26 *Graphical model structure learning*

26.1 Introduction

We have seen how graphical models can be used to express conditional independence assumptions between variables. In this chapter, we discuss how to learn the structure of the graphical model itself. That is, we want to compute $p(G|\mathcal{D})$, where G is the graph structure, represented as an $V \times V$ adjacency matrix.

As we discussed in Section 1.3.3, there are two main applications of structure learning: knowledge discovery and density estimation. The former just requires a graph topology, whereas the latter requires a fully specified model.

The main obstacle in structure learning is that the number of possible graphs is exponential in the number of nodes: a simple upper bound is $O(2^{V(V-1)/2})$. Thus the full posterior $p(G|\mathcal{D})$ is prohibitively large: even if we could afford to compute it, we could not even store it. So we will seek appropriate summaries of the posterior. These summary statistics depend on our task.

If our goal is knowledge discovery, we may want to compute posterior edge marginals, $p(G_{st} = 1|\mathcal{D})$; we can then plot the corresponding graph, where the thickness of each edge represents our confidence in its presence. By setting a threshold, we can generate a sparse graph, which can be useful for visualization purposes (see Figure 1.11).

If our goal is density estimation, we may want to compute the MAP graph, $\hat{G} \in \text{argmax}_G\, p(G|\mathcal{D})$. In most cases, finding the globally optimal graph will take exponential time, so we will use discrete optimization methods such as heuristic search. However, in the case of trees, we can find the globally optimal graph structure quite efficiently using exact methods, as we discuss in Section 26.3.

If density estimation is our only goal, it is worth considering whether it would be more appropriate to learn a latent variable model, which can capture correlation between the visible variables via a set of latent common causes (see Chapters 12 and 27). Such models are often easier to learn and, perhaps more importantly, they can be applied (for prediction purposes) much more efficiently, since they do not require performing inference in a learned graph with potentially high treewidth. The downside with such models is that the latent factors are often unidentifiable, and hence hard to interpret. Of course, we can combine graphical model structure learning and latent variable learning, as we will show later in this chapter.

In some cases, we don't just want to model the observed correlation between variables; instead, we want to model the *causal* structure behind the data, so we can predict the effects of manipulating variables. This is a much more challenging task, which we briefly discuss in

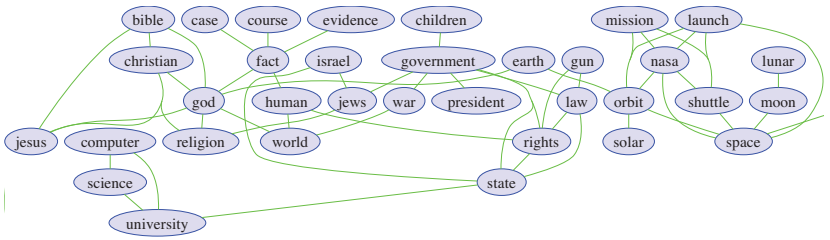


Figure 26.1 Part of a relevance network constructed from the 20-news data shown in Figure 1.2. We show edges whose mutual information is greater than or equal to 20% of the maximum pairwise MI. For clarity, the graph has been cropped, so we only show a subset of the nodes and edges. Figure generated by `relevanceNetworkNewsgroupDemo`.

Section 26.6.

26.2 Structure learning for knowledge discovery

Since computing the MAP graph or the exact posterior edge marginals is in general computationally intractable (Chickering 1996), in this section we discuss some "quick and dirty" methods for learning graph structures which can be used to visualize one's data. The resulting models do not constitute consistent joint probability distributions, so they cannot be used for prediction, and they cannot even be formally evaluated in terms of goodness of fit. Nevertheless, these methods are a useful ad hoc tool to have in one's data visualization toolbox, in view of their speed and simplicity.

26.2.1 Relevance networks

A **relevance network** is a way of visualizing the pairwise mutual information between multiple random variables: we simply choose a threshold and draw an edge from node i to node j if $\mathbb{I}(X_i; X_j)$ is above this threshold. In the Gaussian case, $\mathbb{I}(X_i; X_j) = -\frac{1}{2}\log(1-\rho_{ij}^2)$, where ρ_{ij} is the correlation coefficient (see Exercise 2.13), so we are essentially visualizing $\boldsymbol{\Sigma}$; this is known as the covariance graph (Section 19.4.4.1).

This method is quite popular in systems biology (Margolin et al. 2006), where it is used to visualize the interaction between genes. The trouble with biological examples is that they are hard for non-biologists to understand. So let us instead illustrate the idea using natural language text. Figure 26.1 gives an example, where we visualize the MI between words in the newsgroup dataset from Figure 1.2. The results seem intuitively reasonable.

However, relevance networks suffer from a major problem: the graphs are usually very dense, since most variables are dependent on most other variables, even after thresholding the MIs. For example, suppose X_1 directly influences X_2 which directly influences X_3 (e.g., these form components of a signalling cascade, $X_1 - X_2 - X_3$). Then X_1 has non-zero MI with X_3 (and vice versa), so there will be a $1-3$ edge in the relevance network. Indeed, most pairs will be

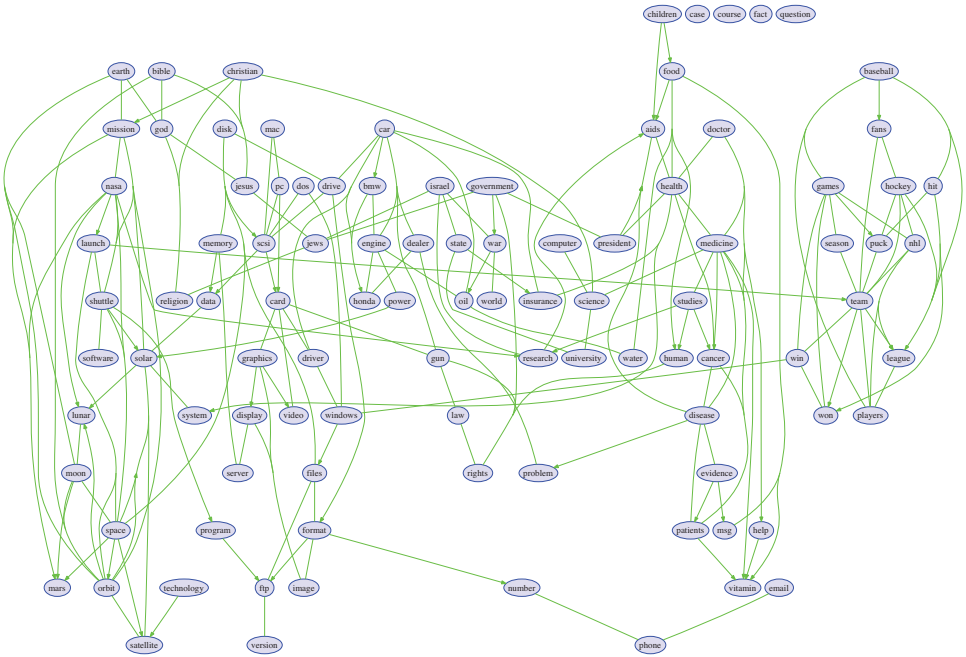


Figure 26.2 A dependency network constructed from the 20-news data. We show all edges with regression weight above 0.5 in the Markov blankets estimated by ℓ_1 penalized logistic regression. Undirected edges represent cases where a directed edge was found in both directions. From Figure 4.9 of (Schmidt 2010). Used with kind permission of Mark Schmidt.

connected.

