

- $I(X;Y) = H(X)$ iff X is totally predictable given Y.

So, the higher the score of $\ell(G : D)$, the larger the dependency of each variable is on its parents. But when you add extra arcs, the score always gets higher: $I(X;Y) \leq I(X;Y,Z)$. The maximum score is attained by a fully connected network. We need to do something to prevent this, and thus penalize complexity (more arcs).

Minimum Description Length (MDL) is a method of assigning scores to networks that penalize complexity. The description length is given by:

$$DL(D:G) = DL(G) + \frac{\log M}{2} \dim(G) - \ell(G:D) \ ,$$

where $DL(G)$ is the number of bits needed to encode G, $\frac{\log M}{2} \dim(G)$ the number of bits needed to encode Θ_G, and $\ell(G : D)$ the number of bits needed to encode D using (G, Θ_G). Minimizing the description length, is equivalent to maximizing

$$MDL(G:D) = \ell(G:D) - \frac{\log M}{2} \dim(G) - DL(G) \ .$$

The likelihood is roughly linear in M. The penalty is logarithmic in M. As we get more data, the penalty for a complex structure is less.

The MDL score is consistent, that is:

- As $M \to \infty$ the true structure G^* maximizes the score.

- For sufficiently large M, the maximal scoring structures are equivalent to G^*.

11.2.2 Bayesian Inference

For Bayesian learning, we have to compute the expectation over the unknown G:

$$P(x[M+1]|D) = \sum_G P(x[M+1]|D,G)P(G|D) \ ,$$

where

$$P(G|D) \propto P(D|G)P(G) = \int P(D|G,\theta)P(\theta|G)d\theta P(G) \ .$$

We made the assumption that the Gs are mutually exclusive and exhaustive. $P(G|D)$ is the posterior score, $P(D|G)$ the marginal likelihood, $P(D|G,\theta)$ the likelihood, $P(\theta|G)$ the prior over the parameters, and $P(G)$ the prior over the structures.

Now, assume we have observed a sequence of coin tosses and we want to predict the next one. The chain rule gives us:

$$P(x[1],\ldots,x[M]) = P(x[1])P(x[2]|x[1])\cdots P(x[M]|x[1],\ldots,x[M-1]) \ .$$

Recall that

$$P(x[m+1] = H|x[1],\ldots,x[m]) = \frac{N_G^m + \alpha_H}{m + \alpha_H + \alpha_T} \ ,$$

where N_H^m is the number of heads in the first m samples. The marginal likelihood is given by:

$$\begin{aligned} P(x[1],\ldots,x[M]) &= \frac{\alpha_H}{\alpha_H+\alpha_T}\cdots\frac{N_H-1+\alpha_H}{N_H-1+\alpha_H+\alpha_T}\cdot \\ &\quad \frac{\alpha_T}{N_H+\alpha_H+\alpha_T}\cdots\frac{N_T-1+\alpha_T}{N_H+N_T-1+\alpha_H+\alpha_T} \ . \end{aligned}$$

We simplify this by using the Gamma function: $(\alpha)(1+\alpha)\ldots(N-1+\alpha) = \frac{\Gamma(N+\alpha)}{\Gamma(\alpha)}$.

$$P(x[1],\ldots,x[M]) = \frac{\Gamma(\alpha_H+\alpha_T)}{\Gamma(\alpha_H+\alpha_T+N_H+N_T)}\frac{\Gamma(\alpha_H+N_H)}{\Gamma(\alpha_H)}\frac{\Gamma(\alpha_T+N_T)}{\Gamma(\alpha_T)}$$

We generalize this for multinomials, where we have $P(\Theta)$ as a Dirichlet prior and $\alpha_1,\ldots,\alpha_K$ as hyperparameters. D is a dataset with sufficient statistics $N_1,\ldots,N_K$.

$$P(D) = \frac{\Gamma(\sum_l \alpha_l)}{\Gamma(\sum_l + N_l)}\prod_l \frac{\Gamma(\alpha_l+N_l)}{\Gamma(\alpha_l)}$$

We get the following marginal likelihood:

$$P(D|G) = \prod_i \prod_{Pa_i^G} \frac{\Gamma(\alpha(pa_i^G))}{\Gamma(\alpha(pa_i^G)+N(pa_i^G))}\prod_{x_i}\frac{\Gamma(\alpha(x_i,pa_i^G)+N(x_i,pa_i^G))}{\Gamma(\alpha(x_i,pa_i^G))}$$

where $N(..)$ are the counts from the data and $\alpha(..)$ are the hyperparameters for each family given G.

To compute the given formula we need the prior counts $\alpha(..)$ for each network structure G. Since there are exponentially many structures, this is a difficult task.

MDL and the Bayesian score are both consistent and asymptotically equivalent. They also are score-equivalent: they assign the same score to equivalent networks.

Chapter 12

Learning parameters and an unknown structure with incomplete data

This chapter is similar to the previous chapter, but with the difference that now we have to learn the structure and parameters from incomplete data. This one is the hardest of the four learning problems.

12.1 MDL

To use the MDL score in case of incomplete data we need to find the maximum likelihood parameters in order to compute the mutual information and entropy. These maximum likelihood parameters have to be found, for each structure, using the methods mentioned in chapter 10, since the data is incomplete.

12.2 Bayesian score

In case of the Bayesian score it is not possible to evaluate the marginal likelihood, so we have to resort to approximations. We can use different approximations:

- Asymptotic approximations. This evaluates the score around MAP parameters that you can find using, for example, the EM algorithm.
- Stochastic approximations. These methods are usually much slower.

Usually a greedy search algorithm is used to make small changes in an initial structure to improve it. The search procedure adds and removes arcs to find a better structure. Why we do not evaluate all possible structures is explained in the next section.

12.3 The problem

The problem of these two approaches, MDL and Bayesian score is that they require to compute the optimal parameters for each structure candidate. Even when it turns out that the candidate is a low scoring one, we spend non-neglegible computation time calculating the optimal parameters. So in practice, such learning procedures are only feasible when we consider small sets of candidate structures.

Part IV

Refinement of BNs Using Cases

Chapter 13

The Expectation-Maximization Algorithm

We use the Expectation-Maximization (EM) algorithm [5] to refine a Bayesian network with cases. We will see how EM is capable of combining expert knowledge, in the form of a Bayesian network, and cases, in a principled way. We also take a look at SMILEARN and see how EM is implemented.

13.1 Definition

The Expectation-Maximization (EM) algorithm is actually a class of algorithms that can learn from datasets with missing data. In our context, the EM algorithm searches for maximum likelihood and MAP estimates of parameters in BNs, where the BN depends on unobserved latent variables. MAP estimates are much like maximum likelihood estimates, but the parameters have a prior distribution. EM alternates between two steps:

- Expectation: The expected values of the latent variables are calculated.
- Maximization: Computes the new maximum likelihood estimates of the parameters given the data and setting the latent variables to their expectation.

Let $\mathbf{E}$ be the known variables and $\mathbf{H}$ the unobserved latent variables. Together, $\mathbf{E}$ and $\mathbf{H}$ form the complete data. Assume that P is a joint model of the complete data with parameters θ: $P(\mathbf{E}, \mathbf{H}|\theta)$. EM will then iteratively improve the initial parameter setting θ_0 to θ_1 through θ_n until convergence. For the discrete case we get the following formulas for re-estimating θ:

$$\theta_{n+1} = \arg\max_{\theta} \sum_{h} P(\mathbf{H}|\mathbf{E}, \theta_n) \log P(\mathbf{E}, \mathbf{H}|\theta) \ .$$

And maximizing:

$$Q(\theta) = \sum_h P(\mathbf{H}|\mathbf{E}, \theta_n) \log P(\mathbf{E}, \mathbf{H}|\theta) \ .$$

Informally, the idea of EM is this. With complete data we were able to compute the optimal parameters, so what we try to do is complete the data by using inference. We have to take the following steps. First, we choose the initial parameters. We can initialize them randomly, or assign prior probabilities and assign confidence in those priors. In the next chapter we will explain how to use priors in EM. After initialization, inference is used to complete the data and compute the *expected* sufficient statistics. For every local probability distribution of a node the parents are initialized, and if the parent configuration consistent with the evidence in the network, the counts are updated by the posterior probability of the node multiplied by the probability of the instantiation of the parents. Based on the expected sufficient statistics we can update the parameters like we did in the case of complete data. We repeat this process until convergence.

