Research Note

Irrelevance and parameter learning in Bayesian networks

Nevin Lianwen Zhang *
Department of Computer Science, University of Science & Technology, Clear Water Bay, Kowloon, Hong Kong

Received December 1995; revised July 1996

Abstract

It is possible to learn the parameters of a given Bayesian network structure from data because those parameters influence the probability of observing the data. However, some of the parameters are irrelevant to the probability of observing a particular data case. This paper shows how such irrelevancies can be exploited to speedup various algorithms for parameter learning in Bayesian networks. Experimental results with one of the algorithms, namely the EM algorithm, are presented to demonstrate the gains of this exercise.

Keywords: Bayesian networks; Parameter learning; Irrelevance; Efficiency

1. Introduction

Of concern to this paper are problems of learning relationships among variables from data. The solution of such a problem is usually divided into two steps: (1) selecting a model for the relationships and (2) estimating the parameters in the model using data. Many models exist. There are statistical models such as regression and loglinear models; and there are AI models such as decision trees and artificial neural networks (ANN). Recently, there have been growing interests, from both statisticians and AI researchers, in learning with a new type of models called Bayesian networks.

Bayesian networks ([12], also called belief networks or probabilistic networks) are directed acyclic graphs, where each node represents a random variable and is attached with a conditional probability of the node given its parents. They have been widely

* E-mail: izhang@cs.ust.hk.

0004-3702/96/$15.00 Copyright © 1996 Elsevier Science B.V. All rights reserved.
PH S0004-3702 (96)00035-5
accepted as a suitable knowledge representation framework for reasoning and decision making under uncertainty.

In applications, Bayesian networks are usually constructed from experts' knowledge. Knowledge acquisition, especially assessment of numerical probabilities, is usually difficult and unreliable. There is a need to refine Bayesian networks using data [16], and even to automatically construct Bayesian networks from data (see [4] for an overview). This is one cause for the growing interests in learning Bayesian networks.

A second cause is the realization that Bayesian network learning can potentially serve as an alternative to [13] or complement [6] neural network learning as a general learning paradigm. Since arcs in Bayesian networks can be interpreted as representing causal relationships, experts' qualitative knowledge, which is often readily available and reliable (e.g., it can be from a domain textbook), can be used to aid model selection and reduce the amount of training data.

There is one other advantage of Bayesian network learning that has received little attention up to now. Suppose a Bayesian network structure (model) has been selected and we want to estimate the parameters from data. This is possible because the parameters influence the probability of observing the data. For a particular data case, however, there is, due to the given network structure, a subset of parameters that are irrelevant to the probability of observing the data case [3,10]. Consequently, the data case should not contribute anything to the estimation of those parameters. Such irrelevancies can be exploited to speed up learning.

Exploiting irrelevancies are especially important when there is a large number of variables in the model and there are multiple data sets, each concerning a different subset of variables. In such a case, the number of parameters that are relevant to a data set can be much smaller than the total number of parameters.

This paper examines three types of algorithms for parameter learning in a given Bayesian network model, namely the EM algorithm [8], gradient descent [13], and sequential learning [15]. We show how irrelevancies can be exploited to speed up all those algorithms. Experimental results with the EM algorithm are reported to demonstrate the gains of this exercise.

2. Parameter learning in Bayesian networks

Let $x_1, x_2, \ldots, x_n$ be all the variables in a Bayesian network structure and let $\pi_i$ be the set of parents of $x_i$. This paper assumes that all variables have a finite number of possible values. The network structure is a model about the relationships among the variables; it states that the joint probability $p(x_1, x_2, \ldots, x_n)$ can be factorized into the multiplication of the conditional probabilities of the variables given their respective parents, i.e.,

$$p(x_1, x_2, \ldots, x_n) = \prod_i p(x_i | \pi_i). \tag{1}$$

This paper is concerned with learning conditional probabilities for a given network structure. We shall use $\theta_{ijk}$ to denote $p(x_i = j | \pi_i = k)$, where $j$ is a value of variable $x_i$.
and $k$ is a combination of the values of the parents of $x_i$. For convenience, we shall say that $k$ is a value of $\pi_i$ and call $\theta_{ijk}$ a parameter pertaining to variable $x_i$. We shall also use $\theta$ to denote the vector of all the parameters $\theta_{ijk}$. The joint probability over all the variables determined by a parameter vector $\theta$ will be denoted by $p(x_1, x_2, \ldots, x_n|\theta)$.

