MSc Thesis

Implementing and Improving a Method for Non-Invasive Elicitation of Probabilities for Bayesian Networks

ing. Martinus A. de Jongh

TU Delft
Delft University of Technology

University of Pittsburgh

Implementing and Improving a Method for Non-Invasive Elicitation of Probabilities for Bayesian Networks

by

ing. Martinus A. de Jongh

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

Presented at

Delft University of Technology,
Faculty of Electrical Engineering,
Mathematics and Computer Science
Man-Machine Interaction Group
Mekelweg 4, Delft

January 2007

Man-Machine Interaction Group
Faculty of Electrical Engineering,
Delft University of Technology,
Mathematics and Computer Science
Mekelweg 4 2628 CD Delft
Netherlands

Decision Systems Laboratory
Department of Information Science and Telecommunications
University of Pittsburgh
135 North Bellefield Avenue
Pittsburgh
PA 15260
United States of America

Members of the Supervising Committee
drs. dr. L.J.M. Rothkrantz
dr. S. Ooms
dr. K. van der Meer
Dr. ir. M.J. Druzdzel (Decision Systems Laboratory)

ing. Martinus A. de Jongh
1191713

Keywords
Artificial intelligence, Bayesian networks, conflict detection,
Dirichlet distribution, GeNIe,knowledge elicitation,
linear programming, probability theory, SMILE

Abstract

Implementing and Improving a Method for Non-Invasive Elicitation of Probabilities for Bayesian Networks

Man-Machine Interaction Group
Faculty of EEMMCS
Delft University of Technology

Members of the Supervising Committee
drs. dr. L.J.M. Rothkrantz, dr. S. Ooms,
dr. K. van der Meer, dr. ir. M.J. Druzdzel (DSL)

Elicitating knowledge from experts is always a difficult task. Many interviewing techniques exist (N. J. Cooke, 1994), but it is often difficult to select the right technique for the task at hand. Knowledge elicitation is especially difficult for expert systems that are based on probability theory. The elicitation of probabilities for a probabilistic model of a problem requires a lot of time and interaction between the knowledge engineer and the expert. Through games and other techniques the expert has to be calibrated to get good probability estimates (R. Cooke, 1991).

Bayesian networks (BNs) are an example of a structure that can be used to create a probabilistic model. They consist out of two parts: a graph representing the variables of the model and their conditional dependencies, and conditional probability tables (CPTs) for every node that represent the probabilistic behavior of a variable of the model. BNs need specific conditional probabilities for their CPTs. If an expert does not know these probabilities, but knows other useful probabilistic information, this information generally cannot be used directly to fill the CPTs of the Bayesian network. It will be necessary to perform calculations before the information is transformed into conditional probabilities directly usable for BNs.

Druzdzel and van der Gaag (1995) have proposed a theoretical framework that would allow for the direct use of other types of probabilistic information. This framework has been used as a starting point for the implementation of a non-invasive elication method. Here, non-invasive stands for the ability of the method to directly use any information the expert is willing to state to acquire conditional probabilities for a Bayesian network.

There were many possibilities to improve the framework. Among them were: finding methods for conflict detection and conflict resolution and improving the sampling process that is at the core of the method.

This thesis describes the work and research that was done to implement and improve the method. It describes the performed research, the design and implementation of the method, and an empirical evaluation of the method. The implemented method works, but will need further development to achieve better results.

Acknowledgments

This thesis is the result of my graduation project, on which I have worked from January to December 2006. I have had the opportunity to do most of my work at the Decision Systems Laboratory(DSL), Department of Information Science and Telecommunications, University of Pittsburgh. The work I have done there and in Delft would not have been possible without the help, support, or advice from the following people.

- drs. dr. Leon Rothkrantz, who has offered me the possibility to do my graduation work in Pittsburgh.
- dr. Marek Druzdzel for allowing me to do my work at his laboratory. I have learned a lot during my stay in Pittsburgh and most of it was due to dr. Druzdzel.
- My parents Martien en Jolanda de Jongh, and my sister Nadine, for supporting me during my stay abroad.
- dr. Changhe Yuan, for very useful advice on the subject of importance sampling.
- Tomek Sowinski, for some expert advice on C++ programming.
- Everyone at and connected to DSL, but in particular Mark Voortman, Tomek Loboda, Divyasheel Sharma, Paul Maaskant, Joris Hulst and Joost Koiter. Thank you for giving me advice on my work and my thesis, providing me with a place to sleep at the beginning of my stay, and in general making my stay in Pittsburgh a time to remember forever.

Finally my stay at the Decision Systems Laboratory in Pittsburgh would not have been possible without the financial support from the following parties:

- Fundatie Vrijvrouwe van Renswoude,
- Stimuleringsfonds voor Internationale Universitaire Samenwerkingsrelaties (STIR),
- Universiteitsfonds Delft,
- EWI Reisfonds,
- Zorgverzekeraar DSW.

Contents

Chapter 1

Introduction

This thesis reports about the implementation and improvement of a method for non-invasive elicitation of probabilities that are to be used in Bayesian networks (BNs). In the last two decades probabilistic modeling and reasoning has become one of the main topics in artificial intelligence research. A Bayesian network is an example of a structure that can be used to create a probabilistic model. Bayesian networks have recently become quite popular in the AI community for modeling problems that deal with uncertainty.

1.1 Motivation

Through education or practical experience an individual can acquire a vast amount of knowledge. If someone works hard and spends a large amount of time studying or working in a specific field, he or she can get a very deep understanding of this field. Someone who has such an understanding of a field is considered an expert. Some example fields where human experts have a large influence are: medicine, oil exploration, and economics, where financial experts try to predict the course of the stock market.

Experts are very valuable, their knowledge and opinions can be used to advice people that have a less extensive understanding of their field. Consulting an expert may considerably reduce the time necessary to solve problems or to complete projects. A problem with experts is that they are human, meaning that they can only be at one place at a time, need sleep, grow old and will eventually die. The vast amount of knowledge they have acquired during their lifetime has an expiration date. Unless their knowledge is stored somewhere it will eventually disappear. To save their knowledge, experts can write books or papers, or teach what they have learnt to others.

In the 1970s the AI community started working on expert systems, computer programs which purpose it is to store an expert's knowledge of a domain and to reason with this knowledge to be able to answer domain related questions. Expert systems have some advantages over human experts.

The knowledge stored in an expert system can be accessed at any time and can be copied as many times as necessary. It is also very likely that the knowledge stored in an expert system will be available forever.

The first expert systems were rule based. The expert's knowledge of a domain was represented as a set of logical rules. The rules of these expert systems had a IF ... THEN ... structure. Facts, statements about the problem domain, are inputted into expert systems and the rules are applied to the facts to derive new facts. This process is repeated until no new facts are generated. Now, by examining the facts that are present in the expert system, conclusions can be drawn and the expert system can generate an advice for its user.

These first expert systems worked with boolean logic, a rule could either be true or false, a fact could only be present or absent. In situations where a lot of uncertainty is present these expert systems would not perform optimally. Different approaches were tried to make expert systems more able to cope with uncertainty. Some examples are certainty factors (Shortliffe, 1976), fuzzy logic (Zadeh, 1978), and probability theory.

The use of probability theory with expert systems allows for mathematically correct approaches to deal with uncertainty. Different approaches have been tried, examples are adding probabilities to rules and facts, and completely representing the knowledge as a joint probability distribution (j-PDF). A joint probability distribution is a table that assigns a probability to every possible combination of facts. A big problem with this approach is if the number of facts, or variables of the problem that is being modeled increases, the number of probabilities necessary for the j-pdf increases exponentially. Very quickly the number of probabilities become so large that it is no longer feasible to use this approach.

Bayesian networks (BNs) (Pearl, 1988), developed in the 80s, are a solution to this problem. BNs are graphical models that very efficiently represent joint probability distributions. The number of probabilities a Bayesian network needs to represent a joint probability distribution can be many orders of magnitude less than the number of probabilities necessary for the j-PDF. This reduction of probabilities is possible because of extra assumptions that are added to the model. It is assumed that some of the variables in the model are conditionally independent of each other. When variables are (conditionally) independent of each other it means that the variables do not influence each other and that it is not necessary to specify a probability for every possible combination of these variables.

A BN has a graph that contains the modeled variables. Each node of the graph represents one of the variables of the model and has a local conditional probability distribution that is conditioned on the parents of the node. This distribution is a table (CPT) that has probabilities for each possible value of the node and each possible combination of the node's parents. If two nodes in the graph are not connected by an edge, the two variables the nodes

represent are conditionally independent of each other. The graph typically represents a causal structure, i.e., the parents of a node represent the cause of this node. When modeling a causal process using a BN, cause - effect relations in the process are modeled as parent - child relations in the BN, where the child is conditionally dependent on the parent(s). An example BN is shown in Figure 1.1, due to (Beinlich, Suermondt, Chavez, & Cooper, n.d.).

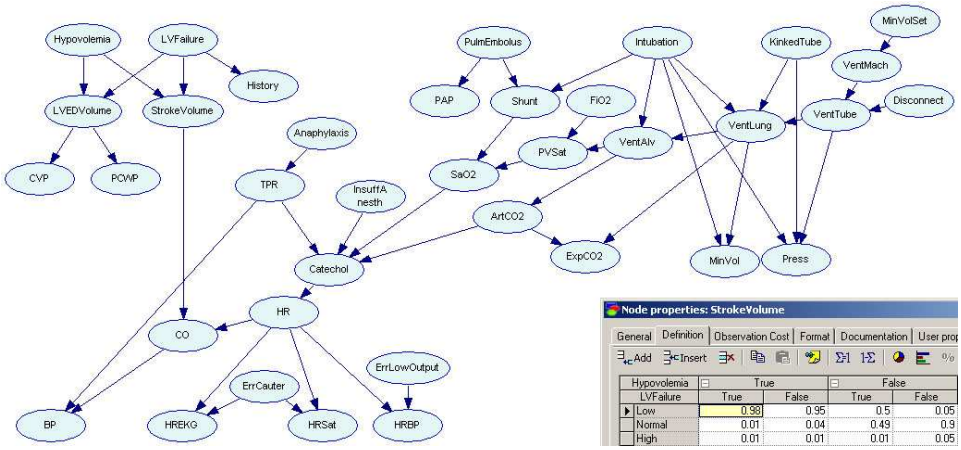


Figure 1.1: Example Bayesian network used for monitoring patients in intensive care wards

A completely defined Bayesian network can answer any probabilistic query for the model. Typical queries concern the probability that one of the nodes has a certain value given that other variables of the model all have a certain value. An example query for a Bayesian network that models a desease and its symptoms could be: “what is the probability that the desease is present given the fact that one of its symptoms is present?” Another could be “What is the most likely combination of symptoms to be present when a patient has the desease?”

Bayesian networks have many applications, their ability to make predictions while taking uncertainties into account makes them very useful. They have been used for applications including medical diagnosis, speech recognition, face recognition, and decision support. But for any of these application to work, first their Bayesian networks have to be created. Their graphs need to be created and the CPTs for every node need to be filled with the necessary probabilities. Basically three approaches exist:

- Elicit the necessary information from a domain expert.
- Learn the necessary information from data.
- The knownledge engineer that is creating the BN estimates the probabilities himself, using relevant literature.

Both the graph and the CPTs can be created using all of the methods. Depending on the application one method may be more appropriate than the other. For speech recognition applications a lot of data is available, learning the probabilities for the CPT entries from this data will most likely give better results than to elicit these probabilities from experts. In other situations using the opinions of experts may be the only viable option. If for example a model would be created to predict the possibility of nuclear war, expert opinions are the only source of information. Generally, in situations where there is not enough data, or no data at all, elicitation of knowledge from experts is the only approach to get the necessary probabilities for a model.

Knowledge elicitation in general can be difficult to perform effectively. Acquiring knowledge from an expert and putting it into a model can be a daunting task. It may be very difficult for an expert to explain to a layman, in this case the knowledge engineer trying to elicit the knowledge from the expert, what the important aspects of his or her field are and why these aspects should be included in the model that they are developing. Also an expert may be unwilling to cooperate, because perhaps he or she is afraid to be replaced by the system for which knowledge is being supplied. There have been many studies in several related fields to find optimal procedures for the elicitation of knowledge (N. J. Cooke, 1994), but in general the result of a knowledge elicitation procedure depends mostly on how much the expert is cooperating with the knowledge engineer.

Knowledge elicitation for systems based on probability theory have an extra difficulty. To describe this difficulty in a nutshell: the expert has to assign probabilities, which are numerical values, to all the possible events the application is modeling. Beside the fact that the number of probabilities increases exponentially with the size of the model, experts may find it difficult to assign an *exact* value to an event. The probabilities stated by an expert are subjective, which means they represent the degree of belief the expert has that the events the probabilities describe will occur. When an expert is stating a (subjective) probability, he or she is usually not performing mental calculations (R. Cooke, 1991). Experts will generally rely on rules of thumb, or heuristics. Using heuristics can cause an expert to become biased. The expert may not be able to provide probabilities that accurately represent the true probabilities of the events to be modeled. To counter this bias the expert will need to be calibrated. According to (R. Cooke, 1991) an expert is *well-calibrated* if

> *"for every probability value r, in the class of all event to which the expert assigns subjective probability r, the relative frequency of occurrence is equal to r."*

Calibration is a difficult, but necessary process to acquire probabilities that approach the "real" probabilities more closely.

After calibration the elicitation process can begin. Research in knowledge elicition has shown that asking an expert to state probabilities directly generally leads to probability estimates of low quality (R. Cooke, 1991). It is generally better to use an indirect approach that uses a sort of betting game to determine the desired probabilities (R. Cooke (1991); Clemen (1996)).

Until now it is assumed that the expert actually can give, maybe by using betting games, an estimate of the desired probabilities. However, it is possible that an expert cannot give an estimate for a certain probability directly, but only implicitly by estimating other probabilities first and calculating the desired probability using, for instance, Bayes' rule (Section 2.2.2). Also an expert may have information that is not quantitative of nature, i.e. not in the form of numerical probabilities. Other types of probabilistic information exist that are qualitative of nature and cannot be directly interpreted as probabilities for a Bayesian network. If the expert can provide these types of information and it is relevant for the model to be developed it would be very inefficient if it would be impossible to use this information for the model.

Assuming that the graph necessary for a Bayesian network has been developed, now only the probabilities for the CPTs will have to be elicited. Any probability of a CPT entry the expert can estimate directly without calculations can be considered as relatively "easy" to use for the model. Probabilities or qualitative statements that are relevant for the model, but cannot be elicited in a way so that they can be used in the Bayesian network directly, can be considered to be "hard" to use for the model. How these probabilities, or qualitative information types, are exactly acquired is not even really relevant. The main problem is that a Bayesian network requires conditional probabilities, and only the specific conditional probabilities that are necessary for the CPTs of its nodes.

If it were possible to use probabilities and probabilistic information other than the necessary conditional probabilities directly this would make the eliciation process easier for the expert. Here with easier it is meant that any information relevant to the model can now be be used for filling the CPT entries of the Bayesian network, and it would no longer be necessary to let the expert transform the information he or she has into conditional probabilities that can be directly put in the BN. According to (Druzdzel & van der Gaag, 1995) such an elicitation method can be considered to be *non-invasive*. They define non-invasive as:

> *"Allowing any type of probabilistic information, quantitative or qualitative of nature, the expert is willing to state to be interpreted directly for the elicitation of probabilities for a Bayesian network."*

Druzdzel and van der Gaag (1995) have proposed a theoretical framework for an elicitation method that lets an expert specify various statements of

qualitative and quantitative nature and interprets these statements as constraints used to guide the determination of the CPT entries for a Bayesian network.

This framework has been used as a starting point for the implementation of a non-invasive elicitation method that allows the direct use of different types of probabilistic information, quantitative and qualitative, and that guides the elicitation process of the CPT entries for a Bayesian network.

1.2 Problems and Challenges

The proposed framework has never been implemented before, there may be problems with it that have never been revealed. Some aspects of the original framework may also need revision to achieve a higher efficiency, and to be able to take advantage of research conducted after the paper's publication. The basic framework and its possible improvements provide for the following challenges that could lead to a MSc thesis:

- The framework relies on constraints to reduce the sample space of the probability hyperspace. If there are constraints that are contradicting each other, we speak of a conflict. If there are conflicting constraints present, the result is that sampling has become impossible because it will be impossible for a sample to satisfy all constraints. A conflict resolution method is necessary to detect when constraints are conflicting and to notify the expert that these constraints will have to be changed. An investigation should be performed to find conflict detection and resolution methods.

- The framework uses stochastic sampling to acquire 2^{nd} order distributions. Since the hyperspace is very large, even when using constraints to prune the search space, plain sampling will not lead to very good results. Recently, a lot of work has been done at the DSL lab on importance sampling (IS) (Yuan & Druzdzel, 2006). Using IS can improve the sampling results and generally requires less samples to reach a good estimate. The quality of the sampling process, however, depends on the quality of the importance sampling function used in the process. Applying IS instead of plain sampling might lead to a significant improvement of efficiency. Designing an IS algorithm should be considered to improve the sampling process.

- It was proposed to divide the Bayesian networks into smaller networks To combat the effects of the method's computational complexity. it was proposed to transform the BN into a chordal graph, and to decompose this graph into cliques. Generally it is easier for an expert to provide probability statements for a smaller network. But due to the transformation to a chordal graph the expert might find the new

structure a bit confusing, because the graph no longer represents a causal structure. Other types of BN decompositions might lead to better results. An other potential problem could be merging the decomposed network parts back into the complete network. Because the decomposed parts are processed independently, overlapping variables may need special treatment to get an optimal result when merging the parts. It should be researched how to decompose and merge a Bayesian network in such a way that the resulting sub networks are easy to understand for the experts.

- In the paper a linear programming technique was proposed to determine probability intervals for the constituent probabilities. Linear programming faces the problem of conflicting constraints as well, and it may be possible to discover conflicts and pin-point the conflicting constraints at this point. It should be researched which extra steps are necessary to make it possible for this linear programming technique to be used automatically in a non-invasive elicitation method.

- A challenge will be to design an effective user interface that is user friendly and is capable of efficiently acquiring the necessary statements from the expert that the method needs to generate the constraints.

1.3 Goals

The main goals to be achieved for my M.Sc. thesis were:

- Research:
  - Design an efficient sampling scheme for the method.
  - Design a method for conflict detection and conflict resolution for constraints.
  - Develop an importance sampling scheme for deriving 2^{nd} order probability distributions over the constituent probabilities.
  - Design a method for merging decomposed family networks into the complete BN as accurately and efficiently as possible.
  - Design a graphical interface for GeNIe for interactive elicitation of constraints.

- Implementation:
  - Design and implement a data structure for the constraints.
  - Design and implement a data structure for the constituent probabilities.
  - Implement a linear programming algorithm for calculating probability intervals over the constituent probabilities.
  - Improve the efficiency of the method.
  - Test the method and report the results in the thesis.

1.4 Thesis Outline

The rest of this thesis is organised as follows.

Chapter 2 describes the theoretical background of the different aspects of this assignment. Chapter 3 addresses the performed research for the assignment. Chapter 4 discusses the design of the software for the method. Chapter 5 discusses the implementation of the method. Chapter 6 discusses an empirical evaluation of the method as a whole, and a separate evaluation of a part of the method. In Chapter 7 this thesis concludes the contributions, possible future work, and ends with the last concluding remarks.

Chapter 2

Theoretical Background

2.1 Knowledge Elicitation

2.1.1 Expert Systems

In the 1970's a new type of Artificial Intelligence (AI) system appeared. This type of system relied on logic inference using facts and rules. The facts represent the inputs and outputs of the system and the rules represent knowledge about the problem being modeled with the system. These systems were called Expert Systems. Some famous examples are the DENDRAL system (Buchanan, Sutherland, & Feigenbaum, 1969), that was used for analyzing chemicals, and the MYCIN system (Shortliffe, 1976), which diagnosed infectious blood diseases. Expert systems can be very general, as each is build around an inference engine that takes the facts and the rules as input, and by applying the rules on the facts it computes new facts that serve as output for the "query". This inference engine is typically called "expert system shell". The rules and facts are specified with the help of a human expert and describe a certain problem domain. If other rules and other facts are added to the system that apply to another domain, then the system will model this domain instead. Examples of expert system shells are the CLIPS system (Riley, November 1991), and a domain independent version of MYCIN: Empty MYCIN (EMYCIN) (Melle, Shortliffe, & Buchanan, 1981).

Expert systems using logical inference have some limitations, especially when dealing with situations involving uncertainty and erroneous data. The systems use Boolean logic and this type of logic has no acceptable means of dealing with errors or uncertainty. Erroneous data could occur because of some sort of misunderstanding or sensor failure, depending on the type of inputs the expert system has. Uncertainty plays a role when rules for a domain are not really crisp like Boolean logic is. A rule might only be accurate in about 80 percent of the cases, but this cannot be modeled by using boolean logic. The same could apply to the amount of confidence

one has in an input value presented to the expert system. Researchers have searched for solutions to take uncertainty into account. The MYCIN system, for instance, uses an ad-hoc formalism called certainty factors. These factors are added to rules to give an estimation of the amount of certainty that this rule is correct. When performing inference the certainty factors of the rules, the rules, and the facts are used as input and, at the end, a certainty factor is calculated for the conclusion.

Another approach was to create expert systems based on probability theory. Specifically, using Bayes' Rule to update knowledge. Here, instead of certainty factors, probabilities are added to the rules to represent the degree of belief in the accuracy of the rule. Now, when performing inference, the probabilities and Bayes' rule are used to calculate a degree of belief in the result of the inference. The first expert system that used Bayesian updating successfully was PROSPECTOR (Duda, Hart, & Nilsson, 1979), a consultant system for mineral exploration. This system still used rules as the representation of knowledge. Another viewpoint is to only work with a probability distribution. Every factor of the problem would then be modeled as a stochastic variable and a joint probability distribution function (J-PDF) over all these stochastic variables would represent the knowledge of the expert system. Asking the expert system a question would reduce to calculation a specific probability given some instantiated variables. Since the J-PDF contains all the information about the domain, every possible query can be answered. An example of such a system was created by Dombal, Leaper, Horrocks, and Staniland (1974), which diagnosed acute abdominal illnesses and was demonstrated to perform better than human experts. The problem with this approach is that when the number of variables increases, the number of entries in the J-PDF increases exponentially. The size of the table will become so huge that it will no longer be feasible to work with the J-PDF as a form of knowledge representation. This was the reason that this approach was abandoned very quickly.

Other methodologies exist that are not based on probability theory, examples are fuzzy logic (Zadeh, 1978) and Dempster-Shafer theory (Shafer, 1976), but during the 1980's some interesting work has been done to make probabilistic reasoning systems feasible. To reduce the number of probabilities necessary for modeling a problem, researchers started to exploit available structure in the problem domains they were modeling. An example of this are Bayesian Networks (BNs) (Pearl, 1988). Conditional independence between variables is exploited to reduce the number of probabilities. Now, instead of having one very large table with the J-PDF, a BN has conditional distribution tables for every variable in the problem that depends on the other variables. A BN consists of a directed acyclic graph (DAG), where every node represents a variable of the problem, and conditional probability distributions for every node of the network. Effectively, the J-PDF is factored into smaller conditional probability distributions for every node.

This relieves pressure from the complexity problem because now the total number of probabilities no longer is exponential in the number of variables. The complexity is more "localized", since now every variable has it's own probability table. The number of probabilities in this table is exponential in the number of parent nodes. This makes the whole network much more scalable as long as the number of parents stays small.

2.1.2 Overview of Knowledge Elicitation Technique Families

These different types of expert systems have one thing in common; the knowledge needs to be gathered and put into the systems. To do this, a knowledge engineer can use the relevant literature, try to formulate the knowledge himself, or consult an expert. Consulting an expert will generally be the best option to pursue, because an expert will probably be able to easily identify the most important aspects of a problem domain. The process of acquiring knowledge from an expert is called Knowledge Elicitation (KE) and will mostly consist out of the knowledge engineer interviewing or working with the expert to describe his knowledge in the form that is most suitable for input in the expert system. In the fields of artificial intelligence and cognitive psychology, there has been a lot of research, describing knowledge elicitation techniques for different situations. N. J. Cooke (1994) has organised the different knowledge elicitation techniques into three families:

- Observations and Interviews.
- Process Tracing.
- Conceptual Techniques.

The first family of techniques describes methods where the knowledge engineer either observes the expert at work or interviews the expert to acquire the desired knowledge. She makes a distinction between three subgroups: observations, interviews and task analysis. By observation is meant watching the expert at work and recording his or her actions and deriving the desired knowledge from these recordings. Interviews are the most common form of knowledge elicitation. Here the knowledge engineer and the expert interact with each other. This can be an interview or a discussion. Cooke describes many variations, examples are unstructured interviews, usage of the twenty questions game, questionnaires, and role playing. Task analysis is a more formal set of techniques where *the focus is on what the expert does as opposed to what the expert knows* (N. J. Cooke, 1994) and this set of techniques is not used in knowledge elicitation for expert systems as much.

The second family represents more formal techniques that follow the execution of tasks performed by the expert. The data that is recorded during the time the expert performs the task is formatted in a prespecified type. This, unlike the more informal observations and interviews mentioned

earlier where more uncertainty exists about what eventually will be observed or discussed during the interview. The family is subdivided into four groups: verbal reports, non-verbal reports, protocol analysis and decision analysis. Most relevant for this thesis is the subgroup called decision analysis, that also describes techniques for eliciting estimations of probability and utility.

The third family describes so-called conceptual techniques, that *produce representations of domain concepts and their structure or interrelations* (N. J. Cooke, 1994). These techniques generate very general information and there has been some debate on the usefulness of these techniques. Some have argued that the generated information by using these techniques may be unrelated or irrelevant to task performance.

2.1.3 Elicitation Techniques for Probabilities

This thesis describes the implementation and improvement of a method that allows for non-invasive elicitation of probabilities for a Bayesian network. As the final part of this section an overview will be given of knowledge elicitation techniques for probabilities.

Knowledge elicitation is difficult and time consuming, especially if the model grows in size. This causes, as mentioned earlier, the number of probabilities to grow. Because it is hard to assign numerical values to events, AI researchers and psychologists have tried to find ways to "sugar coat" the determination process of the numerical value of the probability. These "devices" might relax the expert and let the expert come to a better probability assessment. Clemen and Reilly (2003) describe betting games and probability wheels as examples of indirect gathering of probabilities. In the case of a binary variable A with values true and false, a betting game works as following, two bets are presented to the expert:

1. Win $X if A is true.
   Lose $Y if A is false.

2. Lose $X if A is true.
   Win $Y if A is false.

X and Y represent the amount of money that is put into the betting "pot". The knowledge engineers choose the values for X and Y. The idea is to ask the expert to choose between the bets and to adjust the values of X and Y until the expert is indifferent between the bets. When the point of indifference is reached, the expected values (see section 2.2) of the two bets must be equal. Equating the two expected values and solving for the probability that A is true gives:

$$P(A = true) = \frac{Y}{X + Y}.$$

Now the determined values of X and Y can be used to approximate the probability that A is true, and, respectively, the probability that A is false. The approach is not without problems, some people may not like the betting analogy. For these people, the game may be more distracting than direct probability assessment and the result will be opposite of the original intention of the game. Also, most people do not like the idea of losing money although the game is purely hypothetical. This "fear" of losing money will have an influence on the decision that the expert will make when picking bets (Clemen & Reilly, 2003). Another type of game is the lottery: here the expert has to choose between two different lotteries:

1. Win Prize X if event A is true.
   Win Prize Y if event A is false.

2. Win Prize X with know probability p.
   Win Prize Y with probability 1–p.

The second lottery serves as a reference, where the mechanism to get the probability must be well defined. This might be spinning a wheel with two areas (true and false) or drawing a colored ball from a collection of balls that has a color distribution equal to the probability distribution defined by p. Now, just as with the betting game, the expert is asked to choose the lottery he would like to try. After the expert chooses a lottery, the knowledge engineer changes the value of p to make the other lottery more attractive to the expert. This process continues until the expert is again indifferent between the lotteries. At this point, the current value of p is the probability that should be chosen for the event A = true. Interesting to notice is that the lottery device uses a probability distribution as reference and the expert has to choose the preferred lottery by looking at the values of the distribution. The numbers of the distribution might again influence the expert and to solve this problem, a graphical representation of the distribution could be used. One example is what Clemen and Reilly call a probability wheel, a pie chart where the different areas represent the probabilities of the distribution. By changing the boundaries of the areas, the probabilities change. The visual representation of this process might give the expert a better view of the situation, and improve his probability estimates. Also other graphical representations could be used. GeNIe, software developed at University of Pittsburgh's Decision System Lab, allows the use of probability wheels and bar charts for determining probabilities. Just as betting games are not always accepted by experts, lotteries also may have some negative aspects. Again some people may not like playing games or have difficulty getting 'into' the game. Both methods can be expanded to be able to handle discrete variables (with more than two values) and continuous variables. See Clemen and Reilly (2003) for a more detailed explanation.

2.2 Probability Theory

2.2.1 Axioms of Probability

We live in a world full of uncertainty. Our perception of this world is limited by the range and the sensitivity of our senses. What happens outside our view is unknown to us and we can only guess or try to logically infer conclusions that help us to define the state of the (unknown) world. On a smaller scale, let us imagine a world that consists out of a deck of cards. A deck has 52 cards: the cards are red or black of color, belong to one of the four suits: hearts, clubs, spades, or diamonds and show either a number ranging from two to ten or one of the figures: jack, queen, king, or ace. If we shuffle a deck of cards, we will not know which card will be on top of the deck. We will be uncertain until we turn the card around. But we can try to guess the card. If we try to guess a specific card, the ace of spades for instance, we have a one in 52 chance to be correct. The reason for this is that there are 52 cards, there is only one ace of spades, and only one card can be on top of the deck. A little less ambitious would be to try to guess the suit of the card. Since every suit has 13 cards, one fourth of the total number of cards, there is a 1 in 4 chance to guess the right suit. For those who really do not like to take a risk, there is always the color of the card that can be guessed. Half of the cards is red and the rest is black, so there is a one in two chance of guessing this right. The observations of the odds that a certain type of card is on top are interesting, using probability theory we can formalise them and give them some theoretical backing.

We start by looking at all the different possible outcomes of drawing a card from the deck. In this case, there are 52 possible outcomes, because there are 52 different cards in the deck. Together these outcomes form what is called in probability theory the *sample space*. Formally, the sample space of an experiment is a set that contains all the possible outcomes of the experiment. In the example with the playing cards, the experiment is the drawing of the top card of the deck. Now a probability function can be used that assigns a probability, a numerical value between zero and one, to every outcome. Depending on the experiment, the sample space can be divided into different subsets called events. Examples of events are: “all cards that are red”, “all cards that belong to the hearts suit”, and “all aces”. The probability of an event can be calculated by adding all the probabilities of the outcomes that belong to the event. The probability of the sample space is calculated in the same way as the probability of an event. It is calculated by adding the probabilities of all the outcomes, the result of this calculation is 1.

More formally one can say:

> A probability function P is defined on a sample space Ω to assign a numerical value P(A) to an event A in Ω in the range [0, 1] such that:
>
> 1. The probability of the whole sample space, P(Ω), is equal to 1.
> 2. If events A and B are disjoint, the probability of the union of the events equals the sum of the probabilities of the events: P($A \bigcup B$) = P(A) + P(B).
>
> The next step is the definition of the axioms of probability:
>
> 1. All probabilities are between 0 and 1: $0 \leq$ P(A) ≤ 1.
> 2. True events have probability 1, false events have probabilities 0: P(true) = 1, P(false) = 0.
> 3. The probability of a disjunction is given by: P($A \bigcup B$) = P(A) + P(B) – P($A \bigcap B$).

The axioms of probability are known as Kolmogorov's Axioms, named after the Russian mathematician Andrei Kolmogorov, who showed how to build the rest of probability theory from these axioms (Kolmogorov, 1950).

2.2.2 Conditional Probabilities

The card guessing example can be changed by adding another person; we can let someone else draw the card and reveal one of the features of the card (color, suit or number). Will this influence our ability to guess the right card and change the probability of guessing the card? Yes, it will. If we know that the color of the card is red, we can eliminate half the cards of the deck and this will influence the probability of guessing the right card. Even more clear is the example where we know that the card is an ace. Now there are only four possibilities left, which gives us a probability of 25% of guessing the right card. This new probability of 0.25 is known in probability theory as a conditional or posterior probability and the original probability of $\frac{1}{52}$ is known as the prior probability. The notation for a conditional probability stating "the Probability of event A given event B" is P($A|B$). The conditional probability that the card drawn is an ace of spades given that it is known that the card is an ace, using this notation, is: P($Card = Ace\,of\,Spades|Number = Ace$) = 0.25. A conditional probability can be calculated by using prior probabilities:

$$P(A|B) \quad = \quad \frac{P(A \bigcap B)}{P(B)} \; . \tag{2.1}$$

This equation holds when P(B)> 0. Sometimes it is necessary to compute P($B|A$), but it might be hard to calculate P($A \bigcap B$). Using Bayes' Rule this problem can be solved as follows:

$$P(B|A) = \frac{P(A|B) \cdot P(B)}{P(A)} . \tag{2.2}$$

A more general form exists, making use of the law of total probability:

$$P(B_i|A) = \frac{P(A|B_i) \cdot P(B_i)}{\sum_j P(A|B_j) \cdot P(B_j)} . \tag{2.3}$$

The law of total probability states that P(A) can be found by adding all the probabilities of conjunctions of A with all events B_j. Probabilities of conjunctions can be written as a product of a conditional and a prior probability, this can be derived from equation 2.1 and the axioms of probability.

2.2.3 Random Variables

If one defines a sample space and a probability function that assigns probabilities to every outcome in this sample space, this is sufficient to describe a probabilistic experiment. But there might be to much information in the experiment for it to be useful. To focus on the part of the experiment that is the most interesting or the most relevant *random variables* are used. A definition of a random variable is:

> *"Let Ω be a sample space. A random variable is a function X: $\Omega \rightarrow \Re$ which transforms a sample space Ω into another sample space Ω', which lies in $\Re$. The events of Ω' are more directly related to the features of the experiment which are to be studied (Dekking, Kraaikamp, Lopuhaä, & Meester, 2004)."*

In the card game world an example random variable could be a game where two players each draw a card and the player with the highest card wins. To be able to do this an order has to be assigned to the cards. The random variable would assign a number to each card; color and suit can be ignored. Now it is possible to compare cards and decide which is ranked higher. The random variable function for this game would map every card to a number. Let's name the random variable H. A part of the mapping function for H would be:

$$H(card) = \begin{cases} 2 & card\ number = 2 \\ 3 & card\ number = 3 \\ 4 & card\ number = 4 \\ \vdots & \\ 13 & card = king \\ 14 & card = ace \end{cases}$$

With this mapping the original sample space Ω is transformed into a new one, now consisting out of the numbers 2 to 14, each having the probability $\frac{1}{13}$. Now new events are possible, like the event $\{D > 5\}$ that describes the probability to draw a card with a value higher than five.

Discrete and Continuous Random Variables

Random variables can be divided into two types: discrete (like the playing cards example) or continuous random variables. An example of a continuous variable could be temperature. The easiest way to distinguish between discrete and continuous random variables is that discrete variables in general can only be assigned a finite number of values. This allows for probabilities to be assigned to every possible value of the random variable. Continuous variables on the other hand cannot have a probability assigned to every possible value. This happens because continuous variables can be assigned an uncountable number of values. The number of values for continuous variables is uncountable because they generally lie in the domain of real numbers. The domain of real numbers is uncountable (Sipser, 1996, Chapter 4), which makes it impossible to assign probabilities to every possible value of continuous variables. If it were possible to assign probabilities to every value, then by definition the number of values for a continuous variable would be countable. When the number of values of a variable is countable it means that it is possible to assign a unique number to every possible value of the variable.

Because of the difference between discrete and continuous variables, they are also treated different when assigning probabilities to the variables. Discrete variables are assigned a probability mass function which work with discrete numbers and summations to calculate probabilities. Continuous variables on the other hand are assigned a probability density function, which works with densities, areas and integrals to calculate probabilities. More similar is when the probability $P(X \leq a)$ is calculated. This probability is also represented by the function $F(a)$, known as the distribution function or the cumulative distribution function. For the discrete case, the function is as follows:

$$F(a) = \sum_{i \leq a} P(X = i) \, , \tag{2.4}$$

where $P(X = i)$ is a probability mass function. For the continuous case, the function looks as following:

$$F(a) = \int_{-\infty}^{a} f(x)dx \, , \tag{2.5}$$

where $f(x)$ is a probability density function.

A more formal definition would be (Dekking et al., 2004):

The distribution function F of a random variable X is the function $F : \Re \rightarrow [0, 1]$,defined by:

$$F(a) = P(X \leq a) \; for \; -\infty < a < \infty \; .$$

Expected Value and Variance

The distribution function completely defines the probability distribution for a random variable and is used the most for calculating probabilities. Although having the distribution function is sufficient, there are some measures that can give useful information, even in the situation that the distribution function is unknown. Here is referred to the *expected value* and the *variance* of a random variable.

The expected value can be described as the *average outcome* when the experiment is performed a large number of times. In gambling, expected value can give an indication if it is profitable to keep playing. Most, if not all casino games are constructed to have a negative expected value, meaning that in the long run the house will gain money instead of the players. The expected value can be calculated using the probability mass function or the probability density function if they are known, or estimated by calculating the average of collected samples of the experiment. For the computation of the expected value there are formulas for discrete and continuous variables, respectively:

$$E[X] = \sum_i x_i P(X = x_i) \; , \tag{2.6}$$

$$E[X] = \int_{-\infty}^{\infty} x f(x) dx \; . \tag{2.7}$$

Actually the computation of the expected values for a probability mass or density function is equal to calculating the center of mass of a function. The center of gravity of a function is calculated by evaluating:

$$\frac{\int_{-\infty}^{\infty} x f(x) dx}{\int_{-\infty}^{\infty} f(x) dx} \; .$$

the term $\int_{-\infty}^{\infty} f(x) dx$ disappears from the calculation of expected value because according to the axioms of probability this integral evaluates to one.

Another measure that can give some useful information is the variance. The variance of a random variable gives an indication of how far values of the random variable typically lie from its expected value. The variance is calculated in a way that is similar to the calculation of the expected value. The variance function can be expressed using the expected value operator:

$$var(X) = E[(X - E[X])^2] \; . \tag{2.8}$$

Another form of this equation can be derived and might be a little easier to compute:

$$var(X) = E[X^2] - E[X]^2 \ . \tag{2.9}$$

Computing $E[X^2]$ is done by evaluating the integral:

$$E[X] = \int_{-\infty}^{\infty} x^2 f(x) dx \ ,$$

or the equivalent summation for the discrete case. Generally, calculating the expected value of a function over a random variable is:

$$E[g(X)] = \int_{-\infty}^{\infty} g(x) f(x) dx \ . \tag{2.10}$$

Equation 2.10 can be used to derive Equations 2.8 and 2.9.

Like the expected value, the variance can be estimated when the probability distribution is not known. If N samples have been collected, the variance can be calculated, but two variants exist:

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \overline{x})^2 \ , \tag{2.11}$$

$$s^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \overline{x})^2 \ , \tag{2.12}$$

where $\overline{x}$ is the sample mean, the estimated version of the expected value, calculated with:

$$\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x_i \ . \tag{2.13}$$

The difference between the two formulas has to do with the fact that when one takes a finite sample from the population, the calculated variance of this sample will not be equal to the variance of the total population since not all possible samples have been generated. The formulas are estimators, they estimate the variance of the total population using the data that is collected. Equation 2.11 is a biased estimator. This means even if the amount of available data becomes infinitely large, the result of the calculation will always be not exactly the variance of the population, but very close. Equation 2.12 is an example of an unbiased estimator. Given enough data, the estimator will be able to exactly calculate the variance of the population. Practically, when the amount of available data grows, the difference between σ^2 and s^2 becomes very small and it does not really matter which one is used to estimate the variance.

The result of the calculation of variance has an outcome that has the squared unit of the experiment. For example, if a random variable is used to model length in m, the variance will give a result in m^2. This may be

inconvenient for working with the results and because of this instead of the variance the *standard deviation* is used often. The standard deviation is defined as the square root of the variance. This way the unit of the standard deviation is equal to the unit of the random variable.

2.2.4 Joint Probability Distributions

Once a more complex problem is being modeled, it will be very likely that the model or experiment will contain more than one random variable. If a model has multiple random variables, it is possible that these variables will influence each other. If this is the case then it is no longer a good idea to model the variables independently of each other. This means that there will be one probability distribution function describing all variables instead of multiple distribution functions each describing the probability distribution of one of the random variables in the model. This probability distribution is called the *joint probability distribution*, it can be created by assigning probabilities to every possible combination of outcomes of all the random variables in the model.

The joint probability distribution is needed to describe the whole model, but with this function it is possible to calculate the probability distributions for each of the random variables. This process is called *marginalization*. This process only works one way, from a joint distribution you can derive the distributions for each of the random variables, the *marginalized distributions*, but you cannot combine marginalized distributions to get the joint distribution. The reason for this is that different joint probability distributions can have the same marginal probability distributions.

If we would have a joint distribution function with two variables X and Y, the marginalized distribution functions would be obtained by the following procedure:

$$F_X(a) = P(X \leq a) = F(a, +\infty) = \lim_{b \to \infty} F(a, b) \ ,$$
$$F_Y(b) = P(Y \leq b) = F(+\infty, b) = \lim_{a \to \infty} F(a, b) \ .$$

It is also possible to directly marginalize joint probability mass functions and joint probability density functions. For the probability mass function, the formulas would be:

$$p_X(a) = \sum_{i=1}^{\infty} p(a, b_i) \ ,$$
$$p_Y(b) = \sum_{i=1}^{\infty} p(a_i, b) \ .$$

And for the probability density function they would be:

$$f_X(x) = \int_{-\infty}^{\infty} f(x, y)\, dy \ ,$$

$$f_Y(x) = \int_{-\infty}^{\infty} f(x,y)\,dx \ .$$

When a model or an experiment has more than two random variables, the formulas can be easily extended to accommodate for this. For every extra variable either a summation, an integral or a limit condition has to be added, depending on the used formula.

2.2.5 Independence

In a model with multiple random variables it is possible, as stated earlier, that the different random variables influence each other, but also the possibility exists that different variables do not influence each other. If two random variables do not influence each other they are called *independent* of each other. A more formal definition of independence is (Dekking et al., 2004):

> The random variables X and Y, with joint distribution function F_{XY} are independent if
>
> $$P(X \leq a, Y \leq b) = P(X \leq a)\,P(Y \leq b) \ ,$$
>
> that is,
>
> $$F_{XY}(a,b) = F_X(a)\,F_Y(b) \ ,$$
>
> for all possible values a and b. Random variables which are not independent are called dependent.

It is important to be careful with the independence concept when working with more than two variables: even though the combination of all variables could signal independence between the variables, it is still possible that between subsets of variables dependencies still exist. Independence between variables can be checked with the following formulas:

$$\begin{aligned} P(A\,|B) &= P(A) \ , \\ P(B\,|A) &= P(B) \ , \\ P(A \cap B) &= P(A)\,P(B) \ . \end{aligned}$$

If any of the formulas hold, they all hold, they are equivalent statements. These formulas are derived from the axioms of probability, Equations 2.1 and 2.2.

Another form of independence is when two random variables are independent given that other random variables have a certain value. This type of independence is called *conditional independence*. This concept works in the same way as regular independence, but with the extra condition that another variable has a specified value. Checking conditional independence

for random variables A and B given variable C can be done by means of the following formulas:

$$\begin{aligned} P(A|BC) &= P(A|C) \ , \\ P(B|AC) &= P(B|C) \ , \\ P(A \cap B|C) &= P(A|C)\, P(B|C) \ . \end{aligned}$$

The concept of conditional importance is very important for Bayesian networks, which will be the subject of the next section.

