Delft University of Technology
Faculty of Electrical Engineering, Mathematics & Computerscience

Masters Thesis

A Causal Model for Qualitative Reasoning

Paul Maaskant

June 2006,
Supervising Committee

Drs. Dr. L.J.M. Rothkrantz
Ir. F. Ververs
Dr. Ir. C.A.P.G van der Mast
Dr. Ir. M.J. Druzdzel (University of Pittsburgh)

Abstract

Many problems can be solved by means of qualitative reasoning. When the complexity of a qualitative problem in an uncertain domain becomes nontrivial, the merit of a reasoning framework becomes apparent. Bayesian networks [20, 12] (BNs) provide such a network. BNs provide a concise way to represent a fully joint probability distribution, by exploiting independencies among variables. Each variable is specified by a Conditional Probability Table (CPT). A CPT captures the interaction between a child variable and its immediate predecessors (parents) variables.

However, BNs are more quantitative in nature than qualitative, and thus a translation is required. Furthermore, a fundamental problem of BNs is the exponential growth of a CPT by the number of parent variables for a local probability distribution. A Causal interaction model [18], such as the Noisy-OR model [20, 11], can provide a solution for this type of problem. The contribution of this thesis is the proposal of a causal model, baptized the Probabilistic Qualitative (PQ) model, that provides a reasoning framework for qualitative problems, in an uncertain domain.

In this thesis, I review theory related to the PQ-model. I discuss the most prominent existing solution to our research problem, i.e., CAusal STrenght (CAST) logic [2]. I give a detailed account of the theoretic PQ-model. I describe an experiment, conducted at the University of Pittsburgh, that proves the merits of the PQ-model, and I given an account of the software engineering project that led to the software application that captures the functionality of the PQ-model.

Acknowledgments

The contents of this thesis constitute my graduation project, on which I worked from September 2005 to June 2006. I experienced these months as challenging and exciting. I have learned many things and I am very pleased with what I have accomplished. However, I cannot claim these accomplishments as solely my own. I would like to express my sincere gratitude to the following people:

- Mira van der Velden
- Joris Hulst
- Leon Rothkrantz
- Marek Druzdzel
- Adam Zagorecki
- Mark Voortman
- Tomek Loboda

Furthermore, my stay at the University of Pittsburgh from September 2005 to March 2006 would not have been possible without the financial aid of the following parties:

- My family, Rob & Jolien Maaskant, Miss N.Maaskant & Miss P. van Duin
- Fundatie Vrijvrouwe van Renswoude
- Stimuleringsfonds Internationale Universitaire Samenwerkingsrelaties (STIR), Delft University of Technology

"What is the use of a book," thought Alice, "without pictures or conversations?"

Alice

Contents

1 Introduction

To understand the world around us, we, as mankind, have a tendency to create models of our surroundings. Galileo Galilei understood and predicted the movement of astral bodies by creating his own model. Aristotle provided us with a model for reasoning with model logic, known as syllogisms, which has been used for centuries.

A model need not be correct in order for us to better understand the corresponding phenomena. It is accepted that Newtons universal gravity theory cannot explain all we see, yet it suffices for most many problems concerning apple trees. By its very definition, a model is never infallible. But this does not necessarily prevent it from being useful.

The problem that we are interested in is qualitative reasoning. When we see a kettle of water on a fire, we reason that it the water will start to boil with a certain amount of time. We reason that the water will eventually evaporate and the kettle will be empty. We know this, without having to measure the volume of water, without keeping track of its temperature, and without measuring up the kettle. It is sufficient to know the causal apparent relations.

1.1 Motivation

Now imagine a less trivial problem. Suppose the United Nations plans to invade a country that is oppressed by a fanatic tyrant. They know many things such as: *the tyrant has large monetary funds*, *the tyrant has poor diplomatic relations with neighboring countries*, and *the tyrant is not likely respond to UN sanctions*. By analogy, in our water kettle problem, we would have statements such as: *there water in the kettle*, and *the kettle is on a fire*.

Now suppose the UN knows over 50 of such statements, and they want to know whether it would be a good thing to liberate the country by force, or diplomatic action. The (causal) interactions are less obvious in this case, and uncertainty plays a major role. Therefore, the goal of a model would not necessarily be to know when to invade and when not, but rather, what areas can we influence to maximize the chance of success given a certain action.

This is exactly the problem domain that I am interested in. Problems like these are inherently vague and often unique. It would be difficult, if possible at all, to use machine learning techniques for such a problem. There would simply be no data to learn from. What about an alternative approach? We could use the knowledge of human experts, but often problems are so complex that this is a near impossible task. To clarify, suppose that our model would support over a thousand different situations and similar number of conclusions. We could lock a field expert in a confined cell and keep him there until he has answered all our questions. In addition to the humanitarian problems, it is doubtful that such an approach gives us a useful model for two reasons: (1) due to weariness/boredom the experts answers grow less accurate, and (2) no expert has been in every situation that we can model.

How then should we go about building such a model? *How can we solve qualitative problems of non-trivial complexity in uncertain problems domains?*. This question is, in fact, the research question that drives the research behind this thesis.

1.2 Research Problem

Given an arbitrary problem, we often know many things, however vague. Nice weather increases ice cream sales (but not always). People with beards and mustaches have a kind disposition (but not always). Like in the UN example, we might know dozens of things concerning a particular problem. But how must we incorporate all those vague bits of knowledge into a single useful model?

Probability theory is a common solution to the problem of reasoning under uncertainty. We could translate all we know to a number of variables, e.g., the variable *water in kettle* can have the values [cold, hot, boiling], and build a comprehensive probability distribution, i.e, for every conceivable situation, we assign a probability to each possible outcome. However, like we argued in the previous section, this approach is infeasible if the problem is of non-trivial complexity.

Bayesian networks [20, 12] provide a concise way to represent a fully joint probability distribution, by exploiting independencies among variables. For example, whether or not the water will evaporate is independent of the color of the kettle. A Bayesian network consists of two parts: the qualitative part, consisting of the graphical representation of the network, revealing the independencies between variables, and the quantitative part, consisting of local probability distributions. Such a distribution captures the interaction only the local variables.

Although Bayesian networks take us a step closer to a feasible solution, we are only halfway. It is fairly common that even a group of local variables is too large for a solution to be feasible. Imagine 1 'conclusion' variable, e.g, *water evaporated*, and 10 'situation' variables, e.g, *kettle is round, fire is on, etc.*. Each variable can have values [True, False]. With only 11 variables, we still have to ask for the likelihood of 2048 possible situations.

More formally, a fundamental problem of Bayesian networks is the exponential growth of local probability distributions by the number of local variables. Causal interaction models can reduce the number of required probabilities logarithmically, by assuming Independence of Causal Influence (ICI) [8]. For example, a fire may be caused by (a) arson, or (b) a short circuit, but whether or not (a) is true, has no influence on the value of (b), and vice versa. So instead of listing the probability of a fire for every combination of values for (a) and (b), we only have to list the probabilities of a fire for (a) and (b) separately.

We could see this approach as kind of a black box. We provide the box with a number of parameters. In the case of our example we have 2 parameters: the probability that (a) causes a fire and the probability that (b) causes a fire. These parameters have clear meaning. In contrast, we could also invent some ingenious formula that takes a number of fancy configuration parameters, e.g., $\sqrt{\pi}$, subsequently looks at the input variables, and gives us the desired outcome.

However, for such functions and the meaning of the parameters are not *necessarily* transparent to an expert, and so, learning its parameters from a human expert is not necessarily feasible. An important subclass of ICI models consists of the amechanistic ICI models [10]. The mechanism in an amechanistic model is extended by additional assumptions, which, in short, ensures a clear meaning for the domain expert and effectively solving this problem. By we mean that, the parameter is the answer to a simple question.

So out problem may be solved an amechanistic ICI model, but we still figure out which one. We are looking for are quantitative model, but the parameters have a good fit to qualitative problems. Many existing ICI models can only handle problems within a limited scope. For example, in the Noisy-OR model, parent variables can be interpreted as representing 'promoting causes,' but its not possible to represent a mix of positive and negative influences within the same model.

1.3 Goals and objectives

The goal of my research is to introduce a theoretical model that is useful in qualitative modeling under uncertainty. Like I pointed out before, we can choose an arbitrary function based on a number of subjective rules of thumb, and simply claim that it is applicable for qualitative problems. Since qualitative problems are inherently vague, this is a tempting caveat. To ensure the validity of my proposed model, I have come up with a set of guidelines.

The first guideline is that the proposed model is *built on propositional logic.* Propositional logic provides a solid platform for any model that captures reasoning. A good example of this is the Noisy-OR model. While it is true that propositional logic might not be adequate to model every problem, the author believes that, when combined with probability theory, its application is broad enough to serve as a bounding framework for the proposed model. The most important consequence is that the proposed model is not susceptible to so-called 'hand waving.' Furthermore, any logical relationship can be captured by only 'AND,' 'OR,' and 'NOT' operators. Thus, by bounding the proposed model to the same restrictions, using elemental building blocks, the I hope to create a transparent framework.

The second guideline is that the proposed model belongs to the class of *amechanistic ICI models.* This will ensure that the mechanism that converts the elicited parameters into a set of probability distributions is meaningful to the expert. This, in turn, will ensure the feasibility of acquiring the parameters for the proposed model. I believe that this will result in networks that are able to more accurately represent qualitative problems, more so than networks built conventional Bayesian networks, elicited from the human expert. This claim is put to the test in Chapter 6.

The third guideline is that the proposed model must apply to a *broad class of qualitative problems.* This means that it has to incorporate some way of combining opposing influences in a non-ambiguous manner. For example, a fire may be caused by a match, and may be inhibited by rain. To combine both influences is a non-trivial problem. Also, the model must be able to represent interactions which can be classified as conditional, rather than causal. For example, for a plant to grow we need both water *and* light. Neither water, nor light alone will be enough to ensure a flourishing plant.

Finally, I require that the proposed model is *empirically validated.* I assume propositional logic and probability theory to be indisputably mathematically correct. However, the goodness of fit of the proposed model to real-life qualitative problems can only be validated empirically.

The objectives of my research are:

- Defining a theoretical causal model, that is in particular applicable to qualitative problems, obeying the guidelines as described above.
- Validating the proposed model through empirical research.
- Designing, implementing and testing the proposed model as part of the SMILE/GeNIe software.
- Creating a non-trivial, example network with the proposed model to demonstrate practical significance.

Section 10.1 discusses to what extend these objectives were satisfied, and what room for improvement is left for future endeavors.

1.4 Thesis Overview

The remainder of this thesis is composed as follows.

Chapter 2 introduces Bayesian networks, concentrating on those aspects of Bayesian networks that bear relevance to later chapters. Chapter 3 introduces causal interaction models. These models address the problem of the exponential growth of local probability distributions with the number of local variables. One particular subclass of causal interaction models are amechanistic models under assumption of independence of causal influence, which we will discuss in more detail. We discuss several popular canonical models, as proposed in contemporary literature.

Chapter 4 will introduce Causal Strength (CAST) logic [2], which has been implemented in the Situation Influence Assessment Module (SIAM) software. CAST logic is an alternative and existing solution to our research problem, i.e., it provides a framework for modeling qualitative problems in an uncertain domain. SIAM is existing software used for qualitative reasoning by the Air Mobility Command (AMC). We give a concise overview of CAST logic and several examples pointing out the strengths and caveats of this model. In addition we briefly discuss Fault Tree theory, because like CAST, it bears sigifican relevance to our proposed model.

Chapter 5 introduces the theoretical contribution of my research, baptized the Probabilistic Qualitative (PQ) model. The PQ-model is a quantitative model, which is particularly applicable to qualitative problems. The mechanism behind the PQ-model allows the parent variables to represent different types of qualitative interactions. We introduce the theory, applicability, and driving assumptions of this model.

Chapter 6 discusses an experiment, conducted at the University of Pittsburgh. The goal of this experiment was to validate the theoretic PQ-model, by showing that this model outperforms a conventional Bayesian approach in a network which is built using knowledge elicitation from domain experts. Therefore we measured not how well the theoretic PQ-model the an arbitrary phenomena, but how well it captured the knowledge of the experts.

Chapter 7 discusses the system requirements and system design design for a software application, including the functionality of the PQ-model within the context of qualitative reasoning. We state the functional and nonfunctional requirements for such a system and go over the relevant use cases. We provide an environment independent system design and in addition we provide a rudimentary design for the Graphical User Interface (GUI) of this application. Chapter 8 discusses how we adjusted our system design in order for the application to be part of the GeNIe and SMILE software. Furthermore we review our testing activities for system verification and validation.

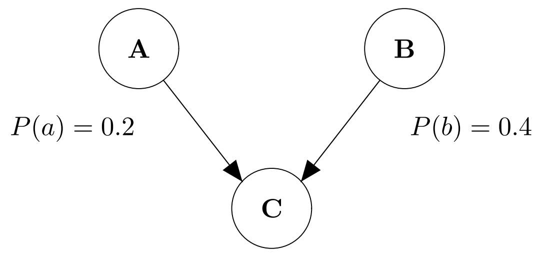
Chapter 9 discusses an example of a network, representing a qualitative problem, modeled by the software that we designed and implemented. This problem will provide the reader with a *proof of concept* concerning the application of the PQ-model. In Chapter 10, this thesis concludes with an evaluation, an overview of my research and gives some direction for future research on the topic of qualitative reasoning with Bayesian networks.

The remainder of this thesis is written in the first person plural. Please note that this gives no indication whether or not the described work was my own, mine in cooperation with others, or not mine at all. Section 10.3 gives a concise summary of the research described in this thesis, and my (the authors) part in particular.

2 Bayesian Networks and Causality

As a starting point in our search for a solution, we have Bayesian network theory. Although we have briefly mentioned Bayesian networks in our introduction, this chapter will give a more formal introduction. Once we have attained a sufficiently thorough understanding of Bayesian network theory, we will adapt the conventional Bayesian network to find our solution, i.e. the problem of modeling a non-trivial qualitative problem in an uncertain domain. However, let us not get too hasty, and deal with first things first.

Within the last 20 years the probabilistic approach has proved to be the most popular approach for reasoning under uncertainty. The credit for this popularity can be attributed mainly to Bayesian networks [20]. The Bayesian network combines qualitative information in the form of a directed acyclic graph (DAG) with quantitative information in the form of conditional probability tables (CPTs). These CPTs can be elicited from both data and/or human experts. By exploiting the independencies among variables, Bayesian networks can be used to concisely define a full joint prob-



$$P(c|ab) = 0.9$$
$$P(c|\overline{a}b) = 0.7$$
$$P(c|a\bar{b}) = 0.5$$
$$P(c|\overline{a}\bar{b}) = 0.2$$

Figure 2.1: A simple Bayesian network.

ability distribution, while reducing the problem complexity. Bayesian networks are very popular in practice, and have been applied to may different problems, including medical diagnostics [19], speech recognition [13], facial expression recognition [17] and credit rating [7].

For the remainder of this thesis, we will use uppercase letters to denote random variables (e.g., X) and lowercase letters to denote their state (e.g., x). Because we deal only with discrete binary variables, we denote the range of X as $Range(X) = \{x, \overline{x}\}$. Bold uppercase letters will denote a set of random variables (e.g., $\mathbf{X}$) and bold lowercase letters will denote values for a set of random variables (e.g., $\mathbf{x}$). We use $P(X)$ to denote the probability distribution over a variable X. We use $P(X|Y)$ to denote the probability distribution over a variable X, conditioned on a variable Y.

2.1 Bayesian networks

A Bayesian network is a DAG that represents a set of variables. The variables are represented by the nodes in the graph. The dependencies among variables are represented by directed arcs between nodes. Figure 2.1 shows a simple BN. Variable C is dependent on both variable A and B, and is referred to as the 'child' variable of variable A and B. Variable A and B are referred to as the 'parent' variables of variable C. A set of variables that consists of a child variable and its parent variables is referred to as a 'family'.

In the case that a variable does not have any parents, the probability that the variable is in a particular state is given by an unconditional (marginal) probability. Otherwise, the probability that a variable is in a particular state is dependent on its parents, as defined by the conditional probability table (CPT). The CPT defines a probability distribution for a variable for each possible

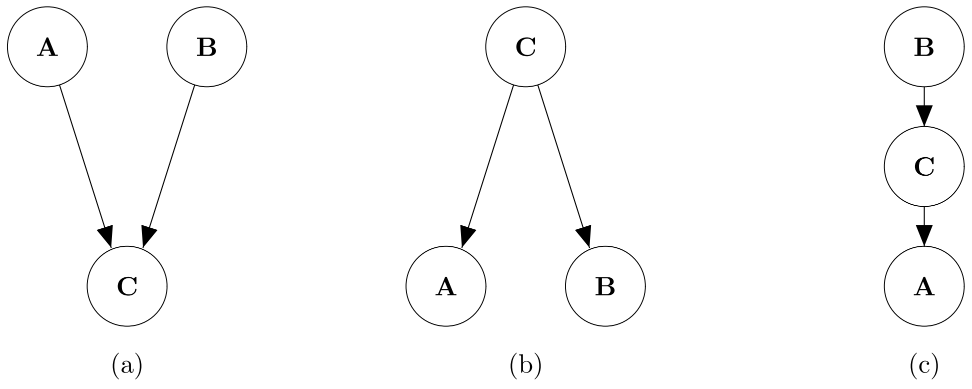


Figure 2.2: Three types of connections:(a) converging, (b) diverging, and (c) serial.

configuration of the states of the parent variables. The CPT for variable C is shown in Figure 2.2. Please note that because the variables are all binary, i.e., $P(x) = 1 - P(\overline{x})$, it suffices to give only the probability that a variable is in (either) one of its states.

A fundamental problem of CPTs is in the exponential growth in the number of parent variables. For nodes with over 10 parents, not uncommon in practical models, eliciting the CPT from human experts is practically infeasible. Learning a model from a dataset may also be hard, as there are typically not enough data to learn every distribution in a CPT reliably. Learning parameters from a data set is beyond the scope of this thesis. In chapter 3 we pay more attention to elicitation from human experts.

2.2 Conditional Independence

In Figure 2.2, the absence of an arc between variables A and B indicates that they are conditionally independent of each other. In other words, $P(a|bc) = P(a|c)$. To see why this is important, imagine you have to define a fully joint probability distribution, with n variables, each with s possible states. This means you would have to provide s^n probability distributions. Now imagine that a variable is dependent on at most m parent variables. You now have to provide no more than $n \cdot s^m$ probability distributions, i.e., a number of distributions polynomial, rather than exponential, in the number of variables. One of the most important traits of BNs is that they provide us with a graceful way to exploit conditional independence .

Two variables that are not directly connected are not necessarily conditionally independent. A variable is conditionally independent of all other variables in the network, given its parents, its children, and its children's parents, i.e., given its Markov blanket. A general topological criterion for deciding whether or not two nodes are conditionally independent is called D-separation.

D-separation is best explained by defining the three types of connections, shown in Figure 2.1. Variables A and B have a converging connection if they are both parents of variable C. Variables A and B have a diverging connection if they are both children of variable C. Variables A and B have a serial connection connection if one of them is a parent to variable C, while the other is a child of variable C.

Variables A and B are D-separated, i.e., conditionally independent, if for every path between them, there is an intermediate variable A such that (1) the connection is diverging or serial, and A is known, or (2) the connection is converging and neither A, nor any of its descendants are known.

2.3 Inference in Bayesian networks

If the CPT of each node in the network is known, we can calculate any conditional probability, using Bayesian inference. We need only Bayes' Theorem, given in Equation 2.1, and the chain rule of multipication, given in Equation 2.2:

$$P(y|x) = \frac{P(x|y) \cdot P(y)}{P(x)} \tag{2.1}$$

$$P(x) = P(x|y) \cdot P(y) + P(x|\overline{y}) \cdot P(\overline{y}) \tag{2.2}$$

For example, suppose we want to know the probability $P(a|c)$ in Figure 2.2. Please note that $P(ab) = P(a) \cdot P(b)$, due to their mutual independence. First, we calculate P(c) using the chain rule of multiplication:

$$\begin{aligned} P(c) = P(c|ab) \cdot P(ab) + P(c|\overline{a}b) \cdot P(\overline{a}b) + P(c|a\overline{b}) \cdot P(a\overline{b}) + P(c|\overline{a}\overline{b}) \cdot P(\overline{a}\overline{b}) \\ = 0.9 \cdot 0.2 \cdot 0.4 + 0.7 \cdot 0.8 \cdot 0.4 + 0.5 \cdot 0.2 \cdot 0.6 + 0.2 \cdot 0.8 \cdot 0.6 \\ = 0.452 \end{aligned}$$

Next, we calculate $P(c|a)$, again using the chain rule of multiplication:

$$\begin{aligned}P(c|a) &= P(c|ab) \cdot P(b) + P(c|a\bar{b}) \cdot P(\bar{b}) \\ &= 0.9 \cdot 0.4 + 0.5 \cdot 0.6 \\ &= 0.66\end{aligned}$$

Finally, now that we have calculated $P(c)$ and $P(c|a)$, we can use Bayes' Theorem to calculate the conditional probability $P(a|c)$:

$$P(a|c) = \frac{P(c|a) \cdot P(a)}{P(c)} = \frac{0.66 \cdot 0.2}{0.452} \approx 0.292 \tag{2.3}$$

Although this is a trivial example, probabilistic inference in general is a worst-case NP-hard problem [3]. Although approximate probabilistic inference to any desired precision is worst-case NP-hard as well, it is the only feasible alternative to exact inference. Stochastic Sampling Algorithms form a prominent subclass of approximate inference methods, in which a (large) number samples are propagated through the network in order to estimate posterior distributions. Some current work on inference can be found in [30]. Heckerman published a comprehensive tutorial on Bayesian networks [9].

2.4 Causality

In a philosophical sense, the term causality captures the concept of a cause-and-effect relationship. Yet it is hard to define a generally accepted definition of causality. Many statisticians will even go so far as to dismiss causality altogether, attributing everything to correlation. Nevertheless, causality is a very useful concept. One of the main traits of Bayesian networks is that they capture causality with such ease. This is desirable for multiple reasons.

First of all, a causal relation is directed, as opposed to a correlated relation between variables. A Bayesian networks is a Directed Acyclic Graph (DAG), which is an essential property if we want to update beliefs, i.e., if the graph is not acyclic, the process of updating beliefs might result in endless iteration. Hence, assuming causality can make a problem feasible.

Second, humans prefer to think in terms of causality rather than correlation, even when causality is provably not the case. For example, it is common belief that 'lighting causes thunder,' while lightning and thunder are in fact bound by a common cause: an electric discharge. In psychology, *Attribution theory* is the theory concerning how people explain individual occurrences of causation.

Hence, causality provides Bayesian networks with an appealing intuitive quality.

Third, assuming causality can often reduce the complexity of a modeled problem. If we want to model the relation between the variable 'lightning' and the variable 'thunder', it is considerably simpler for us to assume that 'lighting causes thunder', rather than including a third variable 'electric discharge,' representing the mutual cause. Although the extended model is more correct, its needlessly complex, while the gain in accuracy (if any) is negligible.

Finally, we can exploit the concept of causality in local probability distributions, by assuming that multiple 'causes' can produce the same effect, *independent* of each other.

2.5 Calculating CPT Distances

In Chapter 6 we are interested in determining the difference between two CPTs. For example, one CPT might represent the set of *estimated* probability distributions, while the other CPT represents the set of *actual* probability distributions. The purpose of such a comparison would be to determine the accuracy of the estimated CPT.

We briefly discuss 3 measures that determine the distance between two probability distributions: the Euclidean measure, Hellinger's measure, and the Kullback-Leibler (KL) measure. Let us assume that we have probability distributions P and Q, which have an equal, discrete number of states (N). The distance measures are defined as:

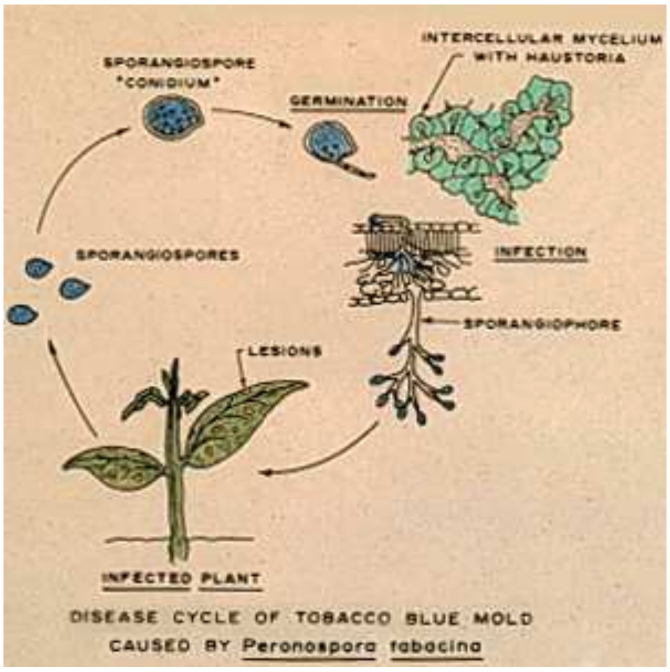
$$d_{Euclidean} = \sqrt{\sum_{i}^{N} (p_i - q_i)^2} \ , \tag{2.4}$$

$$d_{Hellinger} = \sqrt{\sum_{i}^{N} (\sqrt{p_i} - \sqrt{q_i})^2} \ , \tag{2.5}$$

$$d_{Kullback-Leibler} = \sum_{i}^{N} p_i \ log\left(\frac{p_i}{q_i}\right) \ , \tag{2.6}$$

in which the index i indicates the state of distribution P (or Q).The Euclidean distance can be interpreted as the absolute distance between two points in an N dimensional space. The Hellinger and KL measure, are similar, but are more sensitive to (give a higher penalty to) extreme values, i.e., probabilities close to 1 or 0. Such a property might be desirable in cases where a CPT has extremely low values, yet small differences are important.

Finally, we note that a CPT is not a probability distribution, but rather a *set* of probability distributions. In other words, the child has a single probability distribution for each configuration of parent states. When comparing two CPTs as a whole, it is a common approach to take the average of all distances represented the first CPT, and compare it to the average distance of the other CPT. However, in some cases a weighted approach is more desirable, e.g., when a particular probability distribution is more relevant than the next, given the problem domain. The choice of which measure and how those measures are combined within a CPT, depends largely on the problem at hand.


SPORANGIOSPORE "CONIDIUM"
GERMINATION
INTERCELLULAR MYCELIUM WITH HAUSTORIA
SPORANGIOSPORES
INFECTION
SPORANGIOPHORE
LESIONS
INFECTED PLANT
DISEASE CYCLE OF TOBACCO BLUE MOLD CAUSED BY Peronospora tabacina

3 Causal Models In Bayesian Networks

Now that we have attained some understanding of Bayesian networks, and how they might help us find a solution to our research problem, we also understand some of the problems that must be overcome. In Bayesian networks, a fundamental problem of CPTs is in the exponential growth in the number of parent variables. For nodes with over 10 parents, not uncommon in practical models, eliciting the CPT from human experts is practically infeasible. Learning a model from a dataset may also be hard, as there are typically not enough data to learn every distribution in a CPT reliably.

Causal interaction models provide a solution to this problem. At a rudimentary level, a causal model can be viewed as a black box: we provide the box with a relatively small number of parameters and it magically produces some output. However, the range of available causal models is extensive, and there are many subclasses. If we want to devise our own causal model, we first have to gain some insight into the relevant properties and assumptions of these models.

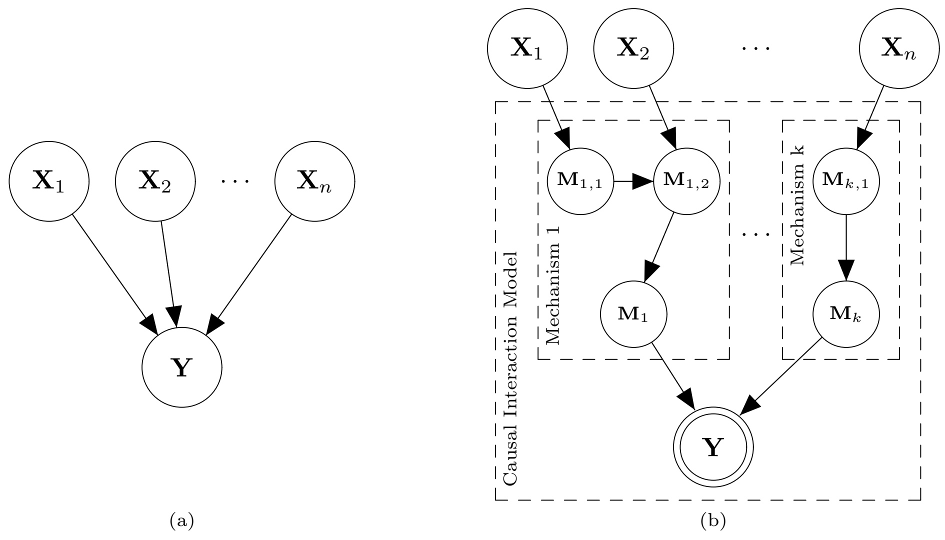


Figure 3.1: Replacing the CPT (a) with a causal interaction model (b).