A better approach is to use graphical models, which represent conditional *independence*, rather than *dependence*. In the above example, X_1 is conditionally independent of X_3 given X_2, so there will not be a $1-3$ edge. Consequently graphical models are usually much sparser than relevance networks, and hence are a more useful way of visualizing interactions between multiple variables.

26.2.2 Dependency networks

A simple and efficient way to learn a graphical model structure is to independently fit D sparse full-conditional distributions $p(x_t|\mathbf{x}_{-t})$; this is called a **dependency network** (Heckerman et al. 2000). The chosen variables constitute the inputs to the node, i.e., its Markov blanket. We can then visualize the resulting sparse graph. The advantage over relevance networks is that redundant variables will not be selected as inputs.

We can use any kind of sparse regression or classification method to fit each CPD. (Heckerman et al. 2000) uses classification/ regression trees, (Meinshausen and Buhlmann 2006) use ℓ_1-regularized linear regression, (Wainwright et al. 2006) use ℓ_1-regularized logistic regression (see `depnetFit` for some code), (Dobra 2009) uses Bayesian variable selection, etc. (Meinshausen

and Buhlmann 2006) discuss theoretical conditions under which ℓ_1-regularized linear regression can recover the true graph structure, assuming the data was generated from a sparse Gaussian graphical model.

Figure 26.2 shows a dependency network that was learned from the 20-newsgroup data using ℓ_1 regularized logistic regression, where the penalty parameter λ was chosen by BIC. Many of the words present in these estimated Markov blankets represent fairly natural associations (aids:disease, baseball:fans, bible:god, bmw:car, cancer:patients, etc.). However, some of the estimated statistical dependencies seem less intuitive, such as baseball:windows and bmw:christian. We can gain more insight if we look not only at the sparsity pattern, but also the values of the regression weights. For example, here are the incoming weights for the first 5 words:

- **aids**: children (0.53), disease (0.84), fact (0.47), health (0.77), president (0.50), research (0.53)
- **baseball**: *christian* (-0.98), *drive* (-0.49), games (0.81), *god* (-0.46), *government* (-0.69), hit (0.62), *memory* (-1.29), players (1.16), season (0.31), *software* (-0.68), *windows* (-1.45)
- **bible**: *car* (-0.72), *card* (-0.88), christian (0.49), fact (0.21), god (1.01), jesus (0.68), orbit (0.83), *program* (-0.56), religion (0.24), version (0.49)
- **bmw**: car (0.60), *christian* (-11.54), engine (0.69), *god* (-0.74), *government* (-1.01), *help* (-0.50), *windows* (-1.43)
- **cancer**: disease (0.62), medicine (0.58), patients (0.90), research (0.49), studies (0.70)

Words in italic red have negative weights, which represents a dissociative relationship. For example, the model reflects that baseball:windows is an unlikely combination. It turns out that most of the weights are negative (1173 negative, 286 positive, 8541 zero) in this model.

In addition to visualizing the data, a dependency network can be used for inference. However, the only algorithm we can use is Gibbs sampling, where we repeatedly sample the nodes with missing values from their full conditionals. Unfortunately, a product of full conditionals does not, in general, constitute a representation of any valid joint distribution (Heckerman et al. 2000), so the output of the Gibbs sampler may not be meaningful. Nevertheless, the method can sometimes give reasonable results if there is not much missing data, and it is a useful method for data imputation (Gelman and Raghunathan 2001). In addition, the method can be used as an initialization technique for more complex structure learning methods that we discuss below.

26.3 Learning tree structures

For the rest of this chapter, we focus on learning fully specified joint probability models, which can be used for density estimation, prediction and knowledge discovery.

Since the problem of structure learning for general graphs is NP-hard (Chickering 1996), we start by considering the special case of trees. Trees are special because we can learn their structure efficiently, as we disuscs below, and because, once we have learned the tree, we can use them for efficient exact inference, as discussed in Section 20.2.

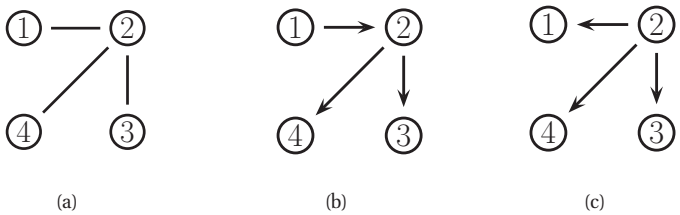


Figure 26.3 An undirected tree and two equivalent directed trees.

26.3.1 Directed or undirected tree?

Before continuing, we need to discuss the issue of whether we should use directed or undirected trees. A directed tree, with a single root node r, defines a joint distribution as follows:

$$p(\mathbf{x}|T) = \prod_{t \in V} p(x_t|x_{\mathrm{pa}(t)}) \tag{26.1}$$

where we define $\mathrm{pa}(r) = \emptyset$. For example, in Figure 26.3(b-c), we have

$$\begin{aligned} p(x_1, x_2, x_3, x_4|T) &= p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_2) & (26.2)\\ &= p(x_2)p(x_1|x_2)p(x_3|x_2)p(x_4|x_2) & (26.3)\end{aligned}$$

We see that the choice of root does not matter: both of these models are equivalent.

To make the model more symmetric, it is preferable to use an undirected tree. This can be represented as follows:

$$p(\mathbf{x}|T) = \prod_{t \in V} p(x_t) \prod_{(s,t) \in E} \frac{p(x_s, x_t)}{p(x_s)p(x_t)} \tag{26.4}$$

where $p(x_s, x_t)$ is an edge marginal and $p(x_t)$ is a node marginal. For example, in Figure 26.3(a) we have

$$p(x_1, x_2, x_3, x_4|T) = p(x_1)p(x_2)p(x_3)p(x_4)\frac{p(x_1, x_2)p(x_2, x_3)p(x_2, x_4)}{p(x_1)p(x_2)p(x_2)p(x_3)p(x_2)p(x_4)} \tag{26.5}$$

To see the equivalence with the directed representation, let us cancel terms to get

$$\begin{aligned} p(x_1, x_2, x_3, x_4|T) &= p(x_1, x_2)\frac{p(x_2, x_3)}{p(x_2)}\frac{p(x_2, x_4)}{p(x_2)} & (26.6)\\ &= p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_2) & (26.7)\\ &= p(x_2)p(x_1|x_2)p(x_3|x_2)p(x_4|x_2) & (26.8)\end{aligned}$$

where $p(x_t|x_s) = p(x_s, x_t)/p(x_s)$.

Thus a tree can be represented as either an undirected or directed graph: the number of parameters is the same, and hence the complexity of learning is the same. And of course, inference is the same in both representations, too. The undirected representation, which is symmetric, is useful for structure learning, but the directed representation is more convenient for parameter learning.