2.3 Bayesian Networks

2.3.1 Introduction

As mentioned in section 2.1.1, the problem with probabilistic reasoning systems is that once the size of models increases, the number of necessary probabilities grows exponentially. Until the 1980s not much progress was made. Pearl introduced the concept of Bayesian networks (Pearl, 1988). Bayesian networks exploit conditional independence assertions to be able to decrease the number of probabilities necessary for the model. The idea is to break up the original joint probability distribution into smaller conditional probability distributions. This is possible because of Bayes' rule (Equation 2.2). For example a J-PDF with four random variables A, B, C and D can be broken up into four smaller conditional probability distributions using Bayes' rule as follows:

$$\begin{aligned} P(A,B,C,D) &= P(A|B,C,D)\, P(B,C,D) \\ &= P(A|B,C,D)\, P(B|C,D)\, P(C,D) \\ &= P(A|B,C,D)\, P(B|C,D)\, P(C|D)\, P(D) \ . \end{aligned}$$

This is one of the 24 (4!) possible decompositions of the joint probability distribution. Every conditional probability distribution can be calculated using Bayes' rule and by applying marginalization. These decompositions are not very useful, as they still have many probabilities that need to be specified. The original J-PDF needs $2^4 - 1 = 15$ probabilities to be specified and the above decomposition needs $2^3 + 2^2 + 2^1 + 1 = 8 + 4 + 2 + 1 = 15$ probabilities to be specified. Nothing is gained by decomposing this distribution into conditional distributions. To be able to reduce the number of probabilities, conditional independence assumptions are necessary. If for instance, we assume that A and B are conditionally independent of C given variable D then the J-PDF can be decomposed into:

$$P(A,B,C,D) = P(A|D)\, P(B|D)\, P(C|D)\, P(D) \ .$$

This decomposition is more efficient: it only needs $2+2+2+1 = 7$ probabilities to be defined, which is less than half of the original number of necessary

probabilities. As the number of variables in the model grows, the difference between the number of probabilities of the original distribution and the decomposition typically becomes much larger. It is important to note that if the conditional independence assumptions necessary for the BN are not valid, the BN will not represent the original J-PDF accurately.

This is not always necessarily a bad thing. An example of a model with an extreme number of conditional independence assumptions is the *naive Bayes model*. In this model there are two types of variables: cause or class variables and evidence or feature variables. Normally a naive Bayes model will have one cause variable and a number of evidence variables. The model assumes that all evidence variables are conditionally independent of each other given the cause variable. This assumption makes it possible to decompose the J-PDF in the following way:

$$P(C, E_1, \ldots, E_n) = P(C) \prod_i P(E_i | C) \ . \tag{2.14}$$

The model is called "naive" because it is very unlikely that all evidence variables will be (conditionally) independent of each other in a real world situation. Interestingly enough, systems using a naive Bayes model can actually be quite effective (Russell & Norvig, 2003). Another very important advantage of using naive Bayes model are its space and time complexities. They are both linear ($O(n)$) in the number of evidence variables. So, when the number of variables doubles, so does the necessary amount of space to store the model and the amount of time necessary to perform inference. Compared to using the original joint probability distribution, this is an extreme improvement, for which the time and the space complexity are exponential ($O(2^n)$). Further improvements in space complexity can be achieved by adding some extra assumptions for the parent-child relationship. An example is the noisy-OR relationship, this relationship can further reduce the number of necessary probability entries but with the cost of the model having to satisfy the added assumptions necessary for noisy-OR to be effective (Russell & Norvig, 2003).

2.3.2 Definition

Naive Bayes models are a special case of Bayesian networks, and are mostly used for classification problems. Bayesian networks can be used for more complex problems. A formal definition of Bayesian networks by Russell and Norvig (2003) is the following:

> A Bayesian network is a directed graph in which each node is annotated with quantitative probability information. The full specification is as follows:
>
> 1. A set of random variables makes up the node of the network. Variables may be discrete or continuous.

2. A set of directed links or arrows connect pairs of nodes. If there is an arrow from node X to node Y, X is said to be a parent of Y.
3. Each node X_i has a conditional probability distribution $P\left(X_i \left| Parents\left(X_i\right)\right.\right)$ that quantifies the effect of the parents on the node.
4. The graph has no directed cycles (and, hence is an acyclic directed graph[1]).

The graph visualizes the conditional (in)dependence relationships between variables. If two nodes (variables) are connected by an arrow, they are conditionally dependent. If there is an arrow from node X to node Y, then node Y is conditionally dependent on node X.

The conditional probability distribution tables for each node define the conditional probabilities for each node given its parents. If a node does not have any parents, the node has a prior probability distribution. The total joint probability distribution is represented by the product of the conditional (and prior) probability distributions over all nodes. The J-PDF can be represented as follows:

$$P\left(x_1, \ldots, x_n\right) = \prod_{i=1}^{n} P\left(x_i \left| parents\left(X_i\right)\right.\right) \ . \tag{2.15}$$

This representation is correct when Bayes' rule is applied to break up the the joint probability distribution into conditional distributions, the ordering of the variables fits the decomposition, and the variables are conditionally independent of their predecessors in the variable ordering given their parents. The conditional independence relations between nodes of a BN fulfill the following conditions (Russell & Norvig, 2003):

- A node is conditionally independent of its non-descendants, given its parents.
- A node is conditionally independent of all other nodes in the network, given its parents, children, and children's parents (this subset of nodes is also know as the node's Markov Blanket).

An example Bayesian network, created by Pearl (1988), is displayed in Figure 2.1.

2.3.3 Inference

Bayesian networks are used to efficiently store and represent a joint probability distribution. But the purpose of the J-PDF is to be used to calculate probabilities for events; to perform inference. Posterior probabilities of

[1] Historically called a DAG

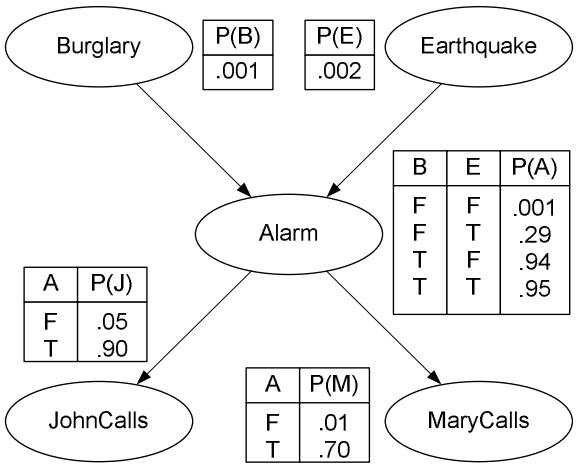


Figure 2.1: Example Bayesian Network

query variables are calculated given a set of evidence variables. A general probabilistic inference procedure of J-PDFs and BNs is given by Russell and Norvig (2003):

> *"Let X be the query variable, let $\boldsymbol{E}$ be the set of evidence variables, let $\boldsymbol{e}$ be the observed values for them, and let $\boldsymbol{Y}$ be the remaining unobserved variables, the query $\boldsymbol{P}(X \,|\, \boldsymbol{e})$[2] can then be evaluated as*
>
> $$\boldsymbol{P}(X \,|\, \boldsymbol{e}) = \alpha \boldsymbol{P}(X, \boldsymbol{e}) = \alpha \sum_{y} \boldsymbol{P}(X, \boldsymbol{e}, \boldsymbol{y}) \ , \qquad (2.16)$$
>
> *where the summation is over all possible combinations of the unobserved variables $\boldsymbol{Y}$."*

This inference procedure simply uses marginalization to sum out all the unobserved variables. The factor α is a normalization factor that ensures that the calculated probabilities of the distribution sum up to one. The factor is necessary because the procedure uses Bayes' rule, but leaves out the denominator of its equation. So α can be determined by:

$$\mathbf{P}(X \,|\, \mathbf{e}) = \frac{\mathbf{P}(X, \mathbf{e})}{P(\mathbf{e})} = \alpha \mathbf{P}(X, \mathbf{e}) \Rightarrow \alpha = \frac{1}{P(\mathbf{e})} \ .$$

It is not necessary to calculate α separately. When all the probabilities of the distribution $\mathbf{P}(X, \mathbf{e})$ are calculated, it suffices to sum all the probabilities to get $P(\mathbf{e})$. This is a valid procedure to perform inference on J-PDFs and BNs

[2]The bold notation used by Russell and Norvig for the letter P of the probability function means that a complete probability distribution is being calculated for the query variable(s) (in this case X).

but practically it is useless. It is very inefficient and when used for Bayesian networks, it has a worst case time complexity of $O(n2^n)$ (Russell & Norvig, 2003). This is even worse than performing the procedure on a full joint probability distribution. To improve efficiency, algorithms using dynamic programming techniques calculate intermediate results of the inference only once and reuse these results later. An example of an algorithm that uses dynamic programming techniques is the *Variable Elimination Algorithm*.

Variable Elimination Algorithm

The variable elimination algorithm works by evaluating query expression in right-to-left order. First the necessary summations (to sum variables out) are moved inwards as far as possible. Then for every conditional probability a factor is created. A factor is represented as a matrix or a vector and contains the probabilities needed for multiplication and summation to compute the final answer. Starting at the summation that is moved most inward the different factors are multiplied with each other using a special multiplication technique called a point-wise product. The result of a point-wise product of two factors is another factor, whose variables are the union of the variables of the two original factors. The probability entries of the new factor are the product of the of the probability entries of the two input factors (for a more in-depth explanation see Russell and Norvig (2003)). After the factors have been multiplied, the variable of the summation is summed out, giving a new factor containing the remaining variables. The process is repeated until the last variable is summed out. Now only one factor remains and after normalization of the entries of the factor the answer of the query is found.

Approximate Inference

The variable elimination algorithm and the general probabilistic inference process are examples of exact inference procedures. These procedures can compute a query exactly, without any error. Another form of inference is approximate inference. This type of inference can only calculate an approximation of the query, but can sometimes find this approximate result much faster. It is known that exact inference in Bayesian networks is #P-hard (Russell & Norvig, 2003), which means that performing exact inference in BNs is even more difficult than solving NP-complete problems. This is also the case for the approximate networks, but in the situations that approximate algorithms outperform the exact algorithms the time difference in computation can be very large. Popular approximate inference algorithms are based on randomized sampling algorithms, also known as Monte Carlo algorithms. Two examples for approximate inference are direct sampling and Markov chain sampling.

Direct Sampling Methods

Direct sampling methods are relatively easy; they use an easy-to-sample distribution to produce samples from a hard-to-sample distribution. In this case, the hard-to-sample distribution is the joint probability distribution represented by the Bayesian network. There are some different variations of direct sampling methods, differing in complexity and quality. The simplest one samples from the prior distribution, beginning at the root variables of the network and moving down to the leaves. At every variable, it generates a random number and using the prior distribution of the variables it assigns a value to the variable. If the variable has parents, the conditional distribution is conditioned on the values of the parents. After all variables have been assigned a value, a sample has been generated. This sample can be assigned a probability by multiplying the prior probabilities of the values of all the variables. This probability can be used as an estimate of query. If the sampling process is repeated a large number of times, more accurate estimates of the real probability of the query can be acquired. To get an estimate of a query, probabilities necessary for the query have to be sampled. For a query with query variable X and evidence variables $\mathbf{e}$, this means estimating the probability distribution $\mathbf{P}(X, \mathbf{e})$. Since sampling is a random process samples will be generated that will have evidence that is not consistent with the desired query. These samples are not very useful for the approximation of the query and to get better estimates these samples have to be dealt with.

Rejection Sampling

One way of dealing with this problem is a method called rejection sampling. The prior probability distribution is sampled N times and every sample which is not consistent with the evidence of the query is discarded. The remaining samples are used to obtain the estimate of the probability distribution $\mathbf{P}(X, \mathbf{e})$. A frequency table is created where the number of times X takes a certain value is counted. This table is normalized to get the estimate of the probability distribution. The problem with rejection sampling is when there are many evidence variables, the number of samples with consistent evidence will drop exponentially. The exponential drop is caused by the exponential growth of the possibilities of the evidence. This means that there will be only a very small number of samples with consistent evidence for the calculation of the estimate of the probability distribution of the query. The result is that the rejection sampling method is simply unusable for large, complex problems (Russell & Norvig, 2003).

A method that gives better results and does not throw away any of the samples is likelihood weighting. It calculates the likelihood of a sample and weights the sample with this likelihood. Now, samples with inconsistent evidence will only have a small influence on the final result. When the

number of evidence variables increases, likelihood weighting will also perform worse, again because of the exponential growth of evidence possibilities.

Markov Chain Simulation

Totally different from direct sampling algorithms is Markov chain simulation. Events created by direct sampling algorithms are independent from each other. For every event, the algorithm starts again with the prior probability distribution. The Markov Chain Monte Carlo (MCMC) algorithm generates a new event by making a random change to the current event. More accurately (Russell & Norvig, 2003):

> *"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."*

In this context, a state is a complete assignment of values to all variables. The MCMC algorithm walks through the state space by randomly assigning values to non-evidence variables. The evidence variables are kept constant during the process. Every state visited represents a sample for the estimation of the query probability. The number of times a query variable has a certain value is used for computing the probability estimate. The values are normalized by the total numbers of states visited.

The MCMC algorithm works because of properties of Markov chains. When the process is run for a long time, the process stabilizes into a dynamic equilibrium. When this equilibrium is reached, the time spent in each state is proportional to the posterior probability of being in this state. So, as with the other sampling algorithms: the longer the algorithm is run, the better the estimate of the query probability will become.

2.4 Importance Sampling

2.4.1 Monte Carlo

During the 1940s, at the Los Alamos laboratory in New Mexico, stochastic simulations were used to numerically evaluate integrals for the Manhattan project. These stochastic simulations were code-named "Monte Carlo" (MC) (E. C. Anderson, 1999). The MC method can be used to estimate integrals. A general definition of MC is (E. C. Anderson, 1999):

> *"Monte Carlo is the art of approximating an expectation by the sample mean of a function of simulated random variables."*

The method makes use of properties of the expected value of a random variable (see Section 2.2.3). Most importantly, the general form of calculating

the expected value is being used (Equation 2.10). The expected value can be estimated by calculating the average of N samples, in general:

$$E\left[g\left(X\right)\right] \approx \frac{1}{N}\sum_{i=1}^{N} g\left(x_i\right) \ . \tag{2.17}$$

The right side of the equation is called the Monte Carlo estimator. If $E\left[g\left(X\right)\right]$ exists, then a result from the weak law of large numbers states that for any arbitrarily small ϵ

$$\lim_{N\to\infty} P\left(\left|\frac{1}{N}\sum_{i=1}^{N} g\left(x_i\right) - E\left[g\left(X\right)\right]\right| \geq \epsilon\right) = 0 \ .$$

It explains that when N increases, it will become very unlikely that the Monte Carlo estimator will deviate much from the expected value. To estimate an integral like $\int_a^b g\left(x\right) dx$, we change it into an expected value for a random variable. To do this, we add the probability density function of the random variable we are going to use for the sample process to the integral:

$$V\int_a^b g\left(x\right) f\left(x\right) dx \ ,$$

where V is the total volume to be integrated over (here $b-a$). The EV can now be estimated with the summation in Equation 2.17. The result serves as an estimation for the integral. The estimator for this integral would look like:

$$\frac{V}{N}\sum_{i=1}^{N} g\left(x_i\right) \ .$$

The quality of the estimate depends on the variance of the random variable. The smaller the variance, the better the estimate will be of the real value of the integral.

2.4.2 Importance Sampling

This is where *importance sampling* (IS) comes in. IS, also developed at the Los Alamos lab (H. L. Anderson, 1986), is a technique to reduce variance. This is achieved by adding an importance function to the integral, the integral now looks like:

$$V\int_a^b \frac{g\left(x\right)}{f\left(x\right)} f\left(x\right) dx \ . \tag{2.18}$$

The Monte Carlo estimator now looks like:

$$\frac{V}{N}\sum_{i=1}^{N} \frac{g\left(x_i\right)}{f\left(x_i\right)} \ .$$

Now, the estimator will almost surely converge towards the real expected value (and the real value of the integral) if the following weak assumptions are met (Geweke, 1989):

Assumption 1 $g(x)$ is proportional to a proper probability density function defined on Ω.

Assumption 2 $\{X_i\}_{i=1}^{\infty}$ is a sequence of i.i.d. random samples, the common distribution have a probability density function $f(x)$.

Assumption 3 The support of $f(x)$ includes Ω.

Assumption 4 The integral exists and is finite.

IS now will assign more weight to regions where $g(x) > f(x)$ and less weight to the regions where $g(x) < f(x)$ to correctly estimate the integral (Yuan & Druzdzel, 2006). The only assumption we have any control over is assumption 3. We can choose which importance function we are going to use for $f(x)$ as long as it contains the domain Ω we want to sample. The other assumptions are either inherent properties of the integral or characteristics of Monte Carlo simulation.

Theoretically there exists an optimal importance function that will lead to 0 variance. Rubinstein (1981) has proven that if $g(x) > 0$, then the optimal importance function is:

$$f(x) = \frac{g(x)}{I}, \tag{2.19}$$

where I is the result from the integral of Equation 2.18. The problem is that this importance function can only be found by evaluating the integral that we are trying to find an importance function for. The result does not look very useful, but it does say that the importance function should preferably be proportional to the function $g(x)$. Or it should at least approximate $g(x)$ as close as possible.

2.4.3 Advantages and Disadvantages

Using IS for a problem has consequences, some can be positive and other are negative. Depending on the situation, it may or may not be desirable to use IS. Some advantages of IS are the following:

- With respect to normal MC methods, using IS can reduce the variance of the sampling process. This leads to more accurate sampling. The proportion of samples that fall into the desired area that has to be examined is larger and now less samples have to be discarded. The importance function is used to "direct" more samples towards the more important regions in the sample space. This way IS improves sampling efficiency because now fewer samples are necessary to reach the desired level of accuracy (Yuan & Druzdzel, 2006).

- Because of the ability of the importance function to steer samples towards certain areas of the sample space, this ability can be used to direct samples to areas where samples normally are probabilistically unlikely to occur when using normal Monte Carlo methods. This process is known as *rare-event simulation* (Denny, 2001). For example it has applications in physics and communication systems (Smith, Shafi, & Gao, 1997).

- IS allows to generate samples from a difficult-to-sample distribution.

Using IS can also have some disadvantages or cause some difficulties. Two disadvantages or difficulties are listed:

- To get an optimal result, the importance function should resemble the original function as much as possible. This means that to get good results it is necessary to create a new importance function for every problem that IS is to be applied to. The fact that importance functions are very problem specific means that it is hard to make a general purpose IS implementation and it will take extra time to find an importance function that fits the problem well.

- Another problem is that when the original problem gets more complex, it will become increasingly harder to find an importance function that will be easy to sample from. This is due to the requirement that to get a good estimator, the importance function must resemble the problem function. The shape of the importance function may make easy sampling impossible and may cause the sampling process to become very slow when generating samples from the importance function. However, if a simpler importance function is chosen that does not fit the problem function very well, but is easy to sample from, the generation of samples might be fast but the overall result could be worse. Because it is now possible that a large number of samples may need to be discarded because they do not "fit" the problem function, it could be the case that using IS is slower and less accurate then simply using plain MC sampling.

2.5 Linear Programming

2.5.1 Introduction

Linear Programming is a mathematical framework that emerged from different fields, most prominently the fields of military, economy, industry, and mathematics. During and shortly after the second world war, a large number of research groups was working on describing a diverse group of large logistical problems in different areas. In 1947, the area of LP emerged by

the creation of the *simplex* algorithm by Dantzig (1948). The availability of this algorithm gave a huge boost to the different research areas, because now there was an unifying mathematical framework that could be used to model and solve these problems. Very quickly LP was being used in many areas, solving problems ranging from planning crop rotations to planning large scale military actions to routing ships between harbors and the assessment of the flow of commodities between industries of the economy (Dantzig, 1966).

2.5.2 Linear Programming Model

LP belongs to the family of optimization techniques, more accurately it is used to describe constrained optimization problems. A constrained optimization problem has four main elements (Chinneck, 2001):

- Variables, which represent factors of the problem that can be changed or controlled.
- Objective function, which is a mathematical function that has the variables of the problem as input and maps this to a numerical result that represents the goal of the optimization procedure.
- Constraints, which are mathematical expressions used to describe rules and limitations that apply to the variables of the system.
- Variable bounds, which are used to limit the the values the variables can take.

These elements are defined for general constrained optimization problems. LP is a subset of these problems and makes some extra assumptions about the elements. LP assumes that both the objective function and the constraints are linear functions. So, only linear models are used to model problems. Although this looks like a very severe restriction on the types of problems that can be modeled with LP, this is not the case. A very large, diverse number of problems can be solved by using LP. The method is the most widely used method of constrained optimization.

Building a Linear Model

Creating a linear model means decomposing a problem into a number of elementary functions called *activities*. Dantzig (1966) defines an activity as a "black box" which has inputs and inputs. Raw materials can flow into the box and the processed result, whatever it may be, flows out of the box. The different types of flows of products or materials are called items. The quantity of the output of each activity is called the activity level. To change the activity level it is necessary to change the flows into and out of the activity.

Dantzig has stated four assumptions necessary for LP models:

Assumption 1: Proportionality The quantities of flow of items in and out of an activity should be proportional to the activity level.

Assumption 2: Nonnegativity Negative quantities of activities are not possible.

Assumption 3: Additivity It is required for each item that the total amount specified by the system as a whole equals to the sum of the amounts flowing into the various activities minus the sum of the amounts flowing out.

Assumption 4: Linear Objective Function The objective function is the (weighted) sum of the different items that contribute either positive or negative.

These assumptions must be satisfied when building a LP model. Building a model consists of performing five steps (Dantzig, 1966):

1. Define the Activity Set.
2. Define the Item Set.
3. Determine the Input-Output Coefficients.
4. Determine the Exogenous Flows.
5. Determine the Material Balance Equations.

After performing these steps, the LP model is finished and it can be solved by a LP solver.

2.5.3 Simplex Algorithm

Solving a LP model means maximizing or minimizing the objective function while satisfying the constraints and the variable bounds. The most frequently used algorithm for this action is the simplex algorithm defined by Dantzig in 1947. Even now it is the most popular algorithm, despite its exponential worst case complexity and the availability of guaranteed polynomial algorithms. The simplex method exploits some inherent properties of linear programs. One of those is that a feasible linear program, a program without errors in the the constraints, forms a convex polyhedron in the space spanned by the variables of the linear program. The constraints of the linear program describe what the convex polyhedron looks like. A convex polyhedron is a geometrical shape. A very simple example of a convex polyhedron is a cube, but more complex versions exist. Another well know example is the truncated icosahedron, which is the shape of a soccer ball.

These shapes are both examples of three dimensional convex polyhedra, but convex polyhedra also exist in higher dimensions. The simplex algorithm works by moving from corner point-to-corner point of the convex polyhedron. The simplex algorithm was designed to take into account three key properties of linear programs (Chinneck, 2001):

1. The optimum point is always at a feasible corner point. By feasible is meant that all the constraints and bounds are satisfied. This is the result of working with linear constraints which are straight lines, or more general: straight hyperplanes, and the highest point will be at the intersection of two lines, i.e., a corner point.

2. If a corner point feasible solution has an objective function value that is better than or equal to all adjacent corner point feasible solutions, then it is optimal.

3. There are a finite number of corner point feasible solutions. This means that the algorithm will eventually stop after looking at all corner points of the polyhedron and then must have found the optimal value.

Other algorithms exist that do not move from corner point-to-corner point. Examples are the ellipsoid algorithm by Leonid Khachiyan in 1979 and interior-point algorithms, pioneered by Narendra Karmarker in 1984. These algorithms have a polynomial worst case complexity, but only become more efficient than the simplex algorithm when performed on very large linear programs.

2.5.4 Applications

Linear programming has many applications. It is an optimization method, so any problem where parameters may need optimization under the restriction of (linear) constraints LP may be applicable. There are many areas where LP has been used with success, examples can be found in the areas of economics, logistics, production, agriculture, military and many others. When used correctly LP can be used to either reduce expenses or increase profits, this has been one of the reasons that it has been used in so many different areas.

One famous example is the input-output table with 500 linear equations that describe 170 sectors of the US economy, developed by Wassily Leontief in 1949. He received the Nobel prize in economics in 1973 for this work. His model describes the flows of goods between the different sectors and can be used the predict the change in demand of prerequisite materials if the production volume of the end product changes. His work stimulated research in economic planning and the collection of large amounts of empirical data.

Chapter 3

Research

3.1 Introduction Non-Invasive Elicitation Method

The research assignment is mostly based on the paper written by Druzdzel and van der Gaag (1995). The paper, "Elicitation of Probabilities for Belief Networks: Combining Qualititative and Quantitative Information", introduces a method that can combine information from various types of sources and does this in a way so the elicitation process is non-invasive. For informative purposes the paper is reviewed here. The section is concluded with a discussion of the strong and weak points of the proposed method.

3.1.1 Summary

Introduction

The authors give a short introduction to Bayesian networks and note that the construction of a BN consists out of two parts: the qualitative part, the construction of the network structure, and the quantitative part, the creation of the CPTs of the nodes in the BN. The quantitative part is considered the hardest part, eliciting probabilities from experts can be difficult because of an expert's possible reluctance of specifying exact numbers for the probabilities of the CPTs. Sometimes the available information cannot be used directly for CPTs or experts cannot provide exact numbers. But an expert may be able to define an interval within the real value should lie. Other types of probabilistic information exist than just pure probabilities and different schemes have been developed for using these other types of probabilistic information for reasoning under uncertainty. The authors note that none of these schemes can handle all types of probabilistic information and that a unifying principle that handles the various types probabilistic information has been lacking so far.

The authors propose a method that can use both qualitative and quantitative probabilistic information that describes a yet unknown probability distribution Pr over a set of variables V. The approach of the authors is to consider the distribution hyperspace of all possible joint probability distributions over V. A point somewhere in this hyperspace will be the true probability distribution, Pr, over the set of variables V. If there is no information available, qualitative or quantitative, then the Pr can be any point in the hyperspace. Once more information is known about Pr, some of the probability distributions in the hyperspace will become incompatible with this information. Using the method proposed by the authors, probability elicitation can be looked upon as constraining the distribution hyperspace as much as possible to find the true distribution Pr. The authors now express all information for the distribution Pr as constraints for the hyperspace, and, assuming that all compatible distributions are equally likely, then derive 2^{nd} order probability distributions over the probabilities of the distribution Pr. These 2^{nd} order distributions can then be used for determining the probabilities of the joint probability distribution. Since the method allows the use of various types of probabilistic information it is possible to use any information the expert is willing to state. This allows for the process of eliciting probabilities to be non-invasive.

Example

Throughout their paper the authors have used an example Bayesian network. This network is a very simple model of the causes of HIV virus infection. The network has 4 variables: *HIV infection (H)*, *needle sharing (N)*, *sexual intercourse (I)*, and *use of a condom (C)*. The network is illustrated in Figure 3.1. To make this network complete the CPTs of the nodes need to

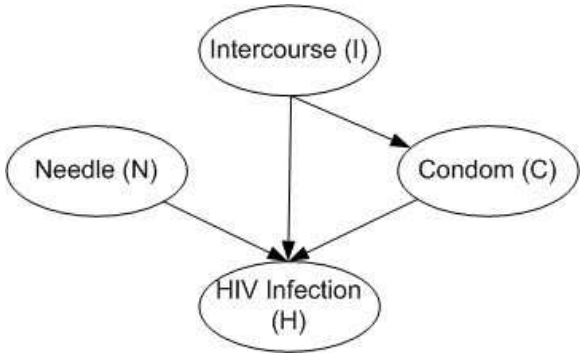


Figure 3.1: Example Bayesian network for HIV infection

be filled with the necessary probabilities. Some of these probabilities can be estimated directly by using statistical data, examples are the probability of having HIV $Pr\,(h)$, frequency of sexual intercourse $Pr\,(i)$, and condom

usage $Pr(c|i)$. Others can be determined by using some common sense. Condoms are primarily used during intercourse, so $Pr(\bar{i}|c)$ will be very close to 0. Also some more indirect information is available: looking at the general population there a lot more people that are sexually active than there are people using intravenous drugs. This means that it is very likely that $Pr(i) > Pr(n)$.

The types of probabilistic information described above are of a quantitative or (semi-)numerical nature. Also some information exists on the subject that is more qualitative of nature. The authors mention that sharing a needle and sexual intercourse with a HIV carrier will make HIV infection more likely. This interaction is known as a *positive additive synergy*. The presence of N and I make H more likely to be true. An example of a *negative additive synergy* would be using a condom during intercourse. The presence of C and I make H less likely to be true. Other types of qualitative information exist, but these are discussed later.

Canonical Form

The basic idea of the approach of the authors is to have a canonical form for the interpretation of probabilistic information. Their form builds on the property that any joint probability distribution on a set of variables V is uniquely defined by the probabilities of all possible combinations of values for all variables from V. With all these values known, any probability over the set V can be computed by using marginalization (Section 2.2.4) and conditioning (Section 2.2.2). The authors call combinations of values for all variables *constituent assignments*. The probabilities of constituent assignments in a joint probability distribution are called its *constituent probabilities*. The authors look upon the set of all possible joint probability distributions on V as spanning a hyperspace whose dimensions correspond with constituent probabilities.

Any information about the true distribution Pr can now be represented as a system of (in)equalities with the constituent probabilities as unknowns. Any solution of this system will be a joint probability distribution that is compatible with all the specified probabilistic information. If there are no solutions, then the provided information is inconsistent. The authors have based their view of probability on the early work of Boole (1958) on the foundations of probability theory.

For the canonical form the authors have introduced some notational conventions:

> *"We take* $V = \{V_1, \ldots, V_n\}, n \geq 1$, *to be a set of variables, where each variable* V_i *can take one of* k_i *values. We will use* v_{i_j} *to denote* V_i *taking the* j*-th value from its domain,* $j = 1, \ldots, k_i$*. Note that the set of all constituent assignments for* V *comprises* $\prod_{i=1,\ldots,n} k_i$ *elements."*

Now, using these conventions an assignment b for an arbitrary subset of variables from V and its unknown probability $Pr(b)$ can be considered. The assignment b can be written as a disjunction of constituents c_i using basic logical laws. In fact, there exists a unique set indices $I_b \subseteq \{1, \ldots, k\}$, called the *index set* for b, such that $b = \bigvee_{i \in I_b} c_i$. An example could be that b consists out of the disjunction of the constituents c_1, c_2,and c_5. The result would be that $b = c_1 \vee c_2 \vee c_5$. Since all constituent assignments are mutually exclusive, the probability $Pr(b)$ can be expressed as the sum of the probabilities of the constituent assignments b is built from. So from $Pr(b) = \sum_{i \in I_b} Pr(c_i)$ the authors found that $Pr(b)$ can be expressed as

$$d_1 x_1 + d_2 x_2 + \cdots + d_k x_k \ ,$$

where $x_i = Pr(c_i)$, $i = 1, \ldots, k$, and $d_i = 1$ if $i \in I_b$ and $d_i = 0$ otherwise.

Interpretation of Probabilistic Information

With the canonical form defined, now different types of probabilistic information can be interpreted and translated into the canonical form so that it can be used for the elicitation process. The most basic form of probabilistic information that has to be interpreted are the axioms of probability. A point in the hyperspace must be compatible with the axioms or it will not be a valid probability distribution.

One of the axioms states that the probability of a true event, $Pr(true)$, is equal to 1. This means that the sum of the probabilities of the distribution should add up to 1. This axiom can be translated into the canonical form by the equality

$$x_1 + \cdots + x_k = 1 \ ,$$

where $x_i = Pr(c_i), i = 1, \ldots, k$. Another axiom states that any probability must be a nonnegative, real number. So this means that all constituent probabilities must be larger or equal to 0. In canonical form this can be expressed as

$$x_i \geq 0 \ ,$$

for $i = 1, \ldots, k$. Also all probabilities must be smaller than or equal to 1, but this is implied by the two statements above.

The authors have defined some types of probabilistic information of a quantitative nature. These three types are:

1. Point estimate.
2. Probability intervals.
3. Comparison.

A *point estimate* is a statement in the form $Pr(b) = p, 0 \leq p \leq 1$, where b is an assignment for an arbitrary subset of variables. Let I_b be the index set for b. Then the point estimate is expressed in canonical form as

$$d_1 x_1 + \cdots + d_k x_k = p \ ,$$

where $x_i = Pr(C_i), i = 1, \ldots, k$, and $d_i = 1$ if $i \in I_b$ and $d_i = 0$ otherwise. For a conditional probability $Pr(b_1 | b_2)$ a point estimate is in roughly the same form, but the conditional probability is first transformed into $\frac{Pr(b_1 b_2)}{Pr(b_2)}$ (using Equation 2.2) and the final canonical form will become:

$$Pr(b_1 b_2) - p \cdot Pr(b_2) = 0 \ .$$

Also because of Equation 2.2 an extra constraint has to be added to the system. $Pr(b_2)$ has to be larger than 0 or else the conditional probability will become infinite and thus invalid. So, the extra added constraint is the inequality:

$$Pr(b_2) > 0 \ .$$

Probability intervals and *comparisons* have similar canonical representations. A probability interval has instead of one bound, an upper and a lower bound: $p_1 \leq Pr(b) \leq p_2$. A comparison is between two probabilities and has the form: $a_1 \cdot Pr(b_1) \leq a_2 \cdot Pr(b_2)$, where a_1 and a_2 are non-negative real numbers.

The authors have also defined some types of probabilistic information that are qualitative of nature, these are:

1. Qualitative influences.
2. Qualitative synergies.

A *qualitative influence* is a symmetric property describing the sign of probabilistic interaction between two variables V_1 and V_0, and builds on an ordering of these variables' values. The authors have described three different types of qualitative influences: positive, negative, and zero qualitative influences. A positive qualitative influence expresses that when a higher value of V_1 is observed, it is more likely that higher values for V_0 will also be observed. This is denoted by $S^+(V_1, V_0)$. The condition is that this relation is valid if and only if for all values v_{0_m} of V_0, for all pairs of distinct values

$v_{1_i} > v_{1_j}$ of V_1, and for all possible assignments b for the set of V_0's direct predecessors other than V_1, it is valid that

$$Pr\left(V_0 \geq v_{0_m} \left| v_{1_i} b\right.\right) \geq Pr\left(V_0 \geq v_{0_m} \left| v_{1_j} b\right.\right) .$$

For the negative and zero qualitative influences a similar expression exists. The only difference is the expression type. For the negative influence the left probability must be smaller or equal to the right one, and for zero qualitative influence it becomes an equation instead of a inequality.

To express the statement in canonical form a set of inequalities is needed. A inequality is necessary for each combination of one value v_{0_m} of V_0, one pair of values v_{1_i}, v_{1_j} of V_1, and one assignment b of V_0's other predecessors than V_1. Every inequality has the form

$$\sum_{l=m}^{k_0} Pr\left(v_{0_l} \left| v_{1_i} b\right.\right) \geq \sum_{l=m}^{k_0} Pr\left(v_{0_l} \left| v_{1_j} b\right.\right) ,$$

and there are $\left(\begin{array}{c} k_1 \\ 2 \end{array} \right) \cdot (k_0 - 1) \cdot K$ such inequalities, where K is the number of possible assignments for the set of direct predecessors of V_0 other than V_1. As these inequalities involve conditional probabilities, each of them creates two additional inequalities to ensure that that conditional probabilities exist. Another type of qualitative probabilistic information is a *qualitative synergy*. The authors describe two types of synergies: additive synergies and product synergies. Both these types come in the form of positive, negative, or zero variants. An additive synergy describes the joint influence of two variables V_1 and V_2 on a third variable V_0, and, similarly to qualitative influence, builds on an ordering of these variables' values. A positive additive synergy of V_1 and V_2 with respect to V_0 expresses that the joint influence of V_1 and V_2 is greater than the sum of their individual influences. This is denoted by $Y^+\left(\{V_1, V_2\}, V_0\right)$. The condition is that this relation is valid if and only if for all values v_{0_m} of V_0, for all pairs of values $v_{1_i} > v_{1_j}$ of V_1 and $v_{2_{i'}} > v_{2_{j'}}$ of V_2, and for all possible assignments b for the set of V_0's direct predecessors not including V_1 and V_2, it is valid that

$$Pr\left(V_0 \geq v_{0_m} \left| v_{1_i} v_{2_{i'}} b\right.\right) + Pr\left(V_0 \geq v_{0_m} \left| v_{1_j} v_{2_{j'}} b\right.\right)$$

$$\geq Pr\left(V_0 \geq v_{0_m} \left| v_{1_i} v_{2_{j'}} b\right.\right) + Pr\left(V_0 \geq v_{0_m} \left| v_{1_j} v_{2_{i'}} b\right.\right) .$$

As with the qualitative influences, negative additive synergy and zero additive synergy are defined in a similar manner. Also the above statement is transformed into canonical form in the same way as with the qualitative influences. The number of inequalities that the statement transforms in is different, there are $\left(\begin{array}{c} k_1 \\ 2 \end{array} \right) \cdot \left(\begin{array}{c} k_2 \\ 2 \end{array} \right) \cdot (k_0 - 1) \cdot K$ inequalities, where K is the

number of possible assignments for the set of direct predecessors of V_0 other than V_1 and V_2. Again these inequalities involve conditional probabilities, so extra inequalities are added to ensure that all the conditional probabilities will be valid.

Another form of qualitative synergies are *product synergies*. Product synergies describe the interaction between two variables V_1 and V_2 conditional on their common descendant V_0 and expresses the sign of what is known as inter causal influence between V_1 and V_2. The most common type of product synergy is the negative product synergy. This type captures the notion of "explaining away." The authors state that the variables V_1 and V_2 exhibit negative product synergy with respect to a particular value v_{0_m} of variable V_0, written as $X^{-}\left(\{V_1, V_2\}, v_{0_m}\right)$, if for all pairs of values $v_{2i} > v_{2_j}$ of V_2 and for all possible assignments b for the set of V_0's direct predecessors not including V_1 and V_2, it is valid that

$$Pr\left(V_1 \geq v_{1_i} \left| v_{2_i} v_{0_m} b\right.\right) \leq Pr\left(V_1 \geq v_{1_i} \left| v_{2_j} v_{0_m} b\right.\right) \ .$$

Positive and zero product synergy are defined in a similar manner and the translation to the canonical form is performed in the same way as qualitative influences and additive synergies. A difference between product synergies and additive synergies is that product synergies are with respect to separate values of the common effect V_0. Thus there are as many product synergies as there are values of V_0.

Elicitation of Probabilities

To derive the 2^{nd} order distributions for the probabilities to be assessed the authors have proposed to use sampling. For the computation of the 2^{nd} order distributions randomly selected points from the distribution hyperspace, under the assumption that all points in the hyperspace are equally likely to be the true distribution, are selected. Every selected distribution is verified to be sure that it is compatible with all available information. All distributions that are compatible with all constraints are collected and used for the generation of the 2^{nd} order distributions over the probabilities. The process is computationally expensive as it involves generating and investigating joint probability distributions. These J-PDFs are described by their constituent probabilities and the number of these probabilities is exponential in the number of variables in the BN. The authors have created a very simple prototype for the method, but it is very straightforward and there is a lot of room for algorithmic improvement.

When there is very restrictive information available about the real distribution, a large number of the randomly selected distributions will be not be compatible with all the available information. This makes the method very inefficient. To improve the ratio of compatible samples the authors propose a preprocessing step prior to the sampling process. They would like

to use all the linear constraints to calculate upper and lower bounds for all the constituent probabilities. This step will restrict the sampling space and thus will exclude some samples that certainly would have been rejected.

To battle the computational complexity of the method the authors propose to divide the BN into smaller sub networks and to perform the method on each of the sub networks. The authors propose to transform the BN into a chordal graph[1], using the transformation scheme designed by Lauritzen and Spiegelhalter (1988), and then to divide this graph into smaller subgraphs. Besides the computational complexity of the method, another reason to divide a BN into smaller parts is that human experts typically express information about short causal reasoning chains and feel uncomfortable when forced to provide more global information.

Discussion

The authors state that even though a non-invasive method of collecting information from experts may be less prone to conflicts, the constraints elicited may still turn out to be inconsistent. Conflicts need to be detected and dealt with. The authors see an opportunity here to use the conflicts to refine the elicitation by confronting the expert with the conflicting statements.

3.1.2 Evaluation

The method presented in (Druzdzel & van der Gaag, 1995) has been examined thoroughly, and advantages and disadvantages of using the elicitation method have been identified. Advantages of using the method are listed below:

- The method allows for non-invasive elicitation, which means that it can accept many different types of probabilistic information as input. Any probabilistic information, the expert may have can directly be used by the method without any conversions. The method translates all the different types of information into one canonical form that is used for further computations.

- The method's use of a canonical form allows for flexibility when new types of probabilistic information become available. the method can easily be adapted to be able to handle the new information types as input.

- The method allows for the use of quantitative and qualitative information. Generally, other elicitation methods are usually restricted to use probabilistic information that is either quantitative or qualitative.

[1] A chordal graph is a graph that is triangulated, meaning that if a graph contains cycles larger than 3, extra vertices are added to turn these cycles into triangles.

The ability of the elicitation method to use both is the result of using the described canonical form for the constraints.

Disadvantages of using the elicitation method are:

- The method uses sampling to generate the joint probability distributions used to determine the histograms. If the number of variables the input Bayesian network has increases, the dimension of the sample space will increase exponentially in the number of variables. If the number of dimensions of the sample space becomes sufficiently large, the method may become intractable. Finding samples that satisfy all constraints will become increasingly difficult and the method as proposed in the paper currently has a very basic sample generator that is very inefficient.

- Another problem with the method as it was proposed in the paper is that it does not currently allow for the detection or resolution of conflicts between constraints. In the situation that a conflict occurs between constraints the method will run indefinitely.

The method looks very interesting and promising, but it definitely needs improvement to be sure that the necessary computation time stays reasonable. Improvements must be made to the sampling process, as few samples as possible should be rejected. Also conflict detection and resolution methods are necessary to ensure that the method will terminate.

3.2 Decomposing a Bayesian Network

The elicitation method works by sampling the hyperspace of possible joint probability distributions of the Bayesian network provided by the expert. The problem is that if the joint probability space for the whole BN would be used for sampling, the method would become computationally intractable in space and time. The reason BNs are used in the first place is that they are far more efficient at representing the complete J-PDF.

An obvious solution is to decompose the BN into smaller sub networks that contain fewer variables and thereby have smaller J-PDFs, that are only valid for their respective subnetwork. This way the J-PDFs that the method has to work with have a more manageable size. In the paper a transformation scheme designed by Lauritzen and Spiegelhalter (1988) is mentioned. This transformation extracts from the BN's directed acyclic graph (DAG)[2] an undirected triangulated graph and creates a tree whose vertices are the

[2] A directed acyclic graph is a directed graph, a graph that only has directed arcs (arcs that are only valid in one direction), and it does not have a set of nodes that point at each other in such way that together they form a circle of nodes.

cliques[3] of this triangulated graph (Neapolitan, 1990). The original BN is turned into a chordal graph, which has the property that the whole J-PDF nicely marginalizes over the cliques created by the transformation.

However, for the expert who has created the BN, this new graph may not be as easy to understand and to create meaningful probabilistic statements for. Although the transformation guaranties that no causal links are broken, for the expert the original BN may be more clear.

Another way of decomposing a BN is proposed, which might be more clear for the expert and is easier to implement. The idea is to break up the BN into families; sub networks that consist out of a node and its parents. Using family networks (FNs), the expert can focus on providing information for a only a small part of the network without having to worry about other nodes.

Breaking up a Bayesian network into FNs is very easy. There is only one way of breaking it up into its set of family networks. The process is linear in the number of variables the BN has. Every node is checked to see if it has parents. If this is the case, then a family network must be created containing this node and its parents. Once all nodes have been checked and all the family networks have been generated, each of these networks can now be processed by the elicitation method.

It is important to note that a single node can appear in multiple family networks, once as the child and zero or more times as parent of other nodes. This means that after the J-PDFs of the family networks have been estimated by the elicitation method, somehow this information will have to be merged into the J-PDF of the original BN. Using this procedure will also influence the accuracy of the method, because some causal links are temporarily removed from some family networks. This will happen in the case that one of the parents of the child node is also a child in another family network. The merging process (Section 3.7) addresses this problem.

3.3 Translation of Expert Statements into Constraints

With the BN decomposed, the expert can provide probabilistic statements per family for each family. Once the expert is done, the statements will have to be translated into constraints for the probability hyperspace. These constraints are, as stated in Section 3.1, equations and inequalities that have the constituents as variables. Since the constraints are going to be used to evaluate samples generated by the system, it is important that the form chosen for the constraints makes this computation possible and preferably also easy. It has been decided to represent the equations/inequalities by

[3] A clique is a set of nodes that are completely connected to each other. In this case every node of the clique is connected to every other node of the clique by an arc.

using binary expression trees. The reason for this choice was that binary expression trees are most commonly used for similar problems, so a lot of reference material would be available.

