Accuracy and robustness of learning and inference in Bayesian networks
Nunnink, J.R.J.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Accuracy and robustness of learning and inference in Bayesian networks

Jan R. J. Nunnink
ACCURACY AND ROBUSTNESS OF LEARNING AND INFERENCE IN BAYESIAN NETWORKS

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Universiteit van Amsterdam op gezag van de Rector Magnificus prof. dr. D.C. van den Boom ten overstaan van een door het college voor promoties ingestelde commissie, in het openbaar te verdedigen in de Agnietenkapel op vrijdag 6 juni 2008, te 10:00 uur.

door

Johannes Robertus Jacobus Nunnink

geboren te Alkmaar
Contents

1 Introduction 1
1.1 Bayesian Networks ............................................. 1
1.2 Applications of Bayesian Networks .......................... 3
1.3 Bayesian Network Grounding .................................. 3
1.4 Our Approach ..................................................... 4
1.5 Thesis Outline .................................................... 5
1.6 Brief History of the Thesis .................................... 5

2 Overview of Bayesian Networks 7
2.1 Probabilistic Models ............................................. 7
  2.1.1 Factors and Fragments ...................................... 9
  2.1.2 Factor Graphs ............................................... 11
2.2 Inference ........................................................... 12
  2.2.1 Lambda-Pi Message Passing ............................... 12
  2.2.2 Inference on Arbitrary Bayesian Networks ............ 14
  2.2.3 Inference in Factor Graphs ............................... 16
2.3 Learning a Bayesian Network .................................. 18
  2.3.1 Structure Learning ......................................... 18
  2.3.2 Parameter Learning ........................................ 19
  2.3.3 The EM Algorithm .......................................... 20
  2.3.4 MAP learning .............................................. 21
  2.3.5 Gradient Descent .......................................... 22
2.4 Conclusion ....................................................... 22

3 Accuracy and Belief Propagation 25
3.1 Introduction ....................................................... 25
3.2 Topology and Factors ........................................... 26
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1</td>
<td>State Estimation</td>
<td>29</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Combining Factor Supports</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>Accuracy</td>
<td>33</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Estimation Accuracy</td>
<td>33</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Fragment Accuracy</td>
<td>34</td>
</tr>
<tr>
<td>3.4</td>
<td>Belief Propagation</td>
<td>36</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Belief Propagation through Voting</td>
<td>36</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Experiments</td>
<td>43</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Computational Complexity</td>
<td>46</td>
</tr>
<tr>
<td>3.5</td>
<td>Discussion</td>
<td>47</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Related Work</td>
<td>47</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Conclusion</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>Parameter Learning</td>
<td>51</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>51</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Rationale</td>
<td>53</td>
</tr>
<tr>
<td>4.2</td>
<td>Quantized EM</td>
<td>55</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Parameter Space Quantization</td>
<td>56</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Convergence Properties</td>
<td>58</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Experiments I</td>
<td>60</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Experiments II</td>
<td>61</td>
</tr>
<tr>
<td>4.3</td>
<td>Improving Quantized EM</td>
<td>63</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Data perturbation</td>
<td>64</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Experiments III</td>
<td>66</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Computational Cost</td>
<td>67</td>
</tr>
<tr>
<td>4.4</td>
<td>Conclusion</td>
<td>67</td>
</tr>
<tr>
<td>5</td>
<td>Fault Analysis</td>
<td>73</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>73</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Related Work</td>
<td>74</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Our Approach</td>
<td>74</td>
</tr>
<tr>
<td>5.2</td>
<td>Adequate Factors</td>
<td>75</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Lower Bound</td>
<td>76</td>
</tr>
<tr>
<td>5.3</td>
<td>Voting Properties</td>
<td>79</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Filtering</td>
<td>82</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Experiments</td>
<td>84</td>
</tr>
<tr>
<td>5.4</td>
<td>Localization of Inadequacies</td>
<td>86</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Factor Adequacy</td>
<td>86</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Localization</td>
<td>89</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Experiments</td>
<td>93</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Inadequacy or Inaccuracy</td>
<td>96</td>
</tr>
<tr>
<td>5.5</td>
<td>Conclusion</td>
<td>96</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Network Limitations</td>
<td>97</td>
</tr>
</tbody>
</table>
Contents

6 Discussion and Future Research 99
   6.1 Future Research .......................... 101

Bibliography 103

Summary 111

Samenvatting 113

Publications 115

Acknowledgments 117
Introduction

Forecasting, diagnosis, and monitoring are some of the most common aspects of applications in artificial intelligence (e.g. [Edw98] [HHN92] [BSCC89]). The key component of such applications is the automated reasoning about observable and hidden events in their problem domain. This reasoning can be complex since the domains typically involve a high degree of uncertainty.

The amount of available relevant information is increasing. This information can be very diverse, coming from sensors, databases, and human reports. It is inherently heterogeneous, ambiguous, and uncertain. Sensors are subject to noise and occlusion, and automated interpretation of their signals can introduce uncertainty and errors. Human reports can be biased and are difficult to interpret directly.

Accurate reasoning in domains with such characteristics requires an adequate type of model. An adequate model should be able to handle uncertainty, heterogeneity and conflicting observations. It should have associated inference algorithms that are efficient in large domains. Furthermore, it should be robust against errors in its parameters, since small errors often cannot be avoided.

1.1 Bayesian Networks

Over the past two decades, Bayesian networks have emerged as a popular model for dealing with uncertainty and heterogeneity [Pea88]. They are popular for a variety of reasons. Bayesian networks are probabilistic graphical models that represent uncertainty and ambiguity using a rigorous probabilistic framework. The information heterogeneity is handled by modeling all events in a domain as states of random variables. Probabilistic inference is used to compute distributions over the states of unobserved variables given observations about other events. The ability to handle uncertainty and conflicting observations is a crucial
feature, and this is one of the advantages of Bayesian networks over logic or rule-based approaches, even though some efforts have been made to combine logic and probabilistic methods [RD06].

Another strength of a Bayesian network is that it combines both diagnostic reasoning (from effects back to the cause) and predictive reasoning in a seamless way [Pea88]. Based on a set of independence assumptions captured by the Bayesian network, efficient inference algorithms exist. The efficiency of inference is one of the advantages over the Dempster-Shafer approach, which uses complex propagation functions, which involve a powerset of the events in the domain. In Bayesian networks, the complete model can be specified using a limited number of direct probabilistic dependencies. These dependencies are conveniently visualized using a directed graph.

Another advantage is that these ‘local’ conditional dependencies are a very intuitive way of encoding domain knowledge. They correspond directly to basic elements of human reasoning, namely which events directly influence which other events [Pea88]. This is especially true in the case where the local dependencies correspond to causal relations. The strengths of these local influences are defined through conditional probabilities. This again mimics human reasoning. As a result, a Bayesian network can be considered a ‘white box’; all parameters, namely the network graph and conditional probabilities, have clear semantics, and it is usually possible to explain why a Bayesian network gives a certain output. This is an advantage over an approach such as neural networks, in which the actual meaning of the weights and functions is not clear. Consequently, those parameters can only be found through learning methods, and not through prior knowledge from domain experts.

Furthermore, this transparent parametrization facilitates the use of prior knowledge in the modeling process. Human domain experts are generally proficient at specifying the graph topology. The conditional probabilities are more difficult to extract from prior knowledge, however it is often possible to give rough estimates. Prior knowledge can easily be combined with statistical data. If the network graph is specified by humans, the parameters can be learned from data. Prior knowledge could give a plausible starting point for the learning process. In the case of rare events for which there is insufficient learning data, prior knowledge can fill the gaps.

Lastly, the output of probabilistic inference with Bayesian networks is a posterior probability distribution over some event(s) of interest. For most applications, this can be considered sufficient information. For example, it allows for optimal decision making and is used in filtering processes. If we combine Bayesian networks with utilities, we get decision graphs, which are powerful decision making tools [Jen01].
1.2 Applications of Bayesian Networks

Bayesian networks are used for many different applications. As mentioned above, the posterior distribution over the states of some variable can be combined with utilities or pay-offs for optimal decision making. State estimation, or classification, is a related task. In this case, one is interested in the most probable state of some variable, or the state whose posterior probability exceeds some threshold. State estimation with Bayesian networks has for example been applied to spam e-mail filtering [SDHH98], or monitoring the state of a water purification plant [JKOP89].

Prediction, or forecasting, is a popular application of Bayesian networks. From the current observable events, the probability distribution over possible future events is inferred. Examples are weather forecasting [Edw98], oil price forecasting [AF91], or predicting poker actions [KNJ99].

Another interesting application is data mining. By learning both the graph and the conditional probabilities of a Bayesian network from a large set of data, one can discover new relations among the data. Data analysis has seen applications such as earthquake analysis [AR03], search-and-rescue-operation analysis [NQRvdM07], and risk analyses [BKRK97, HWLM05].

Many applications are related to the medical field of research. The problem domains in this field often have a causal structure, such as diseases causing symptoms. Bayesian networks are well suited to model such domains. Examples are: diagnosis systems [AJA89, HHN92], data analysis [AHB91, JK96], and patient monitoring [BSCC89].

1.3 Bayesian Network Grounding

While a Bayesian network is a suitable model for many application domains, building an adequate model introduces some new problems, however. For example, data is typically incomplete, and learning algorithms can only find local maxima. Domain experts may be biased and imprecise, or can simply make mistakes.

The result is that the probability distribution defined by a Bayesian network will never be equal to the true distribution over events in the real world. Still we continue to use Bayesian networks in good faith. A question that needs to be answered is why this is allowed. What is the justification for working with models which we know are not perfect?

Some literature studies have shown that Bayesian networks can be very robust against small errors in parameter values [PHP96]. Many of these studies dealt with the case of so-called naive Bayesian networks, heavily simplified versions of the typical networks [DP96, CD03]. Others showed this robustness from the perspective of a sensitivity analysis [CD02, CvdG02]. This is a method to analyze how sensitive the outcome of probabilistic inference is to a change in the value of
Many of these analyses are done in the context of state estimation. This allows for a large margin of error, since the precise posterior distribution is not as important. The only thing that matters is which state is most probable. In the case of prediction, one is interested in the precise probability of a certain event. Any deviation from the correct distribution can then be significant.

In other words, it is known that in a state estimation context a Bayesian network is probably still adequate if we make small errors in the parameter values. However, this raises the question whether we can quantify this 'small'. When does the error become too large? [vdGR01a, RvdG04] show how to compute the maximum amount of change that can be applied to a single parameter value without changing the state estimation outcome.

However, we seek a more general answer. Is there some crucial relation between the parameters and network structure that needs to be satisfied to make a certain Bayesian network work reliably in reality? Are there problem domains that are inherently difficult to model in a robust way?

1.4 Our Approach

The approach in this thesis towards answering that question begins with considering the equation for the posterior distribution over some variable of interest. This equation contains a factorization where each factor corresponds to a conditionally independent part of the Bayesian network graph. Each factor represents observations about the outcomes of independent stochastic processes.

It can be experimentally shown that networks that feature many such independent parts are very robust against changes in its parameter values. This can be explained by the fact that a change in the value of one of the factors has a small influence on the probabilities of the possible outcomes, as long as the factor’s value is not close to 0. On the other hand, networks with very few parts are not as robust.

Inspired by this observation, we propose a perspective which views the inference process as a voting process. Namely, each independent part can be seen as an independent expert, giving a weight for each state of the variable of interest. These vectors of weights are multiplied and normalized to form the posterior probability distribution. State estimation tries to find the most probable state of a variable of interest. In the case of state estimation, we can also perform majority voting on the maximum elements of each weight vector, instead of performing the multiplication. So each expert gives a vote about what it believes is the most probable state. This perspective explains the experimental observation that slightly changing the weights does not change the state estimation outcome, as long as the votes do not change.

Furthermore, if simple relations between the true distributions and the model
parameters hold, we can identify several properties of this voting process. These properties tell us for which types of Bayesian networks the voting perspective works well. This implies that under certain conditions, Bayesian networks are very robust, and remain accurate even if their parameter values are quite different from the true values. The perspective also allows us to quantify the extent to which parameters are allowed to deviate from the true values, without changing the votes. And it allows us to talk about the probability of obtaining an accurate vote.

1.5 Thesis Outline

The thesis is structured as follows: In Chapter 2, we give an overview of Bayesian networks, factor graphs, and the associated inference and learning algorithms. Chapter 3 presents the voting perspective mentioned in the previous section, and gives the conditions under which the voting procedure is exact. We show how to apply the voting principle on an inference algorithm, and we identify the properties of that algorithm.

In Chapter 4, we present an algorithm for parameter learning of Bayesian networks. The approach we take is to split the search for a good model into two phases. First, we perform a search in a quantized parameter space, and then continue the search in the full parameter space. The choice for a particular quantization is inspired by the voting perspective.

In Chapter 5, we investigate the probability of obtaining a correct support or vote, which is relevant for the properties of the algorithms presented. Furthermore, the voting principle is applied on the detection of incorrect state estimations, and the localization of modeling errors in Bayesian networks. These methods are based on using the number of votes for each state as a coarse confidence measure.

Finally, in Chapter 6 we conclude and indicate directions of possible future research.

1.6 Brief History of the Thesis

The results presented in this thesis are based on research performed over the past four years. However, it should be noted that during those years, other topics were studied that are not covered by this thesis.

I started out with a continuation of the research done during my final Master’s degree project. The topic was learning of Gaussian mixture densities with the EM-algorithm. We presented a method to learn from very large data sets, by partitioning the data in a hierarchical manner, and combined it with a greedy approach to adding mixture components [NVV04, VNV06].
Next, work began as part of a project at the Decis lab. The aim of this project was to develop automated sensing and decision support system for crisis environments. Our contribution to the project was to design a sensing and world-modeling system. It incorporated probabilistic information fusion in order to deal with the information uncertainty. A multi-agent approach was chosen in order to deal with the spatially distributed sensors, and with robustness against hardware failures.

Therefore, the next year of research focused on combining (Bayesian) probabilistic networks with multi-agent systems. This lead to several publications. Some dealt with the design of the distributed network itself, which we called distributed perception networks [PMN04, PdON05, PdOM+08]. Others with more specific subjects, such as the accuracy of probabilistic fusion of sensor data received in sequence [NP05a]. Another paper presented an approach to resolve conflicts between agents that both want control over a single information resource [NP05b]. In [dONP05], we compared the distributed approach to another popular distributed probabilistic network approach in the context of very dynamic agent networks (i.e. where agents will join and leave the network often).

However, at this point in the project, we decided to drop the multi-agent aspect, and continue research on probabilistic information fusion. In the interest of the overall project, we investigated the fusion accuracy in crisis domains, which are characterized by small training data sets, and uncertain parameters describing rare cases. This has led to the results presented in this thesis. The multi-agent based methods were less relevant in the context of this thesis, and therefore we decided to leave them out to create a more coherent story.
Chapter 2

Overview of Bayesian Networks

In this chapter we give an overview of discrete Bayesian networks and how they can be used to model a problem domain. We describe several methods to perform probabilistic inference with Bayesian networks. These methods either work directly on the Bayesian network, such as the lambda-pi algorithm, or on compiled structures such as a junction tree.

Furthermore, we describe factor graphs. This type of model is related to Bayesian networks. The standard inference method for factor graphs, sum-product, is a general case of some of the inference algorithms in Bayesian networks. One of the algorithms presented in Chapter 3 is based on this sum-product algorithm.

Finally, we discuss the construction of Bayesian networks, in particular learning from (incomplete) example data. We focus especially on maximum-likelihood parameter learning using the expectation-maximization algorithm. Methods from Chapter 4 are based on the EM learning algorithm and its variants.

2.1 Probabilistic Models

A Bayesian network [Pea88, Jen01, CDLS99] is a probabilistic model of a problem domain. All relevant concepts and events in that domain are modeled through discrete random variables. These variables will be denoted by upper case letters. The complete set of variables is \( \mathcal{X} = \{X_1, \ldots, X_n\} \). Each variable can take one out of a finite set of discrete values, also called states. These will be denoted by lower case letters, or sometimes by numbers. For example, variable \( X \) could have possible states \( \{x_1, x_2, \ldots, x_k\} \), or \( \{1, 2, \ldots, k\} \). In the case of binary variables we often use \( x \) and \( \bar{x} \) to denote the states ‘true’ and ‘false’ of a variable \( X \). One of the strengths of Bayesian networks lies in the fact that all information in a domain, no matter how heterogeneous in nature, is encoded in the same manner, through discrete variables.
In order to do probabilistic reasoning on the variables in the domain, a *joint probability distribution* (JPD) over the combinations of all variables’ states is defined, \( p(\mathcal{X}) \). Computation of the marginal distribution over a variable’s states, given evidence about the states of other variables, is done through marginalization and applying Bayes’ rule. For example for a domain consisting of 10 variables, and with observations \( X_1 = 3 \) and \( X_4 = 1 \):

\[
p(X_2 | X_1 = 3, X_4 = 1) = \frac{p(X_2, X_1 = 3, X_4 = 1)}{p(X_1 = 3, X_4 = 1)} = \frac{\sum_{X \setminus \{X_1, X_2, X_4\}} p(X_1 = 3, X_2, X_3, X_4 = 1, \ldots, X_{10})}{\sum_{X \setminus \{X_1, X_4\}} p(X_1 = 3, X_2, X_3, X_4 = 1, \ldots, X_{10})} \tag{2.1}
\]

However, since the JPD is exponentially large in the number of variables, this marginalization becomes intractable in large domains. A Bayesian network makes the computation tractable in the following way: First, the product rule allows us to factorize the JPD as follows:

\[
p(\mathcal{X}) = p(X_1 | X_2, \ldots, X_n) p(X_2 | X_3, \ldots, X_n) \ldots p(X_n).
\tag{2.2}
\]

This holds for every order of the variables (in this case \( X_1, X_2, \ldots, X_n \)). A Bayesian network chooses a particular ordering. Next, we can simplify the factors in (2.2), by using a set of *conditional independence assumptions*. Namely, two variables \( X_2 \) and \( X_4 \) are conditionally independent given \( X_3 \) if \( p(X_2 | X_3, X_4) = p(X_2 | X_3) \). In this manner, the factorization in (2.2) can be simplified to a factorization containing local distributions over less variables.

This ordering and set of assumptions is visualized with the help of a directed acyclic graph (DAG). In this graph, each node \( X_i \) corresponds to one variable \( X_i \), and the parents \( \text{Pa}(X_i) \) of node \( X_i \) correspond to the variables \( X_i \) is conditionally dependent on. An example is given in Figure 2.1. This Bayesian network represents the factorization

\[
p(\mathcal{X}) = p(A)p(B)p(C|A)p(H|A, B)p(E|C)p(F|H)p(G|H) \tag{2.3}
\]

In general, a Bayesian network represents the following factorization:

\[
p(\mathcal{X}) = \prod_{i=1}^{n} p(X_i | \text{Pa}(X_i)). \tag{2.4}
\]

Thus, we do not define the JPD directly, but through a number of *conditional probability tables* (CPTs), one for each variable. As we will show later, this factorization makes inference more efficient. In many applications it is convenient to let the
2.1 Probabilistic Models

![Bayesian Network Diagram]

Figure 2.1: An example Bayesian network.

\[
\begin{array}{c|cccc}
\text{Pa}(X_i) & 1 & \ldots & k \\
\hline
1 & p(X_i|\text{Pa}(X_i)) \\
X_i & \vdots \\
j & \\
\end{array}
\]

Figure 2.2: CPT structure.

direct conditional dependencies correspond to causal dependencies in the problem domain. In the conditional probability table for a variable \(X_i\), we let each row correspond to a state of \(X_i\), and each column to a state of its parents, as in Figure 2.2.

Some of the variables in the domain might be observed, so we know their value or state. These observations are called evidence, and collectively denoted by \(E\). Evidence variables are instantiated to their observed state. Bayesian networks make the process of inferring the marginal distribution over some variable(s) given the evidence more tractable compared to direct marginalization of the JPD. As we will explain in Section 2.2, marginalization of factorization (2.4) can be performed using more efficient algorithms.

In this thesis, in the context of state estimation, we will often denote the variable whose state we are trying to estimate by \(H\). The hidden true state of \(H\) in a particular case is then denoted by \(h^*\). We will also regularly refer to true distributions, meaning the distributions that perfectly describe the processes in the modeled domain. A true distribution will be denoted using a tilde: \(\tilde{p}\).

2.1.1 Factors and Fragments

The analysis in the next chapters requires a notion of dependence between different sections of a graph given a node of interest. For this we first need to define \(d\)-separation [Pea88]. \(D\)-separation directly corresponds to conditional independence: if two nodes \(X\) and \(Z\) are \(d\)-separated by a third node \(Y\) in a Bayesian network graph, the variables \(X\) and \(Z\) are assumed by the model to be conditional independent given \(Y\).
D-separation can be defined as follows: a \emph{path} in a graph is a sequence of nodes, where from each of its nodes there is a connection to the next node in the sequence. Consider three adjacent nodes on a path, denoted $\circ$, $\bullet$, and $\circ$. In a directed graph these three nodes can be linked in three different ways: serial ($\circ \rightarrow \bullet \rightarrow \circ$), diverging ($\circ \leftarrow \bullet \rightarrow \circ$), and converging ($\circ \rightarrow \bullet \leftarrow \circ$). Nodes $X$ and $Z$ are said to be d-separated by a set of nodes $Y$ iff every path between $X$ and $Z$ (i) contains a serial or diverging link where the middle node ($\bullet$) is an element of the set $Y$, or (ii) contains a converging link where neither the middle node ($\bullet$) nor any of its descendants is in $Y$. Similarly, two sets $X$ and $Z$ are d-separated by $Y$ if the d-separation holds for every pair of nodes from $X$ and $Z$.

For example, in Figure 2.1, $C$ and $H$ are d-separated given $A$, but $A$ and $B$ are not d-separated given $\{C, G\}$. If $Y$ corresponds to the evidence set $E$, d-separation can indicate the network parts that are rendered (in)dependent of each other. Similarly, we can use d-separation to identify fragments of the graph that are conditionally independent given a node of interest.

\textbf{Definition 1} Given a graph containing node of interest $H$, we can identify sets of nodes $X_k$, where every set is d-separated from the other sets given $H$. A fragment $F^H_k$ is defined as the subgraph consisting of nodes $X_k \cup \{H\}$, and all arcs between those nodes. Node $H$ is called the root of all fragments $F^H_k$. The number of fragments rooted in $H$ is called the branching factor.

See Figure 2.1 for an example. Given node $H$, 3 fragments can be identified, containing nodes $\{A, B, C, E, H\}$, nodes $\{F, H\}$, and nodes $\{G, H\}$. The reason that $B$ is in the first fragment is that $A$ and $B$ are not d-separated given $H$. Note that instantiation of certain nodes in a network with evidence can change the fragments.

In a state estimation context we are interested in the probability distribution over the states of a variable of interest and the evidence, $p(H = h_i, E)$. The fragmentation described above has the property that this probability distribution can be factorized such that each factor corresponds with a fragment. The distribution is obtained by marginalizing (2.4). If we regroup the terms per fragment, we can write (excluding the evidence for now):

\[
p(H = h_i) = \sum_{X \setminus H} \prod_{X_j \in X} p(X_j | \text{Pa}(X_j)) = p(h_i | \text{Pa}(H)) \sum_{X_j \in X_1} \prod_{X_j \in X_1} p(X_j | \text{Pa}(X_j)) \cdot \phi_1(h_i)
\]  
\[
\vdots
\]  
\[
\sum_{X_l \in X_K} \prod_{X_l \in X_K} p(X_l | \text{Pa}(X_l)) \cdot \phi_K(h_i)
\]  

(2.5)
Evidence can easily be included in this equation, without changing its general form. Namely, by instantiating the evidence variables, and excluding them from the sums. The sets $X_i$ were those from Definition 1, and correspond to the nodes in a fragment. In this way we can identify factors $\phi_k(h_i)$ whose product is the joint probability for each state $h_i$ and the evidence. $\phi_k(h_i)$ denotes the value of factor $\phi_k$ for state $h_i$. Note that the parents of $H$ are always in fragment 1. If $H$ is a root, this first factor is equal to the prior distribution over $H$.

The value of a factor depends on the evidence about variables in the corresponding fragment. The d-separation between the fragments implies that the factors in (2.5) are mutually independent, given variable $H$. In other words, if the evidence in one fragment changes, this does not influence the value of a factor corresponding to another fragment (as long as the fragments do not change).

### 2.1.2 Factor Graphs

Graphical models such as Bayesian networks factorize a global function, and can therefore be translated to a factor graph (FG) [Loe04, Bis06, KFL01]. Algorithms for inference in graphical models are often special cases of the inference algorithm in factor graphs. In the next chapter we will present an inference algorithm that operates on factor graphs.

If $X$ is the complete set of variables in the domain and $f$ is the global function over $X$, then we have a factorization of the following general form:

$$f(X) = f_1(X_1) \cdots f_n(X_n), \quad (2.6)$$

where $X_i$ are subsets of variables, and $f_i$ are the factors.

In the case of a factorization defined by a Bayesian network (see (2.4)), the global function is the joint probability distribution, the factors are the CPTs, and each subset consists of a node and its parents, $X_i \cup \text{Pa}(X_i)$.