The parameter vector $\theta$ is to be estimated from a collection $D$ of data cases $D_1, D_2, \ldots, D_m$ that are independent given $\theta$. Each data case is a set of variable-value pairs. Different data cases might involve different variables. This implies that data cases might be incomplete in the sense that they might not contain values for all variables.

If $\theta$ is the true parameter vector, then the probability of observing $D$ is $p(D|\theta) = \prod_i p(D_i|\theta)$. Viewed as a function of $\theta$, $p(D|\theta)$ is called the likelihood function of $\theta$ given data $D$. The maximum likelihood (ML) estimate of $\theta$ is a value of $\theta$ that maximizes the likelihood function.

When there is a prior probability (density function) $p(\theta)$ about $\theta$, one can compute the posterior probability using Bayes' rule, $p(\theta|D) = \frac{c p(\theta)p(D|\theta)}{\int \theta p(\theta)p(D|\theta)d\theta}$, where $c$ is the renormalization constant. In this case, $\theta_{ijk}$ can be estimated by using the posterior expectation $\int \theta_{ijk} p(\theta_{ijk}|D)d\theta_{ijk}$.

Lauritzen [8] and Russell et al. [13] have respectively explained how the EM algorithm [2] and gradient descent can be used to compute an approximate ML estimate of $\theta$, while Spiegelhalter and Lauritzen [17], Spiegelhalter and Cowell [15], and Titterington [18] have described approximate methods for obtaining the posterior expectation of $\theta_{ijk}$ for every $i, j,$ and $k$. Although very different in appearances, those methods share one thing in common. In obtaining the next estimate of $\theta$ from the current estimate $\theta^{(t)}$, a major part of the computation is spent in calculating the conditional probability

$p(x_i, \pi_i|D_i, \theta^{(t)})$

for every variable $x_i$ and for one or every data case $D_i$. This conditional probability is used to revise the parameters $\theta_{ijk}^{(t)}$ (for all $j$ and all $k$) pertaining to $x_i$.

2.1. Purpose of this paper

For any set $Y$ of variables in a Bayesian network, the ancestral set $\text{an}(Y)$ consists of variables in $Y$ and ancestors of those variables, i.e., variables from which there are directed paths to at least one variable $Y$. It is evident that $x_i \in \text{an}(Y)$ implies $\pi_i \subset \text{an}(Y)$.

Let $Y_i$ be the set of variables whose values are observed in data case $D_i$. The purpose of this paper is to show

**Main result.** In various parameter learning algorithms, there is, for any data case $D_i$, no need to calculate $p(x_i, \pi_i|D_i, \theta^{(t)})$ for variables $x_i$ outside the ancestral set $\text{an}(Y_i)$.

The main result is interesting because it cuts down computation and hence speeds-up learning. The amount of computation is reduced not only because the conditional probability $p(x_i, \pi_i|D_i, \theta^{(t)})$ is calculated only for variables $x_i$ in the ancestral set $\text{an}(Y_i)$,
but also because calculating $p(x_i, \pi_i | D_l, \theta^{(t)})$ for variables $x_i$ outside the ancestral set can be much more difficult than for those in the ancestral set. If one uses the clique tree propagation (CTP) inference algorithm \cite{5,9,14}, the main result ensures that it suffices to construct a clique tree for the restriction of the Bayesian network onto $an(Y_l)$. Without the main result, one would need to construct a clique tree for the entire network. The maximum clique in the latter clique tree can be much larger than its counterpart in the former clique tree and hence inference can take much more time. It is not rare that inference in the entire network is infeasible, while inference in a subnetwork can be done in real time.

The next section will establish the main result for the EM algorithm and report experimental results to demonstrate the efficiency gains that the main result brings about. Section 4 will extend the main result to the gradient descent and the sequential learning algorithms.