To translate the statements into binary expression trees, a parser has been used. The SMILE library already had a basic parser implemented for the parsing of expressions in text form into expression trees. This parser was extended so that it was be able to parse the probabilistic statements presented in text form. Letting the expert write down his probabilistic statements in text as input for the system might be a bad idea. In this case the system must be able to handle errors in statements, which if detected too late could cause the program to crash. Therefore a better idea is to create a user interface that the expert uses to create statements and that these statements are automatically converted into the text format necessary for the parser. Perhaps, if this is desirable, the option could be left open to add statements directly in text form, for "expert" users.

The parsing of the different types of probabilistic statements was designed in such a way that the more complex probability statements make use of the simpler ones. An example is the qualitative influence statement. This statement can first be translated into a number of (quantitative) probability statements. The number of statements depends on the number of variables in the family network. These probability statements can be translated into one or more equations/inequalities consisting out of constituents and the basic arithmetic elements. These equations are the final form and are represented using expression trees. Instead of generating the resulting expression trees directly from the qualitative statement, first the probability statements are generated, which are then parsed and then from each probability statement the resulting trees are generated. A similar mechanism is necessary for conditional probability statements. An extra constraint must be added to ensure that the conditional part of the probability has a probability larger than 0. A simple example would be: the statement $P(A|B) > 0.2$ needs an extra statement $P(B) > 0$ because otherwise the conditional probability does not exist (this is due to Equation 2.2).

3.4 Identification of Probability Bounds

When the system has acquired the constraints from the expert, it could start the sampling process, but this will be quite inefficient. Druzdzel and van der Gaag (1995) propose that first a preprocessing step should be performed to reduce the size of the sample space. They have envisioned using linear programming (LP) to tighten the bounds for every constituent.

To calculate the upper and lower bounds for each constituent, the proposed LP method has been used.

Upper Bound

$$\begin{aligned} &Maximize && P(c_i) \\ &Subject\ to \\ & && D\underline{c} = \underline{p} \\ & && \underline{c} \geq 0 \end{aligned}$$

Lower Bound

$$\begin{aligned} &Minimize && P(c_i) \\ &Subject\ to \\ & && D\underline{c} = \underline{p} \\ & && \underline{c} \geq 0 \end{aligned}$$

Figure 3.2: Linear Program definition for calculating probability intervals for constituents

Figure 3.2 shows the linear programing problems that should be solved to calculate a probability interval for a constituent:

Where c_i is the constituent where the bounds are calculated for, D is the matrix containing the linear constraints, $\underline{p}$ are the respective right hand side values of the constraints and $\underline{c}$ is a vector containing all the constituents. This problem can now be solved by every standard LP algorithm. To calculate probability bounds, this process has to be repeated for every constituent.

It is important to notice that only the linear constraints provided by the expert can be used for these calculations. It was researched how it would be possible to automatically determine if a constraint was linear and to extract the necessary information from the constraints to create the matrices necessary for the LP procedure.

Using some knowledge of how the different constraints typically look like and by implementing some symbolic mathematical operations for the expression trees, a method was developed that manipulates the trees into a standard form that makes it very easy to decide if a constraint is linear or nonlinear. There may exist situations where this method will fail, but in these cases the method will mistake a linear equation for an nonlinear. In this situation the constraint will be excluded from the LP process, which is not as bad as trying to include a nonlinear constraints in the LP process.

The developed method works as follows. After examination of the differ-

ent constraints that will be processed by the method, it became clear that the constraints would only contain the basic arithmetic operators: addition, subtraction, multiplication, and division. Furthermore divisions only appear when dealing with conditional probabilities (as a result of Equation 2.1). If there are divisions present in the constraint, it can be manipulated in such way that all the divisions are multiplied out of the expression, only addition subtraction and multiplication operators will remain. The next step is to place all the multiplications as "deep" in the tree as possible. This is accomplished by "multiplying out", a simple example where variables are multiplied out is shown below:

$$x_1 * (x_2 + x_3) = x_1 * x_2 + x_1 * x_3 \ .$$

The multiplying out procedure is performed repeatedly on the expression tree until it is no longer possible to perform the procedure on the tree. Now, the tree has been been manipulated enough that it is possible to determine if it is linear, but some extra tree manipulation procedures have been added to simplify the expression even further, which has advantages when the constraint is being used to evaluate a sample. An example of further simplification is that terms that have identical variables, but have different multiplying constants are merged into a new term with a new multiplying constant. A simple example:

$$5 * x_1 + 4 * x_1 = 9 * x_1 \ .$$

To make the merging easier a few preprocessing steps are performed. First all the terms are collected. Three types of terms can be distinguished, a term can consist out of:

- one constant,
- one variable,
- or a combination of constants and variables.

Terms consisting out of one variable are (temporarily) transformed into the third type by adding a multiplication with one to the tree:

$$x_1 = 1 * x_1 \ .$$

This preprocessing step simplifies the merging process, since now the constraint will only have two different types of terms and less code is necessary for identifying if two constraints are eligible to be merged. To make it easier to compare two terms consisting out of multiple variables and constants they are changed into a standard form. The constants are multiplied together and the result is put in front of the term. The variables, which are

always in the form $x_1, x_2, \ldots, x_n$, are sorted so that they appear in the term in ascending order. An example would be:

$$x_4 * 3 * x_2 * 7 * 2 * x_1 * x_3 * 2 = 84 * x_1 * x_2 * x_3 * x_4 \ .$$

Deciding if two terms are eligible for merging has now become very easy. Constants can only merge with constants and terms with combinations of variables and constants can be merged if both have the same number of variables and the variables are identical. Since the variables are sorted it can be determined very quickly if two terms cannot be merged. If the two terms that are to be compared have different number of variables it can be concluded directly that they cannot be merged. When the terms have an equal number of variables, the variables in the terms are compared from left to right and when two variables are found that are unequal, it can be concluded that the whole terms are unequal and that they cannot be merged. Finally, after merging all eligible terms, all variables are moved to the left hand side of the expression and all the constants to the right hand side of the expression. Now, when the constraints have been manipulated into this form the linearity of the constraints can be checked by checking if there are any multiplications of variables with variables in the left hand side of the constraints. If these multiplications are found, a constraint is not linear and cannot be used for the LP procedure.

After all the manipulations, extracting the necessary information from the linear constraints to perform the LP procedure is easy. The coefficients of the variables can easily be found in the expression trees because the trees have been standardized. A term now consists out of a multiplication of a variable and a coefficient, the coefficient will always be on the left hand side of the multiplication node and can be easily extracted. All the coefficients of all the linear constraints are collected and put into a matrix to be used in the LP procedure.

Using the LP procedure does indeed decrease the size of the sample space and thus will improve the efficiency of the sampling process, but because the nonlinear constraints are not considered in this process, there should still be more to gain in sampling efficiency.

3.5 Derivation of the 2^{nd} Order Distributions

To derive the the 2^{nd} order distributions over the CPT entries, the hyper-space of joint probability distribution has to be sampled. After the LP preprocessing step described in Section 3.4 the size of the sample space has most likely been decreased, but still, especially if the number of dimensions increases, the sample space can be very large and finding samples that satisfy all constraints is still a very inefficient process. Two approaches have been proposed that try to generate samples as efficient as possible. The first

proposed approach, described in Section 3.5.1 focuses on using a projection algorithm to move samples towards the feasible region where all constraints are satisfied. The second proposed approach, descibed in Section 3.5.2 revolves around using a truncated Dirichlet distribution to generate samples. In Section 3.5.3 the approaches have been evaluated and an approach has been selected for implementation. Section 3.5.4 describes the necessary calculations after sampling has been completed.

3.5.1 Projection Approach

To improve the sampling efficiency several heuristics were created that should decrease the sample space and speed up the sampling process by trying to force the samples to satisfy the axioms of probability so that less samples, that are unvalid by default, have to be evaluated by the constraints. This approach uses a projection algorithm to iteratively move samples towards the feasible region, the region in the hyperspace where samples satisfy all constraints.

Order Heuristic

The first created heuristic changes the ordering of the constituents in such way that constituents with the smallest sample interval are sampled first. Using the global upper and lower bounds that have been calculated by the LP procedure, the difference between upper and lower bounds are calculated to get the width of the sample intervals of the constituents. The heuristic is very simple; it calculates the interval size for the constituents and then sorts the constituents accordingly. It is performed as a preprocessing step for the second heuristic, so that it performs better when it creates the importance function.

Importance Heuristic

The second created heuristic approaches sampling in a manner similar to importance sampling (Section 2.4). Since the shape of the constrained space (assuming that it exists) will differ for every set of constraints it is necessary to learn the shape of the importance function every time the method is performed. To learn the shape, an algorithm designed by Chinneck (2004) called the constraint consensus algorithm was used. This algorithm has the ability to find points in the sample space that are close to the feasible region, i.e. the region where all constraints are satisfied. It works by iteratively moving an arbitrary point towards the feasible region. The algorithm resembles gradient descent algorithms and Chinneck has classified the algorithm as "*a form of simultaneous component-averaging gradient-projection algorithm.*" The constraint consensus algorithm is described in Figure 3.3. For more information on the algorithm see (Chinneck, 2004).

Inputs:

- a set of constraints,
- an initial point $\mathbf{x}$,
- a feasibility distance tolerance α,
- a movement tolerance β.

1. NINF = 0; for all j: $n_j = 0$, $s_j = 0$.
2. For every constraint c_i:
    1. If c_i is violated (without considering a tolerance) then:
        1. Find the feasibility vector and the feasibility distance.
        2. If the feasibility distance is greater than α then:
            1. NINF = NINF + 1.
            2. For ever variable x_j in c_i:
            $n_j \leftarrow n_j + 1$; $s_j \leftarrow s_j + f_{ij}$.
3. If NINF = 0 then exit successfully.
4. For every variable x_j:
    1. $t_j = \frac{s_j}{n_j}$.
5. If $\|\mathbf{t}\| \leq \beta$ then exit unsuccessfully.
6. $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{t}$.
7. If necessary, reset $\mathbf{x}$ to respect any violated variable bounds.
8. goto Step 1.

Figure 3.3: Constraint Consensus Algorithm

To gather information about the shape of the joint distribution function the constraints describe, a number of samples was generated, and the constraint consensus algorithm was applied to each of the samples. The number of samples depends on the number of dimensions of the sample hyperspace. After applying the constraint consensus algorithm on the samples, they should now be close to the feasible region, if it exists. Now these samples can be used to create an importance function.

To keep the sampling process simple, uniform sampling was used to sample values for each of the constituents (dimensions). Uniform sampling has

only two simple parameters: the upper bound and the lower bound. Values between the two bounds are selected as sample with equal probability. The implemented importance function is based on a simple idea: currently there are only global bounds for every variable. These global bounds describe the absolute minimum and maximum values of the variables. But all variables are linked to each other by the constraints and the probabilistic axioms. Once one variable has been given a value, this will influence the values the other variables can take. A simple example is the probability axiom that states that the sum of the variables should be one. If a variable gets value a, then there is only $1 - a$ left for the other variables.

It can be assumed that the variables are conditionally dependent on each other. If the variables are ordered and sampled in this order, a Bayesian network can be created that contains all the variables (which represent the values of the constituents) and the links of the Bayesian network represent the dependencies between the variables. This Bayesian network represents a joint probability distribution and this J-PDF could represent the importance function for the problem.

A simpler representation was chosen for the importance function. It was assumed that there are only dependencies between constituents adjacent in the chosen constituent ordering. This is where the results of the first created heuristic are used. The constituent with the smallest sample interval is sampled first, followed by the constituents with larger intervals, ending with the constituent with the largest interval. The next step was to divide the sample interval for each constituent into ten subintervals of equal size. Using the Bayesian network analogy this would mean to create discrete nodes with ten different value possibilities and connecting the nodes with only one arrow. The structure of the BN would resemble a linked chain, node C_1 is connected to node C_2, node C_2 is connected to node C_3, etc until node C_{n-1} is connected to node C_n.

At this point the earlier collected samples that should be near the feasible reason come into the picture. Assuming that enough samples were generated and that the randomness of the sampling has caused the samples to be dispersed over the sample space to surround the feasible area, the samples can be used to generate bounds for a constituent conditioned on the value of the previous constituent in the sample order. For every constituent the collected samples are distributed over ten bins, each representing a subinterval of the constituent. After the samples are distributed over the bins, minimum and maximum values are calculated for each bin for the next constituent in the sample order. The minimum and the maximum value become the lower and the upper bound for the next constituent when the current constituent has a value that falls into the subinterval represented by this bin. The process is repeated for all but the last constituent, the last one has nothing to condition so the process is not necessary here. The heuristic is summarized in Figure 3.4.

1. Generate a number of samples, the number is dependent on the number of constituents.
2. Perform the constraint consensus algorithm on these samples.
3. For every constituent:
    1. Initialise the constituent's upper and lower bounds with the global upper and lower bounds calculated by the LP procedure for this constituent.
4. For every constituent except the last one:
    1. Put every sample in one of the bins by using a simple calculation to find the right bin index. The index i is found by using the global upper bound u, the global lower bound l, and the sample value of the current constituent, x as follows: $i = \left\lfloor \frac{x-l}{u-l} * 10 \right\rfloor$.
    2. Calculate for every bin the minimum and the maximum of the sample values from the next constituent.
    3. Assign the minimum and maximum values found in the bins as upper and lower bounds for the next constituent.

Figure 3.4: Heuristic based on importance sampling

Axiom Heuristic

The third created heuristic tries to reduce the number of samples that has to be discarded due to noncompliance to the axioms of probability. Very soon in the development of the sampling process, it became clear that the axiomatic constraint that states that all the constituents should sum to 1 causes a very large number of samples to be discarded. By using an implementation of the constraint consensus algorithm, especially optimized for using the axiomatic constraints, the heuristic moves the samples to the probability plain in the hyperspace. On this plain a sample satisfies the axioms of probability. The heuristic is run after all the constituents of the sample have been generated. Since only the axioms are used as constraints for the algorithm it was possible to remove the constraint dependent parts of the algorithm and optimize the algorithm for faster run times.

Sampling Process

All these heuristics are used to create a sampling process that can be used to generate samples that can be evaluated by the constraints. The whole process is summarized in Figure 3.5.

1. Perform LP using the available linear constraints, to tighten the probability bounds (Section 3.4).
2. Order the constituents in such way that they are sorted according to the size of the sampling interval (heuristic 1).
3. Create an importance function to be used for improved sampling (heuristic 2, Figure 3.4).
4. Create 1000 valid samples:
    1. Generate a sample:
        1. Generate a value for each of the constituents:
            1. For the first constituent, the global bounds are selected.
            2. A value is generated for the constituent, using uniform sampling and the selected bounds.
            3. using the value of the previous constituent, the correct set of bounds for the next constituent is chosen.
            4. Step 2 and 3 are repeated until all constituents have been assigned a value.
        2. After all constituents have been assigned a value, the constraint consensus algorithm is run with only the axioms of probability as constraints (heuristic 3). This heuristic ensures that the sample is on the probability plain in the hyperspace.
    2. The sample is now evaluated by all constraints. If one of the constraints evaluates the sample as false, it is discarded, otherwise it is saved.

Figure 3.5: The Sampling Process

3.5.2 Dirichlet Approach

The second sampling approach proposed in Section 3.5 uses a truncated Dirichlet distribution to generate samples for evaluation by the constraints. The Dirichlet distribution is a continuous, multivariate distribution. The distribution is a generalisation of the beta distribution for any number of dimensions. It has the following density function:

$$f\left(X;\alpha\right)=\frac{1}{\mathbf{B}\left(\alpha\right)}\prod_{i=1}^{N}x_{i}^{\alpha_{i}-1}\delta\left(1-\sum_{i=1}^{N}x_{i}\right), \tag{3.1}$$

where α is a parameter vector with N real elements ≥ 0, X is vector with elements that lie in the range $[0,1]$, $\delta\left(x\right)$ is the Dirac delta function, and $\mathbf{B}\left(\alpha\right)$ is the multidimensional beta function. The beta function is expressed by using the gamma function, an extension of the factorial function:

$$\mathbf{B}\left(\alpha\right)=\frac{\prod_{i=1}^{N}\Gamma\left(\alpha_{i}\right)}{\Gamma\left(\sum_{i=1}^{N}\alpha_{i}\right)}\,. \tag{3.2}$$

Samples generated by a Dirichlet distribution are distributed over a $(N-1)$-dimensional simplex. For a point X in N-dimensional space to lie on the (unit) simplex it must satisfy the following requirement:

$$\sum_{i=1}^{N}x_{i}=1\,. \tag{3.3}$$

This is represented by the density function of the Dirichlet distribution by using the Dirac function. Only points on the $(N-1)$-dimensional simplex get a probability density $f\left(X;\alpha\right)\neq 0$. The general beta function $\mathbf{B}\left(\alpha\right)$ is used as a normalisation factor to make the density function satisfy the axioms of probability, in this situation ensuring that

$$\int_{x_{1}=-\infty}^{\infty}\cdots\int_{x_{N}=-\infty}^{\infty}f\left(X;\alpha\right)dx_{1},\ldots,dx_{n}=1\,.$$

The Dirichlet distribution is an excellent choice to generate samples for the method. This can be understood by examining the distribution and equation 3.3 a little closer. The elements x_i of a sample vector X generated by a Dirichlet distribution must be nonnegative, i.e. $x_i \geq 0$, and all elements must sum to 1 (eq. 3.3). These are exactly the requirements for a sample to be a valid joint (discrete) probability distribution. Any sample from a Dirichlet distribution with any parameter vector α is automatically a sample that represents a valid joint distribution for the method. This means that no longer samples will need to be rejected because they do not satisfy the axioms of probability.

One interesting property of Dirichlet distributions is when the parameters α_i are all chosen to be 1, the resulting samples from the distribution will be distributed uniformly over the simplex. This parameter setting could be a starting point for when the sampling process is just started. Later when more information comes available the parameters of the distribution could be adapted to steer the sampling process towards the area in the simplex that yields more samples that satisfy all the constraints.

Sample Generation

To generate samples from a Dirichlet distribution the following algorithm can be used (Devroye, 1986):

Input:

- a parameter vector $\alpha = \{\alpha_1, \ldots, \alpha_N\}$,

1. Generate N independent samples $Y_1, \ldots, Y_N$ from Gamma(α_i) distributions with density

$$\frac{y_i^{\alpha_i - 1} e^{-y_i}}{\Gamma(\alpha_i)}$$

2. Generate the sample $X = \{x_1, \ldots, x_n\}$ for the Dirichlet(α) by using

$$x_i = \frac{y_i}{\sum_{i=1}^{N} y_i}$$

Figure 3.6: Algorithm for generating samples from a Dirichlet(α) distribution

The generation of samples from a Dirichlet distribution can be reduced to generation of samples from a Gamma Distribution. Many approaches and algorithms exist to generate samples from a Gamma(α) distribution. Some examples are described in (Press, Teukolsky, Vetterling, & Flannery, 1992) and (Devroye, 1986). Exact details are not important since eventually another approach was chosen to generate samples that did not rely on using samples from Gamma distributions.

Although the approach proposed above works, problems start to arise when trying to use the probability bounds provided by the linear programming preprocessing step. Using this information means that now the different entries of the sample vector from the Dirichlet distribution should only be able to be assigned values that lie between the bounds for these entries. It is not possible to directly start sampling between these bounds since the

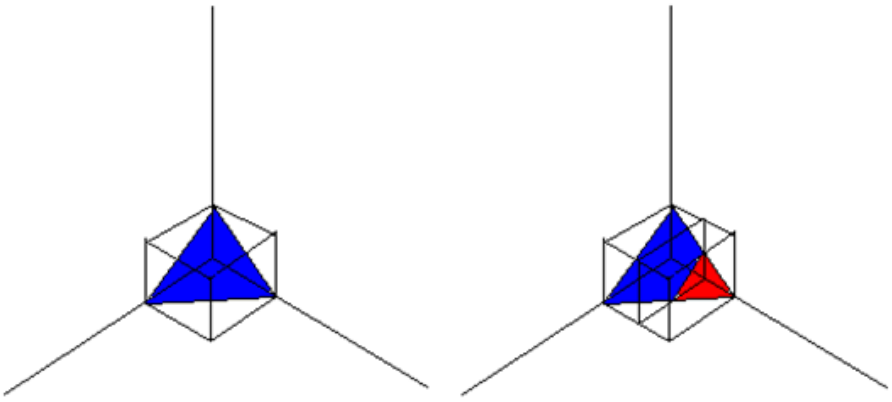
values for the different entries of the vector are calculated in two steps, beginning with samples from Gamma distributions which are then normalized by the sum of these samples to get the samples for the vector entries.

The simplest method of incorporating the probability bounds into the Dirichlet sampler was to use rejection sampling. If a sample does not lie in the volume described by the probability bounds, reject it. This is a workable solution if the bounds are not too tight. Once the volume that is constructed from the bounds becomes sufficiently small, a large number of the samples will be rejected and it will no longer be efficient to use rejection sampling.

Another approach that was tried was to view the problem from a geometrical perspective. By looking at what the probability bounds do to the shape of the simplex the idea was that it might be possible to derive a transformation, that would make it possible to sample from that part of the simplex that intersects the volume of the probability bounds without resorting to rejection sampling.

Geometric Approach

This approach was only partially successful. There are two different types of probability bounds, lower bounds and upper bounds. The result of a probability bound on a simplex is shown in Figure 3.7.



(a) 2-Simplex (b) 2-Simplex with a probability bound

Figure 3.7: Simplices in a three dimensional hypercube

The figure shows a very simple example of what a (unit) 2-simplex looks like in 3 dimensional space. In this situation the simplex has the shape of a triangle (Figure 3.7(a)). It is called a 2-simplex because the simplex itself has a 2-dimensional surface. If a probability bound is defined for one the axes, this will cut the hypercube (defined by the axes x_i, $0 \leq x_i \leq 1$) into two parts. Depending on the type of the probability bound the remaining simplex will be the blue part, if it is an upper bound, or the red part if it is a lower bound. If examined closer, it is clear that the red area of the simplex bounded by the specified lower bound and the standard upper bound actually is another simplex that has the same shape as the initial unit simplex. This fact can be used to create samples for the sub simplex without using rejection sampling. The principle is very simple:

1. Generate a sample for the unit simplex.
2. Scale the simplex (and the sample in it) to the size of the sub simplex.
3. move (translate) the simplex to the position of the sub simplex.

This idea can be generalized to N dimensions and N lower bounds. It can be done in the following way: Assume that S is a $(N-1)$-simplex, X is a point in N dimensional space, A is a N dimensional vector containing the lower bounds, B a N dimensional vector containing the upper bounds, and V is the volume that is bounded by vectors A and B. For a standard unit simplex in a unit hypercube the values for the entries of A and B will be:

$$\begin{aligned} &\forall i \, (a_i = 0) \, , \\ &\forall i \, (b_i = 1) \, . \end{aligned}$$

The simplex has vertices:

$$\begin{aligned} s_1 &= (1, 0, 0, \ldots, 0)^T \\ s_2 &= (0, 1, 0, \ldots, 0)^T \\ &\vdots \\ s_n &= (0, 0, 0, \ldots, 1)^T \ . \end{aligned}$$

A point X lies on the unit simplex if the following conditions are met:

$$\begin{aligned} &\forall i \, (x_i \geq 0) \, , \\ &\sum_{i=1}^{N} x_i = 1 \, . \end{aligned}$$

If one of the lower bounds a_i in A becomes nonzero this will have a direct effect on all the upper bounds in B. For a point X to be on the simplex, it must meet the above specified conditions. This means that if one of the

entries x_j has a minimum bound $0 < a_j \leq 1$, there is only $1 - a_j$ left for the rest of the entries of X. So, assuming all entries of A except a_j are 0, this will create a new upper bound vector B' with entries:

$$\forall i \left(b'_i = 1 - a_j \right) \ .$$

The vector B' can be created by multiplying vector B with scalar $1 - a_j$:

$$B' = (1 - a_j)\, B \ . \tag{3.4}$$

The intersection of the new volume V', created by A and B', and the simplex S results in a sub simplex S'. Now, a point X lies on S' when the following conditions are met:

$$\forall i \left(a_i \leq x_i \leq b_i \right), \\ \sum_{i=1}^{N} x_i = 1 \ .$$

S' is not a standard simplex, its vertices do not resemble these of the standard simplex. The resulting vertices for S' are:

$$\begin{aligned} s'_1 &= \left((1 - a_j), 0, 0, \ldots, 0 \right)^T \\ s'_2 &= \left(0, (1 - a_j), 0, \ldots, 0 \right)^T \\ &\vdots \\ s'_j &= \left(0, 0, 0, \ldots, 1, \ldots, 0 \right)^T \\ &\vdots \\ s'_n &= \left(0, 0, 0, \ldots, (1 - a_j) \right)^T \ . \end{aligned}$$

With a very simple operation S' can be changed into a scaled version of S. All the vertices of S', except s'_j, are already scaled versions of the vertices of S; They are scaled by a factor of $1 - a_j$. Vector s'_j can be changed into a scaled version of the standard vector s_j, by subtracting a_j from element j of the vector. In this particular situation this would be equivalent to subtracting vector A from vertex s_j. Since S' is now a scaled version of S, and S' can be changed into S by dividing all the vertices of S' by $1 - a_j$, it is possible to define a transformation scheme to translate a point X on simplex S to a point X' on simplex S'. This transformation scheme would work as following:

1. Multiply a point X on simplex S with the scale factor $1 - a_j$.
2. Add the value a_j to the entry x_j of the scaled vector X.

Now X has become a point X' on the simplex S'.

It is easy to extend this schema to allow for multiple lower bounds. Adding another lower bound simply means creating a sub simplex in S'.

This happens the same way as S' was created in S, thus the general transformation scheme will be similar to the one used for one lower bound. The general transformation scheme to transform a point X on S to a point X' on a simplex S' bounded by N lower bounds is:

1. Multiply a point X on simplex S with the scale factor $1 - \sum_{i=1}^{N} a_i$.
2. Add the vector A to the scaled vector X.

The differences between the two transformation schemes are easy to explain. The difference in the scaling factor comes from the fact that for every entry x_i that must have at least the value a_i, there is less left to assign freely to the entries X. Since the entries must sum to 1, the amount that can freely be assigned to the all entries will decrease by the size of every lower bound that is added. If a lower bound $a_j = 0.2$ is added there is only 0.8 left to assign freely. If another bound $a_k = 0.35$ is added, there is now only 0.45 left to be assigned to the entries. The assigning of this smaller amount to entries happens in the scaled simplex. If one entries gets the complete remaining amount, the others must get 0 and then the generated point will be on one of the vertices of the simplex. In the example from above, this would mean that the entry would get assigned 0.45 on top of its lower bound. Since this 0.45 is the result of subtracting the lower bounds from 1, the simplex must be scaled with factor $1-\sum_{i=1}^{N} a_i$ to ensure that the vertices have a maximum value of 0.45.

After scaling, the simplex S' must be placed in the right position of S to represent the sub simplex created by the lower bounds. This can be accomplished by adding the values of the lower bounds to the values of the point X' that is on the scaled simplex. This moves the simplex to the correct position. Since selecting a point on the scaled simplex represents dividing the amount that is left after subtracting the values of the lower bounds from 1, it is logical that after this amount is divided over the entries, the values of the lower bounds of the entries are added to the entries. Adding the bounds to the entry values creates entries that together form a point that lies on both simplices S' and S. The point will satisfy all the conditions for the unit simplex, but will only appear in the volume described by simplex S'. Since S' is created by applying the lower bounds to S, it is proven that sampling on a simplex constrained by lower bounds is possible without rejection sampling.

Upper Bounds

Using a geometric approach it is possible to, using a simple transformation scheme, sample in a part of a simplex that is constrained by lower bounds. Upper bounds on the other hand are a lot more difficult to deal with. If Figure 3.7(b) is examined again, but with the bound now representing an upper

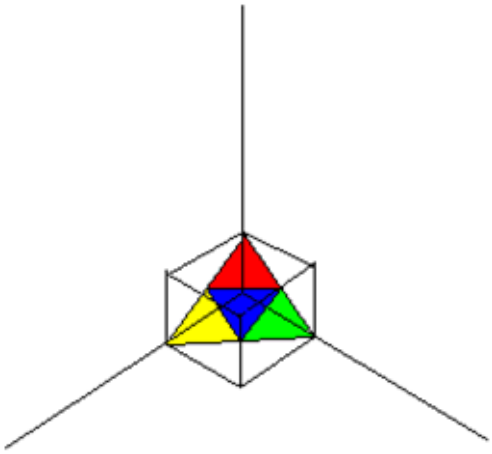


Figure 3.8: Subdivision of a 2-Simplex

bound, the area that needs to be sampled now is the blue area. This area is a truncated simplex and the unit simplex cannot be easily transformed into this shape. There does not exist an easy transformation scheme like the one found for the part of the simplex constrained by the lower bounds.

One idea of dealing with this problem was to subdivide the simplex into smaller sub simplices and then to only sample in the sub simplices that would fall in, or would overlap the area of the simplex bounded by the upper bounds. The effect of subdividing a simplex is shown in Figure 3.8: By subdividing the simplex into smaller simplices it would again be possible to use a simple transformation scheme to sample in the area constrained by lower and upper bounds. An extra difficulty would be the selection of the sub simplices. If it would be implemented poorly, it could influence the results of the whole sampling process.

After searching in the literature a method was found that could be used to subdivide a simplex into sub simplices of the same shape and volume (Edelsbrunner & Grayson, 1999). The authors use a so called abacus model, which is an algebraic interpretation of simplices. The advantage of using an algebraic interpretation for simplices is that when simplices in high dimensional spaces are considered, it is no longer feasible to work with geometric interpretations. The method allows for a d-simplex to be subdivided into k^d d-simplices, where k is an integer $k \geq 1$ that specifies the number of pieces, of equal width, each dimension of the simplex is subdivided in.

This method will subdivide the whole simplex into sub simplices, but most of these simplices will be unnecessary because of the defined upper bounds. Sub simplices that completely fall outside of the area of the simplex do not need to be sampled since these samples will never be valid. The paper describes how the coordinates of the vertices of the sub simplices can be calculated, using this information it is possible, by using the information of

the upper bounds, to only select simplices that will fall into, or overlap the bounded area. In total, at most $k^d - (k-1)^d$ sub simplices will be needed to fill the volume created by the upper bounds. To accomplish this the value of k most be chosen according to the smallest upper bound $b_i \geq 0$:

$$k = \left\lceil \frac{1}{min\left(b_i\right)} \right\rceil . \tag{3.5}$$

The dimensions will be cut into k pieces, but for the dimension with the smallest upper bound, only 1 of the k pieces will be used. For the other dimensions the same observations can be made and a vector K can be created containing the maximum indices to be used when generating simplices. This would further reduce the total number of necessary subsimplices.

Still, when the number of dimensions starts to increase and when the upper bounds become smaller, the number of subsimplices will become extremely large and eventually this solution will become intractable. It will take so many samples to have at least one sample in every sub simplex, that the advantage of using sampling in the area described by the subsimplices instead of rejection sampling has disappeared.

The geometrical approach was abandoned and the search was continued to find another approach for sampling on a simplex between lower and upper probability bounds. After some search a solution was found that would solve the problem completely. A paper was found that described truncated Dirichlet distributions (Fang, Geng, & Tian, 2000).

Truncated Dirichlet Distributions

A truncated Dirichlet distribution (TDD) is a variant of the Dirichlet distribution where the entries of the sample vector can be constrained by lower and upper bounds. A vector X generated from a TDD, with

$$X = \left(x_1, \ldots, x_N\right)^T ,$$

and

$$x_{-N} = 1 - \sum_{i=1}^{N-1} x_i,$$

has a density function for vector $X\left(x_1, \ldots, x_{N-1}\right)^T$ (Fang et al., 2000):

$$c^{-1} \prod_{i=1}^{n-1} x_i^{\gamma_i - 1} \left(1 - \sum_{i=1}^{n-1} x_i\right)^{\gamma_N - 1} , X_{-N} \in V_{N-1}\left(A, B\right), \tag{3.6}$$

where c is the normalizing constant, and $V_{N-1}\left(A, B\right)$ is the volume created by lower bound vector $A = \left(a_1, \ldots, a_N\right)$ and upper bound vector

$B = (b_1, \ldots, b_N)$:

$$V_{N-1}(A,B) = \left\{ X : \begin{array}{c} 0 \le a_i \le x_i \le b_i \le 1, \\ i = 1, \ldots, N-1, \\ a_N \le 1 - \sum_{i=1}^{N-1} x_i \le b_N \end{array} \right\}. \tag{3.7}$$

When all entries of vector A are 0 and the entries of vector B are 1, the TDD reduces to a standard Dirichlet distribution. Again, as with a standard Dirichlet distribution, when all the parameters γ_i are chosen to be 1 the distribution reduces to a uniform distribution. This time the samples are uniformly distributed over a convex polyhedron

$$T_N(A,B) = \left\{ X : 0 \le a_i \le x_i \le b_i, i = 1, \ldots, N, \sum_{i=1}^{N} x_i = 1 \right\}. \tag{3.8}$$

To generate samples from a TDD (Fang et al., 2000) used the conditional distribution method (Devroye, 1986). The idea of this method is that any joint distribution can be broken up into a product of marginal, conditional distributions (as discussed in Section 2.3.1). To generate a vector X from a $TD(A, B; \gamma_1, \ldots, \gamma_N)$ distribution, the joint density function of the vector X_{-N} is broken up into

$$\begin{array}{rl} f(x_1, \ldots, x_{N-1}) = & f(x_{N-1}) * f(x_{N-2} | x_{N-1}) * \cdots \\ & \cdots * f(x_1 | x_2, x_3, \ldots, x_{N-1}). \end{array} \tag{3.9}$$

Samples from the TDD are generated by sequentially generating samples from the marginal distributions, which according to (Fang et al., 2000) are truncated beta distributions, $Tbeta\left(\gamma_k, \sum_{i=1}^{N} \gamma_i - \sum_{i=k}^{N-1} \gamma_i; \xi_k, \eta_k\right)$. $Tbeta$ is a truncated version of the standard $Beta(\alpha, \beta)$ distribution, where ξ is the lower bound and η is the upper bound. For every entry of the TDD vector new values for ξ_k and η_k must be calculated, this is done by using

$$\xi_k = max\left(\frac{a_k}{1 - \sum_{j=k+1}^{N-1} x_j}, 1 - \frac{\sum_{i=1}^{N} b_i - \sum_{j=k}^{N-1} b_j}{1 - \sum_{j=k+1}^{N-1} x_j} \right), \tag{3.10}$$

$$\eta_k = min\left(\frac{b_k}{1 - \sum_{j=k+1}^{N-1} x_j}, 1 - \frac{\sum_{i=1}^{N} a_i - \sum_{j=k}^{N-1} a_j}{1 - \sum_{j=k+1}^{N-1} x_j} \right). \tag{3.11}$$

Where $\sum_{j=k+1}^{N-1} x_j$ is defined as 0 when $k + 1 > N - 1$.

An algorithm can be created for generating samples from a TDD by sequentially generating samples from a truncated beta distribution with the appropriate parameters. This algorithm has been described in Figure 3.9. Here $F_k^{-1}(\cdot)$ is the inverse of the cumulative distribution function of the $Tbeta\left(\gamma_k, \sum_{i=1}^{N} \gamma_i - \sum_{i=k}^{N-1} \gamma_i; \xi_k, \eta_k\right)$ distribution.

Input:

- a parameter vector $\gamma = (\gamma_1, \ldots, \gamma_N)^T$,

1. For $k = N-1, k > 0, k = k-1$
    1. Calculate the values for ξ_k and η_k using eq. 3.10 and 3.11.
    2. Generate a random value u_k from a uniform $U[0,1]$ distribution.
    3. Calculate $x_k = \left(1 - \sum_{j=k+1}^{N-1} x_j\right) F_k^{-1}(u_k)$
2. Calculate $x_N = 1 - \sum_{i=1}^{N-1} x_i$
3. Output vector $X = (x_1, \ldots, x_N)^T \sim TD(A, B; \gamma_1, \ldots, \gamma_N)$

Figure 3.9: Algorithm for generating samples from a truncated Dirichlet distribution using the method described by Fang et al. (2000)

To calculate $F_k^{-1}(x)$, the cumulative distribution function $F_\beta(x)$ and the inverse cumulative distribution function $F_\beta^{-1}(x)$ of the standard $Beta(\alpha, \beta)$ distribution are used.

$$F_k^{-1}(x) = F_\beta^{-1}(F_\beta(\xi_k) + x * (F_\beta(\eta_k) - F_\beta(\xi_k))) \ . \tag{3.12}$$

The algorithm described in Figure 3.9 allows for the sampling between upper and lower bounds on a simplex. This is accomplished without the use of rejection sampling. It is important to note that for the calculation of $F_\beta(x)$ and $F_\beta^{-1}(x)$ numerical approximations have been used. To evaluate these functions the incomplete beta function and its inverse must be calculated. The incomplete beta function is defined as:

$$B_x(a, b) = \int_0^x t^{a-1}(1-t)^{b-1}\, dt \ . \tag{3.13}$$

To calculate the inverse of the incomplete beta function, an iterative technique is used to find the root of

$$B_x(a, b) - y = 0, \tag{3.14}$$

where y is the input value of the inverse incomplete beta function. The function returns the value x that equates Equation $B_x(a, b)$ to y.

A C++ software library has been found that implemented $B_x(a, b)$ and its inverse. This library, Cephes (Moshier, 1989), has been used to implement the TDD sample generator.

Importance Sampling Possibilities

Dirichlet distributions have a parameter vector α that can be used to control the behavior of the distribution. The Dirichlet distribution is a multidimensional generalisation of the beta distribution and behaves in a similar manner. When the values for the vector α all are $\alpha_i = 1$, the samples generated by the distribution will be uniformly distributed over the simplex. When all α's are larger than 1, the distribution over the simplex resembles a Gaussian distribution. The simplex will have a Gaussian shaped area where it will be more likely for samples to appear. The coordinates for this area depends on the α vector. This allows control over where the Gaussian $(N-1)$-dimensional "bell" curve will appear on the simplex. This is shown graphically in Figure 3.10. This property of the Dirichlet distribution can

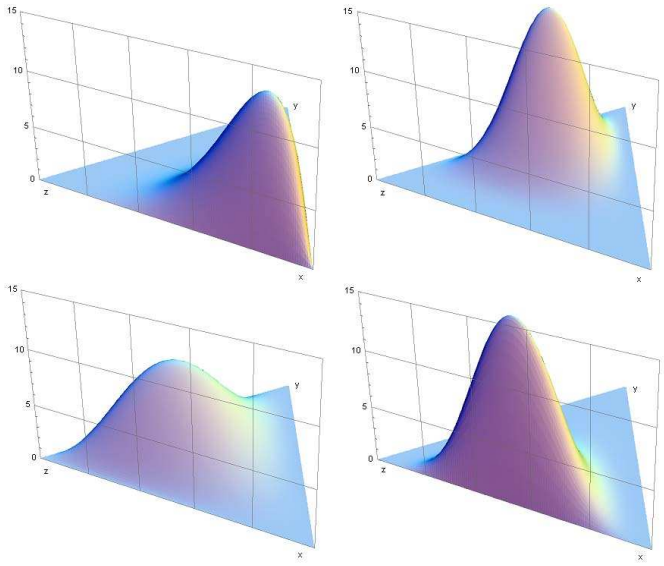


Figure 3.10: Several images of probability densities of the Dirichlet distribution as functions on the 2-simplex. Clockwise from top left: $\alpha = (6,2,2), (3,7,5), (6,2,6), (2,3,4)$. The z-axis in the figure shows the value of the density functions in relation to the (x, y, z) coordinate on the 2-simplex.

be used for an importance sampling scheme (Section 2.4) for the sampling process. The Dirichlet distribution can be used as the importance function used to approximate the probability distribution of points in the hyperspace that satisfy all constraints. The probability density function for these points can be written as:

$$g(X) = \begin{cases} c & \forall i\,(s_i = true) \\ 0 & otherwise \end{cases}, \tag{3.15}$$

where c is a constant and s_i are the constraints provided by the expert. According to the requirements stated by (Cheng & Druzdzel, 2000), the Dirichlet distribution can be used as an importance function. These requirements are:

- There must exist an algorithm for generating samples from importance function $f(X)$.
- Importance function may only be 0 when the original function is 0:

$$f(X) = 0 \Rightarrow g(X) = 0 \, .$$

This simply means that the importance function must be able to generate samples everywhere where the original distribution can generate samples. Since samples generated from a Dirichlet distribution can represent valid probability distributions, a property that is also inherent to the region described by the expert constraints, it is certain that any possible sample that would satisfy all constraints can be generated by a Dirichlet distribution.

A possible importance sampling scheme has been explored, but not implemented. It has been decided to put this under future work. The scheme has been based on the approach presented by Cheng and Druzdzel (2000). The idea is to start with a Dirichlet distribution with parameters

$$\alpha_i = 1, i = 1, \ldots, N \, ,$$

and after a number of samples has been found that satisfies all the constraints to iteratively adapt the parameters so the Dirichlet distribution will increasingly start to approximate the real distribution. To accomplish this an algorithm is proposed that estimates a parameter vector for the Dirichlet distribution using the acquired samples that satisfy all, or perhaps if this leads to better results, a very large portion of the constraints. The algorithm (Figure 3.11), will have to be refined further before it will be ready for implementation in the sampling process.

The literature was searched to find methods for estimating the parameters of a Dirichlet distribution. An overview paper was found that compared 4 different methods for maximum likelihood estimation of the Dirichlet parameters (Huang, 2005). The 4 methods that were compared are:

1. Gradient Ascent.
2. Fixed Point iteration.
3. Newton-Raphson Method.
4. Separately estimating Mean and Precision.

1. Initialise the α vector with values $\alpha_i = 1$;
2. Initialise the upper and lower bound vectors A and B with the values from the linear programming process.
3. While 1000 valid samples have not been generated
    1. Generate a sample X from distribution $TD(A, B, \alpha)$
    2. If the sample satisfies all the constraints keep it, else discard it.
    3. After d valid samples have been generated, update the parameter vector
        1. Save the current vector α for later use.
        2. Using the gathered valid samples, calculate new values for the entries of α by using a Maximum Likelihood Estimation procedure.
4. Assign the samples appropriate weights using the likelihood function, the saved α vectors, and perhaps other factors.

Figure 3.11: Proposed algorithm for adaptive importance sampling by changing the Dirichlet parameter vector

Further research will be necessary to determine if one of the compared methods should be used for the estimation process, or that the search would have to be continued to find other methods. Important criteria will be speed of convergence, and precision.

Using the importance sampling algorithm will result in differences in likelihood between samples. This results from generating samples from Dirichlet distributions using different α's, and samples generated from a $Dir(\alpha)$ distribution can have different likelihood values. The generated samples will have to be weighted to be able to incorporate the differences in likelihood between samples into the calculation process of the CPT entries.

Currently no importance sampling algorithm has been implemented and all the entries of parameter vector α will, during the complete sampling process, have the value 1. Thus the area of the simplex, truncated by the lower and upper bounds, will be sampled uniformly.

3.5.3 Evaluation

The 2 discussed approaches have been compared with each other, criteria for selecting an approach were the ability to generate samples uniformly over a simplex and the easy of extending the approach to allow for importance sampling. Section 6.1.2 describes an experiment that compares 7 different sampling approaches, including the Dirichlet and the projection approaches, using the criteria mentioned above. The result of the experiment was that the Dirichlet approach was chosen to be implemented as the sampling process. The approach has a strong mathematical background, which allows for controllable behavior during sampling. Extending the approach to use importance sampling techniques can also be done with relatively ease, when compared to the projection approach, in a mathematically correct manner.

The problem with the projection approach is that it generates samples in a very simplistic way and then relies on the constraint consensus algorithm to move a sample point from a random point in the hyperspace to the surface of the simplex. Since it is possible that different samples may be projected on the same point on the simplex, there is no simple definition of a density function for the sampling process using the projection approach. This makes it a lot more difficult to implement an importance sampling scheme. A heuristic has been created that tries to work in a similar way, but test results during implementation of the heuristic showed that when the number of dimensions started to increase, Performance would eventually drop to a level where no real benefit was gained by running the heuristic.