A factor graph represents a factorization using an undirected graph, containing variable nodes and factor nodes. Each factor node $f_i$ is connected to exactly those nodes that correspond to variables in its $X_i$. Each variable node $X$ is connected to exactly those factor nodes $f_i$ for which $X \in X_i$. Hence, it is bipartite, meaning that two nodes of the same type can never neighbor each other. An example factor graph is shown in Figure 2.3(a), corresponding to the Bayesian network in Figure 2.1. It represents the following factorization:

$$f(A, \ldots, H) = f_1(A)f_2(B)f_3(A, C)f_4(A, B, H)f_5(H, G)f_6(C, E)f_7(F, H)$$

$$= p(A)p(B)p(C|A)p(H|A, B)p(G|H)p(E|C)p(F|H) \quad (2.7)$$

It is easy to see that any Bayesian network can be translated to a factor graph. Note that we could have built many different factor graphs representing the same global function as in (2.7), because factors can always be combined into 'larger' factors through multiplication. For example, see Figure 2.3(b) for the factor graph
2.2 Inference

Probabilistic inference involves the computation of probability distributions over one or more variables’ states given certain evidence. The simplest method for computing these posterior distributions, marginalization of the full joint distribution, is intractable for large networks. Therefore, more efficient algorithms were invented that exploit the conditional independencies represented by the network structure. Here we will give a short overview of some of these algorithms.

2.2.1 Lambda-Pi Message Passing

The first efficient algorithm for inference in Bayesian networks was proposed by Pearl [Pea86, Pea88, KP83]. This lambda-pi algorithm works on poly-tree networks, that is, networks without loops. It is used to obtain, for each network node $X$, the posterior probability distribution $p(X = x|\mathcal{E})$ over its states, given some evidence. This can be achieved by passing so-called lambda ($\lambda$) and pi ($\pi$) messages between nodes. Lambda messages are sent ‘upwards’ against the direction of the arcs, while pi messages are sent along the arc direction.

In general, two possible message passing schemes can be used. A sequential scheme treats one node at a time, using the incoming messages to compute new outgoing messages, before processing the next node. In poly-tree networks, by

where we multiplied $f_1 \cdot f_3 = f'_8$ and $f_2 \cdot f_4 \cdot f_5 = f'_9$. The global function on its own makes for a star-shaped factor graph.
choosing a suitable node order, only two visits to each node are required to obtain all posterior probabilities.

A parallel scheme processes all nodes in parallel. This is a more general scheme, since it requires no node order; its stopping criterion can be based on message convergence, and it can be applied to general networks, including those with loops. In this case the algorithm becomes approximate (more about this later).

In both schemes the same computations are performed for each individual node. These consist of recomputing its posterior probability and outgoing lambda and pi messages, given the current incoming messages (see Figure 2.4). The posterior probability is given by

\[ p(X = x | \mathcal{E}) = \alpha \lambda(x) \pi(x), \]  

where \( \lambda(x) \) is given by

\[ \lambda(x) = \lambda_X(x) \prod_j \lambda_{Y_j}(x). \]  

\( \lambda_X(x) \) is evidence about \( X \) which can overrule all incoming messages if set to a point mass distribution. \( \lambda_{Y_j}(x) \) is the current lambda message from \( Y_j \), the \( j \)-th child of \( X \). \( \pi \) is given by

\[ \pi(x) = \sum_z p(X = x | Z = z) \prod_k \pi_X(z_k). \]  

\( Z \) is the set of parents of \( X \), \( z \) is their combined state, and \( z_k \) is the state of the \( k \)-th parent within this particular combined state. \( \pi_X(z_k) \) is a value contained in the current pi message from \( Z_k \), a parent of \( X \). If \( X \) is a root node, then \( \pi \) is equal to its prior distribution.

Note that incoming lambda messages contain a value for each state of \( X \), whereas pi messages contain a value for each state of \( Z_i \), a parent of \( X \). Therefore, pi messages are multiplied with the CPT \( p(X | Z) \) (see (2.10)), before they are multiplied with the lambda messages to form the posterior probability over \( X \). This requires summing over all possible combined parent states in (2.10), which can be costly if \( X \) has many parents.

The new outgoing lambda message sent from \( X \), \( \lambda'_X \), to its \( k \)-th parent \( Z_k \) is a vector of values, one for each state of \( Z_k \). It is given by

\[ \lambda'_X(z_k) = \sum_x \lambda(x) \sum_{z_1, \ldots, z_j} \left( p(x | z_k, z_1, \ldots, z_j) \prod_{i=1}^j \pi_X(z_i) \right), \]  

where \( Z_1, \ldots, Z_j \) are the other parents of \( X \). The new pi message sent to child \( Y_j \), \( \pi'_{Y_j} \) is given by

\[ \pi'_{Y_j}(x) = \pi(x) \lambda_X(x) \prod_{k \neq j} \lambda_{Y_k}(x). \]  

Generally speaking, the new outgoing messages to a node \( A \) represents the information of all incoming messages excluding the message from \( A \) itself.
2.2.2 Inference on Arbitrary Bayesian Networks

The lambda-pi algorithm is only exact in poly-tree networks. That is, it is only guaranteed to compute the correct value of the posterior distributions in networks without loops. One of the solutions to this problem was proposed by Lauritzen and Spiegelhalter [LS88]. They convert a general network graph to a particular undirected tree structure, called a junction tree. This is done by combining nodes into hypernodes or cliques. Cliques which share variables can then be connected to form a network without loops.

The inference algorithm on this tree structure is analogous to the lambda-pi algorithm. Messages are sent between the cliques, containing information about the shared variables between two cliques. The distribution over the variables in a clique is represented by a potential, which is multiplied with incoming messages and marginalized to compute new outgoing messages.

During the construction of a junction tree, several different choices can be made. Hence, different junction trees can be constructed that each represent the same Bayesian network. The efficiency of the algorithm depends on the size of the cliques, since the size of a potential is exponential in the number of nodes in its clique. Finding the optimally efficient junction tree is a difficult task, but heuristics exist that can find good junction trees.

The further algorithms in this thesis do not depend on the exact junction tree procedure, and therefore we omit further details, which can for example be found in [LS88, CDLS99, Jen01].

Another method to perform inference in loopy graphs, is called loop-cutset conditioning [Pea88]. Instantiating certain variables (setting them to one of their states) can introduce new d-separations in a graph, effectively breaking loops (see Figure 2.5 for an example). Inference is then performed for every state of the instantiated variables, and the results are combined afterwards. However, the number of inference runs required is in general exponential in the number of loops, making the approach impractical in networks with many loops.
A method that is also exact in arbitrary networks is \textit{variable elimination} [ZP96]. Given a particular evidence set and variable of interest, it starts with the standard equation for the posterior probability. This equation is a marginalization of the factorization, similar to Equation (2.5). The algorithm eliminates the variables out of this equation one-by-one. For each variable, it does this by multiplying all factors that involve the variable, marginalizing the variable out, and putting the resulting factor back into the equation. For example, for the network from Figure 2.1, eliminating $A$ amounts to the following steps:

$$
p(H) = \sum_{X \setminus H} p(A)p(B)p(C|A)p(H|A,B)p(G|H)p(E|C)p(F|H)
= \sum_{X \setminus \{A,H\}} p(B)p(G|H)p(E|C)p(F|H) \sum_A p(A)p(C|A)p(H|A,B)
= \sum_{X \setminus \{A,H\}} p(B)p(G|H)p(E|C)p(F|H) \sum_A q(A, B, C, H)
= \sum_{X \setminus \{A,H\}} p(B)p(G|H)p(E|C)p(F|H)q(B, C, H),
$$

where $q$ is a new factor.

Evidence variables and the variable of interest are not eliminated. The order in which variables are eliminated determines the complexity of the algorithm (for example, eliminating $A$ was inefficient). Finding an optimal ordering is a similar problem to finding an optimal junction tree, as the algorithms are based on the same general principle. However, variable elimination does not require compilation into another network structure. If one is only interested in the distribution over one (set of) variable(s), and the evidence is fixed beforehand, then variable elimination can be more efficient.

The above algorithms are exact, and can be computationally expensive. Popular algorithms for approximate inference are sampling methods, in particular \textit{Gibbs sampling} [Pea87]. Sampling methods are used in cases where some distri-
bution of interest is too hard or impossible to compute directly, but where it is possible to obtain samples from that distribution. A histogram of those samples is then an approximation of the distribution.

Gibbs sampling in Bayesian networks works as follows: First, the evidence variables are set to their observed states, and all other variables are set to a random state. Next, each variable in turn has their state re-sampled from their local distribution conditioned on the current state of its neighbors. After all variables are re-sampled, the complete sample is stored, and the loop over the variables is repeated. The sampled data set will slowly converge to a stationary distribution. Because this final distribution also depends on the initial (random) sample, the first group of samples is usually not stored.

2.2.3 Inference in Factor Graphs

As described in Section 2.1.2, most graphical models (such as hidden Markov models, Bayesian networks, etc) can also be represented by an equivalent factor graph. It turns out that most of the inference algorithms in these graphical models are special cases of a general inference algorithm on factor graphs. This algorithm is called the sum-product algorithm, and like the lambda-pi and junction tree algorithms, it works through message passing in a graph. In the case of factor graphs this means sending messages from factor nodes to variable nodes and vice versa [Bis06, KFL01].

For some variable $X$, the marginal distribution can be found by summing the global function over all variables (denoted by $\mathcal{X}$), except for $X$,

$$f(X) = \sum_{\mathcal{X} \setminus X} f(\mathcal{X}). \tag{2.14}$$

Since first computing the global function and then performing summation is very inefficient, inference algorithms exploit the factorization.

Sum-product works by sending messages between the nodes in the factor graph. Each outgoing message can be computed from the incoming messages with the following equations: for the message from a variable node $X_m$ to a factor node $f_s$ (see Figure 2.6(a)):

$$\mu_{X_m \rightarrow f_s}(X_m) = \prod_{t \in \text{ne}(X_m) \setminus f_s} \mu_{f_t \rightarrow X_m}(X_m), \tag{2.15}$$

where $\text{ne}(\cdot)$ denotes the neighbors of a node in the factor graph. For the message from factor to variable node (see Figure 2.6(b)):

$$\mu_{f_s \rightarrow X}(X) = \sum_{X_1} \cdots \sum_{X_M} f_s(X, X_1, \ldots, X_M) \prod_{m \in \text{ne}(f_s) \setminus X} \mu_{X_m \rightarrow f_s}(X_m) \tag{2.16}$$
Finally, the marginal over a variable $X$ is given by:

$$p(X) = \prod_{s \in \text{ne}(X)} \mu_{f_s \rightarrow X}(X) \quad (2.17)$$

The order in which all these messages are sent depends on the structure of the graph. If the factor graph contains no loops, then we begin by choosing one (variable) node as the ‘root’ node. If we know beforehand which variable we want to compute the marginal distribution for, it is convenient to pick that variable as the root. This creates an undirected tree. We will then start at the leaves and send messages (only upwards) towards the root. Each node waits with sending a message until it has received messages from all its other neighbors. The starting messages are trivial; for the factor leaves, $\mu_{f_s \rightarrow X}(X) = f_s(X)$, and for variable leaves $\mu_{X \rightarrow f_s}(X) = 1$. Evidence (clamped variables) can be incorporated by having such a variable node send a message to itself, containing only 0’s and a 1.

If we are only interested in the marginal over the root variable, then we can stop and use (2.17). If we are interested in the marginal for other variables as well, we have to do a second message ‘sweep’, this time outwards from the root. Once a variable node has received a message from all its neighbors, it can use (2.17) to compute its marginal.

If a factor graph has loops, then the above method cannot be directly applied, because messages could run around within the loops indefinitely. Loops in the graph can be eliminated by multiplying factors. This is related to converting a Bayesian network to a junction tree. In the case that one wishes to operate on the loopy graph, a parallel scheme can be used. In this scheme, all nodes send messages to all their neighbors simultaneously. This can be stopped when the messages have converged, or a maximum number of iterations has been reached. The initial messages can again be set to 1. This version of sum-product is approximate. However, the convergence properties have been studied (mainly in the context of ‘loopy’ lambda-pi propagation), and it turns out that the outcomes are
Overview of Bayesian Networks

quite good for many networks [MWJ99, WF01, YFW01]. Furthermore, it is easy to implement and fast, while methods such as junction tree can require a lot of computations to construct the structure.

2.3 Learning a Bayesian Network

Construction of a Bayesian network can be done manually, using prior knowledge from domain experts. Especially the network graph can often be specified adequately by humans. Defining the precise conditional probabilities is more difficult. An alternative is then to learn a Bayesian network from example data using learning algorithms.

Most learning algorithms attempt to maximize some score. In the case of Bayesian networks, the most often used scores are the model’s posterior probability and the model likelihood.

Let $BN_i$ denote a particular Bayesian network, that is, a particular graph and set of conditional probabilities. Let $D$ be the training data set. The maximum a posteriori (MAP) model $BN_{map}$ maximizes the model posterior, given the training data:

$$BN_{map} = \arg \max_{BN_i} p(BN_i|D) = \arg \max_{BN_i} p(D|BN_i)p(BN_i).$$  \hspace{1cm} (2.18)

However, this expression requires a prior distribution over the different models $p(BN_i)$. To avoid specifying this prior distribution, it is often assumed to be uniform, in which case the MAP model is equal to the maximum likelihood (ML) model. The ML model maximizes the parameter likelihood:

$$BN_{ml} = \arg \max_{BN_i} p(D|BN_i).$$  \hspace{1cm} (2.19)

As explained in Section 2.1, a Bayesian network consists of two elements: (i) the network graph, which defines the direct dependencies among the variables, and (ii) the conditional probabilities. When constructing a Bayesian network, these two elements are typically determined separately.

2.3.1 Structure Learning

Learning the structure of a Bayesian network is a difficult task. The number of possible network structures is more than exponentially large in the number of nodes in the graph. Many approaches to the problem have been proposed [Nea04, Hec99, Fri98, Pea88, CL68], but no method is currently accepted as the standard.

In this thesis we have chosen not to focus on the subject of structure learning. Structure learning is the most difficult of the two tasks, and an adequate structure can often be found with the help of domain experts. Humans are typically
2.3 Learning a Bayesian Network

good at specifying the direct dependencies in a problem domain, especially when it involves causal dependencies. Humans are less good at specifying precise probabilities. Therefore, we assume the network structure to be fixed prior to learning the parameters.

2.3.2 Parameter Learning

The parameters of a Bayesian network are the entries of the conditional probability tables. Let the set of all parameters jointly be denoted by $\Theta$. $\theta_i$ denotes the set of parameters associated with one variable $N_i$, and $\theta_{ijk}$ is one particular entry, namely the value of $p(N_i = j|Pa_i = k)$, i.e. the probability that variable $i$ is in state $j$, given that $i$’s parents are in the (combined) state $k$.

Learning the parameters of a Bayesian network means finding a value for $\Theta$ that maximizes a certain score, for example ML or MAP, given example data $D$. $D$ denotes a set of cases $\{x_1, \ldots, x_n\}$, where each case $x_i$ contains the states of (a subset of) all variables. Let us focus first on the maximum likelihood score. We are trying to find

$$\Theta_{ML} = \arg \max_\Theta p(D|\Theta).$$  \hspace{1cm} (2.20)

To simplify computations, one typically maximizes the parameter likelihood by maximizing the log-likelihood $L$. By assuming independent and identically distributed data and independence between the parameters from different CPTs, the likelihood can be maximized by maximizing for each variable $N_i$:

$$L(D, \theta_i) = \sum_{x \in D} \log p(N_i|Pa_i, x, \theta_i).$$  \hspace{1cm} (2.21)

We need to distinguish two possible situations with respect to the data set. The data set is called complete if in every data case the state of all variables is known. If the state of one or more variables is unknown in part of the data set, it is incomplete. If the data is complete finding $\Theta_{ML}$ is a trivial task, and the solution is given by:

$$\hat{\theta}_{ijk} = \frac{\text{Count}(N_i = j, Pa_i = k)}{\text{Count}(Pa_i = k)},$$  \hspace{1cm} (2.22)

where Count($a = b$) returns the number of data cases in which a set of variables $a$ was in state $b$. Since these counts are the only statistics of the data required to estimate the ML parameters of the model they are called the sufficient statistics.

If the data is incomplete, we cannot compute the counts, and therefore cannot find the ML parameters using (2.22). In this case we need to perform a search for the model with the highest score. The expectation-maximization (EM) algorithm and gradient descent are the most used methods.
2.3.3 The EM Algorithm

The EM algorithm is a popular method for finding the maximum likelihood parameters of a probabilistic model given an incomplete data set with missing values [DLR77]. It performs a local search of the parameter space by iteratively improving the current parameter estimates until convergence.

An incomplete data set prevents us from directly computing sufficient statistics. We cannot compute the counts if we are unsure about the state of a variable. Subsequently, we cannot use (2.22) to maximize the data likelihood. The EM algorithm provides the following recipe for finding a likelihood maximum despite having incomplete data: First, we compute the expected value of the sufficient statistics, given the current parameter estimates and the incomplete data. Next, the parameters are maximized under the expected sufficient statistics. These are called the E and the M step, and the sufficient statistics are the counts:

**E-step** For each variable $N_i$ and all $j$ and $k$, compute:

$$E[\text{Count}(N_i = j, Pa_i = k)] = \sum_{x \in D} p(N_i = j, Pa_i = k|x; \Theta) \quad (2.23)$$

**M-step** For each variable $N_i$, update the entries of its CPT:

$$\hat{\theta}_{ijk} = \frac{E[\text{Count}(N_i = j, Pa_i = k)]}{E[\text{Count}(Pa_i = k)].} \quad (2.24)$$

The count over $Pa_i$ can be obtained by summing the counts in the numerator over $j$ [DLR77, Lau95, CDLS99].

The E-step requires one full belief propagation per data case, in order to compute the posterior probabilities for each variable, given the observed variables. A by-product of this propagation is the total data likelihood, which can be used for monitoring the algorithm’s convergence.

**Variational View**

A different view of the EM algorithm was given in [NH98]. They showed that the E-step (computing the expected sufficient statistics) and the M-step (maximizing the log-likelihood given those statistics) are equivalent to iteratively maximizing two forms of a function $F(q, \Theta)$ they called free energy. The free energy depends on the parameters $\Theta$, and some distribution $q$.

In the E-step, the following form gets maximized with respect to the distribution $q$ while keeping $\Theta$ fixed:

$$F(q, \Theta) = L(D, \Theta) - \text{KL}(q(y) \| p(y|D, \Theta)), \quad (2.25)$$
where \( y \) denotes the states of the unobserved variables, as used for the sufficient statistics. KL denotes the Kullback-Leibler divergence, a measure for the difference between two distributions \( p_1 \) and \( p_2 \):

\[
KL(p_1(X) \parallel p_2(X)) = \sum_{x \in X} p_1(x) \log \frac{p_1(x)}{p_2(x)}. 
\] (2.26)

Since the log-likelihood term does not depend on \( q \), maximizing this boils down to setting \( q(y) = p(y|D, \Theta) \). This is exactly what happens in the classical EM view’s E-step, if we think of \( q \) as the expected sufficient statistics.

In the M-step, the following form gets maximized with respect to the parameters \( \Theta \), while keeping \( q \) fixed:

\[
F(q, \Theta) = E_q[\log p(y, D|\Theta)] + H(q). 
\] (2.27)

Here, the entropy term \( H \) does not depend on \( \Theta \), so instead we maximize the expected log-likelihood given the distribution \( q \). This is equivalent to the classical M-step.

To guarantee convergence to a (local) free energy maximum, we do not necessarily have to maximize the free energy in each step. Merely strictly increasing \( F \) in each step is sufficient for convergence. It can also be shown that a (local) maximum of the free energy is a (local) maximum of the true likelihood. Hence, this view opens the way for different variations on the standard EM algorithm.

It is important to note that the total free energy can be written as a sum of the free energy of every data case. So we could choose to perform the E-step for only a subset of the data cases, or the M-step for only a subset of the variables.

It is allowed to choose any new \( q \) in the E-step, as long as it decreases the KL-divergence to the posterior probability distribution over the variables. For example we could use a winner-takes-all variant, where for each data case, we add 1 to the count of only the most likely combination:

\[
E[\text{Count}(N_i = j, Pa_i = k)] = 
\sum_{x \in D} \begin{cases} 
1 & \text{if } j, k = \arg \max_{j,k} p(N_i = j, Pa_i = k|x, \Theta) \\
0 & \text{otherwise}
\end{cases} 
\] (2.28)

As long as we restrict \( q \) to a certain type of distribution, it is often possible to show that it still guarantees convergence. Similarly, we can restrict \( \Theta \) to a particular set of values, as long as a new estimate is not worse than the previous estimate.

### 2.3.4 MAP learning

As mentioned above, learning the maximum likelihood model is a special case of learning the maximum a posteriori model. Instead of searching for the \( \Theta_{ML} \) that
maximizes the data likelihood, $p(D|\Theta)$, MAP learning tries to find the $\Theta_{MAP}$ with maximum posterior probability given the data:

$$\Theta_{MAP} = \arg \max_{\Theta} p(\Theta|D) = \arg \max_{\Theta} p(D|\Theta)p(\Theta). \quad (2.29)$$

We need to choose a prior distribution $p(\Theta)$ over the multinomial distributions parameterized by $\Theta$. If we choose to give every possible value of $\Theta$ equal prior probability, then MAP estimation becomes equivalent to ML estimation.

If we choose a Dirichlet distribution which is parameterized by a vector $\alpha$, then the MAP estimate for complete data is given by

$$\hat{\theta}_{ijk} = \frac{\text{Count}(N_i = j, Pa_i = k) + \alpha_{ijk}}{\text{Count}(Pa_i = k) + \alpha_{ik}}, \quad (2.30)$$

where $\alpha_{ik} = \sum_j \alpha_{ijk}$ [Hec99]. The parameters $\alpha_{ijk}$ are manually chosen. The priors on the parameters take the form of extra counts, added to the counts obtained from the data set. In the context of the EM algorithm, these extra counts are added similarly to the expected counts computed in the E-step. With these extra counts in the E-step, the algorithm is also called penalized EM [Gre90].

This method can be used for example to avoid unwanted extreme values, such as 0 or 1, while learning, by setting all $\alpha_{ijk} = 1$. Another application could be to ‘smoothen’ the CPTs during learning, by setting all $\alpha_{ijk}$’s to the same (high) value, and decreasing this value during learning. This can help to avoid overfitting.

### 2.3.5 Gradient Descent

Another approach to parameter learning is to do gradient descent on some score function [Nea04, Hec99, Jen01]. The log-likelihood (Equation (2.21)) is a typical score. Other score functions can for example be based on the difference between the posterior probabilities under the current model and the observations in the data set. The Euclidian distance or KL-divergence are typical error scores.

In an iterative process, partial derivatives for each parameter on the score function can be used to improve the model’s score with each iteration. Computing these derivatives requires a number of belief propagations similar to those required by EM, one per data case. However, a tuning parameter has to be introduced which controls the size of the step up the gradient surface. Therefore, the EM algorithm can be preferred above gradient descent, although gradient descent is generally more versatile.

### 2.4 Conclusion

This chapter gives an overview of Bayesian networks and their associated algorithms. A Bayesian network is a probabilistic model that can be used to model
problem domains involving uncertainties. It factorizes the joint probability distribution into a number of conditional probability distributions. Direct dependencies between variables are visualized in a directed graph. Thus, the (causal) network structure and the conditional probabilities can be specified separately.

Efficient algorithms for probabilistic inference exist, which exploit the factorization. A Bayesian network can be translated into an equivalent factor graph. The sum-product belief propagation for factor graphs can therefore also be used for inference in Bayesian networks.

The network structure can be specified by human domain experts. Humans are however not good at specifying precise conditional probabilities. Therefore, these probabilities are often learned from a set of example data cases. Learning algorithms try to maximize a score, such as model posterior probability or model likelihood. If the data sets are incomplete, search algorithms such as EM can be used to find the parameter values that maximize the score.
Chapter 3

Accuracy and Belief Propagation

In this chapter, we investigate properties of Bayesian networks in the context of accurate state estimation.\(^1\) The relation between the accuracy of state estimation and the correctness of model parameters is difficult to quantify. Several studies have shown that in certain situations state estimation can be robust against noise and small errors in the value of parameters. The question remains how much difference between the perfect and the actual parameter values is allowed for an accurate state estimation.

We show that under certain conditions, we can use a coarse voting perspective on the inference process. It allows us to show that state estimation with Bayesian networks can be very robust, even if we use very uncertain models and evidence. Based on this perspective, a voting propagation algorithm is presented, which achieves a good accuracy at a low computational cost.

3.1 Introduction

State estimation is based on the estimated probability distribution over states of a hidden variable of interest. This distribution can be estimated using a Bayesian network, which facilitates the systematic combination of observations with prior knowledge about the stochastic processes. Bayesian networks define a mapping between observations and variables of interest. Consequently, the model has a significant impact on the state estimation accuracy. On the other hand, one of the most challenging problems associated with Bayesian networks is determination of adequate modeling parameters [DvdG00].

We emphasize a fundamental difference between the model accuracy and estimation accuracy. A Bayesian network is a generalization over many possible situ-

\(^1\)Parts of the material in this chapter have appeared in [NP06, NP07].
26 Accuracy and Belief Propagation

ations. This generalization captures the probability distribution over the possible events in the problem domain. If a model exactly captures the true distribution, we can call it an accurate model. However, an accurate model does not necessarily support an accurate state estimation in a particular situation.