3. The EM algorithm

The EM algorithm \cite{2} is a general iterative algorithm for computing approximate ML estimates from incomplete data. Lauritzen \cite{8} has shown how it can be used for parameter learning in Bayesian networks. In this context, major computation at each iteration is spent on calculating the conditional probability $p(x_i, \pi_i | D_l, \theta^{(t)})$ for all data cases $D_l$ and all variables $x_i$. This section describes a modification of the EM algorithm (for Bayesian networks). For each data case $D_l$, the modified algorithm calculates $p(x_i, \pi_i | D_l, \theta^{(t)})$ only for variables $x_i$ in the ancestral set $an(Y_l)$ and hence runs faster than the original algorithm. Moreover, the modified algorithm has a higher rate of convergence than the original algorithm.

3.1. A brief review of the EM algorithm

Let us begin with a brief review of the EM algorithm. The loglikelihood function of $\theta$ given data $D$ is defined by

$$l(\theta|D) = \ln p(D|\theta) = \sum_i \ln p(D_i|\theta).$$

It is evident that maximizing the likelihood function is the same as maximizing the loglikelihood function. It is usually more convenient to work with the latter.

A data case $D_l$ is complete if all variables are observed in the case. From Eq. (1) it is easy to see that when all the data cases are complete, the loglikelihood function is

$$l(\theta|D) = \sum_{i,j,k} f(x_i = j, \pi_i = k) \ln \theta_{ijk}.$$  

where $f(x_i = j, \pi_i = k)$ stands for the number of data cases where $x_i = j$ and $\pi_i = k$. Consequently, the ML estimate $\theta^*$ of $\theta$ can be obtained by setting

$$\theta_{ijk}^* = \frac{f(x_i = j, \pi_i = k)}{\sum_j f(x_i = j, \pi_i = k)} \text{ for all } i, j, \text{ and } k.$$
Often, data cases are incomplete. This is when the EM algorithm comes into play. The algorithm starts with an initial estimate $\theta^{(0)}$ of $\theta$ and improves it iteratively. From the current estimate $\theta^{(t)}$, the next estimate $\theta^{(t+1)}$ is obtained in two steps: the expectation step (E-step) and the maximization step (M-step). The E-step computes the current expected loglikelihood function of $\theta$ given data $D$, i.e.,

$$Q(\theta|\theta^{(t)}) = \sum_{i} \sum_{X_i} \ln p(D_1, X_i|\theta) p(X_i|D_1, \theta^{(t)}),$$

where $X_i$ is the set of variables whose values are missing from data case $D_1$. The M-step chooses the next estimate $\theta^{(t+1)}$ by maximizing the current expected loglikelihood:

$$Q(\theta^{(t+1)}|\theta^{(t)}) \geq Q(\theta|\theta^{(t)}) \quad \text{for all } \theta.$$

By making use of Eq. (1), we obtain that

$$Q(\theta|\theta^{(t)}) = \sum_{i,j,k} f_i(x_i = j, \pi_i = k) \ln \theta_{ijk},$$

where

$$f_i(x_i, \pi_i) = \sum_{j} p(x_i, \pi_i|D_1, \theta^{(t)}).$$

Hence, the E-step reduces to computing the function $f_i(x_i, \pi_i)$ for each variable $x_i$, while the M-step reduces to computing the next estimate of $\theta$ by setting

$$\theta_{ijk}^{(t+1)} = \frac{f_i(x_i = j, \pi_i = k)}{\sum_j f_i(x_i = j, \pi_i = k)} \quad \text{for all } i, j, \text{ and } k.$$

Because of Eq. (2), the distribution $p(D|\theta)$ of complete data is from the regular exponential family with natural parameter $\ln \theta_{ijk}$. This fact implies that the loglikelihood $l(\theta^{(t)}|D)$ of incomplete data increases with $t$ and hence converges [2,19].

From Eq. (5), we see that at each iteration, the EM algorithm needs to calculate the conditional probability $p(x_i, \pi_i|D_1, \theta^{(t)})$ for every data case $D_1$ and every variable $x_i$, and this is the dominant part of the computation.

3.2. Modifying the EM algorithm

Let $Y$ be a subset of variables in a Bayesian network. The joint probability $P(\text{an}(Y))$ of the variables in the ancestral set $\text{an}(Y)$ is given by

$$p(\text{an}(Y_i)) = \prod_{i \text{ s.t. } x_i \in \text{an}(Y_i)} p(x_i|\pi_i).$$

We say a data case $D_1$ is upward complete if $\text{an}(Y_i) = Y_i$. By Eq. (7), we have that when all the data cases are upward complete, the loglikelihood function is

$$l(\theta|D) = \sum_{i,j,k} g(x_i = j, \pi_i = k) \ln \theta_{ijk},$$
where \( g(x_i = j, \pi_i = k) \) stands for the number of data cases \( D_l \) such that \( Y_l \) contains \( x_i \) and that \( x_i = j \) and \( \pi_i = k \). Consequently, the ML estimate of \( \theta \) can be obtained by using Eq. (3) with the \( f \)-function replaced by the \( g \)-function.