Generally, using the projection approach will most likely not lead to very good results, but perhaps some heuristics of the approach can be used to improve the Dirichlet approach. It might be useful to investigate if ordering the constituents in a certain way improves the Dirichlet sample generator. Ordering criteria could be related to the lower or the upper bounds, or a function of the two. The proposed importance sampling algorithm can also possibly be improved by using parts of the constraint consensus algorithm. A part of this algorithm calculates feasibility vectors and feasibility distance scalars that give an estimation of how far a sample lies from the feasible area where samples satisfy all constraints. An idea could be to store rejected samples with a feasibility distance lower than a specific value and use these samples in addition to the samples that satisfy all the constraints when estimating the α vector. Another criterium to add rejected samples to this calculation could be to look at the ratio of constraints that have accepted or rejected the sample. With a sufficiently high acception ratio the sample could be used as data for the parameter estimation. Further investigation will be neccesary to determine if these suggestions will in fact improve the Dirichlet approach.

3.5.4 Post-Sampling Processing

After the sampling process has finished and 1000 valid samples have been collected, all information is now present to calculate all the necessary histograms for all the nodes in a FN. To calculate a histogram for a CPT entry the set of samples is transformed into a new set of samples that contains samples of the CPT entry. The transformation process here is equivalent to performing inference on a J-PDF (see Section 2.3.3, Equation 2.16). The basic general inference procedure is used to perform the calculations. It is performed for every sample, so the 1000 samples of probability distributions of the FN are transformed into 1000 samples of the desired CPT entry. This process is repeated for all the CPT entries of all the nodes. Once all entries are processed the histograms for the entries can be generated at any time. The histograms represent the estimates of the 2^{nd} order distributions over the CPT entries. The generation of histograms is not implemented in the method itself. The raw data, sets of samples for all the CPT entries, is exported to GeNIe where there already is code implemented to generate histograms.

3.6 Conflict Detection

One of the goals set in Chapter 1 was to design a method for the detection and resolution of conflict between constraints. All the constraints have to be satisfied for a sample to be valid, any sample that does not satisfy all constraints must be discarded. There is one big problem with this approach: conflicting constraints will prohibit any sample to be valid. A set of constraints is conflicting with each other when it is impossible for all the constraints to be satisfied at the same time. Detecting conflicts is hard, one can never be sure that the absence of valid samples is because of conflicting constraints or that the sampling procedure just has not hit inside the feasible area. Pin-pointing the conflicting constraints is even harder, and there does not yet exist a good procedure for finding conflicting constraints for the nonlinear, non convex case, which is the worst case situation encountered when applying the method.

3.6.1 Method Overview

For the case that all the constraints are linear, there are good algorithms that can find Irreducible Infeasible Subsystems (IIS). An IIS is a subset of the complete set of constraints that contains a conflict. If one of the constraints is removed from the subset, the subset becomes feasible.

Some proposed algorithms for finding IIS are (Chinneck, 1996):

- Deletion filter (Chinneck & Dravnieks, 1991).
- Elastic filter (Chinneck & Dravnieks, 1991).
- The simplex algorithm (Loon, 1981).
- Algorithm based on isolating IIS set covers (Gleeson & Ryan, 1990).

Finding IIS is only very effective when there are only linear constraints, once there are also nonlinear constraints, the algorithms do not perform very well anymore. Some special cases may exist where the constraints satisfy certain conditions, like the nonlinear constraints are quadratic, that using IIS algorithms still may lead to reasonable results. In other cases checking feasibility is very difficult. In general, solving nonlinear programs is difficult and researchers normally will specialise on a certain type of nonlinear programs. Some examples are:

- Linear constraint optimization. Here all the constraints are linear.
- Quadratic programming. In this case the objective function is at most quadratic.
- Unconstrained optimization. Here there are no constraints.
- General NLPs with nonlinear constraints. In the worst case scenario here both the objective function and the constraints are nonlinear.

To be able to give an indication of the feasibility of nonlinear programs, information needs to gathered. If an indication of the shape of the constraints can be given, this can simplify the process of determining feasibility. Normally one would like to check if the shape of a constraint is convex or not. If the complete set of constraints is convex, then deciding feasibility is much easier, but according to Pardalos (1994) "there is no known computable procedure to decide convexity." This is problematic, but reasonably good results can be obtained by using sampling (Chinneck, 2002). Chinneck has defined a measure called *constraint effectiveness*, this measure is defined as "the fraction of the sample points that violate the inequality." The higher the effectiveness of the constraints, the more sample points it has rejected during the sampling process. The highest possible value for constraint effectiveness is 1.0, this means that the constraint has rejected all sample points. If a constraints has a constraint effectiveness of 1.0 this could mean that the constraint by itself is causing infeasibility, although there still exists the possibility that a sample that will satisfy the constraints has not been generated yet. There are some differences for calculating constraint effectiveness for equality and inequality constraints. This is necessary because otherwise

equality constraints will almost always have a constraint effectiveness of 1.0, because it is very unlikely that samples will satisfy the constraint. Constraint effectiveness can be used to draw some conclusions about feasibility. If there is at least one constraint with a constraint effectiveness of 1.0, it is likely that the system is infeasible. Still there is the possibility that the feasible region has not been sampled yet. In the case that all constraints have a effectiveness less than 1.0 no real conclusion can be given on the feasibility of the total set of constraints.

The feasibility problem for the nonlinear case is very difficult and perhaps unsolvable deterministically. The absence of a feasibility check can cause the method to run indefinitely, because it is not known if the set of constraints is infeasible or that the feasible region has not been found yet. To be sure that the method will eventually terminate, successfully or not, a maximum number of samples has been defined. However, once a valid sample has been found, it is known that the set of constraints is feasible and the maximum number of samples can be removed. Eventually, the desired number of valid samples will be found by the method. If the set of constraints is infeasible, no valid samples will be found and once the total number of samples reaches the maximum value, the method will terminate. After unsuccessful termination of the sampling process, the method will heuristically try to suggest changes to the set of constraints supplied by the expert so the method can be run again with perhaps a better result.

3.6.2 Heuristics

Two heuristics are being used to aid the the expert when deciding which constraint(s) need to be changed or removed. One heuristic (majority heuristic) was devised by the author and the other, the constraint effectiveness heuristic, was created by Chinneck (2002). The majority heuristic is based on the idea that when the majority of the constraints evaluates a sample as true that there might be something wrong with the minority of constraints that has evaluated the sample as false. In this situation 1 is added to the minority counter for each constraint that belonged to the minority. In the situation that the majority evaluated the sample as false, the minority counters are left unchanged. The reason for this is that there are no conflicting constraints only when all constraints have simultaneously evaluated a sample as true at least once, when all constraints have simultaneously evaluated a sample as false there is no guarantee that the set of constraints does not contain any conflicting constraints. Thus when a constraint evaluates a sample as true, this can be considered as stronger evidence then when it evaluates the sample as false. After all samples have been processed by the heuristic, some steps are performed to normalize the results so that each statement provided by the expert gets a heuristic value between 0 and 1. The closer the value is to 1.0 the "worse" the constraint is. The heuristic is described

in Figure 3.12:

1. Re-evaluate all saved, failed samples and save the results of the evaluations.
2. For every sample:
    1. Calculate the ratio of true evaluations versus false evaluations.
    2. If the ratio is larger than 0.5 then add 1 the minority counters of the constraints in the minority and add 1 to the counter for the summing variable.
3. If an statement from an expert consists out of multiple constraints, add the counters of the constraints together to get the value for the statement.
4. Divide the minority counter values of all the statements by the value of the summing variable to get values between 0 and 1.

Figure 3.12: Majority Heuristic

The majority heuristic will most likely work better if the set of constraints is larger. If the number of constraints is very small it might become less frequent that a majority of constraints evaluates samples as true. In this situation the minority counters of the constraint will be hardly updated or even not at all. When this occurs the results from the heuristic may not be very useful and it might be better that Chinneck's constraint effectiveness heuristic is used for making decisions. The constraint effectiveness heuristic, as implemented in the program, is described in Figure 3.13.

In both the heuristics high values are "bad", and when the results are presented to the user, the constraints are sorted so that the highest ranking ones are on the top of the list. A high value for the constraint effectiveness heuristic means that a constraints has rejected a large number of the samples and that it might be to constraining. A high value for the majority heuristic means that a constraint or constraints from an expert statement have been minority "voters" for a large number of the samples and that it may need to be changed or removed. Using the heuristics is only a attempt to offer the expert the possibility to refine his or her probability statements so that revised statements may generate constraints that have feasible region. The heuristics are not foolproof, but they are the only available option at the moment.

1. Re-evaluate all saved, failed samples and save the results of the evaluations.
2. For every sample:
    1. If a constraint has evaluated the sample as false then add 1 to the constraint effectiveness counter of the constraint.
3. If an expert statement has multiple constraints, take the maximum value among the constraints as the value for the statement.
4. Divide the values of the constraint effectiveness counters of the statements by the number of samples to get a value between 0 and 1.

Figure 3.13: Constraint Effectiveness Heuristic

3.7 Merging Family Networks

After the method is performed on each of the family networks, the results must be combined into the original Bayesian network. One of the goals set in Chapter 1 was to find a method that was able to merge family networks and their results back into the original Bayesian network.

To give the expert as much influence on the probabilities for the CPT entries as possible, the histogram for each CPT entry is to be shown to the expert. These histograms are calculated using the collected constituent samples. For most nodes this will be relatively easy, but some nodes are in multiple family networks, and therefore multiple batches of samples and histograms, calculated from these samples, can exist for the same CPT entry. The different sets of samples and histograms for the CPT entries in overlapping nodes must be merged into one set of samples and one histogram for each CPT entry.

Merging the family networks and the samples generated for the nodes in the different family networks is not an easy process. The sample process has been performed on each of the family networks independently and possibly the results for the CPT entries may vary per FN. Since different FNs contain different variables it is likely that different FNs do not have the same set of constraints. The difference in constraints has influenced the sampling process and through the collected samples, the shapes of the histograms for the different CPT entries of the nodes. Another problem that contributes to the difficulty of merging FNs is that when the BN is decomposed in the FNs,

some nodes will be a child node in one FN and a parent in possibly multiple other FNs. When this happens the node will have different CPTs for the different FNs. When the node is a child, its values will be conditioned on all its parents. When the node is a parent, it may not have any parents itself in this FN and in this case it will have a prior distribution and not a conditional one. This will result in probability tables with different sizes. A binary node without parents only has 2 entries, but in another FN the same node may be a child with 2 parents and have a CPT with 8 entries. Somehow, when merging the FNs, the different sample sets must be merged together to get the sample sets and the histograms for the CPT entries for the node in the original BN. After merging the FNs into the BN, the size of the CPT for every node is equal to the largest CPT size of the node among the FNs. For nodes that have parents this will be the FN where they are the child of the FN. Nodes that do not have any parents, will have the same size CPT in any FN they appear in. The CPT size for these nodes will also not change after merging the FNs into the BN.

A possible method was devised for the merging of the samples of the nodes by creating a weighted average of the samples for a CPT entry. From this weighted average of samples the final histogram for a CPT entry would be created. Samples of a CPT entry would be put together into a new "bin". The weighting would be accomplished by adding samples multiple times to the bin. The more times the sample would be added, the heavier it would count for the end result. The number of times a sample would be added was to depend on the number of parents the node had in the FN the sample originated from. If a node had no parents the sample would only be added once, if a node had x parents, it would be added $x + 1$ times. The samples of nodes with less or no parents would be added to the bins of multiple CPT entries to be sure that they have still approximately the same effect. In the case that the node would have any parents, but less than in another FN, then samples would be added to bins where the parents match as much as possible. After closer inspection the proposed method was not mathematically sound.

The merging method must be mathematical sound to be sure that after merging the samples from the different FNs, the resulting histograms for the node's CPT entries still represent good probability distributions for the values of the CPT entries. Theoretically, the proposed merging process runs in the wrong direction. Samples from a node in parent form are merged with samples of the node in child form. Figure 3.14 shows the merging of two FNs into the original BN.

The CPTs are omitted from the figure, but they contain the following probabilities:

- FN A:
  - The node X has two: $P(X)$ and $P(\bar{X})$.
  - The node Y has four: $P(Y \mid X)$, $P(\bar{Y} \mid X)$, $P(Y \mid \bar{X})$, and $P(\bar{Y} \mid \bar{X})$.
- FN B:
  - The node Y has two: $P(Y)$ and $P(\bar{Y})$.
  - The node Z has four: $P(Z \mid Y)$, $P(\bar{Z} \mid Y)$, $P(Z \mid \bar{Y})$, and $P(\bar{Z} \mid \bar{Y})$.

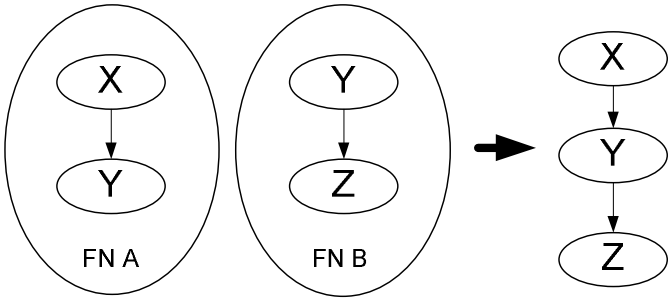


Figure 3.14: Merging two FNs into a BN

During the merging process, the nodes X and Z will be processed without problems. These nodes do not appear in multiple FNs and thus no merging has to be performed for these nodes. The node Y on the other hand appears in both of the FNs, and merging will be necessary. In FN A, the node Y is a child and has a CPT with 4 entries. in FN B, the node has a prior distribution and only has 2 entries. After merging, the node will have a CPT with 4 entries, which is exactly the same as in FN A. Node Y in FN B has the entries $P(Y)$ and $P(\bar{Y})$, these then must be merged with the entries from Y in FN A. Since these entries are conditioned on the node X, a problem arises. FN B does not contain the node X and thus for example, there are no samples available that can be used to calculate $P(Y \mid X)$. The other way around, it is possible to calculate $P(Y)$ from samples of FN A. This is a result of Bayes' Rule (Equation 2.2):

$$P(Y) = \sum_i P(Y \cap X_i) = \sum_i P(Y \mid X_i)\, P(X_i) \ . \tag{3.16}$$

So it is possible to calculate $P(Y)$ using the distributions $P(Y \mid X)$ and $P(X)$ and although this could be used when a BN is decomposed into the FNs, this is useless for merging FNs. As mentioned before, samples from a prior distribution must be changed into samples for a conditional distribution to be able to merge them with the other conditional samples of the other FN. The necessary information is simply not available in the other FN (in the case of the example in Figure 3.14 this is FN B).

A merging method for merging the FNs into the BN that is mathematical sound was not found. Therefore there was decided that at this time to not include FN merging in the program. Now the FNs with their samples and histograms are shown to the user of program (most likely a domain expert and/or a knowledge engineer), and the user must decide how he or she wants to fill in the BN. This was already the original idea, to give the user more control over the end result, but with just one histogram for each entry of the nodes. Perhaps as part of some future work a merging method can be found that is mathematically sound, or that is a very close approximation.

Chapter 4

Design

4.1 General Overview

The proposed elicitation method can be used as a tool to make the neccesary knowledge elicitation process to develop a Bayesian network less invasive for the domain expert. It is to be integrated in GeNIe, where a knowledge engineer and/or a domain expert can graphically create a Bayesian network and use the noninvasive method to fill the CPTs.

To develop a Bayesian network the following steps will normally be performed by the knowledge engineer and/or domain expert:

1. Design a BN structure.
2. Elicit probabilities for the CPTs that resulted from step 1.
3. Test the BN to see if it accurately represents the knowledge from the domain expert.
4. If necessary, revisit step 1 and/or 2 to adjust the BN structure or CPT entries to improve the result.

When using the method for the elicitation process, step 2 will need to be changed to represent the change in the elicitation process:

1. Elicit probabilistic information statements.
2. Run the method.
3. Using the results from the method determine probabilities for the CPTs.

These new steps will replace the original step 2. Figure 4.1 shows the steps, new and old, in a block diagram. Most of these steps are already present in some way in GeNIe. It can already be used to develop a BN. The structure can be designed, the CPTs can be filled, and the BN can be tested by using

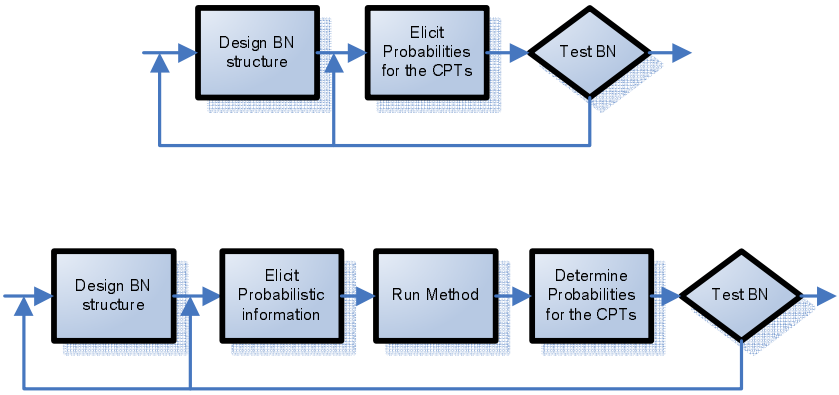


Figure 4.1: Steps to develop a BN

the inference engine provided by the SMILE library. The missing steps are the elicitation of probabilistic information, the execution of the method, and determining the CPT entries using the result of the method.

Thus, still missing are the implementation of the method and a graphical interface to use the method in GeNIe. These are the parts that have been designed. The GUI is designed as an extension of GeNIe and the method is implemented as a subsystem used by GeNIe, as illustrated in Figure 4.2. The method makes use of the SMILE library and a few very small adjustments and additions have been make to the SMILE library that were necessary to ensure that the method would be working correctly. The design of the user interface will be discussed in Section 4.4, this section will end with a general overview of the design of the method.

The method, as proposed in (Druzdzel & van der Gaag, 1995) and further extended in Chapter 3, can be divided into a couple of steps. First, the inputted Bayesian network is decomposed into the different family networks and the inputted statements are presented to all the FNs. Second, every FN translates every relevant probability statement into constraints. A probability statement is relevant when it only contains information on nodes that are in the FN. Irrelevant statements are discarded. Third, every FN gathers all linear constraints and uses them to tighten the probability bounds for all its constituent probabilities. If a FN does not have linear constraints, the boundaries stay unchanged with an lower value of 0 and an upper value of 1. Fourth, the FNs perform sampling in their respective probability hyperspace to generate 1000 valid samples for each FN. The samples must be compatible with all the constraints relevant for the FN. Fifth, all the FNs process the acquired samples to generate samples for each of the CPT entries of all the nodes in the FN. Finally, this data is outputted back to GeNIe where it will be able to display histograms for all the CPT entries. In Figure 4.3 these steps are shown in a block diagram.

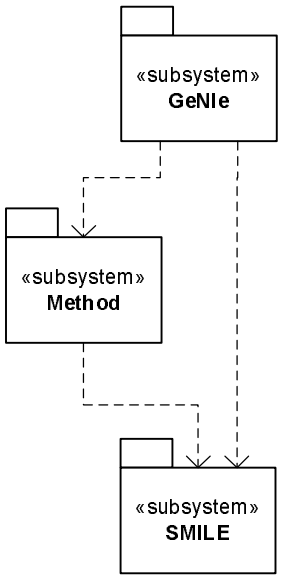


Figure 4.2: The different subsystems

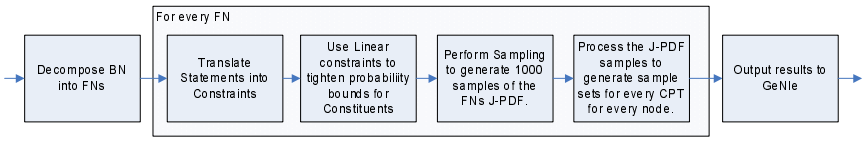


Figure 4.3: Overview of the method

4.2 Actors and Use Cases

This section discusses the actors and use cases that are relevant for the design of the method and its user interface.

4.2.1 Actors

Actors are people, organisations, or external systems that will be using a system. In the case of GeNIe and the method three different actors can be identified:

- Domain expert,
- Knowledge engineer,
- Decision maker.

The *domain expert* is someone that has a considerable amount of knowledge in the domain that is to be modeled by a Bayesian network. It is the knowledge of this expert that normally will have to be elicited to build the

BN. The expert may or may not build the BN in GeNIe himself, this depends on the situation.

When the problem that has to be modeled by a BN becomes more complex it is likely that a *knowledge engineer* will work with the domain expert to build the BN. In this situation the knowledge engineer will elicit knowledge from the expert and translate this knowledge into a BN structure and CPT entries for the nodes of the BN. Then together the expert and the engineer will test the BN to see if it accurately represents the elicited knowledge.

The *decision maker* is the person or organisation that will actually use the BN developed by the expert and the knowledge engineer for making decisions. By feeding the BN information, by setting evidence variables, and performing inference on the BN the decision maker will get a posteriori distributions over the other variables in the BN and he or she will be able to use this information to hopefully make better decisions. The decision maker actor is only mentioned for completeness. The actor plays no role in the knowledge elicitation process, which is the focus of the method, and it will not be necessary to use it in use cases for the method and its GUI.

4.2.2 Use Cases

Use cases are used to describe functional requirements of a system. They describe scenarios of how an actor will interact with the system. In Table 4.1 the most important use cases for the method and its GUI are presented.

Table 4.1: Use Cases

Use case name	`AddConstraint`	Use case name	`EditConstraint`
Actor	domain expert, knowledge engineer	Actor	domain expert, knowledge engineer
Entry condition	1. The actor has left-clicked the 'Add constraint' button.	Entry condition	1. The actor has left-clicked a constraint in the constraint list. 2. The actor has left-clicked the 'Edit constraint' button.
Flow of events	2. The 'Add constraint' window appears. 3. The actor creates a constraint using the controls in the window. 4. The actor left-clicks the 'Add' button.	Flow of events	3. The 'Edit constraint' window appears. 4. The actor edits constraint using the controls in the window. 5. The actor left-clicks the 'Done' button.
Exit condition	5. The constraint is added to the constraint list and the 'Add constraint' window is closed.	Exit condition	6. The constraint is edited and the 'Add constraint' window is closed.
Use case name	`RemoveConstraint`	Use case name	`ExecuteMethod`
Actor	domain expert, knowledge engineer	Actor	domain expert, knowledge engineer
Entry condition	1. The actor has left-clicked a constraint in the constraint list. 2. The actor has left-clicked the 'Remove constraint' button.	Entry condition	1. A Bayesian network structure must be present. 2. The constraint list contains at least one constraint. 3. The actor has left-clicked on the 'Start method' button.
Flow of events	3. The actor is prompted by a confirmation box. 4. The actor left-clicks on the 'Yes' button.	Flow of events	4. The method is performed using the constraints and the BN.
Exit condition	5. The constraint is removed from the constraint list.	Exit condition	5. The 'Results' window is opened and the actor can now view the Histograms.
Use case name	`SetCPTValue`		
Actor	domain expert, knowledge engineer		
Entry condition	1. The method must have been executed and completed. 2. The 'Results' window must be opened. 3. The actor has selected a node and a CPT entry int the 'Results' window.		
Flow of events	4. The actor types the value in the appropriate edit box in the 'Results' window. 5. The actor left-clicks on the 'Save entry' button.		
Exit condition	6. The value for the CPT entry is saved to the BN.		

4.3 Classes

GeNIe and SMILE are written in C++, an object-oriented programming language. Since the method was to be implemented in GeNIe it was a natural choice to also implement the method using C++. When designing software using an object-oriented design methodology the focus of the design is centered around objects. Objects encapsulate relevant data and programming code. The programming code is referred to as methods, functions that are (normally) performed on the encapsulated data of the object. Different

types of objects can interact with each other by calling each others methods. Objects are described in what is called a class. When an object is created in a program it is instantiated. An object is an instance of a class. A class could be called the blueprint of the object. In the class all the variables and the methods are defined.

UML class diagrams have been created that describe the different classes created for the method and the interaction between the different classes. First an overview of the classes is discussed, then the classes are discussed in a little more detail.

4.3.1 Overview

As mentioned in Section 4.1 and illustrated in Figure 4.2 the method is implemented as a subsystem that is used by GeNIe and that uses parts of SMILE. An overview of its class structure is illustrated in Figure 4.4. The method's GUI is a part of GeNIe and is not shown separate of GeNIe. The main classes are DSL_noninvasiveElicitation, DSL_familyNetwork, DSL_Order, and the classes used for the implementation of an expression tree. To keep the class diagram clear the classes that implement the expres-

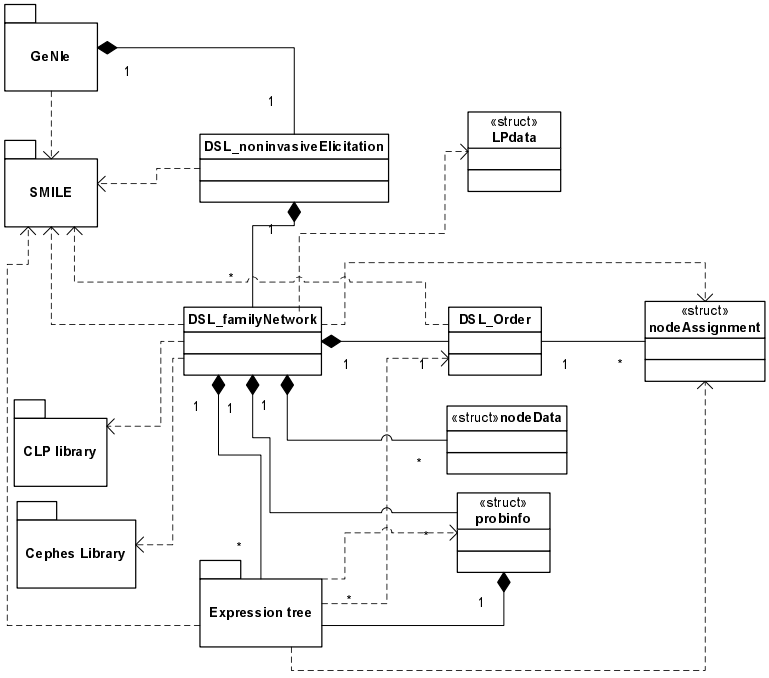


Figure 4.4: Overview of the classes

sion tree have been grouped into a package, called Expression tree. Figure 4.5 shows the class structure of those classes. Here the main classes are DSL_expTree and DSL_Ineq_algebraicElement. The former acts as a con-

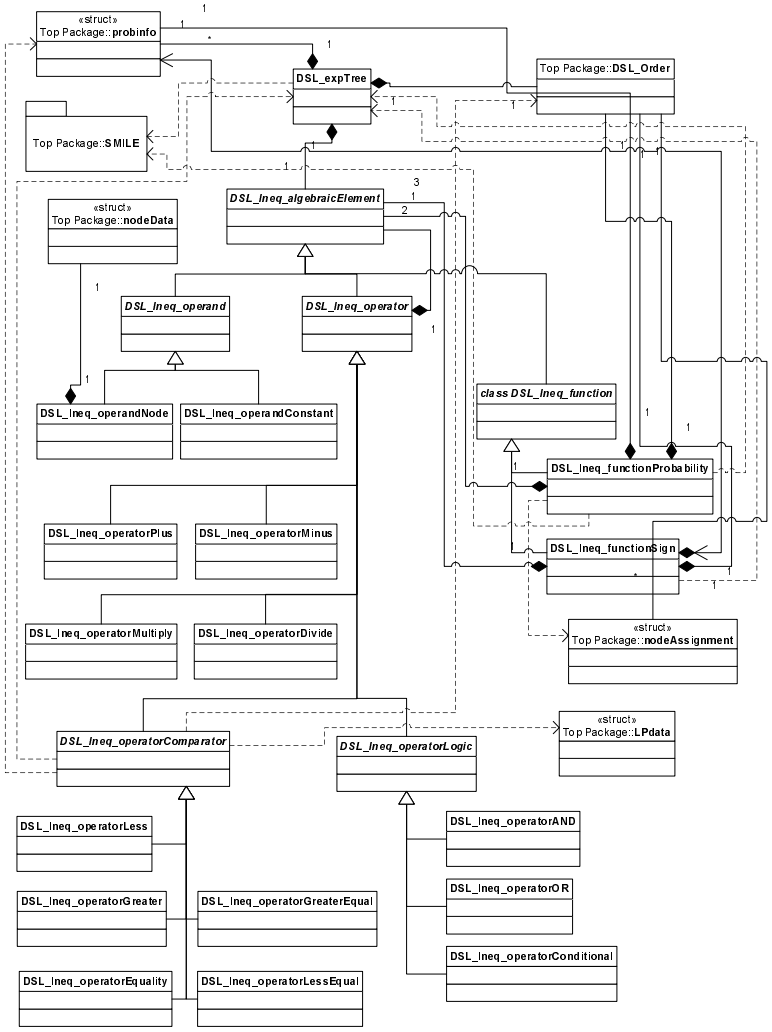


Figure 4.5: Expression tree package

tainer for the expression tree and the latter is the base class of all the different types of nodes that the tree can have. The different nodes represent the different operators that can be used to create an expression. The different types of operators and operands that are implemented are:

- Variables,
- Constants,
- Arithmetic operators: $+, -, *, /$,
- Comparators: $=, >, <, >=, <=$,
- Logical operators: AND and OR,

- Special node to support the creation of conditional probabilities,
- Functions:
  - Probability function: $P(\cdots)$,
  - Qualitative Influence function: $S(V_0, V_1, +)$.

The implementation of the expression tree is based on the implementation of an equation tree in SMILE. It was necessary to create a new implementation because inequalities needed to be supported by the expression tree to be able to implement some of the probabilistic statements proposed by (Druzdzel & van der Gaag, 1995).

4.3.2 DSL_noninvasiveElicitation

The DSL_noninvasiveElicitation class, is the class that directly interacts with the method's user interface. To use the method an object of this class must be created and initialised with the Bayesian network the method must be run on. After initialisation the execute method can be called to run the elicitation method. Necessary parameters are the probability statements from the expert and an object to store the results from the method in.

The class' execute method decomposes the BN into the different family networks and then creates instances of the DSL_familyNetwork class for each of the FN. All the probability statements are sent to each of the FNs. Later, each FN decides which statements are relevant for itself.

The method has been divided into 4 phases:

1. Setup,
2. Linear Programming,
3. Sampling,
4. Output Generation.

These phases are all implemented in the DSL_familyNetwork class. The DSL_noninvasiveElicitation class is responsible for the execution of these phases. To detect possible errors in the probability statements earlier each of the phases is processed by all the DSL_familyNetwork instances before the start on the next phase. This way a lot of time can be saved by terminating the method before some of the more time consuming phases are executed. The sampling phase for instance, depending on the number of dimensions of the sample space, can take many hours to complete.

When all the phases are performed by all the FNs the output data is passed back to the user interface and then the DSL_noninvasiveElicitation object can be deleted.

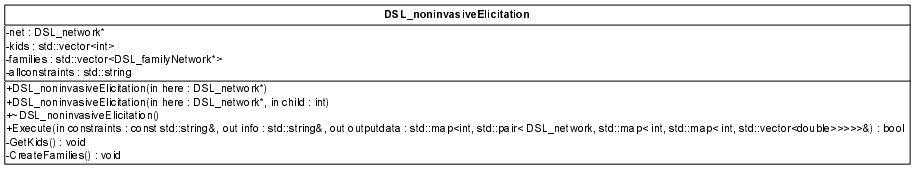


Figure 4.6: The DSL_noninvasiveElicitation class

4.3.3 DSL_familyNetwork

The DSL_familyNetwork class can be considered the work horse of the whole system. It contains a complete implementation of the method which, as mentioned in Section 4.3.2, is divided into four phases. Each phase is implemented using a couple of functions.

The **setup phase** creates the datastructure that will contain all the FNs probabilistic statements. It will decide if a statement is relevant for the FN and will discard it if it is not. The relevant statements are translated into the expression trees necessary for checking samples, and these trees are linked with the datastructure that will contain the values of the constituent probabilities during sampling. This constituent probability datastructure (CPD) that the trees are linked to has been created earlier, during the initialisation of the DSL_familyNetwork object.

Linking the trees to the datastructure means that variables in the trees will point to the values of the same variables in the probability datastructure. This way values can be updated in the datastructure and then when determining if a sample is valid, it is not necessary to copy the new sample values to the variables of all the trees. The trees read these values directly from the CPD. This can save a lot of time, especially when the trees get larger and contain a lot of variables.

In the **Linear Programming** phase the linear expression trees are gathered and used to try to tighten the probability bounds of the constituent probabilities. Using symbolic math the expression trees are manipulated into a form that makes it easier to decide if an expression is linear. The steps necessary to decide linearity of an expression are implemented in the classes that together form the expression tree package (Figure 4.5). Once the linear trees are collected, the coefficients of the variables are extracted to create the matrix necessary for the CLP library to perform LP. The LP process is performed for every constituent probability variable twice: once to calculate the minimum bound and once to calculate the maximum bound. There is no need to change the matrix, but the objective function will need to be changed everytime the calculation is performed. The results of the calculation, the new upper and lower bounds for the constituent, are saved in the CPD of the family network.

If there are conflicts (Section 3.6) between the linear constraints the CLP

library can detect this, and the conflicting probability statements, which can be found by using a link to the statement in the conflicting constraint, are reported back to the user using a text message. After a conflict has been detected the method is terminated.

During the **Sampling** phase samples of joint probability distributions for the FN are generated and then are evaluated by the constraints. The Dirichlet sample generator discussed in Section 3.5.2 is used in this phase to generate samples as efficient as possible. Also, as long as no valid sample is found, rejected sampled are saved to be used as data to determine if there are constraints that are causing conflicts. Once a valid sample is found, the saved samples are discarded. The sampling process as described in Figure 3.9 has been implemented as functions for the DSL_familyNetwork class.

The **Output generation** phase is the last phase of the method. Here the samples of the joint probability distribution of the FN are transformed into samples for all the CPT entries of all the nodes in the FN. This has been implemented hierarchically. The DSL_noninvasiveElicitation object calls a function to start the phase. This function, CreateFNSamples, ensures that all the nodes get samples for their CPT entries. It does this by calling another function for each node in the FN. This function, CreateNodeSamples, will ensure that samples will be created for all CPT entries of 1 node. CreateNodeSamples relies again on another function, CreateEntrySamples, that all the samples will be created for an entry. And finally, the CreateEntrySamples function relies on the getCPTEntry function to calculate a sample for the CPT entry from the original J-PDF sample. After all the data has been collected it is sent back to the DSL_noninvasiveElicitation object, where it will combine it with the data from the other FNs so the end result can be sent to the user interface of the method.

4.3.4 DSL_Order

For every family network the method creates the equivalent joint probability distribution. Every constituent in this distribution represents a state of all the variables. Although it does not really matter in what order the constituents are placed in the CPD, it does become relevant when the output data is being generated. The data is presented to GeNIe, and GeNIe uses a certain format (provided by SMILE) to store CPT entries.

The DSL_Order class is designed to define a node ordering for the nodes, so that a (complete) state of nodes can be translated to the respective constituent variable and vice versa. Other classes use the class to translate a probability into the equivalent summation of constituents, to check if a probability statement is relevant for the FN, or to get more information on the nodes in the FN and their outcome values.

Some useful information that the DSL_Order class can give is: the names of parent nodes of a node, the names of the outcomes of a node, the number

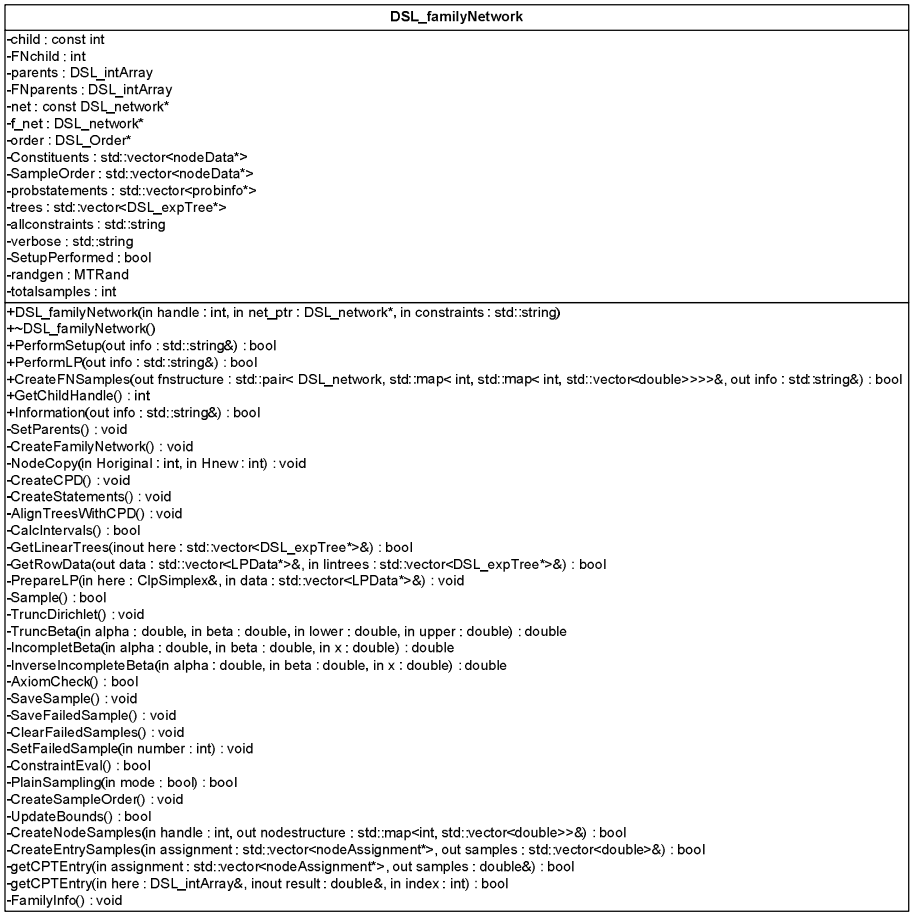


Figure 4.7: The DSL_familyNetwork class

of outcomes of all the nodes, and the constituent index of a state of the FN. The availability of this information makes implementing other parts of the method easier.

4.3.5 DSL_expTree

An important part of the method consists out of translating the expert's probability statements into constraints that consist out of the constituent variables. These constraints are represented using a (binary) expression tree. The constraints are inputted as plain text. The text is parsed to build the expression tree. The parser used for the method is based upon a parser that is implemented in SMILE which is used for parsing equations and creates equation trees. The parser has been extended to be able to deal with probabilities and qualitative probability statements.

The implementation for the expression tree can be divided into two parts:

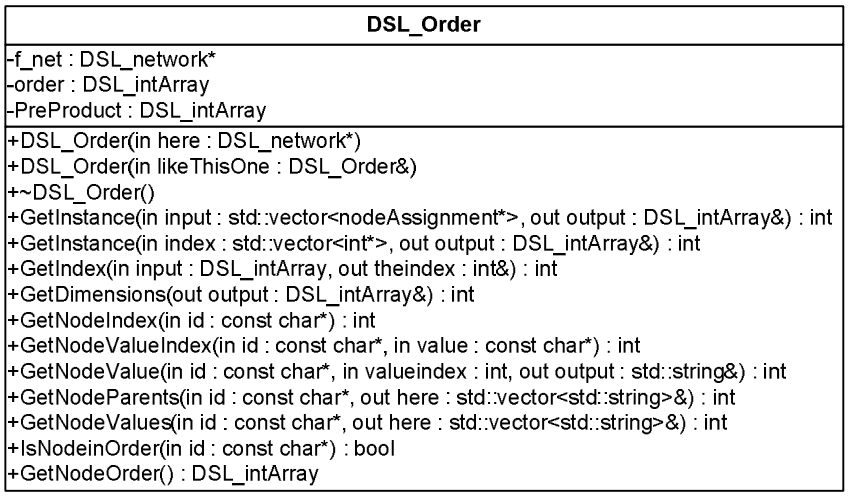


Figure 4.8: The DSL_Order class

the DSL_expTree class that acts as a container of the expression tree and the DSL_Ineq_algebraicElement class and its descendants that represent the nodes of the tree. The DSL_expTree class contains the parser that reads a probabilistic statement in text form and creates the expression tree. The class is the interface to the tree. When a constraints needs to be evaluated, or when there needs to be determined if a constraint is linear, relevant functions from the DSL_expTree class are called.

4.3.6 DSL_Ineq_algebraicElement

All the different types of nodes in the expression tree are derived from this class. The class contains some of the basic attributes and functions that are necessary for all the different types of nodes. The class is based on a similar class used in SMILE for equation trees. Most of the functions were left unchanged but some were altered and several functions were added to the class. Extra functions were necessary to be able to perform some symbolic mathematic actions on the expression tree. The extra symbolic math operations were necessary to be able to manipulate the expression tree into a form that would simplify checking if a constraint is linear.

The functions that implement the symbolic math procedures described in Section 3.4 and other simplification procedures are divided over the different types of nodes. Most of it is implemented in the different operator nodes, but most of these functions are defined in the DSL_Ineq_algebraicElement class to ensure that the functions can always be used.

The different types of nodes available to create expressions can be divided into three types: operands, operators, and functions. This is represented in the design by the three classes that inherit from DSL_Ineq_algebraicElement.

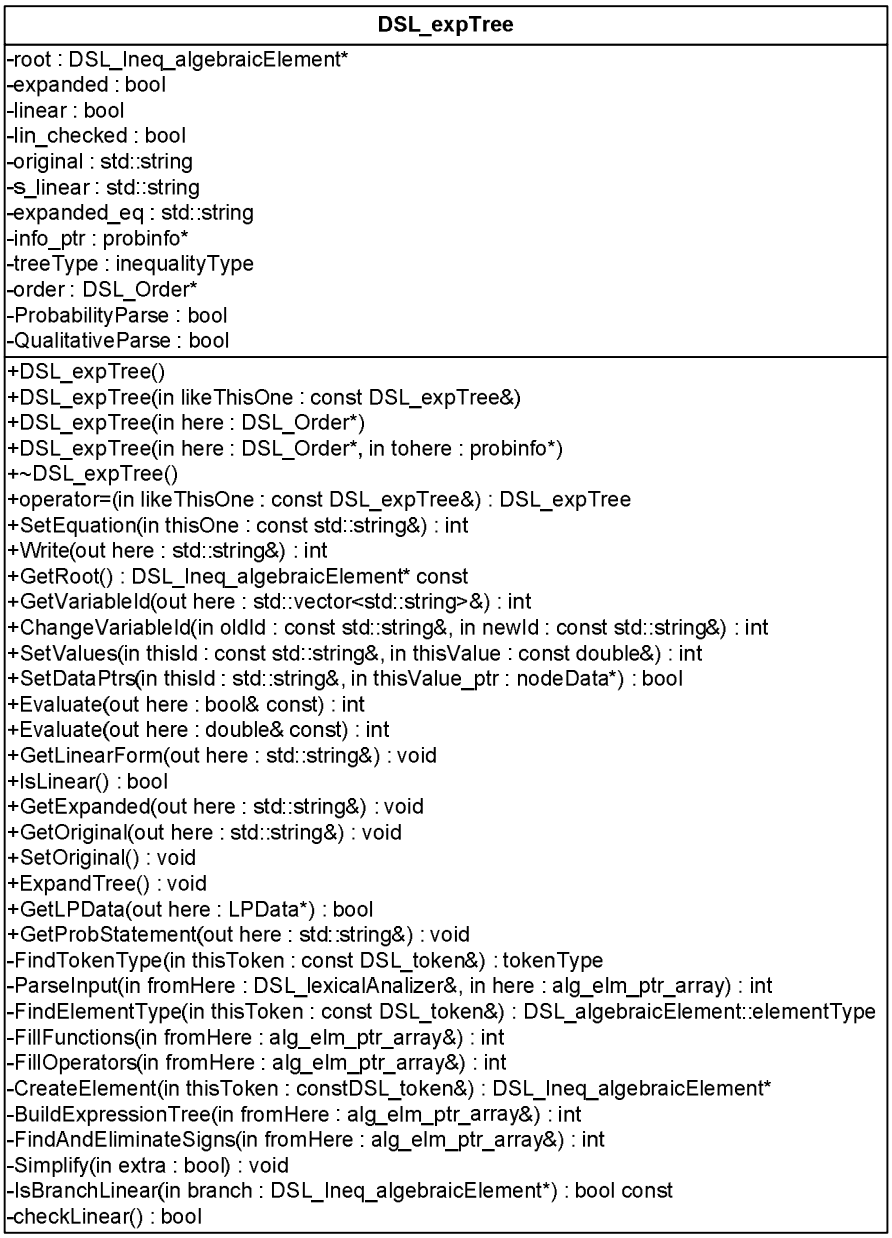


Figure 4.9: The DSL_expTree class

These classes are:

- DSL_Ineq_operand,
- DSL_Ineq_operator,
- DSL_Ineq_function.