EM will always lead to a local or global maximum, except for one case. The parameters will not converge if the initial parameters are exactly a local or global minimum, which is very unlikeli.

13.2 Structural EM

It is possible to make EM suitable to learn structure too, this is called Structural EM [10] [9]. The idea is that is each loop is either a normal parametric EM step or a structural EM step. So the Structural EM algorithm performs a search in the joint space of parameters and structure.

13.3 Learning Noisy-MAX

Noisy-MAX gates are not directly learnable using the EM algorithm. One approach to make them learnable is to decompose them to only CPTs, learn those CPTs, and convert back to noisy-MAX. More about this approach in part V.

13.4 Design of SMILEARN

In Figure 13.1 an important part of the design of the SMILEARN API is shown. The idea is that DSL_dataset contains all the data, and that all learning algorithms that are used, will take a DSL_dataset object as input. DSL_dataSource is an interface that defines all the methods that are needed to construct a DSL_dataset. DSL_casesDataSource implements the DSL_dataSource interface and in this way it is possible to build a DSL_dataset object that is based on cases. The cases are loaded directly from the network. Note that it is easy

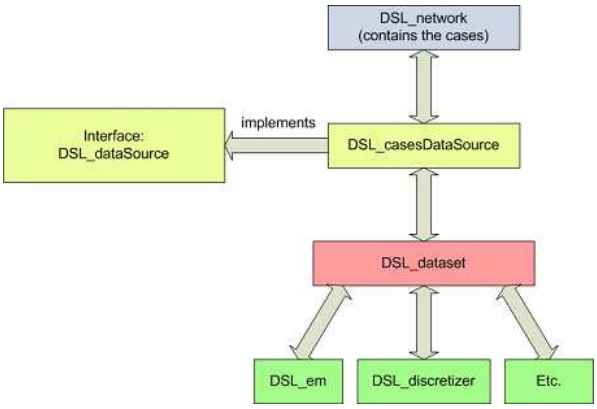

Figure 13.1: Class design in the SMILEARN API.

to create a DSL_fileDataSource or DSL_databaseDataSource if needed, just by implementing the DSL_dataSource interface.

13.5 SMILEARN and EM

Figure 13.2 sketches the design of EM in SMILEARN. We can distinguish four classes organized in an hierarchy: DSL_em, DSL_suffStats, DSL_suffStatsOfVar and DSL_configuration. At the top of the hierarchy is DSL_em and the method Learn can be called to execute the learning process. DSL_suffStats is nothing more then a container of DSL_suffStatsOfVar instances. DSL_suffStatsOfVar maintains the sufficient statistics of all the local probability distributions that are stored in DSL_configuration. So DSL_configuration is one local probability distribution and DSL_variable contains the local probability distributions of one variable. DSL_suffStatsOfVar updates the parameters in the network after one EM loop.

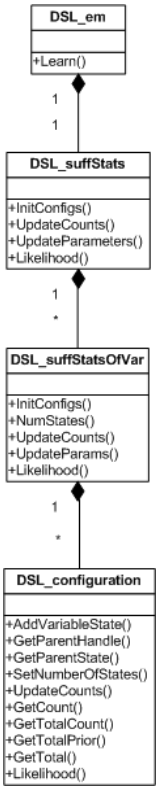

Figure 13.2: Overview of the classes and methods used to implement the EM algorithm.

Chapter 14

Priors and Hierarchical Confidence

There are different approaches of refining Bayesian networks with data [11]. Our approach, MAP parameters and hierarchical confidence are both subject matter of this chapter.

14.1 MAP Parameters

Priors in EM can be realized by using MAP parameters. Instead of calculating maximum likelihood parameters, we initialize all the local probability distribution by consulting an expert and assigning the confidence in each of the local probability distributions by a positive number. Note that this is not a Bayesian approach, because the parameters themselves are point-estimates and not random variables. The use of priors, however, is Bayesian like.

If we take the coin toss example again, the MAP parameters can be computed in this way:

$$P(X = h|\alpha_h, \alpha_t, N_h, N_t) = \frac{\alpha_h + N_h}{\alpha_h + N_h + \alpha_t + n_t} ,$$

with α_h and α_t as virtual count of heads and tails. N_h and N_t are the counts that can be extracted from the dataset. Now lets take a look at an example. Suppose an expert estimated the prior probability on $\alpha'_h = 0.5$ and $\alpha'_t = 0.5$, and the expert is not very confident, so say the expert based his estimate on $N' = 4$ virtual counts. In the dataset we observe the counts $N_h = 3$ and $N_t = 7$. We can compute the posterior like this:

$$P(X = h|\alpha_h = 2, \alpha_t = 2, N_h = 3, N_t = 7) = \frac{3 + 2}{3 + 2 + 7 + 2} = \frac{5}{14} \approx 0.36 .$$

If we take $N' = 40$, the posterior prediction will be:

$$\frac{3 + 20}{3 + 20 + 7 + 20} = \frac{23}{50} = 0.46 .$$

The data has a lot less influence here, because the expert is a lot more confident about his prediction. But if there is enough data, it washes away the prior. If we take as prior counts $N_h = 3000$ and $N_t = 7000$, the posterior predictions become:

$$\frac{3000 + 2}{3000 + 2 + 7000 + 2} \approx 0.3 \ ,$$

and

$$\frac{3000 + 20}{3000 + 20 + 7000 + 20} \approx 0.3 \ .$$

When the data is incomplete we have to do more computations. This example is not feasible, because we have only one variable. Suppose we have two magic coins that are causally related: if the first one lands heads, the probability that the other one lands heads increases. If we want to compute the count of N_{hh} with an incomplete dataset, we have to use inference to 'complete' the dataset. If of only one of the coins the value is known, we use inference to compute the expected value of the other coin. In this way we compute the expected sufficient statistics for the whole dataset and we can resort to updating the parameters like we did with complete data. Then we start the process again and continue until convergence, i.e., the parameters do not change anymore.

14.2 Hierarchical Confidence

The idea of hierarchical confidence is that experts can assign their confidence about each estimate on a high level and, if necessary, re-assign it on a lower level. This assignment on a lower level overwrites the assignment on the higher level. There are multiple levels in the hierarchy ranging from confidence in the network to confidence in a local probability distribution in a CPT. The virtual count given in the previous section is a simple example of an hierarchical confidence specification, and shows how the the hierarchical confidence can be used in EM.

Hierarchical confidence is nothing more than specifying for each distribution, in an organized manner, how fast the distributions will change when the network is updated with cases. High confidence means that the prior distributions change slowly when new data arrives. Low confidence means that the cases are relatively important, so they will change the prior distribution a lot, if necessary. When a lot of data is available, the confidence of the expert can be low, since the learning algorithm should be capable of estimating the parameters quite accurately. The other way around is also true: if not much data is available, the expert should be confident about a prior. Otherwise a few cases will change the local distributions in a way that is not desired.

In GeNIe it is possible to set the confidence by right clicking on an item and selecting 'Set confidence'. A popup appears where positive number can be entered. This information will be used in SMILEARN as confidence. There are three possible location to specify the confidence:

- On the network level.

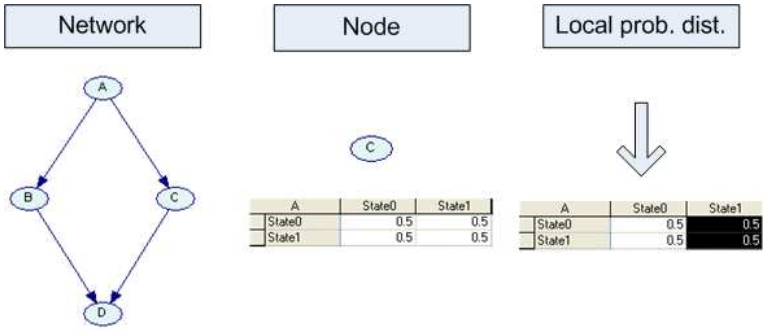

Figure 14.1: Different layers of hierchical confidence.