Given that it is very difficult to find the perfect generalization, it seems important to know what characterizes a model that is robust against sub-optimal parameters. If we would be interested in the precise distribution over a variable of interest, almost no deviation from the perfect generalization would be allowed. Therefore, it is more relevant to focus on the state estimation accuracy. A state estimation can be the basis for decision making processes, and a wrong decision can be disastrous. State estimation allows more room for sub-optimal parameter values [DP96]. The question is how much the parameter values are allowed to change without changing the state estimation.

To help answer this question, we present a voting perspective on the inference process. We derive the condition under which this perspective gives the same state estimation as exact inference. This perspective allows us to identify certain inequality relations between the true and estimated parameter values. When these relations are satisfied, the estimation accuracy of a Bayesian network will be close to that of a perfect generalization. As these inequalities can be satisfied by a large interval of values, a Bayesian network is inherently robust against small parameter changes, as long as the changes stay within the interval.

Based on this perspective, we derive a coarse propagation algorithm based on voting. The coarseness is based on retaining the aspects that the perspective identifies as significant for the state estimation accuracy, while ignoring other details. This allows the algorithm to be faster than existing algorithms, while still maintaining a good estimation accuracy for certain types of networks. It is implemented as a variant of the sum-product algorithm in factor graphs.

3.2 Topology and Factors

We first investigate the influence of different network topologies on the robustness to parameter value changes. Given the diversity of possible topologies, this seems very challenging. Therefore, we will compare two basic structural elements that often appear as elements of larger networks, namely predictive and diagnostic elements [Pea88].

Predictive Elements

A simple predictive element consists of a single hypothesis node $H$, with $n$ parent nodes $E_i$. See Figure 3.1 for an illustration. In a causal context, the parent nodes can represent causes and the hypothesis node the effect. The effect depends on the combination of the states of the cause nodes. In this simple setting, we assume
that we have observed the states of the parent nodes. The parent nodes are
instantiated with evidence and we infer the posterior distribution over the states
of $H$. Thus, we reason in the causal direction about the possible outcome of a
stochastic causal process.

Node $H$ is associated with a conditional probability table $p(H|E_1, \ldots, E_n)$. Assume that all variables have two states. The evidence set contains the observed
state of each parent node. Let the observed state of node $E_i$ be $e_i$. If the evi-
dence is $\mathcal{E} = \{e_1, \ldots, e_n\}$, the distribution over the states of node $H$ is given by
$p(H|e_1, \ldots, e_n)$. This directly corresponds to a column in $p(H|E_1, \ldots, E_n)$ (see
also Figure 2.2).

**Diagnostic Elements**

A simple diagnostic element consists of a hypothesis node $H$, with $n$ child nodes
$E_i$ (see Figure 3.2). In a causal context, $H$ represents a cause and the child nodes
its effects. In this simple setting, we assume that $H$ is hidden, and the effects
are observed. We are trying to infer the state of $H$ from evidence about variables
$E_1, \ldots, E_n$. Inference is based on reversal of the causal relations.

Each child node $E_i$ is associated with a conditional probability table $p(E_i|H)$. Again, assume that all variables are binary, and the evidence is denoted by $\mathcal{E} = \{e_1, \ldots, e_n\}$. The posterior distribution over the states of $H$ is given by

$$p(H|\mathcal{E}) = \alpha p(H) \prod_{e_j \in \mathcal{E}} p(e_j|H),$$

where $\alpha$ is a normalizing constant.
Structural Comparison

A perfect model will guarantee, with high probability, a small divergence between the true and estimated distribution over a variable. Changing the value of conditional probabilities (parameters) in the model will increase this divergence. Therefore, the robustness to parameter changes can be measured as the size of the parameter value domain that guarantees a small divergence.

In other words, a model can be considered robust, if it gives a greater domain of values from which the designer or learning algorithm can choose adequate parameters. This is important in the context of model grounding. After all, if a model allows more room for error, it is more likely that the model will work in practice. With the following small experiment we will illustrate the difference between predictive and diagnostic elements, with respect to this robustness.

The Kullback-Leibler (KL) divergence is a common measure for the difference between two distributions $p_1$ and $p_2$:

$$
\text{KL}(p_1(X) \parallel p_2(X)) = \sum_{x \in X} p_1(x) \log \frac{p_1(x)}{p_2(x)}. \quad (3.2)
$$

We will use it to measure the divergence between the true distribution over the hypothesis node and the inferred distribution. The true distribution is denoted by $\tilde{p}(H|\mathcal{E})$, and the inferred distribution by $p(H|\mathcal{E})$, given an evidence set $\mathcal{E}$.

We first consider the predictive network. We can assume a particular instantiation $\{e_1, \ldots, e_n\}$ of the $n$ parent nodes. The combination of these states corresponds to a single column in the conditional probability table. We want to see how much the divergence increases when we change the value of the parameters in that column.

Suppose that the true distribution is given by $\tilde{p}(h|\mathcal{E}) = 0.7$. We compute the KL divergence $\text{KL}(\tilde{p}(H|\mathcal{E}) \parallel p(H|\mathcal{E}))$ as a function of the relevant modeling parameter. We choose a threshold of 0.005 to indicate a ‘small’ divergence. The result is shown in Figure 3.3. The figure shows that a sufficiently small divergence can be achieved if the relevant parameter $p(h|e_1, \ldots, e_n) \in [0.65, 0.75]$, which is a narrow interval.

Next, consider the diagnostic network based on Figure 3.2. Let all $n$ children be associated with identical conditional probabilities. With binary variables, each CPT can be specified by two parameters $p(e|h)$ and $p(e|\overline{h})$. Suppose that the true values are $\tilde{p}(e|h) = 0.7$ and $\tilde{p}(e|\overline{h}) = 0.3$. We investigate the effect of changing $p(e|h)$ for all the evidence variables.

We assume that the true state of $H$ is $h$. The evidence set is sampled from the true distribution. This means that on average 70% of the evidence nodes are observed in state $e$. The true distribution is compared to the posterior distribution over $H$ after changing $p(e|h)$ in all conditional probability tables.

Figure 3.4 depicts the divergence between these two distributions for different values of $p(e|h)$. Each curve represents a different number of evidence nodes (10,
3.2 Topology and Factors

![Figure 3.3: Divergence between the true and the posterior distribution given different values of a parameter. The horizontal dotted line is a threshold on KL(\(\tilde{p}(H|\mathcal{E})\|p(H|\mathcal{E})\)).](image)

20, and 30). On the horizontal axis we can identify intervals for values of \(p(e|h)\), for which the divergence \(\text{KL}(\tilde{p}(H) \| p(H|\mathcal{E})) < 0.005\). From this diagram it is apparent that the interval, from which we can choose an adequate value for \(p(e|h)\), grows with the number of evidence nodes. In other words, a simple diagnostic network becomes inherently robust if we increase the number of children. In such cases we can pass the threshold under a wide choice of parameter values. This implies that the likelihood of choosing inadequate modeling parameters in a given situation is reduced.

Contrary to the predictive inference example, we see that increasing the number of evidence nodes improves the robustness to small parameter changes. A simple predictive network is sufficiently accurate only if we can obtain parameters that are very close to the true conditional probabilities. For simple diagnostic networks, parameter precision does not appear to be crucial. The explanation for this phenomenon lies in the conditional (in)dependencies between the evidence nodes. The ideas in the next sections are inspired by this observation.

3.2.1 State Estimation

State estimation tries to find the (hidden) state of a variable of interest. We adopt the view that a variable is estimated to be in the state with the highest posterior probability. Instead of the posterior probability we can also use the
joint probability with the evidence, since they are proportional. Formally, the state estimation of a variable $H$, given evidence $\mathcal{E}$, is then $h_i$ iff

$$h_i = \arg \max_{h_j} p(H = h_j, \mathcal{E}).$$  \hfill (3.3)

The joint probability for a variable and the evidence can be factorized based on the conditionally independent network fragments that connect to it (as explained in Section 2.1.1). Let the fragments of $H$ be denoted by $\mathcal{F}_1^H, \ldots, \mathcal{F}_k^H$, and their corresponding factors by $\phi_1, \ldots, \phi_k$. If we plug this factorization into state estimation equation (3.3), we get that $H$ is estimated to be in state $h_i$ iff

$$h_i = \arg \max_{h_j} \phi_1(h_j) \cdots \phi_k(h_j),$$  \hfill (3.4)

Each factor $\phi_i$ is a vector, giving a weight to each state of $H$:

$$\phi_i(H) = \begin{pmatrix} \phi_i(h_1) \\ \vdots \\ \phi_i(h_n) \end{pmatrix}$$  \hfill (3.5)

In (3.4), the maximum is taken after an element-wise multiplication of the vectors. Hence, a factor can be said to ‘support’ a state if it gives that state the highest
weight. Namely, it will contribute towards that state being the most probable one. This leads to the following definition:

**Definition 2** If $\phi_i$ is a factor of $H$, the supported state of $H$ by $\phi_i$ is denoted as $h_{\phi_i}$, and defined as

$$h_{\phi_i} = \arg \max_{h_j} \phi_i(h_j).$$

(3.6)

$h_{\phi_i}$ is also called the factor support.

### 3.2.2 Combining Factor Supports

There is a relation between which state has the highest posterior probability and which states are supported by factors. Obviously, if a certain state is supported by all factors, it will also be the most probable state. In case the factors do not support the same state, we can perform voting on the factor supports.

Let us first consider binary variables. Suppose that the magnitudes of the normalized weights (summing to 1) in the factors are equal for all factors. For example, the factors could be

\[
\begin{pmatrix}
0.15 \\
0.85
\end{pmatrix},
\begin{pmatrix}
0.15 \\
0.85
\end{pmatrix},
\begin{pmatrix}
0.15 \\
0.85
\end{pmatrix}.
\]

(3.7)

The bold numbers correspond to the supported states. It is easy to see that the state that gets supported by most factors is also the state with the highest posterior probability.

In practice, weights will not have exactly the same magnitudes. Consider the following example of a binary variable $H$ with 6 fragments:

\[
\begin{pmatrix}
0.15 \\
0.35
\end{pmatrix},
\begin{pmatrix}
0.65 \\
0.35
\end{pmatrix},
\begin{pmatrix}
0.35 \\
0.6
\end{pmatrix},
\begin{pmatrix}
0.4 \\
0.6
\end{pmatrix},
\begin{pmatrix}
0.4 \\
0.15
\end{pmatrix},
\begin{pmatrix}
0.85 \\
0.15
\end{pmatrix}.
\]

(3.8)

The posterior probability distribution is now $(0.31, 0.69)$. The state with the highest posterior ($h_2$) also turns out to be the state with the most factor supports. We can see here that in the case that the normalized weights are sufficiently close, or ‘balanced’, the most supported state is also the most probable state.

This can be formalized as follows: Let $a_i$ be the value of the first state in the $i$-th factor, and let $b_i$ be the second state. The first state is the most probable iff

$$\frac{a_1}{b_1} \cdot \frac{a_2}{b_2} \ldots \frac{a_n}{b_n} > 1,$$

(3.9)

and taking the logarithm

$$(\log a_1 - \log b_1) + \ldots + (\log a_n - \log b_n) > 0.$$

(3.10)
Let \( m \) be the set of factors \( i \) for which \((\log a_i - \log b_i) > 0\), and \( n \) its complement. Take the term with the greatest absolute value, and let this absolute value be denoted by \( q \). Now we can rewrite the inequality as

\[
\sum_{i \in m} (q + r_i) + \sum_{i \in n} (-q + r_i) > 0. \tag{3.11}
\]

The \( r_i \) are residues, which indicate how much each term differs from \( q \). Note that for all elements of \( m \), \( r_i \) is negative, and positive for elements of \( n \). After dividing by \( q \):

\[
\sum_{i \in m} \frac{1}{q} - \sum_{i \in n} \frac{1}{q} + \sum_{i} \frac{r_i}{q} > 0. \tag{3.12}
\]

If the sum of the residues (the third term) is 0, we can see that the process is equal to majority voting on the supported states.

If we fill in the example factors from above, we get that \( q = |\log 0.15 - \log 0.85| = 1.73 \), the residues are \( \{0, -1.12, 1.12, 1.33, 1.33, 0\} \), their sum is 2.66, and divided by \( q \), the residue term equals 1.53. The final inequality becomes

\[
-1 + 1 - 1 - 1 + 1 + 1.53 > 0. \tag{3.13}
\]

This inequality is not true, and state \( h_1 \) was indeed not the most probable state. Removing the 1.53 term would not change this.

We can now say that the factors are sufficiently balanced, if ignoring the residue term will satisfy the same inequality.

**Definition 3** A set of factors containing weights for two states \( a \) and \( b \) are balanced, if the following is true:

\[
\begin{align*}
\text{if } |m| - |n| > 0 \text{ then } & -\sum_{i} \frac{r_i}{q} < |m| - |n| \tag{3.14} \\
\text{if } |m| - |n| < 0 \text{ then } & -\sum_{i} \frac{r_i}{q} > |m| - |n| \tag{3.15}
\end{align*}
\]

where \(|·|\) denotes the size of a set, and the terms are defined as in Equation (3.12).

In the case that the residue term can be ignored, majority voting on the factor supports gives the state with the greatest probability. From the above equation, we can see that the imbalance offsets the voting. If one of the states wins the votes with a large margin, much imbalance is allowed. Note that imbalance is only a problem if it supports the losing state.

In the case of multi-state variables, the state \( a \) with the highest probability satisfies inequality (3.9) with respect to each other state \( b \). This inequality for each pair of states can be rewritten as in (3.12). In order for majority voting to work, \( a \) has to win the vote against every state \( b \). Hence, each pair of states \( a \)
3.3 Accuracy

3.3.1 Estimation Accuracy

Let $h^*$ denote the true state of variable $H$ in a particular case. We estimate the state of $H$ according to (3.3). We say that a state estimation is accurate if a variable is estimated as its true state, i.e. iff

$$h^* = \arg \max_{h_i} p(H = h_i, \mathcal{E}).$$  \hfill (3.16)

If we plug Definition 2 into (3.16) and use the view from the previous section, we can extend state estimation accuracy to fragment accuracy. Here we start making a distinction between the correctness of a factor in a particular case, and the correctness of its associated fragment in general. We use the terms factor and fragment differently on purpose. A fragment is a static element of a model, whereas the value of a factor depends on the particular evidence about variables in the fragment.

The correctness of a factor will be called its adequacy. A factor is adequate if it supports the true state of a variable.

**Definition 4** A factor is called adequate in a particular case, iff

$$h^* = h_{\phi_i} = \arg \max_{h_j} \phi_i(h_j).$$  \hfill (3.17)

We will use accuracy to indicate the correctness of a fragment. A fragment can be considered accurate if the conditional probabilities associated with its nodes are close to the true conditional probabilities. The problem facing this statement is obvious; assuming that we can never exactly get the parameters correct, how close to the true distribution is considered sufficiently close?

We want factors to be adequate. Therefore, under the voting perspective, the parameters of a fragment also can be considered accurate if they are most likely to cause an adequate factor support. It is known that the optimal parameter values are those that exactly correspond to the true distributions. Hence, adequate factor supports will be most likely to occur, if we could set the parameters equal to the true distributions. However, we are only interested in getting the correct supports, which are based on coarse inequalities. Therefore, we only need to set
the parameters such that they produce the same supports as the true distributions would. This will be made more explicit in the next section.

### 3.3.2 Fragment Accuracy

The next step is to show how parameters cause certain factor supports. The smallest possible fragment consists of only one node, and its associated CPT. However, note that any fragment can be represented by a single CPT, through multiplication of its CPTs and marginalization. Consider a small network of two nodes: $A$ and its child $B$. For a given instantiation of $B$, say $b$, the factor weights are given by the vector $p(b|A)$, a row in the CPT. The supported state is the index of the maximum value in the row. As long as that entry is the maximum, it stays the supported state. As explained above, we want to set the parameters such that they are equivalent to the true parameters, with respect to the factor supports.

Let $\tilde{p}$ denote the true parameter values. $a_{max}(b_i)$ denotes the supported state of $A$, if $B = b_i$ is evidence, under the true parameters:

$$a_{max}(b_i) = \arg\max_{a_j} \tilde{p}(b_i|a_j).$$

We want the supports of the model to be equal to the supports given the true distribution. Therefore, for any possible evidence $b_i$ about $B$, the actual CPT parameters in the Bayesian network should be such that this $a_{max}(b_i)$ is also the supported state:

$$\forall b_i \in B, \forall a_j \neq a_{max}(b_i) : \quad p(b_i|a_j) > p(b_i|a_{max}(b_i)).$$

This equation can be extended to fragments of any size. If $H$ is the fragment root node and $E$ is evidence about nodes in the fragment, then we get that

$$\forall E, \forall h_j \neq h_{max}(E) : \quad p(E|h_{max}(E)) > p(E|h_j).$$

Summarizing, we can define the following relation between two CPTs:

**Definition 5** Two CPTs $p_1(B|A)$ and $p_2(B|A)$ are called support-equivalent if, for all possible evidence, the supported states are the same:

$$\forall b_i \in B : \quad \arg\max_{a_m} p_1(b_i|a_m) = \arg\max_{a_n} p_2(b_i|a_n).$$

Hence, we get the following definition about the accuracy of a fragment:

**Definition 6** A fragment is called accurate if the value of its corresponding CPT is support-equivalent to the true CPT, i.e. the true distribution over the associated variables.
### 3.3 Accuracy

<table>
<thead>
<tr>
<th></th>
<th>( a_1 )</th>
<th>( a_2 )</th>
<th>( a_3 )</th>
<th></th>
<th>( a_1 )</th>
<th>( a_2 )</th>
<th>( a_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_1 )</td>
<td>0.7</td>
<td>0.4</td>
<td>0.2</td>
<td>( b_1 )</td>
<td>0.4</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>( b_2 )</td>
<td>0.2</td>
<td>0.3</td>
<td>0.6</td>
<td>( b_2 )</td>
<td>0.5</td>
<td>0.3</td>
<td>0.8</td>
</tr>
<tr>
<td>( b_3 )</td>
<td>0.1 ( \text{bold} )</td>
<td>0.3</td>
<td>0.2</td>
<td>( b_3 )</td>
<td>0.1</td>
<td>0.5 ( \text{bold} )</td>
<td>0.1</td>
</tr>
</tbody>
</table>

(a) (b)

Figure 3.5: Example conditional probability tables. The two CPTs are support-equivalent.

For the small two-node network, the inequality relation leaves a large interval of possible values for all parameters in the CPT. This also explains the robustness of the diagnostic element in Section 3.2; any value of \( p(e|h) \) between 0.3 and 1 maintains the CPT’s support-equivalence to the true distribution.

To illustrate this, consider the CPT in Figure 3.5(a). Suppose that these values correspond to the true distribution. The maximum element in each row is shown in bold. One can see that for example the entry \( p(b_2|a_3) \) will be the maximum in its row as long as its value lies in the interval between 0.3 and 1. For example, the CPT in Figure 3.5(b) is support-equivalent.

In the explanation above, we considered diagnostic fragments, whose factors are given by \( p(E|H) \). However, the definitions also hold for the predictive fragment, whose factor is given by \( p(H|E) \). The derivation is similar, but with the rows and columns of the CPT switched.

These simple inequality relations imply a relatively large interval of values. Therefore, we assume that it is fairly easy for model builders or learning algorithms to recognize the support-equivalence relations. For example, consider a gas detection network, containing fragments that map human observations to the presence of a toxic gas. It should be easy for domain experts to indicate whether a certain observation (such as a foul smell) increases or decreases the likelihood of the presence of a gas. Or, in the case of multiple possible gasses, which type of gas is most likely.

Summarizing, if a model contains many conditionally independent fragments and is ‘balanced’, then we can use a coarse view of the inference process, namely voting on factor supports. An important consequence of this perspective is that, to achieve optimal factor supports, we are allowed to choose suboptimal CPT parameters as long as simple inequalities (3.20) are satisfied.

This observation is significant in the context of model grounding. It tells us that getting the CPTs support-equivalent to the true distribution is crucial to the accuracy of Bayesian networks in a state estimation context. This leaves a lot of room for parameter changes. Or, to put it differently, precise parameter values are not needed as long as the support-equivalence is maintained. This implies that in certain situations, we can ignore the precision. This is the basis for most of the algorithms presented in this thesis. The remainder of this chapter presents
a coarse belief propagation algorithm that propagates only the factor supports.

3.4 Belief Propagation

In this section, an algorithm is presented that performs a variant of belief propagation that is based on voting. It is based on the sum-product algorithm on factor graphs, as described in Section 2.2.3. This voting variant is exact if certain conditions are satisfied.

3.4.1 Belief Propagation through Voting

For state estimation, we are interested in finding the state of a variable with the highest posterior probability. Using the standard equation (2.17), this state is given by

$$\hat{X} = \arg \max_X p(X) = \arg \max_X \prod_{s \in \text{ne}(X)} \mu_{f_s \rightarrow X}(X),$$

(3.22)

where $\text{ne}(X) = \{f_1, \ldots, f_S\}$ are the neighbors of $X$ in the factor graph. Evaluating this equation requires the element-wise multiplication of vector messages. Here we can plug in the coarse inference perspective from Section 3.2.2. Namely, the messages $\mu_{f_s \rightarrow X}(X)$ are equivalent to the fragment factors, as in (3.8). Under the condition of balance as in Definition 3, we can perform majority voting on the maximum elements of each message.

We will therefore replace (3.22) with majority voting on the indices of the largest element in each message:

$$\hat{X} = \text{vote}(\arg \max_X \mu_{f_1 \rightarrow X}(X), \ldots, \arg \max_X \mu_{f_S \rightarrow X}(X))$$

(3.23)

Each $\arg \max_X \mu_{f_i \rightarrow X}(X)$ is a vote. The maximization can be performed locally at the neighboring nodes, before the messages are sent. These voting messages are denoted by $\hat{\mu}(X) \equiv \arg \max_X \mu(X)$. Hence, we get

$$\hat{X} = \text{vote}(\hat{\mu}_{f_1 \rightarrow X}(X), \ldots, \hat{\mu}_{f_S \rightarrow X}(X)).$$

(3.24)

Factor to Variable Node Message

Using the standard equation (2.16), the factor to variable messages in the voting algorithm are given by

$$\hat{\mu}_{f_s \rightarrow X}(X) = \arg \max_X \sum \cdots \sum_{X_M} f_s(X, X_1, \ldots, X_M) \prod_{m \in \text{ne}(f_s) \setminus X} \mu_{X_m \rightarrow f_s}(X_m),$$

(3.25)
where \( \text{ne}(f_s) = \{X, X_1, \ldots, X_M\} \) are the neighbors of \( f_s \) (as shown in Figure 2.6(b)). We can rewrite the right-hand side of this equation, by defining a factor \( g(X, X_1) \) as follows:

\[
g(X, X_1) = \sum_{X_2} \cdots \sum_{X_M} f_s(X, X_1, X_2, \ldots, X_M) \prod_{m \in \text{ne}(f_s) \setminus \{X, X_1\}} \mu_{X_m \rightarrow f_s}(X_m). \tag{3.26}
\]

Plugging this into (3.25), we get

\[
\hat{\mu}_{f_s \rightarrow X}(X) = \arg \max_X \sum_{X_1} g(X, X_1) \mu_{X_1 \rightarrow f_s}(X_1). \tag{3.27}
\]

Let the maximum element of the message from \( X_1 \) be denoted by \( x_{max} = \arg \max_{X_1} \mu_{X_1 \rightarrow f_s}(X_1) \). We can now rewrite the last equation, by taking this state \( x_{max} \) out of the sum and dividing by its value in the message:

\[
\hat{\mu}_{f_s \rightarrow X}(X) = \arg \max_X \left[ g(X, x_{max}) + \sum_{x_i \in X_1 \setminus x_{max}} g(X, x_i) \frac{\mu_{X_1 \rightarrow f_s}(x_i)}{\mu_{X_1 \rightarrow f_s}(x_{max})} \right]. \tag{3.28}
\]

The idea is to simplify this computation by ignoring the sum term. This is allowed if it does not change the outcome of the maximization. Let \( a = \arg \max_X g(X, x_{max}) \). We can safely ignore the sum term in (3.28), if for every state \( b \) of \( X \) not equal to \( a \), it holds that

\[
g(a, x_{max}) - g(b, x_{max}) > \sum_{x_i \in X_1 \setminus x_{max}} [g(b, x_i) - g(a, x_i)] s_i \tag{3.29}
\]

These conditions help to satisfy this inequality:

- The ratio between the maximum element of a message \( \mu_{X_1 \rightarrow f_s}(x_{max}) \) and the other elements \( \mu_{X_1 \rightarrow f_s}(x_i) \) should be large. This will cause \( s_i \) to be low, decreasing the right-hand side of (3.29).

- The difference among the values in the factor \( g \) should be large. This will cause \( g(a, x_{max}) \) of to be much greater than any \( g(b, x_{max}) \). The left-hand side of (3.29) will then be high.

After ignoring the sum term in (3.28), the maximization over \( X \) of \( g(X, x_{max}) \) remains. This can be done in exactly the same way as above. Namely, we can define a new factor \( h(X, x_{max}, X_2) \):

\[
h(X, x_{max}, X_2) = \sum_{X_3} \cdots \sum_{X_M} f_s(X, x_{max}, X_2, X_3, \ldots, X_M) \prod_{m \in \text{ne}(f_s) \setminus \{X, X_1, X_2\}} \mu_{X_m \rightarrow f_s}(X_m), \tag{3.30}
\]
Accuracy and Belief Propagation

such that

\[ \hat{\mu}_{f_s \rightarrow X}(X) = \arg \max_X g(X, x_{\text{max}}) = \arg \max_X \sum_{X_2} h(X, x_{\text{max}}, X_2) \mu_{X_2 \rightarrow f_s}(X_2). \]  

(3.31)

Again, we split up the sum similar to (3.28), and continue as above, by ignoring a term. In this way we recursively treat each variable \( X_1, \ldots, X_M \) in turn, each time instantiating the variable to its maximum entry in its message (an example is given below).