3.1 Causal Interaction Models

Meek and Heckerman [18] formally introduced *causal interaction models*, replacing the CPT for a local probability distribution with a causal mechanism of an arbitrary form. Instead of defining the probability distributions directly, which has a cost exponential with the number of parents, we define the parameters for the mechanism, which (usually) has a cost linear with the number of parent variables. In turn, the parameterized mechanism defines the CPT. Figure 3.1 shows an example of a causal interaction model.

A mechanism consists of an arbitrary number of mechanism variables which obey the following four restrictions: (1) a mechanism variable can be the child of both parent variables outside the mechanism, and other mechanism variables; (2) there are no cyclic connections; (3) each mechanism has one *distinguished* mechanism variable, (4) that is a parent to effect variable Y. The effect variable Y is given by a deterministic function (double edged node in the network) in the form $Y = f(M_1, ..., M_k)$.

In the model domain, a mechanism is like a black box, i.e., mechanism variables are always hidden. Consequently, we have to make assumptions about their parameterization and interaction. This is, in fact, the main drawback of causal interaction models: causal interaction models lack practical significance, because mechanism variables lack semantic meaning in the modeled domain. In other words, because a mechanism variable does not *mean* anything, it is difficult to assign a useful value.

3.1.1 Independence of Causal Influence

Heckerman and Breese [8] introduced a subclass of causal interaction models, namely *causal independence models*, later referred to as models under assumption of *independence of causal influence* (ICI). In contrast to causal interaction models, ICI models have great practical significance.

ICI models assume conditional independence between the mechanisms. This implies one important thing: each parent variable (outside the ICI model) has associated with it only one single mechanism. Because of this, each mechanism need only contain one single mechanism variable, i.e., if there are more, we can always marginalize them into one single variable.

Due to the assumed conditional independence of mechanisms, elicitation of the mechanism parameters is greatly facilitated, because for each parameter we know exactly which parent variable (outside the model) is associated with it. However, there is still no guarantee that the parameters have semantic significance, i.e., actually *mean* something. CAusal STrenght (CAST) logic, which we will discuss in Chapter 4 belongs to the class of ICI models.

3.1.2 The Amechanistic Property

Amechanistic causal independence models [10] are a subclass of ICI models that address this problem by making two additional assumptions: (1) each variable has a 'typical' state, referred to as the *distinguished* state. This is usually the default state for that variable, such as 'OK' for a car diagnostic model, or 'healthy' for a medical diagnostic model; (2) if all parent variables are in their distinguished states, then so is the child variable.

Assumption (2) has one important consequence. It enables us to directly elicit the causal influence of a parent variable X_i by simply asking for $P(y|\overline{x}_1, ..., x_i, ..., \overline{x}_n)$, with the question:*'What is the probability that Y is in its non-distinguished state, given that all parents are in their distinguished states, except parent X_i.'* In other words, assumption (2) basically guarantees that there is no interference from variables that are in their distinguished state, and all the observed behavior of Y can be attributed to X_i. In other words, *if a parent variable is in its distinguished state, the child variable is effectively independent of that parent.*

Consequently, the parameters for an amechanistic ICI model can be obtained by asking simple and clear questions, and so, such a model is particularly suited for parameterization by human experts. The most popular causal models, i.e., the Noisy-OR and Noisy-MAX models, belong to the class of amechanistic ICI models.

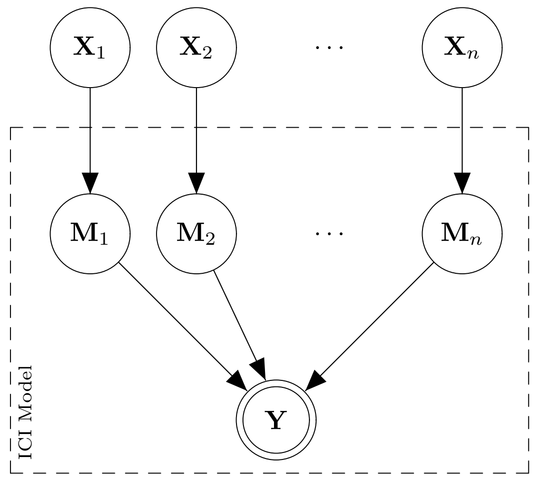


Figure 3.2: Structure of an ICI model.

3.2 The Noisy-OR Model

he Noisy-OR [20, 11] model is a non-deterministic extension of the deterministic OR relation. Its variables, e.g., X, are binary and can be either 'present,' i.e., the non-distinguished state, denoted as x, or 'absent,' i.e., the distinguished state, denoted as $\overline{x}$. Each 'present' parent event can independently produce the child effect with a present, i.e., certain probability. It is, in fact, not required that the distinguished state is the 'absent' state, it might just as well be 'present,' although this is uncommon in practice. The Noisy-OR's amechanistic property assumes (assumption 2) that if none of the parent variables $X_1, ..., X_n$ are 'present,' then neither is the child variable Y:

$$P(\overline{x}_1, ..., x_i, ..., \overline{x}_n) = 1 \ . \tag{3.1}$$

We define the probability that x_i produces y as z_i:

$$P(y|\overline{x}_1, ..., x_i, ..., \overline{x}_n) = z_i \ . \tag{3.2}$$

Because of the ICI assumption, i.e., the influence of parent variables is conditionally independent, we define the probability that the set of parent variable configurations $\mathbf{x}$ produces y as:

$$P(y|\mathbf{x}) = 1 - \prod_{X_i = x_i} (1 - z_i) \ . \tag{3.3}$$

Repeating this process for each possible parent configurations $\mathbf{x}$ gives us the CPT.

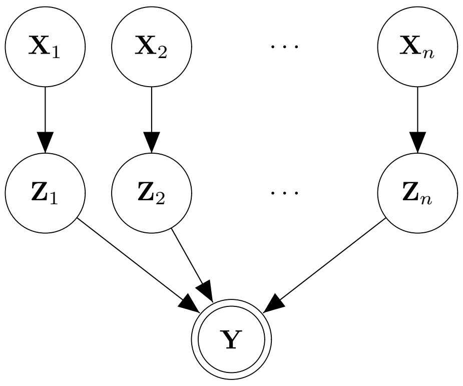


Figure 3.3: Noisy-OR Bayes net with multiple causes and a single effect.

3.2.1 The Leaky Noisy-OR Model

The noisy-OR model is only applicable if we assume the modeled causes to be collectively exhaustive i.e. the child effect can only be produced by the modeled cause events. This is a very rigid assumption and prevents us from applying the noisy-OR model to real world situations. For instance, the child event could be produced by some cause that we have not included in our model. However, it is inconceivable to include every possible cause. Instead we define a leak probability, which is the probability that the child effect is produced by a cause not included in the model. The leak probability can also be interpreted as the combination of all probabilities of all the causes that are not an explicit part of the model, but are able to produce the child event. We denote the leaky node as Z_L and the probability that Z_L is observed as c_L.

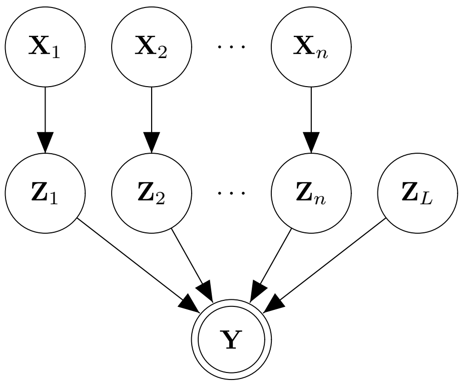


Figure 3.4: Leaky noisy-OR Bayes net with multiple causes and a single effect.

Now that we have introduced the possibility of a leak, we must also revise how we apply inference. Equation 3.4 shows how to incorporate the leak probability:

$$P(y|\mathbf{x}) = 1 - (1 - z_L) \prod_{X_i = x_i} (1 - z_i) \tag{3.4}$$

Including the leak probability may enhance the applicability of the noisy-OR to real world situations, however, it also introduces a new ambiguity. This difficulty presents itself when asking an expert: (1) *what is the probability of Y occuring, given that none of the parents, except X_i, are observed.* This question essentially queries the probability of event Y occuring as a consequence of X_i *or* as a consequence of the leak node Z_L. Because the leak probability is often not explicit, this question is difficult to answer for an expert. If we want to isolate the leak probability from our query, we have have to ask: (2) *what is the probability that X_i causes Y when none of the other parents events are observed.* In both cases we have to query the leak probability separately.

The two questions correspond to two slightly different parameterizations. Question (1) corresponds to Henrion's parameterization [11], which assumes that the queried parameters include the influence from causes not included in the model. Question (2) corresponds to Diez' parameterization [ref], which assumes that the queried parameters do not include influence from causes not included in the model. Henrion's parameters correspond better to learning from data, i.e., it captures the correlation between events. Diez' parameters correspond better to learning from expert knowledge as it captures the causality between events. Research has shown that the storage of human knowledge follows causal, rather than correlational, as briefly discussed in Section 2.4. The relation between Henrion's parameters (q_i) and Diez' parameters (z_i) is given by Equation 3.5:

$$z_i = 1 - \frac{1 - q_i}{1 - z_L} \tag{3.5}$$

Note that if the leak probability converges to zero, both parameterizations become the same. An additional practical disadvantage of Henrion's parameters is encountered when a model is extended to include additional causes. We have to assume that before the extra causes were modeled explicitely, those causes were implicit to the model and contributed to the leak probability implying that the leak probability decreases. Because all parameters include influence from the leak probability, all parameters in the model have to be adjusted when the leak probability changes.

3.2.2 The Recursive Noisy-OR Model

A very recent extension of the noisy-OR model has been proposed in [ref] and is introduced as the recursive noisy-OR (RNOR) model. The additional merit of the RNOR model is that it allows an inhibitor to be assigned to a *combination* of two or more cause events, as opposed to the traditional noisy-OR model, which only allows inhibitors to be assigned to the separate cause events. In other words, the recursive noisy-OR model does not require the cause events to be independent.

The RNOR allows the modeler to capture additive synergy between the cause events. Additive synergy occurs when the presence of a one cause variable increases or reduces the influence of another cause variable. Consider a situation where an expensive vase is placed upon a small table. In order to make the vase fall of the table (effect) we can either hit it with a baseball (cause 1) or rock the table (cause 2). However, if we simultaneously hit the vase with the baseball and rock the table, the imbalance of the vase caused by rocking the table will increase the chances of the baseball knocking over the vase.

Because of the recursive nature of this model, it is convenient to define the conditional probability $P^R(\mathbf{x})$ as a function of the set of observed parents $\mathbf{x}$. Inference with the RNOR model can be achieved by applying Equation 3.6:

$$P^R(\mathbf{x}) = \begin{cases} P^E(\mathbf{x});\ \text{if the expert provides this information} \\ 1 - \prod_{i=0}^{n-1} \frac{1-P^R(\mathbf{x}\backslash\{x_i\})}{1-P^R(\mathbf{x}\backslash\{x_i\, x_{(i+1)mod n}\})};\ \text{otherwise} \end{cases} \tag{3.6}$$

The symbol $\backslash$ denotes set substraction and $(a)mod(b)$ represents the modulas function. To conclude our example, Equation 3.6 allows us to assign a specific probability $P^E(\mathbf{x})$ of the vase being knocked over if the two causes should occur simultaneously.

3.3 The Noisy-AND Model

In the deterministic AND model, the parent events represent conditions rather than causes. All of the parent events need to be observed in order for the child event to occur. The noisy-AND model allows two kinds of noise. The first type of noise allows each condition, when satisfied, to have some probability (c_i) of not "counting", effectively preventing the child event from occurring. The second type of noise allows for the possibility that even when a particular condition is not satisfied, there is some probability (s_i) that it will be subsituted by an influence excluded by the model:

$$P(M_i|X_i) = \begin{cases} c_i & \text{if } X_i = x_i \\ s_i & \text{if } X_i = \overline{x}_i \end{cases}, \tag{3.7}$$

Inference can be obtained by Equation 3.8:

$$P(+y|\mathbf{x}) = \left(\prod_{X_i=x_i} c_i\right) \cdot \left(\prod_{X_i=\overline{x}_i} s_i\right) \tag{3.8}$$

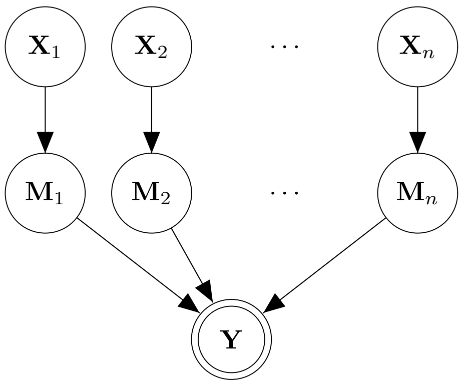


Figure 3.5: Noisy-AND Bayes net with multiple conditions and a single effect.

Each 'present' cause ($X_i = x_i$) contributes a factor c_i and each 'absent' cause ($X_i = \overline{x}_i$) contributes a factor s_i to the product. If we do not wish to allow for the possibility that an absent condition can be substituted, we define $s_i = 0$. If we do not wish to allow for the possibility that a present condition 'does not count,' we define $c_i = 1$.

3.3.1 The Leaky Noisy-AND model

Besides assigning a probability of inhibition to each condition separately, the leaky noisy-AND model allows a probability of general inhibition. This translates as the probability that the child event will not occur even though all the conditions are satisfied and uninhibited. Equation 3.9 shows us how to apply inference:

$$P(+y|\mathbf{x}) = (1 - c_L)\left(\prod_{X_i = x_i} c_i\right) \cdot \left(\prod_{X_i = \overline{x}_i} s_i\right) \tag{3.9}$$

Note that in Equation 3.9 the leak parameter c_L can be absorbed by the parameters for any particular link. For example we could replace the parameter values c_i for condition X_i by $(1 - c_L) \cdot c_i$. In this case, Equation 3.8 and Equation 3.9 are equivalent. However, from a point of view of knowledge elicitation, we benefit from their distinction. By "mixing up" the values of the parameters, the parameters loose clear meaning. Without clear meaning, we cannot obtain the parameterization from domain experts.

3.3.2 Simple Noisy-AND Model

We could have a situation in which the domain expert has a general notion of the probabilities of the child event occuring when all or some conditions are satisfied, but lacks knowledge detailed enough to specify c_i and s_i for each separate condition. In this case we benefit from the simplistic extension of the noisy-AND model which only requires one or two parameters: parameter c to indicate the probability that the child event occurs when all of the conditions are satisfied, and parameter s to indicate the probability that the child event occurs when some of the conditions fail.

The CPT for this model can be obtained by Equation 3.10:

$$P(y|\mathbf{x}) = \begin{cases} c & \text{if } \forall X_i(X_i = x_i) \\ s & \text{otherwise} \end{cases} \tag{3.10}$$

3.4 Summary

Independence of Causal Influences (ICI) models [9], provide a solution by assuming that the parent variables cause the effect independently of each other. The effect of this assumption is such that the number of required parameters is linear, rather than exponential, in the number of parent variables.

Amechanistic ICI models [10] are a subclass of ICI models in which every variable has a distinguished state, and if all parent variables are in their distinguished states, then the child variable is in its distinguished state. This property allows us to obtain the model parameters from experts by asking simple questions. We believe that the unquestionable popularity of the Noisy-OR [20][11] and Noisy-MAX [5] models is in part due to their amechanistic property.

Humans have a disposition of modeling the observed world as a system of cause and effect relations. The noisy-OR model can intepreted as a model dat mimics a non-deterministic causal relation between a number of causes and a single effect. Therefore, the noisy-OR model represents an interaction intuitively appealing, and tends to fit models conveyed by human experts regardless of the knowledge domain in which we seek application.

The noisy-AND model is not as popular as the noisy-OR model. Possibly this is due to the way in which we model our reality. Intuitively, it seems unlikely that a certain child event can only be produced by a single, specific set of parent events. This assumption seems too restrictive to approximate reality. Instead we are prone to argue that any given child event can be produced by a number of combinations of parent events.

Although the concepts of inhibition and substitution have clear semantics, this does not necessarily mean that it mitigates the process of knowledge elicitation. Note that the noisy-AND described in section 3.3 does not have the amechanistic property (!). This is due to the fact that there is no guarantee that the child is in its distinguished state, given that all parents are in their distinguished states, regardless of which state we choose to be the distinguished state.

To illustrate how the lack of the amechanistic property in the discussed Noisy-AND model results in problems during elicitation, consider asking the question: *what is the probability of the child event to occur when all conditions are observed to be 'present,' except condition X_i?* To answer this question, the expert has to consider not only the probability that X_i is substituted, but also the probability that none of the other observed conditions are inhibited, i.e., that all other probabilities 'count.' Consequently, the answer we will get will be the product of all those factors, and we are unable to marginalize the effect of any individual condition. Because the interaction (mechanism) between inhibitors and substitutions is hidden from an outside observer, there is no way for the expert to give a useful answer.

An excellent overview of local probability distributions and related issues can be found in [32]. An good overview of a wide range of causal models can be found in [6]. On a final note, we remark that, although causal models may be the most common solution to exponential CPT growth, there are other solutions. Das [4] proposed a solution that generates a CPT from only a handful very well known parent variable configurations. However, because we want to completely avoid models that are based heuristics and subjective functions, we will seek our solution in a amechanistic ICI model.

4 An Alternative Solution: CAST

Like we said before, a causal model is like a black box and this box may behave in an infinite number of ways. In the previous chapter, all causal models relied on propositional logic. But this is not required. We could also take an arbitrary function, e.g., a standard normal distribution, find some way to translate the input variables to a mean and deviation, and voila (!), we have our output.

In this chapter we will discuss CAusal STrengh (CAST) logic [2]. CAST uses a function, rather than logic, to define local probability distributions. We want to alert the reader that CAST is an alternative and existing solution to our research problem. Although CAST is popular, especially in military logistic areas, it does suffer from some inherent parameterization problems. Therefore, in order to ensure that the solution that we propose in Chapter 5, i.e., the Probabilistic Qualitative (PQ) model, does not suffer from the same ailments, we will carefully go over the CAST model, and try to learn from its mistakes.

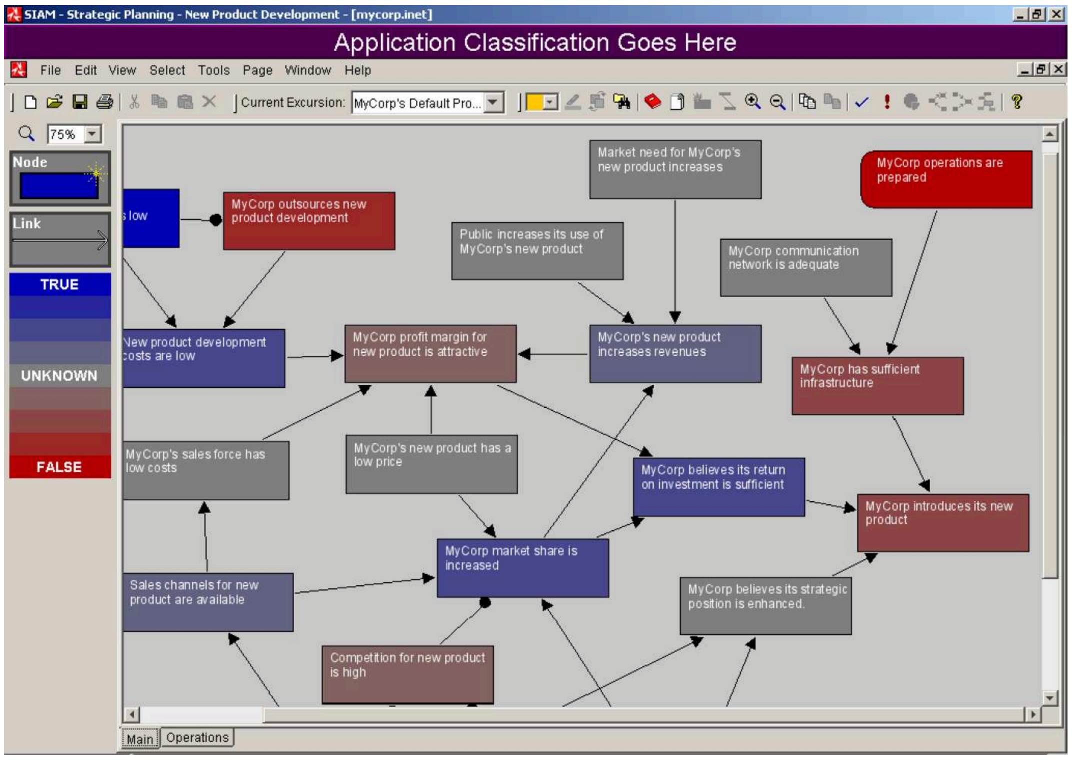

SIAM - Strategic Planning - New Product Development - [mycorp.inet]
Application Classification Goes Here
File Edit View Select Tools Page Window Help
Current Excursion: MyCorp's Default Pro...
75%
Node
Link
TRUE
UNKNOWN
FALSE
Market need for MyCorp's new product increases
MyCorp operations are prepared
MyCorp outsources new product development
Public increases its use of MyCorp's new product
MyCorp communication network is adequate
New product development costs are low
MyCorp profit margin for new product is attractive
MyCorp's new product increases revenues
MyCorp has sufficient infrastructure
MyCorp's sales force has low costs
MyCorp's new product has a low price
MyCorp believes its return on investment is sufficient
MyCorp introduces its new product
MyCorp market share is increased
Sales channels for new product are available
MyCorp believes its strategic position is enhanced.
Competition for new product is high
Main
Operations

Figure 4.1: The Situational Influence Assessment Model (SIAM) software.

In addition, we will briefly discuss Fault Tree theory. Although fault trees do not provide an alternative solution to qualitative problems under uncertainty, it has to a certain level inspired the PQ-model, especially the concept of solving problems through interaction of logical operations.

4.1 Causal Strength logic

The next reasoning framework we discuss is an integral part of the Situational Influence Assessment Model (SIAM) software, which is a product of the Science Applications International Corporation (SAIC). SIAM uses the Causal Strength (CAST) [2] model to generate a Bayesian network, given only a relatively small number of expert defined parameters. In the SIAM software, experts assign parameters, in the form of causal strengths, rather than conditional probabilities. Given these parameters, the CAST algorithm is able to generate a CPT for each family of variables.

The main advantage of the CAST model is that parent variables in the same family are allowed to have opposing, i.e., a mixture of positive and negative, influences. Parents are allowed to have influence on the child variable in both their states, i.e., a parent can have a causal influence when 'absent' or/and when 'present.' The main disadvantage of the CAST model is the lack of clear semantics for the parameters, as we discuss in Section 4.1.3.

4.1.1 The Algorithm

In the CAST model, variables represent statements that can be either 'true' or 'false.' The value of a variable expresses the measure of belief that the corresponding statement is true, i.e., value 0 indicating certain falsity and value 1 indicating certain truth.

Each variable (node) is assigned a baseline parameter $b \in [0, 1]$. Each arc is associated with two parameters $h \in [0, 1]$ and $g \in [0, 1]$. Suppose node X is the parent of node Y. Parameter g specifies the change in the belief of Y relative to b_Y, when X is 'true.' Parameter h specifies the change in the belief of Y relative to b_Y, when X is 'false.' Causal strength $c_{Y|X}$ is defined as g, if $X = true$, and as h, if $X = false$. A positive value of $c_{Y|X}$ indicates an increase in the belief of Y:

$$b_{Y|X} = b_Y + (1 - b_Y) \cdot c_{Y|X} \ . \tag{4.1}$$

A negative value of $c_{Y|X}$ indicates a decrease in b_Y.

$$b_{Y|X} = b_Y \cdot c_{Y|X} \ . \tag{4.2}$$

In the case of multiple parents with values $\mathbf{X}$, we determine the causal strength $c_{Y|X_i}$ for each parent variable X_i, and aggregate positive (C_+) and negative (C_-) causal strengths separately. For all $c_{Y|X_i} \geq 0$ we have

$$C_+ = 1 - \prod_i \left(1 - c_{Y|X_i}\right) \ , \tag{4.3}$$

and all $c_{Y|X_i} \leq 0$ are aggregated by:

$$C_- = \prod_i \left(1 - \left|c_{Y|X_i}\right|\right) \ . \tag{4.4}$$

We then combine the aggregated causal strengths C_+ and C_-. If $C_+ \geq C_-$, then overall influence O is defined as

$$O = 1 - \frac{1 - C_+}{1 - C_-} \ , \tag{4.5}$$

and for $C_+ \leq C_-$, we have

$$O = -\left|1 - \frac{1 - C_-}{1 - C_+}\right| \ , \tag{4.6}$$

Note that O is always negative in Equation 4.6. Finally, we determine the conditional probability distribution over Y for a given parent configuration $\mathbf{X}$:

$$P(Y|\mathbf{X}) = \begin{cases} b_Y + (1 - b_Y)O & \text{for } O \geq 0 \\ b_Y(1 - |O|) & \text{for } O < 0 \end{cases} . \tag{4.7}$$

Repeating this procedure for all possible parent configuration gives us a CPT. Note that the CAST model belongs to the class of models under assumption of independence of causal influence (ICI), as each parent variable has is associated with only one mechanism, taking the parameters h and g.

4.1.2 An Example

Figure 4.2 shows an influence net using the CAST model. Three influence variables are defined: (E) regular exercise, (D) balanced diet and (S) frequent smoker. The effect variable (H) measures our belief that the person in question is in good health. Notice that the presence (D=true) of a balanced diet encourages the overall health ($g_D = 0.6$), while its absence (D=false) has a negative influence ($g_D = -0.3$) on health. Frequent smoking has a strong negative influence on the overall health, while its absence has no influence on health. The baseline parameter is $b_H = 0.2$.

Suppose we want to calculate the probability of good health for a person that is a frequent smoker (S=true), follows a balanced diet (D=true), and never exercises (E=false). Thus we have causal influences $c_{H|S} = -0.7$, $c_{H|D} = 0.3$, and $c_{H|E} = -0.5$. Because we have only one positive causal influence, the aggregated positive causal influence is $C_+ = 0.3$. The aggregated negative causal influence is calculated by:

$$C_- = 1 - (1 - |-0.7|)(1 - |-0.5|) = 1 - 0.3 \cdot 0.5 = 0.65$$

Next, we calculate the combined positive and negative influences by:

$$O = -\left|1 - \frac{1 - 0.65}{1 - 0.3}\right| = -|1 - 0.5| = -0.5$$

Finally, we are able to calculate the conditional probability:

$$P(h|s, d, \overline{e}) = 0.2 \cdot (1 - |-0.5|) = 0.1 \tag{4.8}$$

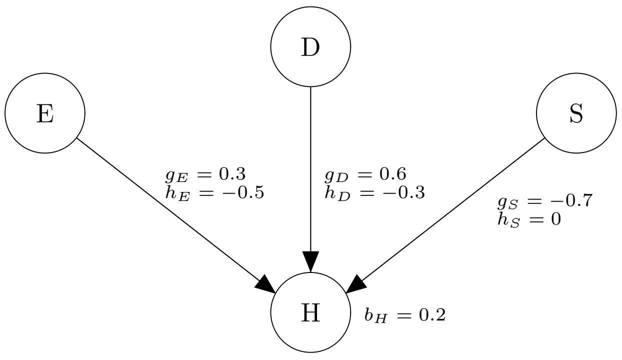


Figure 4.2: Influence net using the CAST model

Repeating the same calculations for each possible configuration of parent variables will give us the CPT. Once we have the CPT, we can apply inference to calculate any conditional probability in the network.

4.1.3 The Elicitation Problem

The main disadvantage of the CAST model is that is suffers from unclear parameterization. This applies in particular to the baseline parameter. In our experience, experts have trouble understanding its meaning. It is not the prior probability, nor is it the leak probability.

The first interpretation defines the baseline as the prior probability of the child event to occur, i.e., without regard to the parent events. In CAST, however, the absence of parent events can also incur a causal influence. This means that regardless of whether or not the parent variable is absent or present, there is no guarantee that there is no influence. Consequently, the CAST model does not allow to disregard the parent events, and the notion of the baseline parameter being a prior probability is invalid.

The second way to interpret the baseline probability is as a leak probability, i.e., the probability that the child event is caused by some event that is not encompassed by the model. In this case, the ICI assumption is violated: modeled parent variables have a causal influence on the influence of the un-modeled parent variable, i.e., the baseline.

Note that the cause of the problem is exactly the lack of the *amechanistic property*: parent variables have no state in which they are guaranteed to have absolutely no influence on the child variable, i.e., a distinguished state. Therefore, we cannot assign a meaning the the baseline parameter, and, consequently, we cannot devise a question that will directly give us the baseline parameter.

In practice, this means that finding the right parameters for a CAST model becomes an iteration process, changing the parameters until the model shows the desired behavior. Obviously, such a process is undesirable, not only because it is inefficient, but also because it manifests the lack of semantic definition.

4.1.4 Discussion

The causal models discussed in chapter 3 have appealing properties, yet hey are not powerful enough to cover all real-life situations, because most situations cannot be modeled by either causes or conditions exclusively, but can only be accurately represented by a combination of both.

The CAST model shows a powerful and flexible approach, yet it suffers from ambiguous meaning for the parameterization. The main advantage of the CAST model is that it is able to combine encouraging and discouraging influences. However, allowing a single parent event to have both an encouraging effect when present and a discouraging effect when absent (and vice versa) is disputable. Consider the example in figure 4.2. The absence of a balanced diet will certainly deteriorate a persons health, as an abundance of fat and sugar will lead to health problems. However, we could argue that the presence of a diet simply maintain the status quo, rather than improving health. Consider this analogy: fire causes smoke, but the absence of fire does not *cause* the absence of smoke i.e. if smoke is present due to some other cause, the absence of a fire will not make the smoke disappear.