The rest of this subsection describes an EM style algorithm for situations where data cases are not upward complete. An intuitive way to understand the EM algorithm is that it makes an incomplete data case \( D_l \) complete by filling in missing data \( X_l \). The algorithm to be described below makes a data case that is not upward complete by filling in missing data.

Let \( \theta^{(t)} \) be the current estimate of \( \theta \). We obtain the next estimate in the following two steps. First, we compute the function

\[
Q'(\theta|\theta^{(t)}) = \sum_l \sum_{Z_l} \ln p(D_l, Z_l|\theta) p(Z_l|D_l, \theta^{(t)}),
\]

where \( Z_l \) is the set of variables in the ancestral set \( an(Y_l) \) whose values are missing from the data case \( D_l \), i.e., \( Z_l = an(Y_l) \setminus Y_l \). Then, we choose the next estimate \( \theta^{(t+1)} \) by maximizing the \( Q'- \)function.

It follows from Eq. (7) that

\[
Q'(\theta|\theta^{(t)}) = \sum_{i,j,k} g_l(x_i, \pi_i) \ln \theta_{ijk},
\]

where

\[
g_l(x_i, \pi_i) = \sum_{\text{s.t. } x_i \in an(Y_l)} p(x_i, \pi_i|D_l, \theta^{(t)}).
\]

Consequently, our first step reduces to computing \( g_l(x_i, \pi_i) \) for all variables \( x_i \), and our second step reduces to obtaining the next estimate \( \theta^{(t+1)} \) by setting

\[
\theta_{ijk}^{(t+1)} = \frac{g_l(x_i = j, \pi_i = k)}{\sum_j g_l(x_i = j, \pi_i = k)} \quad \text{for all } i, j, \text{ and } k.
\]

Because of Eq. (8), the distribution \( p(D|\theta) \) of upward complete data is from the regular exponential family with natural parameters \( \ln \theta_{ijk} \). This fact enables one to prove the convergence of the modified EM algorithm by following the same line of reasoning as in the proof of the convergence of the original EM algorithm [2, 19].

3.3. Implementation, experiments and discussions

3.3.1. Implementation

We have implemented the modified EM and EM based on the version of clique tree propagation (CTP) by Shafer and Shenoy [14]. When implementing the modified EM, we have assumed that data cases are grouped into data sets such that data cases in the same data set involve the same variables. We have also assumed that identical data cases are aggregated.

In our implementation, there is a preprocessing module which computes, for each data set, the ancestral set \( an(Y_l) \) of the set \( Y_l \) of variables involved and constructs a clique
tree structure for the restriction of the Bayesian network onto an(Yi). The clique tree structures are kept static over iterations of the learning algorithm.

Ancestral sets are found by backward search; the complexity is linear in the number of arcs. Clique tree structures are constructed by first obtaining the moral graphs of the restricted Bayesian networks [9], finding elimination orderings by using the minimum deficiency heuristic [1,7], and then constructing the clique tree structures from the orderings. The total complexity is O(kn²), where n is the number of nodes in a restricted Bayesian network and k is the maximum number of neighbors of a node in the moral graph.

All the work in one iteration of the modified EM is carried out by the main module. It consists of three submodules: an initialization submodule, a CTP submodule, and a learning submodule. The initialization submodule initializes each clique tree structure by using the current network parameters. For each node in the Bayesian network, its conditional probability is attached to one clique in the tree, if it exists, that contains the node. If more than one conditional probability is attached to one clique, they are multiplied into one function. If nothing is attached to a clique, the identity function is attached to it. Unlike in the version of CTP by Lauritzen and Spiegelhalter [9] and Jensen et al. [5], marginal probabilities of the cliques are not computed.

The CTP submodule implements the CTP algorithm. It is called once for each data case (with the corresponding clique tree), and it updates the g_c-function according to Eq. (10). Finally, the learning submodule uses Eq. (11) to obtain the network parameters for the next iteration.