26.3.2 Chow-Liu algorithm for finding the ML tree structure

Using Equation 26.4, we can write the log-likelihood for a tree as follows:

$$\begin{aligned}\log p(\mathcal{D}|\boldsymbol{\theta}, T) &= \sum_t \sum_k N_{tk} \log p(x_t = k|\boldsymbol{\theta}) \\ &+ \sum_{s,t} \sum_{j,k} N_{stjk} \log \frac{p(x_s = j, x_t = k|\boldsymbol{\theta})}{p(x_s = j|\boldsymbol{\theta}) p(x_t = k|\boldsymbol{\theta})} \end{aligned} \tag{26.9}$$

where N_{stjk} is the number of times node s is in state j and node t is in state k, and N_{tk} is the number of times node t is in state k. We can rewrite these counts in terms of the empirical distribution: $N_{stjk} = N p_{\text{emp}}(x_s = j, x_t = k)$ and $N_{tk} = N p_{\text{emp}}(x_t = k)$. Setting $\boldsymbol{\theta}$ to the MLEs, this becomes

$$\frac{\log p(\mathcal{D}|\boldsymbol{\theta}, T)}{N} = \sum_{t \in \mathcal{V}} \sum_k p_{\text{emp}}(x_t = k) \log p_{\text{emp}}(x_t = k) \tag{26.10}$$

$$+ \sum_{(s,t) \in \mathcal{E}(T)} \mathbb{I}(x_s, x_t|\hat{\boldsymbol{\theta}}_{st}) \tag{26.11}$$

where $\mathbb{I}(x_s, x_t|\hat{\boldsymbol{\theta}}_{st}) \geq 0$ is the mutual information between x_s and x_t given the empirical distribution:

$$\mathbb{I}(x_s, x_t|\hat{\boldsymbol{\theta}}_{st}) = \sum_j \sum_k p_{\text{emp}}(x_s = j, x_t = k) \log \frac{p_{\text{emp}}(x_s = j, x_t = k)}{p_{\text{emp}}(x_s = j) p_{\text{emp}}(x_t = k)} \tag{26.12}$$

Since the first term in Equation 26.11 is independent of the topology T, we can ignore it when learning structure. Thus the tree topology that maximizes the likelihood can be found by computing the maximum weight spanning tree, where the edge weights are the pairwise mutual informations, $\mathbb{I}(y_s, y_t|\hat{\boldsymbol{\theta}}_{st})$. This is called the **Chow-Liu algorithm** (Chow and Liu 1968).

There are several algorithms for finding a max spanning tree (MST). The two best known are Prim's algorithm and Kruskal's algorithm. Both can be implemented to run in $O(E \log V)$ time, where $E = V^2$ is the number of edges and V is the number of nodes. See e.g., (Sedgewick and Wayne 2011, 4.3) for details. Thus the overall running time is $O(NV^2 + V^2 \log V)$, where the first term is the cost of computing the sufficient statistics.

Figure 26.4 gives an example of the method in action, applied to the binary 20 newsgroups data shown in Figure 1.2. The tree has been arbitrarily rooted at the node representing "email". The connections that are learned seem intuitively reasonable.

26.3.3 Finding the MAP forest

Since all trees have the same number of parameters, we can safely used the maximum likelihood score as a model selection criterion without worrying about overfitting. However, sometimes we may want to fit a **forest** rather than a single tree, since inference in a forest is much faster than in a tree (we can run belief propagation in each tree in the forest in parallel). The MLE criterion will never choose to omit an edge. However, if we use the marginal likelihood or a penalized likelihood (such as BIC), the optimal solution may be a forest. Below we give the details for the marginal likelihood case.

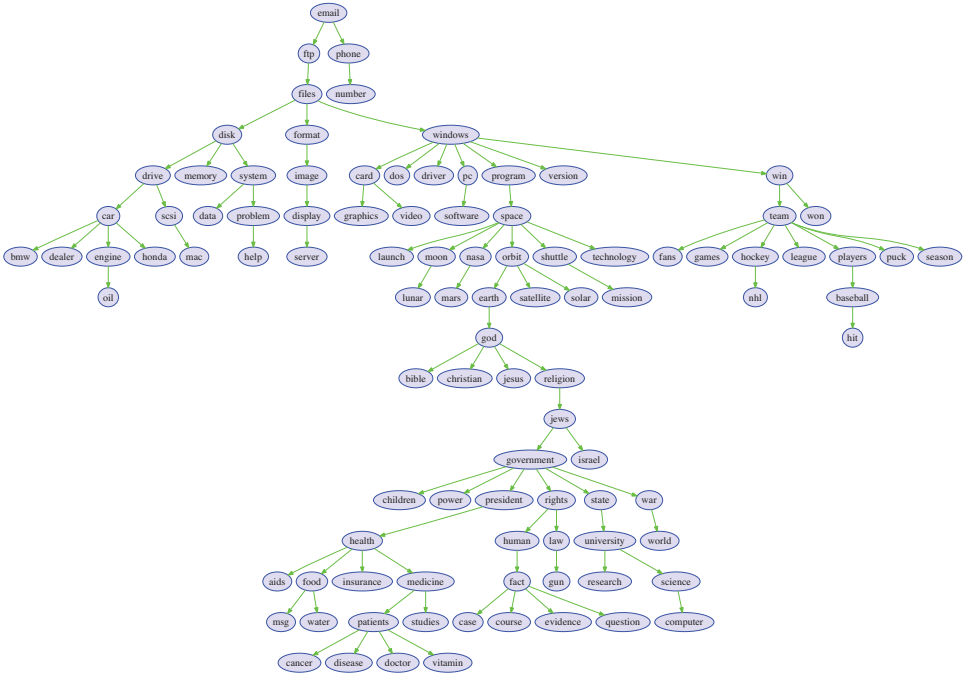


Figure 26.4 The MLE tree on the 20-newsgroup data. From Figure 4.11 of (Schmidt 2010). Used with kind permission of Mark Schmidt. (A topologically equivalent tree can be produced using `chowliuTreeDemo`.)

In Section 26.4.2.2, we explain how to compute the marginal likelihood of any DAG using a Dirichlet prior for the CPTs. The resulting expression can be written as follows:

$$\log p(\mathcal{D}|T) = \sum_{t \in \mathcal{V}} \log \int \prod_{i=1}^{N} p(x_{it}|\mathbf{x}_{i,\mathrm{pa}(t)}|\boldsymbol{\theta}_t) p(\boldsymbol{\theta}_t) d\boldsymbol{\theta}_t = \sum_t \mathrm{score}(\mathbf{N}_{t,\mathrm{pa}(t)}) \tag{26.13}$$

where $\mathbf{N}_{t,\mathrm{pa}(t)}$ are the counts (sufficient statistics) for node t and its parents, and score is defined in Equation 26.28.

Now suppose we only allow DAGs with at most one parent. Following (Heckerman et al. 1995, p227), let us associate a weight with each $s \to t$ edge, $w_{s,t} \triangleq \mathrm{score}(t|s) - \mathrm{score}(t|0)$, where $\mathrm{score}(t|0)$ is the score when t has no parents. Note that the weights might be negative (unlike the MLE case, where edge weights are aways non-negative because they correspond to mutual information). Then we can rewrite the objective as follows:

$$\log p(\mathcal{D}|T) = \sum_t \mathrm{score}(t|\mathrm{pa}(t)) = \sum_t w_{\mathrm{pa}(t),t} + \sum_t \mathrm{score}(t|0) \tag{26.14}$$

The last term is the same for all trees T, so we can ignore it. Thus finding the most probable tree amounts to finding a **maximal branching** in the corresponding weighted directed graph. This can be found using the algorithm in (Gabow et al. 1984).