Again, these classes are based on their equation tree counterparts from the SMILE library. These classes stay abstract; they are not instantiated. Only classes that represent actual nodes in the tree will be instantiated. But the classes that represent the actual nodes inherit functionality, specific for the type of node, from the base classes mentioned above. This includes specific symbolic math functions that are relevant for these nodes.

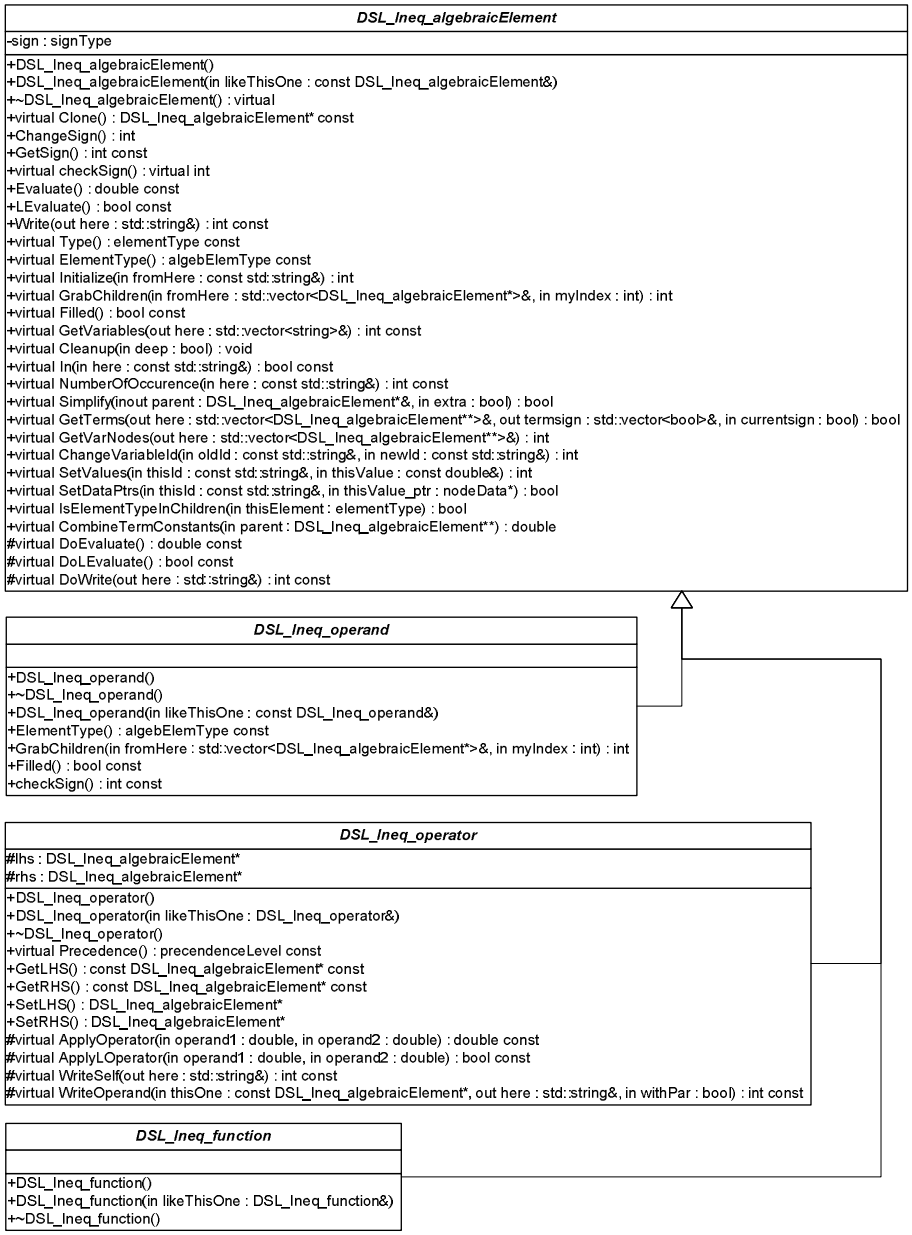


Figure 4.10: The DSL_Ineq_algebraicElement class

4.3.7 DSL_Ineq_operand

The DSL_Ineq_operand class serves as the base class for the two different operands implemented for the expression tree: constants and variables. Constants are represented by DSL_Ineq_operandConstant objects that have the value as data. Variables are represented by DSL_Ineq_operandNode objects that have a pointer to the constituent variable they represent as data. Both classes have some basic functionality used for getting and setting data. The DSL_Ineq_operandNode class has also some functions that makes it possible to compare and sort variables.

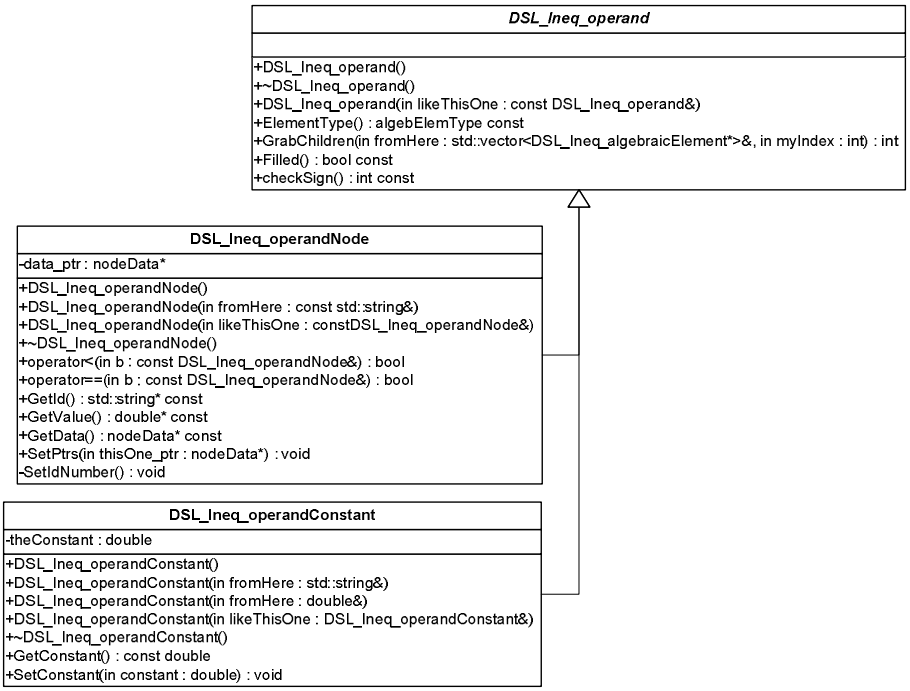


Figure 4.11: The DSL_Ineq_operand Class

4.3.8 DSL_Ineq_operator

The DSL_Ineq_operator class is the base class of all the different operator classes. The class implements the basic functionality necessary for all the operators. The class has two DSL_Ineq_algebraicElement pointers that are used to create the tree. Furthermore some common functions for getting and setting the pointer data are implemented and also some symbolic math functions that are used by all operators are implemented in the class.

The different types of operators can be divided into three groups:

- Arithmetic operators,
- Comparator operators,
- logic operators.

For the four different arithmetic operators $(+, -, *, /)$ four classes have been created that are based on their counterparts from the equation tree implementation from SMILE. The created classes are:

- DSL_Ineq_operatorPlus,
- DSL_Ineq_operatorMinus,
- DSL_Ineq_operatorMultiply,
- DSL_Ineq_operatorDivide.

These classes each implement functions for the specific arithmetic operation that is to be performed by the node they represent. In these classes parts of the symbolic math functionality is implemented. Each class has specific math routines that are relevant for the arithmetic operator that the objects of the class represent. The other types of operators are represented by the base classes DSL_Ineq_operatorComparator and DSL_Ineq_operatorLogic, and their respective derived classes.

4.3.9 DSL_Ineq_operatorComparator

The original tree implementation in SMILE was meant for equation trees, but constraints provided by the expert could also be inequalities. This meant that the SMILE implementation needed to be changed so that equations and inequalities could be represented with the tree. Originally the equation operator was implemented in a class similar to DSL_expTree, but simply adding the other comparator operators to the class was not an option. It was necessary to create a base class for all the comparators, DSL_Ineq_operatorComparator, and the comparators are implemented as derived classes of this base class. The reason that this was necessary was that comparators were going to be used in more locations in the tree than

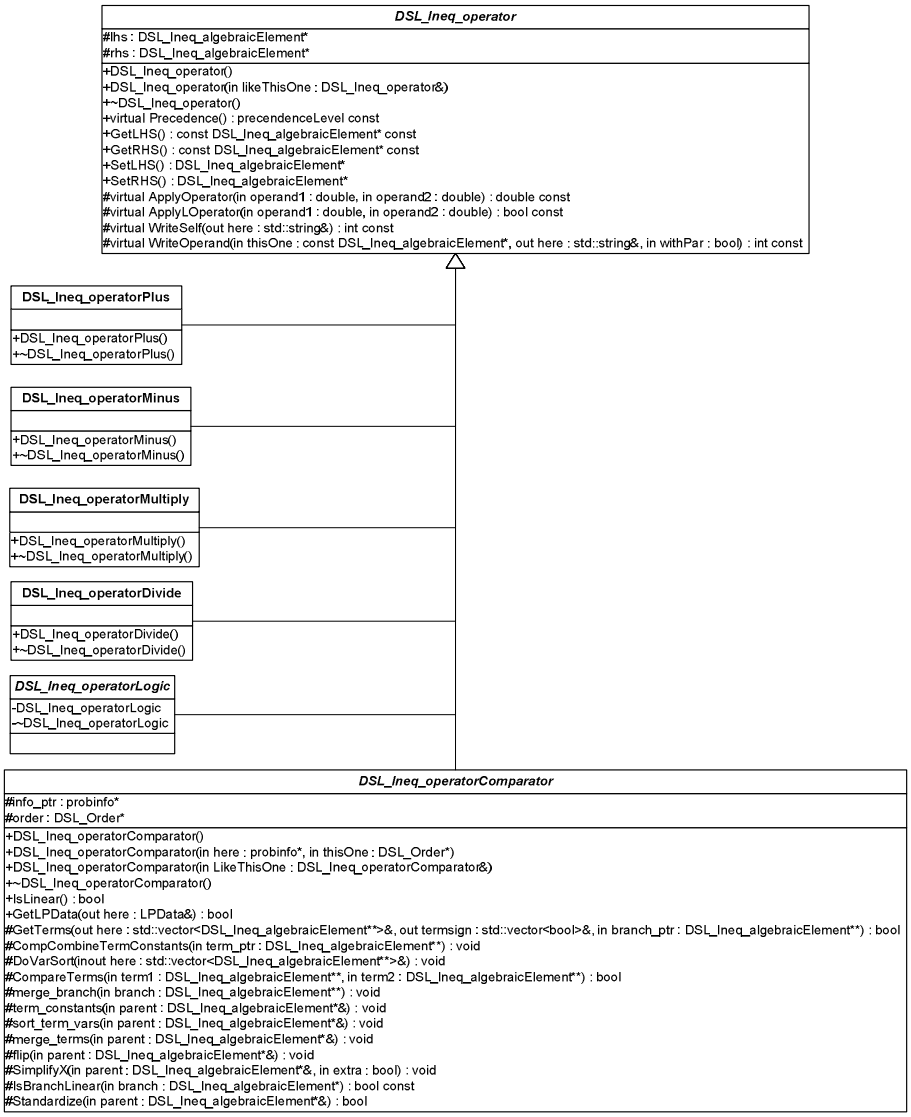


Figure 4.12: The DSL_Ineq_operator class

just the root. Another type of node that was added to the expression tree was the probability node. The node is used as an intermediate form to represent probabilities before the constraint is translated into its canonical form. The node represents the probability of a certain event:

$$P\left(VariableName = value\right) \ .$$

In the expression tree the function $P\left(\ldots\right)$ is represented by the function node object DSL_Ineq_functionProbability, and underneath this node there is a subtree that represent the logical expression $VariableName = value$.

Besides equations valid logical expressions for probabilities can also be inequalities, so it is necessary to have nodes that represent all the different comparators in the probability subtree. To represent variable name and variable value, the class DSL_Ineq_operandNode has been used. The subtree under the probability node only exists temporarily and is removed with the probability node and replaced by a subtree that contains a representation of the probability using the constituent variables.

The DSL_Ineq_operatorComparator class contains a large part of the symbolic math routines. Since comparators will always be the root of the expression tree of the constraints, math routines for the merging of terms, determining of linearity, and standardization of the constraint are implemented in the base class. This way all types of expressions (equations and inequalities) have easy access to the math routines.

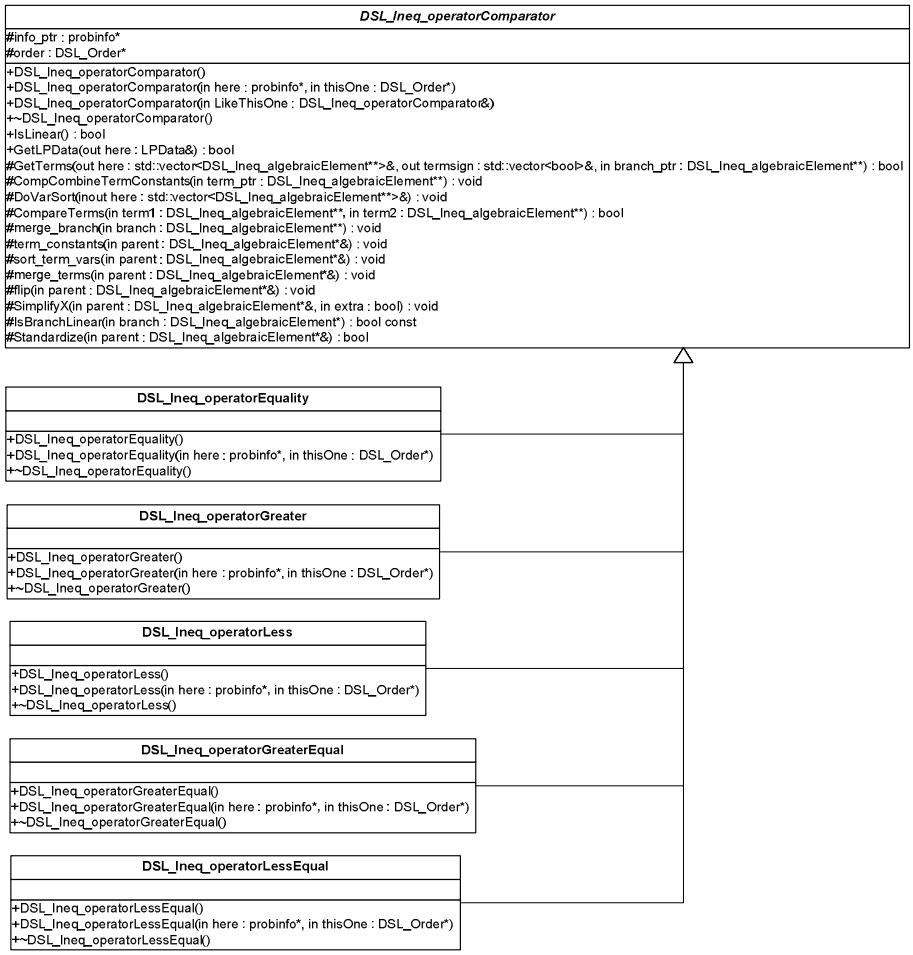


Figure 4.13: The DSL_Ineq_operatorComparator class

4.3.10 DSL_Ineq_operatorLogic

The DSL_Ineq_operatorLogic class has been created to support the AND and the OR operators for the probability subtree (mentioned in Section 4.3.9). This allows for more elaborate probabilities to be specified. Also the operators are necessary for the conversion of probabilities of the type $P(A \geq a)$. For further processing the inequality needs to be changed into a equation. For a variable A with three different, ordered, outcomes a_1, a_2, and a_3, and probability $P(A \geq a_2)$, this is done in the following way:

$$\begin{aligned} P(A \geq a_2) &= P((A = a_2) \vee (A = a_3)) \\ &= P(A = a_2) + P(A = a_3) - P((A = a_2) \wedge (A = a_3)) \\ &= P(A = a_2) + P(A = a_3) \ . \end{aligned}$$

In this simple example the third term with the AND operator disappears because the example only has one variable and a variable can only have one outcome at a time. Since A cannot be a_2 or a_3 at the same time $P((A = a_2) \wedge (A = a_3))$ must be 0. In situation where there are multiple variables the AND term does not have to be 0. The desired form of the probability subtree is the situation where the sub tree consists only out of one variable assignment or multiple assignments connected by AND operators. This is because probabilities in this form can easily changed into the form using constituent probability variables. This is done by using an algorithm based on the marginalization process (Section 2.2.4).

Three different classes are derived from DSL_Ineq_operatorLogic:

- DSL_Ineq_operatorAND,
- DSL_Ineq_operatorOR,
- DSL_Ineq_operatorConditional.

The first two represent the AND and the OR nodes, the third is used to be able to define conditional probabilities in the expression tree. The conditional node is only used temporarily, as quickly as possible the conditional probability is changed into the form:

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} \ .$$

At this point a new constraint is created that ensures that $P(B) > 0$.

4.3.11 DSL_Ineq_function

The original SMILE equation tree defined some functions that were used a lot in the context the equation tree was designed for. These functions were not as useful for the context the expression tree is used in. However, it

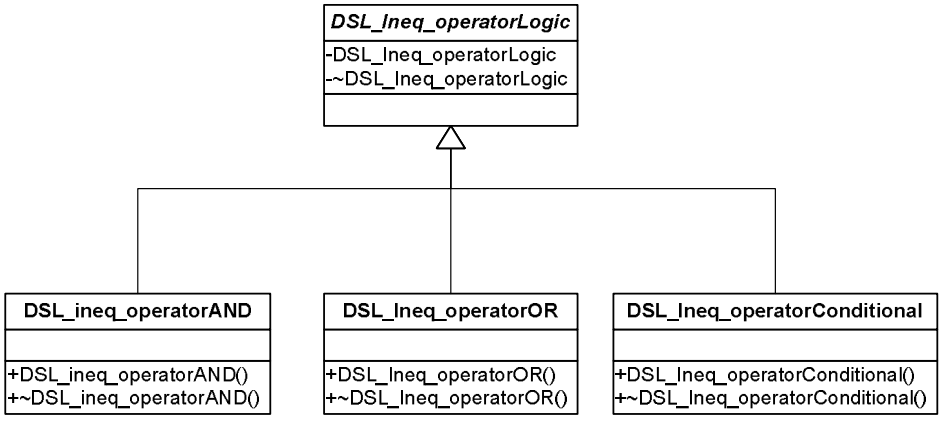


Figure 4.14: The DSL_Ineq_operatorLogic class

was necessary to be able to represent functions in the expression tree. The same structure for representing functions was used for the expression tree. This was more because of the way functions are parsed than for actually using functions as part of a constraint. The parser used in the SMILE implementation sees a function as follows:

$$functionname\,(parameter_1, \ldots, parameter_n)\ .$$

By examining the functionname the parser know what type of function it is and then creates the correct function object than knows how to handle the supplied parameters.

Two implementations for functions have been designed and implemented:

- DSL_Ineq_functionProbability,
- DSL_Ineq_functionSign.

The first represents probability nodes and the second represents qualitative influences. As explained in Section 4.3.9 probability nodes only stay in the expression tree temporarily. They are replaced by a subtree with constituents. This is done in three steps. First, the probability node and its subtree are manipulated in such way that there are no OR or inequality operators in the subtree. These are replaced by AND and equation operators. When necessary extra probability nodes are added in the tree. This happen when OR operators are replaced with AND operators:

$$P\,(A \vee B) = P\,(A) + P\,(B) - P\,(A \wedge B)\ ,$$

or when changing the inequalities into equations (see Section 4.3.10). Second, the subtree of the probability tree is replaced by an array representing the state of the family network. Each entry of the array represents a variable

and the data stored in the array represents the indices of the outcomes of the variables. When a probability represents a complete state of the FN, every entry of the array will contain the index of the outcome that the respective variable has in the state of the FN. When a probability does not represent a complete state but a marginalization, it will be necessary to sum out the variables that do not have an outcome assigned to them. In the array this is represented by giving the respective entries the value -1. Third, the array is used to create the subtree containing the summation of constituent assignments that will replace the probability node in the expression tree. A recursive algorithm has been designed that moves through the array and everytime a variable is encountered that needs to be summed out, a copy is created of the array, the index of the variable is set to 0, and the algorithm is called again using the new array as an input parameter. This process repeats itself until the array has a complete assignment of outcomes for all the variables. Then, using the DSL_Order object the index for constituent variable can be calculated and the constituent variable is added to a text string. Then the algorithms ends, and control is given back to the recursive algorithm that called it. Now, the index of the variable in the array is increased by one and another recursive call is performed. This process repeats itself until all relevant variables are summed out. When the process is done the text string contains the summation of the constituent variables. To create the subtree a DSL_expTree object is created and the text string is parsed. After parsing the probability node is replaced by the new tree containing the constituents.

The node that describes the qualitative influence works in the same way. It creates multiple new constraints (see Section 3.1.1), consisting out of probability nodes. These constraints will be processed like explained above until they only have constituent variables. The new constraints are created by the DSL_Ineq_functionSign object in text form, so they will need to be parsed first. This makes it easier to implement since each of the expression trees for the new constraints are now created by the parser and not by the DSL_Ineq_functionSign object. The object creates new DSL_expTree objects and initialises them with the constraints in text form. The tree objects then create the expression trees. The original qualitative influence constraint is changed into one of the new probability constraints that resulted from it. There can still be determined that originally it was a qualitative influence constraint.

4.4 User Interface Design

Most of the work performed by the method is done “under the hood”, i.e. it is not visible to the user. But the user does need to input the probabilistic information and has to evaluate the end result provided by the method. A

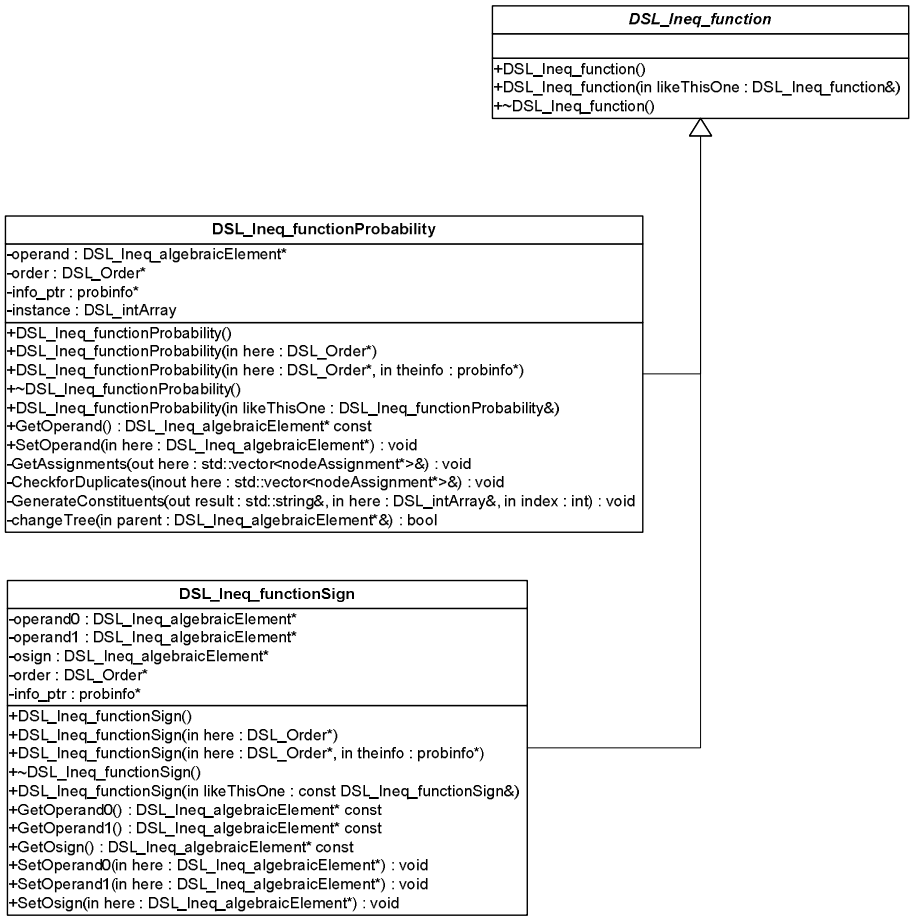


Figure 4.15: The DSL_Ineq_function class

good user interface will be necessary to make the interaction between user and the program as simple and effective as possible. A first design for a prototype graphical user interface has been created, but it will be necessary to perform usability studies to create an interface that can be used in a "production environment". At the moment more research and work needs to be done to perfect the method and currently any user interface usable by researchers and developers working on the method will do.

The user interface for the method is to be implemented in GeNIe. This implementation work will be done by Tomek Sowinski, the Decision Systems Lab Main programmer and the maintainer of the GeNIe code. The User interface can be divided into two parts: inputting the probabilistic information so the method can process them and presenting the results of the method to the user. Depending on if the method was successful the interface will display the different histograms for each of the CPT entries, or it will inform the user that something has gone wrong and will advise the

user what to do to get a valid result.

It is important that the collection of the probabilistic information from the user is performed in a user friendly way. This step represents the knowledge elicitation process and if it is performed in a way that confuses the user, the user may have problems inputting all the probabilistic information he or she has.

Since the method supports various types of probabilistic information, it is a good idea to examine the different information types and to specialise parts of the user interface for each of the information types. This specialisation will allow the elicitation process of the different types of information to be optimised.

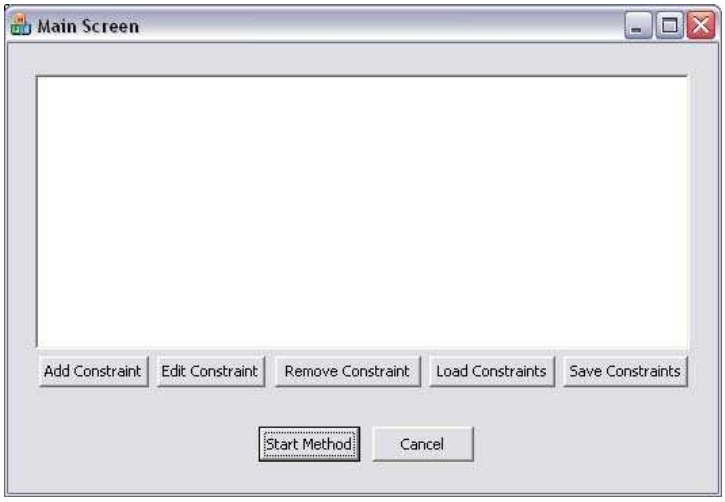


Figure 4.16: Main Screen

4.4.1 Prototype

Work has begun on designing a graphical user interface (GUI) for the method. Some very preliminary designs have been created for the different screens of the GUI. When the method is started, the main screen appears where the user will be able to add, edit, and remove constraints. The possibility to load and save a set of constraints is also to be possible. When all constraints are entered the method can be executed by pressing the "Start Method" button. When constraints are added to user will see a new screen where the constraints can be created. The basic design is shown in Figure 4.17, this particular screen allows for the creating of a point probability constraint. The upper right drop down box in the screen allows for the selection of the type of constraint to be created. The screen shows the Bayesian network

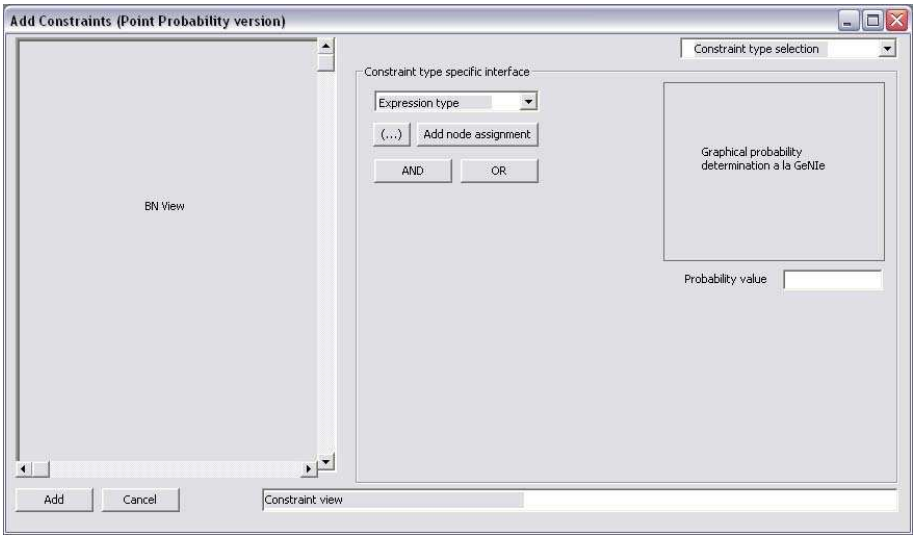


Figure 4.17: Adding a point probability constraint

that needs elicitation in the left half of the screen. At the moment it is assumed that this will be the current active BN opened in GeNIe. This screen is used to create a constraint of the type $P(\cdot)\,\#x$, where $\cdot$ is a node assignment created by using the popup window shown in Figure 4.18, the buttons AND, OR, and $(\ldots)$. The button $(\ldots)$ allows the use of operator precedence in the logical expression $\cdot$. The symbol $\#$ is one of the (in)equality signs $\# = \{=, >, <, \geq, \leq\}$, and x is a numerical value between 0 and 1. An example valid constraint that can be created with this window is:

$$P(A = true \wedge (B = false \vee C = true)) \geq 0.25\ .$$

The progress of the creation of the constraint is shown in the bottom of the screen where the constraint is displayed in text form.

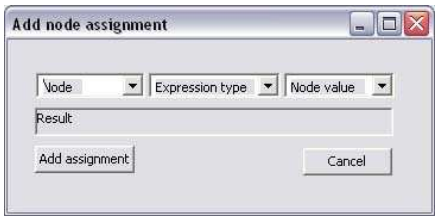
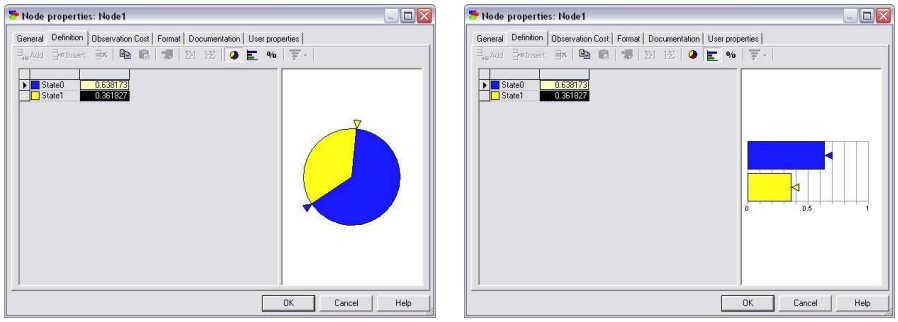


Figure 4.18: Adding an probability assignment

The node assignment popup window allows for a simple way to create a node assignment. Using the drop down boxes on the screen the node, its value and the desired expression type ($=, >, <, \geq, \leq$). The result is shown underneath and when the assignment is done, it can be added to the constraint by clicking the "Add assignment" button.

For the determination of the value x the graphical aids already implemented in GeNIe are to be used. GeNIe allows for the use of a pie chart or a bar chart to graphically determine values for (conditional) probabilities of a node.



(a) Pie Chart (b) Bar Chart

Figure 4.19: Graphically determining values for node entries in GeNIe

Although this is less useful for point probability constraints, for probability interval constraints this will be more effective since there are two values that need to be determined. In this situation using the pie chart representation may be the best approach. The pie chart can be divided into two parts, an area created by the upper and the lower value, and the rest.

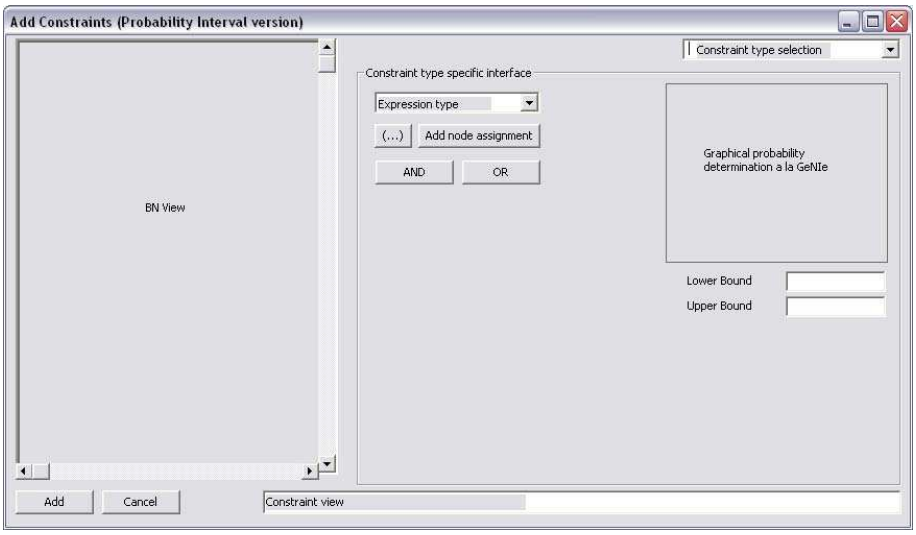


Figure 4.20: Adding a probability interval constraint

The probability interval constraint screen is very similar to the point probability screen. The only difference is that are two numerical values necessary. The constraint has the following form:

$$\begin{aligned} x_1 \# P(\cdot) \# x_2 \ , \\ x_1, x_2 \in [0,1] \ , \\ \# \in \{=, >, <, \geq, \leq\} \ . \end{aligned}$$

The constraint is created in the same way as with the point probability constraint. Comparison constraints, again, are created in the same way as point probability and probability interval constraints. The constraint has the following form:

$$\begin{aligned} a_1 * P(\cdot) \# a_2 * P(\cdot) \ , \\ a_1, a_2 > 0 \ , \\ \# \in \{=, >, <, \geq, \leq\} \ . \end{aligned}$$

The only notable difference between this constraint and the others is that the values a_1 and a_2 do not represent probabilities but real number, and that the constraint has two probabilities that are compared with each other instead of one. The three previous constraint types were quantitative constraints. The last constraint type that was implemented, was the one modeling qualitative influences. A first version of the GUI screen has been designed. At the moment it is very simplistic, it only contains drop down boxes for the parameters necessary to create the constraint. Necessary are two nodes and the type of qualitative influence (positive, negative, zero). The form of this

type of constraint is:

$$S\left(Node_1, Node_2, InfluenceType\right) \ .$$

The exact meaning of this statement is explained in Section 3.1.1.

It will be necessary to make the constraint input screens easier to understand. Adding online help that describes the different types of constraints and shows some examples might improve the GUI. At the present some knowledge of the different constraints will be necessary to be able to use

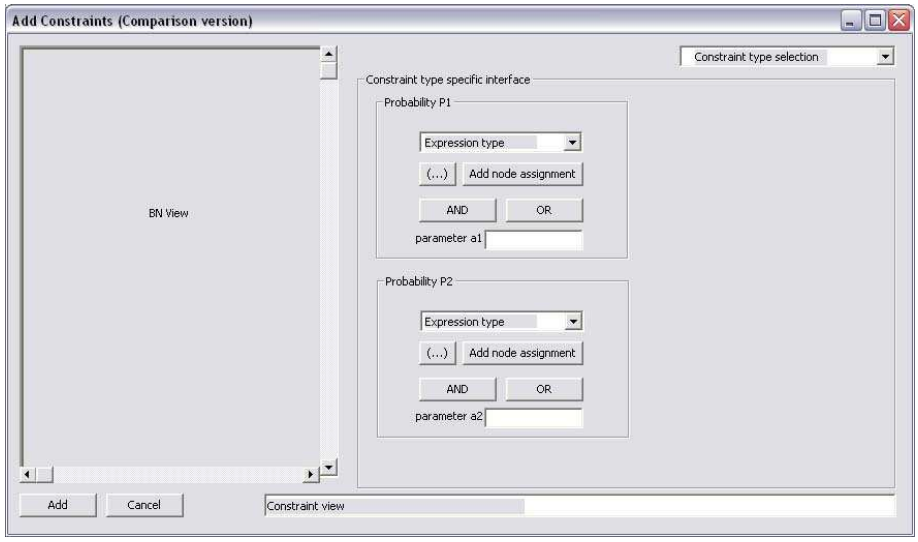


Figure 4.21: Adding a comparison constraint

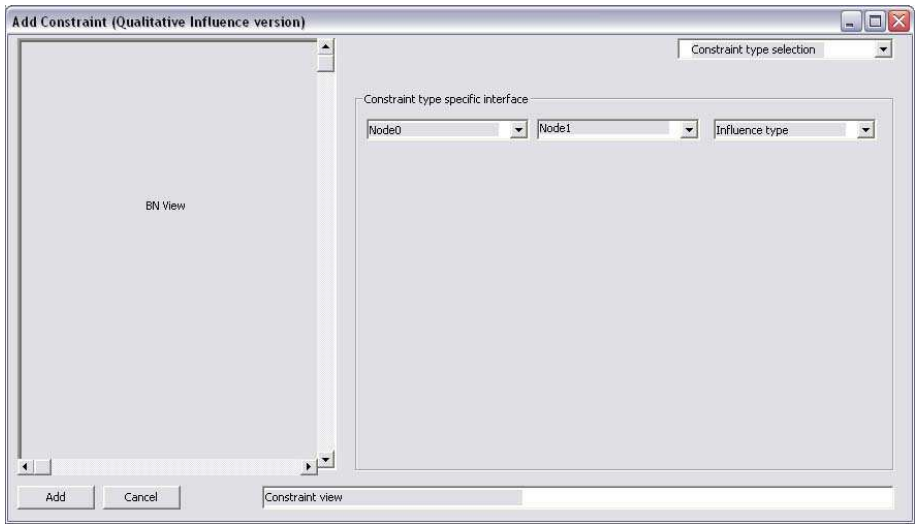


Figure 4.22: Adding a qualitative influence constraint

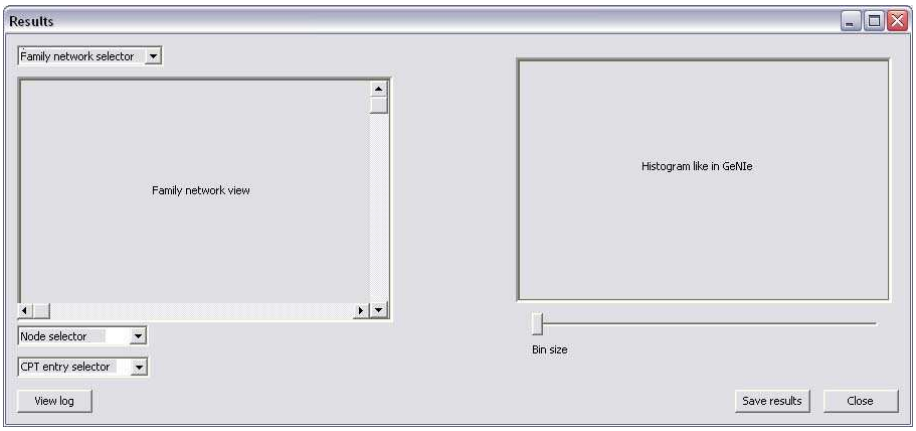


Figure 4.23: Screen showing results

the proposed GUI screens for the constraints effectively. The final screen designed for the GUI shows the results generated by the method. This first version of this screen shows the results of the method for every family network. Using a drop down box the family network can be selected. With a FN selected the results generated by the method for each node and CPT entry can be selected by using drop down boxes. When a node and an entry have been selected the screen shows the histogram of the data exported by the method to GeNIe. GeNIe already has code available for showing histograms of data. It allows for the bin size of the histogram to be changed by the user. The user can specify the number of bins used to divide the data. This code is to be used in the GUI.

At the moment it is still unclear how to transform results from the method (histograms) into concrete values for the different CPT entries. The histograms are to serve as guidelines when choosing the values of CPT entries. To make the GUI more effective, it will be necessary to be able to assign values for the CPT entries from the GUI. It still has to be examined how this can be accomplished with the main goals being that the GUI should be clear and user friendly.

4.4.2 Conclusion

The currently designed GUI is still just on the drawing board, mostly because effort and attention was focused on implementing and improving the method itself first before creating the GUI. Before the GUI will be completed several design iterations and usability studies will be necessary to get a GUI that is user friendly. When creating the constraints help should be available about the constraints. If something goes wrong when executing the method, i.e. conflicting constraints or not finding any valid samples within the set maximum number of samples, this must be communicated to the user in such a way that he or she should be able solve the problem. When the method does return results, the user should be guided in the process of choosing values for the entries. Further GUI design will be future work, currently no GUI has been implemented.

Chapter 5

Implementation and Testing

5.1 Approach

The method was implemented in an incremental fashion. five phases were identified that each needed to be completed before moving on to the next one. The five phases are:

1. Implementing the parts that process the input data.
2. Finding and integrating a suitable linear programming solver library.
3. Implementing Sampling.
4. Implement the parts that generate the output data.
5. implementing the user interface.

The implemented objects and functions of every phase were all tested to ensure correct operation.

5.2 Current Software

5.2.1 SMILE

SMILE stands for Structural Modeling, Inference, and Learning Engine and has been developed by the Decision Systems Laboratory of the University of Pittsburgh. It is a library of functions and classes that can be used for reasoning in graphical probability models. It is written in C++ and it is platform independent and fully portable.

Some parts of the SMILE library have been used for the implementation of the method. Some classes of the implementation have been based on classes originally in SMILE, but these classes have been extended in such a way that they no longer would be directly compatible with classes using the original classes from the SMILE library.

5.2.2 GeNIe

GeNIe, coming from Graphical Network Interface, is a development environment for building graphical decision-theoretic models. Like SMILE, it has been developed at the Decision Systems Laboratory of the University of Pittsburgh. It is written in C++ and it makes use of the MFC (Microsoft Foundation Classes) framework. Due to the heavy use of the MFC framework GeNIe is not as portable as SMILE and runs only on Windows operating systems. It is mainly used to develop graphical probabilistic or decision-theoretic models, once such a model is developed it can be used by other programs that use the SMILE library. This way models can be developed and tested in GeNIe, running on Windows, and then be implemented in an application that runs on another operating system.

The implemented method is a tool to simplify knowledge elicitation for Bayesian networks and will be used in GeNIe to make it easier for an Expert or a knowledge engineer to fill in the CPTs of a BN. A user interface has been designed (see Section 4.4) that will be implemented so that the method will be usable in GeNIe.

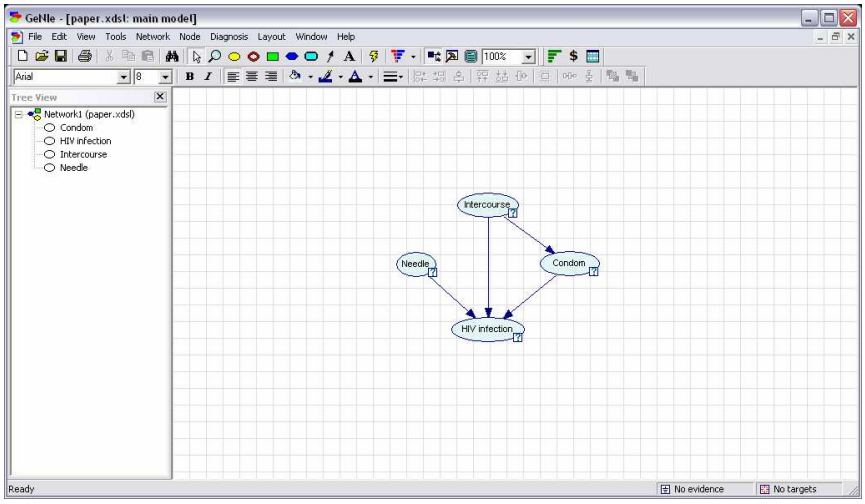


Figure 5.1: GeNIe screenshot

5.2.3 CLP Library

The CLP (COIN-OR Linear Programming) library is an open source (under the Common Public License (CPL)) linear programming solver that is written in C++. It has been developed by The Computational Infrastructure for Operations Research (COIN-OR) project. It is primarily meant as a library that is to be called from an application, but there does exist a standalone version that can solve lp problems in a certain file format.