Thus we can rewrite Equation (3.25) for the voting message from factor \( f_s \) to variable \( X \) as:

\[ \hat{\mu}_{f_s \rightarrow X}(X) = \arg \max_X f_s(X, \arg \max_{X_1} \mu_{X_1 \rightarrow f_s}(X_1), \ldots, \arg \max_{X_M} \mu_{X_M \rightarrow f_s}(X_M)). \]  

(3.32)

Again, the maximization over a message can be performed at the node that sent the message. Using the same \( \hat{\mu} \) notation as above,

\[ \hat{\mu}_{f_s \rightarrow X}(X) = \arg \max_X f_s(X, \hat{\mu}_{X_1 \rightarrow f_s}(X_1), \ldots, \hat{\mu}_{X_M \rightarrow f_s}(X_M)). \]  

(3.33)

Note that the more neighbors a factor node in a factor graph has, the more conditions have to be satisfied. The number of neighbor nodes of a factor node in a factor graph is given by the number of parents of a node in the Bayesian network.

**Factor to Variable Node Message: Examples**

As an example of the above derivation, consider a factor \( f_s \) over variables \( A, B, C, \) and \( D \), given by the table in Figure 3.6(b). We want to compute the voting message to \( A \), given the incoming messages

\[
\begin{align*}
\mu_{B \rightarrow f_s}(b) &= \begin{pmatrix} 0.8 & 0.2 \end{pmatrix}, \\
\mu_{C \rightarrow f_s}(c) &= \begin{pmatrix} 0.3 & 0.7 \end{pmatrix}, \\
\mu_{D \rightarrow f_s}(d) &= \begin{pmatrix} 0.1 & 0.9 \end{pmatrix}.
\end{align*}
\]  

(3.34)

The voting message to \( A \) is given by (3.25):

\[ \hat{\mu}_{f_s \rightarrow A}(A) = \arg \max_A \sum_B \cdots \sum_D f_s(A, B, C, D) \prod_{X \in \{B, C, D\}} \mu_{X \rightarrow f_s}(X). \]  

(3.35)

Factor \( g(A, B) \) is computed with Equation (3.26):

\[ g(A, B) = \begin{pmatrix} 0.55 & 0.57 \\ 0.39 & 0.53 \end{pmatrix}, \]  

(3.36)

where each row corresponds to a state of \( A \).
3.4 Belief Propagation

$b_{\text{max}} = b$, the maximum element of the message from $B$. If we fill in Equation (3.28), we get

$$\hat{\mu}_{f_s\rightarrow A}(A) = \arg\max_A \left[ \begin{pmatrix} 0.55 \\ 0.39 \end{pmatrix} + \begin{pmatrix} 0.57 \\ 0.53 \end{pmatrix} \frac{0.2}{0.8} \right].$$

(3.37)

In this case, it is clear that ignoring the second term of the sum does not change the result of the maximization. One can see that this second term could be ignored for many different values of the second vector $(0.57, 0.53)$, because it gets multiplied by 0.25.

We now compute a factor $h(A, b_{\text{max}}, C)$ given by (3.30):

$$h(A, b_{\text{max}}, C) = \begin{pmatrix} 0.59 & 0.53 \\ 0.44 & 0.37 \end{pmatrix}.$$  

(3.38)

Now, $c_{\text{max}} = \bar{c}$. Filling in (3.31),

$$\arg\max_A g(A, b_{\text{max}}) = \arg\max_A \left[ \begin{pmatrix} 0.53 \\ 0.37 \end{pmatrix} + \begin{pmatrix} 0.59 \\ 0.44 \end{pmatrix} \frac{0.3}{0.7} \right].$$

(3.39)

Again, we can ignore the second term of the sum. The last factor we need is $f_s(A, b_{\text{max}}, c_{\text{max}}, D)$:

$$f_s(A, b_{\text{max}}, c_{\text{max}}, D) = \begin{pmatrix} 0.72 & 0.51 \\ 0.26 & 0.38 \end{pmatrix}.$$  

(3.40)

$d_{\text{max}} = \bar{d}$, so we can write

$$\arg\max_A h(A, b_{\text{max}}, c_{\text{max}}) = \arg\max_A \left[ \begin{pmatrix} 0.51 \\ 0.38 \end{pmatrix} + \begin{pmatrix} 0.72 \\ 0.26 \end{pmatrix} \frac{0.1}{0.9} \right].$$

(3.41)

We can again ignore the term, and thus we get that

$$\hat{\mu}_{f_s\rightarrow A}(A) = \arg\max_A g(A, b_{\text{max}}) = \arg\max_A h(A, b_{\text{max}}, c_{\text{max}})$$

(3.42)

$$= \arg\max_A f_s(A, b_{\text{max}}, c_{\text{max}}, d_{\text{max}}) = \arg\max_A f_s(A, b, \bar{c}, \bar{d}).$$

We see that in this case, the voting message to $A$ is given by maximizing over the entries of the factor table after instantiating the other variables to the maximum entry in their message.

An alternative way of illustrating how this works in practice goes as follows: Consider that the factor can be represented by a multi-dimensional matrix. Through multiplication with each incoming message, parts of the matrix get multiplied with high message values, while other parts get multiplied with low values. Thus, the parts of the matrix that got multiplied with the maximum of
each message, get relatively high values compared to the rest of the matrix. After all multiplications, we sum over all dimensions, except for the one corresponding to $X$. The entries in the matrix that got multiplied with the largest numbers are likely to have a larger impact on this summation. These entries correspond to the combination of the message votes. Therefore, we could approximate the evaluation of (3.25) by ignoring the states of each variable that did not correspond to the supported state of a message. Then the summation becomes unnecessary.

A numerical example of this process is shown in Figure 3.6. The local graph is shown in (a). The table in (b) shows factor $f_s$, and the messages are: $\mu_{B\rightarrow f_s}(b, \bar{b}) = (0.8, 0.2)$, $\mu_{C\rightarrow f_s}(c, \bar{c}) = (0.3, 0.7)$ and $\mu_{D\rightarrow f_s}(d, \bar{d}) = (0.1, 0.9)$. The values of the factor function after multiplication with the messages from $B$, $C$, and $D$ are shown in (c), (d) and (e) respectively. Next, the factor in (e) is summed over all variables, except $A$. In this case, that means that all values in the left half (which correspond to $A = a$) are summed, and all values in the right half (which correspond to $A = \bar{a}$). After this summation, the values are: $f_s(a) = 0.55$ and $f_s(\bar{a}) = 0.42$. The maximum of these two is $a$. As one can see, the bold entries in (e) dominate the summation. These are the entries of the table that, with each multiplication, got multiplied with the maximum value in the message. Since they dominate the summation, we can approximate (3.25), by taking the maximum over only these two entries. This would indeed in this case also give us the solution $\hat{\mu}_{f_s\rightarrow A}(A) = a$.

**Variable to Factor Node Message**

The voting messages from a variable node $X$ to a factor node $f_s$ is given by

$$\hat{\mu}_{X\rightarrow f_s}(X) = \arg \max_X \prod_{t \in \text{ne}(X) \setminus f_s} \mu_{f_t\rightarrow X}(X),$$

(3.43)

where $\text{ne}(X) = \{f_s, f_1, \ldots, f_S\}$ (as shown in Figure 2.6(a)). We can rewrite this equation in the same way as we did for (3.24), replacing the maximum over a product of messages by voting on the message supports. This leads to the last update equation:

$$\hat{\mu}_{X\rightarrow f_s}(X) = \text{vote}(\arg \max_X \mu_{f_1\rightarrow X}(X), \ldots, \arg \max_X \mu_{f_S\rightarrow X}(X))$$

(3.44)

Again, the maximization can be performed before sending the messages, and thus the message to a neighboring factor node contains the state of $X$ that got the most votes:

$$\hat{\mu}_{X\rightarrow f_s}(X) = \text{vote}(\hat{\mu}_{f_1\rightarrow X}(X), \ldots, \hat{\mu}_{f_S\rightarrow X}(X)).$$

(3.45)

How to compute the voting messages $\hat{\mu}_{f_i\rightarrow X}(X)$ is shown in (3.33).
Figure 3.6: Example process of evaluating (3.25) for variable $A$, given messages from $B$, $C$ and $D$. 
Further Details

Ties can occur when multiple states get an equal number of votes. One solution to breaking these ties, is to simply send multiple messages, one for each winner. This can be incorporated seamlessly into the variable to factor node update equations; it would simply amount to more votes. For the factor to variable node equation, extra matrix entries will have to be considered corresponding to the extra messages. One can choose to still take the maximum, but over more entries, or first sum over the dimensions for which multiple messages arrived.

In the same manner the initial messages from non-evidence variable nodes can be handled by sending multiple messages, each voting for a different state of the variable. Initial messages from factor nodes are given by the approach described above, assuming that the variable nodes have sent their initial messages first.

Summarizing, the voting variant of sum-product amounts to replacing (2.17), (2.16) and (2.15) by their approximations (3.24), (3.33) and (3.45), respectively.

The message passing scheme of the voting algorithm is similar to that of the sum-product algorithm. This implies that if a factor graph contains loops, a parallel message sending scheme can be used (Section 2.2.3). In this case, all nodes are processed in parallel, until a certain stopping criterion is met. In our implementation we used message convergence (the votes do not change), or, if the messages do not convergence, a maximum number of iterations.

Sometimes, one is interested in the joint probability distribution over all variables from a node family in a Bayesian network. The family of a node is the set consisting of a node and all its parents. In a factor graph, a node family corresponds to all variable node neighbors of the factor node.

With the regular sum-product algorithm we can obtain this joint distribution by multiplying a factor with the messages from all its variable node neighbors. In the voting case, we would be interested in the combined state of variables with the greatest joint probability. This combined state is given by the combination of all incoming voting messages to a factor node.

Algorithm Properties

As mentioned above, the equations for computing the messages in the factor graph are exact (computing the same state as exact inference) under certain conditions. In the case of the variable to factor node messages, the voting result is indeed the most probable state (the one chosen by exact inference), if the messages are balanced conform Definition 3. The more states a variable has, the more conditions of balance need to be satisfied. Therefore, it can be expected that the probability that the algorithm is exact decreases with the number of variable states.

Factor to variable node messages are exact if inequality (3.29) is satisfied. It needs to be satisfied for each state $b$ of $X$, and for each factor $(g, h, \ldots)$ that is
defined (one for each neighbor of the factor node). Therefore, it can be expected that the probability that the algorithm is exact decreases with the number of variable states and the number of neighbors of factor nodes. The number of factor node neighbors in a factor graph depends on the number of parents of a node in a Bayesian network.

However, the voting scheme used in the variable to factor node messages has another important property. If the probability that a vote supports the true state is greater than 0.5, then increasing the number of votes increases the probability that the true state wins the voting. This probability converges to 1 [LS97]. In Chapter 5 we explain under which conditions we can assume that the probability of receiving a vote for the true state is greater than 0.5. The number of votes depends on the number of neighbors of a variable node in a factor graph. The number of neighbors depends on the number of children of a node in a Bayesian network. Hence, the algorithm is more likely to find the true state of a variable, if the network contains many independent network fragments. This might compensate for having many variable states or many node parents, which can have a negative influence.

### 3.4.2 Experiments

In these experiments we investigate the quality of the inference algorithm, with respect to the properties that we outlined above. First, we investigate each of the properties separately on artificial networks in which we can control the characteristics. Then, we apply the algorithm on several real networks, and observe the algorithm’s quality.

The quality of the voting algorithm can be defined in two different ways. One is similarity, which counts the percentage of cases in which the voting algorithm outputs the same state estimation as exact inference does. The other is accuracy, which counts the percentage of cases in which the voting algorithm outputs the true state. Let \( \mathcal{D} \) be a set of data cases, \( H \) the variable of interest, \( \hat{H} \) the output of the voting algorithm, and \( h^* \) the true state of \( H \). Then the quality measures are defined as

\[
similarity = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \begin{cases} 1 & \text{if } \hat{H} = \arg \max_H p(H|x) \\ 0 & \text{otherwise} \end{cases} \quad (3.46)
\]

and

\[
accuracy = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \begin{cases} 1 & \text{if } \hat{H} = h^* \\ 0 & \text{otherwise} \end{cases} \quad (3.47)
\]

where \(|\mathcal{D}|\) is the size of the data set.

First, we investigate the influence of the number of variable states. It can be expected from the algorithm properties that an increase of the number of states lowers the similarity measure. We test this on three artificial Bayesian networks.
Table 3.1: Similarity between state estimation based on the voting algorithm and exact inference. The number variable states is varied. The CPTs are initialized randomly, so they represent a worst case scenario.

<table>
<thead>
<tr>
<th>network</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>alarm</td>
<td>0.84</td>
<td>0.74</td>
<td>0.69</td>
<td>0.64</td>
<td>0.60</td>
</tr>
<tr>
<td>insurance</td>
<td>0.84</td>
<td>0.71</td>
<td>0.62</td>
<td>0.56</td>
<td>0.51</td>
</tr>
<tr>
<td>tree</td>
<td>0.88</td>
<td>0.80</td>
<td>0.72</td>
<td>0.65</td>
<td>0.62</td>
</tr>
</tbody>
</table>

One is a tree, with 3 layers and a branching factor of 5. The other two use the network structure from a real network, namely Alarm and Insurance. The number of variable states is varied from 2 till 6, and the conditional probabilities are chosen randomly. We let each network generate a data set. From the data set, we instantiate a number of variables in the network. Then we perform state estimation on different variables, using both exact inference and the voting algorithm. We record the similarity, and average this over 20 random initializations of the conditional probabilities.

The result is shown in Table 3.1. As expected, the similarity decreases as the number of states increases. The differences in similarity score between the three networks can be attributed to the difference in graph structure. The general tendency is the same, however. This experiment indicates that the voting algorithm can be similar to exact inference for Bayesian networks with a low number of variable states. Note that the CPTs were randomized, so they represent a worst case scenario with respect to the balance condition.

Next, we investigate the influence of the number of child nodes. As explained in the previous section, an increase of the number of child nodes increases the accuracy score, if the probability that a vote supports the true state is at least 0.5. We use a tree network with 3 layers, of which we vary the branching factor from 2 to 5. The number of states is fixed to three. We set the CPTs to random values, however such that the probability of getting a vote for the true state is greater than 0.5. Section 5.2 describes conditions for CPTs that guarantee this. A data set is generated from this network. The leaf nodes are instantiated to their value in the data case, and state estimation is performed on the root node. For this experiment, we record both the similarity and accuracy measure.

The result is shown in Table 3.2. The accuracy increases with the number of children. However, the number of children has almost no influence on the similarity measure.

Thirdly, we investigate the influence of the number of parents of nodes in the Bayesian network. From the properties outlined in the previous section, it can be expected that an increase of the number of parent nodes decreases the similarity
measure. To test this, we can use a ‘converging’ tree network, which is a tree network with the direction of all arcs reversed. We use 3 layers, and vary the number of parent nodes from 2 to 5. The number of states is fixed to two. The experiment is performed similarly to the previous experiment, except that we instantiate the root nodes, and perform state estimation on the leaf node. For this experiment, we record both the similarity and accuracy measure.

The result is shown in Table 3.3. We can see that an increase of the number of parents decreases the similarity to exact inference, as expected. The influence on the accuracy is low. Accurate state estimation on this type of network seems difficult in general.

Lastly, we test the voting algorithm on several real networks. The experiment was conducted similar to the ones above, by randomly inserting evidence into the network and doing state estimation on several randomly chosen variables. This was repeated 1000 times, and the similarity and accuracy were recorded. Furthermore, we computed some of the characteristics of the networks, such as the average number of variables states, and the average number of parents (in-degree) and children (out-degree). Since the average in-degree and out-degree of a graph are always equal, this would not be very informative. Therefore, we only consider non-root nodes for the average in-degree, and only non-leaf nodes for the average out-degree.

Table 3.4 shows the results. We can see that in general, the number of variable states has the most significant influence on the similarity and accuracy. The only value in the table that does not follow this tendency, is the similarity score of the
Table 3.4: Similarity and accuracy of state estimation based on the voting algorithm and exact inference.

<table>
<thead>
<tr>
<th>network</th>
<th>in-degree</th>
<th>out-degree</th>
<th>states</th>
<th>accuracy</th>
<th>similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>alarm</td>
<td>1.84</td>
<td>1.77</td>
<td>2.8</td>
<td>0.89</td>
<td>0.95</td>
</tr>
<tr>
<td>barley</td>
<td>2.21</td>
<td>2.10</td>
<td>8.8</td>
<td>0.57</td>
<td>0.68</td>
</tr>
<tr>
<td>hailfinder</td>
<td>1.69</td>
<td>1.53</td>
<td>4.0</td>
<td>0.60</td>
<td>0.79</td>
</tr>
<tr>
<td>insurance</td>
<td>2.08</td>
<td>2.48</td>
<td>3.3</td>
<td>0.73</td>
<td>0.80</td>
</tr>
<tr>
<td>car-starts</td>
<td>2.13</td>
<td>1.21</td>
<td>2.1</td>
<td>0.97</td>
<td>0.99</td>
</tr>
<tr>
<td>water</td>
<td>2.75</td>
<td>2.75</td>
<td>3.6</td>
<td>0.64</td>
<td>0.68</td>
</tr>
</tbody>
</table>

water-network. However, its low similarity can be explained by its high average in-degree.

These experiments show that the voting algorithm can give output that is very similar to exact inference with respect to state estimation, under certain conditions. Namely, if the average number of variable states is low (around 2 or 3). A higher number of states can be compensated by a high out-degree. In trees, with a high out-degree, the voting algorithm can be very accurate.

The experimental results of the voting algorithm also indirectly show the robustness properties of Bayesian networks. We see that for certain types of networks (e.g. trees), the voting algorithm has similar output to exact inference. Suppose that in such type of network we change one of the parameters’ value. If the votes do not change, the voting algorithm output does not change. Since the voting output (for these types of networks) was similar to exact inference, this implies that the state estimation of exact inference also does not change. In other words, these types of Bayesian networks are robust against parameter changes, as long as these changes do not alter the votes (i.e. the supports). This observation supports the validity of our perspective on model accuracy, as described in Definition 6.

### 3.4.3 Computational Complexity

The computational complexity of the voting algorithm is significantly lower than that of the original sum-product algorithm. Let $n$ be the number of neighbors of a node, and $k$ be the number of variable states. The computational complexity of the original equation for variable to factor node (2.15) is $O(nk)$, as it requires multiplying $n$ messages with $k$ elements. The corresponding equation of the voting algorithm (3.45) has complexity $O(n)$, as it requires voting over $n$ messages.

The voting approach makes the computation of factor to variable messages significantly faster. The original equation (2.16) has complexity $O(nk^n)$, as the number equations are linear with the size of the factor. The voting algorithm’s
3.5 Discussion

### 3.5.1 Related Work

The robustness of Bayesian networks to parameter changes has been studied for some time. The earliest analyses were of an empirical nature. Essentially, they added noise to parameter values and observed the accuracy of the outcome of certain queries [PHP+96]. The results showed a strong robustness against small parameter changes, as long as the added noise was symmetrical, and the expected accuracy was used as the measure. [KW01] later argued that this measure was not the most informative, and that it might give different results when looking for changes in the state estimation outcome.

Later, this was supported by theoretical work, looking from the perspective
of sensitivity of the query outcome to the parameter changes [CD02]. Robustness
was measured by whether the query probability stayed in a small interval. They
observed the same robustness, as long as the parameter were not near their ex-
treme values, 0 or 1. Several other authors looked at the problem in the context
of naive Bayesian networks, which are much less complex models, and thus easier
to analyse [DP96, CD03].

Another approach comes from the direction of sensitivity analyses [CvdG02].
A sensitivity function describes a probability of interest in terms of a parameter
under investigation. The derivative of this function can be used to detect param-
eters that have a large influence on the probability of interest. In [vdGR01a], the
concept of admissible deviation was introduced. Using the sensitivity functions,
it showed how to compute the interval of values for a specific parameter under
which the state estimation outcome does not change (the admissible deviation).
This interval depended on the specific evidence inserted into the network. How-
ever, in [RvdG04] this dependency is dropped, by using bounds on the sensitivity
function and thus on the admissible deviation.

This work comes closest to our view. The difference is that each interval is valid
for only a single parameter, and for a given probability of interest. Combinations
of parameter changes, for example within a single CPT, are not computed. In our
view, the relations between CPT entries, and between different network fragments,
are crucial to the state estimation outcome.

Several researchers have addressed the parameter imprecision problem by using
convex sets to capture uncertainties of the CPT parameters [FZ98, Coz97, Tes92].
However, representations based on convex sets require complex approaches to
belief propagation, such as linear programming. Furthermore, they can result in
not very informative distributions, as the convex set can diverge through belief
propagation.

Moreover, the voting algorithm introduced seems to have some similarity to
inference in qualitative probabilistic networks (QPN) [Wel90, DH93]. Namely,
both approaches use a more abstract view on the conditional probability distri-
butions. Similarly to the QPN approach, we assume that designers or machine
learning processes can identify a few coarse grained relations between the true
distributions.

However, due to a very coarse representation of distributions, the QPN ap-
proach becomes inconclusive in cases where different network fragments introduce
conflicting updates of the distribution over a variable [RvdGP02]. In stochastic
domains, this is quite common and the chance of having conflicting influences
grows with the number of network fragments. In order to be able to cope with
such problems the basic QPN principles were extended by sophisticated repre-
sentation and updating algorithms [Par95, RPvdG01]. However, it seems that
implementation of such approaches is relatively complex and results might be
difficult to interpret. For example, the algorithm described in [RvdGP02] con-
siders the evidence entering order and ignores inter-causal influences, while the
3.5 Discussion

The approach in [Par95] considers relative and absolute magnitudes of influence and introduces complex operations. On the other hand, the propagation algorithm introduced in this chapter can cope with conflicting evidence in a very robust way since it is based on different assumptions. Contrary to QPNs, we do not assume any preprocessing of the BNs in order to obtain a coarser grained representation of the distributions between the different events of interest. In QPNs relations between the true distributions are encoded through different types of influence and combined through special operators.

Furthermore, the presented robustness perspective is complementary to the common approaches to fine grained sensitivity analysis [CD04, CPOH99, CGH97]. Sensitivity analysis computes the gradient on a posterior distribution with respect to model parameters. It can tell a model builder which parameters require extra attention. Contrary to these approaches, we take into account the relations between true distributions and the modeling parameters and do not consider the entire network topology along with the instantiations. Furthermore, we quantify the extent to which parameter values are allowed to change.

3.5.2 Conclusion

The first contribution of this chapter is the introduction of a coarse perspective on the inference process, in the context of state estimation. We considered network fragments as independent ‘experts’ that each support a certain state of the variable of interest. Under certain conditions of balance, the most probable state can be found through majority voting on the supported states.

The consequence of this perspective is that it allows us to quantify the robustness to changes in parameter values. Namely, an accurate model is one that produces correct votes. These votes depend on inequality relations between conditional probabilities. We can identify intervals of parameter values which satisfy these inequalities, and thus result in adequate votes. These intervals show that Bayesian networks can be very robust to certain changes in parameter values, and they relax the problem of obtaining appropriate parameters [DvdG00].

The second contribution is a voting-based belief propagation algorithm inspired on the voting perspective. Large intervals of parameter values can be adequate in a state estimation setting, under the voting perspective. Therefore the voting algorithm ignores the precise values. It is presented as a variant of the sum-product algorithm in factor graphs. The algorithm finds the most probable state of each variable, and is exact under certain conditions.

From these conditions, we can predict for which type of Bayesian networks the algorithm works well. Namely, in networks where variables have a around 2 to 4 states, and a high number (at least 4 or 5) of child nodes. The experiments confirm these properties. The advantages of the algorithm are its very low computational complexity and the ability to work with very coarse and uncertain networks. Since the precise details are ignored, the parameters do not need to be set to
precise values. Furthermore, the algorithm has certain advantages over related approaches to belief propagation with uncertain and imprecise Bayesian networks, as explained in the previous section.

Possible applications of the algorithm are situations where speed is important. For example, most learning methods require many repeated belief propagations, which can make them slow when learning from large data sets. The algorithm can also be applied in situations where we are very uncertain about the correctness of the parameters, for example from lack of adequate training data. This is likely to occur if we attempt to model very rare events in a problem domain.
This chapter focuses on learning the parameters of a Bayesian network from incomplete data. Iterative search methods such as Expectation-Maximization (EM) perform a local search, and therefore do not necessarily find the global maximum. To avoid getting stuck in poor local maxima, many improvements have been proposed, typically based on changing the score function or the search procedure. Based on the voting perspective from the previous chapter, we investigate how the use of a quantized parameter space can help to avoid poor local maxima, and speed up learning. At the same time, it can increase the robustness against overfitting when training on small data sets, without requiring parameter priors. We show how this approach can easily be incorporated in a general EM procedure.

4.1 Introduction

Parameter learning for Bayesian networks is the process of finding adequate parameter values, given a fixed network structure. Typically, the dependency structure can be found with the help of human domain experts. Finding precise parameter values is difficult for humans, so we often use methods to learn the parameters from a data set of representative cases. These algorithms typically attempt to maximize some score. This maximization is not trivial, since the data cases are often incomplete.