We conclude that CAST suffers from some semantic problems, which lead to problems when building a model. Paradoxically, CAST logic is designed specifically for building influence models based on parameters elicited from human experts. Despite this, CAST has been applied in practice with great success, in particular by the military for the assessment of complex emergencies [22, 23, 24, 21, 25].

4.2 Fault Tree Analysis

Before we propose our own model in Chapter 5, we will briefly discuss Fault Tree Analysis (FTA). Some of the ideas that are represented in our own model, have been been inspired by FTA. Consequently, we think the reader would benefit from a short introduction. FTA is a technique frequently used for system analysis in the context of plant processes or other automated procedures. A fault tree is a directed, acyclic graph that models the critical elements of a system. In the event of a system failure, the fault tree uses boolean logic to determine the cause(s) of the failure.

4.2.1 Fault Trees

The nodes of a fault tree represent either (1) sensor readings, (2) events deduced from sensor readings or (3) logical operations. The edges between the nodes define which nodes interact with each other. We discern three types of logical operators: AND, OR and INHIBITION. The INHIBITION-gate only produces TRUE when its input is TRUE and the condition associated with that inhibition is satisfied. The AND-gate will only be TRUE when all its inputs are TRUE, while the OR-gate will be TRUE if at least one of its inputs is TRUE. More logical gates are available but we will not consider them in this example.

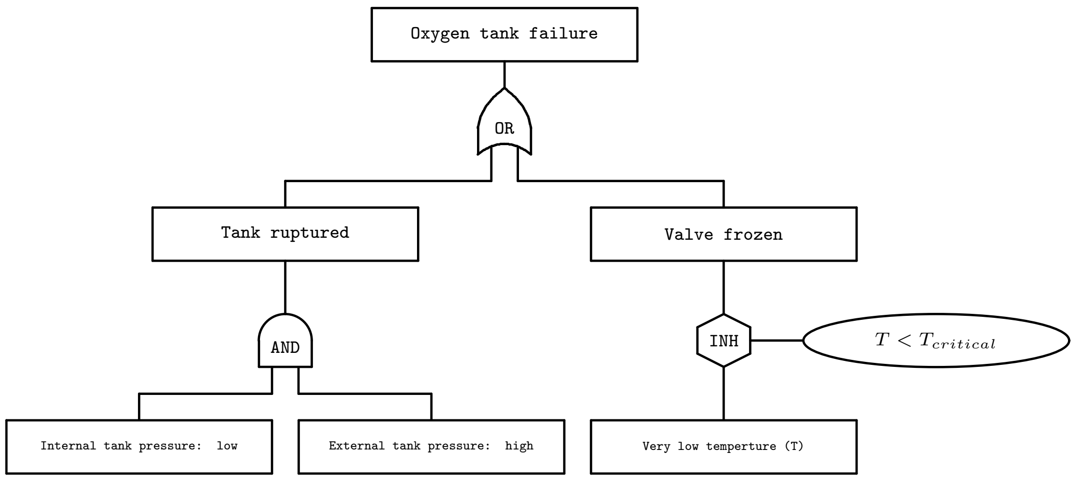


Figure 4.3: Oxygen tank fault tree

Figure 4.3 shows a fault tree that models the failure of an oxygen tank in a satellite. A failure of the oxygen tank occurs as a consequence of either a ruptured tank, or a frozen valve. We have placed a sensor both inside and outside the tank, both measuring pressure. In the event that we read a decrease in pressure inside the tank, while simultaneously reading a significant increase in pressure directly outside the tank, the model assumes that the tank has ruptured. Just one of both readings is not enough to assume a rupture e.g. in the event that we just read low internal pressure, the tank might simply be low on oxygen. A low temperature can cause the tank valve to freeze, preventing the tank from releasing oxygen. However, a low temperature can only cause the valve to freeze under the condition that it reaches a certain critical level.

4.2.2 Discussion

Fault trees and the model proposed in the next chapter have common properties. Both are graphical representations which use propositional logic and logical operators to model an arbitrary problem. However, their goals are different. FTA focuses on predicting future failures based on sensor readings and explaining already observed failures. Our modal focuses on revealing the causal interaction between a number of events. Furthermore, both models use uncertainty in a slightly different context. In FTA the uncertainty is assigned to the observed node state, questioning the accuracy of the sensor readings. In Conjunction Model Logic we assign uncertainty to the relation between two nodes, questioning the extent of causal influence between the two corresponding events.

5 The PQ-Model

At this point we have attained enough theoretical baggage to devise a useful model of our own. Remember that we are looking for a model that will act as a reasoning framework from qualitative problems under uncertainty. In this chapter we propose exactly such a model, which is the main theoretical contribution of this thesis.

The Probabilistic Qualitative (PQ) model is based on the assumption that humans have a strong tendency to regard real-world events as the consequence of some combination of *causes* and *conditions*. For example, a child event occur if it is caused by at least one parent event *and* all the conditions for its occurrence are satisfied. Thus we reason that the occurrence of the child event depends on the occurrence of a combination of parent events. This is not necessarily a single combination, as several combinations of observed parents may produce the child event, e.g., a plant will flourish of we give it water regularly *and* its is exposed to sufficient light and warmth.

We restate our guidelines: (1) the proposed model is consistent with propositional logic and probability theory, (2) the proposed model belongs to the class of amechanistic models under assumption of independence of causal influence (ICI), and (3) the model is able to handle a mix of positive and negative influences. To make the theory underlying the PQ-model as clear as possible, we will (re)build the theoretical model in three sections:

- Introduce our reasoning framework (Section 5.1)
- Introduce our theoretic deterministic model (Section 5.2)
- Extend this model to incorporate uncertainty (Section 5.3)

Section 5.1 is rather straightforward, and needs no lengthy discussion. Section 5.2 is more complex, as we introduce four kinds of qualitative relations and their corresponding logical statements. Once we have worked out the underlying logic, we take the next step; including uncertainty. Section 5.3 takes the logical framework, and incorporates probability theory.

5.1 Framework of Reasoning

We use propositional logic as the framework for our gate, as it corresponds well to the concept of causes and conditions. However, the primary reason for using propositional logic as a framework is that it is infallibly correct. This to the contrary of a heuristic approach, which is both likely to be revised whenever a new insight is acquired, and hard to defend because its validation is based not on mathematical proof, but on experience.

However, propositional logic is not without its drawbacks. We implicitly assume that we can always translate a qualitative problem to a deterministic logical proposition. Although we recognize that not all qualitative problems will translate well, if at all, to logical propositions, we believe that propositional logic will help us model such a problem more often than not. It would be naive to expect that the proposed model provides a perfect fit for every problem.

In our model, when we translate a qualitative problem to propositional logic, the variables will represent events and their values represent the measure of belief we have in the occurrence of these events. Thus, in the case that event Y is true, i.e., $P(Y = y) = 1$, we believe that event Y is absolutely certain to occur. In the case that we find Y false, i.e., $P(Y = y) = 0$, we believe that it is absolutely certain that we will not observe event Y. In the case that we have $P(Y = y) = 0.7$, we belief the event Y to occur with probability of 70%.

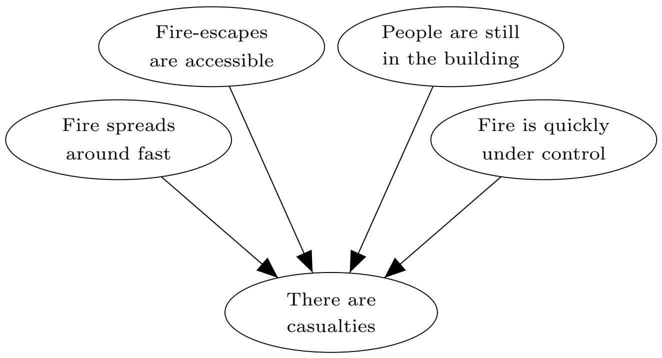

Fire-escapes are accessible
People are still in the building
Fire spreads around fast
Fire is quickly under control
There are casualties

Figure 5.1: Network modeling the probability of casualties in a fire

5.2 Modeling Causal Influences

Our first step toward modeling influences is finding the corresponding deterministic model, so that we have a thorough understanding of the models behavior, before we include noise. We accomplish this by finding the logical expression that corresponds each of the influence types that will model. We distinguish four ways in which a parent variable could behave. A parent variable may act as:

- a promoting event, Section 5.2.1
- a barring event, Section 5.2.2
- an inhibiting event, Section 5.2.3
- a required event, Section 5.2.4

This section is structured so that we discuss each type of 'event' separately. Finally we will discuss how to combine these influences into a deterministic model. To clarify the discussed theory, we will use the example network in Figure 5.1 as we go along. The purpose of the example network is to determine the probability of casualties in case of a building on fire. Such a network might be of practical significance when a fire department receives two reports simultaneously, and has to determine quickly which fire is assigned priority. The input, i.e., the parent variables, are four statements concerning the fire, which can be either 'true' or 'false.'

To ensure that the PQ-model is amechanistic (Section 3.1.2), one of the two states is always the distinguished state, i.e., the state in which the parent variable has no influence on the child variable.

For the remainder of this chapter, we use the terms 'promoting event,' 'barring event,' 'inhibiting event,' and 'required event' to refer to variables which have the corresponding type of influence on the child variable. We refer to the 'effect variable' to describe the child variable.

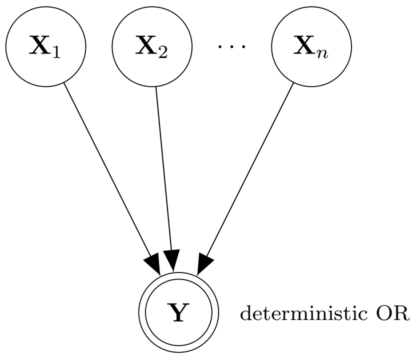


Figure 5.2: Modeling n encouraging causes and a single effect

5.2.1 Promoting Events

One single effect can often be produced by different causes. In the example network in Figure 5.1, a fast spreading fire may result in casualties. Another cause might be poor ventilation, causing the people in the building to suffocate. In a situation with multiple causes, we assume independence of causal influence (ICI), which means that the presence of one cause does not impact the influence of any other cause. We assume ICI not only because this reduces the complexity of our model, but often because we simply do not have a clear understanding of the interaction between causes.

Consider the case where we have three independent promoting events X_1, X_2 & X_3, and a single child event Y. The presence of any of the promoting events is sufficient to produce the child event. So we have:

$$Y \equiv X_1 \vee X_2 \vee X_3 \ . \tag{5.1}$$

Notice that because we claim the two terms to be equivalent instead of a relationship of directed implication. This means we assume that Y cannot be caused by any other event than X_1, X_2 & X_3. This assumption rarely holds in real life. This problem can easily be overcome by introducing a new cause X_L, which represents the accumulation of all un-modeled causes. So, for n promoting events, we have:

$$Y \equiv X_1 \vee ... \vee X_n \vee X_L \ . \tag{5.2}$$

The leak variable can be interpreted in two ways, which, in our case, are mathematically equivalent. The first interpretation is that the leak represents a cause that is not an explicit part of the model. The second interpretation is that the leak represents the prior probability of a child event to be in some particular state, without interference from any modeled influences, e.g, an unbiased coin toss has a 50% chance of ending up heads.

We assume that the distinguished state for any promoting event is ‘false,’ i.e., when ‘true’, the posterior probability of the child event to be ‘true’ increases. When ’false,’ the promoting event does not influence the posterior of the child event. Because we cannot ‘cause’ an event that is already true, we choose the distinguished state of the child event to be ‘false.’ If we translate Equation 5.1 to a graphical representation, we get Figure 5.2. A node drawn with a double circle represents a deterministic interaction, i.e., the values of its parents are sufficient to imply the value of the child node.

5.2.2 Barring Events

It is not required that the distinguished state of a promoting event is the ‘false’ state. Suppose we assume that the ‘true’ state is the distinguished state, i.e., when the event is ‘false’, the posterior probability of the child event to be ‘true’ increases. When ’true,’ the event event does not influence the posterior of the child event. We refer to such an event as a *barring* event.

For example, in case of the building on fire, if the statement *fire-escapes are accessible* is false, the chance of casualties increases. If ‘true,’ it does not influence the chance of casualties, i.e, this is already accounted for by the prior probability (leak) of casualties. In the binary case, changing the distinguished state is equivalent to negating the input variables:

$$Y \equiv \overline{X}_1 \vee ... \vee \overline{X}_n \ . \tag{5.3}$$

We omit the leak variable for reasons that will be made apparent later. For now, it suffices to know we use the leak variable only to model promoting events.

One might ask, if we simply negate the variables, why take the trouble of defining barring events, when you could also simply change the semantics of the variable, e.g., change *fire-escapes are accessible* to *fire-escapes are NOT accessible*? Although this makes sense if the network consists of a single family, to see why barring events are not trivial, consider the following example.

Figure 5.3 shows a Bayesian a network consisting of two families. Suppose that if the fire-escapes are *not* accessible, the probability of casualties in case of fire increases. Also, suppose that if the fire-escapes *are* accessible, the probability that the building is conform to fire-department regulations increases. In this case, changing the semantics of the parent variable simply switches the types of the two influence arcs: both are still needed if we want to built a comprehensive model.

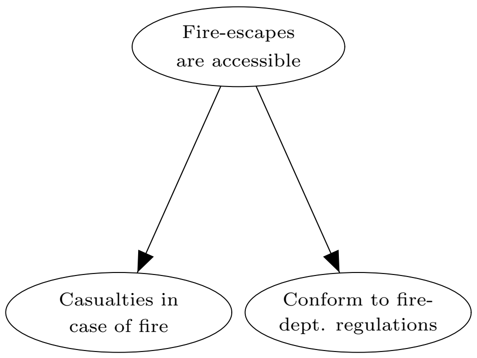

Fire-escapes are accessible
Casualties in case of fire
Conform to fire-dept. regulations

Figure 5.3: A Bayesian network consisting of two families

5.2.3 Inhibiting Events

Now suppose that the fire department makes a timely response, and the fire is under control in matter of minutes. Instead of *increasing* the chance of casualties, a quick response will actually *decrease* the posterior probability of casualties. In Figure 5.1, this event is modeled by the variable *fire is quickly under control.* If this event is 'true,' the posterior probability of casualties is decreased. If this event is 'false,' it does not influence the posterior of the child event. We refer to such an event as an *inhibitive* event.

Let us consider the case in which n inhibitive events and a single child event Y. The presence of any of the inhibitive events is sufficient to inhibit the child event. We use the following logical equivalence:

$$\overline{Y} \equiv U_1 \vee ... \vee U_n \ . \tag{5.4}$$

The child event is 'false' if at least a single inhibitive event is 'true.' We assume the distinguished state of an inhibitive event to be 'false.' Because we cannot inhibit an event that is 'false' to begin with, common sense dictates that the distinguished state of the child event is 'true.'

5.2.4 Required Events

Again, it is not necessary that the distinguished state of an inhibitive event is 'false.' Suppose we switch the distinguished state to 'true,' like we did earlier for promoting/barring influences. If such an event is 'false,' the posterior probability of casualties is decreased. If the event is 'true,' it does not influence the posterior of the child event. We refer to such an event as a *required* event.

In Figure 5.1, the variable *people are still in the building* is a required event. Obviously, if there are no people in the building, the chance of casualties is greatly diminished. If there *are* people

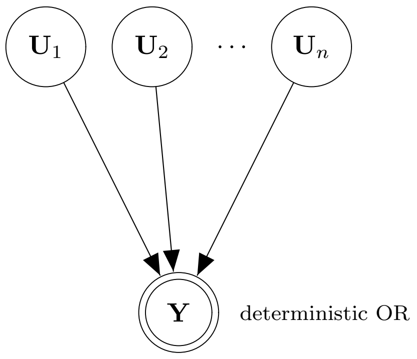


Figure 5.4: Modeling n inhibiting causes and a single effect

in the building, the posterior probability of casualties does not change, because this was already assumed. Switching the distinguished state for binary variable is the same as a negation of the input (parent) variables:

$$Y \equiv \overline{U}_1 \wedge ... \wedge \overline{U}_n \ . \tag{5.5}$$

The distinguished state of a required event is 'true.' The distinguished state of the child event is also 'true.' Thus the child event can only be 'true,' if all required events are also 'true'.

Required events are simply negations of inhibitive events. In Section 5.2.2 we argued that barring events are needed to be able to build comprehensive models, even though they are the negations of promoting events. The same argument holds for required events. In addition, more so than barring events, we believe required events are intuitive to human reasoning. Take a simple plant for example: it needs water, light and warmth. Only one of three is not sufficient. Instead, all three are *needed.*

5.2.5 Combining Different Types of Influence

Although we are able to model both positive influences and negative influences separately, it is less obvious how to combine both into a single model. We start by searching for the logical proposition which defines the interaction between parent and child variables. When we try to combine the logical equitations corresponding for the four types of events, we run into a problem.

For promoting and barring events, the distinguished state of the child event is 'false;' for inhibiting and required events, the distinguished state is 'true.' However, if we want to encompass all four types into a single model, by definition of an amechanistic model, only one of both states of the child event can be the distinguished state.

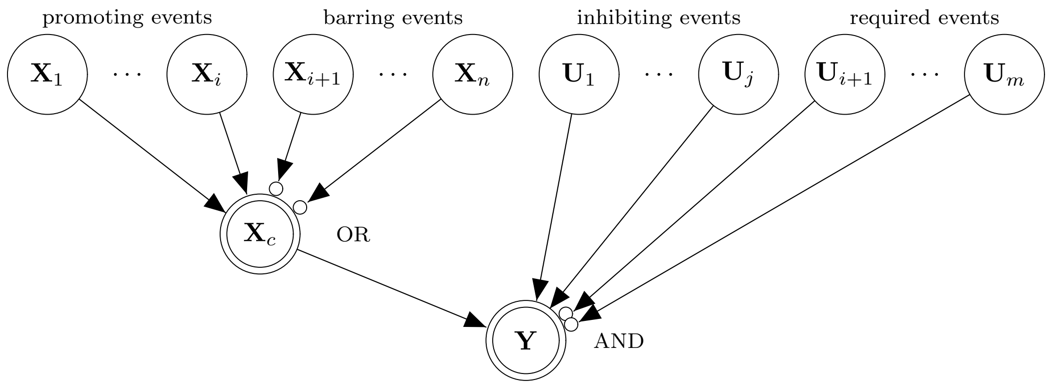


Figure 5.5: Combining different influences into a single model

We can achieve this by making use of the laws of DeMorgan, given by Equation 5.6 and 5.7:

$$\overline{A \wedge B} \equiv \overline{A} \vee \overline{B}\ , \tag{5.6}$$

$$\overline{A \vee B} \equiv \overline{A} \wedge \overline{B}\ . \tag{5.7}$$

First we negate both sides of Equation 5.4 in order so that Y is implied instead of $\overline{Y}$:

$$\overline{Y} \equiv U_1 \vee ... \vee U_n\ ,$$

$$Y \equiv \overline{U_1 \vee ... \vee U_n}\ ,$$

$$Y \equiv \overline{U_1} \wedge ... \wedge \overline{U_n}\ . \tag{5.8}$$

Now we can deduce the proposition that we will use to combine opposite influences. In order to be consistent, the combined logical proposition must only be true for a particular parent configuration if Equation 5.2, 5.3, 5.4 and 5.5 are also true for that same configuration. Given that, by DeMorgan's laws, Equation 5.4 and 5.8 are equivalent, we have:

$$Y \equiv \left(X_L \vee X_1 \vee ... \vee X_i \vee \overline{X}_{i+1} \vee ... \vee \overline{X}_n\right) \wedge U_1 \wedge ... \wedge U_j \wedge \overline{U}_{j+1} \wedge ... \wedge \overline{U}_m\ . \tag{5.9}$$

We now have the logical proposition we need to define the interaction between promoting, barring, inhibiting and required events. We note that the proposition on the righthand-side of Equation 5.9 is in the Conjunctive Normal Form. Because all of the conjuncts, save one, consist of a single variable, we can build a simple model to represent this proposition, using only two operators: one deterministic-OR and one deterministic-AND. The model in Figure 5.5 represents a single gate that combines promoting, barring, inhibiting and required events.

Because a leak variable represents the aggregated influence of all influences that are not explicitly in the model, it unnecessary to include multiple leaks. Given that the distinguished state of the child variable is 'false,' including the leak in the form of an inhibitive or required event makes little sense. For these reasons we have included only one leak, in the form of a promoting event.

5.2.6 Building a Conjunctive Normal Form

It can be shown that each logical statement S can be written in the disjunctive normal form (DNF). We can do so by creating a truth-table and making disjuncts of the literals in each row for which S is true. We can create the conjunctive normal form (CNF) for S, by first determining the DNF for $\overline{S}$ and then applying DeMorgan's law.

Table 5.1: Truth table for S

X	C	S	$\overline{S}$
0	0	0	1
1	0	0	1
0	1	0	1
1	1	1	0

Consider Table 5.1, which holds the truth values for statement S. To find the DNF, we isolate all the cases for which statement S is not true. We find $\overline{S}$ to be true in three rows, indicating we have three disjuncts in the DNF:

$$\overline{S} \equiv (\overline{x} \wedge \overline{c}) \vee (x \wedge \overline{c}) \vee (\overline{x} \wedge c) \ ,$$

and we find the CNF by applying the laws of DeMorgan:

$$S \equiv \overline{(\overline{x} \wedge \overline{c})} \wedge \overline{(x \wedge \overline{c})} \wedge \overline{(\overline{x} \wedge c)} \ ,$$

$$S \equiv (x \vee c) \wedge (\overline{x} \vee c) \wedge (x \vee \overline{c}) \ .$$

The derived CNF is not unique. We can easily see the far more obvious CNF: $x \wedge c$. However, following these steps, we can rewrite every logical proposition to a CNF. Figure 5.6b shows how we can model an arbitrary CNF with a single deterministic-AND gate and one deterministic-OR gate for each conjunct. Figure 5.6a shows how we can make an equivalent model with an arbitrary number of interconnected instances of the PQ-model, by the logical identity of associativity of $\wedge$.

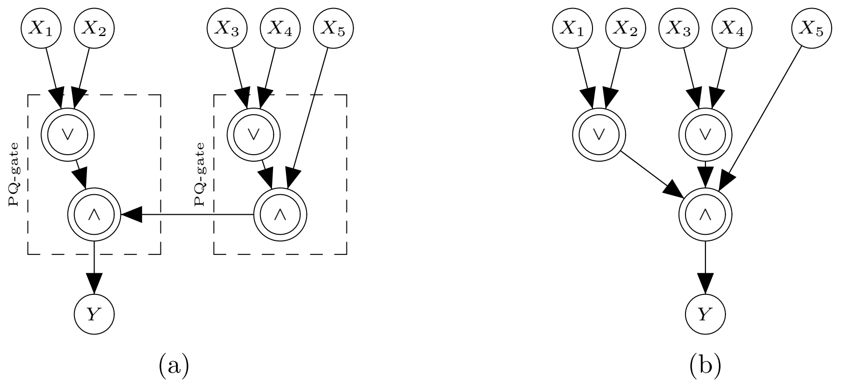


Figure 5.6: Modeling a CNF with PQ-gates (a) and modeling a CNF with AND/OR gates (b)

For reasons of clarity, we will refer to a single instance of the PQ-model as a PQ-gate. We argue that it is possible to use PQ-gates as building blocks to model every problem that can be translated to a logical proposition.

5.3 Modeling Uncertainty

Up to this point we have only discussed the deterministic version of the PQ-model. A deterministic model leaves no room for uncertainty. However, if we want to address real-life problems, we have to deal with uncertainty. To approximate uncertainty in a deterministic model, we will include noise.

Before we include noise, we again stress the importance of the amechanistic property. To guarantee the amechanistic property we assume (1) that every variable has one distinguished state and (2) when all parent variables are in their distinguished state, then the child variables is certain to be in the distinguished state.

By definition, only parent variables that are in their non-distinguished state can have influence on the child variable Y. Consequently it is very easy to determine the influence of a parent variable X_i: we simply ask for the probability of the child event being in its non-distinguished state, given that all parent variables are in their distinguished state, except X_i. Because we are sure that none of the other parents have any influence on Y, we can attribute all behavior of Y to parent X_i. This is the very reason why we desire the amechanistic property. Note that we can still build an accurate model without the amechanistic assumption, but determining the right parameters becomes a sort of 'tuning' process in which the parameters have lost clear meaning. This is precisely one of the caveats of CAST, which we must avoid.

For the rest of this thesis, when we refer to an influence as being positive, we mean that this influence will increase the posterior probability that the child event is 'true.' When we refer to an influence as being negative, we mean that this influence will decrease the posterior probability that the child event is 'true.'

First we will briefly discuss how to include probabilities into our model in general. We then go into detail on uncertainty relating to each type of influence, in the following order:

- uncertainty in promoting events, Section 5.3.2
- uncertainty in barring events, Section 5.3.3
- uncertainty in inhibiting events, Section 5.3.4
- uncertainty in required events, Section 5.3.5

Subsequently we will discuss the model that combines the influences of all four event types, i.e., the leaky noisy Probabilistic Qualitative (PQ) model, and clarify it with an example. This comprehensive model is the main theoretical contribution of this thesis.

5.3.1 Probability Theory

The PQ-model is a quantitative model, but by providing concise and intuitive semantics for the model parameters the model will be applicable to qualitative problems. Therefore, we must have some translation from a qualitative problem to a quantitative structure if we want to create a mathematically sound reasoning structure.

Each parent variable that is in its non-distinguished state has some measure of influence $v \in [0, 1]$. If $v = 0$, then the parent variable has no influence whatsoever and the child variable is effectively independent of the parent. If $v = 1$, the relation and the parent variable and the child variable is deterministic. In practice we have $0 < v < 1$, because both independent and deterministic behavior are rarely observed. However, both extremes are of use, e.g., if we have a promoting event (which is in its non-distinguished, i.e., 'true' state) with influence $v = 0.8$, we know that, more often than not, the child event is also in its 'true' state.

5.3.2 Uncertainty In Promoting Events

Recall that in Subsection 5.2.1 we discussed a structure for modeling promoting events. We assumed that the distinguished state of a parent that models promoting event is 'false.' Consequently, when the parent is 'true' it *may produce* the child event, but when it is 'false,' it *is certain not to produce* the child event.

When parent X_i is in its non-distinguished state, we assign a probability v_i that parent event X_i produces the child event. We can include this noise in the network structure by adding a noise variable Z_i which has the following behavior:

$$P(z_i|X_i) = \begin{cases} v_i & \text{if } X_i = x_i \\ 0 & \text{if } X_i = \overline{x}_i \end{cases} , \tag{5.10}$$

and the child variable Y interacts with the variables $Z_1, ..., Z_n$ as a deterministic-OR gate. We can interpret this model in Figure 5.7 as a plumbing system. The variables $X_1, ..., X_n$ represent water mains. The variables $Z_1, ..., Z_n$ represent valves, which randomly open and close. A closed valve stops the flow of water and each valve Z_i is open with probability v_i. The variable Y represents a fountain, which works when connected to at least one water main. Therefore, the fountain will only be dry when none of the water mains connect:

$$P(\overline{y}|\mathbf{X}) = \prod_{x_i \in \mathbf{x_d}} (1 - v_i) \ . \tag{5.11}$$

Hence, the probability of the fountain working is given by its complement:

$$P(y|\mathbf{X}) = 1 - \prod_{x_i \in \mathbf{x_d}} (1 - v_i) \ . \tag{5.12}$$

We define $\mathbf{x_d}$ the subset of $\mathbf{X}$ which holds all parents that are in their distinguished state. To complete Equation 5.12, we introduce the leak parameter v_L, which represents the probability that the child event occurs even when none of the modeled causes are observed. In the example this would mean that we can switch on the fountain even though none of the water mains are in use. Suppose there is an external water pump connected to a reservoir. So we have:

$$P(y|\mathbf{X}) = 1 - (1 - v_L) \prod_{x_i \in \mathbf{x_d}} (1 - v_i) \ . \tag{5.13}$$

Due to the amechanistic property, we can query the influence parameter v_i by asking a simple question (a): *what is the probability that Y is true, given that all parents are in their distinguished states, except event X_i?* However, because the leak parameter is not explicit, the answer will constitute the influence of both X_i *and* the leak parameter. This problem was first pointed out by Diez [5]. To separate both influences, we first find the leak parameter by asking (b): *what is the probability that Y is true, given that all parents are in their distinguished states?* To distill the influence v_i, we apply:

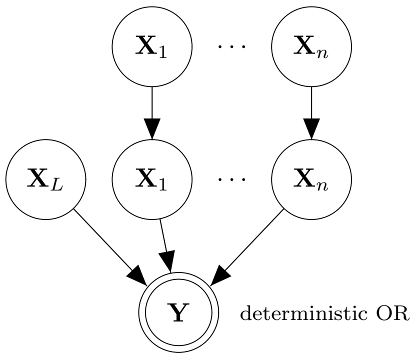


Figure 5.7: Modeling n promoting events and a single effect

$$v_i = \frac{1 - P(y|x_i)}{1 - v_L} \ . \tag{5.14}$$

$P(y|x_i)$ represents the answer to quetion (a) and v_L represents the answer to question (b).