- On the node level.
- On the level of a local probability distribution.

See also Figure 14.1. If the confidence on a lower level is not specified, the confidence of the level above is taken. This means that the confidence of the network always has to be specified. Specifying is convenient for the expert: if the confidence on the network level is not good enough for a node, the expert only has to change the confidence in that node.

14.3 Implementation in SMILE

The incorperation of priors confidence in SMILE is easy, because it is a small extension to the EM algorithm. Instead of randomizing the initial parameters, we use the prior parameters given by the expert. It works exactly like the coin toss example of MAP learning in section 14.1, but then for more than one variable.

Part V

Exploiting Decomposable Causal Independence

Chapter 15

Introduction to Decomposable Causal Independence

As we said before, specifying CPTs requires a lot of parameters. This is either overwhelming for an expert or leads to inferior quality of parameters learned from small data sets [21]. The most popular proposal that addresses this problem is the noisy-OR model (and its extension to multi-valued nodes, the noisy-MAX model) [15] for local probability distributions. The noisy-OR model, successfully adopted in numerous practical applications, reduces number of parameters required to specify the CPT from exponential to linear in number of parents, achieving it at the expense of making some assumptions about influences of the parent variables on the child variable. The noisy-OR model is not only capable of reducing the number of parameters in the model, but by exploiting its decomposability property it has been proven to improve the efficiency of belief updating, making some intractable models tractable [12].

15.1 Related Work

In the past, some researchers tried to exploit models for local probability distributions in BN. Meek and Heckerman [20] experimented with learning various causal independence models for both parameter and structure learning, but they did not explicitly exploited the decomposability property. Dagum and Galper [22] attempted to introduce linear models for modeling local probability distributions. However, their approach differs from the average model that will be proposed here. Our decompositions preserve independence of parameters and allow for the use of the EM algorithm.

15.2 New Models

We propose new models for local probability distributions that exploit decomposable causal independence. The new models release some assumptions of causal independence to achieve greater flexibility at the cost of clarity of parameters interpretation. We present the results of an empirical study based on Bayesian networks used in practice that show that the new models can be suitable for learning Bayesian networks from small data sets.

15.3 Fewer Parameters

We propose models for local probability distributions that require fewer parameters than CPTs (especially for nodes with a large number of parents) and are flexible to sufficiently approximate arbitrary patterns of interactions among causes in producing the effect. We believe that such models can be especially useful in cases when there is an insufficient amount of data to reliably learn local probability distributions in BNs, which is one of the bottlenecks in learning BN models from data. Also, we introduce alternatives to the noisy-OR model that allow for capturing a wider range of interactions between parent variables than noisy-OR. It is achieved at the expense of clarity of parameters in our model — unlike the noisy-OR model, the models proposed here do not have parameters expressed by means of conditional probabilities of variables explicitly included in the model (the amechanistic property [12]).

15.4 Emperical Study

We present an empirical study that shows the performance of learning parameters of our models using the EM algorithm [5]. We will not use priors and hierarchical confidence like we did in the case management system, but we will compute the maximum likelihood parameters. The results indicate that the proposed models can provide better approximations of local probability distributions than CPTs for cases when the number of data records is relatively small compared to the number of parameters in the CPT. We conclude that the models presented here can be a practical alternative to learning full CPTs, and they are especially suitable for situations where variables have many parents in the model, as both learning from data and inference can be improved.

Chapter 16

Local Distributions in Bayesian Networks

In this chapter, we briefly introduce the noisy-OR model and subsequently use it to explain the foundations of causal independence (CI) models. Let us consider a variable in a BN model and its parents. We will denote the child variable as Y and its n parents, typically referred to as causes, by $\mathbf{X} = \{X_1, \ldots, X_n\}$. We assume that a parent X_i takes m_i states and the child variable Y takes m_y states. The CPT for node Y specifies a conditional probability distribution $P(Y|\mathbf{X})$, and it consists of $\prod_{i=1}^{n} m_i$ distributions.

16.1 Noisy-OR

One of the approaches to address the problem of a large number of parameters required to specify a CPT is the noisy-OR model. It requires a number of parameters that is linear in the number of parents. It is achieved by the assumption that the causes $X_1, \ldots, X_n$ act independently in producing the effect on Y. This assumption can be represented explicitly by the BN presented in Figure 16.1(a). This is the decomposed noisy-OR that also used by the EM algorithm for learning. When the EM algorithm is done, the decomposed version is transformed back to noisy-OR (the same holds for noisy-MAX). The variables X_i and Y represent the parent and effect variable in the model, while the variables M_i are hidden and represent the influence of each parent on the effect variable considered separately. In this representation, variable Y is a deterministic variable (denoted by double circle). This means that its CPT consists of only 0s and 1s. We will refer to the function defined by Y as *combination function* — it combines individual influences of each parent defined by hidden mechanism variables. In case of noisy-OR, the combination function is simply the deterministic OR relation. For noisy-MAX it is the MAX relation defined over the states of Y.

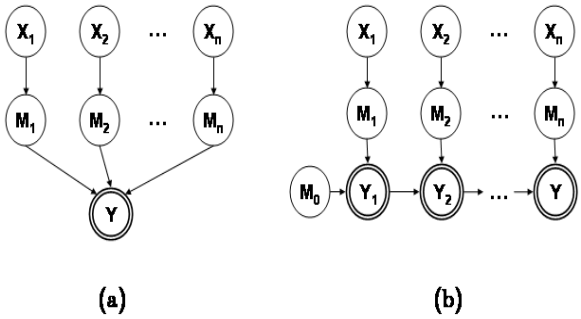

Figure 16.1: Decomposition of the causal independence models: (a) decomposition into mechanisms and deterministic combination function, (b) decomposition into mechanism and decomposable combination function.

16.2 Decomposable Independence Models

If the combination function can be decomposed into a series of functions, the causal independence model is said to be *decomposable*. It basically says that the combination function can be decomposed into series of binary functions. For example, the logical OR relation of n inputs can be decomposed into series of pairwise OR relations:

$$OR(X_1, X_2, \ldots, X_n) = OR(X_n, OR(X_{n-1}, OR(\ldots OR(X_2, X_1)\ldots)) \ .$$

The general decomposable CI model is shown in Figure 16.1(b). The decomposable property can lead to significant improvements of inference performance for the CI models [12] by reducing clique sizes in the join tree algorithm. This property has been exploited in more sophisticated algorithms for inference with CI models [6, 27].

Although the causal independence is capable of defining multiple potentially interesting models, in the literature only models that have OR, AND, XOR, MAX, and MIN as the combination function are discussed.

16.3 Alternative Models

In this thesis we propose other models that do not strictly belong to this family, but draw ideas from it. In general, our models are less restrictive in terms of made assumptions, but it comes mainly at the expense of knowledge elicitation. The models mentioned earlier have parameterizations that are convenient for asking experts (in terms of conditional probabilities of variables in the models, without the need of explicit mentioning the hidden mechanisms variables). Our models are parameterized in term of hidden variables, making them more difficult to work with for human experts. Since the parameters of CI models are represented as parameters of regular, but hidden, variables, the EM algorithm

can learn the parameters of the decomposition nodes in the same manner as it would learn parameters for any other hidden variable or missing value.

Chapter 17

Ladder Decompositions and the Average Model

In this section we introduce new models for local probability distributions. We propose new models that are related to the decomposable CI models. The main difference is that we release the assumption that the combination function is a deterministic function and allows CPTs of nodes Y_i to take values different from 0 or 1. We propose 3 different decompositions.

17.1 The Compositions

The first decomposition, *ladder with mechanisms* (LM), presented in Figure 17.1(a) is basically a generalization of the decomposable CI defined by Heckerman and Breese [12]. The difference is that we do not impose any constraints on the combination function — the parameters in CPTs of nodes M_i and Y_i can take arbitrary values. The second model, a *simple ladder* (SL), presented in Figure 17.1, differs from the previous in that that it does not have mechanism nodes, only nodes Y_i that define pairwise interactions between parent i and cumulative influence of previous parents.