Typical scores are data likelihood and model posterior probability. In the case of a complete data set, finding the maximizing model is trivial. However, no direct solution exists for the maximum likelihood or model posterior parameter values from incomplete data. In these cases, we require a search method that iteratively improves the score of the Bayesian network. One such search algorithm is the expectation-maximization (EM) algorithm [DLR77, Lau95]. A major disadvantage of these methods is that they perform a local search, and thus find maxima
in the locality of the starting point of the search. These local maxima can be quite poor.

Approaches to overcome this disadvantage have been proposed. The most common solution is to use many restarts from random initializations, and keep the best solution that was found. Some more advanced approaches focus on changing the search procedure to allow certain non-improving steps [GL97, KGV83], and others on changing the search surface or score function to guide the search to better maxima [UN98, ENFS02]. Section 4.2 gives a more detailed description of some approaches. One downside of methods capable of finding models that fit the training data very well, is that they become more likely to overfit on small data sets. A small data set is more likely to badly represent the true distribution. Thus, if a model is learned that fits the training data very well, it might not be a good generalization of the true application domain.

In this chapter we investigate a third type of approach based on a quantization of the parameter values. In other words, we allow the parameters to take only a finite number of possible values, instead of any value from a continuous space. The intuition behind this is that by choosing an appropriate quantization of the space, we are discarding unnecessary and potentially harmful details. This simplifies the search surface and can smoothen out local maxima. The learning procedure can consist of two phases, one phase that searches in the quantized space, and a second phase that refines the solution in the full parameter space. Higher precision details are learned separately, after a good solution in the quantized space has been found. Which quantization we choose and the reasons why certain details can be smoothed out is further discussed in Section 4.2.1.

A quantized parameter space approach can have several advantages. Firstly, the number of allowed parameter values directly influences the number of local maxima, which is elaborated in see Section 4.1.1. Less local maxima implies more stable results, and less need for random restarts, speeding up the search procedure. Secondly, in a space with less possible values, reaching any maximum is likely to take less steps. Hence, convergence is likely to be reached faster. Thirdly, the quantization can facilitate to avoid very extreme parameter values, which are often unwanted. Similar to smoothing CPTs by using parameter priors, this can help prevent overfitting.

This seems to imply that the quantization approach can be effective in the case of small training data sets. In such a context, typically many local maxima are present that do not correspond to maxima in the ‘true’ space. Quantization can lower the number of local maxima and disallowing extreme parameter values. Therefore the quantized search becomes more robust against overfitting. It focuses on learning a rough model first, without committing itself to finding precise values in the first phase of learning.

Most approaches that aim to counteract overfitting, such as using priors on parameters or adding a penalization term to the score function, use problemspecific prior or external knowledge about which parameter values are (un)wanted.
Our approach does not require such knowledge. According to the perspective from the previous chapter, even a coarse or quantized model can give accurate state estimation, under certain conditions. For this we need an inference algorithm that can handle the coarse representation. Therefore, an interesting experiment is to couple quantized learning to the voting algorithm from Chapter 3. Quantized learning can produce a model that satisfies the accuracy conditions, but has precise parameter values that are not optimal. If the voting algorithm can achieve good state estimation accuracy despite this sub-optimality, this is a validation for our perspective on model grounding.

4.1.1 Rationale

To illustrate how the number of local maxima is influenced by the quantization level of the search space, we performed a small experiment. We took an X-shaped network, consisting of a central node with 2 parents and two child nodes, as shown in Figure 4.1(a). All variables had two states. The conditional probabilities were initialized to random values, and a data set was sampled from the network. The value of the central node was made hidden in all data cases.

If we had to learn the network parameters from this data set, we would be searching for the optimal values of the CPTs of the central node and its two children. This amounts to 8 parameters, as the other 8 are their complements. In the quantized space, parameters can only take values that are a multiple of a certain step size. For example, values \{0.05, 0.1, \ldots, 0.95\} for a step size of 0.05. To get an estimate of the number of local maxima under different quantization levels, we performed a large number of random-restart hill-climbs. This simple hill climbing considers all neighbors of the current parameter estimate in the quantized space and selects the one with the greatest data likelihood. We stored the location of every hilltop that was found in this way, and counted how often certain maxima were found. Varying the step size and repeating the process led to the results shown in Figure 4.1(b). A clear trend can be seen; a higher step size leads to less local maxima. In other words, a coarser quantization of the space contains less local maxima.

A key question is whether a maximum in the quantized space is close, in terms of quality, to a maximum in the full space. The answer depends on the particular quantization. Here, the voting perspective from the previous chapter comes in. Under that perspective, we defined parameters to be accurate if they are support-equivalent to the true distribution (they produce the same supports, Definition 5). Large intervals of parameter values are support-equivalent. Within each interval, the CPTs are similar with respect to state estimation quality.

The approach in this chapter is to have one quantized point per support-equivalence interval. The closeness within each interval is therefore not defined in terms of likelihood, but in terms of state estimation accuracy. The quantized learning phase searches among the quantized CPTs for the one that is support-
equivalent to the true distribution, which is represented by the training data set. In the second learning phase, the quantized maximum is refined by a search in the complete parameter space. If the second phase converges to a maximum that lies in the same support-equivalence interval, this maximum is also support-equivalent to the true distribution. Therefore, this maximum is accurate in a state estimation sense, and close to the quantized maximum.

In other words, the quantized value in each support-equivalence interval is close to the other values in the interval, in a state estimation sense. Therefore it is a good representative of the values in the interval, and a search in the quantized space makes sense.

See Figure 4.2 for an illustration. The solid curve represents an example likelihood surface, for different parameter values. The circles are the quantized values. The vertical dotted lines bound the intervals of values that are support-equivalent. The horizontal dashed lines show the likelihood of parameter values corresponding to the circles. One would like to find the parameter value that corresponds to the global likelihood maximum (the triangle in the figure). In this figure, the maximum likelihood parameter value can be found by starting a search from the quantized maximum (the middle circle). Hence, the quantized maximum will be support-equivalent to the global maximum.
4.2 Quantized EM

In Section 2.3.3, the standard EM algorithm for learning the parameters of a Bayesian network was discussed. A disadvantage of the algorithm is that it only performs a local search of the parameter space, and any found maximum could be a local maximum of quite poor quality. Several improvements for avoiding poor local maxima have been proposed. Two types can be distinguished, one that alters the search procedure in order to escape local maxima, and one that adapts the likelihood landscape to increase the chance of ending up in a (near) global peak.

Changing the Search Procedure

The most trivial method for improving the quality of the output of the EM algorithm is to do multiple restarts from random initializations of the parameters. The thought behind this is that even though EM is doing a local search, at least one of the random starting positions should be in the locality of a (near) global solution. This is generally considered the baseline method.

Tabu search is a local iterative improvement method that puts constraints on the space of possible parameter values in the next iteration [GL97]. The most basic version maintains a set of states that the algorithm is not allowed to visit in the next iteration. Simulated annealing is a modification that allows downhill steps with some probability [KGV83]. This probability becomes smaller as the quality of the current solution improves and as a temperature parameter decreases.

Figure 4.2: Schematic view of a likelihood landscape (solid curve) with support-equivalent parameter intervals (dotted boundaries).
over time. These two methods are not trivial to apply to BN parameter learning and are not often used.

Changing the Score Function

The deterministic annealing approach attempts to find global maxima by smoothing the likelihood landscape [RGF92, UN98]. The degree of smoothing is regulated by a temperature parameter, which is decreased over time. This lets the landscape gradually return to its original unsmoothed shape. The thought behind this method is that the global maximum of the maximally smoothed landscape is easily found. By making the assumption that the global maximum at each annealing step is close to the global maximum of the previous step, one can show that the final outcome is the global maximum of the unannealed landscape. [UN98] shows that the EM algorithm can be adapted to this method by a simple change to the E-step.

Data perturbation is a method that influences the likelihood landscape by assigning weights to data cases [ENFS02]. Changing the weights in a sensible manner can help escape poor local maxima. Higher weights are given to the data cases which have low likelihood under the current model. Again a temperature parameter is used to make the distribution converge to a uniform distribution over time. The idea is that poor maxima are based on outliers in the data set, whereas good maxima are based on common cases in the data. This method can also be implemented by changing the E-step.

4.2.1 Parameter Space Quantization

The quantization approach can be seen as a third type. It does not alter the search procedure, as it will still take maximizing steps. Nor does it change the score function. The choice for the particular quantization is based on the perspective of voting on factor supports from the previous chapter.

One of the consequences of this perspective is that the state estimation of a variable is strongly dependant on the factor supports it receives from its neighboring nodes. If all factors support the true state of a variable, the state estimation will be accurate. It was also observed that many different CPTs can cause the same supports, since the supports are given by inequality relations.

These inequality relations define intervals of parameter values which are similar in terms of state estimation accuracy. As explained in Section 4.1.1, a good quantization of the parameter space is one where each interval is represented by exactly one quantized value, and the quantized value is representative of all values in its interval, possibly including the global maximum.

To achieve this quantization, we map all CPTs in an interval to one single quantized CPT. In other words, all support-equivalent CPTs (recall Definition 5) map to one point in the quantized space. We first define the following function.
For any node $X$, it returns the combined state of the parents of $X$, which are most likely given $X = x_j$:

$$\text{support}(x_j) = \arg \max_k p(X = x_j | \text{Pa}_x = k). \quad (4.1)$$

The CPT of a node $X$ consists of $J$ by $K$ entries denoted by $\theta_{ijk}$ (as shown in Figure 4.3). Each child state $j$ corresponds to one row, and each combined-parent state $k$ to one column. The state support($x_j$) is thus the maximum entry in a row.

The mapping consists of two parts. First, the supported parent states should remain the maximum in its row. This ensures that the supports do not change and we stay in the same interval. However, these entries are replaced by a ‘high’ value denoted by $\alpha$. Second, we smooth out everything else, by replacing all other entries by a ‘low’ value $\beta_k$, such that columns still sum to 1. In this way, any CPT in an interval is changed into the same quantized CPT representing this interval.

As for the value of $\alpha$ and the $\beta_k$, there are several restrictions. There can be between 0 and $J - 1$ $\alpha$’s in a column. No probabilities should be set to 0, since extreme values are unwanted. Columns must sum to 1, and $\alpha$’s must be the maximum entries in each row. The maximum value a $\beta_k$ can take is $1/J$, namely when a column contains no $\alpha$’s. Since $\alpha$ must be greater than any $\beta_k$, it follows that $1/J < \alpha$. In the case that a column contains $J - 1$ $\alpha$’s, the column should still be able to sum to 1. Hence, $(J - 1)\alpha < 1$, and thus $\alpha < 1/(J - 1)$. $\alpha$ can be freely chosen from that interval.

The mapping from normal CPT parameters $\theta_i$ to quantized $\tilde{\theta}_i$ for some variable $X$, is then as follows, for all $j$ and $k$:

$$\tilde{\theta}_{ijk} = \begin{cases} \alpha & \text{if } k = \text{support}(x_j) \\ \beta_k & \text{otherwise} \end{cases} \quad (4.2)$$

$\beta_k$ depends on the number of $\alpha$’s in its column $k$. If $|\alpha|_k$ denotes the number of
Algorithm 1: Quantized EM

/* quantized phase */
while not converged do
    $E[\text{Count}] \leftarrow$ E-step;
    $\Theta \leftarrow$ M-step;
    Replace each $\theta_i \in \Theta$ with $\tilde{\theta}_i$ using (4.2);
end

/* refining phase */
while not converged do
    Regular EM iterations;
end

\[\alpha\]'s in column $k$, then $\beta_k$ is given by
\[
\beta_k = 1 - \frac{\alpha|\alpha|_k}{J - |\alpha|_k}.
\] (4.3)

From the interval for $\alpha$, it follows that $\alpha > \beta_k > 0$ for all $k$, and thus the supports are unchanged. Table 4.1 shows some examples.

The quantized algorithm (Algorithm 1) consists of two phases. The first phase performs a search through the quantized space. After it has converged, the second phase continues with a regular EM search to refine the solution. The quantized search works as follows: After each EM iteration, the solution is mapped to its corresponding quantized value. This new CPT is then used for the next iteration. Essentially, it is a search for the best placement of the $\alpha$'s in the CPT, corresponding to a search for the best supports. From a quantized point, EM will try to make an up-hill step. If this step is large enough, a new interval will be reached (c.f. Figure 4.2), and subsequently a new quantized point.

Another observation from the previous chapter was that the more children a node has, the less important do the actual values in the children’s CPTs become. Since each child has an independent support for a particular state, the redundancy decreases the sensitivity to (small) parameter changes, with respect to state estimation. It makes it more likely to satisfy the conditions that make the majority voting exact. This also implies that all CPTs mapping to the same quantized CPT will result in a state estimation quality that is close to that of the quantized CPT. Thus, the quantized CPT becomes a better representative of its interval. Therefore, the quantization is likely to work best on networks where nodes have many conditionally independent children.

4.2.2 Convergence Properties

The convergence properties of the algorithm require some discussion. One important issue to note is that the current definition of the algorithm does not strictly
4.2 Quantized EM

<table>
<thead>
<tr>
<th></th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
<th>(y_1)</th>
<th>(y_2)</th>
<th>(y_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
<td>0.6</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.4</td>
<td>0.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\Rightarrow\]

<table>
<thead>
<tr>
<th></th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(z_1)</th>
<th>(z_1)</th>
<th>(z_2)</th>
<th>(z_2)</th>
<th>(y_1)</th>
<th>(y_2)</th>
<th>(y_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.40</td>
<td>0.20</td>
<td>0.50</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.32</td>
<td>0.10</td>
<td>0.44</td>
</tr>
<tr>
<td></td>
<td>0.28</td>
<td>0.70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\Rightarrow\]

<table>
<thead>
<tr>
<th></th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(z_1)</th>
<th>(z_1)</th>
<th>(z_2)</th>
<th>(z_2)</th>
<th>(y_1)</th>
<th>(y_2)</th>
<th>(y_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.33</td>
<td>0.28</td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.33</td>
<td>0.28</td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td>0.33</td>
<td>0.45</td>
<td>0.10</td>
<td>0.33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: Some examples of the quantization. (top) CPT \(p(Y|X)\), \(\alpha\) is chosen as 0.8. (bottom) CPT \(p(Y|X, Z)\), \(\alpha = 0.45\).

To improve the solution quality in each EM-loop. In the variational view of EM [NH98] the free energy need not be maximized in every E or M-step. An increase is sufficient for guaranteeing convergence (as explained in Section 2.3.3).

In a regular M-step, the free energy is maximized within the full parameter space. To maximize in the quantized space, we should choose the quantized CPT resulting in the highest free energy. Since the free energy is proportional to the KL-divergence, we could achieve this maximization by choosing the quantized CPT with the lowest divergence to the maximizing CPT in the full space.

The mapping (4.2) defined in the previous section does not necessarily achieve this. There are (rare) situations in which the KL-divergence is not minimized. The reason for this is the fact that \(\beta_k\) depends on the number of \(\alpha\)'s in a column. See Figure 4.4 for an example. During learning this could cause the algorithm to switch back and forth between two solutions.

To solve this problem, convergence can be guaranteed by performing an (exhaustive) search among the possible quantized CPTs. The number of CPTs that needs to be considered is small, and since the search only needs to be done once per CPT per M-step, the computational burden is insignificant compared to the E-step. Another fast method is to directly compare the new quantized CPT with the one found in the previous iteration, in terms of divergence to the point in the full space. In case the new CPT does not decrease the divergence, the old one can be kept.
Parameter Learning

<table>
<thead>
<tr>
<th></th>
<th>$x_1$</th>
<th>$x_2$</th>
<th></th>
<th>$x_1$</th>
<th>$x_2$</th>
<th></th>
<th>$x_1$</th>
<th>$x_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1$</td>
<td>0.60</td>
<td>0.05</td>
<td>$y_1$</td>
<td>0.4</td>
<td>0.3</td>
<td>$y_1$</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>$y_2$</td>
<td>0.25</td>
<td>0.20</td>
<td>$y_2$</td>
<td>0.4</td>
<td>0.3</td>
<td>$y_2$</td>
<td>0.3</td>
<td>0.4</td>
</tr>
<tr>
<td>$y_3$</td>
<td>0.15</td>
<td>0.75</td>
<td>$y_3$</td>
<td>0.2</td>
<td>0.4</td>
<td>$y_3$</td>
<td>0.3</td>
<td>0.4</td>
</tr>
</tbody>
</table>

(a) (b) (c)

Figure 4.4: (a) Maximizing CPT in full space. (b) Subsequent support-equivalent quantized CPT obtained through (4.2). (c) Alternative quantized CPT. KL-divergence between (a) and (b) is 0.38. Between (a) and (c) it is 0.36.

4.2.3 Experiments I

An experiment was conducted to compare our quantized variant of EM with regular EM and EM with parameter smoothing through parameter priors (as explained in Section 2.3.4). Like quantized EM, smoothed EM splits the learning in two phases. The first phase has uniform penalization terms added to the counts, thus smoothing the CPTs, and the second is done without penalization. As we mentioned in the introduction, the quantized algorithm’s characteristics suggest a robustness against over-fitting and the local maxima present when learning from small data sets.

We applied the algorithm on the Insurance, Hailfinder, and Water Bayesian networks. From each BN we sampled a training data set and test data set, using forward sampling. The training data set consisted of 100 samples, and the test set of 1000 samples. We made several central nodes hidden, and randomized the parameters of the network. New parameter values were then learned from the data set using the three different algorithms, using the same convergence threshold. We measured the quality of the new parameters using (i) the average log-likelihood of the test set, and (ii) the state estimation accuracy (percentage of correct state estimations) of one of the variables on the test set. The randomization of the CPTs and relearning was repeated 50 times.

The results averaged over these 50 runs are shown in Table 4.2. As can be seen, quantized EM performs at least as good as the other two methods on the three networks in likelihood and accuracy. With respect to the log-likelihood, quantized EM performs better on the Insurance and Hailfinder networks. The very low improvement in the state estimation accuracy can be explained through the perspective in the previous section; state estimation can be accurate for many different parameter values as long as the supports are adequate. Apparently the other learning methods also learn parameter values that result in adequate supports.
4.2 Quantized EM

Table 4.2: (a) Average log-likelihood and (b) state estimation accuracy on the test set for different networks and methods.

<table>
<thead>
<tr>
<th>network</th>
<th>quantized EM</th>
<th>smoothed EM</th>
<th>regular EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>insurance</td>
<td>−20.83</td>
<td>−21.67</td>
<td>−21.75</td>
</tr>
<tr>
<td>water</td>
<td>−16.28</td>
<td>−16.28</td>
<td>−16.28</td>
</tr>
<tr>
<td>hailfinder</td>
<td>−35.94</td>
<td>−36.03</td>
<td>−36.14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>network</th>
<th>quantized EM</th>
<th>smoothed EM</th>
<th>regular EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>insurance</td>
<td>0.63</td>
<td>0.62</td>
<td>0.62</td>
</tr>
<tr>
<td>water</td>
<td>0.78</td>
<td>0.78</td>
<td>0.78</td>
</tr>
<tr>
<td>hailfinder</td>
<td>0.41</td>
<td>0.41</td>
<td>0.40</td>
</tr>
</tbody>
</table>

Table 4.3: Average number of iterations required by three EM algorithms on different networks. The ‘+’ sign separates the average number of iterations for each learning phase of the quantized and smoothed methods.

<table>
<thead>
<tr>
<th>network</th>
<th>quantized EM</th>
<th>smoothed EM</th>
<th>regular EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>insurance</td>
<td>4.2 + 7.1</td>
<td>17.0 + 15.0</td>
<td>20.0</td>
</tr>
<tr>
<td>water</td>
<td>3.0 + 3.0</td>
<td>4.7 + 3.1</td>
<td>6.0</td>
</tr>
<tr>
<td>hailfinder</td>
<td>2.9 + 3.0</td>
<td>3.3 + 8.4</td>
<td>9.3</td>
</tr>
</tbody>
</table>

**Computational Cost**

The computational cost per EM-iteration is of the same order of complexity for the three algorithms. A clear difference can be observed with respect to the number of iterations needed for convergence, though. Table 4.3 shows the average number of iterations for the three algorithms, given equal convergence thresholds. For the quantized and smoothed variants, the average number of iterations used by their two individual learning phases are shown, separated by a ‘+’. The quantized variant in general requires less iterations than the other two methods. The fact that quantized EM is faster than regular EM even though it consists of two phases is interesting. Apparently, the quantized phase guides the search very quickly to the neighborhood of a (local) maximum, requiring much less iterations in its second phase.

4.2.4 Experiments II

In another experiment, we combine the quantized learning phase with the voting algorithm from the previous chapter. From the experiments above, we see that
Table 4.4: Combining quantized learning with the voting algorithm. Table (a) shows the state estimation accuracy for different networks and inference methods. Table (b) shows the similarity between the voting algorithm applied on the quantized network, and exact inference on the original network.

<table>
<thead>
<tr>
<th>network</th>
<th>inference</th>
<th>$s = 0.4$</th>
<th>$s = 0.6$</th>
<th>$s = 0.7$</th>
<th>$s = 0.8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantized</td>
<td>voting</td>
<td>0.39</td>
<td>0.48</td>
<td>0.78</td>
<td>0.93</td>
</tr>
<tr>
<td>quantized</td>
<td>exact</td>
<td>0.33</td>
<td>0.38</td>
<td>0.67</td>
<td>0.86</td>
</tr>
<tr>
<td>original</td>
<td>exact</td>
<td>0.46</td>
<td>0.64</td>
<td>0.79</td>
<td>0.92</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>network</th>
<th>$s = 0.4$</th>
<th>$s = 0.6$</th>
<th>$s = 0.7$</th>
<th>$s = 0.8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>original</td>
<td>0.48</td>
<td>0.50</td>
<td>0.86</td>
<td>0.96</td>
</tr>
</tbody>
</table>

The quantized learning phase requires very few iterations to converge. In the previous chapter we said that the state estimation can be very accurate as long as the CPTs result in the correct supports. These supports are learned by the quantized phase. Therefore, combining the two methods could result in a very fast algorithm that still scores a high state estimation accuracy.

To see under which conditions this approach works best, we apply the experiment on a number of networks with varying CPTs. The networks were simple trees with a branching factor of 4 and 3 layers, so with 1 root node and 16 leaf nodes. All variables have 3 states, and the CPT between a variable $A$ and $B$ is characterized by $p(b_1|a_1) = p(b_2|a_2) = p(b_3|a_3) = s$, and the other values are equal to $(1 - s)/2$. Thus, $s$ controls the ‘spikiness’ of the CPTs. Spikier CPTs model stochastic processes that are closer to deterministic relations, which are easier to find by learning algorithms.

We let this $s$ take on values $\{0.4, 0.6, 0.7, 0.8\}$. From each network we sampled a data set with the middle layer hidden. This set was then used to train a new network with random CPTs. We performed voting and exact state estimation on the network that was obtained after only performing the quantized phase of quantized EM. This was compared to exact inference on the original network (which generated the data set). We recorded the state estimation accuracy and similarity (as defined in Section 3.4.2) on the root node, averaged over 20 runs.

Table 4.4 shows the results. We can see that for the more spiky networks, using the voting algorithm on the quantized network results in a state estimation accuracy that is equal to the accuracy of exact inference on the original network. The similarity is also high for the more spiky networks.

The simple trees used in this experiment are well suited for the quantized and voting approaches. For comparison, we also applied the experiment on two
4.3 Improving Quantized EM

In the previous section we showed that quantized EM can be faster than regular EM, and find higher likelihood parameters. However, quantized phase of quantized EM can be further improved. To illustrate the significance of the quantized phase we conducted the following small experiment: We created a simple polytree BN with random parameters and sampled a training data set and test data set from it. Some of the variables were made hidden in both sets. Both regular EM and quantized EM were then used to learn the parameters of a new BN, starting from random initialization. The results of 50 random restarts are shown in Figure 4.5. The figure shows a histogram of the test set log-likelihoods of the trained networks. We see that quantized EM (black bars) on average finds better models than regular EM (gray bars). Quantized EM also has a lower variance in its output than regular EM, indicating that fewer local maxima exist in the networks for which it is less suited. The first is a ‘converging’ tree, namely a regular tree with all arc directions reversed. In the corresponding equivalent factor graph, each factor node has many neighbors, and each variable node only 2 neighbors. The second network is the Insurance network, which is quite multiply connected and does not have many conditionally independent fragments. The results are shown in Table 4.5. The scores are much lower, as expected.

Summarizing, combining quantized learning with the voting algorithm works well in a particular type of network, namely trees with spiky CPTs. On these networks, we can use a fast learning algorithm to learn a coarse, quantized model, and then use a fast inference algorithm to get accurate state estimations.

### Table 4.5: Combining quantized learning with the voting algorithm.

<table>
<thead>
<tr>
<th>network</th>
<th>inference</th>
<th>conv. tree</th>
<th>insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantized</td>
<td>voting</td>
<td>0.32</td>
<td>0.54</td>
</tr>
<tr>
<td>quantized</td>
<td>exact</td>
<td>0.32</td>
<td>0.28</td>
</tr>
<tr>
<td>original</td>
<td>exact</td>
<td>0.41</td>
<td>0.82</td>
</tr>
</tbody>
</table>

(a) accuracy

<table>
<thead>
<tr>
<th>network</th>
<th>conv. tree</th>
<th>insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>original</td>
<td>0.63</td>
<td>0.39</td>
</tr>
</tbody>
</table>

(b) similarity

Table (a) shows the state estimation accuracy for different networks and inference methods. Table (b) shows the similarity between the voting algorithm applied on the quantized network, and exact inference on the original network.
quantized space.