5.3.3 Uncertainty In Barring Events

For barring events, Equation 5.12 holds as well. Like a promoting event, a barring event, that is in its non-distinguished state, has a positive influence on the child event. However, it is important to note that the distinguished state of a barring event is 'true' instead of 'false.' So for barring event X_i we define the behavior of Z_i as:

$$P(z_i|X_i) = \begin{cases} v_i & \text{if } X_i = \overline{x}_i \\ 0 & \text{if } X_i = x_i \end{cases} , \tag{5.15}$$

5.3.4 Uncertainty In Inhibiting Events

In subsection 5.2.3 we discussed a structure for modeling inhibiting influences. We assumed the distinguished state of a parent that models inhibiting event is the 'false' state. Consequently, when the inhibiting event is 'true,' it *may prevent* the child event, but when it is 'false,' it *is certain not to prevent* the child event.

When we observe parent U_j, we assign a probability d_j that parent event U_j inhibits the child event. We can include this noise in the network structure by adding a noise variable W_i which has the following behavior:

$$P(w_j|U_j) = \begin{cases} d_j & \text{if } U_j = u_j \\ 0 & \text{if } U_j = \overline{u}_i \end{cases} , \tag{5.16}$$

and the child variable Y interacts with the variables $\overline{W}_1, ..., \overline{W}_m$ as a deterministic-AND gate. We have shown in Subsection 5.2.5 that by DeMorgan's law, the model in Figure 5.4 is the logical equivalent of the model in Figure 5.8.

Again we use the fountain example for the model in Figure 5.8. The variables $U_1, ..., U_n$ represent corrosion spots in the fountain mechanism. The variables $W_1, ..., W_n$ represent leaks caused by corrosion. A leak will cause a pressure drop and will turn the fountain off. A corrosion spot U_i is causes a leak W_i with probability d_i. The variable Y represents a fountain, which works when there are no leaks.

Equation 5.17 gives us the function for determining the probability if Y occurring:

$$P(y|\mathbf{U}) = \begin{cases} \prod_{u_i \in +\mathbf{u}} (1 - d_i) & \text{if } +\mathbf{u} \neq \varnothing \\ 1 & \text{if } \mathbf{u_d} = \varnothing \end{cases} . \tag{5.17}$$

We define $\mathbf{u_d}$ the subset of $\mathbf{U}$ which holds all parents that are in their distinguished state. Here we run into a problem. We cannot ask for the negative influence u_j of parent U_j in the same way we do promoting positive influences, because we have assumed that the distinguished state of the fountain is 'false.'It does not makes sense to ask: *if the fountain is turned off, what is the probability that corrosion spot U_j will cause the fountain to turn off?* Obviously, we need to ask another question if we want to determine d_j. To find this question we first restate the interaction between promoting and inhibiting events:

$$Y \equiv (X_1 \vee ... \vee X_n) \wedge \overline{U}_1 \wedge ... \wedge \overline{U}_m \; . \tag{5.18}$$

Adding uncertainty as to logical equation 5.18 gives us:

$$P(y|\mathbf{X}, \mathbf{U}) = (1 - \prod_{x_i \in \mathbf{x_d}} (1 - v_i)) \prod_{u_j \in \mathbf{u_d}} (1 - d_j) \; . \tag{5.19}$$

At this point we assume that we have already queried all positive influences $\mathbf{v}$. Thus, for promoting events, we can determine the cumulative influence of an arbitrary configuration $\mathbf{x_a}$. We ask the question: *what is the probability of child event Y occurring, given that the promoting events are in state configuration $\mathbf{x_a}$, and all inhibiting events $\mathbf{U}$ are 'false,' except U_j?*. We are now able

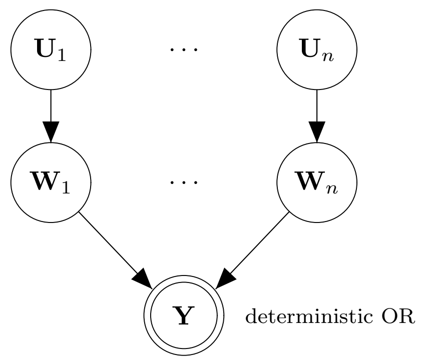


Figure 5.8: Modeling n inhibiting events and a single effect

to marginalize the observed noise to d_j:

$$d_j = 1 - \frac{P(y|\mathbf{x_a}, \overline{u}_1, ..., u_j, ..., \overline{u}_m)}{1 - \prod_{x_i \in \mathbf{x_d}}^{\mathbf{x_a}} (1 - v_i)} \; . \tag{5.20}$$

We simply determine the negative influences relative to positive influences. To clarify, remember the example of the burning building. For the purpose of explanation, we only consider the variables *fire spreads fast* (a promoting event) and *fire is quickly controlled* (a inhibiting event). First we ask (p): *What is the chance of casualties, supposing that the fire spreads fast, and it IS NOT quickly controlled?* Suppose we get the answer $p = 0.8$. Subsequently we ask (q): *What is the chance of casualties, supposing that the fire spread fast, but it IS quickly controlled?* Suppose we get the answer $q = 0.4$. We see that quickly controlling the fire, halves the chance of casualties, thus we assign a negative influence of 0.5 to the variable *fire is quickly controlled.*

5.3.5 Uncertainty In Required Events

For required events, Equation 5.17 holds as well. Like an inhibiting event, a required event, that is in its non-distinguished state, has a negative influence on the child event. However, it is important to note that the distinguished state of a required event is 'true' instead of 'false.' So for required event U_j we define the behavior of W_j as:

$$P(w_j|U_j) = \begin{cases} d_j & \text{if } U_j = \overline{u}_i \\ 0 & \text{if } U_i = u_i \end{cases} \; . \tag{5.21}$$

Table 5.2: PQ-model Parameter Semantics

Type	Dist. state	Param.	Semantics
Promoting event	'false'	v_i	if parent X_i^{pro} is 'true:' $v_i = 0 \rightarrow$ child independent $v_i = 1 \rightarrow$ child present (if not inhibited)
Barring event	'true'	v_j	if parent X_j^{bar} is 'false:' $v_j = 0 \rightarrow$ child independent $v_j = 1 \rightarrow$ child present (if not inhibited)
Inhibiting	'false'	d_k	if parent U_k is 'true:' $d_k = 0 \rightarrow$ child independent $d_k = 1 \rightarrow$ child cannot occur
Enabling condition	'true'	d_l	if parent U_l is 'false:' $d_l = 0 \rightarrow$ child independent $d_l = 1 \rightarrow$ child cannot occur
Prior belief	'false'	v_L	if all parents are in distinguished state: $v_L = 0 \rightarrow$ child false $v_L = 1 \rightarrow$ child true

5.3.6 The Leaky Noisy PQ-model

Now that we have uncertainty for promoting, barring, inhibiting, and required events, we define the leaky noisy PQ-model. For purposes of clarity, we define the set of promoting events as $\mathbf{X}^{pro}$, the set of barring events as $\mathbf{X}^{bar}$, and $\mathbf{X} = \mathbf{X}^{pro} \cup \mathbf{X}^{bar}$. Likewise, we define the set of inhibiting events as $\mathbf{U}^{inh}$, the set required events as $\mathbf{U}^{req}$, and $\mathbf{U} = \mathbf{U}^{inh} \cup \mathbf{U}^{req}$. We derive from Equation 5.9 the following function for determining the probability of child variable y:

$$P(y|\mathbf{X}, \mathbf{U}) = (1 - (1 - v_L) \prod_{X_i = x_i}^{\mathbf{X}^{pro}} (1 - v_i) \prod_{X_j = \overline{x}_j}^{\mathbf{X}^{bar}} (1 - v_j)) \prod_{U_k = u_k}^{\mathbf{U}^{inh}} (1 - d_k) \prod_{U_l = \overline{u}_l}^{\mathbf{U}^{req}} (1 - d_l) \tag{5.22}$$

5.3.7 An example problem

We consider an example of an application for the PQ-model. Figure 5.9 shows our earlier example of the building on fire, including influence types and strengths. Child variable Y represents our belief that the fire will demand casualties. Variable S represents the promoting event: *fire spreads quickly*, variable E represents the barring event: *fire-escapes are accessible*, variable B represents the required event: *people are still in the building*, and variable C represents the inhibiting event: *fire is quickly controlled.*

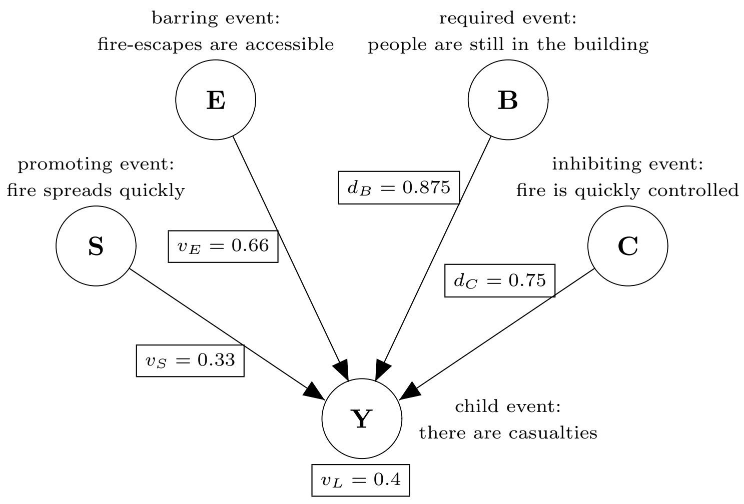


Figure 5.9: Network modeling the probability of casualties in a fire

Parameter v_L To find the value for the leak parameter v_L, we ask for the probability of casualties given that all parent variables are in their distinguished states, i.e., (1) fire *does not* spread quickly, (2) fire-escapes *are* accessible, (3) people *are* in the building, and (4) the fire *is not* quickly controlled. In other words, we ask for probability $P(y|\overline{s}, e, b, \overline{c})$. We assign $v_L = 0.4$.

Parameter v_S To find the value for parameter v_S, we ask for probability that all variables are in their distinguished state, except S, i.e., $P(y|s, e, b, \overline{c})$. Suppose the answer is 0.6. We find v_S by distilling the leak parameter:

$$v_S = 1 - \frac{1 - 0.6}{1 - 0.4} = 0.33$$

Parameter v_E To find the value for parameter v_E, we ask for probability that all variables are in their distinguished state, except E, i.e., $P(y|\overline{s}, \overline{e}, b, \overline{c})$. Suppose the answer is 0.8. We find v_S by distilling the leak parameter:

$$v_E = 1 - \frac{1 - 0.8}{1 - 0.4} = 0.67$$

Parameter d_B To find the value for parameter d_B, we determine the negative influence relative an arbitrary (not empty) set of promoting/barring events. We already know $P(y|s, e, b, \overline{c}) = 0.6$, so now we ask for the probability of casualties for the same parent variables configuration, with the exception of B, i.e., $P(y|s, e, \overline{b}, \overline{c})$. Suppose the answer is 0.1. We now determine d_B:

$$d_B = 1 - \frac{0.1}{0.6} = 0.833$$

Table 5.3: CPT generated by the PQ-model

s	s	s	s	s	s	s	s	$\overline{s}$	$\overline{s}$	$\overline{s}$	$\overline{s}$	$\overline{s}$	$\overline{s}$	$\overline{s}$	$\overline{s}$
e	e	e	e	$\overline{e}$	$\overline{e}$	$\overline{e}$	$\overline{e}$	e	e	e	e	$\overline{e}$	$\overline{e}$	$\overline{e}$	$\overline{e}$
b	b	$\overline{b}$	$\overline{b}$	b	b	$\overline{b}$	$\overline{b}$	b	b	$\overline{b}$	$\overline{b}$	b	b	$\overline{b}$	$\overline{b}$
c	$\overline{c}$	c	$\overline{c}$	c	$\overline{c}$	c	$\overline{c}$	c	$\overline{c}$	c	$\overline{c}$	c	$\overline{c}$	c	$\overline{c}$
0.20	0.60	0.03	0.10	0.29	0.87	0.05	0.14	0.13	0.40	0.02	0.07	0.27	0.80	0.04	0.13

Parameter d_C To find the value for parameter d_C, we determine the negative influence relative an arbitrary (not empty) set of promoting/barring events. We already know $P(y|s, e, b, \overline{c}) = 0.6$, so now we ask for the probability of casualties for the same parent variables configuration, with the exception of B, i.e., $P(y|s, e, b, c)$. Suppose the answer is 0.2. We now determine d_B:

$$d_B = 1 - \frac{0.2}{0.6} = 0.67$$

Now that we have all parameters, we can build a full CPT. Suppose we want to know $P(y|s, \overline{e}, b, \overline{c})$. We use Equation 5.22 to calculate:

$$P(y|s, \overline{e}, b, \overline{c}) = (1 - (1 - 0.4)(1 - 0.33)(1 - 0.66)) = 0.87$$

We can determine the rest of the CPT in a similar manner. Table 5.3.7 shows the full CPT generated from the five (four parents + leak) elicited parameters. Note that the number of parameters is only linear in the number of parent variables.

5.4 Discussion

It is important to take note of the asymmetric property of the PQ-model, inherent to the propositional logic that serves as its framework. By this we mean that the model does not follow the heuristic that the net influence on a child variable, that is subject to both a positive and negative influence of equal strength, is zero.

To illustrate the notion of symmetry, imagine the problem of the building on fire. Assuming $v_S = v_E = d_B = d_C$ and $P(S) = P(E) = P(B) = P(C)$, a symmetric model would cancel out both opposing influences, leaving the probability of causalities equal to the chance casualties when all parent variables are in their distinguished states, i.e., our prior belief in the child variable $P(y) = v_L = 0.40$.

Figure 5.10 is a visualization of how the PQ-model combines influences. To compare, Figure 5.11 is a visualization of how the CAST model combines influences. The x and y axis represent

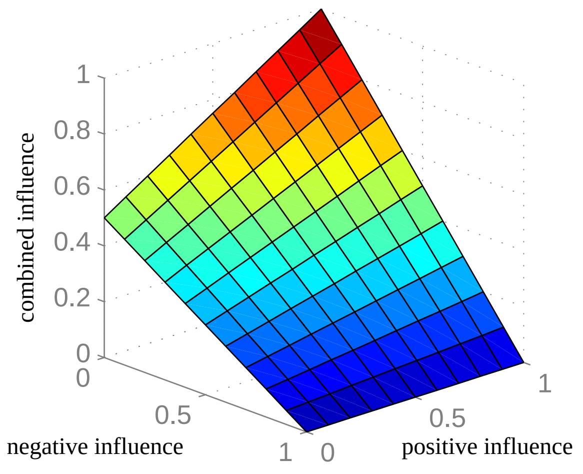


Figure 5.10: Positive and negative influence combined (PQ-model)

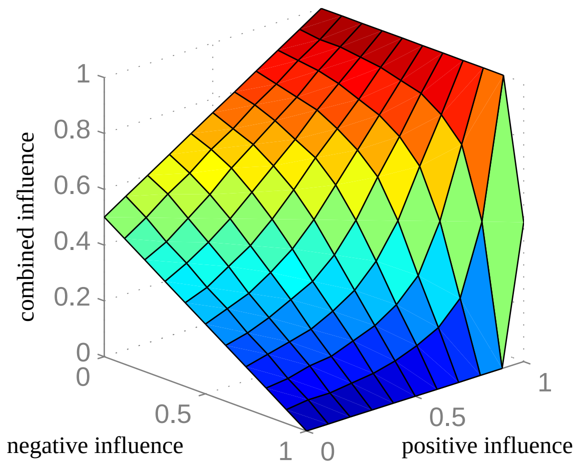


Figure 5.11: Positive and negative influence combined (CAST)

the strength of the positive and negative influence respectively. The z-axis represents the combined influence. Both models assume a prior/baseline of 0.5, i.e., when both the positive and negative influence is zero, the net influence is 0.5.

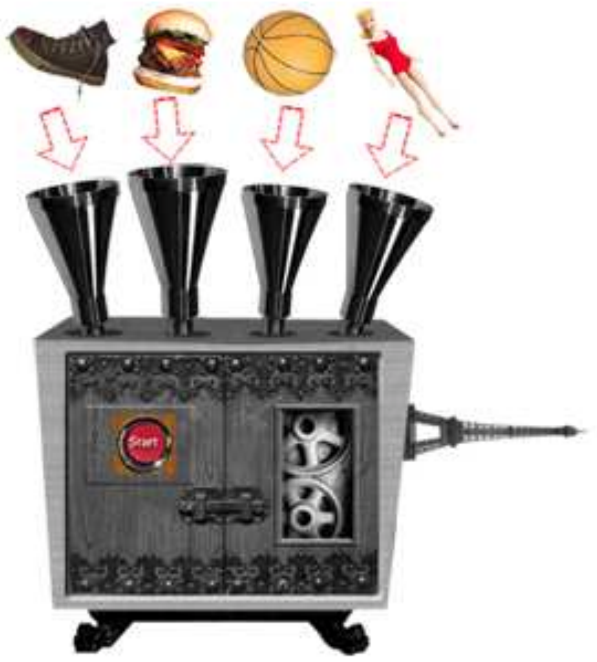
Notice that the CAST model is line-symmetric (z=0.5). Also, we see the asymmetry of the PQ-model. The cause for this is simple: propositional logic does not model opposing influences that cancel each other out. In the PQ-model, inhibiting influences are relatively stronger than promoting influences. In the given example, a negative influence $d = 1$ would prevent the child event, regardless of the strength of the positive influence. Asymmetry, however, is neither necessarily undesirable, nor uncommon in practice.

As pointed out by Lucas [16], there are 16 Boolean functions and each of them or even every combination of them can give raise to a canonical model. The PQ-model is capable of representing any Boolean function by its conjunctive form. Our proposal comes with a clear semantics and an intuitive method to elicit parameters from an expert. These, we believe, have contributed to the enormous popularity of such interactions as the Noisy-OR gate.

5.5 Summary

The PQ-model is a model that provides a reasoning framework applicable to qualitative problems in an uncertain domain. The most important property of the PQ-model is that it is able to handle a combination of positive and negative influences, while preserving both the amechanistic property and probabilistic soundness. By grace of the amechanistic property, all parameters for the PQ-model can be elicited by asking simple questions in the form $P(Y|\mathbf{X})$. This also guarantees that the parameters have a clear meaning: a parameter corresponding to a promoting parent, represents the probability that the uninhibited promoting parent produces the child event; a parameter corresponding to an inhibiting parent, represents the probability that the inhibiting parent prevents an otherwise successful outcome of the child event. Probabilistic soundness ensures that it is mathematically correct, and propositional logic, that lies at its foundations, ensures that our model is meaningful and intuitive for humans.

The fundaments that we used for the PQ-model are consistent with propositional logic. A PQ-model for which all parameters are set to the values of either 0 or 1, will act in accordance with propositional logic. Because our proposed model encompasses the basic logical operations OR, AND, and NEGATION (by switching the distinguished state), we are able to model any problem that can be composed as an expression in propositional logic. Furthermore, the PQ-model also allows for the distinguished states to be switched, which allows us to build comprehensive models.



6
An Experiment

The PQ-model, introduced in Chapter 5 obeys probability theory, propositional logic, and belongs to the class of amechanistic ICI models. Although these properties may be nice in theory, they are worth nothing if they do not prove their use in practice. To determine if the PQ-model is of any practical significance, we have conducted an experiment at the School of Information Sciences, University of Pittsburgh. To determine practical significance, we compare two different models, i.e., an elicited conditional probability table (CPT) and an elicited PQ-model, versus the knowledge of the experiment participants.

We want to alert the reader that the experiment described in this chapter, concerns the validity of the *theoretical* PQ-model, and bears no relevance to the software application discussed in Chapters 7 and 8. Neither are we out to measure how well the PQ-model can explain a given phenomena, as we are of the opinion that such a decision should be left to the knowledge engineer him- or herself.

What is it that we are after then? Our answer is that we want to find out how well the PQ-model corresponds to the structure of an experts *subjective knowledge.* In other words, we are interested in the difference of what the expert *thinks* he knows, and what he *actually* knows, and if that difference is smaller if we use a PQ-model, than if we use a conventional CPT.

For purpose of clarification, consider the following analogy. We flip a coin 10 times. An observer (the expert) sees 7 times heads, i.e., 'frequentist' probability of '0.7.' When we ask him later about the probability of heads, he answers '0.60.' Our goal is to show that the PQ-model minimizes the difference between the guessed 0.60 and observed 0.70. We do not care that the actual phenomena had only a 0.50 probability of occurring, because it is not relevant to our problem.

In Section 6.1 we briefly discuss the participants of our experiment. In Section 7 we elaborate on the experiment design and procedure. We present the results in Section 6.3 and discuss them in Section 6.4.

6.1 Subjects and implementation

Most of the participants (24) have been taken from a graduate course *Decision Analysis and Decision Support Systems* in the Department of Information Science, University of Pittsburgh. A prerequisite for this course is knowledge of elementary probability theory. In addition, a 4 computer science students from the Delft University of Technology participated, making a total of 28 participants. For the purpose of our study, we define a problem in a fictional domain. This ensures that subjects do not have prior knowledge that might bias our results. Subjects are not aware that the PQ-model is used as the underlying causal interaction model.

The application running the experiment is implemented in PHP, Javascript and MySQL. These languages allow for a fast implementation of a visual application, and the application will run on any computer with a browser and an Internet connection.

6.2 Design and procedure

The experiment will be administered by means of a computer application and will consist of three separate phases: (1) introduction to the problem, (2) experimenting with the model, and (3) eliciting parameters for the model. The subjects are introduced to the problem by the following text:

Looking for hidden treasures in an antique shop, you notice a curious machine. When asked, the shopkeeper explains that this machine is able to shrink human beings to miniature sizes, using a "shrinking ray." He tells you that the machine uses four common ingredients as fuel: old shoes, *hamburgers, basketballs, and child's dolls.He warns you that the ingredients can have one of the following two effects:*

(1) the ingredient either promotes the shrinking ray to work, or

(2) the ingredient inhibits the shrinking ray from working.

The shopkeeper also tells you that the machine does not behave deterministically: a promoting ingredient (1) is only effective to some degree, hence, sometimes the ingredient does not produce a shrinking process. An inhibiting ingredients (2) is only inhibitive to some degree, hence, sometimes the ingredient does not inhibit the shrinking process. Unfortunately, the shopkeeper has not been able to figure out which ingredients are inhibitive and which are promoting.

You decide to buy the machine and take it home with you. It is now up to you to find out for each ingredient (a) what type of influence it has and (b) to what degree it is effective. You have enough ingredients for 160 trials. In each trial, you can insert any combination of the ingredients into the machine, for instance: a hamburger and a shoe, but no basketball and no doll.

Be aware that the outcome of the machine is dependent on all four ingredients, so be sure to explore all possible configurations. Also, be aware that sometimes this machine is successful (shrinks a human) even when no ingredients are added. This possibility should also be examined.

Please do not take any notes while experimenting with the machine. Also, we request that you do not rush through the experiment, but take your time to explore all different combinations of ingredients. Finally, during the experiment, a second window will pop up. Please leave this window open, but move it to the background. Now examine the machine!

6.2.1 Underlying model

The actual outcome of the machine will be generated by a PQ-model, to guarantee that the behavior of the 'shrinking machine' can be modeled by both the PQ-model and a CPT with equal accuracy. To avoid the possibility that our results become the artifact of an unfortunate choice of parameters, we will assign random parameters to this model for each experiment participant.

To ensure that promoting influences remain identifiable to subjects, we will set a relatively low prior belief. This means that inhibiting influences can only be detected if they are observed simultaneously with promoting influences. In other words, inhibiting something that is not going to happen anyway, does not produce any observable effect. To avoid this caveat, we always have

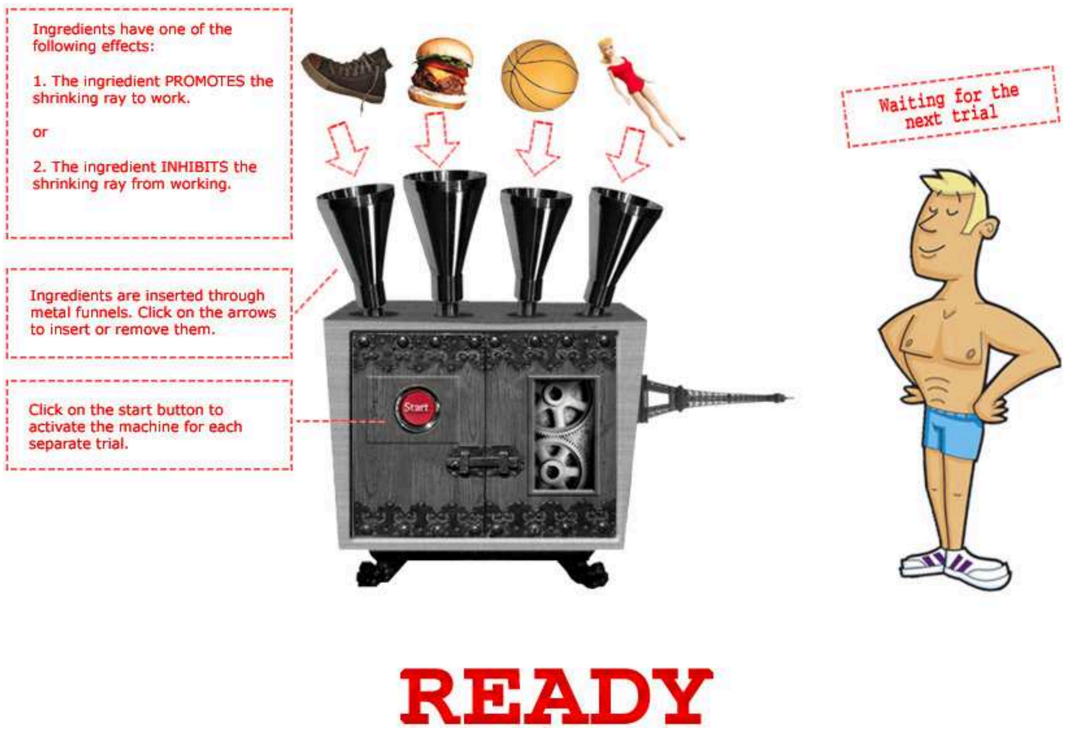

Ingredients have one of the following effects:
1. The ingriedient PROMOTES the shrinking ray to work.
or
2. The ingredient INHIBITS the shrinking ray from working.
Ingredients are inserted through metal funnels. Click on the arrows to insert or remove them.
Click on the start button to activate the machine for each separate trial.
Start
Waiting for the next trial
READY

Figure 6.1: Screenshot taken from the experiment.

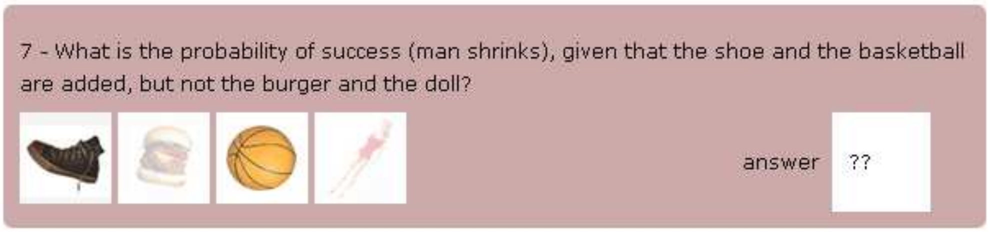
two promoting parents and a two inhibiting parents. For the same reason, both promoting parents are assumed to be reasonably strong (range [0.5, 0.9]. The inhibiting influences are randomized within range [0.3, 0.9]. The leak probability is assumed to be low [0.1, 0.3].

We would like to remind the reader that, other than generating the machine behavior, the underlying model is of no further importance to us. Like we emphasized in the introduction of this chapter, the purpose of this experiment is to compare the output generated by an elicited PQ-model to what an expert has *actually seen*, an not the observed phenomena itself.

6.2.2 Procedure

After the introduction, the subject will be allowed to use the machine for 160 trials. We use four binary input variables, implying that there are $2^4 = 16$ distributions, each one corresponding to a particular configuration of the ingredients. We have chosen the number of trails so that there are exactly 10 trials available for each distribution. Although we urge participants to try every possible distribution in the introduction, we do not enforce it.

We choose a fictional knowledge domain for two important reasons. We mentioned the first reason, i.e., we eliminate any unfair advantages due to prior knowledge within the domain. The second reason is that we can keep track of 'what the participant knows,' i.e., the experts knowledge.


7 - What is the probability of success (man shrinks), given that the shoe and the basketball are added, but not the burger and the doll?

answer ??

Figure 6.2: Eliciting the distributions for the CPT.