Finally, the third decomposition is a model where CPTs of mechanism nodes can take arbitrary values, but CPTs of nodes Y_i have predefined values, such that the probability of node Y is defined as

$$P(Y = y|M_1, \ldots, M_n) = \frac{1}{n} \sum_{i=1}^{n} I(M_i = y) \ ,$$

where I is the identity function that takes 1 when the condition in the brackets is true, and 0 otherwise. This can be achieved when each parameter of CPT of node Y_i $(i = 2, \ldots, n)$ is defined as:

$$P(Y_i = y|M_{i-1} = a, M_i = b) = \frac{1}{i} I(y = a) + \frac{i-1}{i} I(y = a) \ .$$

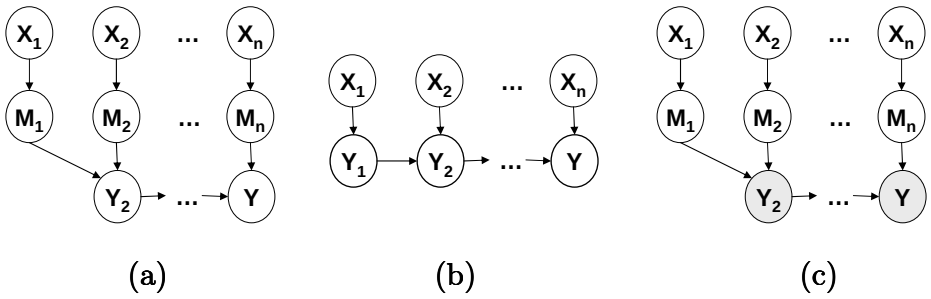

Figure 17.1: Models used in experiments: (a) ladder with mechanisms, (b) simple ladder, and (c) the average model.

Table 17.1: Number of parameters for different decompositions.

Decomposition	Number of parameters
CPT	$\prod_{i=1}^{n} m_i$
LM	$m_y \sum_{i=1}^{n} m_i + (n-1)m_y^3$
SL	$m_y m_1 + m_y^2 \sum_{i=2}^{n} m_i$
Average	$m_y \sum_{i=1}^{n} m_i$
Noisy-MAX	$m_y \sum_{i=1}^{n} (m_i - 1)$

17.2 Characteristics

These three decompositions have different characteristics. The number of parameters required to specify relations between causes and effect in the first two models are shown in Table 17.1. LM should be more suitable for situations where the child has a small number of states (as m_y^3 is the dominating factor), and SL in situations where some parents have a small number of states (the sum of parents states is multiplied by m_y^2). Finally, the average model is a step toward more meaningful parameters, which can be utilized by human knowledge engineers. The parameters of this model are expressed in terms of mechanisms — separate influences of a parent on the effect, and therefore they have meaning in the modeled domain. The combination function is the average number of instantiations of mechanism variables. Such a setting has one important advantage over models like noisy-MAX — it does not require additional semantic knowledge about the values (noisy-MAX assumes an ordering relation) and therefore can be easily applied to learning algorithms, as well as it is more flexible in terms of modeling. But most of all, it preserves the decomposability property, which can be exploited by inference algorithms.

17.3 Noisy-MAX Learning

In order to do noisy-MAX learning, we need to determine what the distinguished states are. For this, we used a simple approximate algorithm to find both the distinguished states of the parents and the child. The selection of distinguished states is based on counting the occurrences of parent-child combinations N_{ij}, where i is the child state and j is the parent state. The next step is to normalize the child states for each parent: $N^*_{ij} = \frac{N_{ij}}{\sum_i N_{ij}}$.

Child state i and parent state j are good distinguished state candidates if N^*_{ij} has a relatively high value. But we have to account for the fact that one child can have multiple parents, so we have to combine the results for each of the parents to determine the distinguished state of the child. For each parent, we select the maximum value of the state of a parent given the child state. We take the average of one of the child states over all the parents. The child state corresponding to the highest value of the average child states values is considered to be the child's distinguished state. Now that we have the child's distinguished state it is possible to find the parents' distinguished states in a similar way.

Chapter 18

Empirical Results

To test our models we performed series of empirical studies. We decided to use existing BN models to generate data and then re-learn the parameters of these models using the new models.

18.1 Experimental setting

For our experiments we used three networks available to the general public at `http://www.sis.pitt.edu/~genie`: Alarm [1], Hailfinder [2] and Hepar II [21]. From each of these networks, we selected the nodes whose total number of parameters in CPT exceeded the number of parameters for each of the decompositions or the number of parameters in CPTs was more than 100. Table 18.1 shows a summary of the most important characteristics of these models. We applied the three decompositions and obtained models that had similar structure to the original model, but that had nodes with large CPTs decomposed. To evaluate the learning performance we generated c data records from the original model (used as the gold standard) using probabilistic logic sampling. We applied the EM algorithm on the models using the generated cases. As a measure of the quality of fit we used the Hellinger distance [16] between the two posterior probabilities for a decomposed node and the corresponding node in the gold standard model. The Hellinger distance between two probability distributions F and G: $D_H(F,G) = \left[\sum(\sqrt{f_i} - \sqrt{g_i})^2\right]^{1/2}$. We decided to use the Hellinger distance because, unlike the Euclidean distance, it is more sensitive to differences in small probabilities and it does not poses difficulties for 0 probabilities as it is the case for Kullback-Leibler divergence.

18.2 Results

We repeated the procedure described above to all three decompositions, and in addition to this noisy-MAX and CPT without decomposition, 25 times. This

Table 18.1: Characteristics of nodes used in the empirical study

Model	Node	Number of Parameters				
Alarm	node33	108	22	40	46	14
Hepar	bilirubin	288	48	168	304	28
Hepar	phosphatase	108	30	78	111	18
Hepar	inr	108	30	72	111	18
Hepar	bleeding	24	14	20	22	10
Hepar	alt	288	48	156	304	28
Hepar	ast	288	48	156	304	28
Hepar	ggtp	384	52	184	372	28
Hailfinder	CombVerMo	256	48	144	176	36
Hailfinder	CldShadeOth	144	33	75	87	24
Hailfinder	Boundaries	108	30	72	84	21
Hailfinder	CompPlFcst	324	39	99	120	27
Hailfinder	AMInsWliScen	216	39	81	93	30
Hailfinder	PlainsFcst	1188	63	165	144	51

helped us to ensure that initial randomization of parameters for EM does not affect our results significantly. We varied the number of data records c between 50 and 500 for each of the three networks. The results are reported in Table 18.3.

The results of our experiments show that the decompositions perform comparatively similar and in some cases better than learning CPTs. We find this result significant, because decompositions can be exploited in other manners, for example in knowledge elicitation and inference. The results of our experiments suggest that none of the decompositions itself is the best. However, it is apparent that the proposed models together with noisy-MAX can outperform CPTs. Interestingly, the results for the largest CPT under our study (PlainsFcst) show the trend that the decompositions can provide a better accuracy when the number of cases is small. One possible explanation is that the CPT with its large number of parameters can easily overfit data, while models with smaller number of parameters can reasonably well approximate the real underlying distribution. Obviously, one should expect that when the number of data records is sufficiently large, CPTs should ultimately outperform the more restrictive models.

[1]Results for Hepar with 500 data records are based on the average of 12 iterations.