However, Figure 4.5 also shows that there is still quite high variance among the models found by the quantized method. Apparently, the quantized search space still contains some local maxima. For comparison, we ran a third algorithm, performing a regular EM run using the ‘golden quantized network’ as initialization. This golden quantized network is obtained by mapping all CPTs of the generating network to quantized CPTs, using (4.2). This algorithm leads to a network whose log-likelihood (indicated by a dashed line) lies within the right-most bin (−4.35) of the histogram. In other words, if the quantized phase would find the golden quantized parameters, the refining phase is likely to lead to a good solution. Improving the quantized search phase will therefore significantly improve the overall algorithm. To achieve this, we propose to use the data perturbation method already mentioned in Section 4.2.

4.3.1 Data perturbation

As mentioned before, data perturbation assigns weights to data cases. This general approach is readily applicable to the EM algorithm. It can also easily be combined with the quantized EM search. The only change takes place in the E-step. If \( w_x \) denotes the weight for case \( x \) then the E-step becomes:

**E-step (Data perturbation)** Given a set of weights \( \mathbf{w} \), for each \( i, j \) and \( k \),
compute:

$$E[\text{Count}(N_i = j, Pa_i = k)] = \sum_{x \in D} w_x \cdot p(N_i = j, Pa_i = k|x; \Theta) \quad (4.4)$$

The weights are regularly changed, for example when the algorithm converges, or after a fixed number of iterations. Changing the weights can help to escape from poor local maxima (also in the quantized space). In [ENFS02], the authors propose two methods for changing the weights. The first is to change the weights to a random Dirichlet distribution. A temperature parameter is used to make the distribution converge to a uniform distribution over time. A second method, that reported better performance, is to give higher weights to the data cases which have low probability under the current model. The thought behind this is that poor maxima are based on outliers in the data set, whereas good maxima are based on common cases in the data. If the weights are uniformly distributed, the algorithm is equivalent to regular EM.

A parameter $\tau$ is used to represent the temperature. A common method to cool down $\tau$ is to multiply it at each time step with some decay value $\delta < 1$. Thus, the temperature at time $t + 1$ is given by: $\tau^{t+1} = \tau^t \delta$.

Random reweighting samples the new weights from a Dirichlet distribution with a parameter $\beta = 1/\tau$ that corresponds to the uniformness of the weights vector. A reweighting scheme that reported better performance was adversarial reweighting. It works by punishing data cases that have a high probability under the current model. This can be achieved by computing the gradient of the weighted log-likelihood score with respect to the weights, and taking a step in the negative direction. In order to converge to a uniform weight vector $w^0$ and decrease the weight changes (from $w^t$ to $w^{t+1}$) over time, two extra terms are added to the score before computing the gradient. The new score function is as follows:

$$Score_w + \beta KL(w^{t+1} \parallel w^0) + \gamma KL(w^{t+1} \parallel w^t) \quad (4.5)$$

and the following reweighting equation is obtained after computing the gradient [ENFS02]:

$$w_{x}^{t+1} = \alpha^{t+1}(w_{x}^0) \frac{\beta}{\beta + \gamma} (w_{x}^t) \frac{1}{\beta + \gamma} e^{-\frac{\eta}{\beta + \gamma} \left( \frac{\partial Score}{\partial w_{x}^{t+1}} \big|_{w_{x}^t} \right)} \quad (4.6)$$

$\alpha$ is a normalizing constant, and the other three factors control the weight cooling, the smoothness of weight changes, and the penalizing, respectively. $\beta$ and $\gamma$ correspond to the importance of the terms that decrease the distance to the uniform ($w^0$) and previous ($w_x^t$) weights, respectively. $\eta$ controls the size of the gradient step. $\beta$ and $\gamma$ should be set proportional to $1/\tau$, making the penalization more important over time. We apply the data perturbation approach on the quantized EM algorithm, and the result is Algorithm 2.

One final issue involves the tuning parameters $\beta$, $\gamma$, $\eta$, and $\delta$. $\delta$ strongly influences the total running time of the algorithm. Letting $\tau$ decay slower will
Algorithm 2: Perturbed Quantized EM

/* perturbed quantized phase */

while $\tau > \tau_{\text{min}}$ do
  Decays $\tau$;
  $w \leftarrow [\beta \gamma \eta \Theta]$ Reweighting;
  while not converged and maximum iterations not reached do
    $E[\text{Count}] \leftarrow [w]$ E-step;
    $\Theta \leftarrow$ M-step;
    Replace each $\theta_i \in \Theta$ with $\tilde{\theta}_i$ using (4.2);
  end
end

/* refining phase */

while not converged do
  Regular EM iterations;
end

generally result in a better model. Values between 0.8 and 0.95 are typically used. The relation between $\beta$ and $\gamma$ determines the importance of moving towards uniformity and the importance of making small changes, respectively. Typically, one would value the convergence to uniformity more, so the values could for example be $\beta = 5/\tau$ and $\gamma = 1/\tau$. The value of $\eta$ relative to $\beta$ and $\gamma$ controls the ‘spikiness’ of the new weight distribution. A reasonable value is $\eta = \tau(\beta + \gamma)$. These values were found after initial testing. It should be noted that the sensitivity of the outcome to the precise values of these tuning parameters was low.

4.3.2 Experiments III

In this section we compare perturbed quantized EM to quantized EM, regular EM, and EM with data perturbation. The algorithms are applied to learning the parameters of the Alarm and Insurance network. We sampled a test data set and train data sets of different sizes from the original network, and made 5 central nodes hidden. The three algorithms were applied to 50 random initializations of the networks’ CPTs. We varied the train set size so we could also investigate the robustness against overfitting. The test set size was fixed at 2000 cases. We recorded the average log-likelihood on the test and train data sets, and the state estimation accuracy on one of the variables for the test set.

Figure 4.6, Figure 4.7, and 4.8 show the outcome of the experiment in terms of log-likelihood and state estimation accuracy. On the training data set, perturbed EM consistently scores the highest likelihood of all four methods. However, it also suffers from overfitting on the smaller data sets, scoring the lowest test set likelihood. Perturbed quantized EM scores lower on the training data set, while
4.4 Conclusion

Table 4.6: Average number of iterations required by four EM variants on different networks.

<table>
<thead>
<tr>
<th>network</th>
<th>pert. quantized EM</th>
<th>quantized EM</th>
<th>pert. EM</th>
<th>regular EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>alarm</td>
<td>47</td>
<td>11</td>
<td>61</td>
<td>16</td>
</tr>
<tr>
<td>insurance</td>
<td>31</td>
<td>12</td>
<td>42</td>
<td>20</td>
</tr>
</tbody>
</table>

reaching the highest test set likelihood for smaller training sets. This shows that it is robust against overfitting. Herein lies one the strength of the algorithm; it focusses on getting the CPTs accurate (support-equivalent to the true distribution), before fine-tuning the solution.

4.3.3 Computational Cost

Again, the computational cost per EM iteration is of the same order of complexity for all three algorithms, as multiplication with the weights requires relatively few extra computations. The reweighting itself requires almost no time. However, applying the temperature cooling scheme causes more EM iterations. Table 4.6 shows the average number of iterations used on the train set of size 800. Both perturbed methods require more iterations than regular EM due to the temperature decay loops. The difference between the perturbed methods is caused by the faster convergence of the quantized variant. A maximum can be reached faster in the quantized space, which was also indicated by the experiments from the previous section.

4.4 Conclusion

This chapter presented a novel approach to parameter learning for Bayesian networks. It is based on the use of a quantized parameter space. The voting perspective from the previous chapter tells us that under certain conditions, large intervals of parameter values have similar state estimation accuracy. For this reason, we have chosen a quantization where each interval is represented by one quantized point.

The mapping from the full space to the quantized space thus maintains support-equivalence, while smoothing out all other details. The resulting algorithm (Algorithm 1) first searches for a maximum in the quantized space. After a (local) maximum is found, one regular EM run (in the full space) is used to refine the solution. Experiments show that for a small training data set, the algorithm scores at least as high as two related algorithms, on test set likelihood. Furthermore, it converges faster. Namely, about twice as fast as regular EM and three times as fast as EM with parameter smoothing. The algorithm is relatively easy to im-
Figure 4.6: Experimental results comparing perturbed quantized EM (solid), quantized EM (dash-dotted), regular EM (dashed), and data perturbed EM (dotted), on learning the parameters of the Alarm network. The horizontal axis shows the train set size. The vertical axis shows average log-likelihood on the train set (a) and on the test set (b).
Figure 4.7: Experimental results comparing perturbed quantized EM (solid), quantized EM (dash-dotted), regular EM (dashed), and data perturbed EM (dotted), on learning the parameters of the Insurance network. The horizontal axis shows the train set size. The vertical axis shows average log-likelihood on the train set (a) and on the test set (b).
Figure 4.8: Experimental results comparing perturbed quantized EM (solid), quantized EM (dash-dotted), regular EM (dashed), and data perturbed EM (dotted). The horizontal axis shows the train set size. The vertical axis shows the state estimation accuracy on one of the variables, for the Alarm network (a) and Insurance network (b).
plement and requires no extra tuning. The algorithm is applicable in situations where we want to speed up learning and the training data set is small.

Next, we performed state estimation by applying the voting algorithm from Chapter 3 on the network that was obtained after the quantized learning phase. The state estimation accuracy was best on tree networks with CPTs that contain clear spikes. On this type of network, the combination of the two methods offers very fast learning and state estimation. This might be useful in situations where one wants fast preliminary state estimation results, while running normal learning and exact inference in the background.

An experiment (Figure 4.5) showed that the variance in the likelihood of the models learned with quantized EM is still quite high (albeit lower than for regular EM). Data perturbation on the quantized phase can decrease this variance, and increase the quality of the learned models. The resulting method (Algorithm 2) was shown to be robust against overfitting, by scoring better on smaller training data sets, compared to existing learning methods. If we are modeling very rare events, it is likely we have only small training data sets. In this case, we can apply perturbed quantized EM to obtain better models, at the expense of more learning iterations.
In Chapter 3, we presented a voting perspective on the inference process, and we gave a definition of model accuracy under this perspective. Namely, we said that a model is accurate if it causes adequate supports. In this chapter, we investigate the probability of encountering such adequate supports. In particular, we show under which conditions we can assume a certain lower bound on this probability. This lower bound has important implications with respect to the properties of the voting algorithm from Chapter 3. Using this lower bound, we also derive two algorithms to detect inaccurate model parts. The first is a filtering method that indicates when the confidence in the accuracy of a state estimation result is low. The second is a method to localize network CPTs that (i) are in a particular case inadequate, or (ii) contain modeling errors that render them inadequate in most cases. The approach is based on monitoring the consistency among supports.

5.1 Introduction

The success of state estimation with Bayesian networks depends on the quality of the model. Faults can be introduced in a Bayesian network through a variety of reasons, such as bad or insufficient training data, or biased human domain experts. We emphasize here the difference between the generalization accuracy and the state estimation accuracy. The generalization represented by a Bayesian network is accurate if it precisely describes the true distributions over its variables. State estimation accuracy deals with correctly estimating the state of a variable in a particular case. In a rare case, a set of observations could result in an erroneous state estimation, even if the model precisely describes the true distributions.

\[^1\] Parts of the material in this chapter have appeared in [NP06, PN06, NP07].
5.1.1 Related Work

Several authors have addressed the problems with reliable inference and modeling robustness. Sensitivity based approaches focus on the identification of modeling components that have a significant impact on the estimation outcome [CvdG98, CGH97]. We must take special care of such components, since eventual modeling errors will have a great impact as well. Sensitivity analysis is carried out prior to the operation and can deal with the accuracy of the generalizations.

Another class of approaches is focusing on determination of the model quality or performance in a given situation at runtime, such as the data conflict [JCNJ90] and straw model [Las91, KV95] methods. The data conflict approach is based on the assumption that given an adequate model all observations should be correlated and \( p(e_1, \ldots, e_n) > p(e_1) \cdots p(e_n) \). If this inequality is not satisfied then this is an indication that the model does not ‘fit’ the current set of observations [Jen01]. A generalization of this method, [KV95], is based on the use of straw models. Simpler (straw) models are constructed through partial marginalization. These models should, in a coherent situation, be less probable than the original model. Situations in which the evidence is very unlikely under the original model and more probable under the straw model indicate a data conflict. Their disadvantage is that the conflict scores are difficult to interpret; it is unclear at which score an action should be undertaken, or what the probability is that a positive score indicates an error. The properties of the method under different model structures are also unknown.

In addition, many of the other approaches focus on the net-performance of models and do not directly support detection of inaccurate parts of a model (for example [vdGR01b]). Exceptions are approaches such as [CDS93], which is based on logarithmic penalty scores. However, in this case the scores can be determined only for the nodes corresponding to observable events, while we want to reason about the nodes modeling hidden events.

5.1.2 Our Approach

We can assume that every variable in a model has a hidden true state, which we are trying to find through inference. In Chapter 3, we presented a perspective based on the conditionally independent fragments in a Bayesian network. Each fragment corresponds to a factor in the posterior probability equation for the variable in which these fragments are rooted. If a factor supports the true state we call it adequate in this particular case (Definition 4).

This is a key difference with respect to the related approaches. We directly connect our notion of adequacy with the hidden truth. We are interested in the probability of encountering such an adequate support. Although it is difficult to compute this probability directly, we can however obtain a lower bound on this probability. We can show that under certain conditions, this lower bound will be
greater than 0.5. We are interested in this lower bound for two reasons.

Firstly, it has important consequences for the convergence of the accuracy of state estimation. Namely, if each support is adequate with at least 0.5 probability, then adding more supports increases the probability that the state with the most supports is the true state. In turn, this has consequences for the robustness of Bayesian networks with many conditionally independent fragments.

Secondly, in certain applications, state estimations are critical to the further course of events. Therefore, it seems prudent to develop methods that, at a relatively low extra cost, can monitor a Bayesian network during run-time, and detect estimation errors. We cannot directly detect whether a factor is adequate or not, because we do not know the true state of all variables. However, based on the lower bound mentioned above, we can derive a measure that can indicate whether a factor is likely to be adequate or not, in a given case. It is based on the following principle: Fragments can be seen as different ‘experts’ giving independent votes about the state of a variable. The degree to which they agree on a state is a measure for their adequacy.

Based on this lower bound we present two algorithms. The first is a filtering algorithm that uses the number of supports for a state as a degree of confidence in the state estimation result. In this way, potentially inaccurate state estimations can be filtered out.

The second is a fault localization method that compares the supports at different nodes in the network to be able to detect network parts that are likely to be inadequate. The advantage of our method over the ones mentioned above, is that we can give a lower bound on its effectiveness, and show that this lower bound has asymptotic properties with respect to the network structure.

### 5.2 Adequate Factors

A central issue in this chapter is the lower bound $p_{\text{sup}}$ on the probability of encountering an adequate factor support, as defined in Section 3.3. Since this depends on the true state of variables, which are typically hidden, it is not possible to give a direct value to this probability. However, as we will show, under certain conditions a lower bound can be found.

In Section 3.3 we defined an accurate fragment as one whose CPT is support-equivalent to the true distribution (Definition 6). In other words, an accurate CPT always gives the same factor supports as the true distribution would. We also argued in Section 3.3.2 that it should be easy for model builders to find these CPTs.

In Definition 4 we said that a factor is adequate if it supports the true state of a variable, in a particular case. An accurate CPT does not guarantee adequate supports in every case. For example, consider the CPT $p(E|C)$ shown in Figure 5.1(a). Suppose this CPT is equal to the true distribution, and thus by
definition the CPT is accurate. If we would observe \( E = e_1 \), factor \( \phi_1(C) \) is equal
to the first row of the CPT. Thus \( \phi_1(C) = (0.7, 0.4) \), and it supports state \( c_1 \) (the
maximum element of the factor). Similarly, observing either \( E = e_2 \) or \( E = e_3 \)
would make factor \( \phi_1(C) \) support state \( c_2 \). Suppose that \( c_1 \) is the true state of
\( C \) in a particular case, then there is a probability of 0.3 that it causes either \( E = e_2 \)
or \( E = e_3 \). In those cases the factor will support \( c_2 \). Since \( c_1 \) was the true state,
the support is inadequate, even though the CPT was accurate. These kind of
situations are sometimes called rare cases.

The question is what is the probability of encountering an adequate factor from
a particular fragment, given that the CPT is accurate. In the above example it
was 0.7 if \( c_1 \) was the true state. If \( c_2 \) had been the true state, observing \( E = e_2 \)
or \( E = e_3 \) would have resulted in an adequate support. The probability of that
observation is 0.6. So if we know the true CPT, we can compute a lower bound on
this probability (0.6 in the example’s case). We formalize this in the next section.

5.2.1 Lower Bound

The probability of getting an adequate factor support depends on the true distributions
and simple relations between the true distributions and the CPT parameters (support-equivalence).
In this section we give conditions under which its lower bound \( p_{sup} \) is guaranteed to be greater than 0.5.

We first introduce support sets. Consider a CPT \( p(E|C) \). For any state \( c_i \),
its support set \( S_{c_i} \) is the set of states of \( E \) that, when observed, support this particular \( c_i \):

\[
S_{c_i} = \{ e_k | c_i = \arg \max_{c_j} p(e_k|c_j) \}.
\]

Each set \( S_{c_i} \) contains the states of \( E \) that, when observed, give the greatest
likelihood to \( c_i \). We assumed that the CPT was accurate, and thus support-
equivalent to the true CPT. Therefore, the true CPT would result in the same
support sets. For the example CPT in Figure 5.1(a), the sets are: \( S_{c_1} = \{ e_1 \} \) and
\( S_{c_2} = \{ e_2, e_3 \} \).
A factor is adequate if it supports the true state. This will happen if the following condition is met: the true state $c^*$ of $C$ causes a state of $E$ that makes the factor to support $c^*$. In other words, $c^*$ causes one of the states of $E$ from its support set. For each possible state $c_i$, we can express the probability $p_{s|c_i}$ that a state from the support set $S_{c_i}$ will be observed:

$$p_{s|c_i} = \sum_{e_j \in S_{c_i}} \tilde{p}(e_j|c_i).$$ (5.2)

Note that here we use the true distribution $\tilde{p}$.

We do not know the true state of $C$ beforehand. However, given the prior over the states of $C$, $\tilde{p}(C)$, the probability of encountering an adequate factor from the fragment corresponding to the CPT, equals

$$p^\times_{cpt} = \sum_{c_i} \tilde{p}(c_i)p_{s|c_i}.$$ (5.3)

This involves the true prior probability on the different states of $C$, which we do not know. However, the expression has the lower bound

$$p_{cpt} = \min_{c_i} p_{s|c_i} \leq p^\times_{cpt}.$$ (5.4)

For the CPT in Figure 5.1(b) we get $p_{s|c_1} = 0.8$ and $p_{s|c_2} = 0.6$, so $p_{cpt} = 0.6$.

For each fragment rooted in a node $C$, the $p_{cpt}$ can be different. We are interested in the probability that any support, coming from any of the fragments, is adequate. We do not know this probability, however we can assume a lower bound $p_{sup}$ on it. For example, if all fragments have a $p_{cpt} > 0.5$, then we know that the probability that any arbitrarily chosen support is adequate is also greater than 0.5. In general, the lower bound is given by

$$p_{sup} = \min_{\Phi_C} p_{cpt},$$ (5.5)

where $\Phi_C$ is the set of fragments rooted in $C$.

For the algorithms presented in this chapter, and for the properties of the voting algorithm presented in Chapter 3, we need to assume that $p_{sup} > 0.5$. Given the equations above, this would require that for all $c_i$, $p_{s|c_i} > 0.5$. The values of $S_{c_i}$ and $p_{s|c_i}$ both depend on the true distribution. Hence, the true CPT needs to satisfy certain conditions to let $p_{sup}$ be greater than 0.5.

**Definition 7** We call a CPT $\tilde{p}(E|C)$ well distributed if $p_{cpt} > 0.5$, in other words, if

$$\forall c_i \in C, \quad p_{s|c_i} > 0.5$$ (5.6)
We illustrate this with the example CPT from Figure 5.1(a) again. We assume that this CPT is equal to both the model parameters and the true distribution. Suppose that $c_2$ is the true state of $C$. We would obtain an adequate factor if we observed either $e_2$ or $e_3$. Thus, for state $c_2$ we have the support set $S_{c_2} = \{e_2, e_3\}$. The probability $p_{s|c_2}$ that either of these observations is caused by $c_2$ is $p_{s|c_2} = \tilde{p}(e_2 \vee e_3|c_2) = 0.6$. Similarly, if $c_1$ were the true state of $C$, then observation of $e_1$ would result in an adequate factor support; i.e. $S_{c_1} = \{e_1\}$. The probability of observing $e_1$ given event $c_1$ is $p_{s|c_2} = \tilde{p}(e_1|c_1) = 0.7$. Thus, for this example we get $p_{cpt} = 0.6$.

Note that in this example the true CPT was well distributed. Therefore, $p_{cpt} > 0.5$ irrespective of the actual CPT parameters, as long as they are support-equivalent to the true CPT. This is an important property, as it allows some room for sub-optimal parameter values.

We can also encounter cases where the relations are such that the lower bound $p_{cpt}$ is less than 0.5. For example, consider the CPT depicted in Figure 5.1(c). We see that for $c_2$, the support set consists of $S_{c_2} = \{e_2\}$. We will in only 40% of the cases observe $e_2$ if $c_2$ is the true state. In other words, there exist domains in which the true distribution is such that the expected state estimation performance can be very poor, even if we had accurate models.

Even more extreme cases are possible. For the CPT in Figure 5.1(d), $p_{s|c_2} = 0$, since $c_2$ will never become the supported state, no matter which state of $E$ we observe. In fact, suppose the model would only consist of these two variables, and the priors over $C$ would be uniform. If $c_2$ were the true state, then the posterior probability of $c_2$ would decrease in 60% of the cases. This is likely to occur when the parents in a CPT have more states than the child, i.e. a CPT has more columns than rows. In a sense, information is lost when $C$ causes a state of $E$ and is then inferred back from observing that state. A solution could be to decrease the number of parent states, by grouping states together. In [PHP+96] it is experimentally shown that this will not significantly decrease the state estimation quality.

If we assume that the CPTs in a model are equal to the true distributions, we can compute the probability of encountering an adequate support for all its CPTs. Since we know the true prior probabilities in this case, we can use $p_{cpt}^*$. Table 5.1 shows this probability for several networks, with the minimum, mean, and maximum taken over all CPTs. We see that all networks, except for the water network, have CPTs with a very low probability and CPTs with very high probability. However, most networks have on average probabilities that are higher than 0.5.

The probability of getting an adequate support has a significant influence on the accuracy of the voting algorithm from Chapter 3. If we look at experiments from that chapter in Table 3.4, we see that the networks with a low accuracy score (Barley, Hailfinder, and Water), also have a low mean value in Table 5.1.

This analysis can be extended to whole fragments by considering that each
### 5.3 Voting Properties

#### Table 5.1: Value of $p^*_\text{cpt}$ for several networks, assuming that their parameters are equal to the true distributions. The minimum, mean, and maximum is taken over all CPTs.

<table>
<thead>
<tr>
<th>network</th>
<th>minimum</th>
<th>$p^*_\text{cpt}$ mean</th>
<th>maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>alarm</td>
<td>0.03</td>
<td>0.80</td>
<td>0.99</td>
</tr>
<tr>
<td>barley</td>
<td>0.00</td>
<td>0.58</td>
<td>1.00</td>
</tr>
<tr>
<td>hailfinder</td>
<td>0.00</td>
<td>0.38</td>
<td>1.00</td>
</tr>
<tr>
<td>insurance</td>
<td>0.05</td>
<td>0.52</td>
<td>0.91</td>
</tr>
<tr>
<td>car-starts</td>
<td>0.75</td>
<td>0.93</td>
<td>1.00</td>
</tr>
<tr>
<td>water</td>
<td>0.00</td>
<td>0.23</td>
<td>0.56</td>
</tr>
</tbody>
</table>

In this section we derive several properties of the voting perspective, based on the assumption that $p^\text{sup} > 0.5$. First, we have to define a support counter for each state of a variable:

**Definition 8** The support counter $n_{h_i}$ of state $h_i$ of a variable $H$ is equal to the number of factors that support $h_i$:

$$n_{h_i} = |\{\phi_k|h_{\phi_k} = h_i\}|,$$

where $|\cdot|$ denotes set cardinality and $h_{\phi_k}$ denotes the supported state from the $k$-th fragment of $H$ (Definition 2). Thus, variable $H$ with $m$ states is associated with a support counter set $\mathcal{N}_H = \{n_{h_1}, \ldots, n_{h_m}\}$.

These counters are the number of votes each state got, in the voting perspective from Chapter 3.

For example, consider a variable $H$ with three fragments and a sequence of factor supports $h_{\phi_1} = h_1$, $h_{\phi_2} = h_1$ and $h_{\phi_3} = h_2$. If $H$ has 3 states, then we would obtain the counter set $\mathcal{N}_H = \{2, 1, 0\}$.