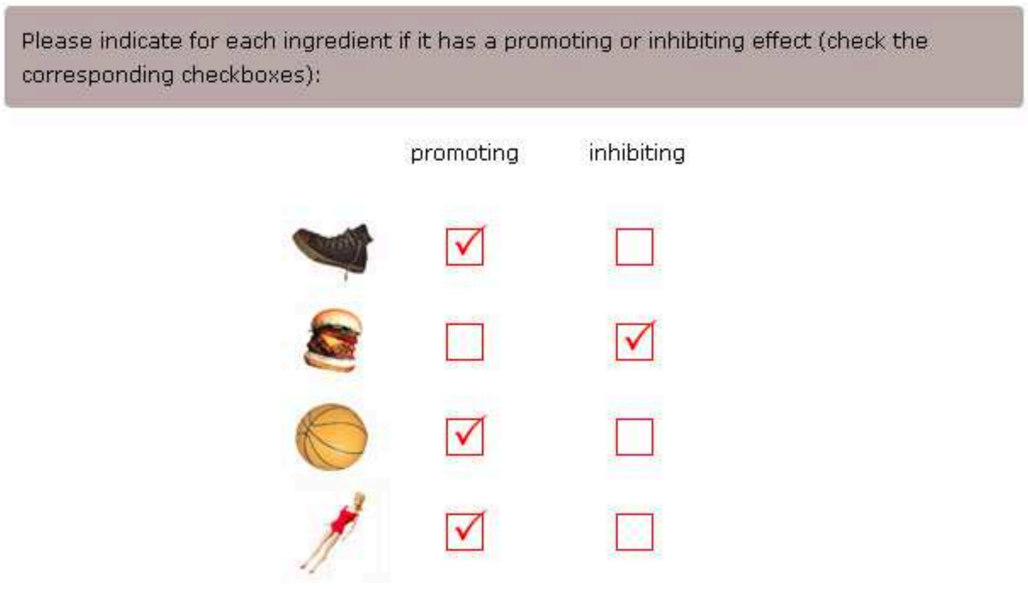
For each separate trial we record (1) the configuration of the ingredients, and (2) whether or not the participant observed success. With this information we can build a CPT that stores the distributions that were actually *observed* by the participant. In other words, we keep track of what the expert actually knows. This allows us to compare an elicited model with the experts knowledge, rather than the underlying model. This methodology was introduced by Wang et al. [29].

6.2.3 Elicitation

After all trials are completed, we elicit the model parameters from the participants. Due to the expected small number of subjects, we choose to use a within-subject design, as opposed to a between-subject design. Because the subjects will be asked to provide parameters for both the PQ-model and a conventional CPT, the knowledge elicitation is susceptible to the carry-over effect. To average out this effect, we randomize the order of the elicitation for both distributions. We ask for three distributions:

(1) Conditional probability table We elicit the entries for a full CPT. Given that we have four parent variables, this amounts to 16 entries. This ensures that the the size of the CPT is within reasonable bounds from an expert elicitation point of view. The participants are asked for the probability of success for each ingredient configuration. Figure 6.2 shows one of the sixteen questions as presented to the participants.

(2) PQ-model We elicit the parameters for the PQ-model. First we ask the subjects to assign a type of influence to each of the ingredients, i.e., promoting or inhibiting, as shown in Figure 6.3). Once the types of influence have been assigned, we ask one question for each individual ingredient. The question depends on the type of influence: for promoting ingredients we ask for the probability of success, given that only that particular ingredient is added (and none of the others); for inhibiting ingredients we ask for the probability that the ingredient *prevents* an otherwise successful outcome.


Please indicate for each ingredient if it has a promoting or inhibiting effect (check the corresponding checkboxes):

	promoting	inhibiting
	✓	
		✓
	✓	
	✓	

Figure 6.3: Eliciting the type of influence.

For example, suppose the subject identifies the shoe as a promoting influence, and the burger as an inhibitive influence. For the shoe we ask: *To what extent does the shoe CAUSE a successful outcome of the machine?* For the burger we ask: *To what extent does the burger PREVENT an otherwise successful outcome?* Finally we query the chance of success if no ingredients were added. These questions correspond to the elicitation as should be applied to the PQ-model. The order of elicitation (1) and (2) are randomized to compensate for the possible carry-over effect.

6.2.4 Experiment objectives

For the experiment we define the gold standard as the observed CPT, i.e. the CPT containing the probability distributions actually observed by a participant. We define two goals for this experiment:

(1) Expert knowledge representation The first objective of the experiment is to measure (a) the discrepancy between the gold standard and the probability distribution generated by the elicited PQ-model parameters, and (b) the discrepancy between the gold standard and the probability distribution provided by the elicited CPT.

(2) Identifying influence types The second goal is to determine the accuracy of the subjects in determining the type of influence for each parent: inhibiting or promoting. This will be done by simply dividing the number of correct subjects by the total number of subjects.

In Chapter 5, we showed that the PQ-model also supports that a parent acts as 'requirement' and 'barrier.' However, we will not explicitly include these types of interaction in the experiment knowledge domain for two reasons. The first reason is that, in the PQ-model, a requirement/barrier is the equivalent of an inhibiting/promoting cause with 'present' as the distinguished state. Therefore, it is not strictly necessary to include these type of interaction, since they are redundant. Our aim is to research how the experiment participants model opposing influences.

Including requirements/barriers is difficult for another reason as well: the distinguished state is 'present.' In the context of the shrinking machine, this would mean the 'added' state of an ingredient, i.e., the ingredient does not have influence when *added* to the machine, but when it is is *not added* to the machine. This may be counter intuitive within the context of the experiment, and result in unwanted confusion with the participants.

6.3 Results

We define each value of the observed CPT by calculating the maximum a posteriori estimate, using a Beta prior distribution with a very small equivalent sample size. We do so to avoid undefined entries for input configurations that are not observed. The estimated value of each entry in the observed CPT is given by:

$$e_j = \frac{s_j + 0.01}{t_j + 0.02} \, , \tag{6.1}$$

where s_j denotes the number of successful trials, t_j denoted the total number of trials, and c_j denotes the entry in the observed CPT for ingredient configuration j.

We measured the difference between the observed CPT and the elicited probability distributions by the averaged Euclidean and Hellinger [14] distance. Another commonly used measure is the Kullback-Leibler distance. A limitation of the Kullback-Leibler measure is that it cannot handle zero probabilities, and for this reason we chose not apply it. The Euclidean and Hellinger measures can only be applied to determine the distance between two single distributions (and not two CPTs), thus we express our results as the averaged distances over all the distributions in the observed CPT.

The subjects each took between 30 and 45 minutes to complete the experiment. We judged one of the subjects to be an outlier, and consequently excluded this subject from further analysis. This particular subject very likely confused the concept of inhibiting with promoting, and vice versa, as very high probabilities were observed, yet very low probabilities were elicited.

Figure 6.4 shows raw data, i.e., the Euclidean distance for each subject: (1) the distance between

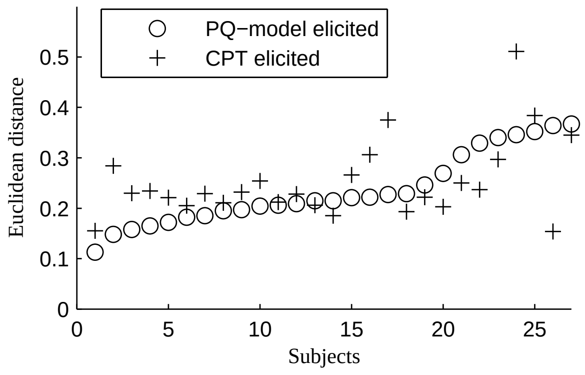


Figure 6.4: Raw data (sorted by increasing Euclidean distance for the PQ-model)

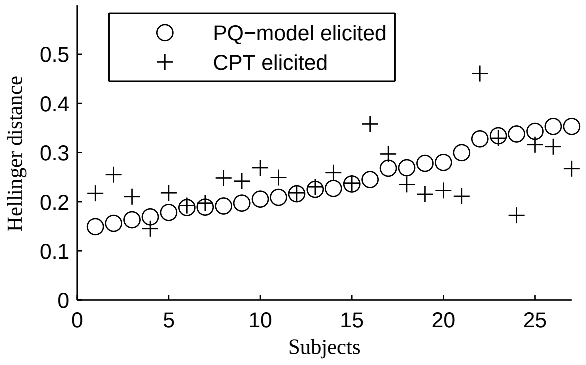


Figure 6.5: Raw data (sorted by increasing Hellinger distance for the PQ-model)

the observed CPT and the CPT generated by the elicited PQ-model, sorted from the smallest to the largest distance, and (2) the distance between the observed CPT and the directly elicited CPT. Figure 6.5 shows the same, using the Hellinger distance as a measure. We would like to point out that the range of distances is lower for the PQ-model. Table 6.1 shows the averaged Euclidean and Hellinger distances.

Table 6.1: Averaged Euclidean and Hellinger distances.

Measure	**Distance**
Averaged Euclidean Distance PQ-model	0.2364
Weighted Hellinger Distance PQ-model	0.2440
Averaged Euclidean Distance CPT	0.2528
Weighted Hellinger Distance CPT	0.2512

To determine the statistical significance of this result, we performed a one-tailed paired t-test for both distance measures. *Paired*, because each (PQ-model) distance in the first sample has a corresponding (CPT) distance in the other sample, i.e., of the same participant. *One-tailed*, because we are only interested to see if there is a significant difference between the two means. A t-test determines the probability that an observed difference, between the means of two paired samples, is due to chance, rather than an actual difference in the underlying model. First, we calculate the the t-value for the paired t-test:

$$t = \frac{\sqrt{n}\mu_z}{\sigma_z^2} \ , \tag{6.2}$$

in which z denotes the set of differences, i.e., one for each participant, between the distance measure for the PQ-model and the CPT. We denote n as the number of pairs. We then use the t-value, the *degrees of freedom* $(n-1)$, and the *t distribution* to determine the probability that the observed difference is due to chance. Comparison of the PQ-model elicitation versus direct CPT elicitation yielded $P \approx 0.14$ for the Euclidean, and $P \approx 0.29$ for the Hellinger distance.

Out of the 27 analyzed participants, 19 participants correctly identified all four influence types of the ingredients, 6 participants mis-identified one ingredient, and 2 participants mis-identified two ingredients. This means that overall 10 ingredients were misidentified. Of those 10 ingredients, 8 were identified as ‘promoting,’ while their type was in fact ‘inhibiting,’ and 2 in were identified as ‘inhibiting,’ while their type was in fact ‘promoting.’

6.4 Discussion

We discuss the results of our experiment by reviewing our goals:

(1) Expert knowledge representation A generally accepted threshold for statistical significance is $P < 0.05$. Although suggestive, the results of our experiment, i.e., $P \approx 0.14$, showed that there is no statistically significant difference between a CPT generated by the PQ-model and a CPT elicited directly. Therefore, we have to assume that the PQ-model and the CPT performed with comparable accuracy for this particular experiment.

(2) Identifying influence types 70 percent of the participants were able to correctly distinguish inhibiting from promoting ingredients. We notice that it is more common to confuse ‘inhibiting’ for a ‘promoting’ influence than vice versa. This result suggests that opposing influences, such as structured by the PQ-model are fairly transparent to human experts.

We identify two aspects of the experiment that leave room for improvement. First of all, the overall knowledge of the participants concerning probability theory and uncertainty was somewhat lacking (even though this was prerequisite for their class). A possible solution is to give participants a short introduction, reviewing the relevant theory. A second improvement is to elicit the prior probability only once. We elicited the prior probability once for each of the three models. However, we noticed that in several cases the same participant gave a different prior for each model, while the question was, in fact, the same. This may be due to the expectation of the participant that no question would be asked more than once. This confusion results in faulty parameters, obscuring the results of the experiment.

Although the results are not statistically significant in favor of the PQ-model, they do suggest that the CPT defined by 9 questions, eliciting only (a) the type and (b) strength for each ingredient, and (c) the prior, is, at least, *just as accurate* as eliciting the probability of success for all 16 input configurations. This becomes a non-trivial advantage when the number of parent variables starts to grow. And so, for a family with 10 parent variables, we have 21 questions for the PQ-model, versus 1024 questions needed to elicit the CPT directly.

In this experiment, we have not considered how well the PQ-model fits to an arbitrary problem. In fact, we have simply assumed that for both a PQ-model and a CPT, a perfect fit is possible. This will hardly ever be the case in a more practical environment. A knowledge engineer is left with the task of judging whether an interaction between a node and its parents can be modeled by means of a PQ-model. However, this was not our goal. Our goal was to prove that the PQ-model accurately models the (subjective) expertise of a human. In other words, should the PQ-model be applicable, it will outperform a conventional CPT. Although we have failed to produce statistically significant conclusion, our results strongly suggest that we have succeeded.

7 System Requirements and System Design

In Chapter 5, we proposed a theoretical model, baptized 'the Probabilistic Qualitative Model.' In Chapter 6, we have seen that the PQ-model is able to represent an expert's knowledge with reasonable accuracy. Our next goal is to build a software application that implements this theoretical model. The goal of this application is to provide a reasoning framework for an arbitrary problem, without requiring the the modeler (user) to be an expert on Bayesian networks and ICI models.

The trajectory for building this application can be roughly divided in four consecutive phases: requirement analysis, system design, implementation and testing. In this chapter, we will discuss the system requirements and design. Implementation and testing are reserved for the next chapter.

Our approach in the next two chapters is loosely based on the software engineering methods discussed in [1]. We remark that documenting a fully fledged software engineering project is beyond the scope of this thesis, and we shall therefore only discuss the most important aspects.

We begin by looking at the system requirements in Section 7.1. In Section 7.2, we propose a system design. Finally, in Section 7.3 we propose a design of the Graphical User Interface (GUI).

7.1 System requirements

In this section we state the constraints, requirements and behavior of the extensions to the SMILE and GeNIe software. We discuss the relevant actors, use cases, and requirements.

7.1.1 Identifying actors

We identify three different actors that are part of the application domain. We identify the `domain expert`, the `network modeler` and the `decision maker`. It might very well be that in some cases, two, or event all three actors are the same person. However, the tasks they perform in relation to the system are subtly different, thus we model them as three different actors.

The `domain expert` is a person that has extensive knowledge/experience of the problem domain. He or she is subject to the elicitation process conducted by the `network modeler` . Even though the `domain expert` does not interact with the system directly, he/she plays an important role in the application domain and we will consider him/her an actor for reasons of completeness.

The `network modeler` is the person that elicits the network structure, influence types and corresponding parameters from the `domain expert`. After the elicitation process, the `network modeler` builds a model that captures the knowledge of the `domain expert` . The `network modeler` interacts directly with the system, but only for the purpose of building a network.

The `decision maker` is the person that uses the network structure in order to make a better informed decision about a problem within the modeled domain. By adjusting the prior belief parameter of several variables, the `decision maker` is able to try different scenarios, observing the interactions and consequences. The `decision maker` does not change the influence strengths, because they are assumed fixed after elicitation.

7.1.2 Identifying use cases

A use case is a formal description of a particular feature of the proposed system. We use them for two purposes: (1) to define the behavior of the proposed system before implementation and (2) to validate the system behavior after implementation. In a greenfield engineering project, i.e., engineering a project from scratch, use cases are used to derive the object model. However, in our case, the object model already exists and we have to abide by the existing rules and constraints to ensure consistency.

Figure 7.1.2 shows the eight most important use cases concerning our extension of the SMILE and GeNIe software. We identify them as: `CreateDeMorganNode`, `CreateArc`, `DeleteNode`, `DeleteArc`, `SetPrior`, `SetInfluence`, `SaveNetwork`, and `LoadNetwork`.

Use case name	`CreateDeMorganNode`	Use case name	`CreateArc`
Actor	Executed by `network modeler`	Actor	Executed by `network modeler`
Entry condition	1. The `network modeler` left-clicks the 'DeMorgan node' button.	Entry condition	1. The `network modeler` left-clicks the 'create node' button.
Flow of events	2. The `network modeler` left-clicks the location on the grid were the new node is to be placed. 3. The node name (default 'node') appears in an active edit box, centered in the node.	Flow of events	4. The `network modeler` left-presses the intended parent node. 3. While pressed, the `network modeler` drags the arc over the intended child node. 4. The `network modeler` releases the mouse-button to create the node. A menu requesting the arc type appears.
Exit condition	4. The `network modeler` enters the name of the node and hits 'Enter.'	Exit condition	5. The `network modeler` chooses the arc type by left-clicking.
Use case name	`DeleteNode`	Use case name	`DeleteArc`
Actor	Executed by `network modeler`	Actor	Executed by `network modeler`
Entry condition	1. The `network modeler` selects the node that is to be removed by left-clicking.	Entry condition	1. The `network modeler` selects the arc that is to be removed by left-clicking.
Flow of events	2. The `network modeler` hits the 'Delete' button. 3. The `network modeler` is prompted by a confirmation dialog box. 4. The `network modeler` selects 'Yes' by left-clicking.	Flow of events	2. The `network modeler` hits the 'Delete' button. 3. The `network modeler` is prompted by a confirmation dialog box. 4. The `network modeler` selects 'Yes' by left-clicking.
Exit condition	5. The node and all connected arcs are removed from the network.	Exit condition	5. The arc is removed from the network.
Use case name	`SetPrior`	Use case name	`SetInfluence`
Actor	Executed by `decision maker`	Actor	Executed by `network modeler`
Entry condition	1. The `decision maker` double-clicks the node for which he/she wants to set the prior belief.	Entry condition	1. The `network modeler` double-clicks the arc for which he/she wants to set the influence parameter.
Flow of events	2. The `decision maker` is prompted by a dialog box containing slider, representing the prior belief. 3. The `decision maker` adjusts the slider with the left mouse-button. 4. The `decision maker` hits 'Enter.'	Flow of events	2. The `network modeler` is prompted by a dialog box containing slider, representing the influence parameter. 3. The `network modeler` adjusts the slider with the left mouse-button. 4. The `network modeler` hits 'Enter.'
Exit condition	5. All node values in the network are updated, given the new prior.	Exit condition	5. All node values in the network are updated, given the new influence.
Use case name	`SaveNetwork`	Use case name	`LoadNetwork`
Actor	The `network modeler` or `decision maker`	Actor	The `network modeler` or `decision maker`
Entry condition	1. The `actor` selects the 'save' option from the menu.	Entry condition	1. The `actor` selects the 'open' option from the menu.
Flow of events	2. The `actor` is prompted by a 'save dialog.' The `actor` specifies a name and location for the file that contains the current network.	Flow of events	2. The `actor` is prompted by an 'open dialog.' The `actor` specifies the location for the file that contains the current network to be loaded.
Exit condition	3. The `actor` left-clicks the 'save' button. The network is saved in the specified file.	Exit condition	3. The `actor` left-clicks the 'open' button. The network is opened.

Figure 7.1: Use cases for the extension DeMorgan of SMILE and GeNIe

7.1.3 Functional requirements

Functional requirements describe the interactions between the system and its environment independent of its implementation. We state the functional requirements for our extension of the SMILE library and the GeNIe interface:

The SMILE library and GeNIe interface must be extended to so that it supports models using the the DeMorgan-gate. This means that is able to model the the four types of arcs for (1) promoting events, (2) barring events, (3) inhibiting events, and (4) required events. Each arc can be assigned a weight representing the strength of the corresponding influence. The DeMorgan nodes can be assigned a prior belief. SMILE is able to calculate the value of each variable (node) in the network, given all priors belief parameters and all influence parameters. SMILE can store models using the DeMorgan-gate and is also able to restore models that were stored earlier.

7.1.4 Non-functional requirements

Non-functional requirements describe the aspects of the system that are not directly related to the functionality of the system, but are still visible for users. We state the non-functional requirements for our extension of the SMILE library and the GeNIe interface:

All added functionality to SMILE has to be documented and added to the SMILE on line documentation. The extension of the SMILE library must be consistent with the current structure. The extension of the GeNIe interface must be consistent with the current GUI. This means that DeMorgan nodes and arcs are represented the same way as in the current GUI. The added functionality is focused at building qualitative models. Consequently, node values and influences must be represented in a qualitative fashion, rather than quantitative fashion, as is the case in the current version of GeNIe.

7.1.5 Pseudo requirements

Pseudo requirements are constraints that affect the implementation of the system, but are not visible to users. We state the pseudo requirements for our extension of the SMILE library and the GeNIe interface:

All extended functionality of SMILE and GeNIe is implemented in the C++ language. All extended functionality must be fully compatible with current functionality.

7.2 System Design

In a software engineering project, we discern two consecutive design phases: 'system design' and 'object design.' During system design, we focus specifically on the functionality of the system, i.e., independent of its environment. During object design, we adapt our initial system design in order for it to 'fit' into a particular environment.

In this section, we propose a system design *independent of GeNIe and SMILE*. In Chapter 8, we will discuss the object design, i.e., how our independent design can be adapted to fit into the GeNIe and SMILE environment.

7.2.1 Design goals

Design goals identify the qualities that our system should focus on. Over the years the Decision Systems Laboratory has had many temporary members that have worked on GeNIe and SMILE in the light of their research. This means that, in particular, the SMILE architecture has been altered numerous times to be able to include these new extensions. Consequently, the SMILE architecture has increased significantly in complexity.

Consider the metaphor of a castle. Each year, several new towers are build. However, sometimes there is no solid foundation for a new tower and the new tower is connected to the side of another tower. Suppose this process continues for several years, and many new towers are built. Eventually the castle will collapse onto itself. To prevent this, and to ensure that SMILE can support future extensions, we define as our two main and design goal:

- consistency with current system architecture
- maintainability and expandability

Our first goal applies mainly to the object design. We satisfy our second goal, by adopting an object oriented approach. We expect future extensions to our system. We also expect these changes will concern system functionality, rather than the systems data structure. For this reason, we need to separate the 'data structure' from the 'functionality.' In the next section we propose a solution.

7.2.2 Identifying Classes

In a Bayesian network, all the information is stored on a local level, i.e., within the nodes and arcs. In other words, there is no global information vital to the functionality of the network. Such an architecture translates nicely to an object oriented approach for our system design. Initially, we define four different object classes that constitute our system design:

- a node object (`QNode`)
- an arc object (`QArc`)

- a network container object (`QNetwork`)
- a CPT object (`CPT`)

The `QNode` and `QArc` objects are the building blocks of a network. Each `QNode` keeps track of one single `CPT`object and all the `QArc` objects which are connected to it. Note that the `CPT` object could have been easily integrated into the `QNode` object. However, in favor of a more object oriented structure, we chose not to do this. The `QNetwork` is simply a container-class for the network. Its purpose is to perform global operations on the network, such as inference, and consistency checking. We chose to implement these global functions using the 'visitor' design pattern.

The visitor design pattern is a way of separating a global function from the data elements (`QNodes`). The advantage of this approach is that we can add new functions without altering our data structure, e.g., when we want to implement a new inference algorithm, we will not have to redefine the `QNode` class. Instead, for each (new) function, we define a visitor object that traverses through the nodes in the network and executes its assigned function. We define two visitor objects:

- checking graph consistency (`ConsistencyVisitor`)
- applying inference (`InferenceVisitor`)

The `ConsistencyVisitor` checks to see if the graph is still a Directed Acyclic Graph (DAG), whenever a new arc is created. If not, the new arc is illegal and cannot be created. The `InferenceVisitor` calculates the value of each node, given the current state of the network.

7.2.3 Class Architecture

Now let us take a closer look at the `QNode` , `QArc`, and visitor objects, and how they behave in relation to each other.

QNode object We consider the attributes, references and operations respectively. A `QNode` object has only three basic attributes:

- a node value (Value)
- a prior belief parameter (Prior)
- a flag parameter (Check)

```
void UpdateCPT()
{
    double entry = Prior;
    for each (configuration of parent states ‘c’)
    {
        for each (parent ‘p’ in ‘c’)
        {
            int type = p.getType();
            switch(type)
            {
                case (type is CAUSE):
                    if (state of ‘p’ is TRUE)
                        entry += (1-entry) * p.getWeight();
                    break;
                case (type is BARRIER):
                    if (state of ‘p’ is FALSE)
                        entry += (1-entry) * p.getWeight();
                    break;
            }
        }
        for each (parent ‘p’ in ‘c’)
        {
            int type = p.getType();
            switch(type)
            {
                case (type is INHIBITOR):
                    if (state of ‘p’ is TRUE)
                        entry = entry * (1-p.getWeight());
                    break;
                case (type is REQUIREMENT):
                    if (state of ‘p’ is FALSE)
                        entry = entry * (1-p.getWeight());
                    break;
            }
        }
        CPT.setEntry(for configuration ‘c’, for ‘p’ state TRUE, entry)
        CPT.setEntry(for configuration ‘c’, for ‘p’ state FALSE, 1-entry)
    }
}
```

Figure 7.2: PQ-model algorithm

Each node in a Bayesian network has a value. In addition, each node in the PQ-model assigns a prior belief parameter. The flag parameter is meant as a check parameter that may be used for an arbitrary purpose by the visitor objects. For example, when applying inference, the visitor objects may need to ‘mark’ the nodes that it already updated to prevent a node from being updated more than once during a single iteration.

The `QNode` object holds references to a single `CPT` object. Remember that a variable is dependent on all variables from which it is not D-separated, so a `QNode` needs to keep track of its parent and child nodes. It does so by holding a reference to each outgoing and incoming arc. The `QNode` object does not know the `QNetwork`, nor any of of he visitor objects, nor any `QArc` objects that it is not connected to.

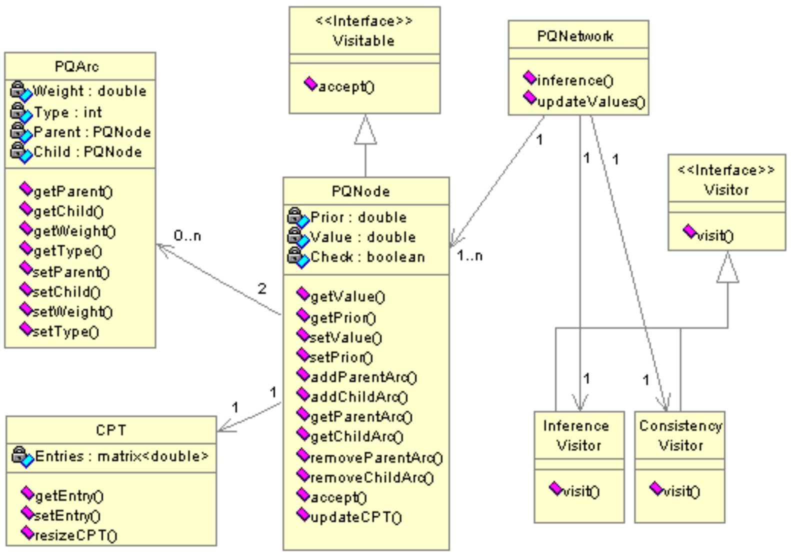


Figure 7.3: UML Class diagram: system design

Almost all of the operations in the `QNode` object are responsible for the handling of local attributes and references, e.g., for the Value attribute we have: getValue() and setValue().

There are two exceptions. The first is the `accept()` operation, which we will discuss later. The second exception is the `updateCPT()` operation. This operation updates the CPT object, given the weights and types of the arcs that connect to the parent nodes. In other words, this operation captures the functionality of the PQ-model, i.e., if we wanted to implement a different causal model, this operation would contain the corresponding algorithm. Figure 7.2 gives the pseudo-code implementation for `updateCPT()` operation.

It seems that updating the CPT from within the `QNode` object, is not consistent with the visitor design pattern, which aims to separate functionality from data elements. However, a CPT is determined purely on a local level, i.e., we only need to know the parent nodes to determine a CPT. Consequently, we only need to recalculate the CPT when the QNode is created, or a parent QNode is added/removed. For this reason, `udateCPT()` is implemented within `QNode`.

QArc object The `QArc` object is a simple object that keeps track of the direct dependency between two `QNodes`. We need only two attributes:

- a weight parameter (Weight)
- a type parameter (Type)

Both attributes represent parameters in the PQ-model. The `Weight` attribute defines the strength of the dependency, and the `Type` attribute stores if the parent acts as either a cause, barrier, inhibitor, or requirement. In addition, the `QArc` object holds two references: the parent `QNode` and the child `QNode`. The operations in the `QArc` object serve only to handle the attributes and references.

Visitor objects We have already briefly discussed the visitor design pattern, but now we will take a closer look. To ensure the smooth interaction of the visitor objects and the data elements, we define two interface classes.

Like a normal class, an interface class defines operations. It does not, however, contain any implementation. Instead, it ensures that another class (that implements that interface), implements the operations that are defined in the interface.

- Visitor
- Visitable

The Visitor interface ensures that a visitor object has a `visit()` operation. Note that a single visitor object can have multiple `visit()` operation, i.e., one for each object class that it may be called upon to visit. In our network, we have only type of data element, i.e., the `QNode` object. However, in the future, we may want to add different types of nodes, e.g., we may want to add a value-node, which, given the states of its parents, calculate some monetary value. Our initial system design contains only two visitor objects, one responsible for inference, the other responsible for the acyclic property of the network. There are multiple algorithms available for both tasks [30], however, which particular algorithms they will implement is not relevant.

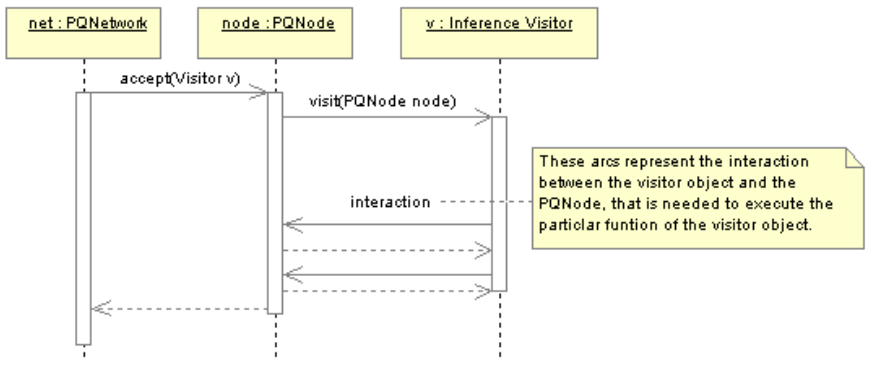

net : PQNetwork
node : PQNode
v : Inference Visitor
accept(Visitor v)
visit(PQNode node)
interaction
These arcs represent the interaction between the visitor object and the PQNode, that is needed to execute the particlar funtion of the visitor object.