Table 18.2: Results of the empirical study

Node	Records	CPT	Average	LM	SL	Noisy-MAX
node33	50	0.0460	0.0578	0.0384	**0.0374**	0.0398
	100	0.0311	0.0708	**0.0273**	0.0278	0.0302
	200	**0.0195**	0.0533	0.0209	0.0211	0.0233
	500	**0.0117**	0.0452	0.0117	0.0119	0.0138
bilirubin [1]	50	**0.0537**	0.0655	0.0618	0.0628	0.0637
	100	**0.0446**	0.0469	0.0468	0.0464	0.0455
	200	**0.0263**	0.0289	0.0264	0.0270	0.0269
	500	0.0189	0.0190	**0.0171**	0.0172	0.0174
phosphatase [1]	50	**0.0512**	0.0539	0.0558	0.0541	0.0530
	100	**0.0366**	0.0385	0.0383	0.0380	0.0373
	200	0.0274	0.0275	0.0281	**0.0274**	0.0275
	500	**0.0147**	0.0154	0.0151	0.0154	0.0156
inr [1]	50	**0.0480**	0.0587	0.0585	0.0568	0.0555
	100	**0.0342**	0.0369	0.0355	0.0365	0.0370
	200	0.0242	0.0260	0.0249	0.0242	**0.0240**
	500	0.0164	**0.0148**	0.0148	0.0149	0.0149
ggtp [1]	50	**0.0644**	0.0738	0.0723	0.0695	0.0687
	100	0.0466	**0.0456**	0.0476	0.0460	0.0478
	200	0.0299	0.0321	**0.0298**	0.0310	0.0308
	500	0.0204	0.0204	0.0201	**0.0200**	0.0202
alt [1]	50	**0.0457**	0.0508	0.0502	0.0491	0.0499
	100	**0.0361**	0.0387	0.0379	0.0379	0.0383
	200	0.0317	**0.0303**	0.0308	0.0308	0.0308
	500	**0.0196**	0.0205	0.0206	0.0207	0.0207
ast [1]	50	**0.0537**	0.0615	0.0649	0.0589	0.0597
	100	**0.0372**	0.0465	0.0477	0.0472	0.0454
	200	0.0294	0.0283	0.0283	**0.0281**	0.0283
	500	**0.0182**	0.0184	0.0186	0.0186	0.0186
CombVerMo	50	0.0534	0.0645	0.0712	0.0685	**0.0534**
	100	**0.0402**	0.0440	0.0411	0.0443	0.0426
	200	**0.0333**	0.0415	0.0353	0.0358	0.0372
	500	**0.0153**	0.0277	0.0163	0.0162	0.0219
CldShadeOth	50	**0.0512**	0.0798	0.0638	0.0655	0.0553
	100	0.0370	0.0609	**0.0348**	0.0373	0.0399
	200	0.0284	0.0610	**0.0266**	0.0280	0.0291
	500	0.0185	0.0510	0.0186	0.0189	**0.0177**
Boundaries	50	0.0604	0.0666	0.0619	0.0636	**0.0583**
	100	0.0448	0.0502	0.0455	0.0444	**0.0432**
	200	**0.0236**	0.0297	0.0243	0.0237	0.0248
	500	0.0170	0.0266	0.0166	0.0160	**0.0218**
CompPlFcst	50	0.0438	0.0544	0.0531	0.0557	**0.0421**
	100	**0.0332**	0.0408	0.0396	0.0415	0.0384
	200	0.0260	0.0258	0.0257	**0.0231**	0.0239
	500	0.0212	0.0207	**0.0203**	0.0207	0.0208
AMInsWliScen	50	0.0479	0.0579	0.0571	0.0574	**0.0479**
	100	**0.0312**	0.0364	0.0328	0.0316	0.0320
	200	**0.0292**	0.0302	0.0301	0.0307	0.0295
	500	0.0192	**0.0190**	0.0173	0.0172	0.0176
PlainsFcst	50	0.1404	**0.0562**	0.0578	0.0563	0.1205
	100	0.1165	0.0333	0.0392	**0.0329**	0.0882
	200	0.0874	0.0227	0.0239	0.0224	**0.0223**
	500	0.0403	0.0167	**0.0165**	0.0174	0.0178

Table 18.3: Results of the empirical study

Part VI

Conclusions and Future Work

Chapter 19

Conclusions and Summary

In this chapter we will conclude and summarize the work that has been done.

19.1 Case Management

The case management system has been developed to make maintaining cases for users possible, and on top of that, use the cases for learning. However, other applications of the case management can be thought of. We saw that case management systems are not available in comparable software. The case management system consists of two parts, one part is implemented in GeNIe, and one part is implemented in SMILE. Methods in SMILE are available to add and remove cases, and it is also possible to edit the attributes, for example, the evidence and target nodes. The cases are tightly connected to a network in the sense that the variables match, and if changes occur in a the network, the cases are updated. Because of this we store the cases in the same XDSL file where the network is also stored. The GeNIe part is the graphical user interface layer build upon the SMILE part. It forwards all the user request made in the case management screens to SMILE.

19.2 EM and Hierarchical Confidence

After giving an overview of the learning theory, we explained our approach to refining Bayesian networks. By acquiring prior probabilities and the confidence of the expert in those priors, we used the EM algorithm to refine a network using the MAP approach. To get the confidence from the experts we introduced the so-called hierchical confidence. Basically this is a way for domain expert to express their confidence in the network they developed, by assigning confidence on different levels of granularity. It expresses how certain the knowledge engineer is about the network he constructed. The cases serve as data that is used to refine the network, and taken as input for the EM algorithm. By learning the MAP parameters, we were able to combine expert knowledge and data. All

of this is implemented in SMILEARN, the new learning API, and EM and hierarchical confidence are part of that API.

The case management system and EM algorithm with hierarchical confidence are not released yet. We hope to do this in the short future to get feedback from the users. However, we gave a demonstration of a prototype of the case management system at Intel, and they were enthousiastic. They have some models designed by domain experts, but they usually have only a few cases available to change the probability distibutions.

19.3 Exploiting Decomposable Causal Independence

We introduced three decompositions of conditional probability tables. The Average decomposition drew ideas from both the linear models and causal independence models. The two remaining decompositions were basically generalizations of the decomposable CI models with some constraints removed. We showed that these decompositions can be exploited in learning large conditional probability distributions for BNs, especially when the number of data records is small. Additional known benefits of CI models with the decomposability property is improvement in performance of inference algorithms, and the proposed models can be exploited in the same manner. The advantages of the decompositions should be more apparent in nodes with a large number of parents.

We believe that our work has significant potential applications in practice. It is often the case that nodes in practical models may have more than 10 parents. Learning parameters of such large CPTs requires volumes of data. Models containing nodes with large in-degrees pose a challenge to inference algorithms. The decompositions proposed here can address two problems at the same time, while providing a reasonably good approximation of the underlying distributions.

In our empirical study, we used three sizeable real-life models and tried to approximate selected large CPTs from these networks using the new models and CPT and the noisy-MAX. In general, we found that none of the methods is superior to others, and particular characteristics of distributions in the CPT are an important factor. However, the decompositions seem to perform better when the number of data records is small.

Chapter 20

Future work

In this chapter we shortly discuss the future work.

20.1 Case Management

The most important future work is to release the new software. In this way we hope to receive a lot feedback and make changes if necessary. Since the case management system is designed very generic, it is extendable and in the future more applications can be implemented using the case management. It was quite easy to use the cases as input for the EM algorithm.

20.2 SMILEARN and EM

The new SMILEARN API will be released when we think it is ready. The SMILEARN API adds a lot of new possibilities on the learning part. It has been designed with extensibility in mind, and in the future we want to extent it with alternative learning methods. Users of the API can select the learning method that is appropriate for the task at hand. One obvious extension is structural EM, but a problem is how to limit the structure candidates to a number that can be run in reasonable time. An interesting feature for users would be the possibility to compare two networks with the same data collection, but with different structures. In this way, a domain expert can experiment with improvements of an existing network, and see if the changes in the network increase the performace.

20.3 Exploiting Decomposable Causal Independence

Based on the current results, we envision the following application of this work. For each (large) CPT in a model one can try to learn parameters of different

models and use the likelihood of the data for the decomposed node to see which decomposition is most promising (preferably using learning and testing data sets to avoid over-fitting). We performed a pilot study on this method and the results in terms of the likelihood function were similar to the goodness of fit measure used to compare the gold standard models.

Appendix A

Exploiting Decomposable Causal Independence for Parameter Learning in Bayesian Networks

This appendix contains the original paper (but not in the original layout) I made together with Adam Zagorecki and Marek J. Druzdzel.

Abstract

Bayesian networks are a successful modeling tool that facilitates a convenient combination of expert knowledge and data. One of their major bottlenecks is the number of parameters required to specify local probability distributions, which grows exponentially in the number of parents of a node in the underlying graph. One widely used solution to this problem is assuming independence of causal interactions that reduces the number of parameters to linear in node's in-degree. In this paper we propose new models for local probability distributions that exploit decomposable causal independence. The new models release some assumptions of causal independence to achieve greater flexibility at the cost of clarity of parameters interpretation. We present the results of an empirical study based on Bayesian networks used in practice that show that the new models can be suitable for learning Bayesian networks from small data sets.