Recall that $p^\text{sup}$ denotes the lower bound on the probability that any obtained support is adequate. We can express the probability that a certain state of a variable $H$ is the true state, given $p^\text{sup}$ and the observed support counters. First
we consider the case where all $p_{cpt}$ are equal, and thus $p_{sup}$ is equal to all $p_{cpt}$. In this case, $p_{sup}$ is exactly the probability of getting an adequate support.

Suppose that $h_1$ is the true state of a variable $H$. If we would observe a sequence of supports

$$\{h_3, h_1, h_1, h_2, h_1\}, \quad (5.8)$$

the adequacies of the supports in this sequence are, given that $h_1$ was the true state,

$$seq = \{\text{inadequate, adequate, adequate, inadequate, adequate}\}. \quad (5.9)$$

Since $p_{sup}$ is the probability of getting a support for the true state, and $1 - p_{sup}$ the probability of getting a support for any of the other states, the probability of this particular sequence is

$$(1 - p_{sup}) \cdot p_{sup} \cdot p_{sup} \cdot (1 - p_{sup}) \cdot p_{sup} = p_{sup}^3 (1 - p_{sup})^2. \quad (5.10)$$

In general, the probability of observing a certain sequence of adequacies, denoted by $seq$, given that the true state is $h_i$, can be expressed as

$$p(seq|h^* = h_i) = p_{sup}^{n_{hi}} (1 - p_{sup})^{N - n_{hi}}, \quad (5.11)$$

where $n_{hi}$ is the number of supports for state $h_i$, and $N$ is the total number of supports.

From this expression we can show an interesting property of the voting perspective from Chapter 3. Namely, we can write the probability that a certain state wins the voting, given that it is the true state. The winner is the state with the greatest support counter, $h_{\text{win}} = \arg \max_{h_i} n_{hi}$. In other words, the probability that the true state got the greatest counter is

$$p(h_{win} = h_i|h^* = h_i) = \sum_{seq} p(h_{win} = h_i, seq|h^* = h_i) \quad (5.12)$$

$$= \sum_{seq} p(h_{win} = h_i|seq) p(seq|h^* = h_i) \quad (5.13)$$

$$= \sum_{seq:h_i} p(seq|h^* = h_i) \quad (5.14)$$

$$\geq \sum_{m = \lceil N/2 \rceil}^{N} \sum_{seq:n_{hi} = m} p(seq|h^* = h_i) \quad (5.15)$$

$$= \sum_{m = \lceil N/2 \rceil}^{N} \binom{N}{m} p_{sup}^m (1 - p_{sup})^{N-m}. \quad (5.16)$$

We applied the following steps: First, we write the expression as a marginalization of the joint probability over all possible sequences. This joint is then factorized,
and simplified because the probability that a certain state wins is conditionally independent of the true state, given a sequence. The probability that $h_i$ wins the voting, given a certain sequence, is a deterministic relation, and we can combine it with the sum. Namely, $\text{seq} : h_i$ denotes all sequences in which $h_i$ is the winner.

We then apply a lower bound. Namely, we restrict the sum to all sequences in which $h_i$ got at least half of the votes. This is a subset of all sequences in which $h_i$ won. We then split the sum, into a sum over the number of votes $h_i$ got ($m$), and a sum over $\text{seq} : n_{h_i} = m$, which denotes all sequences in which the support counter for $h_i$ is equal to $m$. From (5.11), we see that the probability $p(\text{seq} | h^* = h_i)$ is equal for equal $n_{h_i}$. Hence, we can replace summing over $\text{seq} : n_{h_i} = m$ with multiplying with the number of sequences in which $h_i$ has $m$ votes, which is given by the binomial coefficient. Lastly, we filled in the expression from (5.11).

We see that the lower bound (5.16) is a form of a cumulative binomial distribution. This expression has the known property that its value is greater than $p_{\text{sup}}$ if $p_{\text{sup}} > 0.5$ [LS97]. We can conclude from this derivation that if $p_{\text{sup}} > 0.5$, then voting will, with at least 0.5 probability, choose the true state of $H$. Furthermore, this probability will converge to 1 if $p_{\text{sup}}$ increases. The probability also in general increases with the number of supports (votes). It can also be shown that if not all $p_{\text{cpt}}$ are equal, and $p_{\text{sup}}$ is the smallest $p_{\text{cpt}}$, then the same lower bound holds [WM04].

This result indicates a relation between the values in Table 5.1 and the accuracy scores of the voting algorithm in Table 3.4. It explains why the accuracy of voting algorithm is high for the networks which have a high $p_{\text{sup}}$.

Equation (5.16) implies that voting can be accurate as long as we know that $p_{\text{sup}} > 0.5$. This has an important implication for cases in which we have very little training data to estimate the model parameters on. In such cases, we are very unsure about the quality of the estimated parameters, and about the accuracy of state estimation using those parameters. However, even for small training data sets, it is often possible to assume a lower bound on $p_{\text{sup}}$ with high confidence. The parameter estimates are based on frequencies of observations in the data set, sampled from the true distribution in nature. If the sample size is small, the frequencies might not represent the true distribution well. However, through binomial proportion confidence intervals [BCD01] we can compute intervals of values in which the true frequencies lie with high confidence. By taking the lower bound of those intervals, we can make a confident estimate of a lower bound on $p_{\text{sup}}$. This in turn implies a lower bound on the accuracy of the voting perspective.

If the voting algorithm from Chapter 3 is applied on a Bayesian network tree, this result implies that if $p_{\text{cpt}} > 0.5$ for every CPT in the tree, the accuracy of the algorithm’s state estimation is also greater than 0.5. A tree Bayesian network corresponds to a factor graph where each factor node has at most 2 neighbors. If each incoming message at a variable node is correct with probability greater than 0.5, then the outgoing messages to the neighboring factor nodes are also correct with probability greater than 0.5. If the incoming message at a factor node is
correct, then outgoing message to the other neighbor is correct with probability of at least $p_{\text{cpt}}$. This reasoning can be followed from the evidence nodes to every other node in the factor graph. The state estimation accuracy increases with the $p_{\text{cpt}}$’s and with the branching factor of the tree.

To continue the main line of thought, we can use Bayes’ rule on (5.11) to write

$$p(h^* = h_i | \text{seq}) = \frac{p_{\text{sup}}^{n_{h_i}} (1 - p_{\text{sup}})^{N-n_{h_i}} p(h^* = h_i)}{\sum_j p_{\text{sup}}^{n_{h_j}} (1 - p_{\text{sup}})^{N-n_{h_j}} p(h^* = h_j)}. \quad (5.17)$$

The value of this expression is the basis for the confidence measure on the state estimation result. The priors $p(h^* = h_i)$ depend on the precise parameter values in the Bayesian network. However, we want to avoid having the confidence measure depend directly on the precise parameter values, since they can be wrong. Therefore, we choose to set the priors $p(h^* = h_i)$ to be uniform. This also simplifies the derivations in the next sections. Given this choice,

$$p(h^* = h_i | \text{seq}) = \frac{p_{\text{sup}}^{n_{h_i}} (1 - p_{\text{sup}})^{N-n_{h_i}}}{\sum_j p_{\text{sup}}^{n_{h_j}} (1 - p_{\text{sup}})^{N-n_{h_j}}}. \quad (5.18)$$

From this we can make the following observation. $p(h^* = h_i | \text{seq})$ is proportional to $p(\text{seq}|h^* = h_i)$, and the latter increases monotonously with the support counter $n_{h_i}$. Therefore, the state with greatest counter (the voting winner) is the one with the greatest probability of being the true state.

5.3.1 Filtering

Given the derivation from the previous sections, we present an algorithm that can filter out wrong state estimation results. It is based on the idea that the number of factors supporting a state can be used as a confidence measure.

From (5.18) we can see that for a fixed $N$, an increase of the support counter for a specific state $h_i$ will increase the probability that $h_i$ is the true state, assuming that $p_{\text{sup}} > 0.5$. Thus, we can use the support counters as a confidence measure for the state estimation result. Namely, we reject the state estimation, if the state with the maximum posterior probability is not associated with a support counter $n_{h_i}$. Therefore, the state with greatest counter (the voting winner) is the one with the greatest probability of being the true state.
Algorithm 3: Filtering Algorithm

Compute $p(H|\mathcal{E})$ over variable $H$;
Compute set of support counters $\mathcal{N}_H = \{n_{h_1}, \ldots, n_{h_m}\}$;
Determine the state with the highest posterior probability:
\[ h_{\text{max}} = \arg\max_{h_i} p(h_i|\mathcal{E}); \]
\[ \text{if } n_{h_{\text{max}}} < \tau \text{ then} \]
\[ \text{Reject the state estimation based on } p(H|\mathcal{E}); \]
\[ \text{end} \]

is easier for exact inference and voting to agree. On the other hand, making $\tau$ larger than $\lceil N/2 \rceil$, creates cases where the voting does not let any state win. If $\tau = 5$, and the number of votes are 3 and 4, neither state wins, and in this case voting and exact inference always disagree.

Since the confidence measure depends on the voting scheme, it inherits the voting properties outlined in the previous section. One of these properties is the insensitivity to small parameter value changes. Namely, a measure that depends directly on a posterior distribution over is more sensitive to small errors in the model parameters. If the model contains slight errors, such a confidence measure can be erroneous as well. The measure presented here is insensitive to small changes in parameter values, and therefore it will work equally well on models with small errors.

By increasing $\tau$ we raise the minimum required support counter for the most probable state. Since a greater support counter implies a greater probability of that state being the true state, increasing $\tau$ improves the expected accuracy of the cases that pass through the filter. That is, the state estimations that pass the filter are more likely to be correct. For a set of state estimations, this means that the fraction of true positives and negatives compared to false positives and negatives that pass through the filter grows. The downside of increasing $\tau$ is that it creates a larger set of undecided cases.

If we fix $\tau$ to $\lceil N/2 \rceil$, then we accept state estimation $h_{\text{max}}$ if it got at least half of the supports. Equation (5.16) implies that the probability that the true state will get at least half of the supports is greater than 0.5. This probability increases with $p_{\text{sup}}$ and with the number of supports. Similarly, we would reject state estimation $h_{\text{max}}$ if it got less than half of the supports. From (5.16) it also follows that the probability that a non-true state will get more than half of the votes is less than 0.5, and decreasing with $p_{\text{sup}}$ and the number of supports. Thus, any cases that pass through the filter are most likely to be correct state estimations, and any cases that are rejected by the filter are most likely to be incorrect state estimations. This holds for any value of $\tau$. However, higher $\tau$ creates more rejected cases, which might be unwanted, while lower $\tau$ does not reject many cases at all.
5.3.2 Experiments

The filtering effectiveness is illustrated in an experiment. Data was sampled from a fictitious generative model, a Bayesian network with a tree topology, consisting of 3 layers and a branching factor of 7. All variables had two states, and the CPTs are chosen such that $p_{sup}$ could be controlled. Each sampled set of observations was inserted in a network that was identical to the generative model; thus we used the true model for the state estimation.

Figure 5.2 shows the state estimation accuracy, after rejecting the cases that did not reach the threshold. The horizontal axis represents threshold $\tau$ while the vertical axis represents the percentage of correct state estimations. The curves correspond to different values of $p_{sup}$, namely 0.6, 0.7, and 0.8. The curves show that the filtering effectiveness is a function of $p_{sup}$ and threshold $\tau$. For example, the curve with diamonds corresponds to $p_{sup} = 0.7$. By using $\tau = 5$ the percentage of incorrect state estimations was reduced by approximately 50%.

With higher thresholds, the number of estimations that get rejected also grows. Figure 5.3 shows the percentage of rejected estimations, as a function of the threshold, for different values of $p_{sup}$.

If we had large amounts of real world data we could obtain an optimal rejection threshold with the help of ROC curves [DH73]. However, we often deal with applications where the available quantity of data does not allow reliable determination of the thresholds. The objective of the filter is to find incorrect state
5.3 Voting Properties

Figure 5.3: Percentage of rejected state estimations as a function of different thresholds $\tau$. Stars, diamonds and circles correspond to $p_{sup} = 0.6$, $p_{sup} = 0.7$ and $p_{sup} = 0.8$, respectively.

A filtering method like this is useful in applications where an incorrect state estimation could have devastating consequences, while investigating estimations that are rejected is less costly. For example, a fire in a remotely observed section of a chemical plant can be detected with relatively unreliable sensors. A failure to detect the fire could result in a catastrophe. Activation of a flag indicating a potentially incorrect state estimation could prompt a human operator to zoom in with a camera or send a remotely controlled robot. In this case, the damage caused by an incorrect state estimation outweighs the cost of an alternative mode
of observation involving manual work. Of course, this will not work when the number of false alarms are too high.

5.4 Localization of Inadequacies

5.4.1 Factor Adequacy

In this section we present an algorithm that can localize the causes of inadequate factor supports. First we introduce a new measure, factor consistency, which describes the degree to which factors ‘agree’ with each other. Given a variable $H$, we call a set of factors $\Phi_H$ consistent iff they support the same state

$$\forall \phi_k, \phi_l \in \Phi_H \quad h_{\phi_k} = h_{\phi_l}$$

(5.19)

Since there is only one true state $h^*$, it is clear that if each factor in $\Phi_H$ is adequate, then that set must be consistent.

Consequently, if a set of factors is not consistent, then some of the factors in that set are inadequate, that is they violate (3.17). In that case we should determine which of the factors in an inconsistent set are inadequate. For this, we derive a consistency measure, indicating how consistent a certain factor is with the rest of the factors.

Recall Equation (5.18), the expression for the probability of a state being the true state, given a sequence of factor supports. We are interested in the conditions
under which this probability is greater or smaller than 0.5.

Let us first look at \( p(h^* = h_i|\text{seq}) > 0.5 \). For brevity, let \( p \) denote \( p_{\text{sup}} \) here, and \( n_i \) denote \( n_{h_i} \). The lower bound is satisfied if

\[
\frac{p^{n_i}(1 - p)^{N-n_i}}{\sum_j p^{n_j}(1 - p)^{N-n_j}} > \frac{1}{2} \iff 2p^{n_i}(1 - p)^{N-n_i} > p^{n_i}(1 - p)^{N-n_i} + \sum_{j \neq i} p^{n_j}(1 - p)^{N-n_j},
\]

(5.20)

\[
\iff p^{n_i}(1 - p)^{N-n_i} > \sum_{j \neq i} p^{n_j}(1 - p)^{N-n_j}.
\]

(5.21)

Note that the right hand side sums over all states \( j \), except for \( i \). This inequality is difficult to simplify further, because of the sum term. However, we can take an upper bound of the right hand side of the inequality. An upper bound for the sum over a set \( x \) is the size of the set times the maximum element:

\[
|x| \max x \geq \sum x.
\]

Recall that \( m \) denotes the number of states of \( H \). Inequality (5.21) is true if the following inequality is true

\[
\iff p^{n_i}(1 - p)^{N-n_i} > (m - 1) \max_{j \neq i} p^{n_j}(1 - p)^{N-n_j}.
\]

(5.22)

\[
\iff (1 - p)^N \left( \frac{p}{1 - p} \right)^{n_i} > (1 - p)^N (m - 1) \max_{j \neq i} \left( \frac{p}{1 - p} \right)^{n_j}.
\]

(5.23)

Let \( c \) denote \( p/(1 - p) \). Note that \( p = p_{\text{sup}} > 0.5 \) implies \( c > 1 \). Then this becomes:

\[
\iff c^{n_i} > (m - 1)c^{\max_{j \neq i} n_j},
\]

(5.24)

and taking the logarithm, we get:

\[
\iff n_i - \max_{j \neq i} n_j > \frac{\log(m - 1)}{\log c}.
\]

(5.25)

Next we consider \( p(h^* = h_i|\text{seq}) < 0.5 \). Taking the same steps as above, we get

\[
p^{n_i}(1 - p)^{N-n_i} < \sum_{j \neq i} p^{n_j}(1 - p)^{N-n_j}.
\]

(5.26)

We now take a lower bound of the right hand side of the inequality. A lower bound for the sum over a set \( x \) is the maximum element of the set: \( \max x \leq \sum x \). Inequality (5.26) is true if the following is true

\[
\iff p^{n_i}(1 - p)^{N-n_i} < \max_{j \neq i} p^{n_j}(1 - p)^{N-n_j}.
\]

(5.27)
Applying the same steps as above, we get

\[ c^{n_i} < c^{\max_{j \neq i} n_j}, \quad (5.28) \]

\[ n_i - \max_{j \neq i} n_j < 0. \quad (5.29) \]

Recall that \( n_i \) denoted \( n_{h_i} \). Apparently either bound is satisfied depending on the value of the expression \( n_{h_i} - \max_{j \neq i} n_{h_j} \). It’s value is the support counter of a state \( h_i \) minus the maximum counter among the other states. In this sense it is a measure for the consistency of any factor that supports \( h_i \). Namely, the greater the consistency measure value, the more factors ‘agree’ with this factor, hence it is more consistent.

**Definition 9** The consistency measure for a factor \( \phi_k \in \Phi_H \) is expressed as:

\[ C_H(\phi_k) = n_{h_i} - \max_{j \neq i} n_{h_j}, \quad (5.30) \]

where \( h_i \) is the state supported by factor \( \phi_k \).

We now have the following implications:

\[ C_H(\phi) < 0 \quad \Rightarrow \quad p(h^* = h_{\phi}|seq) < 0.5 \quad (5.31) \]

\[ C_H(\phi) > \frac{\log(m - 1)}{\log[p_{sup}/(1 - p_{sup})]} \quad \Rightarrow \quad p(h^* = h_{\phi}|seq) > 0.5 \quad (5.32) \]

These implications give bounds on the probability of an adequate factor support, given its consistency measure. This consistency measure is an easily observable quantity (through the counters \( n_{h_i} \)), that can indicate whether a particular support is adequate. For example, if a factor support has a negative consistency measure, it is probably inadequate.

Implication (5.32) is not trivial to interpret, since the condition depends on the unknown value \( c = p_{sup}/(1 - p_{sup}) \). Note that the consistency measure can only take integer values. For example, if \( \log(m - 1)/\log c = 0.2 \), then \( C_H = 1 \) is sufficient to satisfy the inequality. If \( \log(m - 1)/\log c = 0.7 \), then \( C_H = 1 \) is also sufficient. It is clear that the value of the consistency measure that satisfies the inequality is quite insensitive to the precise value of \( c \). For each value of \( C_H \) and \( m \), we can compute the interval of values for \( c \) (or \( p_{sup} \)) for which the inequality is satisfied. Table 5.2 shows the lower boundary of that interval for different values of \( C_H \) and \( m \). For example, if the number of states \( m = 4 \), then \( p_{sup} \) has to be greater than 0.63 in order to have \( C_H = 2 \) satisfy (5.32). If the variable is binary \( (m = 2) \), then \( C_H = 1 \) is always sufficient.
### 5.4 Localization of Inadequacies

<table>
<thead>
<tr>
<th></th>
<th>$C_H = 1$</th>
<th>$C_H = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m = 2$</td>
<td>0.50</td>
<td>0.50</td>
</tr>
<tr>
<td>$m = 3$</td>
<td>0.66</td>
<td>0.58</td>
</tr>
<tr>
<td>$m = 4$</td>
<td>0.75</td>
<td>0.63</td>
</tr>
<tr>
<td>$m = 5$</td>
<td>0.80</td>
<td>0.66</td>
</tr>
</tbody>
</table>

Table 5.2: Minimum value of $p_{sup}$ that is sufficient to satisfy (5.32) given a certain value of $C_H$ and number of states $m$.

#### 5.4.2 Localization

Using the derivation results from the previous section, we can identify parts of fragments that are likely to contain the cause of an inadequate support. The method is based on a comparison of the support adequacy at different nodes in the graph. This comparison gives an indication of the existence of causes in the part between the nodes.

First we need to define a more local notion of adequacy.

**Definition 10 (Adequate CPT)** A CPT $p(A|B)$ is adequate (in a particular case) if the following is true: if one of the two variables is instantiated to its true state, then the true state of the other variable is the most likely state, i.e. the supported state.

Essentially, Definition 4 of adequacy for factors is a specific case of this definition. After all, we can capture an entire fragment in one CPT $p(Evidence|Root)$, and the evidence variables are always set to their true state. Hence, if the factor is adequate and supports the true state of the root variable, the CPT is adequate following the above definition. Another observation we can make is that every CPT in a fragment is adequate in a given case, the factor corresponding to that fragment is also adequate. The opposite is not necessarily true.

Consider a network section as shown in Figure 5.6(a). It contains at least two nodes, $X$ and $Y$, both being the root of several fragments (denoted by small circles). One of the fragments of $X$ connects $X$ to $Y$.

First we consider one particular fragment $F_k^Y$ rooted in $Y$, and its corresponding factor $\phi_k$ (see Figure 5.6(b)). Given the evidence inserted in this fragment, we can compute the consistency measure $C_Y(\phi_k)$ at node $Y$. Implications (5.31) and (5.32) can then indicate whether the factor $F_k^Y$ is likely to be adequate.

Next, let $F'$ be fragment $F_k^Y$ plus the part between $X$ and $Y$ (see Figure 5.6(c)). $F'$ would be a fragment of $X$ if we would remove all fragments of $Y$ except $F_k^Y$ from the graph. Let $\phi'_k$ be its corresponding factor. Let the value of this factor be computed by instantiating variable $Y$ with the state supported by $\phi_k$. With the value of the factor we can determine the consistency measure $C_X(\phi'_k)$ at node $X$ for fragment $F'$ (see Figure 5.6c). This gives an indication of the adequacy of factor $\phi'_k$. 

---

**Table 5.2** Minimum value of $p_{sup}$ that is sufficient to satisfy (5.32) given a certain value of $C_H$ and number of states $m$. 

<table>
<thead>
<tr>
<th></th>
<th>$C_H = 1$</th>
<th>$C_H = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m = 2$</td>
<td>0.50</td>
<td>0.50</td>
</tr>
<tr>
<td>$m = 3$</td>
<td>0.66</td>
<td>0.58</td>
</tr>
<tr>
<td>$m = 4$</td>
<td>0.75</td>
<td>0.63</td>
</tr>
<tr>
<td>$m = 5$</td>
<td>0.80</td>
<td>0.66</td>
</tr>
</tbody>
</table>
Figure 5.6: (a) Network section. $X$ and $Y$ are nodes, the small circles denote arbitrary (conditionally independent) network parts, and the double arrows denote arbitrary connections (possibly through other nodes). (b) Consistency at node $Y$. (c) Consistency at node $X$.

Multiplying the CPTs of all nodes in the part between $X$ and $Y$, including $Y$, and marginalizing all variables except $X$ and $Y$, results in a CPT $p(Y|X)$. The combination of the adequacies of the two factors tells us about the adequacy of this CPT.

First, we consider the case where both $X$ and $Y$ are binary variables. With respect to the CPT $p(Y|X)$, two situations are possible: either (i) observing $Y = y_1$ supports $X = x_1$, and $y_2$ supports $x_2$, or (ii) observing $y_1$ supports $x_2$, and $y_2$ supports $x_1$. An adequate CPT maps the true state of $Y$ to the true state of $X$, and therefore also the other (non-true) state of $Y$ to the non-true state of $X$. If a factor is adequate up to $Y$, i.e. it supports the true state of $Y$, and the extended fragment supports the true state of $X$, the CPT is adequate. If a fragment supports the false state at $Y$, but the extended fragment will support the true state at $X$, the CPT must be inadequate. Using this reasoning, we can derive a localization rule for every combination, as shown in Table 5.3.

In the case of multi state variables, the mapping from true state to true state, and from non-true to non-true, does not always hold. Therefore, we can only judge the adequacy of the CPT if the support at $Y$ is adequate. In other cases, the adequacy of the CPT is unknown. These localization rules are shown in Table 5.4. If the consistency measure values were not high or low enough to use (5.32) or (5.31), then the adequacy of the CPT is also considered unknown.
5.4 Localization of Inadequacies

| factor $\phi'_k(X)$ | factor $\phi_k(Y)$ | $p(Y|X)$ is adequate? |
|---------------------|-------------------|----------------------|
| adequate            | adequate          | yes                  |
| adequate            | inadequate        | no                   |
| inadequate          | adequate          | no                   |
| inadequate          | inadequate        | yes                  |

Table 5.3: Localization rules if $X$ and $Y$ are binary variables.

| factor $\phi'_k(X)$ | factor $\phi_k(Y)$ | $p(Y|X)$ is adequate? |
|---------------------|-------------------|----------------------|
| adequate            | adequate          | yes                  |
| adequate            | inadequate        | unknown              |
| inadequate          | adequate          | no                   |
| inadequate          | inadequate        | unknown              |

Table 5.4: Localization rules for multi state variables.

In other words, we compare the consistency at two adjacent fragment roots and classify the CPT between those nodes as either (i) inadequate, (ii) adequate, or (iii) unknown. We can show that if a CPT is not classified as unknown, then the classification is correct in the majority of cases.

**Proposition 1 (Fault Localization)** Given $p_{sup} > 0.5$, the localization rules in Table 5.3 and 5.4 will correctly determine whether CPT $p(Y|X)$ is adequate or not with more than 0.5 probability.

**Proof** Consider an arbitrary node $A$. If for this node and factor $\phi$ we observe $C_A(\phi) > \frac{\log(m-1)}{\log c}$ then we assume that the state supported by $\phi$ is equal to the true state of $A$, $a^*$. Equation (5.32) says that, given $p_{sup} > 0.5$, the probability that this assumption is correct, $p(a^* = a_\phi|seq)$, is larger than 0.5. $seq$ is the sequence of supports as defined in Section 5.3.