Figure 7.4: Flow chart diagram: the visitor design pattern

A Visitable object, in our case `QNode`, must have an `accept()` operation. The `accept()` operation is called from the controlling object, in our case `QNetwork`), and passes an arbitrary visitor object as parameter. The QNode then invites that visitor object by calling the `visit()` operation of that visitor object, passing itself along as a parameter. Finally, the visitor applies its designated function to that `QNode`. This flow of events, depicted in Figure 7.4, is known as *double dispatching*.

Figure 7.3 shows our initial system design in a UML class diagram.

7.3 Graphical User Interface Design

Creating a good interface design is an iterative process. At every step during the design process, the designer takes into account the feedback of the user. One method that is often used is called 'prototyping,' meaning that the designer keeps on altering the same design than starting anew every time a change is requested. Such an approach, however, is beyond the scope of this thesis.

Before we start designing, we have to consider an important question: who is the user of our system? GeNIe and SMILE are used by a variety of companies, such as Intel, Boeing and Philips, but also organizations such as logistic branch of the US Air Force. Thus we envision the user to be the decision makers and modelers within such organizations. We assume the user has general understanding of Bayesian networks, e.g., he/she may not understand complex inference algorithms, but he/she understands what they do. We assume the user is proficient with computers, the English language, and does not suffer from any visual impairments. For the rest of this thesis, we refer the the GeNIe version that is extended with the DeMorgan model as QGeNIe (Qualitative GeNIe).

We also have to reconsider our design goal: consistency. We expect the users of QGeNIe to be, in large part, the same people who currently use GeNIe. This implies that, to avoid confusion, we do not change any old features/representations and we try to keep the new features/representations as consistent as possible. We will focus our design efforts only on the new features of QGeNIe.

In the next two sections, we will propose a number of design decisions, clarified by Figures 7.6, 7.7 and 7.8. As a frame of reference for the reader, Figure 7.5 shows a screenshot of the whole environment. Most of the initial design decision were inspired by Tufte [27, 26, 28], and several brainstorm sessions within the Decision Systems Laboratory (DSL).

7.3.1 Node design

The non-functional requirements state that nodes must be fashioned to represent a value qualitatively, rather than quantitatively. We choose to achieve this by using color. Remember that node

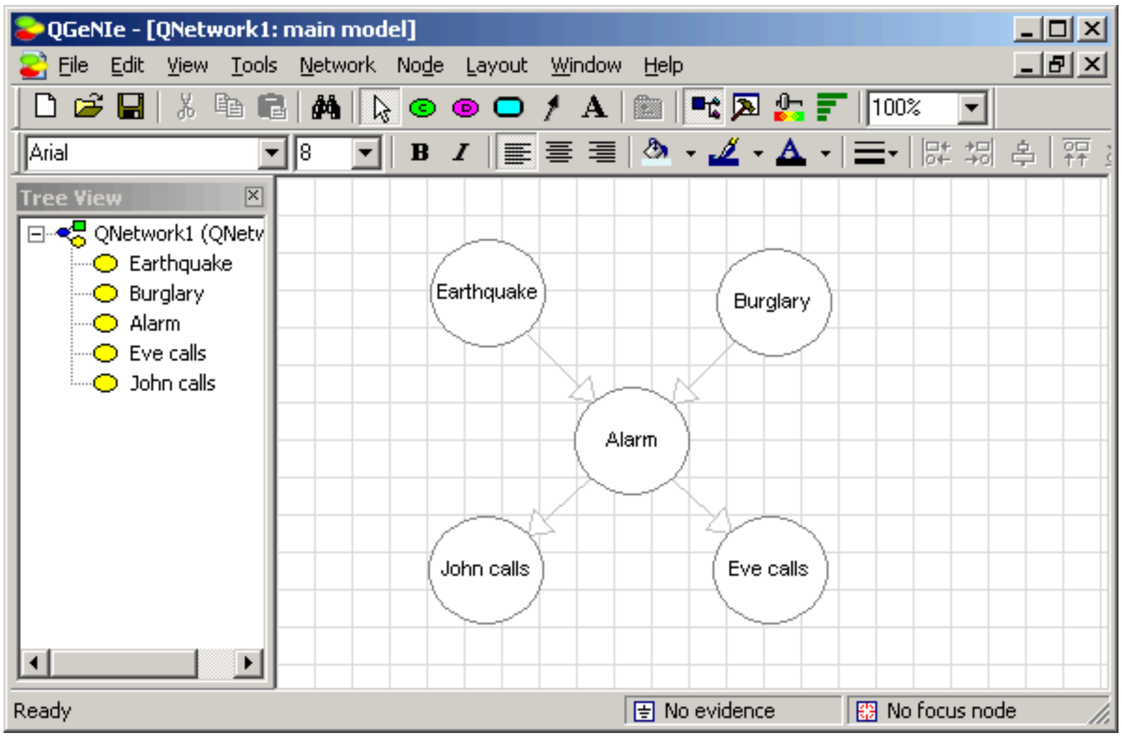


Figure 7.5: The GeNIe environment.

values are probabilities, i.e., values between 0 and 1, 0 representing 'false,' and 1 representing 'true.' To represent a node value, we use the following spectrum: red for 'false' ($P(x) = 0.0$), white for 'uncertain' ($P(x) = 0.5$), and green for 'true' ($P(x) = 1.0$). We chose white as the intermediate color, so that the *hue* of the node color represents the *polarity*, i.e., 'true' or 'false,' and the *saturation* represents the measure of *uncertainty*, i.e., $P(x) = 0.5$ represents total uncertainty and either $P(x) = 0$ or $P(x) = 1$ represents total certainty.

Sometimes, the color of a node is so close to $P(x) = 0.5$, that it becomes hard for the user to distinguish the polarity of the node. To solve this problem we include an anchor circle in the node representation. The left half of the anchor circle represents the polarity, i.e., the closest extreme. The right half show the color of value $P(x) = 0.5$, as a reference point for the user.

Finally, instead of requiring the user to give a point probability as the prior belief, we choose a slider dialog, as shown in Figure 7.8a, right representing 'true,' left representing 'false.'

7.3.2 Arc design

We design arcs in a similar fashion. The color of the arc-head represents whether the arc has a positive or negative influence on the child node, green representing a positive, and red representing a negative influence. The saturation of an arc-head represents the strength of the influence associated

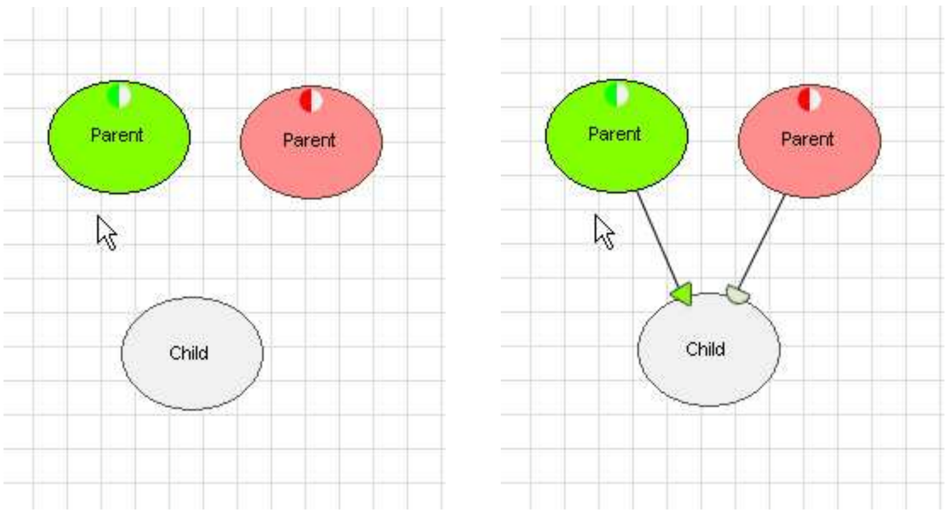


Figure 7.6: QGeNIe: (a) color represents node value, (b) arcs represent dependency.

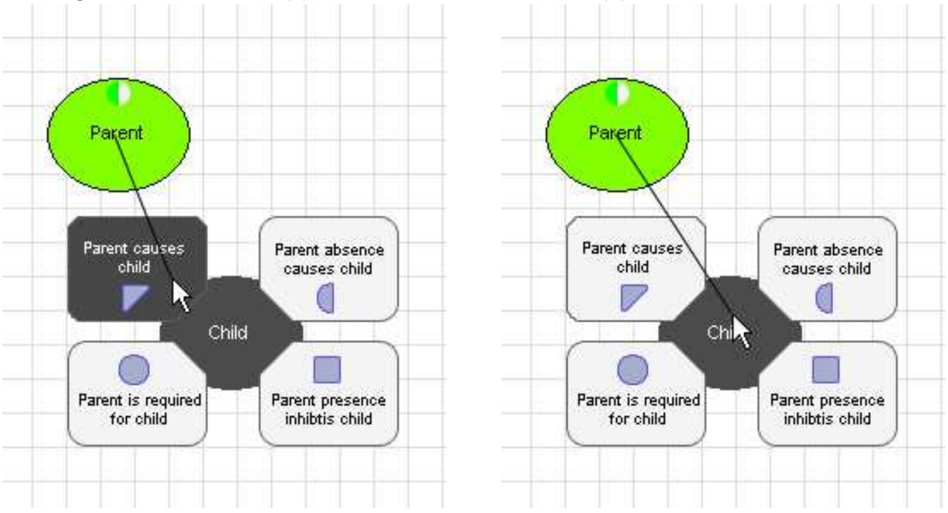

Parent
Parent causes child
Parent absence causes child
Child
Parent is required for child
Parent presence inhibtis child
Parent
Parent causes child
Parent absence causes child
Chi
Parent is required for child
Parent presence inhibtis child

Figure 7.7: QGeNIe: (a) drawing a new arc, (b) selecting the arc type.

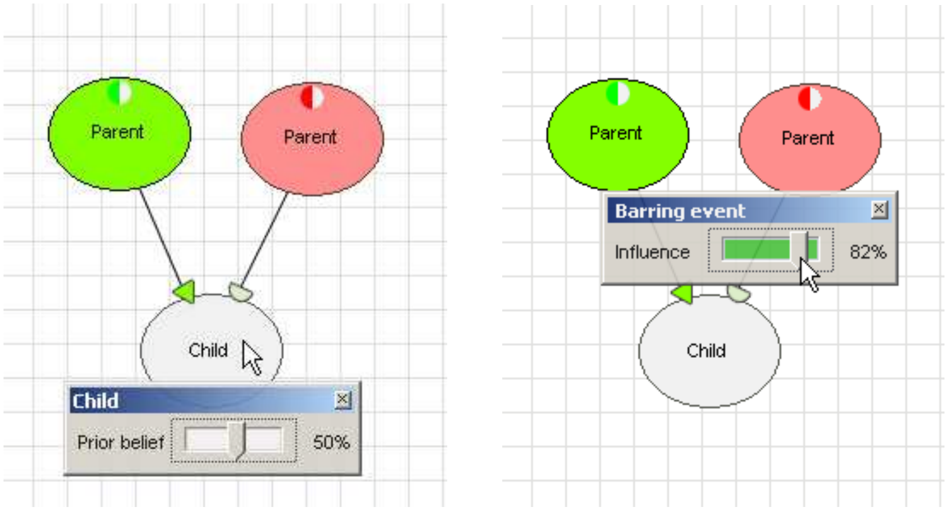


Figure 7.8: QGeNIe: (a) setting prior belief, (b) setting influence strength.

with that arc, i.e., a white arc-head implies independence of the child node, while a bright red/green arc implies a deterministic dependency. The shape of an arc-head represents the type of influence that the parent has, i.e., whether the parent is promoting (triangle), barring (half-circle), inhibiting (circle) or required event (square).

To create and arc between nodes (use case `CreateArc`), users first select the arc-tool from the menu, and then select the parent node. The drag the arc to the intended child node. After releasing the mouse-button, a menu pops appears, requesting the arc-type, as shown in Figure 7.7a and 7.7b. The menu gives a short explanation for each influence type.

Finally, instead of requiring the user to numerically specify the influence parameter, we opt to use a slider dialog, as shown in Figure 7.8b, right representing a deterministic dependency, left representing independence. To be able to distinguish arc that represent a deterministic relation, the arc itself is colored blue. Likewise, an arc that represents an independent relation (n.b. this is the equivalent of having no arc at all) is colored a very light gray.

8 System Implementation and System Testing: SMILE and GeNIe

SMILE (Structure Modeling, Inference, and Learning Engine) is a library of functions supporting the construction of Bayesian networks and influence nets. GeNIe (Graphical Network Interface) is an application providing a graphical user interface (GUI) for the SMILE library. Both SMILE and GeNIe are the product of the Decision Systems Laboratory, University of Pittsburgh, and has been available to the public since July 1998.

The functionality of SMILE and GeNIe have been extended numerous times by many different people. As a result, both systems have grown considerably in size and complexity since their initial release. To include the functionality for the PQ-model, parts SMILE and GeNIe need to be re-engineered. In the previous chapter we proposed a system and GUI design. The GUI changes were implemented by resident DSL members. The changes to SMILE were implemented by the author, and therefore, in this chapter, we will only consider the changes to SMILE.

8.1 Object Design

In this section, we will adapt our initial system design to the SMILE environment. First, we discuss the subsystem decomposition of SMILE/GeNIe. Subsequently, we discuss the the alterations and extensions that are needed to include the functionality of the PQ-model.

8.1.1 Subsystem structure

To reduce the complexity of large systems, we often decompose the system into smaller parts called subsystems. Subsystems are composed of classes that have a strong association with each other or have some common purpose within the system. Similarly, SMILE is decomposed in several subsystems. We identify six subsystems:

SMILE This subsystem contains all basic functionality needed for Bayesian networks and influence nets. This includes the building and managing of networks, managing Conditional Probability Tables (CPTs), managing different types of nodes (Noisy-OR, decision nodes), and inference.

SMILELearn This subsystem contains functionality needed for learning the parameters and CPTs for Bayesian networks from datasets.

SMILEXml This subsystem contains functionality needed for the storing and restoring of networks, CPTs, causal models, parameters etc.

SMILEMech This subsystem contains functionality for including equations into Bayesian networks, i.e., instead of inferring variable values from CPTs, we could use functions and equations to determine a value. This subsystem is currently under development and not yet part of the on-line GeNie/SMILE version.

SMILEInterface This subsystem is essentially a layer around SMILE. Its purpose is to guarantee that SMILE is fully portable, i.e., it can easily be adapted to an arbitrary environment.

GeNIe This subsystem provides a GUI for the SMILE library in the MS Windows environment, implemented in C++.

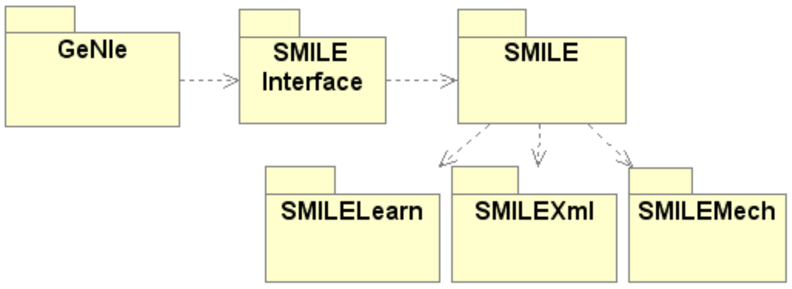


Figure 8.1: Subsystem decomposition: SMILE and GeNIe.

For the purpose of adding the PQ-Model functionality to SMILE/GeNIe, we need to alter and extend only three of the subsystems, i.e., SMILE, SMILEXml and SMILEInterface. We also need to make alterations to GeNIe. However, in the case of GeNIe, we only provide a design, as a full description of the object model and alterations is beyond the scope of this thesis.

8.1.2 Persistent data management

For the management of persistent data, i.e, stored networks, SMILE uses flat files structured by the eXtended Mark-up Language (XML). The main purpose of XML is to provide a standard of describing data, such that it can be used by a variety of different systems. One of the advantages of XML is that it its format is readable to both software and humans, e.g, when opening a XML file we can easily identify variables and their values.

In the next section, we discuss briefly how SMILE uses XML to store and restore Bayesian networks.

8.1.3 Adapting the System Design

It is rarely the case the the system design fits seamlessly into the target environment. Consequently, we have to adapt our design so that if 'fits' into the SMILE library. We have to make two adjustments and two extensions:

- Integrate the `PQArc` class into the `PQNode` class
- Abolish the visitor design pattern.
- Creating objects to load and store `PQNodes` objects
- Extending the interface object that communicates with QGeNIe

In SMILE, node objects keep track of their parents en children directly. Consequently, a node object also keeps track of any attributes that belong to the arcs, rather than the nodes, e.g, a weight or type parameter. This also implies that the node object gains several operators to handle these attributes. This adjustment was easily made.

The second adjustment concerns the visitor design pattern. Although SMILE uses a similar structure to separate the functionality from the data structure, it is slightly different. Instead of dispatching the 'function object,' e.g., an object responsible for inference, to the node objects, the network object serves as a mediator, controlling the communication between them. Even though this works just fine, it a sloppy solution from an object oriented point of view.

Every node in SMILE holds a reference to a CPT object. All 'function objects' directly access this object when they need to. Because of this, we do not have to make drastic adjustments to our design.

Surprisingly, SMILE *does* use the visitor design pattern to store networks, so we have to extend the visitor object with a `visit()` operator for a `PQNode` object. For loading networks, SMILE uses special 'reader' objects. Consequently we have to implement such a reader object for the `PQNode` object.

Finally we have to extend the objects that are burdened with the communication with QGeNIe.

8.1.4 Implementation

We review the class diagrams that correspond to the three subsystems that were altered. We discuss several class diagrams for the purpose of clarifying the alterations to SMILE. To give the reader an idea of the changes that were needed, we show only the operations and attributes that were modified or added. Because many classes and associations are irrelevant for explanations purposes. To prevent the diagrams from becoming unreadable, we chose to omit some classes from our diagrams.

SMILE Figure 8.2 shows the class diagram corresponding the SMILE subsystem.

The single most important class in SMILE is `DSL_network`. The `DSL_network` keeps one instance of the `DSL_nodeEntry` class, which manages all the nodes in the network. The type of a `DSL_node` is stored in an instance of `DSL_nodeDefinition`. Likewise, the value of a node is stored in an instance of `DSL_nodeValue`. To include PQ-model nodes, we implemented the class `DSL_PQNode`, that inherits from `DSL_nodeDefinition`. Because the value of a `PQNode` represents a measure of belief, we use the already existing `DSL_beliefVector` to store values.

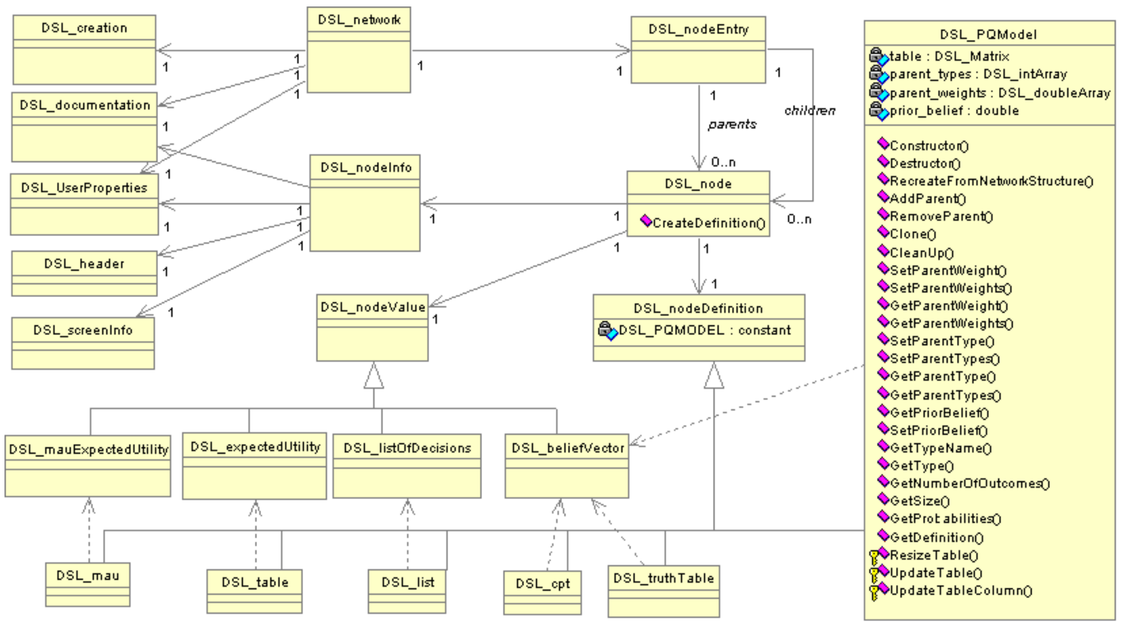


Figure 8.2: Class diagram: SMILE.

Because influence types of incoming arcs and the corresponding strengths are all properties of a child node, the class `DSL_PQNode` includes all functionality such as described in Chapter 5. The most important operation in this class is `UpdateTable()`, which calculates the CPT (attribute table: DSL_Matrix), given the current parent values, influence types, strengths and prior belief.

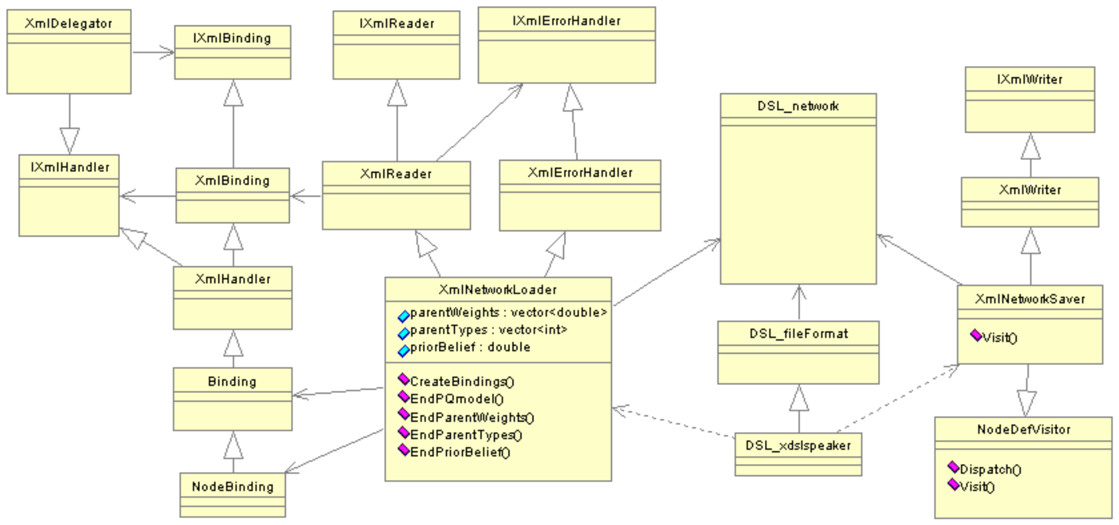


Figure 8.3: Class diagram: SMILE Xml.

SMILEXml Figure 8.3 shows the class diagram corresponding the SMILEXml subsystem. Saving a network is straightforward. The class `XmlNetworkSaver` is a visitor object that visits all the nodes in a DSL_network. XmlNetworkSaver has a different `Visit()` operations for each node type. This operation stores all information that is contained in the visited node, ergo, to be able to store a PQ-model node, we include a `Visit()` specifically for the `PQNode`.

Loading a network is slightly more complex. The class `XmlNetworkLoader` is parser, translating XML to networks and nodes. `XmlNetworkLoader` keeps track of a large number of bindings. An instance of `Binding` contains (1) an XML tag and (2) the operation that is able to extract the value corresponding to that tag, and transform it into an entity, e.g., node value, in a `DSL_network`. Thus we include code that creates the bindings needed for the `PQNode`, i.e. in `CreateBindings()`, and the operations needed to extract PQ-model values.

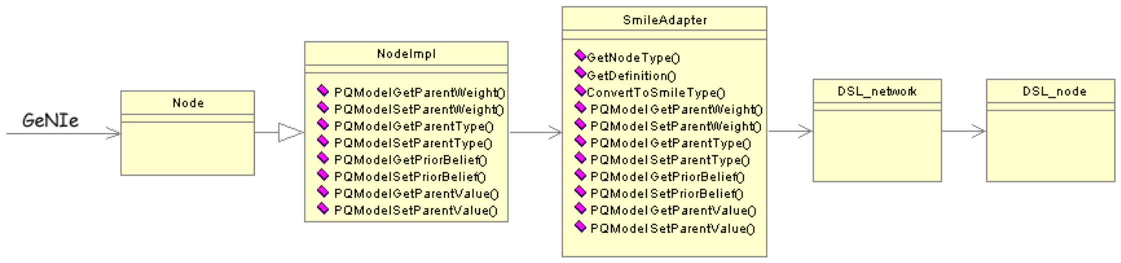


Figure 8.4: Class diagram: SMILE Interface.

SMILEInterface Figure 8.4 shows the class diagram corresponding the SMILEInterface subsystem. The class `Node` represents a node in the GeNIe environment. The class `DSL_node` represents a node in the SMILE environment. The class `SmileAdapter` provides the interface between the two environments. In order to be able to manage PQ-model node in GeNIe, we extend the interface provided by `SmileAdapter`, so that GeNIe is able to recognize and manage PQ-model nodes.

8.2 Application Testing

We divide our testing trajectory in two phases: unit testing, and system testing. During unit testing, we test the functionality of the separate components of our system. During the system testing phase, we make sure that the complete system complies with the functional and nonfunctional requirements of our system, as state in Chapter 7. We review three system testing activities: unit testing, functional testing, and pilot testing.

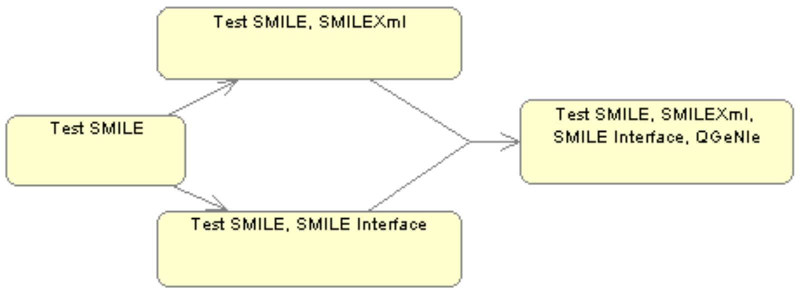

Test SMILE
Test SMILE, SMILEXml
Test SMILE, SMILE Interface
Test SMILE, SMILEXml, SMILE Interface, QGeNIe

Figure 8.5: Unit testing strategy

8.2.1 Unit Testing

During unit testing we tested the four subsystems, that we discussed in the previous section, in separate fashion. Subsystems commonly rely on each other, and consequently we run into the problem that we cannot start testing one subsystem, without first testing another.

To solve this problem, we have devised a testing strategy, shown in Figure 8.5. The strategy shows in which order the subsystems will be tested. For larger projects, a good strategy may save a lot of time, as many testing activities my be executed in parallel.

Approach To test the separate components we implemented a special dummy class. This dummy class behaves as one of the subsystems, but it does not contain any real functionality. For example, to test the SMILE subsystem, the dummy object behaves as the SMILEInterface subsystem, i.e., it calls on all the operations in the SMILE that the SMILEInterface subsystem could call upon. Note that the final step in our testing strategy concerns the whole system, i.e., belongs to the system testing phase.

Criteria None of the operations may result in a crash, either during compilation or during run-time. All of the operations that contain functionality relating to the theoretical PQ-model, must be correct. To ensure this, we worked out an arbitrary PQ-model in the SMILE subsystem and separately on paper. Obviously, there may not be a difference in result.

Incidents We discovered one major error in the `updateTable()` operation. Due to this error the SMILE subsystem and the model on paper gave different results. The algorithm was rewritten and the failure was corrected.

8.2.2 Functional Testing

The goal of functional testing is to find the difference between the functional requirements and the behavior of the system as a whole.

Approach The test cases were derived from the use cases (Figure 7.1.2):

- Create a node
- Create an arc
- Delete a node
- Delete an arc
- Set a prior belief parameter
- Set an influence parameter
- Load a network
- Save a network

Each of the test cases was executed multiple times in the QGeNIe environment.

Criteria The system may not crash during runtime. In addition, the behavior of the model in the QGeNIe environment must translate to the PQ-model, proposed in Chapter 5.

Incidents One failure became apparent during testing: two of the four influence types were switched. A required event acted as a barring event and vice versa. This failure was the result of an interface error between SMILEInterface and QGeNIe. The error was corrected.