The implementation for EM is the same except there is only one clique tree for the entire network.

To make a quick comparison, we see that the modified EM spends more time in the initialization submodule than EM since it needs to initialize potentially many clique tree structures while EM has only one to initialize. Hence EM can possibly outperform the modified EM in some situations. As will be seen in the next subsection, however, the modified EM almost always significantly outperforms EM. There are two reasons. Firstly, the clique trees encountered by the modified EM are smaller (in terms of both the number of cliques and clique sizes) than those encountered by EM. Secondly, the amount of computation in initialization is usually substantially less than the amount of computation in propagation, especially when there are many data cases in a data set. Because of the second reason, EM can seldom significantly outperform the modified EM.

3.3.2. Experiments

Experiments have been performed with two CPCS networks provided by Pradhan to compare the modified EM and EM. One of the networks is a polytree consisting of 40 nodes. The other consists of 145 nodes and contains loops.

Data sets were generated for each network by using original network parameters. The number of data sets, the number of data cases in a data set, and the number of variables in a data case were pre-determined. In generating a data set, a list of variables were first randomly generated. Then the joint probability of those variables were computed and
data cases were randomly generated by using this joint probability. Since the complexity of computing the joint probability is exponential, we have restricted the number of variables to be no larger than 10.

Both EM and the modified EM were called to re-learn the network parameters from the data sets. The algorithms were allowed to run until the loglikelihoods $l(\theta^{(t)}|D)$ at two consecutive iterations do not differ by more than 1.0.

Four experiments were carried out. The setups are described in Table 1. For example, the first experiment was performed in the 40-node network. Fifteen data sets were generated with each data set involving 5 variables and containing 75 data cases.

In all the experiments, the loglikelihoods $l(\theta^{(t)}|D)$ of the parameter vectors $\theta^{(t)}$ produced over time by either EM or the modified EM were recorded and they are summarized in Fig. 1.

We see that when the modified EM is used, the loglikelihood converges quickly. When EM is used, on the other hand, the loglikelihood converges much slower. In all

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Bayesian network</th>
<th>Number of data sets</th>
<th>Number of variables in each data set</th>
<th>Number of data cases in each data set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>the 40-node BN</td>
<td>15</td>
<td>5</td>
<td>75</td>
</tr>
<tr>
<td>2</td>
<td>the 40-node BN</td>
<td>15</td>
<td>10</td>
<td>500</td>
</tr>
<tr>
<td>3</td>
<td>the 145-node BN</td>
<td>50</td>
<td>5</td>
<td>75</td>
</tr>
<tr>
<td>4</td>
<td>the 145-node BN</td>
<td>15</td>
<td>10</td>
<td>500</td>
</tr>
</tbody>
</table>

Fig. 1. Loglikelihoods over time.
Table 2
Time for completing one iteration

<table>
<thead>
<tr>
<th>Experiment</th>
<th>EM</th>
<th>Modified EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.950962</td>
<td>0.094330</td>
</tr>
<tr>
<td>2</td>
<td>4.681479</td>
<td>1.209952</td>
</tr>
<tr>
<td>3</td>
<td>13.697670</td>
<td>1.081957</td>
</tr>
<tr>
<td>4</td>
<td>181.303357</td>
<td>4.822140</td>
</tr>
</tbody>
</table>

Fig. 2. Loglikelihoods over iterations.

experiments EM was able to complete only a few iterations by the time the modified EM had already reached the stopping condition.

3.3.3. Discussions

A closer look reveals that two facts contribute to the better performance of the modified EM. First, it took the modified EM less time to complete one iteration than EM (Table 2). Second, it took the modified EM less iterations to converge than EM (Fig. 2).

The first fact that contributes the better performance of the modified EM can be explained by observing that for each data case \( D_i \), the modified EM calculates the conditional probability \( p(x_i, \pi_i | D_i, \theta^{(i)}) \) only for those variables \( x_i \) in the ancestral set \( an(Y_i) \), while EM calculates it for all variables.