A.1 Introduction

Bayesian networks (BNs) [25] have become a prominent modeling tool for problems involving uncertainty in the last 20 years. Among a wide range of their practical applications are medical diagnosis, hardware troubleshooting, user

modeling, intrusion detection, disease outbreak detection, etc. The BN framework combines strong formal foundations of probability theory with a graphical representation of interactions among variables, providing a formalism that is theoretically sound, yet readily understandable for human knowledge engineers and fairly easy to apply in practice. BNs allow for convenient and flexible fusion of information from various sources, in particular combining expert knowledge with available data. This has proved to be critical in practical applications, where data typically is limited.

A BN is basically a compact representation of a joint probability distribution (JPD). It reduces the number of parameters required to specify the JPD by exploiting independencies among domain variables. These independencies are encoded in the graphical part, and the JPD is specified by means of local probability distributions associated with vertices (variables). In case of discrete variables (this is what we will focus on in this paper), the local probability distributions are encoded in the form of prior probabilities over those vertices that have no parents in the graph, and conditional probability tables (CPTs) for all other nodes. The conditional probability table is a set of conditional probability distributions that define a probability distribution over the child variable given all combinations of values of the parents nodes. This leads to one of the major bottlenecks in building BN models — exponential growth of the size of CPTs in the number of parent variables. For example, assuming that all variables are binary, a CPT of variable with ten parents requires specification of $2^{10} = 1,024$ parameters. Adding one more parent doubles this size to 2,048. This is either overwhelming for an expert or leads to inferior quality of parameters learned from small data sets [21]. The most popular proposal that addresses this problem is the noisy-OR model (and its extension to multi-valued nodes, the noisy-MAX model) [15] for local probability distributions. The noisy-OR model, successfully adopted in numerous practical applications, reduces number of parameters required to specify the CPT from exponential to linear in number of parents, achieving it at the expense of making some assumptions about influences of the parent variables on the child variable. The noisy-OR model is not only capable of reducing the number of parameters in the model, but by exploiting its decomposability property it has been proven to improve the efficiency of belief updating, making some intractable models tractable [12].

In this paper we propose models for local probability distributions that require fewer parameters than CPTs (especially for nodes with a large number of parents) and are flexible to sufficiently approximate arbitrary patterns of interactions among causes in producing the effect. We believe that such models can especially useful in cases when there is an insufficient amount of data to reliably learn local probability distributions in BNs, which is one of the bottlenecks in learning BN models from data. In this paper, we introduce alternatives to the noisy-OR model that allow for capturing a wider range of interactions between parent variables than noisy-OR. It is achieved at the expense of clarity of parameters in our model — unlike the noisy-OR model, the models proposed here do not have parameters expressed by means of conditional probabilities of variables explicitly included in the model (the amechanistic property [12]). We

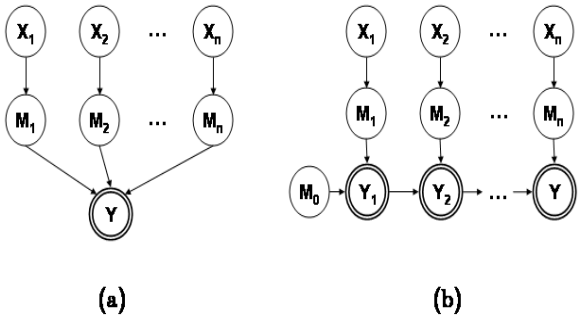

Figure A.1: Decomposition of the causal independence models: (a) decomposition into mechanisms and deterministic combination function, (b) decomposition into mechanism and decomposable combination function.

present an empirical study that shows the performance of learning parameters of our models using the standard EM algorithm [5]. The results indicate that the proposed models can provide better approximations of local probability distributions than CPT for cases when number of data records is relatively small compared to the number of parameters in the CPT. We conclude that the models presented here can be a practical alternative to learning full CPTs, and they are especially suitable for situations where variables have many parents in the model, as both learning from data and inference can be improved.

The remainder of the paper is structured as follows. In Section A.2, we discuss the noisy-OR model and the causal independence family of models. In Section A.3, we formally introduce the new models. We present empirical results in Section A.4 and our conclusions along with future work in Section A.5.

A.2 Local Distributions in Bayesian Networks

In this section, we briefly introduce the noisy-OR model and subsequently use it to explain the foundations of causal independence (CI) models. Let us consider a variable in a BN model and its parents. We will denote the child variable as Y and its n parents, typically referred to as causes, by $\mathbf{X} = \{X_1, \ldots, X_n\}$. We assume that a parent X_i takes m_i states and the child variable Y takes m_y states. The CPT for node Y specifies a conditional probability distribution $P(Y|\mathbf{X})$, and it consists of $\prod_{i=1}^{n} m_i$ distributions.

One of the approaches to address the problem of a large number of parameters required to specify a CPT is the noisy-OR model. It requires only a linear number of parameters in the number of parents. It is achieved by the assumption that the causes $X_1, \ldots, X_n$ act independently in producing the effect on Y. This assumption can be represented explicitly by the BN presented in Figure A.1(a). The variables X_i and Y represent the parent and effect variable in the model, while the variables M_i are hidden and represent the influence of each parent on the effect variable considered separately. In this representation

variable Y is a deterministic variable (denoted by double circle). This means that its CPT consists of only 0s and 1s. We will refer to the function defined by Y as *combination function* — it combines individual influences of each parent defined by hidden mechanism variables. In case of noisy-OR, the combination function is simply the deterministic OR relation. For noisy-MAX it is the MAX relation defined over the states of Y.

If the combination function can be decomposed into a series of functions, the causal independence model is said to be *decomposable*. It basically says that the combination function can be decomposed into series of binary functions. For example, the logical OR relation of n inputs can be decomposed into series of pairwise OR relations:

$$OR(X_1, X_2, \ldots, X_n) = OR(X_n, OR(X_{n-1}, OR(\ldots OR(X_2, X_1)\ldots)).$$

The general decomposable CI model is shown in Figure A.1(b). The decomposable property can lead to significant improvements of inference performance for the CI models [12] by reducing clique sizes in the join tree algorithm. This property has been exploited in more sophisticated algorithms for inference with CI models [6, 27].

In this paper we propose other models that do not strictly belong to this family, but draw ideas from it. In general, our models are less restrictive in terms of made assumptions, but it comes mainly at the expense of knowledge elicitation. The models mentioned earlier have parameterizations that are convenient for asking experts (in terms of conditional probabilities of variables in the models, without the need of explicit mentioning the hidden mechanisms variables). Our models are parameterized in term of hidden variables, making them more difficult to work with for human experts. Since the parameters of CI models are represented as parameters of regular, but hidden, variables, the EM algorithm can learn the parameters of the decomposition nodes in the same manner as it would learn parameters for any other hidden variable or missing value.

A.3 Ladder Decompositions and the Average Model

In this section we introduce new models for local probability distributions. We propose new models that are related to the decomposable CI models. The main difference is that we release the assumption that the combination function is a deterministic function and allows CPTs of nodes Y_i to take values different from 0 or 1. We propose 3 different decompositions. The first decomposition, *ladder with mechanisms* (LM), presented in Figure A.2(a) is basically a generalization of the decomposable CI defined by Heckerman and Breese [12]. The difference is that we do not impose any constraints on the combination function — the parameters in CPTs of nodes M_i and Y_i can take arbitrary values. The second model, a *simple ladder* (SL), presented in Figure A.2, differs from the previous in

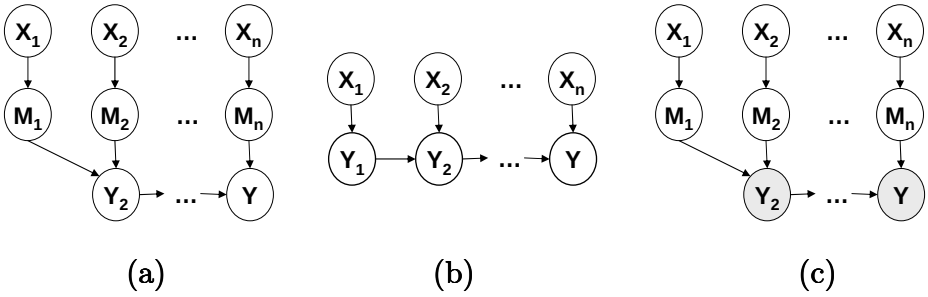

Figure A.2: Models used in experiments: (a) ladder with mechanisms, (b) simple ladder, and (c) the average model.

that that it does not have mechanism nodes, only nodes Y_i that define pairwise interactions between parent i and cumulative influence of previous parents.