The LP solver provided by the library has been used for calculating of new probability bounds for the constituent probabilities to reduce the sample space.

5.2.4 Cephes Library

The Cephes library (Moshier, 1989) is a C++ library written by Stephen Moshier that contains a large number of special mathematical functions. It is downloadable from http://www.moshier.net/#Cephes and can be used freely. The functions come in different types of precision.

5.2.5 Tools

For the implementation of the method Microsoft Visual Studio C++ has been used. The method implementation needs to have access to the SMILE library and the CLP library to function properly. GeNIe has been used for testing purposes. It has been used to create Bayesian networks used as input data for the method, and it has been used to analyse the output data generated by the method. GeNIe has extensive data analysis features that were useful to determine if there was anything wrong with the functions that generated the output data.

5.3 Processing Input Data

The method deals with two types of input data: Bayesian networks and probabilistic information. The Bayesian networks are specified in the format defined by SMILE, a DSL_network object. The probabilistic statements are specified in text form. When the method is run the Bayesian Network and the statements are provided by GeNIe. A prototype program was created that, using the SMILE library, could read a Bayesian network from a .xdsl file[1] and could read the statements from a .txt file. This way integration with GeNIe could be postponed until the method was implemented completely.

First the processing of the BN was implemented. This was implemented in the DSL_noninvasiveElicitation and the DSL_familyNetwork classes as described in Sections 4.3.2 and 4.3.3.

To implement the processing of the probabilistic statements the equation tree implementation of SMILE was examined to see if it could be modified so that it could be used as an implementation of an expression tree. At first the number of changes to the equation tree implementation was kept at a minimum, but soon it became clear that major changes were necessary to be able to provide all the functionality needed by the method. New classes were created that were based on the original SMILE classes. These classes

[1]The file format used by SMILE and GeNIe to store Bayesian networks.

were rewritten so that they would contain the necessary functionality. New classes were added that implemented new types of nodes for the expression tree. The parser supplied by the SMILE classes was extended so that it would support the new types of nodes of the tree. The basic symbolic math capabilities present in the SMILE implementation were expanded to fit the needs of the method, i.e. the ability to determine linearity of a constraint.

5.4 Finding and Implementing a Linear Programming Solver

After implementing input data processing, a preprocessing step to reduce the sample space was implemented. This step uses linear programming to calculate bounds for the constituents variables. To implement the preprocessing step a LP solver was necessary. There were 2 implementation options: create an own LP solver, or use a LP solver from a library. The choice was made to use a LP solver from a library. This would save time that otherwise would be spent designing, implementing, and testing a LP solver.

5.4.1 Choosing a Library

By choosing to use a solver from a library, it would need to satisfy certain requirements. Besides being capable to handle the input provided by the method some nonfunctional requirements needed to be satisfied. These requirements are:

- The license of the library should not influence the license of SMILE or GeNIe.
- The code of the library needs to be portable.

During the selection process the license under which the library is distributed was considered carefully. GeNIe and SMILE are not open source software. They have been developed over several years and to remain competitive with the competition the decision has been made to not make the source code available to the public. GeNIe and SMILE are, however, available free of charge. This rules out the usage of commercial LP solvers, since then it is likely that licenses for the LP library will be necessary for every download of GeNIe and SMILE, making it impossible to keep distributing them for free.

Since it is the intention of the DSL lab to keep GeNIe and SMILE available for free, it is necessary to use an open source library. Using an open source library, however, can also cause some problems. There are different types of open source licenses that have different effects on its "user". Some well known open source licenses are:

- GNU General Public License (GPL),
- GNU Lesser (Library) General Public License (LGPL),
- Berkeley Software Distribution License (BSD),
- Mozilla Public License (MPL).

The GPL has a big influence on non GPL software that uses GPL code. It does not allow proprietary programs to link with a GPL library. Only GPL software may link GPL libraries. So using a GPL LP solver would mean either voilating the GPL or changing the license of GeNIe and SMILE into a GPL, which is not what the decision systems laboratory wants. To provide a little more flexibility the LGPL was defined to make library linking by proprietary software possible. A LP solver licensed under the LGPL is therefore a valid candidate for use in the implementation of the method. Another candidate could be a solver that is licensed under the BSD license. Here the user can do almost everything with the code what he or she wants. The BSD license resembles public domain very closely. After some searching 3 different open source LP solvers were found:

- GNU Linear Programming Kit (GLPK),
- lp_solve,
- COIN-OR Linear Programming (CLP).

All solvers were licensed under a different license: GLPK uses the GPL, lp_solve uses the LGPL, and CLP uses the Common Public License (CPL). Since GLPK is licensed under the GPL it is eliminated by default since it is not allowed to be used in SMILE and GeNIe under their current license. The other 2 solvers could be used for implementation since their licenses allow proprietary software to link the libraries without repercussions.

The other requirement that the solver would need to satisfy, portability, was necessary because SMILE was designed to be a platform independent library. If any part of the method implementation would end up in SMILE, it would have to comply to the same requirements as SMILE. To be sure that these requirements are met the library used for the LP solver should also be platform independent. After some evaluation of the two LP solvers, it was concluded that both solvers could be used in a platform independent manner.

There was chosen to use the CLP solver. The reason for this choice was that the CLP license would fit the current license view of the decision systems lab better than the LGPL. Under the LGPL it is possible that source code licensed under the LGPL is re-licensed under the GPL. Although this does not cause all current code licensed under LGPL to be now licensed under GPL, just the version of the code where the license was changed and its derivatives, but if newer versions of the code would now only appear under the GPL license it could no longer be used in a proprietary program. The CPL does not have such a construction. As long as the DSL license mentions that is uses the CLP library and that the CPL applies to the library, it can be included in GeNIe and SMILE without any problem.

5.4.2 Integrating the Library

After deciding to use the CLP library it was examined more closely to determine how it should be integrated into the implementation of the method. An approach was chosen to feed the solver the data, execute the solver and then to retrieve the data from the solver. The library offered several methods to initialise the solver with the input data. It could read the data from different (standard) file types used by other LP solvers, or using the library's API to create the necessary matrix directly in the format used by the solver. Since the way files are handled differs per platform, it was decided to use classes and functions from the library's API to feed the input data to the solver. This would automatically mean that the output generated by the solver would also be provided by objects from the API.

After the integration of the CLP library in the implementation of the method, the process to execute the LP preprocessing step is as follows:

1. The input data for the LP solver is extracted from the linear constraints.

2. The data is put into a matrix by using functions and objects from the API.

3. The solver is called to process the matrix.

4. The result is retrieved from the solver and is put into the appropriate form to be used by the method.

5.5 Implementing Sampling

The sampling process is the main part of the method and most of the execution time will be spent sampling. Sampling is the area where improvements will have the most influence on the speed of the method. First, the most

basic version of sampling was implemented to see where optimizations could be applied.

This basic sample generator would randomly pick a value between 0 and 1 for every constituent variable and would then present the sample to the constraints. If the sample complies with all the constraints it would be saved, otherwise it would be discarded.

It became clear that the axiomatic requirement that all constituent variables must sum to 1 caused almost all samples to be rejected, and that if the sample generation process is optimized to satisfy the axiom, the ratio of rejected samples would decrease. Different approaches have been tried, some examples are:

- Pick values between 0 and 1 for all constituents c_i but the last constituent c_n, and calculate c_n by using $c_n = 1 - \sum c_i$.
- Pick a value for c_1 between 0 and 1, and Pick values for $c_2 \dots c_n$ by picking a value for c_{i+1} between 0 and $1 - \sum_{j=1}^{i} c_j$.
- Pick values between 0 and 1 for all constituents and divide the value of each constituent by the sum of all constituent values.
- Using the constraint consensus algorithm to move the sample to the plain where the constituents do sum to 1 (see Section 3.5.1).
- Use the method described by (Caprile, 2001) to generate the samples.
- Use a Dirichlet distribution to generate samples (Section 3.5.2).

After some testing of the different variants the standard sample bounds (0 and 1) were replaced by the results from the LP preprocessing step. This did not have a big influence on the sampling efficiency. Eventually the variants using the constraint consensus algorithm, the modified Caprile approach, and the Dirichlet distribution remained. The other approaches were deemed to inefficient to use for the method. In the end the sample generator using the truncated Dirichlet distribution was implemented for the system.

To be able to deal with conflicting constraints (Section 3.6), in the final implementation heuristic were devised and implemented to help detect the conflicts. Discarded samples are saved until a sample is accepted or 1 million samples are rejected. Once the million samples are collected, the majority heuristic (Figure 3.12) and the constraint effectiveness heuristic (Figure 3.13) are run om the samples to try to advise the user which constraint to change or to delete.

5.6 Calculating Results

Implementing the functions for calculating samples for all the CPT entries of all the family networks was straightforward. The functions were implemented in a bottom-up fashion. First, at the lowest level, functions were

written to calculate a single sample. At the next level functions were needed that were able to calculate all samples for a CPT entry. Moving up to the next level, functions were written for calculating all the samples for all CPT entries of a node. The functions for the next level would ensure that all the samples for all nodes of a family network would be calculated. Finally, at the top level, functions were written that would calculate the samples for all the FNs.

With some advice from Tomek Sowinski, DSL's lead programmer, a datastructure was designed that holds all the calculated samples. This structure is sent to GeNIe as the result of the method. Using the datastructure GeNIe will show the data to the user by creating histograms from the samples. There was chosen to sent the raw samples to GeNIe so that it would be possible to allow variable bin sizes for the histograms. The current implementation of histograms in GeNIe supports variable bin sizes and this can be used by the user to get a clearer view of the results.

5.7 User Interface

Most of the work focused on implementing the method itself. A console application was created that had a simple command line interface, and that outputted some development and debugging information to the screen. This application was used to develop and test the implementation of the method. Designing a graphical user interface, despite the important role it has in creating a successful application, was not the biggest priority during the implementation and design process. The implementation of the user interface in GeNIe is to be done by Tomek Sowinski. Tomek is responsible for programming and maintaining GeNIe and will be able to implemented the user interface and integrate the method code much faster. As with designing the GUI, further implementation will be future work.

5.8 Testing

The implementation of the method has been tested during the different implementation phases. Testing was done to ensure that the different parts were implemented correctly. During development of new code, simple test cases would be devised to see if the added code functioned correctly.

Some examples of tests conducted during implementation of C++ code for different phases are:

- Input data phase:
  - To test if the supplied Bayesian network would decomposed correctly into the collection of family networks some extra code was added that would write the newly created FNs to a .xdsl file.

Using GeNIe the different FNs could be checked to see if all the nodes and the links between the nodes that should be in the respective FN were actually put there by the used algorithm.

- During the implementation of the expression tree, the parsing of probabilistic statements and the application of symbolic math on the generated expression trees was test by creating a number of test statements and expressions. These statements and expressions were presented as input to the program. The result of the parsing and tree manipulations were shown as output on the screen. When differences occurred, breakpoints were set in the code and step-by-step execution of code was used to determine the cause of the differences.

- Linear programming phase:

  - The implementation of the LP procedure that calculates probability bounds for the constituent variables has been tested by comparing the results to results generated by a stand alone LP solver. A free LP program, Qsopt, was downloaded to be used as a reference. The linear programs that needed to be solved for the method were inputted in the program and the results were manually processed using the algorithm that was designed for the method. Results were compared to see if there were any differences.

- Sampling phase

  - To determine which variant should be used to generate samples (Section 5.5), all variants were tested. The primary criterium that was checked was if the different variants could generate samples that are uniformly distributed over a simplex. This experiment is described in Section 6.1.2.
  - The heuristics developed for detecting conflicting constraints were tested by creating a simple Bayesian network (the network from (Druzdzel & van der Gaag, 1995) was used) and several sets of constraints were there was at least one conflict between two or more constraints. The results of the heuristics were evaluated to see if they were able to detect the conflicting constraints.

After finishing the complete implementation, it was necessary to test if the whole implementation would also function correctly. To test the implementation the example from (Druzdzel & van der Gaag, 1995) was used and the results produced by the implementation were compared with the results from the paper. This experiment and its results and consequenses are described in Section 6.1.1.

Chapter 6

Empirical Evaluation

After implementation of the method it was necessary to evaluate if it would perform as expected. Since the method had never been implemented before, there were no programs also based on (Druzdzel & van der Gaag, 1995) that the implementation could be compared with. Druzdzel and van der Gaag did create a prototype, but it was only created for illustrative purposes. The prototype implementation was straightforward without any algorithmic optimization and it could only generate samples for the example described in the paper.

The results generated by the method, histograms for all the CPT entries, cannot be easily compared. A possible experiment could be to take a completely defined BN and to define several probabilistic statement that describe the BN. Then the generated histograms could be compared to the CPT entries to see if the histograms would accurately predict that the values of the CPT entries are likely to be "correct." if the statements are too general, a large number of probability distributions will satisfy all the constraints and this will cause the histograms to be less accurate in specifying which values that will likely be correct. On the other hand if the statements are too constraining, it may take a very long time for the method to find enough valid samples. In reality complete or near-complete information is never available so a set of statements that will describe the desired probability distribution very closely will most likely not be available.

Comparing results from the implementation of the method with the results from (Druzdzel & van der Gaag, 1995) is almost impossible. The only available information of the experiment done by Druzdzel and van der Gaag are the illustrations of two histograms in the paper. No data or any information about the prototype developed by dr. van der Gaag was available anymore. The prototype and the raw data were discarded after generating the results for the paper. Its implementation was deemed too inefficient to continue work on it, and it was decided that it should be reimplemented when research would be continued on the method.

The lack of results from the prototype makes it harder to interpret the results generated by the implementation of the method. With no data available the data needs to be interpreted by examining if the histograms fit the intuitive meaning of the different CPT entries.

An interesting opportunity for comparison is how the method generates samples. Different approaches can be used to generate the samples. Some are more efficient than others, and some will perform better. Preferably, when no extra information is available, the probability hyperspace is sampled uniformly so the whole space is searched equally to find samples that satisfy all the constraints. When more information comes available it should be possible to use this information to refine the sample generator.

6.1 Experiments

Two experiments have been defined. The first experiment examines a run of the implementation using the Bayesian network and the set of constraints from the example of (Druzdzel & van der Gaag, 1995). During the execution of the experiment it became clear that an error in the probability statements provided by the paper would make it impossible to generate any results. The cause was found and a new experiment was defined to replace the original.

The second experiment compares different approaches for generating samples of probability distributions. It was examined how well the different approaches could generate samples that were uniformly distributed over a 2-simplex. Approaches that performed equally well, were subjected to further investigation.

6.1.1 Experiment 1: Implementation Run

Experiment 1.1: HIV Network

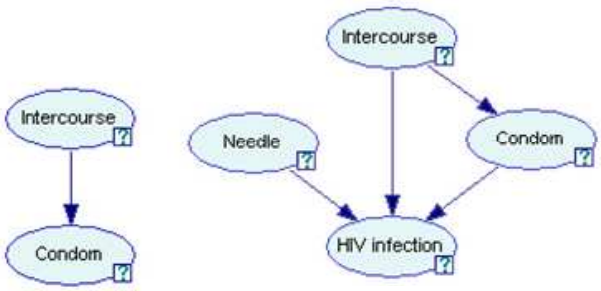
Goal The goal of this experiment is to compare results from the created implementation of the method and the prototype used in (Druzdzel & van der Gaag, 1995).

Design The experiment uses the example presented in the paper. The HIV network and the probability statements mentioned in the paper are used as input for the implementation of the method. The histograms that are outputted are compared with the ones shown in the paper.

The input data for the method was:

- The HIV network (Figure 3.1),
- Probabilistic constraints:
  - $P(i \mid c) = 1$,
  - $P(i) > P(n)$,
  - $P(h \mid n) > P(h \mid i)$,
  - $0.1 \leq P(n \mid h) \leq 0.25$.

Results The data was inputted in the method and it was executed. Two family networks were identified and the method was run on both. The two discovered family networks are shown in Figure 6.1. The constraints



(a) FN 1 (b) FN 2

Figure 6.1: Family networks discovered by the method

generated for the family network from Figure 6.1(a) are (see also Appendix A.1):

- $P(i \mid c) = 1$:
  - $-x_2 = 0$,
  - $x_2 + x_3 > 0$.

Extra constraints were added by the method to ensure that the CPT entries would exist.

- $P(\neg i) > 0$:
  - $x_0 + x_2 > 0$.
- $P(i) > 0$:

- $x_1 + x_3 > 0$.

These constraints are here to ensure that every CPT entry exists and that the situation

$$P(X|Y) = \frac{P(XY)}{P(Y)} = \frac{P(XY)}{0} = \infty$$

cannot occur. For the FN in Figure 6.1(b) several constraints were generated. One example is shown, the rest can be seen in Appendix A.2:

- $P(h|n) > P(h|i)$:
  - $x_2x_9 + x_3x_9 + x_6x_9 + x_7x_9 + x_2x_{11} + x_6x_{11}$
    $+ x_2x_{13} + x_3x_{13} + x_6x_{13} + x_7x_{13} + x_2x_{15} + x_6x_{15}$
    $- x_1x_{10} - x_3x_{10} - x_5x_{10} - x_7x_{10} - x_1x_{11} - x_5x_{11}$
    $- x_1x_{14} - x_3x_{14} - x_5x_{14} - x_7x_{14} - x_1x_{15} - x_5x_{15} > 0$,
  - $x_1 + x_3 + x_5 + x_7 + x_9 + x_{11} + x_{13} + x_{15} > 0$,
  - $x_2 + x_3 + x_6 + x_7 + x_{10} + x_{11} + x_{14} + x_{15} > 0$.

Extra constraints were again added to ensure that all the CPT entries would exist. One example is

$$P(\neg n \neg i \neg c) > 0 : \quad x_0 + x_8 > 0. \tag{6.1}$$

Problems arose when the method started to try to generate samples for the family network shown in Figure 6.1(b). After a long period of time no valid samples were found for the FN. The problem was pinpointed to the probability statement $P(i|c) = 1$. This statement is an equality constraint that would be translated into multiple constraints. It was translated into the following expressions

$$-x_4 - x_5 - x_{12} - x_{13} = 0\ ,$$
$$x_4 + x_5 + x_6 + x_7 + x_{12} + x_{13} + x_{14} + x_{15} > 0\ .$$

The first expression causes a conflict with the extra added expressions $x_4 + x_{12} > 0$ and $x_5 + x_{13} > 0$. The statement $P(i|c) = 1$ causes the probabilities $P(\neg n \neg ic)$ and $P(n \neg ic)$ to become 0, which in turn causes the CPT entries $P(h|\neg n \neg ic)$, $P(\neg h|\neg n \neg ic)$, $P(h|n \neg ic)$, and $P(\neg h|n \neg ic)$ to become undefined.

This means even without the presence of the extra added constraints the method would not be able to generate samples for all the CPT entries. It would be terminated after a division by 0 was detected when calculating samples for the above mentioned entries.

Conclusion The example from (Druzdzel & van der Gaag, 1995) was abandoned and a new example was found to perform a similar experiment on.

Experiment 1.2 Alarm System Network

Goal The goal for the new experiment is to compare the influence of two different probability statement sets on the resulting histograms.

Design The new Bayesian network used in the experiment is the alarm system network discussed in Section 2.3.2. The network is shown in Figure 2.1. Two sets of constraints were created for this network. The first set of constraints can be considered as a sort of control group. For half of the CPT entries of the BN probability statements were created that would be in the form

$$0.9x \leq P(\cdot) \leq 1.1x,$$

where $P(\cdot)$ is the probability of a CPT entry and x is the real value of the entry. These statements create an interval around the real value, where the upper and lower bound differ 10% from the real value. The created statements for the "strict" statement set are:

- $0.0009 \leq P(b) \leq 0.0011$,
- $0.0018 \leq P(e) \leq 0.0022$,
- $0.0009 \leq P(a \,|\neg b, \neg e) \leq 0.0011$,
- $0.846 \leq P(a \,|b, \neg e) \leq 1.0$,
- $0.261 \leq P(a \,|\neg b, e) \leq 0.319$,
- $0.855 \leq P(a \,|b, e) \leq 1.0$,
- $0.045 \leq P(j \,|\neg a) \leq 0.055$,
- $0.81 \leq P(j \,|a) \leq 0.99$,
- $0.009 \leq P(m \,|\neg a) \leq 0.011$,
- $0.63 \leq P(m \,|a) \leq 0.77$.

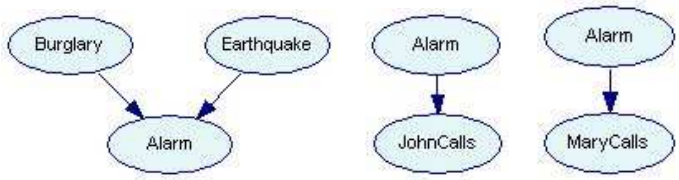
When a calculated bound would have a value outside of the valid range of $[0, 1]$ the bound was clipped at the appropriate value.

The second set of probability statements was created in a manner that is more likely to be encountered in real life. Normally an expert would be necessary to create the statements and he or she would probably be able to translate rules of thumb into probability statements. For the experiment probability statements have been created that describe the CPT entries of the BN loosely. The statements are still based on the true values of the CPTs, but no intervals have been specified and values in the statements have been chosen to be less restrictive.

The created statements for the "loose" statement set are:

- $P(e) < 0.003$,
- $P(e) > P(b)$,
- $P(b) < 0.002$,
- $P(\neg a \,|\neg b, \neg e) > 0.95$,
- $P(a \,|b, e) > 0.9$,
- $P(a \,|\neg b, e) < 0.4$,
- $P(\neg a \,|b, \neg e) < 0.09$,
- $P(\neg j \,|\neg a) > 0.9$,
- $P(\neg j \,|a) < 0.15$,
- $P(m \,|\neg a) < 0.02$,
- $P(\neg m \,|a) < 0.3$.

Results Both the probability statement sets have been inputted in the method together with the alarm system BN. Three different family networks were identified. The networks are shown in Figure 6.2.



(a) FN 1 (b) FN 2 (c) FN 3

Figure 6.2: Identified family networks for the alarm system network

For both sets of probability statements the method has generated constraints that were used for the sampling process. The probability statements are divided over the FNs. Only statements that are relevant for a FN are translated into constraints for the sampling process. The generated constraints for the "strict" statement set have been put in Appendix B.1 and Appendix C.1 contains the constraints for the "loose" statement set. After these constraints were generated the Linear Programming process described

in Section 3.4 was run using the linear constraints. All the constraints were linear, so all could be used to determine probability bounds for all the family networks. The generated probability bounds for the different FNs of the experiments are put in Appendix B.2 (strict set) and Appendix C.2 (loose set). As an example the probability bounds from the first FN of the strict experiment are shown below.

$$\begin{array}{lll} 0.995604 & \leq x_0 \leq & 0.997301, \\ 0 & \leq x_1 \leq & 0.000169323, \\ 0.00047704 & \leq x_2 \leq & 0.00162543, \\ 0 & \leq x_3 \leq & 0.000159428, \\ 0.00089703 & \leq x_4 \leq & 0.00109802, \\ 4.23 \cdot 10^{-7} & \leq x_5 \leq & 0.001099, \\ 0.000182831 & \leq x_6 \leq & 0.00070164, \\ 4.275 \cdot 10^{-7} & \leq x_7 \leq & 0.0010995. \end{array}$$

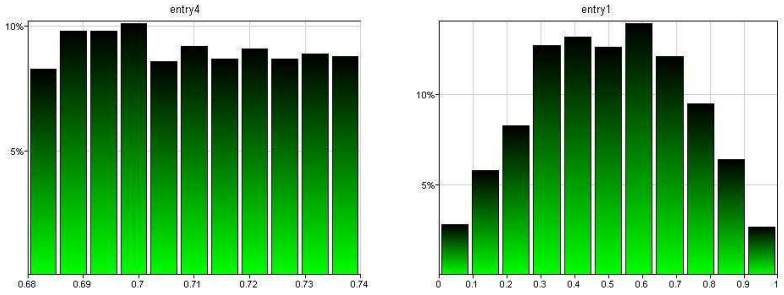
The probability bounds improve the efficiency of the method enormously. Only a very small sub volume of the whole hypercube needs to be sampled now. In the case of FN 1 the probability bounds create a new hypercube that has a volume of only $6.6 \cdot 10^{-27}$ times the whole hypercube volume. And only the intersection of the 7-simplex with this hypercube will be searched by the sample generator.

After the generation of the constraints and the calculation of the probability bounds the sampling process is started. After the method has found 1000 samples that satisfy all the constraints, the samples are used to calculate samples for the CPT entries of the FNs. The result of these calculations can be shown as histograms for the different CPT entries. The complete collection of generated histograms of the probability statement sets have been put in Appendix B.3 and Appendix C.3.

Conclusion Some general observations can be made about a large number of the generated histograms. The shape of most of the histograms is similar. The histograms resemble a uniform distribution. One example is Figure 6.3(a), a histogram generated from the "strict" probability statement set for the Alarm node.

A likely cause for this shape to be present in so many histograms is the use of uniform sampling on the simplex. The generated samples of the joint probability distributions are all as likely to occur, so when calculating samples for the CPTs these samples will also show the same characteristics.

Not all the histograms of the two experiments have a uniform shape. In the smaller family networks consisting out of the nodes Alarm and JohnCalls or Alarm and MaryCalls, the histograms created for the Alarm node did not have a uniform shape. An example is shown in figure 6.3(b). The histogram has a shape that resembles a normal distribution. This resemblance can be



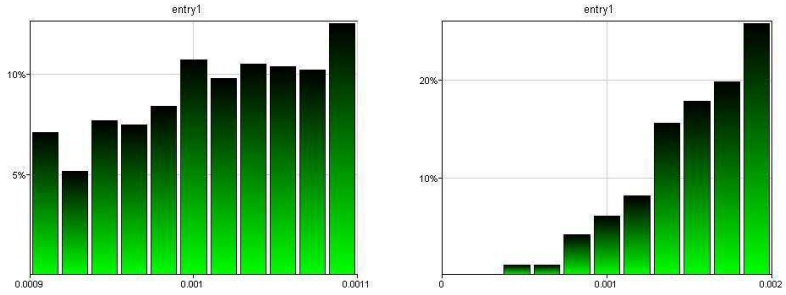
(a) Histogram for entry $P(\neg a \mid \neg b, e)$ (b) Histogram of $P(a)$ with a nonuniform shape

Figure 6.3: Example histograms

explained by examining the constraints used for these family networks. In both experiments constraints were created specifically for CPT entries of the complete BN. No constraints were created for the node Alarm for the specific situation that it would be a parent in one of the family networks. Now the histogram is the result of the calculations of the samples for the FNs which were constrained only for the JohnCalls and MaryCalls nodes. Since the samples and the histograms for the nodes JohnCalls and MaryCalls are uniformly distributed, and calculating samples for the CPT entries of the Alarm node basically means summing entries, this will cause the distribution of the samples for the CPT entries of the Alarm node to resemble a normal distribution. This is due to the central limit theorem (Dekking et al., 2004).

Other examples exist for nonuniform histogram shapes. These differences resulted from using different constraints. The "strict" and "loose" probability statement sets have resulted in different histograms for the same entries. Mostly the differences are in respect to upper and lower bounds, but in a few situations the shape also was different. An example is given in Figure 6.4. The "strict" version of the histogram is much more precise than its "loose" counterpart. This is due to the constraint $0.0009 \leq P(b) \leq 0.0011$. This is the only constraint in the "strict" statement set that has a direct influence on the the entry $P(b)$. The other statements describing the same family network will have an influence, but not as much as this one. The "loose" statement set has 3 constraints that have a direct influence on $P(b)$:

- $P(b) < 0.002$,
- $P(e) < 0.003$,
- $P(e) > P(b)$.



(a) $P(b)$, "strict" statements (b) $P(b)$, "loose" statements

Figure 6.4: Different histograms resulting from the difference in the statement sets

Since these constraints are less precise than the constraints from the "strict" set this allows for more samples to be valid. Which results in histograms that will vary over a larger range. In this case the exact shape is not easy to explain, it also depends on the other constraints used to sample the family network. And since the "loose" constraints allow for more samples to be valid, it could be possible that 1000 samples is not enough in larger dimensional spaces to get a good distribution of the samples in the constrained volume that satisfies all constraints.

An estimate can be given of the size of the constrained volume created by the two probability statement sets. The exact volumes are hard to calculate so they are approximated by a n-dimensional bounding box. This bounding box is created by the probability bounds calculated by the linear programming process. The lower bounds are subtracted from the upper bounds to get the size of the edges of the box. By multiplying the edges the volume of the bounding box is calculated. The volumes for the boxes are:

- "Strict" set: $6,62355 \cdot 10^{-27}$,
- "Loose" set: $2,27375 \cdot 10^{-21}$.

This means that the volume of the sample space of the "loose" probability statement set is a factor $343283,1356$ larger than the volume of the "strict" set. A supporting fact for the hypothesis that the size of the sample space has an influence on the histograms is that the nodes Earthquake and Burglary are a priori nodes. The nodes are not conditioned on other variables and only have 2 entries: true and false. Calculating a sample for one of these entries means summing half of the entries of a raw sample.

The exact influence of the volume of the sample space is unknown, and the influence of finding more samples on the shape of the histograms has

not been investigated any further. For now the differences in the histogram shapes are attributed to the differences in the relevant probability statements.

6.1.2 Experiment 2: Sampling Approaches

The basis for an efficient sampling process is to ensure that the generated samples at least are valid probability distributions. Otherwise samples will be discarded even before they are evaluated by the constraints provided by the expert. Another desired property of the sample generator is that with the absence of any extra information it should uniformly generate probability distributions, i.e. all distributions should be generated with the same probability. This ensures that the whole probability space is searched.

Goal The goal of this experiment is to compare different approaches for generating samples, and to determine which of the approaches should be implemented.

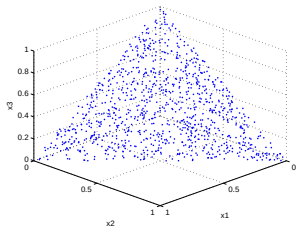
Design Criteria for comparison are the ability to generate samples that are uniformly distributed over a simplex and the ease of adapting the approach to include an importance sampling scheme.

In total there were 8 different approaches tried for generating samples. One was abandoned very quickly because it did not allow for sampling the probability plain directly. It used rejection sampling and was very inefficient. The 7 remaining approaches would allow for sampling on the probability plain without the use of rejection sampling. The question remained if the approaches would allow for uniform sampling on the plain. The remaining approaches were tested by generating 1000 samples of discrete probability distributions in a 3 dimensional space. The remaining approaches were:

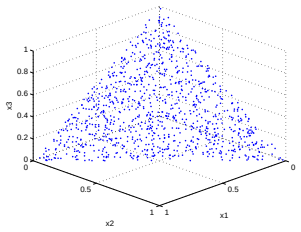
- Use a truncated Dirichlet distribution.
- Use a Dirichlet distribution to generate samples.
- Use the method described by (Caprile, 2001) to generate the samples.
- (Caprile, 2001), but with shuffling of the constituents.
- Uniformly pick values between 0 and 1 for all constituents and divide the value of each constituent by the sum of all constituent values.
- Pick a value for c_1 between 0 and 1, and Pick values for $c_2 \ldots c_n$ by picking a value for c_{i+1} between 0 and $1 - \sum_{j=1}^{i} c_j$ (Fraction Approach).
- Using the constraint consensus algorithm to move the sample to the plain where the constituents do sum to 1.

If after comparison of the different approaches there would remain multiple candidates for implementation, further comparison would be performed by generating samples on a 15-simplex in a 16 dimensional space. For each of the remaining approaches 1000 and 100000 samples are generated and 3D scatter plots of the first three axes are compared. Any differences are investigated by examining the sampling process of the different approaches more closely.

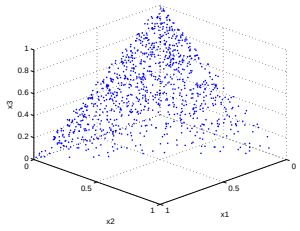
Results The samples generated by the different sampling approaches were put in 3D scatter plots to visualise the distribution of the samples over the 2-simplex. All the scatter plots are gathered in Figure 6.5.



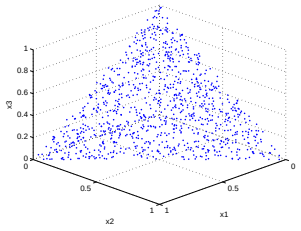
(a) Truncated Dirichlet Distribution



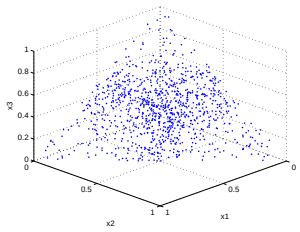
(b) Dirichlet Distribution



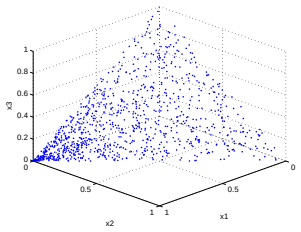
(c) Caprile Method



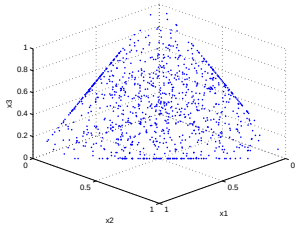
(d) Caprile Method (Shuffling)



(e) Divide by sum Approach



(f) Fraction Approach



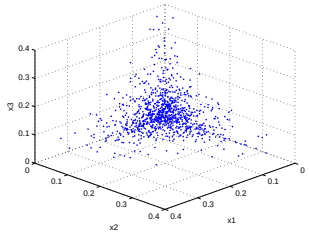
(g) Constraint Consensus

Figure 6.5: Results from the sample approach experiment

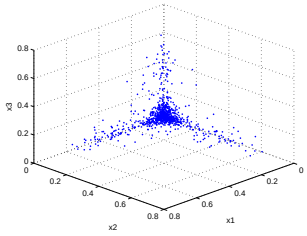
By examining the scatter plots 4 approaches could be eliminated immediately. The fraction approach, the divide by sum approach, the approach using the constraint consensus algorithm, and the method from (Caprile, 2001) exactly as it was described in the paper, all do not result in a uniform distribution over the 2-simplex.

The 3 remaining approaches, the Dirichlet variants and the modified version of (Caprile, 2001), all allow for a uniform distribution of samples over the simplex. Evidently this is also claimed in the literature describing these approaches. To select an approach for implementation in the method further investigation is necessary.

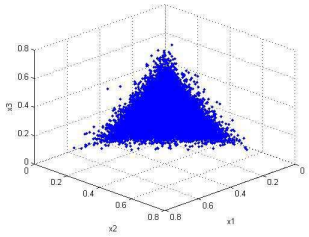
The requirement that it should be possible to sample between probability bounds, reduces the candidates even further. Now only the truncated Dirichlet distribution and the modified approach from (Caprile, 2001) remain, since the original implementation of the Dirichlet distribution as described in Figure 3.6 does not allow for sampling between bounds.



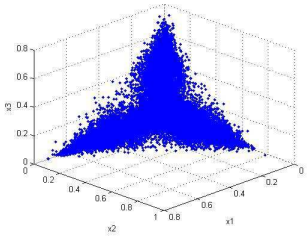
(a) Truncated Dirichlet Distribution 1000 samples



(b) Caprile Method (Shuffling) 1000 samples



(c) Truncated Dirichlet Distribution 100000 samples



(d) Caprile Method (Shuffling) 100000 samples

Figure 6.6: Results from further comparison

To determine if the remaining approaches were equivalent in uniformly generating samples on simplices, experiments were performed using higher dimensional simplices. For both the approaches it is claimed that they allow

for uniform sampling on general n-simplices. Two experiments were performed. In a 16 dimensional space samples were generated on a 15-simplex. For both approaches 1000 and 100000 samples were generated. The results of the experiments, 3D scatter plots of the the first 3 axes of the 16 dimensional space, are shown in Figure These results show that the approaches are not equivalent. The scatter plots show that the distributions of the samples generated by the approaches are different. From these observations two hypothesis can be formulated. Either only one of the two approaches can exactly generate samples with a uniform distribution over a n-simplex, or one of the two is a better approximation of the true uniform distribution over a n-simplex.

When the dimension of the sample space increases the difference between the two approaches becomes larger. This suggest that when generating samples the dimension of the sample space has a direct influence on the sample process.

The approaches calculate their samples in a similar manner. For $N-1$ entries of the sample vectors values are generated using a marginal distribution dependent on the previous entries and the value for the last entry is calculated by $x_N = 1 - \sum_{i=1}^{N-1} x_i$. For the truncated Dirichlet distribution the process to generate samples uniformly distributed over a simplex reduces to the algorithm shown in Figure 6.7.

Input:

- a parameter vector $\gamma = (1_1, \ldots, 1_N)^T$,

1. Set $\xi_k = 0$
2. Set $\eta_k = 1$
3. For $k = N-1, k > 0, k = k-1$
    1. Generate a random value u_k from a uniform $U[0,1]$ distribution.
    2. Calculate $x_k = \left(1 - \sum_{j=k+1}^{N-1} x_j\right) F_k^{-1}(u_k)$, where $F_k^{-1}(u_k)$ is a standard beta distribution with $\alpha = 1$ and $\beta = k$
4. Calculate $x_N = 1 - \sum_{i=1}^{N-1} x_i$
5. Output vector $X = (x_1, \ldots, x_N)^T$

Figure 6.7: TDD Sampling algorithm specifically for generating uniform samples

The algorithm for generating samples using the method described in (Caprile, 2001) is shown in Figure 6.8. The last step was added to improve

1. set $r_1 = 1$
2. set $j = 1$
3. until $j = N - 1$
    1. Randomly extract x_j from $[0, r_j]$ according to the marginal distribution of x_j over the simplex S^{N-1}, given outcomes $X_1 = x_1, \ldots, X_{j-1} = x_{j-1}$, that is according to:

$$F_C(x) = P(X_j = x \,|X_1 = x_1, \ldots, X_{j-1} = x_{j-1})$$

    2. Set $r_{j+1} = r_j - x_j$
    3. Set $j = j + 1$
4. Set $x_N = r_N$
5. Output vector $X = (x_1, \ldots, x_N)^T$
6. Shuffle entries of the vector so next time entries are processed in a different order.

Figure 6.8: Generate samples according to the method described by Caprile

the results of the algorithm. Without it, even in a 3 dimensional space the distribution of the samples would not be uniform. The problem with this correction is that the randomness of the approach now also depends on the randomness of the shuffle function. This is one reason to not use this approach in the final implementation. The marginal distribution function necessary by the Caprile algorithm is easy to calculate. The distribution and its inverse are:

$$F_C(x_j) = 1 - \left(\frac{r_j - x_j}{r_j}\right)^{N-1}, \tag{6.2}$$

$$F_C^{-1}(u) = r_j \left(1 - (1-u)^{\frac{1}{N-1}}\right). \tag{6.3}$$

The factor r_j used in the Caprile algorithm is equal to the expression $1 - \sum_{i=1}^{N-1} x_i$ used in the truncated Dirichlet algorithm. When the formulas for calculating an entry of the sample vector are compared the similarities and

the differences become clear:

$$x_k = \left(1 - \sum_{j=k+1}^{N-1} x_j\right) Beta\left(\alpha = 1, \beta = k\right) \ , \tag{6.4}$$

versus

$$x_k = r_k \left(1 - \left(1 - U\left[0,1\right]\right)^{\frac{1}{N-1}}\right) \ . \tag{6.5}$$

What the two have in common is that both can be calculated in a similar matter. A random value between 0 and 1 is generated using a random distribution and the value is then scaled to fit the sampled value into the remaining available "space". With "space" the remaining free amount in $[0, 1]$ is meant that will allow all the entries to sum to 1.

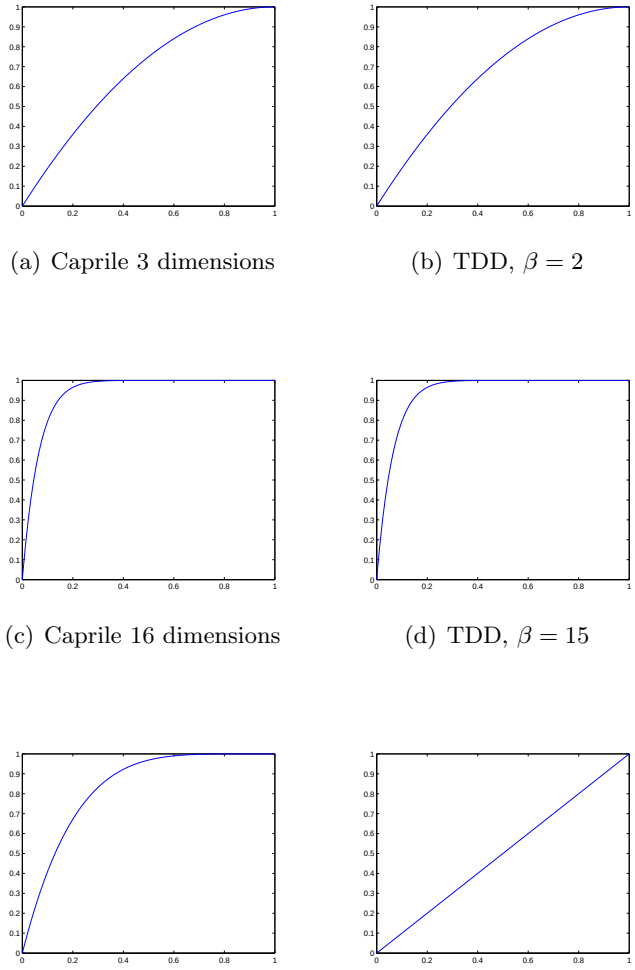
The difference between the two formulas is that the "distribution part" of the formula used by the Dirichlet algorithm changes per entry of the sample vector. The "distribution part" of the formula used by the Caprile algorithm on the other hand does not change during the generation of a sample and only depends on the total number of entries of the sample vector.

When the dimension of the sample space starts to increase this will have an effect on the values of the entries of the sample vectors generated by the two algorithms. For both algorithms this means that the values for the entries will become smaller. This is visible in the cumulative distribution functions of the approaches. Figure 6.9 shows different cumulative distribution functions for different situations. Figure 6.9(a) shows the cdf for the Caprile approach for sampling in a 3 dimensional space. It closely resembles the first cdf used by the dirichlet approach (Figure 6.9(b)) when it is also sampling in a 3 dimensional space.

In the 3 dimensional situation both sample 2 entries from probability distributions and calculate the last entry by subtracting the sum of the 2 samples from 1. The Caprile approach draws samples twice from the distribution in Figure 6.9(a), but the second time the sample from the distribution is scaled to $[0, 1 - x_1]$. The TDD approach first samples from the beta distribution illustrated in Figure 6.9(b). For the next entry a sample is drawn from another beta distribution, the one illustrated in Figure 6.9(f). This sample is also scaled to $[0, 1 - x_1]$. As shown in Equation 6.4 the parameter β of the beta distribution depends on the entry to be calculated. Also a specific sampling order has been defined in (Fang et al., 2000). First entry $N - 1$ must be sampled followed by $N - 2, N - 3, \ldots, 1$, and finally entry N can be calculated.