8.2.3 Pilot Testing

During a pilot test, the system is installed and used by a selected set of users. If the system is tested in the development environment, we refer to it as 'alpha' testing. If the system is tested in the target environment, we refer to it as 'beta' testing. We performed alpha tests with four different users. We review their relevant characteristics.

- Student industrial design, high computer skills, no coding skills
- Student multimedia design, high computer skills, low coding skills

- Student art history, medium computer skills, low coding skills
- Student mechanical engineering, high computer skills, low coding skills

None of the users suffered from any visual impairments. All users were between age 22 and 27.

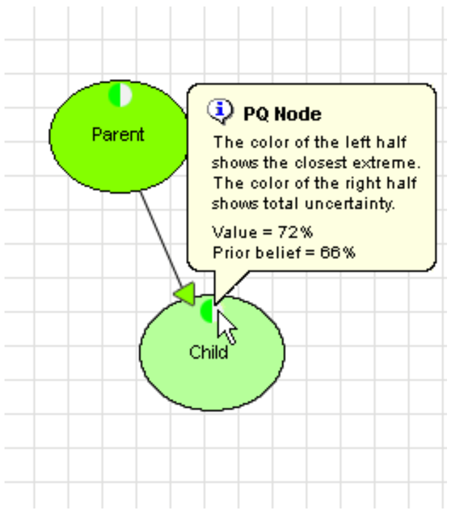
Approach All users were allowed to test the application for 20 minutes. The users were given instructions to build a simple network (each use case was performed at least once), and where asked questions about the visual representation of information.

Criteria Most of the graphical information elements, e.g., color of the nodes, shape of an arc-head, are designed to have an intuitive quality. If the function of such an element is not clear after the 20 minutes, we will treat it as an error. Likewise, if a particular use case ultimately resulted in the confusion of the user, we will treat it as an error.

Incidents None of the use cases resulted in errors. In fact, none of the users needed any detailed step-by-step instructions. The function of the node color was apparent to all four users. The function of the arc-head shape and color was immediately apparent to two of the users. The other two users assured us that, given more time, they would have discovered its functions as well. There were three specific errors:

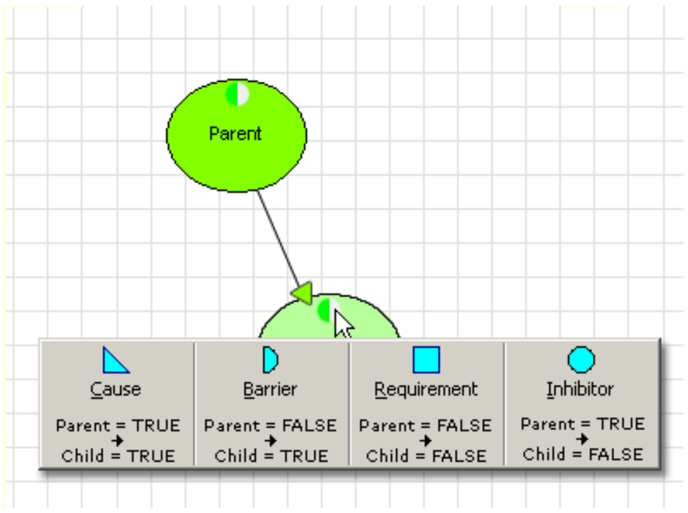
- None of the users understood the function of the anchor circle, in the middle of a node, without detailed clues.
- For all parameters, the default value is set to 0. Three of the users remarked that it would be better to have default value 0.5. This way, the interaction within a model becomes apparent, without requiring the user to set the parameters first.
- The difference between the influence types barrier and inhibitor was difficult to grasp for all four users. Surprisingly, an explanation in natural language was more difficult to grasp than a schematic of the underlying boolean logic, even though none of the users had any background in logic theory. All four users understood the schematic.

To correct these three errors, the GUI design was adjusted, as shown in Figure 8.6 and 8.7. These changes will be implemented by the DSL members responsible for GeNIe and QGeNIe in the near future.



PQ Node
The color of the left half shows the closest extreme. The color of the right half shows total uncertainty.
Value = 72%
Prior belief = 66%

Figure 8.6: QGeNIe: numerical representation of the node value.



Cause: Parent = TRUE → Child = TRUE
Barrier: Parent = FALSE → Child = TRUE
Requirement: Parent = FALSE → Child = FALSE
Inhibitor: Parent = TRUE → Child = FALSE

Figure 8.7: QGeNIe: schematic explanation of influence types.

9 An Example Network: Corporate Turnover

In Chapter 5 we have seen that the PQ-model is theoretically sound. In Chapter 6 we have seen that the PQ-model has some practical significance. All that is lacking now is a practical model that may serve as a vehicle to introduce the proposed model. In this chapter, we discuss a practical qualitative problem, to illustrate the sort of problem to which we can apply the PQ-model.

It is important to realize that our goal is not to find 'the optimal solution.' Rather, we are looking for a model that will give us insight concerning the interaction between variables, i.e., trying difference scenarios not only to confirm obvious consequences, but also discovering non-obvious implications. Rather then learning the parameters from data, we envision the models that are built on the PQ-model to serve as tools that facilitate discussion between human decision makers. Please note that the PQ-model is certainly not restricted to this type of application, and might well be applied to more quantitative problems. However, given the scope of this thesis, we will not propose such a model.

The model discussed in this chapter is the product of a term project associated with a graduate course *Decision Analysis and Decision Support Systems* in the Department of Information Science, University of Pittsburgh. We explicitly inform the reader that the purpose of this model is not to give a detailed account of some practical problem and the resulting solution. Rather we attempt to show the reader a possible application to understand the type of problems that the PQ-model is apt to handle.

9.1 Problem definition

Employee turnover rates are a constant concern for businesses. Losing experienced employees and having to replace them with inexperienced 'rookies' can be very costly. Furthermore, turnover rates are not only associated with monetary costs, but company morale and company knowledge base are also affected. Gaining insight into this problem might prove a substantial benefit for companies.

This problem can be approached from two sides: employee or employer. Both are essentially different problems, involving different sets of options and different sets of influential elements. We shall approach this from the employer point of view. Thus we envision the decision maker to be a group of persons responsible for human resources within the company.

Notice that our problem domain is non-deterministic. For example, increasing salaries will incite some employees to stay, while others are more sensitive to job satisfaction. Some employees are important company assets, while others are less valuable. Because we are dealing with human behavior, our problem will be inherently susceptible to uncertainty. Thus the question that we want the model to answer is: *what causes a company to have high employee turnover and how do the relevant aspects of the problem interact?*

9.2 Identifying variables

The first step toward building a model is to identify the variables that will be incorporated. Having to few variables will render the model trivial, while having to many will make it too complex. To find a balance between both extremes, we simply set the maximal number of variables in advance. Suppose our model will support no more than 20 variables. After a short brainstorm session we come up with the list of variables shown in the left column of Table 9.2.

Once we have the variables, we have to translate them to statements that can be either 'true' or 'false.' To avoid ambiguity, it is usually best to make the statements in one of the two extremes. For example, suppose that, for the variable *responsibility*, we have statement R: *Work is associated*

Table 9.1: A list of variables for the turnover problem.

Issue	Statement
Age	Physical abilities reduced due to age
Perks	Job-related perks are satisfactory
Environment	Satisfactory facilities are available
Education	Average education is high
Equipment	Satisfactory equipment to perform job
Job market	Promising job market
Cost of living	Cost of living is affordable
Commuting	Average commuting cost is acceptable
Salary	Employees receive a satisfactory salary
Benefits	Employees are entitled to satisfactory benefits
Motivation	Employees are motivated to do their job
Schedule	Work schedules are flexible
Supervisor	There is pleasant interaction between employees and supervisors
Coworkers	There is pleasant interaction between coworkers
Recognition	Employees are recognized for their work
Responsibility	Work is associated with a high level of responsibility

with a satisfactory level of responsibility. The problem becomes apparent when R is 'false.' It is unclear whether the responsibility is too high or too low, and both may have different consequences. Therefore we prefer a statement that takes an extreme, e.g., R: *Work is associated with a high level of responsibility.* Note that for some variables this ambiguity does not exist, e.g., a salary can be too low, but is rarely (if ever) experienced as being too high.

9.3 Creating a structure

To create a structure, we categorize all variables into one of four categories: *employee characteristics, job characteristics, work environment, and external influences.* These categories are then transformed into statements and added to the model. These variables aggregate the influence that belong to the corresponding category. Doing so will make it easier to find the general area that may be subject to problems.

Table 9.2: Interaction types.

Type	Interaction
Promoting event	Parent TRUE → Child TRUE
Barring event	Parent FALSE → Child TRUE
Inhibiting event	Parent TRUE → Child FALSE
Required event	Parent FALSE → Child FALSE

In Chapter 5 we discussed four types of interaction between variables. Table 9.3 shows those four types and the way they interact in the deterministic case. Determining which nodes are connected and which type of arc best describes the interaction can be hard, especially if we are building a

Table 9.3: Parameters resulting from elicitation analysis.

Variable	**Points**	**Influence parameter**
Job characteristics	85	0.81
External influence	64	0.63
Employee characteristics	48	0.50
Environment characteristics	43	0.46
Salary	68	0.90
Recognition	44	0.63
Job market	43	0.61
Responsibility	33	0.49
Motivation	31	0.47
Supervisor	30	0.46
Benefits	30	0.46
Perks	11	0.23

noisy (non-deterministic) model. This knowledge is assumed to be part of the experts experience that builds the model. However, some arcs can be derived from common sense, e.g, employees can only be tempted by other jobs if there are tempting jobs available. In our model, this means that the statement *Promising Job Market* describes a required event if the statement *employees are tempted by other job offers* is to be true.

9.4 Eliciting the parameters

Due to the amechanistic property, we can elicit the parameters by asking simple questions. Remember how we found the parameters for the burning building example in Chapter 5. If we are dealing with a promoting or barring event E, we simply ask for the probability of the child being 'true,' given that all parents are in their distinguished state, except E.

In the case of the employee-turnover model, due to time restrictions and the scale of the project, we took an alternative approach. We handed out 24 elicitation forms to employees from three different companies. Each subject was asked to rank the variables in Table 9.2 according to order of importance. The for each subject, the top 5 variables were awarded points, i.e., 5 points for first place, 4 points for second place, etc. Furthermore, each subject was asked to rank the four main categories in order of importance. For each first place, 4 points were awarded, for second place 3 points, etc. To obtain the parameters, the total number of points for a variable is scaled down to values between 0.1 and 0.9. Values of 0 and 1 were avoided because they imply either totally independent or a deterministic interaction.

In order to confirm our prior assumptions about the interaction types between variables, the original elicitation form included several questions regarding the type of events. However, after three test subjects, we found the elicitation form to be too complex and these questions were omitted. Consequently the interaction types are all based upon assumption.

9.5 Results

Table 9.4 shows the parameters that resulted from analyzing the filled-out elicitation forms. We considered the four main categories, and only the eight most influential variables. Figure 9.1 shows the resulting network. Now that we have a model, let us see if it reveals insights that were hard to extract from the elicited forms directly. After trying different scenarios, i.e., different prior belief values for the variables, and observing the interaction, we identify three insights from the data:

- The salary is the most important factor when it comes to deciding whether or not to leave the company. The importance is such that it a unsatisfactory salary cannot be compensated for by other variables, regardless of their state.
- The state of the job market is a deciding factor. However, whether or not a promising job market results in people leaving their job for a 'better' one, almost solely depends on the level of education of the employees.
- Recognition and good relationships with coworkers only cause an atmosphere of good communication if employees have a good relationship with their supervisors.

We note that these insights are based on a relative few subjects and the network structure was assumed. Therefore we cannot assume they are of any practical significance. However, it is easy to the potential significance of the network.

9.6 Discussion

Admittedly, this is a rather crude method for building a practical model. However, it is often easier to see what is wrong with an existing model than to create a satisfactory model from scratch. The employee-turnover model discussed in this chapter can be used as such a prototype, that facilitates building a better model.

The purpose of this chapter was to show how the PQ-model can be applied in practice. We envision that such a model serves as a tool to facilitate the decision process, rather then simply spitting out the best decision. For example, imagine group of decision-makers trying to agree on a particular course of action. Multiple strategies are available, however, it is hard to see which

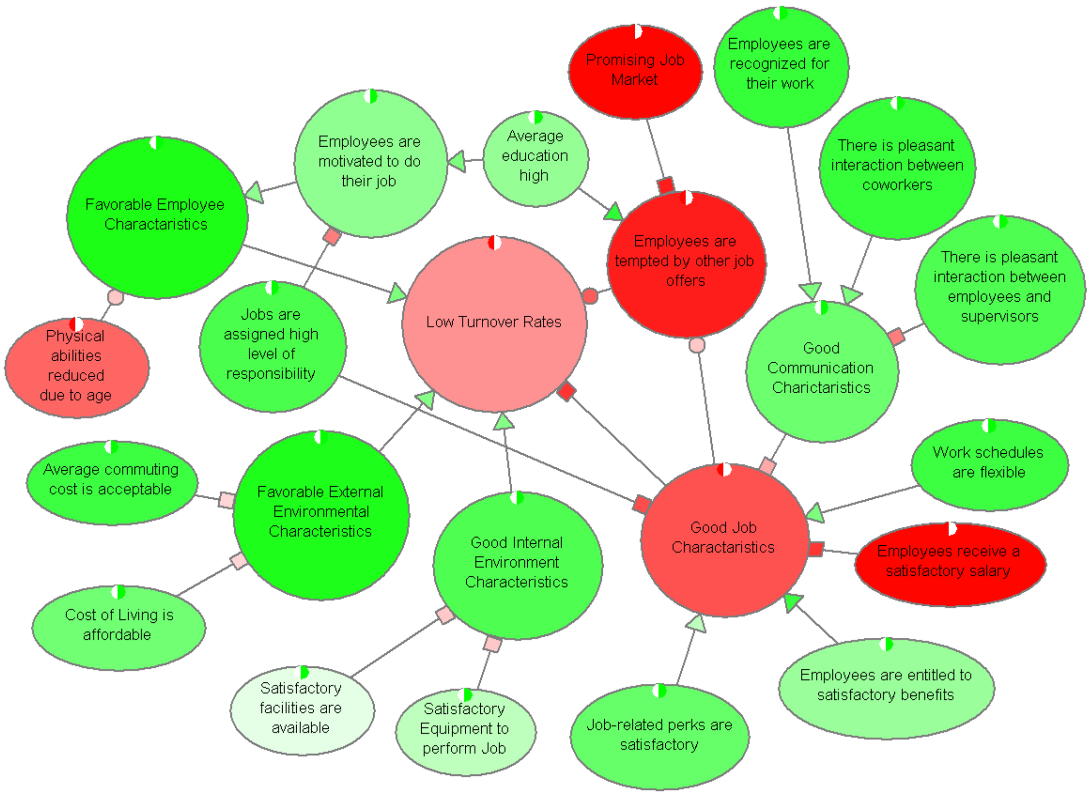

Promising Job Market
Employees are recognized for their work
Average education high
Employees are motivated to do their job
There is pleasant interaction between coworkers
Favorable Employee Charactaristics
Employees are tempted by other job offers
There is pleasant interaction between employees and supervisors
Low Turnover Rates
Jobs are assigned high level of responsibility
Physical abilities reduced due to age
Good Communication Charictaristics
Work schedules are flexible
Average commuting cost is acceptable
Favorable External Environmental Characteristics
Good Internal Environment Characteristics
Good Job Charactaristics
Employees receive a satisfactory salary
Cost of Living is affordable
Satisfactory facilities are available
Satisfactory Equipment to perform Job
Job-related perks are satisfactory
Employees are entitled to satisfactory benefits

Figure 9.1: An example network, modeling a company turnover problem.

is the preferred one. A qualitative model provides the decision-makers with a tool to benchmark strategies and attain insight into the consequences of each separate strategy.

10 Conclusions and Recommendations

In this final chapter we summarize the research described in this thesis. In Section 10.3 we give a brief overview of the discussed research. In Section 10.1 we review our initial goals and objectives and discuss to what extend they were satisfied. Finally, in Section 10.4 we discuss possible ventures for future research.

10.1 Goals and objectives

To determine to what extent we satisfied our initial objectives, as stated in Section 1.3, we discuss each of the four objectives separately.

10.1.1 Defining a theoretical model

A fundamental problem of Bayesian networks is the exponential growth of a CPT by the number of parent variables. Our goal was to introduce a model that solves this problem, while retaining the ability to combine opposing influences. Such a model would be applicable in particular to qualitative problems.

When we set off on finding such a model, we set out three guidelines to guarantee the applicability and practical significance of the model. We restate those guidelines:

- The proposed model is consistent with propositional logic and probability theory.
- The proposed model belongs to the class of amechanistic ICI models.
- The proposed model able to combine both positive and negative influences.

We have taken great care to obey the first guideline. The introduction of the PQ-model (Chapter 5) starts out by describing how we combined the deterministic logical propositions in order to achieve the logical 'skeleton' of the theoretical model. As a direct consequence, the proposed model is probabilistically sound. The greatest advantage of probabilistic soundness is that we avoid combining positive and negative influences by some arbitrary function. We avoid such a function because it would be essentially a heuristic approach. Heuristics are subjective, probability theory is not.

We ensured the amechanistic property of the proposed model by always assuming one of the two states to be the *distinguished* state, and by ensuring that, when all parent variables are in their distinguished states, the child variable is certain to be in the distinguished state. The advantage of the amechanistic property is that we can find the parameters for our model by asking simple and clear questions.

Regarding the third guideline, we have first shown how we could model positive and negative influences separately, using propositional logic. We then combined both by the grace of the laws of DeMorgan. Thus we satisfy our third guideline.

When compared to its only counterpart in current literature, i.e, CAusal STrenght (CAST) logic [2], we observe that the PQ-model offers the significant advantages of probabilistic soundness and a meaningful parameterization. Therefore we claim to have satisfied our first objective.

10.1.2 Validating the proposed model

To ensure the practical significance of the *theoretical* PQ-model, we conducted an experiment at the Department of Information Science, University of Pittsburgh. A total of 24 students partook in the experiment. The goal of the experiment was to show if there was a significant difference between the performance of an elicited CPT and an elicited PQ-model, under the same circumstances.

Although, the results suggested a difference in favor of the PQ-model, it was not statistically significant. However, under the assumption that (at the very least) the PQ-model performs just as well as an elicited CPT, the PQ-model still offers a significant advantage: the number of parameters, i.e., questions needed for the PQ-model is only linear in the number of parent variables, while the number of CPT entries is always exponential in the number of parent variables. Therefore we claim to have satisfied our second objective.

10.1.3 Incorporating the proposed into SMILE and GeNIe

SMILE (Structure Modeling, Inference, and Learning Engine) and GeNIe (Graphical Network Interface) are applications that enable their user to build Bayesian networks and influence nets. Both are complex applications, that have endured many modifications and extensions over the past years. To make matters more difficult, both SMILE and GeNIe lack proper documentation.

SMILE was extended to incorporate the PQ-model. Because the theoretic PQ-model applies in particular to qualitative problems, a separate version of GeNIe was implemented, named QGeNIe (Qualitative GeNIe). Both the extended version of SMILE and QGeNIe, were tested for implementation faults and will be soon available unline on the DSL website (http://genie.sis.pitt.edu/). We claim to have satisfied our third objective.

10.1.4 Proof of concept

The purpose the employee-turnover network in Chapter 9 is to show how the PQ-model can be applied in practice. We envision that such a model serves as a tool to facilitate the decision process, i.e., the decision makers can try out different scenarios by adjusting the prior belief parameters of the model variables.

To build the employee-turnover network, 24 employees from 3 different companies filled out a questionnaire. After analysis of the questionnaires, the we determined the parameterizations for the model. By applying different scenarios, we attained several non-obvious insights, which we attribute to the model. This demonstrates how a model built on PQ-nodes can be useful in practice. Thus we claim to have satisfied our fourth and final objective.

10.2 Evaluation

We claim to have satisfied the goals that we set out for ourselves. Yet, we have not answered the question whether or not this the work described in this thesis constitutes a substantial scientific contribution. In order for us to find the answer to that question, let is take a step back. The problem we chose to solve, is not a new problem. Qualitative reasoning is poplar topic, especially in physics, and many solution have been proposed. Causal interaction models are nothing new. Neither is the amechnistic property, nor boolean reasoning in Bayesian networks.

So we ask: what merit does our proposed model have over other? The answer is not surprising. In fact, as a general rule, any new scientific contribution is really only the repackaging or combination of known theories and applications. And by analogy, our proposal is the first amechanistic ICI model that is applicable to uncertain qualitative problems. So then, granted that our contribution is 'new,' the question remains if our contribution is of any significance.

Our short answer is 'yes.' It is exactly the combination of boolean logic, probability theory, ICI, and the amechanistic property that gives the PQ-model its edge over other solutions. The model consists of a mix of intuitive appeal and mathematical correctness. This is, we believe, what makes our proposed model stand out over other models, such as CAST. Our experiment showed that the PQ-model can more accurately model the way in which a human expert thinks than a conventional CPT. This shows that we are on the right track. By no means, do we claim that the PQ-model is the perfect solution to any conceivable problem. We leave this up the to the knowledge engineers. However, we do claim that there are problems for which the PQ-model is a better 'fit' than any other model. This is what we have shown. And this is where we claim the scientific contribution of this thesis.

Looking beyond the theoretical PQ-model, can we be satisfied with the software application that captured its functionality? A thorough software engineering process is important not only for the initial implementation of a system, but also, to certain extent, determines how easily we can extend the system.

SMILE and QGeNIe are both complex systems. However, remarkably, there is very little documentation available regarding the design, implementation and testing of either system, e.g., all class diagrams in this chapter were created by the author. Regardless, our main strive was to maintain consistency during the implementation process of the extended functionality, in order to mediate future work on SMILE and GeNIe.

SMILE and QGeNIe are not only complex, but also ever changing, i.e., most of the research conducted in the DSL is somehow incorporated into the software. Users from all over the world are requesting new features almost on a monthly basis. For SMILE and QGeNIe, the engineering process is a cycle, rather than a trajectory with a fixed beginning and end. In comparison, the work described in this chapter is only a drop in the ocean, and new requirements, implementation and testing should be expected.

The design of a satisfactory GUI is a project in itself. We have performed only the bare minimum of activities regarding the GUI design. However, the first users, i.e., the logistic branch of the US Airforce, have been introduced to QGeNIe. Soon new feedback will arrive, and we expect that most of that feedback will relate to the GUI design. Perhaps new functionality will be requested. Some of those requests may relate to possible future endeavors, such as proposed in Section 10.4. Some may be completely new. Regardless, the software application that resulted from our research is not a work-in-progress. It is a fully functional and usable piece of software, and satisfies no less than required within the scope of this thesis.

10.3 Research overview

This section gives a general overview of the research discussed in this thesis, and the the authors part in particular. All research and work described in this thesis is inspired under the advisement of Prof.Dr. Marek J. Druzdzel, Ph.D.

Chapter 5 The theoretical PQ-model is attributed solely to the author of this thesis.

Chapter 6 The design of the experiment, conduct of the experiment at the University of Pittsburgh and analysis of the experiment results is attributed solely to the author.

Chapter 8 Regarding the extended functionality of the SMILE library:

- system design is attributed solely to the author.
- system implementation is attributed solely to the author.
- organization of system testing is attributed solely to the author.

Regarding the extended graphical user interface (GUI) for SMILE, i.e., QGeNIe:

- GUI design is attributed to the author, supported by the Decision Systems Laboratory (DSL), University of Pittsburgh, in the form of brainstorm sessions.
- GUI implementation is attributed to the Decision Systems Laboratory.
- organization of GUI testing is attributed solely to the author.

Chapter 9 The employee-turnover model was the product of the joint effort of the author and two other MSc. students at the University of Pittsburgh. Model design and elicitation design is attributed to the author. The presented example model was part of the term-project associated with the course *Decision Analysis and Decision Support Systems* in the Department of Information Science, University of Pittsburgh.

10.4 Future research

We identify three possible ventures for future research. The first is extending the PQ-model to be able to model synergy between variables along the lines of the proposal by [15]. A synergy occurs when the influence of one parent variables somehow affects the influence of another parent variable. Notice, that this is exactly the opposite of the independence of causal influence (ICI) assumption [8], i.e., the recursive noisy-OR model. However, losing the ICI assumption also means losing the amechanistic property. Thus we would have to search for another way to ensure a clear and meaningful parameterization.

The second possible venue of research is extending the PQ-model with multi-state variables, along the same lines that the Noisy-OR extends to the Noisy-MAX. In this thesis we have only considered binary variables, i.e., variables that can be either 'true' or 'false.' Although multi-state variables would allow for building more comprehensive models, some obstacles have to be overcome. Consider a family of variables, i.e., a number of parents and a single child. It is not to difficult to extend the parent variables to multiple (n¿2) states. We can still assume one of the states to be the distinguished state, thus preserving the amechanistic property, and we simply assign different influence for each other state. How to extend the child variable so that it can have more than two states is less obvious, as each parent state should now have multiple influence (one for each non-distinguished child state) and the problem quickly grows in complexity. Zagorecki is currently working on extending the CAST model with multi-state variables [31].

The third and final possible venue for future research is extending the PQ-model so that it can handle contextual influence. We refer to contextual influence when the state of a variable X determines the type of influence of variable Z. For example, in roulette, the variable *I bet a million dollars on red* (true, false) could be either a good or a bad thing, depending on whether the variable *the outcome is red* is true or false. Thus, variable X is a type of switch. Such a switch could also be applied to determine the causal model for a family, e.g., if $X = \overline{x}$, then we use a noisy-OR, if if $X = x$, we use a noisy-AND on the other parent variables.

Bibliography

[1] Bernd Bruegge and Allen A. Dutoit. *Object-Oriented Software Engineering; Conquering Complex and Changing Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[2] K.C. Chang, P.E. Lehner, A.H. Levis, S. Abbas K. Zaidi, and X. Zhao. On causal influence logic. Technical report for subcontract no. 26-940079-80. Technical report, 1994.

[3] Gregory F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks (research note). *Artif. Intell.*, 42(2-3):393–405, 1990.

[4] Balaram Das. Generating conditional probabilities for bayesian networks: Easing the knowledge acquisition problem. *CoRR*, cs.AI/0411034, 2004.

[5] Francisco Javier Diez. Parameter adjustment in Bayes networks. The generalized noisyOR-gate. In *Proceedings of the Ninth Annual Conference on Uncertainty in Artificial Intelligence (UAI–93)*, 1993.

[6] Francisco Javier Diez and Marek J. Druzdzel. Canonical probabilistic models for knowledge engineering. Forthcoming 2004.

[7] Samuel Gerssen. Bayesian networks in credit rating, March 2004. Masters Thesis, Delft University of Technology.

[8] David Heckerman. Causal independence for knowledge acquisition and inference. In *Proceedings of the Ninth Annual Conference on Uncertainty in Artificial Intelligence (UAI–93)*, pages 122–127. Morgan Kaufmann Publishers, 1993.

[9] David Heckerman and John S. Breese. A new look at causal independence. In *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–94)*, pages 286–292, 1994.

[10] David Heckerman and John S. Breese. Causal independence for probability assessment and inference using Bayesian networks. In *IEEE, Systems, Man, and Cybernetics*, pages 826–831, 1996.

[11] Max Henrion. Some practical issues in constructing belief networks. In *Uncertainty in Artificial Intelligence 3*, pages 161–173. 1989.

[12] Ronald A. Howard and James E. Matheson. Influence diagrams. In *The Principles and Applications of Decision Analysis*, pages 719–762. Menlo Park, CA, 1981.

[13] J. Hulst. Research Assignment: Dynamic Bayesian networks and speech recognition. Knowledge Based Systems Group, Faculty of EEMCS, Delft University of Technology, July 2005.

[14] G. Kokolakis and P.H. Nanopoulos. Bayesian multivariate micro-aggregation under the Hellinger's distance criterion. *Research in Official Statistics*, 4(1):117–126, 2001.

[15] J.F Lemmer and Gossink. Recursive noisy-or: A rule for estimating complex probabilistic causal interactions. In *IEEE Transactions on Systems, Man and Cybernetics.*, pages 2252–2261, 2004.

[16] P. Lucas. Bayesian network modelling through qualitative patterns. AI Journal, 2005.

[17] Paul Maaskant. Bayesian networks applied to facial expression recognition. Knowledge Based Systems Group, Faculty of EEMCS, Delft University of Technology, August 2005.

[18] C. Meek and D. Heckerman. Structure and parameter learning for causal independence and causal interaction models. In *Proceedings of Thirteenth Conference on Uncertainty in Artificial Intelligence,* Providence, RI. Morgan Kaufmann, August 1997.

[19] A. Oni'sko, M. Druzdzel, and H. Wasyluk. A probabilistic causal model for diagnosis of liver disorders. In *Proceedings of Seventh International Symposium on Intelligent Information Systems (IIS–98)*, pages 379–387, Malbork, Poland, June 1998.

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

[21] Julie A. Rosen and Wayne L. Smith. Influence net modeling with causal strengths: an evolutionary approach. In *Command and Control Research and Techology Symposium*, 1996.

[22] Julie A. Rosen and Wayne L. Smith. Influending global situations, a collaborative approach. *Air & Space Power chronicles*, 1996.

[23] Julie A. Rosen and Wayne L. Smith. Focus: Distributed collaboration for crisis management, a white paper describing the system architecture, 1998.