To explain the second fact, we need to recall a well-known fact about EM: the rate of convergence is closely related to the amount of information that is needed to make data
cases complete. More specifically, the greater this amount of information, the slower the rate of convergence [2]. Similarly, the rate of convergence of the modified EM is closely related to the amount of information that is needed to make data cases upward complete: the greater this amount of information, the slower the rate of convergence. Since it takes less information to make data cases upward complete than to make them complete, the modified EM algorithm has a higher rate of convergence than the original EM algorithm.

Before leaving this section, let us point out that the differences in performance between EM and the modified EM in experiments 2 and 4 are not as significant as in experiments 1 and 3. This is due to the fact that data involve more variables in experiments 2 and 4 than in experiments 1 and 3. In general, the more variables in data, the lower the extent to which the modified EM outperforms EM.

4. Other algorithms

We have seen that the main result can significantly improve the performance of the EM algorithm. This section extends the result to the gradient descent and the sequential learning algorithms.

4.1. Gradient descent

Finding an ML estimate of $\theta$ is a constrained nonlinear optimization problem. Each parameter $\theta_{ijk}$ must be between 0 and 1 and for fixed $i$ and $k$, $\sum_j \theta_{ijk}$ must be 1.

The gradient projection method [11] can be used to solve this problem. In this method, the next estimate of $\theta$ is obtained from the current estimate $\theta^{(t)}$ as follows: Find the gradient vector $\nabla_{\theta}$ of $l(\theta|D)$ at $\theta^{(t)}$; project it onto the constraint surface to get the gradient vector $\nabla_{\theta}^t$ along the constraint surface; and take a step in the direction $\nabla_{\theta}^t$, with the step-size determined by line minimization (or rather maximization).

The gradient vector $\nabla_{\theta}$ contains a component $\nabla_{\theta_{ijk}}$ for each parameter $\theta_{ijk}$, which is the partial derivative of $l(\theta|D)$ with respect to $\theta_{ijk}$ at $\theta^{(t)}$. Russell et al. [13] have shown that

$$\nabla_{\theta_{ijk}} = \frac{\partial l(\theta|D)}{\partial \theta_{ijk}} \bigg|_{\theta=\theta^{(t)}} = \frac{f_i(x_i = j, \pi_i = k)}{\theta_{ijk}^{(t)}}.$$  \hspace{1cm} (12)

The projection $\nabla_{\theta}^t$ of $\nabla_{\theta}$ onto the constraint surface is

$$\nabla_{\theta_{ijk}}^t = \frac{f_i(x_i = j, \pi_i = k)}{\theta_{ijk}^{(t)}} - \frac{1}{n_i} \sum_j f_i(x_i = j, \pi_i = k),$$  \hspace{1cm} (13)

where $n_i$ stands for the number of possible values of $x_i$.

In each iteration, the conditional probability $p(x_i, \pi_i|D_t, \theta^{(t)})$ needs to be computed for every data case $D_t$ and for every variable $x_i$, and this is a major part of the com-
putation. The following proposition states that for each data case $D_l$, $p(x_i, \pi_i|D_l, \theta^{(t)})$ needs to be computed only for variables $x_i$ in the ancestral set $an(Y_l)$.

**Proposition 1.** The quantity on the right-hand side of Eq. (13) remains unchanged if the function $f_i$ is replaced by the function $g_i$.

**Lemma 2.** Let $Y$ be a subset of variables in a Bayesian network and $D$ be a data case. For any $x_i \notin an(Y)$,

$$p(x_i, \pi_i|D) = p(x_i|\pi_i)p(\pi_i|D).$$

**Proof.** The lemma follows from the fact that

$$p(an(\{x_i\} \cup Y)) = p(x_i|\pi_i)p(an(\pi_i \cup Y)).$$

**Proof of Proposition 1.** By Lemma 2, we have that for any data case $D_l$ and any $x_i \notin an(Y_l)$,

$$p(x_i = j, \pi_i = k|D_l, \theta^{(t)}) = \frac{1}{n_i} \sum_j p(x_i = j, \pi_i = k|D_l, \theta^{(t)})$$

$$= \frac{p(x_i = j|\pi_i = k, \theta^{(t)})}{\theta_{ijk}^{(t)}} p(\pi_i = k|D_l, \theta^{(t)})$$

$$- \frac{1}{n_i} \sum_j \frac{p(x_i = j|\pi_i = k, \theta^{(t)})}{\theta_{ijk}^{(t)}} p(\pi_i = k|D_l, \theta^{(t)})$$

$$= p(\pi_i = k|D_l, \theta^{(t)}) - \frac{1}{n_i} \sum_j p(\pi_i = k|D_l, \theta^{(t)})$$

$$= 0.$$

Hence, the proposition follows.