Finally, the third decomposition is a model where CPTs of mechanism nodes can take arbitrary values, but CPTs of nodes Y_i have predefined values, such that the probability of node Y is defined as

$$P(Y = y|M_1, \ldots, M_n) = \frac{1}{n} \sum_{i=1}^{n} I(M_i = y) \ ,$$

where I is the identity function that takes 1 when the condition in the brackets is true, and 0 otherwise. This can be achieved when each parameter of CPT of node Y_i $(i = 2, \ldots, n)$ is defined as:

$$P(Y_i = y|M_{i-1} = a, M_i = b) = \frac{1}{i} I(y = a) + \frac{i-1}{i} I(y = a) \ .$$

These three decompositions have different characteristics. The number of parameters required to specify relations between causes and effect in the first two models are shown in Table A.1. LM should be more suitable for situations where the child has a small number of states (as m_y^3 is the dominating factor), and SL in situations where some parents have a small number of states (the sum of parents states is multiplied by m_y^2). Finally, the average model is a step toward more meaningful parameters, which can be utilized by human knowledge engineers. The parameters of this model are expressed in terms of mechanisms — separate influences of a parent on the effect, and therefore they have meaning in the modeled domain. The combination function is the average number of instantiations of mechanism variables. Such a setting has one important advantage over models like noisy-MAX — it does not require additional semantic knowledge about the values (noisy-MAX assumes an ordering relation) and therefore can be easily applied to learning algorithms, as well as it is more flexible in terms of modeling. But most of all, it preserves the decomposability property, which can be exploited by inference algorithms.

In order to do noisy-MAX learning we need to determine what the distinguished states are. For this, we used a simple approximate algorithm to find

Decomposition	Number of parameters
CPT	$\prod_{i=1}^{n} m_i$
LM	$m_y \sum_{i=1}^{n} m_i + (n-1)m_y^3$
SL	$m_y m_1 + m_y^2 \sum_{i=2}^{n} m_i$
Average	$m_y \sum_{i=1}^{n} m_i$
Noisy-MAX	$m_y \sum_{i=1}^{n} (m_i - 1)$

Table A.1: Number of parameters for different decompositions.

Model	Node	Number of Parameters				
Alarm	node33	108	22	40	46	14
Hepar	bilirubin	288	48	168	304	28
Hepar	phosphatase	108	30	78	111	18
Hepar	inr	108	30	72	111	18
Hepar	bleeding	24	14	20	22	10
Hepar	alt	288	48	156	304	28
Hepar	ast	288	48	156	304	28
Hepar	ggtp	384	52	184	372	28
Hailfinder	CombVerMo	256	48	144	176	36
Hailfinder	CldShadeOth	144	33	75	87	24
Hailfinder	Boundaries	108	30	72	84	21
Hailfinder	CompPlFcst	324	39	99	120	27
Hailfinder	AMInsWliScen	216	39	81	93	30
Hailfinder	PlainsFcst	1188	63	165	144	51

Table A.2: Characteristics of nodes used in the empirical study

both the distinguished states of the parents and the child. The selection of distinguished states is based on counting the occurrences of parent-child combinations N_{ij}, where i is the child state and j is the parent state. The next step is to normalize the child states for each parent: $N_{ij}^* = \frac{N_{ij}}{\sum_i N_{ij}}$.

Child state i and parent state j are good distinguished state candidates if N_{ij}^* has a relatively high value. But we have to account for the fact that one child can have multiple parents, so we have to combine the results for each of the parents to determine the distinguished state of the child. For each parent, we select the maximum value of the state of a parent given the child state. We take the average of one of the child states over all the parents. The child state corresponding to the highest value of the average child states values is considered to be the child's distinguished state. Now that we have the child's distinguished state it is possible to find the parents' distinguished states in a similar way.

A.4 Empirical Results

To test our models we performed series of empirical studies. We decided to use existing BN models to generate data and then re-learn the parameters of these models using the new models. For our experiments we used three networks available to the general public at `http://www.sis.pitt.edu/~genie`: ALARM [1], HAILFINDER [2] and HEPAR II [21]. From each of these networks we selected the nodes whose total number of parameters in CPT exceeded the number of parameters for each of the decompositions or the number of parameters in CPTs was more than 100. Table A.2 shows a summary of the most important characteristics of these models. We applied the three decompositions and obtained models that had similar structure to the original model, but that had nodes with large CPTs decomposed. To evaluate the learning performance we generated c data records from the original model (used as the gold standard) using probabilistic logic sampling. We applied the EM algorithm on the models using the generated cases. As a measure of the quality of fit we used the Hellinger distance between the two posterior probabilities for a decomposed node and the corresponding node in the gold standard model. The Hellinger distance between two probability distributions F and G: $D_H(F,G) = \left[\sum(\sqrt{f_i} - \sqrt{g_i})^2\right]^{1/2}$. We decided to use the Hellinger distance because, unlike the Euclidean distance, it is more sensitive to differences in small probabilities and it does not poses difficulties for 0 probabilities as it is the case for Kullback-Leibler divergence.

We repeated the procedure described above to all three decompositions, and in addition to this noisy-MAX and CPT without decomposition, 25 times. This helped us to ensure that initial randomization of parameters for EM does not affect our results significantly. We varied the number of data records c between 50 and 500 for each of three networks. The results are reported in Table A.3.

The results of our experiments show that the decompositions perform comparatively similar and in some cases better than learning CPTs. We find this result significant, because decompositions can be exploited in other manners, for example in knowledge elicitation and inference. The results of our experiments suggest that none of the decompositions itself is the best. However, it is apparent that the proposed models together with noisy-MAX can outperform CPTs. Interestingly, the results for the largest CPT under our study (PlainsFcst) show the trend that the decompositions can provide a better accuracy when the number of cases is small. One possible explanation is that the CPT with its large number of parameters can easily overfit data, while models with smaller number of parameters can reasonably well approximate the real underlying distribution. Obviously, one should expect that when number of data records is sufficiently large, CPTs should ultimately outperform the more restrictive models.

[1] Results for Hepar with 500 data records are based on the average of 12 iterations.