Sampling in a 16 dimensional space is done in a similar manner. The Caprile approach samples 15 times from the distribution in Figure 6.9(c), each sampled is scaled accordingly, and the last entry is calculated by subtracting the sum of the sampled entries from 1. The truncated Dirichlet distribution first samples from a beta distribution with parameter $\beta = 15$

(Figure 6.9(d)). The next entry will be a scaled sample from a beta distribution with parameter $\beta = 14$. This process continues until the entry x_1 has been given a value from the beta distribution with parameter $\beta = 1$ (Figure 6.9(f)). Figure 6.9(e) shows another step in the changing of the shape of the beta distributions the TDD algorithm samples from while generating a sample vector. With all this information it can be explained why the orig-



(a) Caprile 3 dimensions

(b) TDD, $\beta = 2$

(c) Caprile 16 dimensions

(d) TDD, $\beta = 15$

(e) TDD, $\beta = 5$

(f) TDD, $\beta = 1$

Figure 6.9: Different cumulative distribution functions

inal Caprile algorithm needed to be extended with the shuffling, and why the truncated Dirichlet distribution approach should be preferred above the Caprile approach. When sampling in increasingly higher dimensional spaces the probability distribution function used by the Caprile approach will gen-

erate more and more samples for the entries that have very small values. Since the same distribution function is used for the $N-1$ entries and a scaling factor is applied to let each entry to be sampled from $[0, 1-\sum x_k]$, the values for these $N-1$ entries will become so small that a large amount will remain for entry x_n. This entry will have a marginal probability distribution that is radically different from the rest. It will in fact resemble a normal distribution. An example is shown in Figure 6.10. The figure shows the histogram of entry x_{16}, which was calculated in the manner described above. Shuffling the sample order prevents the entries from getting

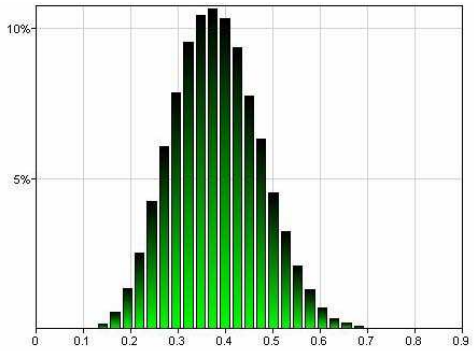


Figure 6.10: Histogram of entry x_{16}

a marginal probability distribution that resembles a normal distribution. These larger values are now distributed over all the entries. This still has an effect on the marginal probability distributions of the entries. This effect is visible in Figure 6.6(d). Most of the samples for an entry will have a very small value, but there are a significant number of samples that have large values.

The Dirchlet approach does not show this behavior because of the changing parameter in the beta distribution used to generate samples for the entries. When moving through the entries, the value for the parameter β decreases. Lower β values make larger sample values (within $[0, 1-\sum x_i]$) more likely which prevents the entry x_n from getting a marginal probability distribution that resembles a normal distribution.

Conclusion Since it is very likely that, during execution of the method, samples will have to be generated in high dimensional spaces, the Dirichlet approach is to be preferred over the (modified) Caprile approach and the others. Another advantage may be that since the Dirichlet distribution has a parameter vector and the caprile approach does not have any parameters, it will most likely be easier to adapt the Dirichlet approach to be able to perform importance sampling.

6.2 Discussion

After performing the experiments some observations have been made that will have an influence on the direction of the further development of the method. Discussed are the use of histograms as output of the method and suggestions on how to improve the use of the histograms. Further discussion focuses on generating samples and the influence of higher dimensional spaces on the sample process.

6.2.1 Use of Histograms

The quality and usefulness of using histograms as outputs will have to be investigated further. The experiments show that the quality of the set of probability statements has a large influence on the quality of the histograms. If the constraints are defined very precisely the generated histograms for the CPT entries will not be very helpful in the process of determining values for the entries.

At the moment all valid samples that satisfy all constraints are treated equally when creating the histogram. A weighting scheme that makes some statements more important than others may make the histograms more precise, but feasibility of this approach will need to be determined.

Another point of attention for the current use of histograms in the implementation is that by calculating the histograms some useful information is lost. To create the histograms 1000 samples of the joint probability distribution of the FN are found by the method and these samples are used to calculated new samples for all the entries of the FN. From each sample values for all CPT entries of the FN are calculated. The histograms show the distribution of the 1000 calculated values from all the samples. What would be helpful in the process of determining values for the CPT entries, using the generated histograms, is the ability to specify a range for a CPT entry withing the user believes the real value of the CPT entry lies. By selecting this range only a subset of the samples will be valid and only these samples should now be used when showing histograms of other CPT entries of the FN to the user. At present it is unclear how the different histograms are connected to each other, but a fact is when a value is chosen for a CPT entry of a node it will influence the possible values the other entries can be assigned.

6.2.2 Sampling and Higher Dimensional Sample Spaces

The sample process has the largest impact on the total performance of the method. Any improvement made here will have a large influence on the necessary computation time and the quality of the results. As mentioned in Section 6.1.2 the minimal requirement for a sample generator is that it should be able to directly generate samples that are valid joint probability distributions, i.e. it should be able to generate samples that lie on the unit n-simplex.

Next it is preferable that the sample generators allows for uniform sampling of the n-simplex. This property guarantees that all possible joint distributions can be generated by the sample generator, which is useful when no extra information is available. When more information does become available this should be used to improve the efficiency of the sample generator.

One example is the use of probability bounds for the different constituents. These bounds, calculated by using linear programming on the linear constraints, can drastically reduce the size of the sample space. This has been demonstrated by the experiments in Section 6.1.1. The usage of the calculated probability bounds ensured that only a very small fraction of the total volume of the sample space was to be searched. It would have been completely unfeasible to generate samples from the constrained volume without these bounds.

Intractability is a problem that never can be completely avoided. The tractability of the method, the ability to produce results in a reasonable amount of time, depends on the quality of the probabilistic statements and the number of variables in a family network. If a FN has a large number of variables, the sample space will have a large number of dimensions. The number of dimensions increases exponentially in the number variables, this definitely has an influence on the running time of the method.

In Section 6.1.2 it was discovered that the uniform sampling performance of the modified Caprile approach decreases when the number of dimensions of the sample space starts to increase. It is not completely certain that the truncated Dirichlet approach does not have the same problems when the number of dimensions increases sufficiently. Although theoretically speaking the Dirichlet distribution should be able to represent a uniform distribution over a simplex in any number of dimensions, when the parameter vector $\mathbf{1}$ is entered the density function reduces to a constant, it is possible that the implementation of the sample generator might be limited in performance by factors as variable precision.

In general it is a good idea to limit the number of dimensions of the sample space. An acceptable maximum number of dimensions might be somewhere between 64 or 128 dimensions. If this is translated back to a number of variables this would that a node in a BN can have a maximum

of 5 or 6 parents. Here it is assumed that the nodes are binary, i.e. they can only be assigned 2 different values (most of the time true and false). When multivalued variables are used the maximum number of parents will decrease further.

The probability statements have a large influence on the running time of the method. The statements create the constrained volume where valid samples are to be found. If the samples are not very restricting the volume will be large and it will be easier to find valid samples. If the samples are constricting or even conflicting it can take a very long time to find a valid sample or to conclude that no valid samples will ever be found. This part of the sample process is simply a form of rejection sampling, no real improvements can be made in this area.

By setting a maximum number of samples it can be prevented that the method will keep running forever. However, until it can be determined without error that a set of constraints has conflicts, one can never be sure if there actually are conflicting constraints that prevent the generation of valid samples.

Chapter 7

Conclusion

This chapter concludes this thesis, the goals set in Section 1.3 are reviewed to see how many have been reached. A summary of contributions is given, followed by opportunities for future work. This chapter ends with the last concluding remarks.

7.1 Goals

When work begun goals were set for the assignment regarding research and implementation. Al the goals are discussed one-by-one to show in which extend the goals have been reached.

7.1.1 Research

- *Design an efficient sampling scheme for the method.*
  Finding an efficient way to generate samples was difficult. In the end the method works with rejection sampling and if the constraints describe a very small feasible area a lot of samples will still be rejected. But a lot of progress has been made in trying to reduce the number of rejected samples as much as possible. The truncated Dirichlet distribution now used to generate samples, in combination with the linear programming process, allows for finding samples in very small feasible areas that would have been impossible to find otherwise.

- *Design a method for conflict detection and conflict resolution for constraints.*
  The elicitation method only works if the constraint sets do not have any conflicts. Otherwise it will be impossible for valid samples to appear and the method will run indefinitely trying to find valid samples. In the worst case scenario there are nonlinear constraints in the constraint set, thus it is important to ensure that any conflict detection

and resolution method can handle this scenario. The problem of conflict detection and conflict resolution with nonlinear constraints is an open problem, and it is not clear if a solution exist. The relevant literature (see Section 3.6.1) describes different heuristics that try to pin-point possible conflicts in constraint sets.

Two heuristics have been implemented, one was due to Chinneck (2002), and the other was developed by the author. Both heuristics show promise, but must be researched further to see how effective they really are.

- *Develop an importance sampling scheme for deriving 2^{nd} order probability distributions over the constituent probabilities.*
  An importance sampling scheme has been developed that is based on the Dirichlet distribution. The basic idea of the scheme is to use collected valid samples to estimate a new parameter vector for the Dirichlet distribution. The new parameter vector will influence the density function of the Dirichlet distribution which will make it more likely for samples to be generated in the area where the other valid samples are. The scheme has not been implemented or tested yet.

- *Design a method for merging decomposed family networks into the complete BN as accurately and efficiently as possible.*
  Work has been done on designing a merging method that could merge samples of different CPT entries from a node that appeared in multiple FNs. The idea was based on using weighted averaging, but in the end this approach did not lead to a method that could merge FNs in a mathematically sound way. No other method was found or developed that could accurately merge decomposed family networks into the complete BN. It was chosen to leave this as an open problem and to present the user the results from the generated FNs as output of the method.

- *Design a graphical interface for GeNIe for interactive elicitation of constraints.*
  Most of the available time was spent on implementing the complete method and finding answers for the other research goals that were important for the inner workings of the method. A very basic and preliminary user interface has been designed. It has not been implemented in GeNIe yet. Further design and development of the user interface is suggested as future work.

7.1.2 Implementation

- *Design and implement a data structure for the constraints.*
  An existing datastructure for equation trees from SMILE has been modified and expanded so that it could be used for equations and inequalities. More extensive symbolic mathematic functions have been added to the implementation to allow for checking of linearity.

- *Design and implement a data structure for the constituent probabilities.*
  A datastructure for the constituents has been designed and implemented.

- *Implement a linear programming algorithm for calculating probability intervals over the constituent probabilities.*
  The CLP library, which implements a linear programming solver, has been found and it has been used for the calculation of the probability bounds of the constituents.

- *Improve the efficiency of the method.*
  A lot of work has been done to improve the efficiency of the method. The use of linear programming to determine probability bounds, as proposed in (Druzdzel & van der Gaag, 1995) has been implemented. Work has been done to automatically extract the linear constraints from the constraint set, that was generated during the parsing of the probability statements provided by the expert. The truncated Dirichlet distribution used for generating samples as described in Section 3.5.2, was implemented to use the calculated probability bounds. It can be used in the future in the proposed importance sampling scheme.

- *Test the method and report the results in the thesis.*
  After finishing the implementation experiments were done (Section 6.1.1) to see how the method performed as a whole. The method itself works, but will need further improvement. The experimental results show that the histograms generated by the method are, at the moment, not really helpful when choosing values for CPT entries (Section 6.2.1).

7.2 Summary of Contributions

This section describe the parts of the work done that can be considered as contributions to field.

The method described by (Druzdzel & van der Gaag, 1995) has never been implemented beyond a very simple prototype. The created implementation allows for the automatic processing of the necessary input data into the end results, the histograms for the CPT entries. The implementation shows that the proposed approach by Druzdzel and van der Gaag is usable, but further research will be necessary to make the approach more tractable and more precise.

In the area of sampling a lot of work has been done to make the sampling process as efficient as possible. The main goal was to reduce the number of samples that had to be rejected as much as possible. The biggest achievement in this area was to use combine the truncated Dirichlet distribution proposed by (Fang et al., 2000) with the calculation of probability bounds for the constituents proposed by (Druzdzel & van der Gaag, 1995). The resulting sample generator allows for the much more efficient generation of samples of joint probability distributions. Use of the truncated dirichlet distribution ensures that every sample will be a valid joint probability distribution, and the ability of the distribution to directly sample between bounds without the use of rejection sampling makes it possible to completely exploit the information extracted from the linear constraints provided by the expert.

Further improvement of sampling can most likely be achieved by implementing the importance sampling scheme that was designed. The designed scheme, based on the idea presented in (Cheng & Druzdzel, 2000) that the importance function should be adapted by using samples previously generated by the importance sampling mechanism, adapts the Dirichlet distribution to focus the sampling process on the region in the sample space where the volume that is constrained by the probability statements is believed to be.

Finally, two heuristics have been implemented to allow for some basic conflict detection and conflict resolution. One of the two heuristics was designed by the author. This heuristic, named the Majority Heuristic, allows for a ranking of probability statements that indicates how likely a probability statement is the cause of a (possible) conflict. The heuristic, based on the idea that if a samples satisfies a majority of the constraints, there may possibly be a conflicting constraint in the minority that the sample did not satisfy. Some preliminary tests were done with the heuristic and when there was enough data (rejected samples) available the results looked very promising.

The method proposed by (Druzdzel & van der Gaag, 1995) is meant for the elicitation of probabilities for Bayesian networks, but there is possibly

another use for the method in other fields. Part of the method could be used to generate mixtures for mixture experiments that have to satisfy certain (non)linear constraints. The output for such an application would be the "raw" samples generated by the truncated Dirichlet distribution that satisfy all collected constraints. Mixture experiments are used to determine the optimal proportions of ingredients of a product. This product could be soup, deodorant, steel alloys, or other products that consist out of a mixture of ingredients. The (modified) method could contribute to fields that use mixture experiments by randomly generating possible mixtures that must satisfy a large set of requirements.

7.3 Future Work

When working on the implementation of the method and when doing research to improve it, there was not always time to actually implement or to pursue all ideas. This section summarizes these ideas and the work that still has to be done.

- *Merging of Family Networks*
  One of the goals not currently met was finding a method for the merging of the results of the family network into results for the complete Bayesian network. A method will have to be found that can merge results from nodes of different family networks with a different number of CPT entries into result for the node in the complete BN. The current problem was that no method was found that was mathematically correct. Future research for this problem should focus on the question if a mathematical correct method exist for merging family networks. If it can be proven that such a method does not exist, research should focus on what other type of network decomposition should be used so merging of the results can be done in a mathematically correct manner.

- *Implementing a Graphical User Interface*
  At the moment no graphical user interface has been implemented. Currently the implementation works with a simple console interface and takes .txt files as input for the constraints. An GUI design has been made, but this design will have to be improved and tested with real users before actually integrating the method into GeNIe.

- *Implementing Importance Sampling*
  An importance sampling scheme has been designed, but it has not been implemented yet. It will be necessary to experiment with the proposed algorithm to see how effective it can be in reducing the number of rejected samples. The importance sampling schema relies on the estimation of a new Dirichlet parameter vector using data. Several methods were proposed by (Huang, 2005). It will be necessary to determine which of the proposed estimation techniques works best with the rest of the implementation of the method.

- *Use of Histograms as Output*
  In Section 6.2.1 the current use of histograms was criticised. A research opportunity here is to investigate the current quality of the generated histograms and to see if for example weighting the samples will improve the quality of the histograms. Another proposed idea in Section 6.2.1 is to let choosing a value for one of the CPT entries (by means of bounds) influence all the histograms of all other CPT entries of the family network.

- *Mixture Experiments*
  During the search for a good sample generator, a very short literature study was done regarding mixture experiments. This was done because mixtures have properties that resemble the discrete probability distribution that have to be generated by the method. Both have elements that cannot be negative, and all the elements of both must sum to one. For the design of mixture experiments standard models are used, examples are Simplex-Lattice designs or Simplex-Centroid designs. The study of mixture experiments did not result in finding a better sample generator. After finding and implementing the current sample generator, the idea evolved that generating mixtures using the sample generator developed for the method might be useful for mixture experiments. It would be possible to specify constraints for the mixtures and to let a program generate mixtures. Possible future research is to investigate if there are any applications for automatically and probabilistically generating mixtures.

7.4 Concluding Remarks

This thesis ends with the last concluding remarks regarding the assignment, the performed research, and the work done to implement the method.

Since not all goals set in Section 1.3 were met, the work is not completely done. Some research questions still need to be solved and the created implementation is not completely ready for integration with GeNIe. What has been accomplished is an implementation that proves the concept presented in (Druzdzel & van der Gaag, 1995). It is indeed possible to let an expert state probabilistic information and to use this information to derive 2^{nd} order distributions over all the CPT entries. But experimental results show that the method needs further fine-tuning, considering the results it currently generates. Even with very strictly specified constraints the generated histograms currently do not give very clear advice on what value to choose for a CPT entry. Possible solutions for this problem have been suggested in Section 6.2.1.

Furthermore the method has its limitations. A limiting factor for the method is the maximum number of parents a node has in a Bayesian network. A node with a large number of parents, has a very large number of CPT entries, which in turn results in a sample space with a very high number of dimensions. One possible problem connected to sampling in a space with a high number of dimensions is a possible limitation of the accuracy of the implementation of the truncated Dirichlet distribution. Inaccuracies could possibly lead to a higher rejection ratio of samples. Another problem, more likely to occur, is that sets of probability statements provided by the expert can result in very few linear constraints. This can cause the linear programming process to be not very successful in reducing the size of the sample space. If the constrained volume is very small and the remaining sample space is much larger, this will again cause a higher ratio of rejected samples.

Basically what will happen is that it will take a very long time before a valid sample, if it exists, will be found. Since there is currently an upper limit implemented on the number of samples to generate before assuming the existence of a conflict somewhere in the set of constraints, it will become more likely that the method will be aborted before a valid sample is found. The upper limit of samples will be reached more frequently.

Ending with a more positive note, the computation time the method needs to generate results completely depends on the input of the user. Factors that are present independent of user input, for example the requirement that every sample must be a valid probability distribution, have been dealt with as efficient as possible. Every generated sample by default satisfies the axioms of probability, and the ability to sample between probability bounds without the use of rejection sampling has increased the efficiency of generating samples by many orders of magnitude. When the proposed importance

sampling scheme is implemented, the efficiency of generating samples may increase even more. It is even likely that, when using importance sampling, the method will perform better in the situation the generated probability bounds have not adequately reduced the sample space. Once valid samples are found the method will increasingly focus on the area where the valid samples were found.

References

Anderson, E. C. (1999). Monte Carlo Methods and Importance Sampling. *Lecture Notes for Stat 578C Statistical Genetics*.

Anderson, H. L. (1986). Metropolis, Monte Carlo, and the MANIAC. *Los Alamos Science*.

Beinlich, I., Suermondt, H., Chavez, R., & Cooper, G. (n.d.). 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* (pp. 247–256). Springer-Verlag.

Boole, G. (1958). *An Investigation of the Laws of Thought on Which Are Founded the Mathematical Theories of Logic and Probabilities*. New York, NY: (Originally published in 1854 by Macmillan) Dover Publications.

Buchanan, B., Sutherland, G., & Feigenbaum, E. (1969). Heuristic DENDRAL: A Program for Generating Explanatory Hypotheses in Organic Chemistry. *Machine Intelligence*, 209–254.

Caprile, B. (2001). Uniformly Generating Distribution Functions for Discrete Random Variables.

Cheng, J., & Druzdzel, M. J. (2000). AIS-BN: An Adaptive Importance Sampling Algorithm for Evidential Reasoning in Large Bayesian Networks. *Journal of Artificial Intelligence Research*, *13*, 155-188.

Chinneck, J. W. (1996). Computer Codes for the Analysis of Infeasible Linear Programs. *Journal of the Operational Research Society*, *47(1)*, 61–72.

Chinneck, J. W. (2001). *Practical Optimization: a Gentle Introduction*. Ottawa, Canada: Carlton University.

Chinneck, J. W. (2002). Discovering the Characteristics of Mathematical Programs via Sampling. *Optimization Methods and Software*, *17(2)*, 319–352.

Chinneck, J. W. (2004). The Constraint Consensus Method for Finding Approximately Feasible Points in Nonlinear Programs. *INFORMS Journal on Computing*.

Chinneck, J. W., & Dravnieks, E. V. (1991). Locating Minimal Infeasible Constraint Sets in Linear Programs. *J. Computing*, *3*, 157–168.

Clemen, R. T. (1996). *Making hard decisions: An introduction to decision analysis*. Belmont, California: Duxbury Press, An Imprint of Wadsworth Publishing Company.

Clemen, R. T., & Reilly, T. (2003). *Making Hard Decisions with DecisionTools®* (2 ed.). Pacific Grove, CA: Duxbury.

Cooke, N. J. (1994). Varieties of Knowledge Elicitation Techniques. *Int. J. Hum.-Comput. Stud.*, *41*(6), 801–849.

Cooke, R. (1991). *Experts in Uncertainty: Opinion and Subjective Probability in Science*. Oxford University Press.

Dantzig, G. (1948). Programming in a Linear Structure. *Comptroller*.

Dantzig, G. (1966). *Linear Programming and Extensions* (3rd ed.). Princeton, New Jersey: Princeton University Press.

Dekking, F., Kraaikamp, C., Lopuhaä, H., & Meester, L. (2004). *Kanstat: Probability and Statistics for the 21st Century*. Delft: Delft University of Technology.

Denny, M. (2001). Introduction to Importance Sampling in Rare-Event Simulations. *Eur. J. Phys.*, *22*, 403–411.

Devroye, L. (1986). *Non-Uniform Random Variate Generation*. New York: Springer-Verlag.

Dombal, F. de, Leaper, D., Horrocks, J., & Staniland, J. (1974). Human and Computer-Aided Diagnosis of Abdominal Pain: Further Report with Emphasis on Performance of Clinicians. *British Medical Journal*, 376–380.

Druzdzel, M., & van der Gaag, L. C. (1995). Elicitation of Probabilities for Belief Networks: Combining Qualitative and Quantitative Information. In *Proceedings of the 11th annual conference on uncertainty in artificial intelligence (uai-95)* (p. 141-148). San Francisco, CA: Morgan Kaufmann Publishers.

Duda, R., Hart, P., & Nilsson, N. (1979). Model Design in the PROSPECTOR Consultant System for Mineral Exploration. *Expert Systems in the Microelectronic Age*, 153–167.

Edelsbrunner, H., & Grayson, D. R. (1999). Edgewise subdivision of a simplex. In *Symposium on computational geometry* (p. 24-30).

Fang, K.-T., Geng, Z., & Tian, G.-L. (2000). Statistical Inference for Truncated Dirichlet Distribution and its Application in Misclassification. *Biometrical Journal*, 1053–1068.

Geweke, J. (1989). Bayesian Inference in Econometric Models Using Monte Carlo Integration. *Econometrica*, *57(6)*, 1317–1339.

Gleeson, J., & Ryan, J. (1990). Identifying Minimally Infeasible Subsystems of Equations. *J. Computing*, *2*, 61–63.

Huang, J. (2005). *Maximum Likelihood Estimation of Dirichlet Distribution Parameters*.

Kolmogorov, A. (1950). *Foundations of the Theory of Probability*. Chelsea, New York: English translation of Kolmogorov(1950).

Lauritzen, S. L., & Spiegelhalter, D. (1988). 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.

Loon, J. van. (1981). Irreducibly Inconsistent Systems of Linear Inequalities. *Eur. J. Opl Res*, *8*, 283–288.

Melle, W. van, Shortliffe, E., & Buchanan, B. (1981). EMYCIN: A Domain-Independent System that Aids in Constructing Knowledge-Based Consultation Programs. *Machine Intelligence, Infotech State of the Art Report 9*, *no. 3*.

Moshier, S. (1989). *Methods and Programs for Mathematical Functions*. New York: Ellis Horwood Limited.

Neapolitan, R. E. (1990). *Probabilistic reasoning in expert systems: Theory and algorithms*. New York: John Wiley & Sons.

Pardalos, P. (1994). On the Passage from Local to Global in Optimization. *Mathematical Programming: State of the Art 1994*.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan Kaufmann Publishers, Inc.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing* (2nd edition ed.). New York: Cambridge University Press.

Riley, G. (November 1991). CLIPS: An Expert System Building Tool. In *Proceedings of the technology 2001 conference* (Vol. 2, pp. 149–158). Washington, DC: NASA.

Rubinstein, R. Y. (1981). *Simulation and the Monte Carlo method*. Hoboken, NJ: John Wiley & Sons.

Russell, S., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (2nd edition ed.). Englewood Cliffs, NJ: Prentice-Hall.

Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton, NJ: Princeton University Press.

Shortliffe, E. (1976). *MYCIN: Computer-Based Medical Consultations*. New York: Elsevier Press.

Sipser, M. (1996). *Introduction to the Theory of Computation*. Boston, MA: International Thomson Publishing.

Smith, P. J., Shafi, M., & Gao, H. (1997). Quick Simulation: A Review of Importance Sampling Techniques in Communications Systems. *IEEE Journal on selected areas in communications*, *Vol. 15 No. 4*.

Yuan, C., & Druzdzel, M. J. (2006). Importance Sampling Algorithms for Bayesian Networks: Principles and Performance. *To appear in Mathematical and Computer Modelling*.

Zadeh, L. (1978). Fuzzy Sets as a Basis for a Theory of Possibility. *Fuzzy Sets and Systems 1*, 3–28.

Appendix A

Results Experiment 1.1: HIV Network

A.1 Constraints Family Network 1

- $P(i \mid c) = 1$:
  - $-x_2 = 0$,
  - $x_2 + x_3 > 0$.

Extra constraints were added by the method to ensure that the CPT entries would exist.

- $P(\neg i) > 0$:
  - $x_0 + x_2 > 0$.
- $P(i) > 0$:
  - $x_1 + x_3 > 0$.

A.2 Constraints Family Network 2

- $P(i\,|c) = 1$:
  - $-x_4 - x_5 - x_{12} - x_{13} = 0$,
  - $x_4 + x_5 + x_6 + x_7 + x_{12} + x_{13} + x_{14} + x_{15} > 0$.
- $P(i) > P(n)$:
  - $x_2 + x_6 + x_{10} + x_{14} - x_1 - x_5 - x_9 - x_{13} > 0$.
- $P(h\,|n) > P(h\,|i)$:
  - $x_2x_9 + x_3x_9 + x_6x_9 + x_7x_9 + x_2x_{11} + x_6x_{11}$
    $+ x_2x_{13} + x_3x_{13} + x_6x_{13} + x_7x_{13} + x_2x_{15} + x_6x_{15}$
    $- x_1x_{10} - x_3x_{10} - x_5x_{10} - x_7x_{10} - x_1x_{11} - x_5x_{11}$
    $- x_1x_{14} - x_3x_{14} - x_5x_{14} - x_7x_{14} - x_1x_{15} - x_5x_{15} > 0$,
  - $x_1 + x_3 + x_5 + x_7 + x_9 + x_{11} + x_{13} + x_{15} > 0$,
  - $x_2 + x_3 + x_6 + x_7 + x_{10} + x_{11} + x_{14} + x_{15} > 0$.
- $0.1 \leq P(n\,|h) \leq 0.25$:
  - $0.75x_9 + 0.75x_{11} + 0.75x_{13} + 0.75x_{15}$
    $- 0.25x_8 - 0.25x_{10} - 0.25x_{12} - 0.25x_{14} \leq 0$,
  - $x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{15} > 0$,
  - $0.1x_8 + 0.9x_9 + 0.1x_{10} + 0.9x_{11} + 0.1x_{12} + 0.9x_{13} + 0.1x_{14} + 0.9x_{15} \leq 0$.

Extra constraints were again added to ensure that all the CPT entries would exist.

- $P(\neg n \neg i \neg c) > 0$:
  - $x_0 + x_8 > 0$.
- $P(\neg n \neg i c) > 0$:
  - $x_4 + x_{12} > 0$.
- $P(\neg n i \neg c) > 0$:
  - $x_2 + x_{10} > 0$.
- $P(\neg n i c) > 0$:
  - $x_6 + x_{14} > 0$.
- $P(n \neg i \neg c) > 0$:
  - $x_1 + x_9 > 0$.

- $P\left(n\neg ic\right) > 0$:
  - $x_5 + x_{13} > 0$.
- $P\left(ni\neg c\right) > 0$:
  - $x_3 + x_{11} > 0$.
- $P\left(nic\right) > 0$:
  - $x_7 + x_{15} > 0$.

Appendix B

Results Experiment 1.2: "Strict" Set

B.1 Generated Constraints

Family Network 1

- $0.0009 \leq P(b) \leq 0.0011$:
  - $x_1 + x_3 + x_5 + x_7 \leq 0.0011$,
  - $-x_1 - x_3 - x_5 - x_7 \leq -0.0009$.
- $0.0018 \leq P(e) \leq 0.0022$:
  - $x_2 + x_3 + x_6 + x_7 \leq 0.0022$,
  - $-x_2 - x_3 - x_6 - x_7 \leq -0.0018$.
- $0.0009 \leq P(a \,|\neg b, \neg e) \leq 0.0011$:
  - $0.9989 * x_4 - 0.0011 * x_0 \leq 0$,
  - $x_0 + x_4 > 0$,
  - $0.0009 * x_0 + -0.9991 * x_4 \leq 0$.
- $0.846 \leq P(a \,|b, \neg e) \leq 1.0$:
  - $-x_1 \leq 0$,
  - $x_1 + x_5 > 0$,
  - $0.846 * x_1 + -0.154 * x_5 \leq 0$.
- $0.261 \leq P(a \,|\neg b, e) \leq 0.319$:
  - $0.681 * x_6 - 0.319 * x_2 \leq 0$,
  - $x_2 + x_6 > 0$,

  - $0.261 * x_2 + -0.739 * x_6 \leq 0$.

- $0.855 \leq P(a \,|b, e) \leq 1.0$:
  - $-x_3 \leq 0$,
  - $x_3 + x_7 > 0$,
  - $0.855 * x_3 + -0.145 * x_7 \leq 0$.

added statements

- $P(\neg b, \neg e) > 0$:
  - $x_0 + x_4 > 0$.
- $P(\neg b, e) > 0$:
  - $x_2 + x_6 > 0$.
- $P(b, \neg e) > 0$:
  - $x_1 + x_5 > 0$.
- $P(b, e) > 0$:
  - $x_3 + x_7 > 0$.

Family Network 2

- $0.045 \leq P(j \,|\neg a) \leq 0.055$:
  - $0.945 * x_2 - 0.055 * x_0 \leq 0$,
  - $x_0 + x_2 > 0$,
  - $0.045 * x_0 + -0.955 * x_2 \leq 0$.
- $0.81 \leq P(j \,|a) \leq 0.99$:
  - $0.01 * x_3 - 0.99 * x_1 \leq 0$,
  - $x_1 + x_3 > 0$,
  - $0.81 * x_1 + -0.19 * x_3 \leq 0$.

added statements

- $P(\neg a) > 0$:
  - $x_0 + x_2 > 0$.
- $P(a) > 0$:
  - $x_1 + x_3 > 0$.

Family Network 3

- $0.009 \leq P(m \,|\neg a) \leq 0.011$:
  - $0.989 * x_2 - 0.011 * x_0 \leq 0$,
  - $x_0 + x_2 > 0$,
  - $0.009 * x_0 + -0.991 * x_2 \leq 0$.
- $0.63 \leq P(m \,|a) \leq 0.77$:
  - $0.23 * x_3 - 0.77 * x_1 \leq 0$,
  - $x_1 + x_3 > 0$,
  - $0.63 * x_1 + -0.37 * x_3 \leq 0$.

added statements

- $P(\neg a) > 0$:
  - $x_0 + x_2 > 0$.
- $P(a) > 0$:
  - $x_1 + x_3 > 0$.

B.2 Generated Probability Bounds

Family Network 1

$$
\begin{array}{lll}
0.995604 & \leq x_0 \leq & 0.997301, \\
0 & \leq x_1 \leq & 0.000169323, \\
0.00047704 & \leq x_2 \leq & 0.00162543, \\
0 & \leq x_3 \leq & 0.000159428, \\
0.00089703 & \leq x_4 \leq & 0.00109802, \\
4.23 \cdot 10^{-7} & \leq x_5 \leq & 0.001099, \\
0.000182831 & \leq x_6 \leq & 0.00070164, \\
4.275 \cdot 10^{-7} & \leq x_7 \leq & 0.0010995.
\end{array}
$$

Family Network 2

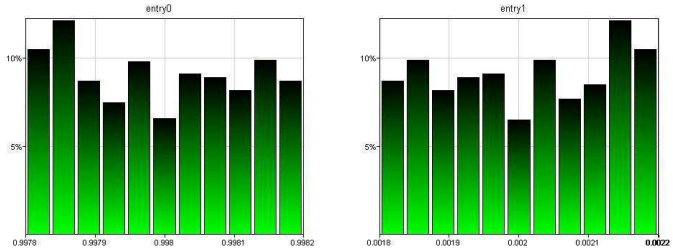
$$
\begin{array}{lll}
4.725 \cdot 10^{-7} & \leq x_0 \leq & 0.955, \\
5 \cdot 10^{-9} & \leq x_1 \leq & 0.19, \\
2.25 \cdot 10^{-8} & \leq x_2 \leq & 0.055, \\
4.05 \cdot 10^{-7} & \leq x_3 \leq & 0.99.
\end{array}
$$

Family Network 3

$$
\begin{array}{lll}
4.945 \cdot 10^{-7} & \leq x_0 \leq & 0.991, \\
1.15 \cdot 10^{-7} & \leq x_1 \leq & 0.37, \\
4.5 \cdot 10^{-9} & \leq x_2 \leq & 0.011, \\
3.15 \cdot 10^{-7} & \leq x_3 \leq & 0.77.
\end{array}
$$

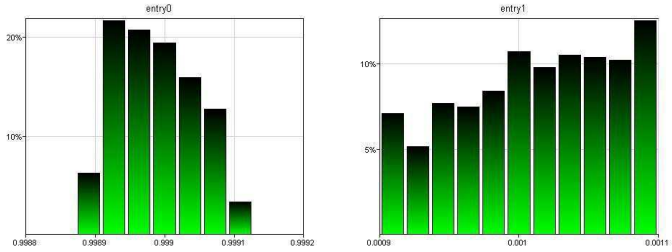
B.3 Histograms

Family Network 1



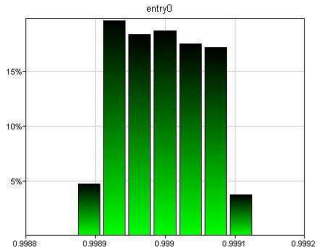
(a) $P(\neg e)$ (b) $P(e)$

Figure B.1: Entries for the node Earthquake

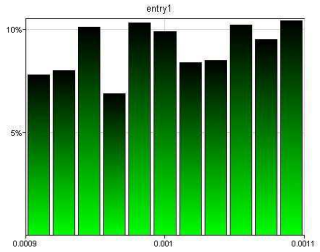


(a) $P(\neg b)$ (b) $P(b)$

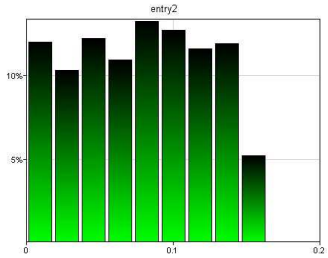
Figure B.2: Entries for the node Burglary



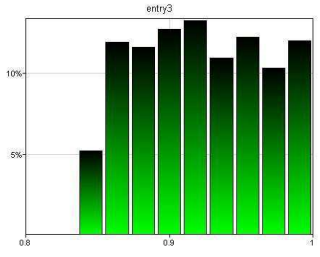
(a) $P(\neg a \,|\neg b, \neg e)$



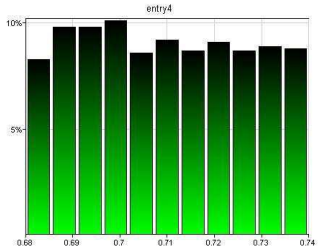
(b) $P(a \,|\neg b, \neg e)$



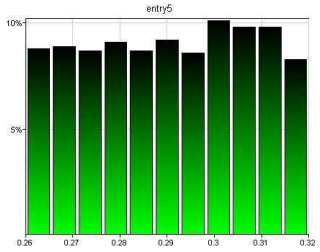
(c) $P(\neg a \,|b, \neg e)$



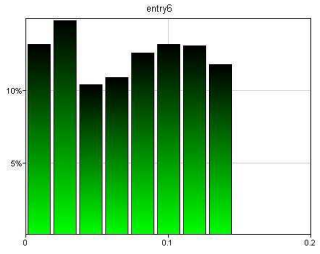
(d) $P(a \,|b, \neg e)$



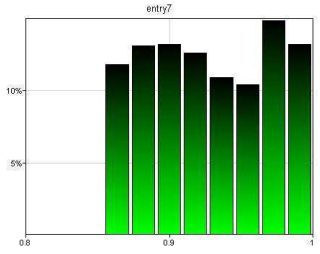
(e) $P(\neg a \,|\neg b, e)$



(f) $P(a \,|\neg b, e)$



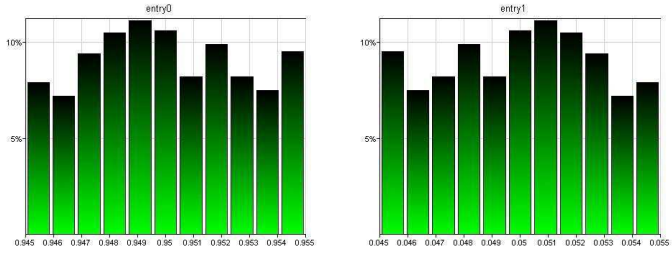
(g) $P(\neg a \,|b, e)$



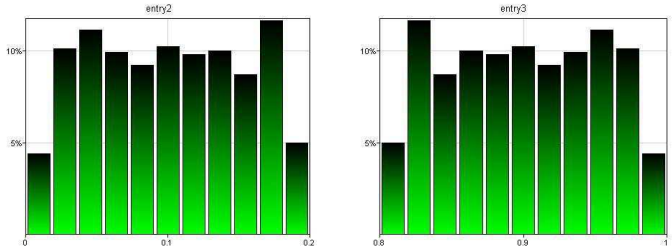
(h) $P(a \,|b, e)$

Figure B.3: Entries for the node Alarm

Family Network 2

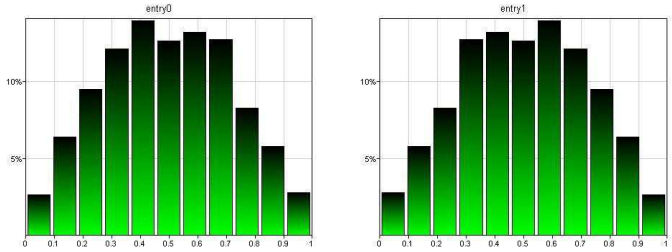


(a) $P(\neg j \,|\neg a)$ (b) $P(j \,|\neg a)$



(c) $P(\neg j \,|a)$ (d) $P(j \,|a)$

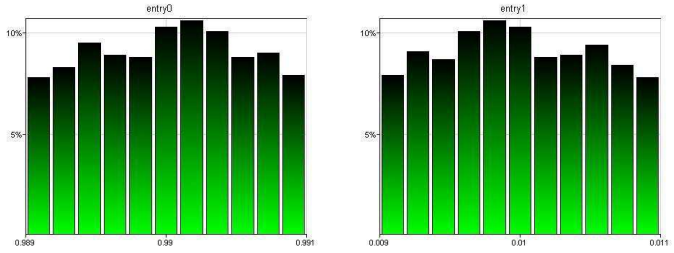
Figure B.4: Entries for the node JohnCalls



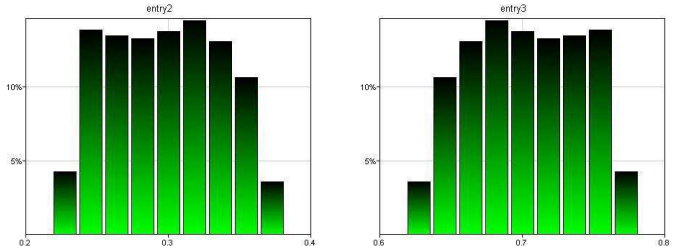
(a) $P(\neg a)$ (b) $P(a)$

Figure B.5: Entries for the node Alarm

Family Network 3

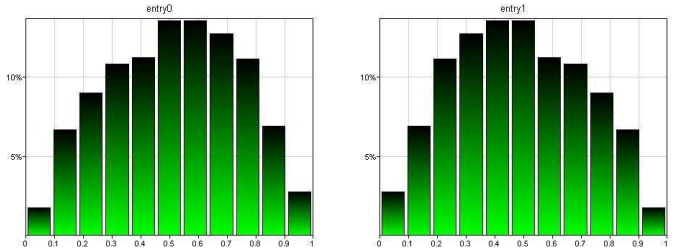


(a) $P(\neg m \,|\neg a)$ (b) $P(m \,|\neg a)$



(c) $P(\neg m \,|a)$ (d) $P(m \,|a)$

Figure B.6: Entries for the node MaryCalls



(a) $P(\neg a)$ (b) $P(a)$

Figure B.7: Entries for the node Alarm

Appendix C

Results Experiment 1.2: "Loose" Set

C.1 Generated Constraints

Family Network 1

- $P(e) < 0.003$:
  - $x_2 + x_3 + x_6 + x_7 < 0.003$.
- $P(e) > P(b)$:
  - $x_2 + x_6 - x_1 - x_5 > 0$.
- $P(b) < 0.002$:
  - $x_1 + x_3 + x_5 + x_7 < 0.002$.
- $P(\neg a \,|\neg b, \neg e) > 0.95$:
  - $0.05 * x_0 - 0.95 * x_4 > 0$,
  - $x_0 + x_4 > 0$.
- $P(a \,|b, e) > 0.9$:
  - $0.1 * x_7 - 0.9 * x_3 > 0$,
  - $x_3 + x_7 > 0$.

- $P(a \mid \neg b, e) < 0.4$:
  - $0.6 * x_6 - 0.4 * x_2 < 0$,
  - $x_2 + x_6 > 0$.
- $P(\neg a \mid b, \neg e) < 0.09$:
  - $0.91 * x_1 - 0.09 * x_5 < 0$,
  - $x_1 + x_5 > 0$.

added statements

- $P(\neg b, \neg e) > 0$:
  - $x_0 + x_4 > 0$.
- $P(\neg b, e) > 0$:
  - $x_2 + x_6 > 0$.
- $P(b, \neg e) > 0$:
  - $x_1 + x_5 > 0$.
- $P(b, e) > 0$:
  - $x_3 + x_7 > 0$.

Family Network 2

- $P(\neg j \mid \neg a) > 0.9$:
  - $0.1 * x_0 - 0.9 * x_2 > 0$,
  - $x_0 + x_2 > 0$.
- $P(\neg j \mid a) < 0.15$:
  - $0.85 * x_1 - 0.15 * x_3 < 0$,
  - $x_1 + x_3 > 0$.

added statements

- $P(\neg a) > 0$:
  - $x_0 + x_2 > 0$.
- $P(a) > 0$:
  - $x_1 + x_3 > 0$.

Family Network 3

- $P(m\,|\neg a) < 0.02$:
  - $0.98 * x_2 - 0.02 * x_0 < 0$,
  - $x_0 + x_2 > 0$.
- $P(\neg m\,|a) < 0.3$:
  - $0.7 * x_1 - 0.3 * x_3 < 0$,
  - $x_1 + x_3 > 0$.

added statements

- $P(\neg a) > 0$:
  - $x_0 + x_2 > 0$.
- $P(a) > 0$:
  - $x_1 + x_3 > 0$.