Russell et al. [13] use an adaptation of the conjugate gradient algorithm, instead of the gradient projection method. There is not enough information in the paper for us to determine whether replacing $f_i$ with $g_i$ there would affect the algorithm behavior in terms of convergence; and if yes, whether the effect is positive or negative.

4.2. Sequential learning

Sequential learning methods [15,17,18] approximate the posterior expectation $\int \theta_{ijk}(\theta_{ijk}|D)d\theta_{ijk}$ for every $i$, $j$, and $k$. They absorb data cases one by one. Suppose $\theta^{(t)}$ is the estimate after absorbing the first $l$ data cases. In absorbing data case $D_{l+1}$, they compute the conditional probability $p(x_i, \pi_i|D_{l+1}, \theta^{(t)})$ for every variable $x_i$. This section shows that the conditional probability needs to be computed only for variables in the ancestral set $an(Y_{l+1})$. 
To understand the sequential learning methods, let us inductively define
\[ p_l(\theta) = cp(D_l|\theta)p_{l-1}(\theta), \quad l = 0, 1, \ldots, m, \]
where \( c \) is the renormalization constant and when \( l = 0 \), \( p_0(\theta) \) is simply the prior probability \( p(\theta) \). Then \( p_m(\theta) = p(\theta|D) \).

Due to the large number of parameters encapsulated in the parameter vector \( \theta \), straightforward computation of \( p_m \) is unrealistic. Simplifying assumptions, and hence approximations, are usually made.

Let \( \theta_{ij} \) denote the collection of all the parameters pertaining to variable \( x_i \), i.e., \( \theta_{ij} = \{\theta_{ijk} \mid \text{for all } j \text{ and } k\} \). The \textit{global independence assumption} states that the parameter groups \( \theta_i \) for different \( i \) are independent under probability \( p_l \), i.e., \( p_l(\theta_i) = \prod_i p_l(\theta_i) \). When it is not the case, \( p_l(\theta) \) is approximated by \( \prod_i p_l(\theta_i) \).

For any \( i \) and \( k \), define \( \theta_{ik} \) to be the collection of parameters \( \{\theta_{ijk} \mid \text{for all } j\} \). The \textit{local independence assumption} states that for any fixed \( i \), the parameter groups \( \theta_{ik} \) for different \( k \) are independent under probability \( p_l \), i.e., \( p_l(\theta_i) = \prod_k p_l(\theta_{ik}) \). When it is not the case, \( p_l(\theta_i) \) is approximated by \( \prod_k p_l(\theta_{ik}) \).

The third assumption, which might be called the \textit{Dirichlet distribution assumption}, states that the marginal probability \( p_l(\theta_{ik}) \) is of a Dirichlet distribution \( \mathcal{D}[\alpha_{ik}^{(1)}, \ldots, \alpha_{in,k}^{(1)}] \), where \( n_i \) stands for the number of possible values of \( x_i \). When it is not the case, \( p_l(\theta_{ik}) \) is approximated by a Dirichlet distribution.

The parameters \( \alpha_{ijk}^{(1)} \) in the Dirichlet distribution can be thought of as representing counts of past cases where \( x_i = j \) and \( \pi_i = k \). The estimate \( \theta_{ijk}^{(1)} \) can be obtained from the counts by
\[
\theta_{ijk}^{(1)} = \frac{\alpha_{ijk}^{(1)}}{\alpha_{ik}^{(1)}},
\]
where \( \alpha_{ik}^{(1)} = \sum_j \alpha_{ijk}^{(1)} \) is called the "precision" underlying our belief about \( \theta_{ijk}^{(1)} \). The quantity \( \alpha_{ik}^{(1)}/\alpha_{ik}^{(1)} \) is actually the mean of \( \theta_{ijk}^{(1)} \).

Under those three assumptions, the probability \( p_l(\theta) \) is completely determined by the counts \( \alpha_{ijk}^{(1)} \). Hence, computing \( p_{l+1}(\theta) \) from \( p_l(\theta) \) reduces to obtaining the counts \( \alpha_{ijk}^{(1)+1} \) from the counts \( \alpha_{ijk}^{(1)} \). We shall refer to this task as count updating.