If the scoring function is prior and likelihood equivalent (these terms are explained in Section 26.4.2.3), we have

$$\text{score}(s|t) + \text{score}(t|0) = \text{score}(t|s) + \text{score}(s|0) \tag{26.15}$$

and hence the weight matrix is symmetric. In this case, the maximal branching is the same as the maximal weight forest. We can apply a slightly modified version of the MST algorithm to find this (Edwards et al. 2010). To see this, let $G = (V, E)$ be a graph with both positive and negative edge weights. Now let G' be a graph obtained by omitting all the negative edges from G. This cannot reduce the total weight, so we can find the maximum weight forest of G by finding the MST for each connected component of G'. We can do this by running Kruskal's algorithm directly on G': there is no need to find the connected components explicitly.

26.3.4 Mixtures of trees

A single tree is rather limited in its expressive power. Later in this chapter we discuss ways to learn more general graphs. However, the resulting graphs can be expensive to do inference in. An interesting alternative is to learn a **mixture of trees** (Meila and Jordan 2000), where each mixture component may have a different tree topology. This is like an unsupervised version of the TAN classifier discussed in Section 10.2.1. We can fit a mixture of trees by using EM: in the E step, we compute the responsibilities of each cluster for each data point, and in the M step, we use a weighted version of the Chow-Liu algorithm. See (Meila and Jordan 2000) for details.

In fact, it is possible to create an "infinite mixture of trees", by integrating out over all possible trees. Remarkably, this can be done in V^3 time using the matrix tree theorem. This allows us to perform exact Bayesian inference of posterior edge marginals etc. However, it is not tractable to use this infinite mixture for inference of hidden nodes. See (Meila and Jaakkola 2006) for details.

26.4 Learning DAG structures

In this section, we discuss how to compute (functions of) $p(G|\mathcal{D})$, where G is constrained to be a DAG. This is often called **Bayesian network structure learning**. In this section, we assume there is no missing data, and that there are no hidden variables. This is called the **complete data assumption**. For simplicity, we will focus on the case where all the variables are categorical and all the CPDs are tables, although the results generalize to real-valued data and other kinds of CPDs, such as linear-Gaussian CPDs.

Our presentation is based in part on (Heckerman et al. 1995), although we will follow the notation of Section 10.4.2. In particular, let $x_{it} \in \{1, \ldots, K_t\}$ be the value of node t in case i, where K_t is the number of states for node t. Let $\theta_{tck} \triangleq p(x_t = k|\mathbf{x}_{\text{pa}(t)} = c)$, for $k = 1 : K_t$, and $c = 1 : C_t$, where C_t is the number of parent combinations (possible conditioning cases). For notational simplicity, we will often assume $K_t = K$, so all nodes have the same number of states. We will also let $d_t = \dim(\text{pa}(t))$ be the degree or fan-in of node t, so that $C_t = K^{d_t}$.

26.4.1 Markov equivalence

In this section, we discuss some fundamental limits to our ability to learn DAG structures from data.

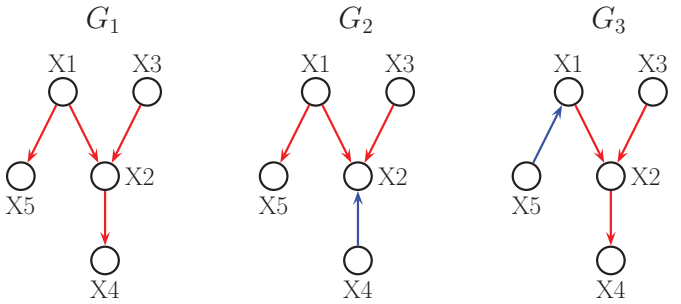


Figure 26.5 Three DAGs. G_1 and G_3 are Markov equivalent, G_2 is not.

Consider the following 3 DGMs: $X \to Y \to Z$, $X \leftarrow Y \leftarrow Z$ and $X \leftarrow Y \to Z$. These all represent the same set of CI statements, namely

$$X \perp Z|Y, \quad X \not\perp Z \tag{26.16}$$

We say these graphs are **Markov equivalent**, since they encode the same set of CI assumptions. That is, they all belong to the same Markov **equivalence class**. However, the v-structure $X \to Y \leftarrow Z$ encodes $X \perp Z$ and $X \not\perp Z|Y$, which represents the opposite set of CI assumptions.

One can prove the following theorem.

Theorem 26.4.1 (Verma and Pearl (Verma and Pearl 1990)). *Two structures are Markov equivalent iff they have the same undirected skeleton and the same set of v-structures.*

For example, referring to Figure 26.5, we see that $G_1 \not\equiv G_2$, since reversing the $2 \to 4$ arc creates a new v-structure. However, $G_1 \equiv G_3$, since reversing the $1 \to 5$ arc does not create a new v-structure.

We can represent a Markov equivalence class using a single **partially directed acyclic graph** (PDAG), also called an **essential graph** or **pattern**, in which some edges are directed and some undirected. The undirected edges represent reversible edges; any combination is possible so long as no new v-structures are created. The directed edges are called **compelled edges**, since changing their orientation would change the v-structures and hence change the equivalence class. For example, the PDAG $X - Y - Z$ represents $\{X \to Y \to Z, X \leftarrow Y \leftarrow Z, X \leftarrow Y \to Z\}$ which encodes $X \not\perp Z$ and $X \perp Z|Y$. See Figure 26.6.

The significance of the above theorem is that, when we learn the DAG structure from data, we will not be able to uniquely identify all of the edge directions, even given an infinite amount of data. We say that we can learn DAG structure "up to Markov equivalence". This also cautions us not to read too much into the meaning of particular edge orientations, since we can often change them without changing the model in any observable way.

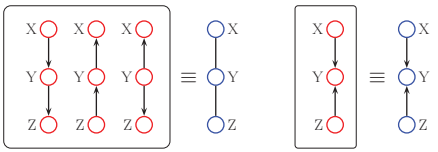


Figure 26.6 PDAG representation of Markov equivalent DAGs.

26.4.2 Exact structural inference

In this section, we discuss how to compute the exact posterior over graphs, $p(G|\mathcal{D})$, ignoring for now the issue of computational tractability.

26.4.2.1 Deriving the likelihood

Assuming there is no missing data, and that all CPDs are tabular, the likelihood can be written as follows:

$$p(\mathcal{D}|G,\boldsymbol{\theta}) = \prod_{i=1}^{N}\prod_{t=1}^{V} \mathrm{Cat}(x_{it}|\mathbf{x}_{i,\mathrm{pa}(t)},\boldsymbol{\theta}_t) \tag{26.17}$$

$$= \prod_{i=1}^{N}\prod_{t=1}^{V}\prod_{c=1}^{C_t} \mathrm{Cat}(x_{it}|\boldsymbol{\theta}_{tc})^{\mathbb{I}(\mathbf{x}_{i,\mathrm{pa}(t)}=c)} \tag{26.18}$$

$$= \prod_{i=1}^{N}\prod_{t=1}^{V}\prod_{c=1}^{C_t}\prod_{k=1}^{K_t} \theta_{tck}^{\mathbb{I}(x_{i,t}=k,\mathbf{x}_{i,\mathrm{pa}(t)}=c)} \tag{26.19}$$

$$= \prod_{t=1}^{V}\prod_{c=1}^{C_t}\prod_{k=1}^{K_t} \theta_{tck}^{N_{tck}} \tag{26.20}$$

where N_{tck} is the number of times node t is in state k and its parents are in state c. (Technically these counts depend on the graph structure G, but we drop this from the notation.)