C.2 Generated Probability Bounds

Family Network 1

$$
\begin{array}{lll}
0.945256 & \leq x_0 \leq & 0.999983, \\
0 & \leq x_1 \leq & 0.000179005, \\
4.13333 \cdot 10^{-6} & \leq x_2 \leq & 0.0029945, \\
0 & \leq x_3 \leq & 0.000198894, \\
0 & \leq x_4 \leq & 0.0499987, \\
5.55556 \cdot 10^{-6} & \leq x_5 \leq & 0.0019945, \\
0 & \leq x_6 \leq & 0.0011973, \\
5 \cdot 10^{-6} & \leq x_7 \leq & 0.00199394.
\end{array}
$$

Family Network 2

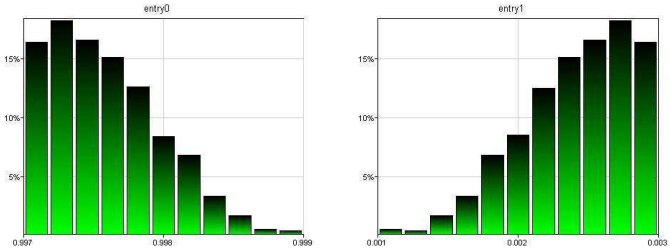
$$
\begin{array}{lll}
5 \cdot 10^{-6} & \leq x_0 \leq & 0.999997, \\
0 & \leq x_1 \leq & 0.149999, \\
0 & \leq x_2 \leq & 0.0999992, \\
3.33333 \cdot 10^{-6} & \leq x_3 \leq & 0.999995.
\end{array}
$$

Family Network 3

$$
\begin{array}{lll}
2.5 \cdot 10^{-5} & \leq x_0 \leq & 0.999998, \\
0 & \leq x_1 \leq & 0.299992, \\
0 & \leq x_2 \leq & 0.0199995, \\
1.66667 \cdot 10^{-6} & \leq x_3 \leq & 0.999975.
\end{array}
$$

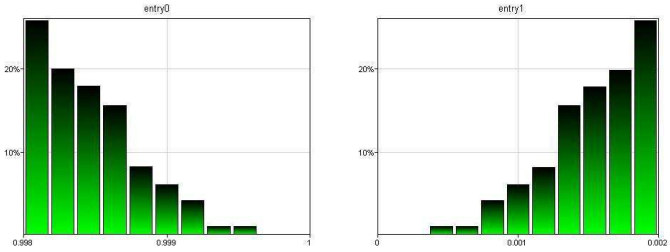
C.3 Histograms

Family Network 1



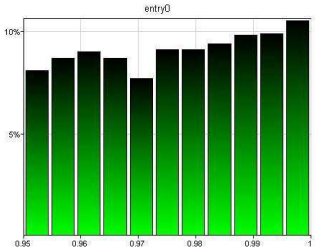
(a) $P(\neg e)$ (b) $P(e)$

Figure C.1: Entries for the node Earthquake

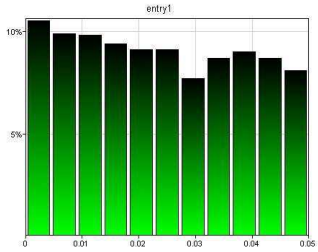


(a) $P(\neg b)$ (b) $P(b)$

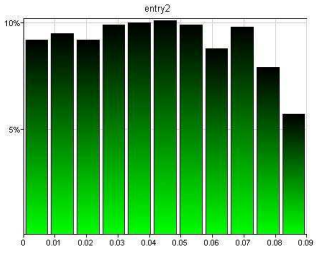
Figure C.2: Entries for the node Burglary



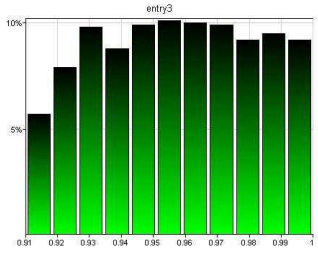
(a) $P(\neg a \,|\neg b, \neg e)$



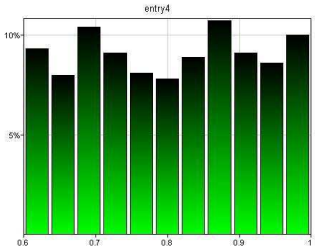
(b) $P(a \,|\neg b, \neg e)$



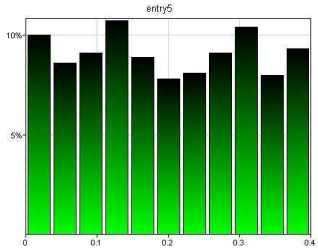
(c) $P(\neg a \,|b, \neg e)$



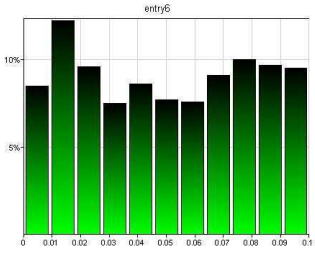
(d) $P(a \,|b, \neg e)$



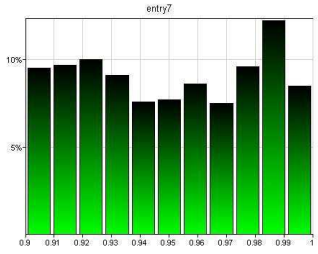
(e) $P(\neg a \,|\neg b, e)$



(f) $P(a \,|\neg b, e)$



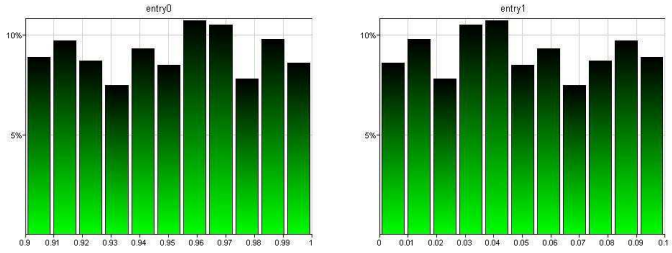
(g) $P(\neg a \,|b, e)$



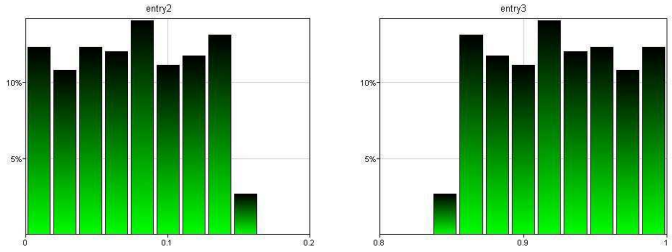
(h) $P(a \,|b, e)$

Figure C.3: Entries for the node Alarm

Family Network 2

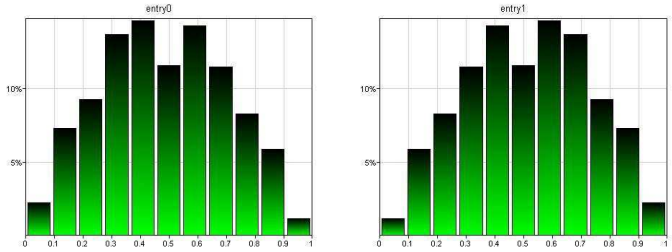


(a) $P(\neg j \,|\neg a)$ (b) $P(j \,|\neg a)$



(c) $P(\neg j \,|a)$ (d) $P(j \,|a)$

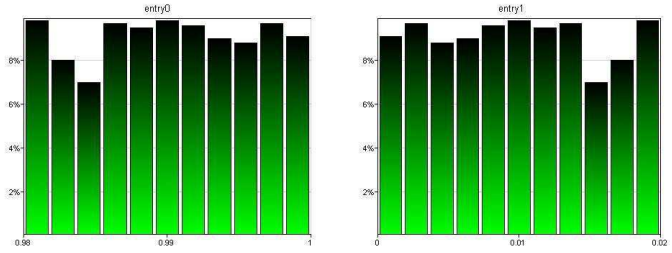
Figure C.4: Entries for the node JohnCalls



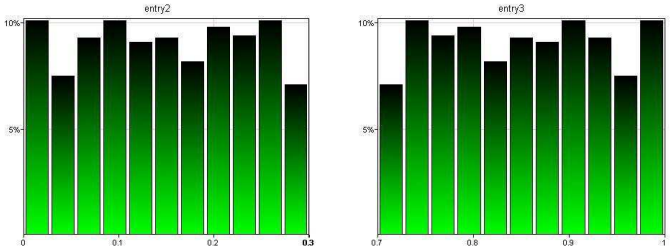
(a) $P(\neg a)$ (b) $P(a)$

Figure C.5: Entries for the node Alarm

Family Network 3

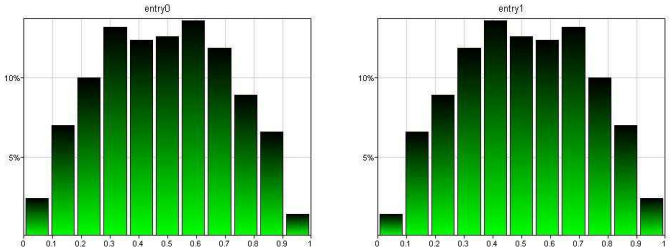


(a) $P(\neg m \,|\neg a)$ (b) $P(m \,|\neg a)$



(c) $P(\neg m \,|a)$ (d) $P(m \,|a)$

Figure C.6: Entries for the node MaryCalls



(a) $P(\neg a)$ (b) $P(a)$

Figure C.7: Entries for the node Alarm

Appendix D

Paper Version

Implementing and Improving a Method for Non-Invasive Elicitation of Probabilities for Bayesian Networks

ing. Martinus A. de Jongh
Man-Machine Interaction Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, Delft
m.a.dejongh@student.tudelft.nl

dr. Marek J. Druzdzel
Decision Systems Laboratory
School of Information Sciences
& Intelligent Systems Program
University of Pittsburgh
Pittsburgh, PA 15260
marek@sis.pitt.edu

Abstract

Knowledge elicitation is difficult for expert systems that are based on probability theory. The elicitation of probabilities for a probabilistic model of a problem requires a lot of time and interaction between the knowledge engineer and the expert.

Bayesian networks (BNs) are an example of a structure that can be used to create a probabilistic model. BNs need specific conditional probabilities. If an expert does not know these probabilities, but knows other useful probabilistic information, this information generally cannot be used directly for BNs. It will be necessary to perform calculations before the information is transformed into conditional probabilities directly usable for BNs.

Druzdzel and van der Gaag (1995) have proposed a theoretical framework that would allow for the direct use of other types of probabilistic information. This framework has been used as a starting point for the implementation of a non-invasive elication method. There were many possibilities to improve the framework. Among them were: finding methods for conflict detection and conflict resolution and improving the sampling process that is at the core of the method.

1 Introduction

Bayesian networks (BNs) (Pearl, 1988) can be used to store the knowledge of human experts in probabilistic models. BNs are graphical models that very efficiently represent joint probability distributions. This efficiency is the result of conditional independence assumptions that are added to the model.

To create a Bayesian network, its graph needs to be defined and its CPTs for every node need to be filled with the necessary probabilities.
Basically three approaches exist:

- Elicit the necessary information from a domain expert.
- Learn the necessary information from data.
- The knownledge engineer that is creating the BN estimates the probabilities himself, using relevant literature.

Both the graph and the CPTs can be created using all of the methods. Depending on the application one method may be more appropriate than the other. Generally, in situations where there is not enough data, or no data at all, elicitation of knowledge from experts is the only viable approach to get the necessary probabilities for a model.

Knowledge elicitation for systems based on probability theory is difficult. To describe this difficulty in a nutshell: the expert has to assign probabilities, which are numerical values, to all the possible events the application is modeling. Beside the fact that the number of probabilities increases exponen-

tially with the size of the model, experts may find it difficult to assign an *exact* value to an event. The probabilities stated by an expert are subjective, which means they represent the degree of belief the expert has that the events the probabilities describe will occur. When an expert is stating a (subjective) probability, he or she is usually not performing mental calculations (R. Cooke, 1991). Experts will generally rely on rules of thumb, or heuristics. Using heuristics can cause an expert to become biased. The expert may not be able to provide probabilities that accurately represent the true probabilities of the events to be modeled. To counter this bias the expert will need to be calibrated. Calibration is a difficult, but necessary process to acquire probabilities that approach the "real" probabilities more closely.

After calibration the elicitation process can begin. Research in knowledge elicition has shown that asking an expert to state probabilities directly generally leads to probability estimates of low quality (R. Cooke, 1991). It is generally better to use an indirect approach that uses a sort of betting game to determine the desired probabilities (R. Cooke (1991); Clemen and Reilly (2003)).

Until now it is assumed that the expert actually can give, maybe by using betting games, an estimate of the desired probabilities. However, it is possible that an expert cannot give an estimate for a certain probability directly, but only implicitly by estimating other probabilities first and calculating the desired probability using, for instance, Bayes' rule. Also an expert may have information that is not quantitative of nature, i.e. not in the form of numerical probabilities. Other types of probabilistic information exist that are qualitative of nature and cannot be directly interpreted as probabilities for a Bayesian network. If the expert can provide these types of information and it is relevant for the model to be developed it would be very inefficient if it would be impossible to use this information for the model.

If it were possible to use probabilities and probabilistic information other than the necessary conditional probabilities directly this would make the eliciation process easier for the expert. Here with easier it is meant that any information relevant to the model can now be be used for filling the CPT entries of the Bayesian network, and it would no longer be necessary to let the expert transform the information he or she has into conditional probabilities that can be directly put in the BN. According to (Druzdzel & van der Gaag, 1995) such an elicitation method can be considered to be *non-invasive*. They define non-invasive as:

> *"Allowing any type of probabilistic information, quantitative or qualitative of nature, the expert is willing to state to be interpreted directly for the elicitation of probabilities for a Bayesian network."*

Druzdzel and van der Gaag (1995) have proposed a theoretical framework for an elicitation method that lets an expert specify various statements of qualitative and quantitative nature and interprets these statements as constraints used to guide the determination of the CPT entries for a Bayesian network.

2 Related Work

A lot of work has been done in knowledge elicitation in general (N. J. Cooke, 1994), many different types of interviewing and observation techniques exist. Specifically for eliciting probabilities usually indirect elicitation techniques are used to elicit probabilities from experts. The most popular techniques are betting games and lotteries (Clemen & Reilly, 2003). In the case of a binary variable A with values true and false, a betting game works as following, two bets are presented to the expert:

1. Win \$X if A is true.
   Lose \$Y if A is false.

2. Lose \$X if A is true.
   Win \$Y if A is false.

X and Y represent the amount of money that is put into the betting "pot". The knowledge engineers choose the values for X and Y. The idea is to ask the expert to choose between the bets and to adjust the values of X and Y until the expert is indifferent between the bets. When the point of indifference is reached, the expected values of the two bets must be equal. Equating the two expected values and solving for the probability that A is true gives:

$$P(A = true) = \frac{Y}{X + Y}.$$

Now the determined values of X and Y can be used to approximate the probability that A is true, and, respectively, the probability that A is false. The approach is not without problems, some people may not like the betting analogy. For these people, the game may be more distracting than direct probability assessment and the result will be opposite of the original intention of the game. Also, most people do not like the idea of losing money although the game is purely hypothetical. This "fear" of losing money will have an influence on the decision that the expert will make when picking bets (Clemen & Reilly, 2003). Another type of game is the lottery: here the expert has to choose between two different lotteries:

1. Win Prize X if event A is true.
   Win Prize Y if event A is false.

2. Win Prize X with know probability p.
   Win Prize Y with probability 1–p.

The second lottery serves as a reference, where the mechanism to get the probability must be well defined. This might be spinning a wheel with two areas (true and false) or drawing a colored ball from a collection of balls that has a color distribution equal to the probability distribution defined by p. Now, just as with the betting game, the expert is asked to choose the lottery he would like to try. After the expert chooses a lottery, the knowledge engineer changes the value of p to make the other lottery more attractive to the expert. This process continues until the expert is again indifferent between the lotteries. At this point, the current value of p is the probability that should be chosen for the event A = true. Interesting to notice is that the lottery device uses a probability distribution as reference and the expert has to choose the preferred lottery by looking at the values of the distribution. The numbers of the distribution might again influence the expert and to solve this problem, a graphical representation of the distribution could be used. One example is what Clemen and Reilly call a probability wheel, a pie chart where the different areas represent the probabilities of the distribution. By changing the boundaries of the areas, the probabilities change. The visual representation of this process might give the expert a better view of the situation, and improve his probability estimates. Also other graphical representations could be used. GeNIe, software developed at University of Pittsburgh's Decision System Lab, allows the use of probability wheels and bar charts for determining probabilities. Just as betting games are not always accepted by experts, lotteries also may have some negative aspects. Again some people may not like playing games or have difficulty getting 'into' the game. Both methods can be expanded to be able to handle discrete variables (with more than two values) and continuous variables.

These techniques are helpfull when an expert can give an estimation of the desired probability. When the expert cannot produce an estimate for the desired probability, but has other relevant and useful knowledge, it should be possible to use the information for the model. There do exist several methodologies that are not based on probability theory and that can incorperate other types of knowledge into a model. Two examples are Possibility Theory (Zadeh, 1978), that by using non-crisp logic allows for dealing with uncertain and imprecise information, and Dempster-Shafer theory (Shafer, 1976), which is a generalization of the Bayesian theory of subjective probability and focuses more on belief functions and the degree of belief one has in a statement.

The approach proposed by Druzdzel and van der Gaag (1995) is to consider the distribution hyperspace of all possible joint probability distributions over V. A point somewhere in this hyperspace will be the true probability distribution, Pr, over the set of variables V. If there is no information available, qualitative or quantitative, then the Pr can be any point in the hyperspace. Once more information is known about Pr, some of the probability distributions in the hyperspace will become incompatible with this information. Probability elicitation can be looked upon as constraining the distribution hyperspace as much as possible to find the true distribution Pr. All information for the distribution Pr is expressed as constraints for the hyperspace, and, assuming that all compatible distributions are equally likely, 2^{nd} order probability distributions over the probabilities of the distribution Pr are derived. These distributions are used for determining the probabilities of the joint probability distribution. Since the method allows the use of various types of probabilistic information it is possible to use any information the expert is willing to state. This allows for the process of eliciting

probabilities to be non-invasive.

2.1 Canonical Form

The basic idea of the approach is to have a canonical form for the interpretation of probabilistic information. The form builds on the property that any joint probability distribution on a set of variables V is uniquely defined by the probabilities of all possible combinations of values for all variables from V. With all these values known, any probability over the set V can be computed by using marginalization and conditioning. Combinations of values for all variables are called *constituent assignments*. The probabilities of constituent assignments in a joint probability distribution are called its *constituent probabilities*. The authors look upon the set of all possible joint probability distributions on V as spanning a hyperspace whose dimensions correspond with constituent probabilities.

Any information about the true distribution Pr can now be represented as a system of (in)equalities with the constituent probabilities as unknowns. Any solution of this system will be a joint probability distribution that is compatible with all the specified probabilistic information. If there are no solutions, then the provided information is inconsistent.

Since all constituent assignments are mutually exclusive, a probability $Pr\,(b)$ can be expressed as the sum of the probabilities of the constituent assignments b is built from. So from $Pr\,(b) = \sum Pr\,(c_i)$ the authors found that $Pr\,(b)$ can be expressed as

$$d_1x_1 + d_2x_2 + \cdots + d_kx_k\ ,$$

where $x_i = Pr\,(c_i)$, $i = 1, \ldots, k$, and $d_i = 1$ if the constituent x_i is part of b and $d_i = 0$ otherwise.

2.2 Interpretation of Probabilistic Information

Different types of probabilistic information can be interpreted and translated into the canonical form so that it can be used for the elicitation process. The most basic form of probabilistic information that has to be interpreted are the axioms of probability. A point in the hyperspace must be compatible with the axioms or it will not be a valid probability distribution.

One of the axioms states that the probability of a true event, $Pr\,(true)$, is equal to 1. This means that the sum of the probabilities of the distribution should add up to 1. This axiom can be translated into the canonical form by the equality

$$x_1 + \cdots + x_k = 1\ ,$$

where $x_i = Pr\,(c_i)\,, i = 1, \ldots, k$. Another axiom states that any probability must be a nonnegative, real number. So this means that all constituent probabilities must be larger or equal to 0. In canonical form this can be expressed as

$$x_i \geq 0\ ,$$

for $i = 1, \ldots, k$. Also all probabilities must be smaller than or equal to 1, but this is implied by the two statements above.

The authors have defined some types of probabilistic information of a quantitative nature:

1. Point estimate.
2. Probability intervals.
3. Comparison.

A *point estimate* is a statement in the form $Pr\,(b) = p, 0 \leq p \leq 1$, where b is an assignment for an arbitrary subset of variables. For a conditional probability $Pr\,(b_1\,|b_2\,)$ a point estimate is in roughly the same form, but the conditional probability is first transformed into $\frac{Pr(b_1b_2)}{Pr(b_2)}$ and the final canonical form will become:

$$Pr\,(b_1b_2) - p \cdot Pr\,(b_2) = 0\ .$$

An extra constraint has to be added to the system. $Pr\,(b_2)$ has to be larger than 0 or else the conditional probability will become infinite and thus invalid. So, the extra added constraint is the inequality:

$$Pr\,(b_2) > 0\ .$$

Probability intervals and *comparisons* have similar canonical representations. A probability interval has instead of one bound, an upper and a lower bound: $p_1 \leq Pr\,(b) \leq p_2$. A comparison is between two probabilities and has the form: $a_1 \cdot Pr\,(b_1) \leq a_2 \cdot Pr\,(b_2)$, where a_1 and a_2 are non-negative real numbers.

The authors have also defined some types of probabilistic information that are qualitative of nature:

1. Qualitative influences.
2. Qualitative synergies.

A *qualitative influence* is a symmetric property describing the sign of probabilistic interaction between two variables V_1 and V_0, and builds on an ordering of these variables' values. Three different types of qualitative influences exist: positive, negative, and zero qualitative influences. A positive qualitative influence expresses that when a higher value of V_1 is observed, it is more likely that higher values for V_0 will also be observed. This is denoted by $S^+(V_1, V_0)$. The condition is that this relation is valid if and only if for all values v_{0_m} of V_0, for all pairs of distinct values $v_{1_i} > v_{1_j}$ of V_1, and for all possible assignments b for the set of V_0's direct predecessors other than V_1, it is valid that

$$Pr\left(V_0 \geq v_{0_m} \left| v_{1_i} b\right.\right) \geq Pr\left(V_0 \geq v_{0_m} \left| v_{1_j} b\right.\right) \ .$$

For the negative and zero qualitative influences a similar expression exists.

Another type of qualitative probabilistic information is a *qualitative synergy*. Two types of synergies are: additive synergies and product synergies. Both types come in the form of positive, negative, or zero variants.

Additive synergies describes the joint influence of two variables V_1 and V_2 on a third variable V_0, and, similarly to qualitative influence, builds on an ordering of these variables' values. A positive additive synergy of V_1 and V_2 with respect to V_0 expresses that the joint influence of V_1 and V_2 is greater than the sum of their individual influences. This is denoted by $Y^+(\{V_1, V_2\}, V_0)$. The condition is that this relation is valid if and only if for all values v_{0_m} of V_0, for all pairs of values $v_{1_i} > v_{1_j}$ of V_1 and $v_{2_{i'}} > v_{2_{j'}}$ of V_2, and for all possible assignments b for the set of V_0's direct predecessors not including V_1 and V_2, it is valid that

$$\begin{aligned} &Pr\left(V_0 \geq v_{0_m} \left| v_{1_i} v_{2_{i'}} b\right.\right) + \\ &Pr\left(V_0 \geq v_{0_m} \left| v_{1_j} v_{2_{j'}} b\right.\right) \\ \geq\ &Pr\left(V_0 \geq v_{0_m} \left| v_{1_i} v_{2_{j'}} b\right.\right) + \\ &Pr\left(V_0 \geq v_{0_m} \left| v_{1_j} v_{2_{i'}} b\right.\right) \ . \end{aligned}$$

As with the qualitative influences, negative additive synergy and zero additive synergy are defined in a similar manner.

Product synergies describe the interaction between two variables V_1 and V_2 conditional on their common descendant V_0 and expresses the sign of what is known as inter causal influence between V_1 and V_2. The most common type of product synergy is the negative product synergy. This type captures the notion of "explaining away." The authors state that the variables V_1 and V_2 exhibit negative product synergy with respect to a particular value v_{0_m} of variable V_0, written as $X^-(\{V_1, V_2\}, v_{0_m})$, if for all pairs of values $v_{2i} > v_{2_j}$ of V_2 and for all possible assignments b for the set of V_0's direct predecessors not including V_1 and V_2, it is valid that

$$\begin{aligned} &Pr\left(V_1 \geq v_{1_i} \left| v_{2_i} v_{0_m} b\right.\right) \leq \\ &Pr\left(V_1 \geq v_{1_i} \left| v_{2_j} v_{0_m} b\right.\right) \ . \end{aligned}$$

Positive and zero product synergy are defined in a similar manner and the translation to the canonical form is performed in the same way as qualitative influences and additive synergies. A difference between product synergies and additive synergies is that product synergies are with respect to separate values of the common effect V_0. Thus there are as many product synergies as there are values of V_0.

2.3 Elicitation of Probabilities

To derive the 2^{nd} order distributions for the probabilities to be assessed the authors have proposed to use sampling. For the computation of the 2^{nd} order distributions randomly selected points from the distribution hyperspace, under the assumption that all points in the hyperspace are equally likely to be the true distribution, are selected. Every selected distribution is verified to be sure that it is compatible with all available information. All distributions that are compatible with all constraints are collected and used for the generation of the 2^{nd} order distributions over the probabilities. The process is computationally expensive as it involves generating and investigating joint probability distributions.

3 Theory

The framework proposed by (Druzdzel & van der Gaag, 1995) was used as a starting point for an implemenation of a non-invasive elicitation method. The framework was implemented

and improved, which was necessary to make it feasible.

3.1 Decomposing a Bayesian Network

The BN is decomposed into smaller sub networks to make the J-PDFs that the method has to work with have a more manageable size. A method for decomposing a BN has been designed. The idea is to break up the BN into families; sub networks that consist out of a node and its parents. Using family networks (FNs), the expert can focus on providing information for a only a small part of the network without having to worry about other nodes.

3.2 Translation of Expert Statements into Constraints

When the BN is decomposed, the expert can provide probabilistic statements per family for each family. Once the expert is done, the statements will have to be translated into constraints for the probability hyperspace. The equations/inequalities are represented by binary expression trees. The reason for this choice was that binary expression trees are most commonly used for similar problems.

The parsing of the different types of probabilistic statements was designed in such a way that the more complex probability statements make use of the simpler ones. An example is the qualitative influence statement. This statement can first be translated into a number of (quantitative) probability statements. The number of statements depends on the number of variables in the family network. These probability statements can be translated into one or more equations/inequalities consisting out of constituents and the basic arithmetic elements. These equations are the final form and are represented using expression trees. Instead of generating the resulting expression trees directly from the qualitative statement, first the probability statements are generated, which are then parsed and then from each probability statement the resulting trees are generated. A similar mechanism is necessary for conditional probability statements. An extra constraint must be added to ensure that the conditional part of the probability has a probability larger than 0. A simple example would be: the statement $P(A|B) > 0.2$ needs an extra statement $P(B) > 0$ because otherwise the conditional probability does not exist.

3.3 Identification of Probability Bounds

When the system has acquired the constraints from the expert, it could start the sampling process, but this will be quite inefficient. Druzdzel and van der Gaag (1995) propose that first a preprocessing step should be performed to reduce the size of the sample space. They have envisioned using linear programming (LP) to tighten the bounds for every constituent. To calculate the upper and lower bounds for each constituent, the proposed LP method has been used.

It is important to notice that only the linear constraints provided by the expert can be used for these calculations. It was researched how it would be possible to automatically determine if a constraint was linear and to extract the necessary information from the constraints to create the matrices necessary for the LP procedure.

Using some knowledge of how the different constraints typically look like and by implementing some symbolic mathematical operations for the expression trees, a method was developed that manipulates the trees into a standard form that makes it very easy to decide if a constraint is linear or nonlinear. There may exist situations where this method will fail, but in these cases the method will mistake a linear equation for an nonlinear. In this situation the constraint will be excluded from the LP process, which is not as bad as trying to include a nonlinear constraints in the LP process.

Using the LP procedure does indeed decrease the size of the sample space and thus will improve the efficiency of the sampling process, but because the nonlinear constraints are not considered in this process, there should still be more to gain in sampling efficiency.

3.4 Derivation of the 2^{nd} Order Distributions

To generate samples for evaluation, a truncated Dirichlet distribution (TDD) has been used. This is a variant of the Dirichlet distribution where the entries of the sample vector can be constrained by

lower and upper bounds. A vector X generated from a TDD, with

$$X = (x_1, \ldots, x_N)^T,$$

and

$$x_{-N} = 1 - \sum_{i=1}^{N-1} x_i,$$

has a density function for vector $X\,(x_1, \ldots, x_{N-1})^T$ (Fang, Geng, & Tian, 2000):

$$c^{-1} \prod_{i=1}^{n-1} x_i^{\gamma_i - 1} \left(1 - \sum_{i=1}^{n-1} x_i\right)^{\gamma_N - 1}, \quad X_{-N} \in V_{N-1}(A, B),$$

where c is the normalizing constant, and $V_{N-1}(A, B)$ is the volume created by lower bound vector $A = (a_1, \ldots, a_N)$ and upper bound vector $B = (b_1, \ldots, b_N)$:

$$V_{N-1}(A, B) = \left\{ X : \begin{array}{c} 0 \le a_i \le x_i \le b_i \le 1, \\ i = 1, \ldots, N-1, \\ a_N \le 1 - \sum_{i=1}^{N-1} x_i \le b_N \end{array} \right\}.$$

When all entries of vector A are 0 and the entries of vector B are 1, the TDD reduces to a standard Dirichlet distribution. When all the parameters γ_i are chosen to be 1 the distribution reduces to a uniform distribution, its samples are uniformly distributed over a convex polyhedron

$$T_N(A, B) = \left\{ X : 0 \le a_i \le x_i \le b_i, i = 1, \ldots, N, \sum_{i=1}^{N} x_i = 1 \right\}.$$

Samples from the TDD are generated by sequentially generating samples from the marginal distributions of the TDD, which according to (Fang et al., 2000) are truncated beta distributions, $Tbeta\left(\gamma_k, \sum_{i=1}^{N} \gamma_i - \sum_{i=k}^{N-1} \gamma_i; \xi_k, \eta_k\right)$. $Tbeta$ is a truncated version of the standard $Beta(\alpha, \beta)$ distribution, where ξ is the lower bound and η is the upper bound. For every entry of the TDD vector new values for ξ_k and η_k must be calculated.

An algorithm can be created for generating samples from a TDD by sequentially generating samples from a truncated beta distribution with the appropriate parameters. This algorithm has been described in Figure 1.

Here $F_k^{-1}(\cdot)$ is the inverse of the cumulative distribution function of the $Tbeta\left(\gamma_k, \sum_{i=1}^{N} \gamma_i - \sum_{i=k}^{N-1} \gamma_i; \xi_k, \eta_k\right)$ distribution.

To calculate $F_k^{-1}(x)$, the cumulative distribution function $F_\beta(x)$ and the inverse cumulative distribution function $F_\beta^{-1}(x)$ of the standard $Beta(\alpha, \beta)$ distribution are used.

$$F_k^{-1}(x) = F_\beta^{-1}\left(F_\beta(\xi_k) + x * \left(F_\beta(\eta_k) - F_\beta(\xi_k)\right)\right).$$

3.5 Conflict Detection

All the constraints have to be satisfied for a sample to be valid, any sample that does not satisfy all constraints must be discarded. There is one big problem with this approach: conflicting constraints will prohibit any sample to be valid. A set of constraints is conflicting with each other when it is impossible for all the constraints to be satisfied at the same time. Detecting conflicts is hard, one can never be sure that the absence of valid samples is because of conflicting constraints or that the sampling procedure just has not hit inside the feasible area. Pin-pointing the conflicting constraints is even harder, and there does not yet exist a good procedure for finding conflicting constraints for the nonlinear, non convex case, which is the worst case situation encountered when applying the method.

Two heuristics are being used to aid the the expert when deciding which constraint(s) need to be changed or removed. One heuristic (majority heuristic) was devised by the author and the other, the constraint effectiveness heuristic, was created by Chinneck (2002). The majority heuristic is based on the idea that when the majority of the constraints evaluates a sample as true that there might be something wrong with the minority of constraints that has evaluated the sample as false. In this situation 1 is added to the minority counter for each constraint that belonged to the minority. In the situation that the majority evaluated the sample as false, the minority counters are left unchanged. The reason for this is that there are no conflicting constraints only when all constraints have simultaneously evaluated a sample as true at least once, when all constraints have simultaneously evaluated a sample as false there is no guarantee that the set of constraints does not contain any conflicting constraints. Thus when a constraint evaluates a sample as true, this can be considered as stronger evidence then when it eval-

Input:

- a parameter vector $\gamma = (\gamma_1, \ldots, \gamma_N)^T$,

1. For $k = N-1, k > 0, k = k-1$
   (a) Calculate the values for ξ_k and η_k using

$$\xi_k = max\left(\frac{a_k}{1-\sum_{j=k+1}^{N-1} x_j}, 1 - \frac{\sum_{i=1}^{N} b_i - \sum_{j=k}^{N-1} b_j}{1-\sum_{j=k+1}^{N-1} x_j}\right), \quad (1)$$

$$\eta_k = min\left(\frac{b_k}{1-\sum_{j=k+1}^{N-1} x_j}, 1 - \frac{\sum_{i=1}^{N} a_i - \sum_{j=k}^{N-1} a_j}{1-\sum_{j=k+1}^{N-1} x_j}\right). \quad (2)$$

   Where $\sum_{j=k+1}^{N-1} x_j$ is defined as 0 when $k+1 > N-1$.
   (b) Generate a random value u_k from a uniform $U[0,1]$ distribution.
   (c) Calculate $x_k = \left(1-\sum_{j=k+1}^{N-1} x_j\right) F_k^{-1}(u_k)$
2. Calculate $x_N = 1 - \sum_{i=1}^{N-1} x_i$
3. Output vector $X = (x_1, \ldots, x_N)^T \sim TD(A, B; \gamma_1, \ldots, \gamma_N)$

Figure 1: Algorithm for generating samples from a truncated Dirichlet distribution using the method described by Fang et al. (2000)

uates the sample as false. After all samples have been processed by the heuristic, some steps are performed to normalize the results so that each statement provided by the expert gets a heuristic value between 0 and 1. The closer the value is to 1.0 the "worse" the constraint is.

The majority heuristic will most likely work better if the set of constraints is larger. If the number of constraints is very small it might become less frequent that a majority of constraints evaluates samples as true. In this situation the minority counters of the constraint will be hardly updated or even not at all. When this occurs the results from the heuristic may not be very useful and it might be better that Chinneck's constraint effectiveness heuristic is used for making decisions.

In both the heuristics high values are "bad", and when the results are presented to the user, the constraints are sorted so that the highest ranking ones are on the top of the list. A high value for the constraint effectiveness heuristic means that a constraints has rejected a large number of the samples and that it might be to constraining. A high value for the majority heuristic means that a constraint or constraints from an expert statement have been minority "voters" for a large number of the samples and that it may need to be changed or removed.

3.6 Merging Family Networks

Merging the family networks and the samples generated for the nodes in the different family networks is not an easy process. The sample process has been performed on each of the family networks independently and possibly the results for the CPT entries may vary per FN. Since different FNs contain different variables it is likely that different FNs do not have the same set of constraints. The difference in constraints has influenced the sampling process and through the collected samples, the shapes of the histograms for the different CPT entries of the nodes. Another problem that contributes to the difficulty of merging FNs is that when the BN is decomposed in the FNs, some nodes will be a child

node in one FN and a parent in possibly multiple other FNs. When this happens the node will have different CPTs for the different FNs. When the node is a child, its values will be conditioned on all its parents. When the node is a parent, it may not have any parents itself in this FN and in this case it will have a prior distribution and not a conditional one. This will result in probability tables with different sizes. A binary node without parents only has 2 entries, but in another FN the same node may be a child with 2 parents and have a CPT with 8 entries. Somehow, when merging the FNs, the different sample sets must be merged together to get the sample sets and the histograms for the CPT entries for the node in the original BN. After merging the FNs into the BN, the size of the CPT for every node is equal to the largest CPT size of the node among the FNs. For nodes that have parents this will be the FN where they are the child of the FN. Nodes that do not have any parents, will have the same size CPT in any FN they appear in. The CPT size for these nodes will also not change after merging the FNs into the BN.

A merging method for merging the FNs into the BN that is mathematical sound was not found. Currently, the FNs with their samples and histograms are shown to the user of program (most likely a domain expert and/or a knowledge engineer), and the user must decide how he or she wants to fill in the BN. This was already the original idea, to give the user more control over the end result, but with just one histogram for each entry of the nodes.

4 Experiment

An experiment was done to evaluate the performance of the implementation created for the method.

Goal The goal for the experiment is to compare the influence of two different probability statement sets on the resulting histograms.

Design The Bayesian network used in the experiment is the alarm system network by (Pearl, 1988). Two sets of constraints were created for this network. The first set of constraints can be considered as a sort of control group. For half of the CPT entries of the BN probability statements were created that would be in the form

$$0.9x \leq P(\cdot) \leq 1.1x,$$

where $P(\cdot)$ is the probability of a CPT entry and x is the real value of the entry. These statements create an interval around the real value, where the upper and lower bound differ 10% from the real value.

The second set of probability statements was created in a manner that is more likely to be encountered in real life. Normally an expert would be necessary to create the statements and he or she would probably be able to translate rules of thumb into probability statements. For the experiment probability statements have been created that describe the CPT entries of the BN loosely. The statements are still based on the true values of the CPTs, but no intervals have been specified and values in the statements have been chosen to be less restrictive.

Results Both the probability statement sets have been inputted in the method together with the alarm system BN. Three different family networks were identified.

For both sets of probability statements the method has generated constraints that were used for the sampling process. The probability statements are divided over the FNs. Only statements that are relevant for a FN are translated into constraints for the sampling process. After these constraints were generated the Linear Programming process described in Section 3.3 was run using the linear constraints. All the constraints were linear, so all could be used to determine probability bounds for all the family networks.

The probability bounds improve the efficiency of the method enormously. Only a very small sub volume of the whole hypercube needs to be sampled now. In the case of FN 1 the probability bounds create a new hypercube that has a volume of only $6.6 \cdot 10^{-27}$ times the whole hypercube volume. And only the intersection of the 7-simplex with this hypercube will be searched by the sample generator.

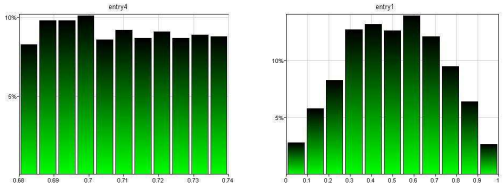
After the generation of the constraints and the calculation of the probability bounds the sampling process is started. After the method has found 1000 samples that satisfy all the constraints, the samples

are used to calculate samples for the CPT entries of the FNs. The result of these calculations are histograms for the different CPT entries.

Conclusion Some general observations can be made about a large number of the generated histograms. The shape of most of the histograms is similar. The histograms resemble a uniform distribution. One example is Figure 2(a), a histogram generated from the "strict" probability statement set for the Alarm node.

A likely cause for this shape to be present in so many histograms is the use of uniform sampling on the simplex. The generated samples of the joint probability distributions are all as likely to occur, so when calculating samples for the CPTs these samples will also show the same characteristics.

Not all the histograms of the two experiments have a uniform shape. In the smaller family networks consisting out of the nodes Alarm and JohnCalls or Alarm and MaryCalls, the histograms created for the Alarm node did not have a uniform shape. An example is shown in figure 2(b).

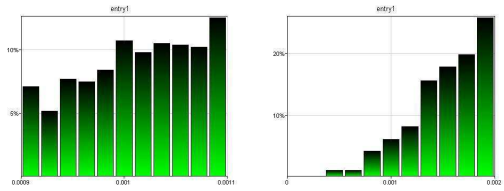


(a) Histogram for entry $P(\neg a \,|\neg b, e)$

(b) Histogram of $P(a)$ with a nonuniform shape

Figure 2: Example histograms

Other examples exist for nonuniform histogram shapes. These differences resulted from using different constraints. The "strict" and "loose" probability statement sets have resulted in different histograms for the same entries. Mostly the differences are in respect to upper and lower bounds, but in a few situations the shape also was different. An example is given in Figure 3. The "strict" version of the histogram is much more precise than its "loose" counterpart. This is due to the constraint $0.0009 \leq P(b) \leq 0.0011$. This is the only constraint in the "strict" statement set that has a



(a) $P(b)$, "strict" statements

(b) $P(b)$, "loose" statements

Figure 3: Different histograms resulting from the difference in the statement sets

direct influence on the the entry $P(b)$. The other statements describing the same family network will have an influence, but not as much as this one. The "loose" statement set has 3 constraints that have a direct influence on $P(b)$:

- $P(b) < 0.002$,
- $P(e) < 0.003$,
- $P(e) > P(b)$.

Since these constraints are less precise than the constraints from the "strict" set this allows for more samples to be valid. Which results in histograms that will vary over a larger range. In this case the exact shape is not easy to explain, it also depends on the other constraints used to sample the family network. And since the "loose" constraints allow for more samples to be valid, it could be possible that 1000 samples is not enough in larger dimensional spaces to get a good distribution of the samples in the constrained volume that satisfies all constraints.

An estimate can be given of the size of the constrained volume created by the two probability statement sets. The exact volumes are hard to calculate so they are approximated by a n-dimensional bounding box. This bounding box is created by the probability bounds calculated by the linear programming process. The lower bounds are subtracted from the upper bounds to get the size of the edges of the box. By multiplying the edges the volume of the bounding box is calculated.

The volumes for the boxes are:

- "Strict" set: $6,62355 \cdot 10^{-27}$,
- "Loose" set: $2,27375 \cdot 10^{-21}$.

This means that the volume of the sample space of the "loose" probability statement set is a factor $343283,1356$ larger than the volume of the "strict" set.

The exact influence of the volume of the sample space is unknown, and the influence of finding more samples on the shape of the histograms has not been investigated any further. For now the differences in the histogram shapes are attributed to the differences in the relevant probability statements.

5 Conclusions

Work on the method is not completely done yet. Some research questions still need to be solved and the created implementation is not completely ready for integration with GeNIe[1]. What has been accomplished is an implementation that proves the concept presented in (Druzdzel & van der Gaag, 1995). It is indeed possible to let an expert state probabilistic information and to use this information to derive 2^{nd} order distributions over all the CPT entries. But experimental results show that the method needs further fine-tuning, considering the results it currently generates. Even with very strictly specified constraints the generated histograms currently do not give very clear advice on what value to choose for a CPT entry.

Furthermore the method has its limitations. A limiting factor for the method is the maximum number of parents a node has in a Bayesian network. A node with a large number of parents, has a very large number of CPT entries, which in turn results in a sample space with a very high number of dimensions. One possible problem connected to sampling in a space with a high number of dimensions is a possible limitation of the accuracy of the implementation of the truncated Dirichlet distribution. Inaccuracies could possibly lead to a higher rejection ratio of samples. Another problem, more likely to occur, is that sets of probability statements provided by the expert can result in very few linear constraints. This can cause the linear programming process to be not very successful in reducing the size of the sample space. If the constrained volume is very small and the remaining sample space is much larger, this will again cause a higher ratio of rejected samples.

Basically what will happen is that it will take a very long time before a valid sample, if it exists, will be found. Since there is currently an upper limit implemented on the number of samples to generate before assuming the existence of a conflict somewhere in the set of constraints, it will become more likely that the method will be aborted before a valid sample is found. The upper limit of samples will be reached more frequently.

Ending with a more positive note, the computation time the method needs to generate results completely depends on the input of the user. Factors that are present independent of user input, for example the requirement that every sample must be a valid probability distribution, have been dealt with as efficient as possible. Every generated sample by default satisfies the axioms of probability, and the ability to sample between probability bounds without the use of rejection sampling has increased the efficiency of generating samples by many orders of magnitude. When the proposed importance sampling scheme is implemented, the efficiency of generating samples may increase even more. It is even likely that, when using importance sampling, the method will perform better in the situation the generated probability bounds have not adequately reduced the sample space. Once valid samples are found the method will increasingly focus on the area where the valid samples were found.

[1]Software for developing Bayesian Networks, developed at the Decision Systems Laboratory

References

Chinneck, J. W. (2002). Discovering the Characteristics of Mathematical Programs via Sampling. *Optimization Methods and Software, 17(2)*, 319–352.

Clemen, R. T., & Reilly, T. (2003). *Making Hard Decisions with DecisionTools®* (2 ed.). Pacific Grove, CA: Duxbury.

Cooke, N. J. (1994). Varieties of Knowledge Elicitation Techniques. *Int. J. Hum.-Comput. Stud., 41*(6), 801–849.

Cooke, R. (1991). *Experts in Uncertainty: Opinion*

and Subjective Probability in Science. Oxford University Press.

Druzdzel, M., & van der Gaag, L. C. (1995). Elicitation of Probabilities for Belief Networks: Combining Qualitative and Quantitative Information. In *Proceedings of the 11th annual conference on uncertainty in artificial intelligence (uai-95)* (p. 141-148). San Francisco, CA: Morgan Kaufmann Publishers.

Fang, K.-T., Geng, Z., & Tian, G.-L. (2000). Statistical Inference for Truncated Dirichlet Distribution and its Application in Misclassification. *Biometrical Journal*, 1053–1068.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan Kaufmann Publishers, Inc.

Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton, NJ: Princeton University Press.

Zadeh, L. (1978). Fuzzy Sets as a Basis for a Theory of Possibility. *Fuzzy Sets and Systems 1*, 3–28.