Node	Records	CPT	Average	LM	SL	Noisy-MAX
node33	50	0.0460	0.0578	0.0384	**0.0374**	0.0398
	100	0.0311	0.0708	**0.0273**	0.0278	0.0302
	200	**0.0195**	0.0533	0.0209	0.0211	0.0233
	500	**0.0117**	0.0452	0.0117	0.0119	0.0138
bilirubin [1]	50	**0.0537**	0.0655	0.0618	0.0628	0.0637
	100	**0.0446**	0.0469	0.0468	0.0464	0.0455
	200	**0.0263**	0.0289	0.0264	0.0270	0.0269
	500	0.0189	0.0190	**0.0171**	0.0172	0.0174
phosphatase [1]	50	**0.0512**	0.0539	0.0558	0.0541	0.0530
	100	**0.0366**	0.0385	0.0383	0.0380	0.0373
	200	0.0274	0.0275	0.0281	**0.0274**	0.0275
	500	**0.0147**	0.0154	0.0151	0.0154	0.0156
inr [1]	50	**0.0480**	0.0587	0.0585	0.0568	0.0555
	100	**0.0342**	0.0369	0.0355	0.0365	0.0370
	200	0.0242	0.0260	0.0249	0.0242	**0.0240**
	500	0.0164	**0.0148**	0.0148	0.0149	0.0149
ggtp [1]	50	**0.0644**	0.0738	0.0723	0.0695	0.0687
	100	0.0466	**0.0456**	0.0476	0.0460	0.0478
	200	0.0299	0.0321	**0.0298**	0.0310	0.0308
	500	0.0204	0.0204	0.0201	**0.0200**	0.0202
alt [1]	50	**0.0457**	0.0508	0.0502	0.0491	0.0499
	100	**0.0361**	0.0387	0.0379	0.0379	0.0383
	200	0.0317	**0.0303**	0.0308	0.0308	0.0308
	500	**0.0196**	0.0205	0.0206	0.0207	0.0207
ast [1]	50	**0.0537**	0.0615	0.0649	0.0589	0.0597
	100	**0.0372**	0.0465	0.0477	0.0472	0.0454
	200	0.0294	0.0283	0.0283	**0.0281**	0.0283
	500	**0.0182**	0.0184	0.0186	0.0186	0.0186
CombVerMo	50	0.0534	0.0645	0.0712	0.0685	**0.0534**
	100	**0.0402**	0.0440	0.0411	0.0443	0.0426
	200	**0.0333**	0.0415	0.0353	0.0358	0.0372
	500	**0.0153**	0.0277	0.0163	0.0162	0.0219
CldShadeOth	50	**0.0512**	0.0798	0.0638	0.0655	0.0553
	100	0.0370	0.0609	**0.0348**	0.0373	0.0399
	200	0.0284	0.0610	**0.0266**	0.0280	0.0291
	500	0.0185	0.0510	0.0186	0.0189	**0.0177**
Boundaries	50	0.0604	0.0666	0.0619	0.0636	**0.0583**
	100	0.0448	0.0502	0.0455	0.0444	**0.0432**
	200	**0.0236**	0.0297	0.0243	0.0237	0.0248
	500	0.0170	0.0266	0.0166	0.0160	**0.0218**
CompPlFcst	50	0.0438	0.0544	0.0531	0.0557	**0.0421**
	100	**0.0332**	0.0408	0.0396	0.0415	0.0384
	200	0.0260	0.0258	0.0257	**0.0231**	0.0239
	500	0.0212	0.0207	**0.0203**	0.0207	0.0208
AMInsWliScen	50	0.0479	0.0579	0.0571	0.0574	**0.0479**
	100	**0.0312**	0.0364	0.0328	0.0316	0.0320
	200	**0.0292**	0.0302	0.0301	0.0307	0.0295
	500	0.0192	**0.0190**	0.0173	0.0172	0.0176
PlainsFcst	50	0.1404	**0.0562**	0.0578	0.0563	0.1205
	100	0.1165	0.0333	0.0392	**0.0329**	0.0882
	200	0.0874	0.0227	0.0239	0.0224	**0.0223**
	500	0.0403	0.0167	**0.0165**	0.0174	0.0178

Table A.3: Results of the empirical study

A.5 Conclusions and Related Work

In this paper we introduced three decompositions of conditional probability tables. The Average decomposition drew ideas from both the linear models and causal independence models. The two remaining decompositions were basically generalizations of the decomposable CI models with some constraints removed. We showed that these decompositions can be exploited in learning large conditional probability distributions for BNs, especially when the number of data records is small. Additional known benefits of CI models with the decomposability property is improvement in performance of inference algorithms, and the proposed models can be exploited in the same manner. The advantages of the decompositions should be more apparent in nodes with a large number of parents.

We believe that our work has significant potential applications in practice.

It is often the case that nodes in practical models may have more than 10 parents. Learning parameters of such large CPTs requires volumes of data. Models containing nodes with large in-degrees pose a challenge for inference algorithms. The decompositions proposed here can address two problems at the same time, while providing a reasonably good approximation of the underlying distributions.

In our empirical study we used three sizeable real-life models and tried to approximate selected large CPTs from these networks using the new models and CPT and the noisy-MAX. In general, we found that none of the methods is superior to others, and particular characteristics of distributions in the CPT are an important factor. However, the decompositions seem to perform better when the number of data records is small. Based on the current results, we envision the following application of this work. For each (large) CPT in a model one can try to learn parameters of different models and use the likelihood of the data for the decomposed node to see which decomposition is most promising (preferably using learning and testing data sets to avoid over-fitting). We performed a pilot study on this method and the results in terms of the likelihood function were similar to the goodness of fit measure used to compare the gold standard models.

In the past, some researchers tried to exploit models for local probability distributions in BN. Meek and Heckerman [20] experimented with learning various causal independence models for both parameter and structure learning, but they did not explicitly exploited the decomposability property. Dagum and Galper [22] attempted to introduce linear models for modeling local probability distributions. However, their approach differs from the average model proposed here. Our decompositions preserve independence of parameters and allow for using the EM algorithm.

Acknowledgments

This research was supported by the Air Force Office of Scientific Research under grant F49620-03-1-0187 and by Intel Research.

Bibliography

[1] B. Abramson, J.M. Brown, W. Edwards, A. Murphy, and R.L. Winkler. Hailfinder: A Bayesian system for forecasting severe weather. In *International Journal of Forecasting*, pages 57–71, Amsterdam, 1996.

[2] I.A. Beinlich, H.J. Suermondt, R.M. Chavez, and G.F. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proceedings of the Second European Conference on Artificial Intelligence in Medical Care*, pages 247–256, London, 1989.

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

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

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

[6] Francisco Javier Díez and Severino F. Galán. Efficient computation for the noisy max. *Int. J. Intell. Syst.*, 18(2):165–177, 2003.

[7] Marek J. Druzdzel and Henri J. Suermondt. Relevance in probabilistic models: "Backyards" in a "small world". In *Working notes of the AAAI–1994 Fall Symposium Series: Relevance*, pages 60–63, New Orleans, LA (An extended version of this paper is in preparation.), 4–6 November 1994.

[8] Marek J. Druzdzel and Linda C. van der Gaag. Elicitation of probabilities for belief networks: Combining qualitative and quantitative information. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI–95)*, pages 141–148, San Francisco, CA, 1995. Morgan Kaufmann Publishers, Inc.

[9] Nir Friedman. The Bayesian structural EM algorithm. pages 129–138.

[10] Nir Friedman. Learning belief networks in the presence of missing values and hidden variables. In *Proc. 14th International Conference on Machine Learning*, pages 125–133. Morgan Kaufmann, 1997.

[11] Nir Friedman and Moises Goldszmidt. Sequential update of Bayesian network structure. pages 165–174.

[12] D. Heckerman and J. Breese. Causal independence for probability assessment and inference using Bayesian networks. In *IEEE, Systems, Man, and Cybernetics*, pages 26:826–831. 1996.

[13] David Heckerman. A Tutorial on Learning Bayesian Networks. Technical Report MSR-TR-95-06, March 1995.

[14] David Heckerman, Dan Geiger, and David Maxwell Chickering. Learning bayesian networks: The combination of knowledge and statistical data. In *KDD Workshop*, pages 85–96, 1994.

[15] Max 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. Elsevier Science Publishing Company, Inc., New York, N. Y., 1989.

[16] G. Kokolakis and P. Nanopoulos. Bayesian multivariate micro-aggregation under the hellinger's distance criterion, 2001.

[17] Pieter Kraaijeveld. GeNIeRate: An interactive generator of diagnostic Bayesian network models.

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

[19] Yan Lin and Marek J. Druzdzel. Computational advantages of relevance reasoning in Bayesian belief networks. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, pages 342–350, San Francisco, CA, 1997. Morgan Kaufmann Publishers, Inc.

[20] C. Meek and D. Heckerman. Structure and parameter learning for causal independence and causal interaction models. In *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 366–375, San Francisco, CA, 1997. Morgan Kaufmann Publishers.

[21] 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.

[22] Dagum Paul and Galper Adam. Additive belief-network models. In *Proceedings of the 9th Annual Conference on Uncertainty in Artificial Intelligence (UAI-93)*, pages 91–98, San Francisco, CA, 1993. Morgan Kaufmann Publishers.

[23] J Pearl. Fusion, propagation, and structuring in belief networks. *Artif. Intell.*, 29(3):241–288, 1986.

[24] Judea Pearl. Evidential reasoning using stochastic simulation of causal models. *Artifical Intelligence*, 32:245–257, 1987.

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

[26] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction and Search*. Lecture Notes in Statistics 81, Springer Verlag, 1993.

[27] N. Zhang and L. Yan. Independence of causal influence and clique tree propagation. In *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 481–488, San Francisco, CA, 1997. Morgan Kaufmann Publishers.