26.4.2.2 Deriving the marginal likelihood

Of course, choosing the graph with the maximum likelihood will always pick a fully connected graph (subject to the acyclicity constraint), since this maximizes the number of parameters. To avoid such overfitting, we will choose the graph with the maximum marginal likelihood, $p(\mathcal{D}|G)$; the magic of the Bayesian Occam's razor will then penalize overly complex graphs.

To compute the marginal likelihood, we need to specify priors on the parameters. We will make two standard assumptions. First, we assume **global prior parameter independence**, which means

$$p(\boldsymbol{\theta}) = \prod_{t=1}^{V} p(\boldsymbol{\theta}_t) \tag{26.21}$$

Second, we assume **local prior parameter independence**, which means

$$p(\boldsymbol{\theta}_t) = \prod_{c=1}^{C_t} p(\boldsymbol{\theta}_{tc}) \tag{26.22}$$

for each t. It turns out that these assumtions imply that the prior for each row of each CPT must be a Dirichlet (Geiger and Heckerman 1997), that is,

$$p(\boldsymbol{\theta}_{tc}) = \mathrm{Dir}(\boldsymbol{\theta}_{tc}|\boldsymbol{\alpha}_{tc}) \tag{26.23}$$

Given these assumptions, and using the results of Section 5.3.2.2, we can write down the marginal likelihood of any DAG as follows:

$$\begin{aligned} p(\mathcal{D}|G) &= \prod_{t=1}^{V}\prod_{c=1}^{C_t} \int \left[\prod_{i:x_{i,\mathrm{pa}(t)}=c} \mathrm{Cat}(x_{it}|\boldsymbol{\theta}_{tc})\right] \mathrm{Dir}(\boldsymbol{\theta}_{tc}) d\boldsymbol{\theta}_{tc} && (26.24)\\ &= \prod_{t=1}^{V}\prod_{c=1}^{C_t} \frac{B(\mathbf{N}_{tc}+\boldsymbol{\alpha}_{tc})}{B(\boldsymbol{\alpha}_{tc})} && (26.25)\\ &= \prod_{t=1}^{V}\prod_{c=1}^{C_t} \frac{\Gamma(N_{tc})}{\Gamma(N_{tc}+\alpha_{tc})} \prod_{k=1}^{K_t} \frac{\Gamma(N_{tck}+\alpha^G_{tck})}{\Gamma(\alpha^G_{ijk})} && (26.26)\\ &= \prod_{t=1}^{V} \mathrm{score}(\mathbf{N}_{t,\mathrm{pa}(t)}) && (26.27) \end{aligned}$$

where $N_{tc} = \sum_k N_{tck}$, $\alpha_{tc} = \sum_k \alpha_{tck}$, $\mathbf{N}_{t,\mathrm{pa}(t)}$ is the vector of counts (sufficient statistics) for node t and its parents, and score() is a local scoring function defined by

$$\mathrm{score}(\mathbf{N}_{t,\mathrm{pa}(t)}) \triangleq \prod_{c=1}^{C_t} \frac{B(\mathbf{N}_{tc}+\boldsymbol{\alpha}_{tc})}{B(\boldsymbol{\alpha}_{tc})} \tag{26.28}$$

We say that the marginal likelihood **decomposes** or factorizes according to the graph structure.

26.4.2.3 Setting the prior

How should we set the hyper-parameters α_{tck}? It is tempting to use a Jeffreys prior of the form $\alpha_{tck} = \frac{1}{2}$ (Equation 5.62). However, it turns out that this violates a property called **likelihood equivalence**, which is sometimes considered desirable. This property says that if G_1 and G_2 are Markov equivalent (Section 26.4.1), they should have the same marginal likelihood, since they are essentially equivalent models. Geiger and Heckerman (1997) proved that, for complete graphs, the only prior that satisfies likelihood equivalence and parameter independence is the Dirichlet prior, where the pseudo counts have the form

$$\alpha_{tck} = \alpha\ p_0(x_t = k, \mathbf{x}_{\mathrm{pa}(t)} = c) \tag{26.29}$$

where $\alpha > 0$ is called the **equivalent sample size**, and p_0 is some prior joint probability distribution. This is called the **BDe** prior, which stands for Bayesian Dirichlet likelihood equivalent.

To derive the hyper-parameters for other graph structures, Geiger and Heckerman (1997) invoked an additional assumption called **parameter modularity**, which says that if node X_t has the same parents in G_1 and G_2, then $p(\boldsymbol{\theta}_t|G_1) = p(\boldsymbol{\theta}_t|G_2)$. With this assumption, we can always derive $\boldsymbol{\alpha}_t$ for a node t in any other graph by marginalizing the pseudo counts in Equation 26.29.

Typically the prior distribution p_0 is assumed to be uniform over all possible joint configurations. In this case, we have

$$\alpha_{tck} = \frac{\alpha}{K_t C_t} \tag{26.30}$$

since $p_0(x_t = k, \mathbf{x}_{\mathrm{pa}(t)} = c) = \frac{1}{K_t C_t}$. Thus if we sum the pseudo counts over all $C_t \times K_t$ entries in the CPT, we get a total equivalent sample size of α. This is called the **BDeu** prior, where the "u" stands for uniform. This is the most widely used prior for learning Bayes net structures. For advice on setting the global tuning parameter α, see (Silander et al. 2007).

26.4.2.4 Simple worked example

We now give a very simple worked example from (Neapolitan 2003, p.438). Suppose we have just 2 binary nodes, and the following 8 data cases:

X_1	X_2
1	1
1	2
1	1
2	2
1	1
2	1
1	1
2	2

Suppose we are interested in two possible graphs: G_1 is $X_1 \rightarrow X_2$ and G_2 is the disconnected graph. The empirical counts for node 1 in G_1 are $\mathbf{N}_1 = (5, 3)$ and for node 2 are

	$X_2 = 1$	$X_2 = 2$
$X_1 = 1$	4	1
$X_1 = 2$	1	2

The BDeu prior for G_1 is $\boldsymbol{\alpha}_1 = (\alpha/2, \alpha/2)$, $\boldsymbol{\alpha}_{2|x_1=1} = (\alpha/4, \alpha/4)$ and $\boldsymbol{\alpha}_{2|x_1=2} = (\alpha/4, \alpha/4)$. For G_2, the prior for $\boldsymbol{\theta}_1$ is the same, and for $\boldsymbol{\theta}_2$ it is $\boldsymbol{\alpha}_{2|x_1=1} = (\alpha/2, \alpha/2)$ and $\boldsymbol{\alpha}_{2|x_1=2} = (\alpha/2, \alpha/2)$. If we set $\alpha = 4$, and use the BDeu prior, we find $p(\mathcal{D}|G_1) = 7.2150 \times 10^{-6}$ and $p(\mathcal{D}|G_2) = 6.7465 \times 10^{-6}$. Hence the posterior probabilites, under a uniform graph prior, are $p(G_1|\mathcal{D}) = 0.51678$ and $p(G_2|\mathcal{D}) = 0.48322$.