Analogously, if for an arbitrary node $A$ and factor $\phi$ we observe $C_A(\phi) < 0$ then we assume that the state supported by $\phi$ does not equal $a^*$. Equation (5.31) implies that, given $p_{sup} > 0.5$, the probability that this assumption is correct, $1 - p(a^* = a_\phi|seq)$, is larger than 0.5.

Thus, we make the correct assumption about the (in)adequacy of a factor with probability $p_c > 0.5$.

First, consider the case that $X$ and $Y$ are binary variables. We correctly classify the CPT, if the assumption about the factor (in)adequacy is correct at both variables. If the assumption at both variables is incorrect, we will make the same classification. Therefore, the probability of making the correct classification is $p_{c}^2 + (1 - p_c)^2$. This probability is greater than 0.5 if $p_c > 0.5$.

Next, consider the multi-state case. Suppose factor $\phi_k$ of $Y$ was really inadequate. Then it would only be correct to classify $p(Y|X)$ as unknown. This
will happen if $\phi_k$ is correctly assumed to be inadequate, which has a probability $p_c > 0.5$.

Suppose factor $\phi_k$ was really adequate. Then wrongly assuming this factor to be inadequate will lead to a classification as unknown, which is not considered wrong. The correctness of the classification will depend on correctly assuming the (in)adequacy of factor $\phi'_k$ of $X$, which has a probability of $p_c > 0.5$. □

This procedure can be repeated for each fragment rooted in $Y$, each time getting an adequacy classification of the part corresponding to $p(Y|X)$. These classifications can be combined through majority voting, ignoring all unknown classifications. We can apply fault localization to all non-leaf nodes by running Algorithm 4.

**Algorithm 4: Localization Algorithm**

```plaintext
for each fragment root $X$ do
  for each fragment $F^X_k$ of $X$ do
    Let $Y$ be the nearest fragment root within $F^X_k$;
    for each fragment $F^Y_l$ of $Y$ do
      Compute $C_X(\phi')$ and $C_Y(\phi)$ for $F^Y_l$;
      Using (5.32), (5.31), and the rules in Table 5.3 and 5.4, classify CPT $p(Y|X)$;
    end
    Use majority voting on all classifications of CPT $p(Y|X)$ based on different $F^Y_l$;
  end
end
```

We observe the following property of Algorithm 4: The effectiveness of the algorithm is increased by the majority voting at the end of the algorithm. Since each vote is correct with more than 0.5 probability (see Proposition 1), the expected localization accuracy increases. Furthermore, this accuracy converges asymptotically to 1 if the branching factors increase, because this amounts to more votes. These kind of properties cannot be shown for the related data conflict and straw model approaches.

Getting a classification as unknown is not likely to occur. Namely, $p_{sup} > 0.5$ implies that in the majority of cases the support at $Y$ will be adequate. Furthermore, the majority voting in Algorithm 4 can compensate for a classification as unknown, if the other fragments of $Y$ produce non-unknown classifications.

Consider the situation where a fragment of $X$ does not contain a fragment root node $Y$. In that case, we can use the consistency measure of the factor corresponding to that fragment to directly determine the adequacy of the factor. This allows us to also localize inadequacies in leaf node CPTs.
5.4.3 Experiments

To verify our claims and illustrate some of the properties of Algorithm 4 we applied it to several synthetic networks, in which we purposely introduced inadequacies. We also applied it to a real network, which we adapted such that it represents an oversimplification of the problem domain, thus introducing additional inadequacies.

Synthetic Networks

We generated Bayesian networks with a tree-shaped graph with fixed branching factor and 4 layers. We initialized all CPTs randomly, but such that the probability of a CPT being adequate in a particular case could be controlled. We call this probability $p_{cpt}$ and let it take values $1, 0.95, \ldots, 0.4$. Then we generated 1000 data cases from each generated network, making all variables hidden, except for the leaves. We then ran inference on each case, applied the Localization Algorithm, and observed its output.

The output was then compared with the ground truth, i.e. which CPTs really were inadequate for the given data case. This ground truth was known to us from the complete sampled data set, without hidden variables. Given the inadequate CPTs that were present for a given case, we recorded the percentage of inadequate CPTs that the algorithm managed to detect (hit rate) and the percentage of detections that turned out to be false alarms.

We applied this procedure on networks with varying branching factors (but the same general structure). The results are plotted in Figure 5.7. The three top curves show the hit rate, and the three bottom curves correspond to false alarm percentage. Each of the three lines correspond to different values of $p_{cpt}$. The figure confirms that higher branching factors increase the algorithm’s effectiveness, for all values of $p_{cpt}$. The effectiveness also seems to converge to 1 for higher branching factors.

Next, we varied the number of states per variable in Figure 5.8. The different curves correspond to different numbers of states, and the horizontal axis to $p_{cpt}$. One observation we can make is that at a certain point the effectiveness of the algorithm becomes too low to be usable. This is the point where on average almost half of the CPTs are inadequate. In that situation the inadequate CPTs cannot be discerned from the adequate CPTs anymore. However, in such a situation, the network is not likely to be useful for state estimation anymore.

Figure 5.8 also shows that the algorithm performs better on networks with more variable states. This can be explained by the fact that in such cases inadequate factor supports are spread over more states. For example, suppose that in a certain case a variable is in state 1, but an inadequate fragment supports a different state. If a variable has more than 2 states, this inadequate factor can support one state among more alternatives. Thus, on average, the difference
Figure 5.7: The effect of branching factors on a network with 4-state variables, for different values of $p_{cpt}$: 0.9 (dash-dotted), 0.7 (solid), 0.5 (dashed). Top curves show hit rate (HR), bottom curves show percentage of false alarms (FA).

Figure 5.8: The effect of variable states on a network with branching factor 5, for different values of $p_{cpt}$. Number of states: 2 (dashed), 3 (solid), 4 (dash-dotted). Top curves show hit rate (HR), bottom curves show percentage of false alarms (FA). The dotted line shows the worst case scenario for 3 and 4 states.
5.4 Localization of Inadequacies

between the support counter of the true state and the other counters increases, making the true state more easily distinguishable. For example, given counters $\mathcal{N} = \{3, 2\}$, a factor supporting state 1 would have a consistency measure of 1, while for $\mathcal{N} = \{3, 1, 1\}$ it would be 2. Note that the amount of spread also influences the quality, as can be seen from the dotted line in Figure 5.8. This line shows the effectiveness if we fix only one alternative state. That is, if a factor is inadequate it will always support the same alternative state, which on average decreases the value of the consistency measure. For example, $\mathcal{N} = \{3, 0, 2\}$. This worst case scenario is equivalent to localization in binary state Bayesian networks. We expect real networks to be somewhere in between the worst and best case scenario.

Real Data

Next, we tested the algorithm on a real network, namely a subgraph of the Munin medical diagnosis network [AJA+89] (see Figure 5.9 for the subtree structure). This tree is a simplification of the Munin problem domain. We first chose the network structure and then set the parameters to their maximum likelihood value using a data set sampled from the original network. Obviously, when we would attempt to classify cases using this simple Bayesian network, many wrong state estimations will occur because of the simplification. The question is whether the algorithm can detect these wrong estimations and localize their causes.

We applied the algorithm on the tree network for a set of sample cases generated by the complete network. Since the state of all (hidden) variables in all cases was known to us (but not to the network), we knew which CPTs were inadequate. On the tree network, the algorithm found 75.7% of all inadequate CPTs, while producing 20.9% false positives, which confirms that the algorithm can be effective in a real world setting, even for an oversimplified model.
5.4.4 Inadequacy or Inaccuracy

We can distinguish between inadequate CPTs due to rare cases, and inaccurate fragments due to modeling errors only by their frequency of occurrence (as they both violate (3.17)). Inaccurate are those CPTs that do not satisfy (3.19), and thus are not support-equivalent to the true distribution. The frequency can be observed by performing fault localization on a Bayesian network for a large set of cases. If a certain CPT is identified as inadequate for a large number of those cases, this is an indication that the CPT parameters might not correctly capture the relations as described by (3.19). Note that the assumption that $p_{sup} > 0.5$ for a particular fragment is still valid if a sufficient majority of the CPTs in that fragment correctly capture those relations. This is a plausible assumption, since we can expect that experts or learning algorithms can easily identify such simple relations for most CPTs.

Alternatively, it might be possible to find model errors by localizing faults on a case from the domain which one knows is not rare. In other words, a case for which we know that the true state of every node is the most likely state given the evidence. This excludes the possibility for inadequacies due to a rare case. Any found inadequacies are then probably caused by modeling errors.

5.5 Conclusion

This chapter expanded the voting perspective from Chapter 3. In that chapter, we defined an adequate factor (or vote) as one that supports the true state of a variable. In this chapter we focussed on the probability $p_{sup}$ of encountering such an adequate factor. We derived conditions under which this probability is greater than 0.5. CPTs that satisfy these conditions were called ‘well-distributed’.

This lower bound of 0.5 has important consequences for the properties of voting perspective. We showed that the probability that the true state wins the voting is greater than 0.5, and converges to 1 if $p_{sup}$ increases.

We derived an expression for the probability that a certain state is the true state given the observed votes. Voting always chooses the state for which this expression has the greatest value. We showed that the value of this expression for the true state increases as the number of votes increases. The number of votes depends on the number of fragments rooted in a node, and hence this branching factor has an significant influence on the quality of the algorithms that use the voting perspective.

Based on that expression, we proposed to use the number of supports for a particular state as a coarse confidence measure for the state estimation result. The coarseness of the measure makes it less sensitive to small changes in parameter values. We showed that a filtering algorithm based on that measure is effective in rejecting incorrect state estimations. On a tree network where 20% of the supports are inadequate, the algorithm can find 90% of incorrect state estimations, while
producing only 15% false alarms. If a large set of data is available, we can use ROC curves to determine the optimal threshold on the confidence measure.

Finally, we introduced a measure for the consistency among a set of supports. We derived implications that set bounds on the probability that a certain state is the true state, given the observed consistency measure. These implications are the basis for an algorithm that can localize inadequate CPTs in a Bayesian network. On networks where 20% of the CPTs are inadequate, the algorithm found 90% of all inadequate CPTs, while producing only 5% false alarms. An advantage of this method is that the effectiveness of the algorithm can be shown to increase and converge to 1 with increasing number of fragments (votes), and with increasing \( p_{sup} \).

The localization algorithm can discover CPTs that are inadequate in a particular case. By applying the localization algorithm to many different cases obtained in different situations, we can localize CPTs which are inadequate in the majority of the cases. Such CPTs represent modeling errors. That is, they do not satisfy inequality relations (3.19) with the true distributions in ‘nature’. Such erroneous CPTs can then be repaired manually or through learning.

Errors cannot be avoided if a model is used in changing domains and the learning examples or expertise that generated the model did not capture the characteristics of the new domain. Fault localization can be especially useful in domains which change sufficiently slowly, allowing us to discover inadequate CPTs and adapt the model gradually to the new domain.

In a given case, the inadequate CPTs most likely form a minority. When certain CPTs contain errors they will likely be inadequate in most cases. However, as long as the number of modeling errors remains low, the inadequate CPTs remain a minority, and the algorithm can still be effective. The experiments illustrate this. The existence of an error will slightly decrease \( p_{sup} \), but if it remains high enough, inadequate CPTs can still be localized.

### 5.5.1 Network Limitations

The presented approach is suitable for networks with a ‘tree-like’ structure and many conditionally independent network fragments. Such models can be relevant for a variety of applications. For example, for monitoring systems that are based on fusing large quantities of heterogeneous and uncertain information [dOPH07, PdOM+07, PdOM+08]. In such settings, we can assume that each event is significantly influenced by only a small fraction of other events. For example, in monitoring systems, hardware components or reports from one sensor do not influence hardware or reports from another sensor. The corresponding Bayesian networks typically consist of several network fragments which are conditionally independent given small sets of variables.

In more connected networks, the instantiation of certain variables can cause parts of the network to become conditionally independent. In some application
we could for example assume that variables associated with risk factors will be instantiated. Furthermore, simple tree networks with high branching factors, such as naive Bayesian networks, have been reported to perform well, despite oversimplifying the problem domain.
Discussion and Future Research

This thesis presented a perspective on the grounding of Bayesian networks. It is known that Bayesian networks are robust against small changes in the values of parameters, when they are used in the context of state estimation (e.g. [PHP+96, DP96, CD03]). An open question is how much these values are allowed to change before the model becomes unusable. This thesis presents a contribution towards answering that question.

The approach we took was to start with a voting perspective on the inference process (Chapter 3). In this perspective, we ignore the precise values of the factors in the equation for the posterior probability over the states of some variable of interest. Instead, we consider for each factor the state that got the highest weight. This was called the supported state of that factor. We identified the conditions on the factor values, for which the state with the greatest estimated probability is the winner of majority voting on the factor supports. In particular, they should not contain very extreme values. These conditions are satisfied for large intervals of parameter values. Only if the values are extreme, and if the voting is won with a small margin, then the majority voting winner can be different from the most probable state.

The implications with respect to the grounding question were as follows: We called a support for the true state of the variable of interest adequate, since such a support contributes to a correct state estimation. A CPT in a Bayesian network that produces the same supports as the true CPT would, is called accurate. Through this definition, we are able to quantify the allowed parameter changes as those that keep a CPT accurate. Since the supports are based on inequality relations, large intervals of parameter values are accurate. This implies that it is easy for model builders or learning algorithms to find accurate values. It simplifies the model building process, and shows that we can use Bayesian networks in good faith. It also shows that Bayesian networks are very robust, which corroborates
the results from previous works on Bayesian network robustness.

In Chapter 5, we investigated the probability of getting adequate supports, since this probability has significant consequences for the correctness of the voting perspective. In particular it is important to set a lower bound greater than 0.5 on this probability. This lower bound is crucial for the convergence properties of the algorithms that are presented. CPTs that satisfy this lower bound are called well-distributed. If the CPTs in a Bayesian network are well-distributed, then the voting perspective is more accurate if we have more supports. In other words, it works well on networks with many independent parts.

Several algorithms were introduced, based or inspired on the voting perspective. They exploit the robustness intervals we defined through the voting, to gain speed-ups or insensitivity to precise parameter values. These algorithms also validate the voting perspective itself. The fact that their output is often comparable to existing methods, especially on tree networks, shows that the voting perspective makes sense in such networks. The experiments confirm the properties that were predicted on the basis of the voting perspective. They show that the assumptions that were made, for example about factor balance, are plausible in many applications.

**Belief Propagation through Voting**

The voting algorithm for belief propagation (Chapter 3) performs state estimation by sending voting message through a graph. It operates on factor graphs, but a Bayesian network can easily be translated to a factor graph. If the CPTs in the network are well-distributed, the accuracy of the resulting state estimation will increase and converge to 1 as the number of children of each node increases. On well-distributed trees, the accuracy is guaranteed to be greater than 0.5. The algorithm is very fast, and works well on networks where variables have few states, and many children. On trees with 3 variable states and branching factor of 5, the algorithm scored a 98% similarity to exact inference. It is useful in situations where one is uncertain about the correctness of the precise parameter values, or situations where one requires fast state estimation results.

**Quantized EM**

Quantized EM (Chapter 4) is an algorithm to learn the parameters of a Bayesian network. With this algorithm, we try to be more robust against overfitting on small data sets. It is based on the use of a quantized parameter space. All CPTs that produce the same supports are represented by one quantized value. In this way, precise details are smoothed out. This helps to prevent overfitting, where precise details are tuned to unrepresentative cases in the training data set. The algorithm consists of two phases; the first searches in the quantized space, and the second continues the search in the full space. The algorithm is faster than
comparable algorithms; on some networks, it is about twice as fast as regular EM, and three times as fast as EM with parameter smoothing. It finds models that are at least as good.

The voting belief propagation algorithm can be applied on the rough model learned in the quantized phase. The result is a very fast algorithm, which works well on trees with spiky CPTs (CPTs with a few entries that are much greater than the other entries). It scores 96% similarity to exact inference on the original network, for trees in which the spikes in the CPTs are 0.8. The combination of the two algorithms is useful in situations where one needs preliminary learning and state estimation results.

Perturbed quantized EM enhances the first learning phase with a data perturbation method to decrease the variance in the quantized EM output. For small training data sets, the algorithm finds better models than other methods. This shows that it is more robust against overfitting.

Filtering and Fault Localization

The filtering algorithm (Chapter 5) is a method to detect incorrect state estimations. If the estimated state got less supports than a certain threshold, the estimation gets rejected. The algorithm becomes more effective with increasing branching factors (number of supports). Because it is based on supports, it is insensitive to small changes in parameter values. Optimal rejection thresholds can be determined through ROC curves. On a tree network with a branching factor of 7, the algorithm can find 90% of incorrect state estimations, while producing only 15% false alarms.

Fault localization (Chapter 5) can localize inadequate CPTs in a Bayesian network. It compares the consistency among supports at different nodes in a network, to classify the CPT in between as adequate or inadequate. The percentage of inadequate CPTs that gets found increases with the branching factor and the number of variable states, and the percentage of false positives decreases. Related methods cannot be shown to have these properties. On networks where 20% of the CPTs are inadequate (a plausible situation), the algorithm found 90% of all inadequate CPTs, while producing only 5% false alarms.

6.1 Future Research

The voting algorithm uses messages which contain the vote for the winning state of a variable. However, many different voting schemes are possible [Far01]. A ranked voting scheme ranks the different candidates in order of preference. In the case of the voting algorithm, this could be implemented by incorporating multiple votes in the messages. For example, instead of sending 1 vote corresponding to the winning state, the winning state could get 2 votes, and the state that got in second
place could get 1. This kind of scheme could help against ties that often occur in the case of variables with many states. A multiple vote scheme works similar, but does not distinguish between the winner and the second place; both get an equal number of votes in the message to the neighbor in the factor graph. Proportional voting spreads a fixed number of votes over the states, proportionally to their score in the voting process. These kind of schemes can be easily incorporated in the variable node computations, but might require some adaptation of the factor node computations.

If we are not interested in state estimation, but in the posterior distribution over the states of a variable, one could use a hybrid propagation approach. This approach mixes voting iterations with regular sum-product iterations. By letting the last few iterations be exact iterations, one can obtain a precise distribution. However, the quality of this distribution is unsure, and would require further research. The algorithm will still be faster than regular sum-product.

With respect to the learning algorithm, other quantization choices are possible. It would be interesting to investigate for which kind of data sets the quantized approach has the biggest advantage over regular EM. Adding specific outliers in a data set, one could investigate in which cases the regular EM algorithm gets stuck in local maxima, while the quantized variant avoids them. By varying the number of outliers or their strength, one can gain insight into the likelihood landscape. The knowledge gained through this kind of investigation can also be valuable for learning methods in general.

An accurate model has many fragments supporting the same state. Therefore, it would be interesting to investigate a learning method that optimizes the degree to which fragments agree. In other words, to optimize the consistency measure from Chapter 5. In each iteration, for each data case, we can change the CPTs such that they agree with the majority. Such a method seems to require tree networks to work well, but could speed up learning, and offer a different score to optimize besides the likelihood.
Bibliography


Probabilistic reasoning is an important aspect of many AI applications, in the context of for example sensor fusion or forecasting. Discrete Bayesian networks facilitate such reasoning, and are popular in applications that have to deal with a high degree of uncertainty. Their modular nature and intuitive parameters make them relatively easy to construct by human domain experts.

Inevitably, small mistakes will be made during construction. Human experts can be biased, or have difficulty expressing their knowledge directly into probabilities. Example data used for learning can be incomplete. This begs the question what influence these mistakes have on the reliability of the outcome of probabilistic reasoning. Small errors in the value of a parameter might be admissible in some cases, but not always. This thesis investigates what characterizes networks that are robust against small parameter errors, and how much parameters are allowed to change without harming the accuracy of the network.

In Chapter 3 we show how we can view probabilistic inference as a voting process, with votes coming from separate network fragments. If a small parameter error does not change the votes, then the outcome is likely not to change. This sets limits on the amount of admissible change.

Also in Chapter 3, we present an algorithm for inference with Bayesian networks, based on the voting view. This voting algorithm is very fast, and achieves a good accuracy for several classes of networks.

In Chapter 4, we focus on learning the parameters of a Bayesian network from incomplete example data. We present a learning algorithm that uses the voting view, in the sense that it first attempts to learn a network that gives correct votes, before fine-tuning the parameters. The algorithm turns out to converge fast, and perform better than other algorithms when learning from small data sets. This is an indication that our algorithm is more resistant against local maxima in the search space.

Chapter 5 deals with the issue of ‘adequate’ parameters; parameters that make a network inherently robust. We identify relations between parameters that
should be satisfied, and argue that the correct relations are easy to find. If the parameters in a network are adequate, the votes (mentioned above) are most likely to be correct.

We present two algorithms that are based on this observation. The first uses the number of votes in favor of an outcome as a measure of the confidence in that outcome. The second algorithm compares the votes of different network fragments to locate parameters that are not adequate. Both algorithms are shown to be effective at identifying inaccurate inference outcomes and locating their causes. They can be used to monitor a Bayesian network for errors.
Automatisch redeneren is een belangrijk aspect van veel toepassingen in de kunstmatige intelligentie. Het wordt bijvoorbeeld gebruikt voor het combineren van sensor informatie, het doen van voorspellingen of het stellen van een diagnose. Discrete Bayesiaanse netwerken vereenvoudigen dergelijk redeneren door de kennis over wereld in een gestructureerd kans-model te representeren. Ze zijn populair in toepassingen die te maken hebben met een hoge graad van informatie onzekerheid. Dankzij de modulaire opbouw en intuïtieve parameters (waarschijnlijkheden) zijn ze relatief eenvoudig te construeren door menselijke experts.

Echter, er zullen altijd kleine fouten worden gemaakt in de parameter schattingen. Menselijke experts kunnen bevooroordeeld zijn, of moeite hebben om hun kennis direct uit te drukken in waarschijnlijkheden. Voorbeeld data voor het automatisch leren van de parameters kan onvolledig zijn. Dit roept de vraag op wat de invloed van deze kleine fouten is op de betrouwbaarheid van de uitkomsten van deze modellen. Kleine fouten kunnen in sommige gevallen toegestaan zijn, maar niet altijd. Dit proefschrift onderzoekt wat de eigenschappen zijn van een model dat robuust is tegen kleine parameter fouten, en in hoeverre parameters mogen afwijken zonder dat het de nauwkeurigheid van het model aantast.

In Hoofdstuk 3 laten we zien hoe het redeneringsproces kan worden gezien als een stemproces, waarbij verschillende fragmenten van het netwerk een eigen stem kunnen uitoefenen. Als een kleine afwijking in een parameter de stemkeuze niet verandert, dan verandert de uitkomst van het model waarschijnlijk ook niet. Dit impliceert een bepaalde limiet aan de toegestane hoeveelheid afwijking.

In Hoofdstuk 3 presenteren we ook een nieuw algoritme voor het redeneren met Bayesiaanse netwerken, gebaseerd op het stemproces. Dit algoritme is erg snel en behaalt een goede nauwkeurigheid op een aantal type netwerken.

In Hoofdstuk 4 kijken we naar het leren van de parameter waarden uit voorbeeld data. We presenteren een nieuw algoritme, wederom gebaseerd op het stemproces. Het probeert eerst een netwerk te leren dat de juiste stemmen geeft, alvorens de precieze parameter waarden te leren. Het algoritme convergeert snel,
en vindt betere netwerken dan andere methodes, wanneer het wordt toegepast op een kleine hoeveelheid voorbeeld data. Dit is een indicatie dat ons algoritme beter bestand is tegen lokale maxima in de zoekruimte.

Hoofdstuk 5 gaat over ‘adequate’ parameters; parameters die een netwerk inherent robuust maken. We identificeren bepaalde relaties die moeten gelden om parameters adequaat te maken, en stellen dat die relaties eenvoudig te vinden zijn tijdens de constructie van een model. Als de parameters in een netwerk adequaat zijn, dan zijn de stemmen (hierboven genoemd) meest waarschijnlijk ook juist.

We presenteren twee algoritmes die gebaseerd zijn op deze observatie. Het eerste gebruikt het aantal stemmen dat een uitkomst krijgt, als maat van vertrouwen in die uitkomst. Het tweede algoritme vergelijkt stemmen van verschillende netwerkfragmenten, om parameters te lokaliseren die niet adequaat zijn. Beide algoritmes blijken effectief te zijn in het identificeren van foute uitkomsten en het lokaliseren van hun oorzaken. Ze kunnen worden gebruikt om het redeneringsproces in Bayesiaanse netwerken te behoeden voor foute uitkomsten.
This thesis was based on the following publications:


Other publications included:


Acknowledgments

I would like to thank Frans Groen for supporting my work over the last 5 years, and providing a critical but ever-important ‘outside view’ on my research. Many thanks of course also to my daily supervisor Gregor Pavlin. We had many long discussions about all aspects of the work, which I greatly appreciated, even though we can both be stubborn from time to time.

Thanks goes to my roommates Daniel Fontijne, Patrick de Oude and Sicco van Gosliga for the discussions on our research, and also for the conversations about non-scientific subjects, providing much-needed breaks from work every now and then. The same goes for all other (former) colleagues over the years. I enjoyed the light-hearted lunch conversations we’ve had.

I wouldn’t have been able to come this far without the tremendous love and support from my family and friends. As a PhD student, you always bring your work back home, but they made sure work never got the upper hand. Finally, I would like to thank Anqi for her support and encouragement in the final stages of writing this thesis. It kept me sane. Thanks all! Xièxiè!