[24] Julie A. Rosen and Wayne L. Smith. Influence net modeling for strategic planning: a structured approach to information operations. In *Phalanx*, volume 33. December 2000.

[25] Julie A. Rosen, Wayne L. Smith, Edward S. Smith, and Micheal A. Maldony. Planning with influence net modeling: an architecture for distributed, real-time collaboration. In *66th Military Operations Research Symposium*, June 1998.

[26] Edward Tufte. *Envisioning information.* Graphics Press, Cheshire, CT, USA, 1990.

[27] Edward R. Tufte. *The visual display of quantitative information.* Graphics Press, Cheshire, CT, USA, 1986.

[28] Edward R. Tufte. *Visual explanations: images and quantities, evidence and narrative.* Graphics Press, Cheshire, CT, USA, 1997.

[29] Haiqin Wang, Denver Dash, and Marek J. Druzdzel. A method for evaluating elicitation schemes for probabilities. In *Proceeding of the Fourteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS–2001)*, 2001.

[30] Changhe Yuan and Marek J. Druzdzel. An importance sampling algorithm based on evidence pre-propagation. In *Uncertainty in Artificial Intelligence: Proceedings of the Nineteenth Conference (UAI–2003)*, pages 624–631, San Francisco, CA, 2003. Morgan Kaufmann Publishers.

[31] Adam Zagorecki. The extended CAST model. Forthcoming, 2006.

[32] Adam Zagorecki. *Local probability distributions in Bayesian networks.* PhD thesis, School of Information Sciences, Department of Information Science and Telecommunications, University of Pittsburgh, 2006.

A

Appendix: PGM 2006 Paper Submission

This appendix contains a paper concerning the theoretic PQ-model, as submitted to the third European Workshop on Probabilistic Graphical Models, September 12-15, 2006, Prague, Czech Republic. Acceptance of the submission is pending. Notification of acceptance: June 28, 2006.

For reasons of clarity of attribution, this thesis refers to the proposed theoretical model as the Probabilistic Qualitative model. However, we alert the reader that, in this paper, the same model is referred to as the DeMorgan model.

A Model for Opposing Influences Under Assumption of Independence of Causal Influences

Paul P. Maaskant
Department of Media and
Knowledge Engineering
Delft University of Technology
2628 CD Delft, The Netherlands
maaskant@ch.tudelft.nl

Marek J. Druzdzel
Decision Systems Laboratory
School of Information Sciences
University of Pittsburgh
Pittsburg, PA 15260
marek@sis.pitt.edu

Abstract

To reduce the knowledge engineering effort in building Bayesian networks, local probability distributions can be expressed by parametric causal models, such as Noisy-OR or Noisy-MAX gates, based on the independence of causal influence (ICI) assumption. While very useful and widely applied, these models are limited in their expressive power. In particular, they cannot model combinations of opposing, positive and negative influences. This paper introduces a new model, the DeMorgan gate, that is able to address a broader subclass of problems, while retaining all desirable properties of ICI models, such as easy parameter elicitation from experts. We describe an empirical study that validates the proposed model.

1 Introduction

Bayesian networks (BNs) (Pearl, 1988) offer a sound framework for reasoning in uncertain problem domains. A BN consists of a qualitative part in the form of a graphical representation of the independencies among variables, and a quantitative part, in the form of Conditional Probability Tables (CPTs). A CPT specifies the relation between a variable and its immediate predecessors in the graph (it is common to refer to these as parents or, somewhat imprecise, causes, while to their direct descendant as a child or effect variable).

A fundamental problem of CPTs is their exponential growth in the number of parent variables. For nodes with over 10 parents, not uncommon in practical models, eliciting the CPT from human experts is practically infeasible. Learning a model from data may also be hard, as there are typically not enough cases to learn every distribution in a CPT reliably. Independence of Causal Influences (ICI) models (Heckerman and Breese, 1994), provide a solution by assuming that parent variables cause the effect independently of each other. The effect of this assumption is such that the number of required parameters is linear, rather than exponential, in the number of parent variables. Amechanistic ICI models (Heckerman and Breese, 1996) are a subclass of ICI models in which every variable has a distinguished state, such that if all parent variables are in their distinguished states, then the child variable is also in its distinguished state. For example, in a noisy-OR model, if all possible causes of engine malfunction are absent, then engine malfunction does not happen. This property allows us to obtain the model parameters from experts by asking simple questions. We believe that the unquestionable popularity of the Noisy-OR (Pearl, 1988; Henrion, 1989) and Noisy-MAX (Diez, 1993) models is in part due to their amechanistic property.

One of the main practical limitations of the Noisy-OR and Noisy-MAX models is that they cannot model opposing influences, i.e., combinations of influences that increase, and decrease the posterior probability of the child variable. Existing attempts to address this problem are, we believe, weak. In this paper, we propose an amechanistic ICI model that combines opposing influences, yet retains a clear parameterization. The amechanistic property ensures that the parameterization is transparent, probabilistic soundness ensures that it is mathematically correct, and propositional logic, that lies at its foundations, ensures that our model is meaningful and intuitive for humans.

We will use uppercase letters to denote random variables (e.g., X) and lowercase letters to denote their states (e.g., x). Because we deal only with binary variables we denote the range of a variable X as $Range(X) = \{x, \overline{x}\}$. Bold uppercase letters will denote sets of random variables (e.g., $\mathbf{X}$) and bold lowercase letters will denote values for a set of random variables (e.g., $\mathbf{x}$). We use $P(X)$ to denote the probability distribution over a variable X.

2 Related Work

We review briefly two models that are directly related to the DeMorgan model.

2.1 The Noisy-OR Model

The Noisy-OR (Pearl, 1988; Henrion, 1989) model is a non-deterministic extension of the deterministic OR relation. Its variables, e.g., X, are binary and can be either 'present,' denoted as x, or 'absent,' denoted as $\overline{x}$. Each 'present' parent event can independently produce the child effect. Noisy-OR's amechanistic property assumes that if none of the parent variables $X_1, ..., X_n$ are 'present,' then neither is the child variable Y:

$$P(\overline{y}|\overline{x}_1, ..., \overline{x}_n) = 1 \ . \tag{1}$$

We define the probability that x_i produces y as:

$$P(y|\overline{x}_1, ..., x_i, ..., \overline{x}_n) = z_i \ . \tag{2}$$

The ICI assumption allows us to derive the probability of $\mathbf{X}$ producing y as:

$$P(y|\mathbf{X}) = 1 - \prod_{X_i = x_i} (1 - z_i) \ . \tag{3}$$

Repeating this process for all possible parent configurations gives us the CPT. Henrion (1989) extended the Noisy-OR model by introducing a leak probability z_L, defined as:

$$P(y|\mathbf{X}) = 1 - (1 - z_L) \prod_{X_i = x_i} (1 - z_i) \ . \tag{4}$$

The leak variable z_L represents the probability that the child variable is in its 'present' state, even when all the parents are 'absent.' An intuitive interpretation of leak is that it represents unmodeled causes of Y, all of which are 'present.'

2.2 Causal Strength Logic

In CAusal STrength (CAST) logic (Chang et al., 1994), the variables represent statements that can be either 'true' or 'false.' The value of a variable expresses the measure of belief that the corresponding statement is true. The main advantage of the CAST model is that parent variables in the same family are allowed to have opposing, i.e., a mixture of positive and negative, influences. The parents can influence the child variable in both of their states.

Each variable (node) is assigned a baseline parameter $b \in [0, 1]$. Each arc is associated with two parameters $h \in [0, 1]$ and $g \in [0, 1]$. Suppose node X is the parent of node Y. Parameter g specifies the change in the belief of Y relative to b_Y, when X is 'true.' Parameter h specifies the change in the belief of Y relative to b_Y, when X is 'false.' Causal strength $c_{Y|X}$ is defined as g, if $X = true$, and as h, if $X = false$. A positive value of $c_{Y|X}$ indicates an increase in the belief of Y:

$$b_{Y|X} = b_Y + (1 - b_Y) \cdot c_{Y|X} \ . \tag{5}$$

A negative value of $c_{Y|X}$ indicates a decrease in Y.

$$b_{Y|X} = b_Y \cdot c_{Y|X} \ . \tag{6}$$

In case of multiple parents with values $\mathbf{X}$, we determine the causal strength $c_{Y|X_i}$ for each parent variable X_i, and aggregate positive (C_+) and negative (C_-) causal strengths separately. For all $c_{Y|X_i} \geq 0$ we have

$$C_+ = 1 - \prod_i \left(1 - c_{Y|X_i}\right) \ , \qquad (7)$$

and all $c_{Y|X_i} \leq 0$ are aggregated by:

$$C_- = 1 - \prod_i \left(1 - |c_{Y|X_i}|\right) \ . \qquad (8)$$

We combine the aggregated causal strengths C_+ and C_-. If $C_+ \geq C_-$, then overall influence O is defined as

$$O = 1 - \frac{1 - C_+}{1 - C_-}, \qquad (9)$$

and for $C_+ \leq C_-$, we have

$$|O| = 1 - \frac{1 - C_-}{1 - C_+}, \qquad (10)$$

Note that O is negative in Equation 10. Finally, we determine the conditional probability distribution over Y for a given parent configuration $\mathbf{X}$:

$$P(Y|\mathbf{X}) = \begin{cases} b_Y + (1 - b_Y)O & \text{for } O \geq 0 \\ b_Y(1 - |O|) & \text{for } O < 0 \end{cases} . \qquad (11)$$

Repeating this procedure for all possible parent configurations gives us a CPT. The main disadvantage of the CAST model is that is suffers from unclear parameterization. This applies in particular to the baseline parameter. In our experience, experts have trouble understanding its meaning. It is neither the prior probability, nor it is the leak probability.

2.3 The Amechanistic Property

Amechanistic causal independence models (Heckerman and Breese, 1996) are a subclass of ICI models that make two additional assumptions: (1) each variable has a 'typical' state, referred to as the *distinguished* state. This is usually the 'default' state for that variable. (2) If all parent variables are in their distinguished states, then so is the child variable. Assumption (2) has one important consequence. It enables us to directly elicit the causal influence of a parent variable X_i by asking: *"What is the probability that Y is in its non-distinguished state, given that all parents are in their distinguished states, except parent X_i."* The parameters for an amechanistic ICI model can, therefore, be obtained by asking simple and clear questions, and so, such a model is particularly suited for parameterization by human experts.

The Noisy-OR has the amechanistic property, as we define the distinguished state to be 'absent,' and assumption (2) is captured by Equation 1. Equation 2 captures the question needed for parameter elicitation. It is, in fact, not required that the distinguished state is 'absent,' it might just as well be 'present,' although this is uncommon in practice.

The CAST model does not have the amechanistic property. Suppose we ask an expert: *"If all parents are in their distinguished states, save X_i, what is the probability of Y occuring?"* The answer will not give us the parameter for parent X_i, because neither presence nor absence of the parent node is a distinguished state, and parent variables can have causal influence in both states. Therefore, any of the parents may have caused the child event.

3 The DeMorgan Model

The model that we propose in this paper follows three guidelines: (1) in its deterministic form, the model is consistent with propositional logic, (2) it belongs to the class of amechanistic ICI models, and (3) it is able to handle a broad class of problems, and, in particular, is able to handle simultaneous opposing influences.

3.1 Promoting Influences

One single effect can often be produced by different causes. A fire can be caused by either arson, a neglected burning stove, or a short circuit, but all will produce the same result. Consider the case of three independent causes X_1, X_2, and X_3, and a single effect Y. Assuming that our model is deterministic, the presence of any of the parent variables is sufficient to cause the child's presence (Henrion, 1989). So, we have:

$$Y \equiv X_1 \vee X_2 \vee X_3 \ . \qquad (12)$$

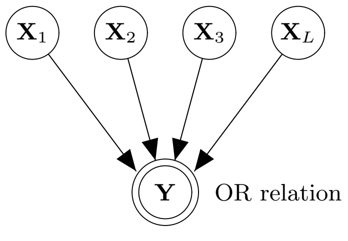


Figure 1: Three encouraging causes and a single effect.

Equation 12 assumes that Y cannot be caused by any other influence than X_1, X_2 and X_3. This assumption rarely holds in real life. However, this problem can easily be overcome by introducing a new cause X_L, which represents the aggregated influence of all un-modeled causes. Thus we have:

$$Y \equiv X_1 \lor ... \lor X_n \lor X_L \ . \qquad (13)$$

We assume the distinguished state of the child variable to be 'absent.' This assumption makes sense from the point of view that we cannot *cause* something that is already 'present.' If we translate Equation 13 to a graphical representation, we obtain Figure 1. A node drawn with a double circle represents a deterministic interaction, i.e., the values of its parents are sufficient to imply the value of the child node.

3.2 Inhibiting Influences

Suppose now that our model for starting a fire includes two possible causes, 'arson' and 'short circuit,' but also the preventive variable 'sprinkler device.' The sprinkler device acts as the opposite of the other two: a parent that has an *inhibitive* effect on the child variable, i.e., causing it to be 'absent.' Let us consider the case in which we have only inhibitive influences and a single child variable. We have inhibitive causes U_1, U_2 and U_3 and the child variable Y. The presence of any of the parents is sufficient to cancel the child effect. We use the following logical proposition:

$$\overline{Y} \equiv U_1 \lor U_2 \lor U_3 \ . \qquad (14)$$

Equation 14 is similar to Equation 12 but now the parents cancel the child event instead of producing it. Variable Y is 'absent' if at least a single parent is 'present.' Again we assume that the distinguished state of the parent events is 'absent,' however, contrary to our previous example, we assume the distinguished state of child Y to be 'present.' This assumption is based on common sense: we cannot cancel an event that is not present.

3.3 Combining Influences

Although we are able to model both promoting and inhibitive influences separately, it is less obvious how to combine both into a single model. To do so, we must find the logical proposition that defines the interaction between parent and child variables. We achieve this by applying one of DeMorgan's laws on Equation 14. We get:

$$Y \equiv \overline{U_1} \land \overline{U_2} \land \overline{U_3} \ . \qquad (15)$$

Now we can deduce the proposition that we will use to combine opposite influences. In order to be consistent, the combined logical proposition can only be true for a particular parent configuration if, both Equation 13 and 15, are also true for that same configuration. This implies that both equations form a conjunction:

$$Y \equiv (X_1 \lor ... \lor X_n \lor X_L) \land \overline{U_1} \land ... \land \overline{U_m} \ . \quad (16)$$

We now have the logical proposition that we need to in order to define the interaction for the combined model with parents $X_1, ..., X_n$, $U_1, ..., U_m$, and a child variable Y. Notice that the proposition on the right hand-side of Equation 16 is in the Conjunctive Normal Form (CNF). Because each of the conjuncts, save one, consists of a single variable, we can build a very simple model to represent this proposition, using only a single deterministic-OR and a single deterministic-AND. Since each logical proposition can be expressed as a CNF, DeMorgan-gates can model any logical proposition.

As pointed out by Lucas Lucas (2005), there are 16 Boolean functions and each of them or even every combination of them can give raise to a canonical model. DeMorgan gate is capable of representing any Boolean function by its conjunctive form. Our proposal comes with a clear semantics and an intuitive method to elicit parameters from an expert. These, we believe, have contributed to the enormous popularity of such interactions as the Noisy-OR gate.

3.4 Modeling Uncertainty

We have only discussed the deterministic version of the DeMorgan gate. Now, we introduce noise separately for promoting and inhibiting influences, and then we combine them.

3.4.1 Noise for Promoting Influence

We have determined the logical proposition for promoting influences in Equation 12. We assume that the distinguished state of a parent is the 'absent' state, i.e., when the parent is absent, it is *certain not to cause the child event* but, when present, it will *produce the child event with some degree of uncertainty.*

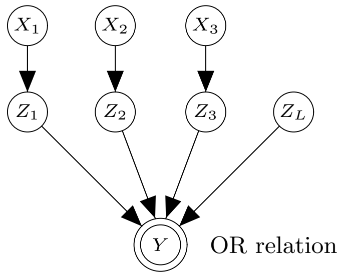


Figure 2: Modeling uncertain promoting causes.

When a parent X_i is in its non-distinguished state, we assign a probability v_i that it will produce the child event. We include this uncertainty in the network by adding a noise variable Z_i that has the following behavior:

$$P(z_i|X_i) = \begin{cases} v_i & \text{if } X_i = x_i \\ 0 & \text{if } X_i = \overline{x}_i \end{cases} , \tag{17}$$

and the child variable Y interacts with the variables $Z_1, ..., Z_n$ as a deterministic-OR gate. The conditional probability of the child variable being present is given by:

$$P(y|\mathbf{X}) = 1 - \prod_{x_i \in +\mathbf{x}} (1 - v_i) . \tag{18}$$

We define $+\mathbf{x}$ to be the subset of $\mathbf{X}$ that contains all parents that are in their non-distinguished states. For each parent X_i, we can elicit the parameter v_i by asking the expert: *"What is the probability that Y occurs when all parents are in their distinguished states, except parent X_i?"* Finally, we incorporate the leak parameter v_L, expanding Equation 18 into:

$$P(y|\mathbf{X}) = 1 - (1 - v_L) \prod_{x_i \in +\mathbf{x}} (1 - v_i) , \tag{19}$$

which is, in fact, the leaky Noisy-OR model (Henrion, 1989), as discussed earlier.

3.5 Noise for Inhibiting Influence

We assume that the distinguished state of a parent that models inhibiting influence is the 'absent' state, i.e., when the parent is absent it is *certain not to inhibit the child event*, but when present, *it will inhibit the child event with some degree of uncertainty.* When a parent U_i is in its non-distinguished state, we assign a probability d_i that it will inhibit the child event. We include this uncertainty in the network by adding a noise variable W_i that has the following behavior:

$$P(w_j|X_j) = \begin{cases} 0 & \text{if } U_j = u_j \\ d_j & \text{if } U_j = \overline{u}_i \end{cases} , \tag{20}$$

and the child variable Y is equivalent to the conjunction of variables $\overline{W}_1, ..., \overline{W}_n$, as given by Equation 15. We have shown by DeMorgan's laws that the Equation 14 is the logical equivalent of Equation 15. Equation 19 gives us the probability of Y occurring:

$$P(y|\mathbf{U}) = \begin{cases} \prod_{u_j \in +\mathbf{u}} (1 - d_j) & \text{if } +\mathbf{u} \neq \varnothing \\ 1 & \text{if } +\mathbf{u} = \varnothing \end{cases} . \tag{21}$$

We define $+\mathbf{u}$ to be the subset of $\mathbf{U}$ that contains all parents that are in their non-distinguished states.

It makes little sense to ask for the effect of rain on a bonfire, if there is no fire in the first place. By analogy, we cannot determine d_j directly if we assume that the distinguished state of Y is 'absent.' Therefore, we determine d_j relative to an arbitrary set of promoting influences. Suppose we know the effect of a promoting influence X_i, denoted as p_i, and the effect of both X_i and inhibiting influence U_j, denoted as q_j:

$$p = 1 - (1 - v_L)(1 - v_i) \ , \tag{22}$$
$$q_j = (1 - (1 - v_L)(1 - v_i))(1 - d_j) \ . \tag{23}$$

We can then determine d_j by Equation 24

$$d_i = 1 - \frac{1-p}{1-q_i} \tag{24}$$

We may also elicit d_j relative to only the leak parameter, although it seems that this elicitation will be reliable for large values of v_L, such as $v_L \approx 1$.

Assessment of the parameters for inhibiting causes is overconstrained in the sense that there are several ways of asking for these. For example, one might pick any of the promoting causes along with the inhibiting cause in question. One might pick a combination of promoting causes or, when the leak is significantly larger than zero, even no promoting cause at all. Readers with some experience in probability elicitation will not be surprised to see that each of these choices may yield a different estimate. But so may different fundamental methods for probability elicitation, such as symmetric bets or reference lottery. Even within one method, elicitation may depend on the choice of questions asked. Elicitation of the parameters of inhibiting causes is in this sense not more overconstrained than probability elicitation in general.

3.6 The Leaky Noisy-DeMorgan Gate

Now that we have found a way to combine simultaneous noisy promoting, and inhibiting causes, we define the leaky noisy-DeMorgan gate. We derive from Equation 16 the following function:

$$P(y|\mathbf{X},\mathbf{U}) = 1-(1-v_L)\prod_{x_i\in+\mathbf{x}}(1-v_i))\prod_{u_j\in+\mathbf{u}}(1-d_j) \tag{25}$$

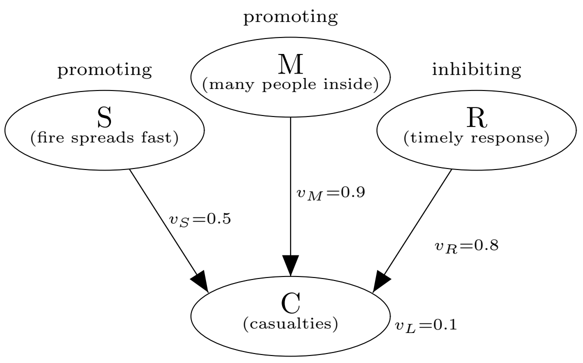


Figure 3: DeMorgan model example network

Figure 3 provides an example of a simple problem, modeled by a DeMorgan gate. Imagine that there is a building on fire. The child variable represents our belief that there will be casualties due to the fire. We identify three influences that act as parent variables: (S) the rate at which the fire spreads, (M) the number of people inside and (R) the speed at which the fire department intervenes.

If the fire spreads fast, the risk of casualties increases. We define this variables as a promoting parent with influence $v_S = 0.50$. Also, if there are many people in the building, we expect a considerable increase of the risk of casualties. We define this variable as a promoting parent with influence $v_M = 0.90$. In the event that the fire department is able to respond quickly, the risk of casualties will diminish. We define this variable as an inhibiting parent with influence $d_R = 0.80$. In the event that the fire does not spread, there are not many people in the building, and there is no timely response, we estimate the probability of casualties to be $V_L = 0.1$. By Equation 25 we now derive the resulting CPT:

M	m				$\overline{m}$			
S	s		$\overline{s}$		s		$\overline{s}$	
R	r	$\overline{r}$	r	$\overline{r}$	r	$\overline{r}$	r	$\overline{r}$
P(C)	0.19	0.95	0.18	0.91	0.11	0.55	0.02	0.10

Table 1: Example Conditional Probability Table

4 Elicitation In Practice

We conducted an experiment similar to the one described by Zagorecki and Druzdzel (2004), following the methodology introduced by Wang et al. (2001), assuming a DeMorgan-gate as the underlying model. Subjects were students in a graduate course *Decision Analysis and Decision Support Systems* in the School of Information Sciences, University of Pittsburgh. Because only 24 subjects participated in the experiment, we chose the within-subject design.

4.1 Experiment Design

Subjects played a simple, fictional game, modeled by a DeMorgan model, consisting of four inputs [true, false] and one binary output [success, failure]. Two of the inputs were promoting influences, with strengths randomized (per subject) between [0.5, 0.9]. The other inputs were inhibiting influences, with strenghts randomized between [0.3, 0.9]. The leak probability ranged between [0.1, 0.3]. Subjects were unaware of the underlying model.

Subjects were allowed 160 trials, each trial consisting of three phases: configuring the input, starting the game, and observing the result. The probability of a successful outcome was determined by the input configuration and the underlying DeMorgan model. Afterwards, the subjects were asked (1) to give the full CPT, and (2) give the parameters for a DeMorgan model. To compensate for a possible carry-over effect, the order of (1) and (2) was randomized.

Because we want to measure how well the DeMorgan model captures an experts knowledge, rather than the modeled phenomena itself, we used the *observed* CPT. For each value of the observed CPT, given the 160 observations of the participant, we calculated the maximum a posteriori estimate using a Beta prior distribution with a small equivalent sample size:

$$OBS_j = \frac{s_j + 0.01}{t_j + 0.02} \, , \qquad (26)$$

where s_j denotes the number of successful trials, and t_j the total number of trials for input configuration j.

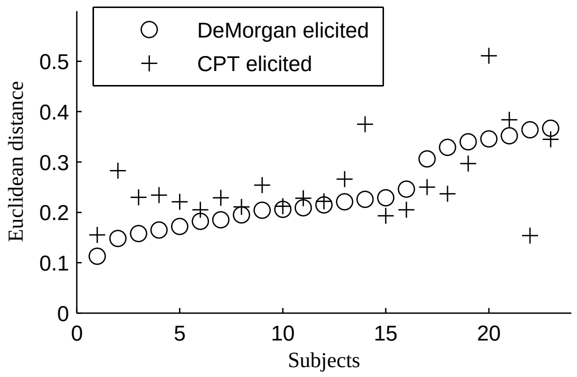


Figure 4: Raw data (sorted by increasing distance for the DeMorgan model)

Measure	Distance
Averaged Euclidean Distance DeMorgan	0.2382
Weighted Hellinger Distance DeMorgan	0.2481
Averaged Euclidean Distance CPT	0.2566
Weighted Hellinger Distance CPT	0.2563

Table 2: Averaged Euclidean and Hellinger distances.

4.2 Experiment Results

We measured the difference between the observed CPT and the elicited probability distributions by the averaged Euclidean and Hellinger (Kokolakis and Nanopoulos, 2001) distance. We express our results as the averaged Euclidean and Hellinger distances over all the distributions in the observed CPT.

The subjects took, on average, 40 minutes to complete the experiment. We judged one of the subjects to be an outlier, and excluded the subject from further analysis. This subject likely confused the concept of inhibiting with promoting, and vice versa, as very high probabilities were observed, yet very low probabilities were elicited. Figure 4 shows raw data, i.e., the Euclidean distance for each subject: (1) the distance between the observed CPT and the CPT generated by the elicited DeMorgan model, sorted from the smallest to the largest distance, and (2) the distance between the observed CPT and the directly elicited CPT. We would like to point out that the range of distances is lower for the DeMorgan gate. Table 2 shows the averaged Euclidean and Hellinger distances.

To determine statistical significance, we performed one-tailed paired t-tests on both distance measures. Comparison of the DeMorgan model versus the elicited CPT yielded $p \approx 0.14$ (Euclidean) and $p \approx 0.29$ (Hellinger). The results are not statistically significant, but they do suggest that the CPT generated by a DeMorgan model is at least just as accurate as an elicited CPT. This is a non-trivial advantage when the number of parent variables starts to grow. For a family with 10 parent variables, we have 21 questions for the DeMorgan model, versus 1024 questions for direct CPT elicitation.

5 Conclusions

The most important property of the DeMorgan model is that it is able to handle a mix of positive and negative influences, preserving both the amechanistic property and probabilistic soundness. By grace of the amechanistic property, all parameters for the DeMorgan model can be elicited by asking simple questions in the form $P(Y|\mathbf{X})$. Probabilistic soundness ensures mathematical correctness, and propositional logic, that lies at its foundations, ensures that our model is meaningful and intuitive for humans.

The results of our experiment showed no statistically significant difference between a CPT generated by the DeMorgan model and a CPT elicited directly. Yet, for the DeMorgan model, the number of questions needed for elicitation is linear, rather than exponential, in the number of parent variables. A knowledge engineer is left with the task of judging whether an interaction between a node and its parents can be modeled by means of a DeMorgan model. The conditions that have to be fulfilled for the DeMorgan model are similar to those listed for other canonical gates by Diez and Druzdzel Diez and Druzdzel (2004): each parent must be able to either cause or to inhibit the child through a separate causal mechanism and there may be no significant interactions among these mechanisms.

A possible extension of the DeMorgan model incorporating synergies between variables, as proposed by Lemmer and Gossink (2004). We are currently working on extending the DeMorgan model to multi-valued variables.

References

K.C. Chang, P.E. Lehner, A.H. Levis, S. Abbas K. Zaidi, and X. Zhao. 1994. On causal influence logic. Technical report for subcontract no. 26-940079-80. Technical report.

Francisco Javier Diez and Marek J. Druzdzel. 2004. Canonical probabilistic models for knowledge engineering. Forthcoming 2004.

Francisco Javier Diez. 1993. Parameter adjustment in Bayes networks. The generalized noisyOR-gate. In *Proceedings of the Ninth Annual Conference on Uncertainty in Artificial Intelligence (UAI–93)*.

David Heckerman and John S. Breese. 1994. A new look at causal independence. In *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–94)*, pages 286–292.

David Heckerman and John S. Breese. 1996. Causal independence for probability assessment and inference using Bayesian networks. In *IEEE, Systems, Man, and Cybernetics*, pages 826–831.

Max Henrion. 1989. Some practical issues in constructing belief networks. In *Uncertainty in Artificial Intelligence 3*, pages 161–173.

G. Kokolakis and P.H. Nanopoulos. 2001. Bayesian multivariate micro-aggregation under the Hellinger's distance criterion. *Research in Official Statistics*, 4(1):117–126.

J.F Lemmer and Gossink. 2004. Recursive noisy-or: A rule for estimating complex probabilistic causal interactions. In *IEEE Transactions on Systems, Man and Cybernetics.*, pages 2252–2261.

P. Lucas. 2005. Bayesian network modelling through qualitative patterns. AI Journal.

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

Haiqin Wang, Denver Dash, and Marek J. Druzdzel. 2001. A method for evaluating elicitation schemes for probabilities. In *Proceeding of the Fourteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS–2001).*

Adam Zagorecki and Marek J. Druzdzel. 2004. An empirical study of probability elicitiation under noisy-or assumption. In *Proceeding of the Fourteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS–2004).*