26.4.2.5 Example: analysis of the college plans dataset

We now consider a more interesting example from (Heckerman et al. 1997). Consider the data set collected in 1968 by Sewell and Shah which measured 5 variables that might influence the decision of high school students about whether to attend college. Specifically, the variables are as follows:

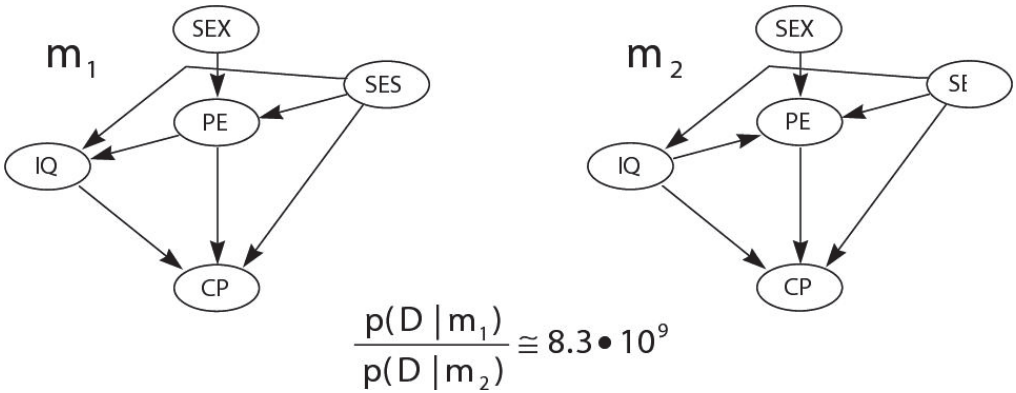


Figure 26.7 The two most probable DAGs learned from the Sewell-Shah data. Source: (Heckerman et al. 1997) . Used with kind permission of David Heckerman

- **Sex** Male or female
- **SES** Socio economic status: low, lower middle, upper middle or high.
- **IQ** Intelligence quotient: discretized into low, lower middle, upper middle or high.
- **PE** Parental encouragment: low or high
- **CP** College plans: yes or no.

These variables were measured for 10,318 Wisconsin high school seniors. There are $2 \times 4 \times 4 \times 2 \times = 128$ possible joint configurations.

Heckerman et al. computed the exact posterior over all 29,281 possible 5 node DAGs, except for ones in which SEX and/or SES have parents, and/or CP have children. (The prior probability of these graphs was set to 0, based on domain knowledge.) They used the BDeu score with $\alpha = 5$, although they said that the results were robust to any α in the range 3 to 40. The top two graphs are shown in Figure 26.7. We see that the most probable one has approximately all of the probability mass, so the posterior is extremely peaked.

It is tempting to interpret this graph in terms of causality (see Section 26.6). In particular, it seems that socio-economic status, IQ and parental encouragment all causally influence the decision about whether to go to college, which makes sense. Also, sex influences college plans only indirectly through parental encouragement, which also makes sense. However, the direct link from socio economic status to IQ seems surprising; this may be due to a hidden common cause. In Section 26.5.1.4 we will re-examine this dataset allowing for the presence of hidden variables.

26.4.2.6 The K2 algorithm

Suppose we know a total ordering of the nodes. Then we can compute the distribution over parents for each node independently, without the risk of introducing any directed cycles: we

simply enumerate over all possible subsets of ancestors and compute their marginal likelihoods.[1] If we just return the best set of parents for each node, we get the the **K2 algorithm** (Cooper and Herskovits 1992).

26.4.2.7 Handling non-tabular CPDs

If all CPDs are linear Gaussian, we can replace the Dirichlet-multinomial model with the normal-gamma model, and thus derive a different exact expression for the marginal likelihood. See (Geiger and Heckerman 1994) for the details. In fact, we can easily combine discrete nodes and Gaussian nodes, as long as the discrete nodes always have discrete parents; this is called a **conditional Gaussian** DAG. Again, we can compute the marginal likelihood in closed form. See (Bottcher and Dethlefsen 2003) for the details.

In the general case (i.e., everything except Gaussians and CPTs), we need to approximate the marginal likelihood. The simplest approach is to use the BIC approximation, which has the form

$$\sum_t \log p(\mathcal{D}_t|\hat{\boldsymbol{\theta}}_t) - \frac{K_t C_t}{2} \log N \tag{26.31}$$

26.4.3 Scaling up to larger graphs

The main challenge in computing the posterior over DAGs is that there are so many possible graphs. More precisely, (Robinson 1973) showed that the number of DAGs on D nodes satisfies the following recurrence:

$$f(D) = \sum_{i=1}^{D} (-1)^{i+1} \binom{D}{i} 2^{i(D-i)} f(D-i) \tag{26.32}$$

for $D > 2$. The base case is $f(1) = 1$. Solving this recurrence yields the following sequence: 1, 3, 25, 543, 29281, 3781503, etc.[2] In view of the enormous size of the hypothesis space, we are generally forced to use approximate methods, some of which we review below.

26.4.3.1 Approximating the mode of the posterior

We can use dynamic programming to find the globally optimal MAP DAG (up to Markov equivalence) (Koivisto and Sood 2004; Silander and Myllmaki 2006). Unfortunately this method takes $V2^V$ time and space, making it intractable beyond about 16 nodes. Indeed, the general problem of finding the globally optimal MAP DAG is provably NP-complete (Chickering 1996),

Consequently, we must settle for finding a locally optimal MAP DAG. The most common method is greedy hill climbing: at each step, the algorithm proposes small changes to the current graph, such as adding, deleting or reversing a single edge; it then moves to the neighboring graph which most increases the posterior. The method stops when it reaches a local maximum. It is important that the method only proposes local changes to the graph,

1. We can make this method more efficient by using ℓ_1-regularization to select the parents (Schmidt et al. 2007). In this case, we need to approximate the marginal likelihood as we discuss below.

2. A longer list of values can be found at `http://www.research.att.com/~njas/sequences/A003024`. Interestingly, the number of DAGs is equal to the number of (0,1) matrices all of whose eigenvalues are positive real numbers (McKay et al. 2004).