Spiegelhalter and Lauritzen [17] have shown that
\[
p_{l+1}(\theta_{ik}) = \sum_j p_l(\theta_{ik}|x_i = j, \pi_i = k)p(x_i = j, \pi_i = k|D_{l+1}, \theta^{(1)}) + p_l(\theta_{ik})(1 - p(\pi_i = k|D_{l+1}, \theta^{(1)})),
\]
where, by the standard conjugate Bayesian updating, \( p_l(\theta_{ik}|x_i - j, \pi_i - k) \) is of the Dirichlet distribution
\[ \mathcal{D}[\alpha_{ik}^{(1)}, \ldots, \alpha_{ijk}^{(1)+1}, \ldots, \alpha_{in,k}^{(1)}]. \]

Eq. (15) reduces count updating to approximating the distribution \( p_{l+1}(\theta_{ik}) \) by a Dirichlet distribution. This can be done in several ways. In the fractional updating method [18], \( p_{l+1}(\theta_{ik}) \) is approximated by the Dirichlet distribution with counts
The probabilistic editor method [15] approximates $p_{l+1}(\theta_{ik})$ by the Dirichlet distribution with counts

$$
\alpha_{ik}^{l+1} = \alpha_{ik}^l + m_{ijk} \quad \text{for all } j,
$$

where

$$
m_{ijk} = \frac{\alpha_{ijk}^l + P_{ijk}}{\alpha_{ik}^l + 1},
$$

and where $p_{ijk}$ is a shorthand for $p(x_i = j, \pi_i = k | D_{l+1}, \theta^{(l)})$.

A common theme in those methods is that the conditional probability $p(x_i = j, \pi_i = k | D_{l+1}, \theta^{(l)})$ is computed for every variable $X_i$ and all the counts are updated. The following proposition states that one needs only update the counts pertaining to variables in the ancestral set $an(Y_{i+1})$, and consequently $p(x_i = j, \pi_i = k | D_{l+1}, \theta^{(l)})$ needs to be computed only for those variables.

**Proposition 3.** For any $x_i \notin an(Y_{i+1})$,

$$
p_{l+1}(\theta_{ik}) = p_l(\theta_{ik}).
$$

**Proof.** By inspecting the pdf of a Dirichlet distribution, one can easily see that

$$
p_l(\theta_{ik} | x_i = j, \pi_i = k) = p_l(\theta_{ik}) \frac{\alpha_{ik}^l}{\alpha_{ijk}^l} \theta_{ijk}.
$$

By Lemma 2, we get

$$
p(x_i = j, \pi_i = k | D_{l+1}, \theta^{(l)}) = p(x_i = j | \pi_i = k, \theta^{(l)}) p(\pi_i = k | D_{l+1}, \theta^{(l)})
\quad = \frac{\alpha_{ijk}^l}{\alpha_{ik}^l} p(\pi_i = k | D_{l+1}, \theta^{(l)}) .
$$

Thus, the proposition follows from the fact that $\sum_j \theta_{ijk} = 1$. $\square$

We would like to point out that the proposition remains true even without the local independence and the Dirichlet distribution assumptions.

5. Conclusions and future directions

It is possible to learn parameters for a Bayesian network structure from data because those parameters influence the probability of observing the data. For a particular data case, one can determine, from the given network structure, a subset of parameters that
are irrelevant to the probability of observing the data case. Consequently, the data case should not contribute anything in learning those parameters. This paper has shown how such irrelevancies can be exploited to speedup various algorithms for parameter learning in Bayesian networks. Efficiency gains have been empirically demonstrated with the EM algorithm.

The amount of efficiency gains is closely related to the number of variables in data. The fewer the number of variables in data, the more significant the efficiency gains. It would be interesting to investigate if and when a data set that involves many variables can be approximated by data sets with less variables. It would also be interesting to determine whether irrelevance has a role to play in structure learning.

Acknowledgements

We thank M. Pradhan for providing the CPCS networks and the anonymous reviewers for insightful comments and suggestions. This paper has benefited from the AI and Statistics group at Hong Kong University of Science and Technology. Special thanks go to D.Y. Yeung. Research was supported by Hong Kong Research Grant Council grant HKUST658/95E.

References