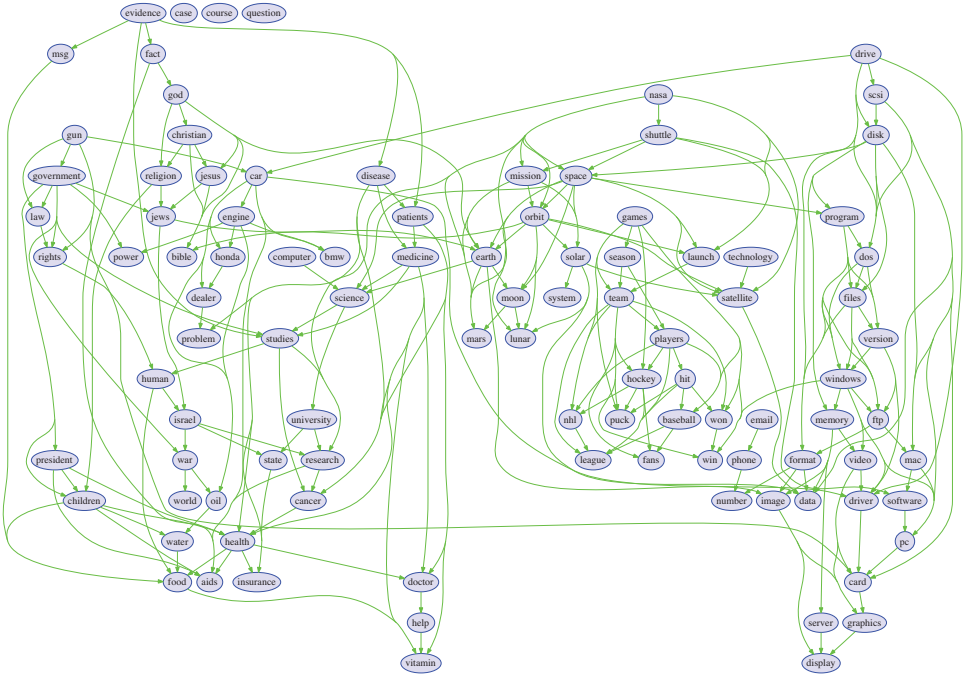


Figure 26.8 A locally optimal DAG learned from the 20-newsgroup data. From Figure 4.10 of (Schmidt 2010). Used with kind permission of Mark Schmidt.

since this enables the change in marginal likelihood (and hence the posterior) to be computed in constant time (assuming we cache the sufficient statistics). This is because all but one or two of the terms in Equation 26.25 will cancel out when computing the log Bayes factor $\delta(G \rightarrow G') = \log p(G'|\mathcal{D}) - \log p(G|\mathcal{D})$.

We can initialize the search from the best tree, which can be found using exact methods discussed in Section 26.3. For speed, we can restrict the search so it only adds edges which are part of the Markov blankets estimated from a dependency network (Schmidt 2010). Figure 26.8 gives an example of a DAG learned in this way from the 20-newsgroup data.

We can use techniques such as multiple random restarts to increase the chance of finding a good local maximum. We can also use more sophisticated local search methods, such as genetic algorithms or simulated annealing, for structure learning.

26.4.3.2 Approximating other functions of the posterior

If our goal is knowledge discovery, the MAP DAG can be misleading, for reasons we discussed in Section 5.2.1. A better approach is to compute the probability that each edge is present, $p(G_{st} = 1|\mathcal{D})$, of the probability there is a path from s to t. We can do this exactly using dynamic programming (Koivisto 2006; Parviainen and Koivisto 2011). Unfortunately these methods take $V2^V$ time in the general case, making them intractable for graphs with more than about 16

nodes.

An approximate method is to sample DAGs from the posterior, and then to compute the fraction of times there is an $s \to t$ edge or path for each (s, t) pair. The standard way to draw samples is to use the Metropolis Hastings algorithm (Section 24.3), where we use the same local proposal as we did in greedy search (Madigan and Raftery 1994).

A faster-mixing method is to use a collapsed MH sampler, as suggested in (Friedman and Koller 2003). This exploits the fact that, if a total ordering of the nodes is known, we can select the parents for each node independently, without worrying about cycles, as discussed in Section 26.4.2.6. By summing over all possible choice of parents, we can marginalize out this part of the problem, and just sample total orders. (Ellis and Wong 2008) also use order-space (collapsed) MCMC, but this time with a parallel tempering MCMC algorithm.

26.5 Learning DAG structure with latent variables

Sometimes the complete data assumption does not hold, either because we have missing data, and/ or because we have hidden variables. In this case, the marginal likelihood is given by

$$p(\mathcal{D}|G) = \int \sum_{\mathbf{h}} p(\mathcal{D}, \mathbf{h}|\boldsymbol{\theta}, G) p(\boldsymbol{\theta}|G) d\boldsymbol{\theta} = \sum_{\mathbf{h}} \int p(\mathcal{D}, \mathbf{h}|\boldsymbol{\theta}, G) p(\boldsymbol{\theta}|G) d\boldsymbol{\theta} \tag{26.33}$$

where $\mathbf{h}$ represents the hidden or missing data.

In general this is intractable to compute. For example, consider a mixture model, where we don't observe the cluster label. In this case, there are K^N possible completions of the data (assuming we have K clusters); we can evaluate the inner integral for each one of these assignments to $\mathbf{h}$, but we cannot afford to evaluate all of the integrals. (Of course, most of these integrals will correspond to hypotheses with little posterior support, such as assigning single data points to isolated clusters, but we don't know ahead of time the relative weight of these assignments.)

In this section, we discuss some ways for learning DAG structure when we have latent variables and/or missing data.

26.5.1 Approximating the marginal likelihood when we have missing data

The simplest approach is to use standard structure learning methods for fully visible DAGs, but to approximate the marginal likelihood. In Section 24.7, we discussed some Monte Carlo methods for approximating the marginal likelihood. However, these are usually too slow to use inside of a search over models. Below we mention some faster deterministic approximations.

26.5.1.1 BIC approximation

A simple approximation is to use the BIC score, which is given by

$$\text{BIC}(G) \triangleq \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \frac{\log N}{2} \dim(G) \tag{26.34}$$

where $\dim(G)$ is the number of degrees of freedom in the model and $\hat{\boldsymbol{\theta}}$ is the MAP or ML estimate. However, the BIC score often severely underestimates the true marginal likelihood (Chickering and Heckerman 1997), resulting in it selecting overly simple models.

26.5.1.2 Cheeseman-Stutz approximation

We now present a better method known as the **Cheeseman-Stutz approximation** (CS) (Cheeseman and Stutz 1996). We first compute a MAP estimate of the parameters $\hat{\boldsymbol{\theta}}$ (e.g., using EM). Denote the expected sufficient statistics of the data by $\overline{\mathcal{D}} = \overline{\mathcal{D}}(\hat{\boldsymbol{\theta}})$; in the case of discrete variables, we just "fill in" the hidden variables with their expectation. We then use the exact marginal likelihood equation on this filled-in data:

$$p(\mathcal{D}|G) \approx p(\overline{\mathcal{D}}|G) = \int p(\overline{\mathcal{D}}|\boldsymbol{\theta}, G)p(\boldsymbol{\theta}|G)d\boldsymbol{\theta} \tag{26.35}$$

However, comparing this to Equation 26.33, we can see that the value will be exponentially smaller, since it does not sum over all values of $\mathbf{h}$. To correct for this, we first write

$$\log p(\mathcal{D}|G) = \log p(\overline{\mathcal{D}}|G) + \log p(\mathcal{D}|G) - \log p(\overline{\mathcal{D}}|G) \tag{26.36}$$

and then we apply a BIC approximation to the last two terms:

$$\begin{aligned} \log p(\mathcal{D}|G) - \log p(\overline{\mathcal{D}}|G) &\approx \left[\log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \frac{N}{2}\dim(\hat{\boldsymbol{\theta}})\right] && (26.37)\\ &\quad - \left[\log p(\overline{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G) - \frac{N}{2}\dim(\hat{\boldsymbol{\theta}})\right] && (26.38)\\ &= \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \log p(\overline{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G) && (26.39) \end{aligned}$$

Putting it altogether we get

$$\log p(\mathcal{D}|G) \approx \log p(\overline{\mathcal{D}}|G) + \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \log p(\overline{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G) \tag{26.40}$$

The first term $p(\overline{\mathcal{D}}|G)$ can be computed by plugging in the filled-in data into the exact marginal likelihood. The second term $p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G)$, which involves an exponential sum (thus matching the "dimensionality" of the left hand side) can be computed using an inference algorithm. The final term $p(\overline{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G)$ can be computed by plugging in the filled-in data into the regular likelihood.

26.5.1.3 Variational Bayes EM

An even more accurate approach is to use the variational Bayes EM algorithm. Recall from Section 21.6 that the key idea is to make the following factorization assumption:

$$p(\boldsymbol{\theta}, \mathbf{z}_{1:N}|\mathcal{D}) \approx q(\boldsymbol{\theta})q(\mathbf{z}) = q(\boldsymbol{\theta})\prod_i q(\mathbf{z}_i) \tag{26.41}$$

where $\mathbf{z}_i$ are the hidden variables in case i. In the E step, we update the $q(\mathbf{z}_i)$, and in the M step, we update $q(\boldsymbol{\theta})$. The corresponding variational free energy provides a lower bound on the log marginal likelihood. In (Beal and Ghahramani 2006), it is shown that this bound is a much better approximation to the true log marginal likelihood (as estimated by a slow annealed importance sampling procedure) than either BIC or CS. In fact, one can prove that the variational bound will always be more accurate than CS (which in turn is always more accurate than BIC).

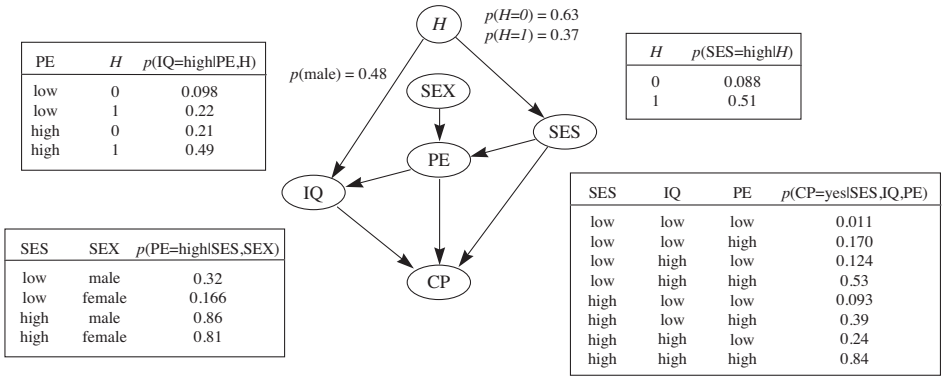


Figure 26.9 The most probable DAG with a single binary hidden variable learned from the Sewell-Shah data. MAP estimates of the CPT entries are shown for some of the nodes. Source: (Heckerman et al. 1997). Used with kind permission of David Heckerman.

26.5.1.4 Example: college plans revisited

Let us revisit the college plans dataset from Section 26.4.2.5. Recall that if we ignore the possibility of hidden variables there was a direct link from socio economic status to IQ in the MAP DAG. Heckerman et al. decided to see what would happen if they introduced a hidden variable H, which they made a parent of both SES and IQ, representing a hidden common cause. They also considered a variant in which H points to SES, IQ and PE. For both such cases, they considered dropping none, one, or both of the SES-PE and PE-IQ edges. They varied the number of states for the hidden node from 2 to 6. Thus they computed the approximate posterior over $8 \times 5 = 40$ different models, using the CS approximation.

The most probable model which they found is shown in Figure 26.9. This is $2 \cdot 10^{10}$ times more likely than the best model containing no hidden variable. It is also $5 \cdot 10^9$ times more likely than the second most probable model with a hidden variable. So again the posterior is very peaked.

These results suggests that there is indeed a hidden common cause underlying both the socio-economic status of the parents and the IQ of the children. By examining the CPT entries, we see that both SES and IQ are more likely to be high when H takes on the value 1. They interpret this to mean that the hidden variable represents "parent quality" (possibly a genetic factor). Note, however, that the arc between H and SES can be reversed without changing the v-structures in the graph, and thus without affecting the likelihood; this underscores the difficulty in interpreting hidden variables.

Interestingly, the hidden variable model has the same conditional independence assumptions amongst the visible variables as the most probable visible variable model. So it is not possible to distinguish between these hypotheses by merely looking at the empirical conditional independencies in the data (which is the basis of the **constraint-based approach** to structure learning (Pearl and Verma 1991; Spirtes et al. 2000)). Instead, by adopting a Bayesian approach, which takes parsimony into account (and not just conditional independence), we can discover

the possible existence of hidden factors. This is the basis of much of scientific and everday human reasoning (see e.g. (Griffiths and Tenenbaum 2009) for a discussion).

26.5.2 Structural EM

One way to perform structural inference in the presence of missing data is to use a standard search procedure (deterministic or stochastic), and to use the methods from Section 26.5.1 to estimate the marginal likelihood. However, this approach is very efficient, because the marginal likelihood does not decompose when we have missing data, and nor do its approximations. For example, if we use the CS approximation or the VBEM approximation, we have to perform inference in every neighboring model, just to evaluate the quality of a single move!

(Friedman 1997b; Thiesson et al. 1998) presents a much more efficient approach called the **structural EM** algorithm. The basic idea is this: instead of fitting each candidate neighboring graph and then filling in its data, fill in the data once, and use this filled-in data to evaluate the score of all the neighbors. Although this might be a bad approximation to the marginal likelihood, it can be a good enough approximation of the difference in marginal likelihoods between different models, which is all we need in order to pick the best neighbor.

More precisely, define $\overline{\mathcal{D}}(G_0, \hat{\boldsymbol{\theta}}_0)$ to be the data filled in using model G_0 with MAP parameters $\hat{\boldsymbol{\theta}}_0$. Now define a modified BIC score as follows:

$$\text{score}_{\text{BIC}}(G, \mathcal{D}) \triangleq \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \frac{\log N}{2} \dim(G) + \log p(G) + \log p(\hat{\boldsymbol{\theta}}|G) \tag{26.42}$$

where we have included the log prior for the graph and parameters. One can show (Friedman 1997b) that if we pick a graph G which increases the BIC score relative to G_0 on the expected data, it will also increase the score on the actual data, i.e.,

$$\text{score}_{\text{BIC}}(G, \overline{\mathcal{D}}(G_0, \hat{\boldsymbol{\theta}}_0)) - \text{score}_{\text{BIC}}(G_0, \overline{\mathcal{D}}(G_0, \hat{\boldsymbol{\theta}}_0) \leq \text{score}_{\text{BIC}}(G, \mathcal{D}) - \text{score}_{\text{BIC}}(G_0, \mathcal{D}) \tag{26.43}$$

To convert this into an algorithm, we proceed as follows. First we initialize with some graph G_0 and some set of parameters $\boldsymbol{\theta}_0$. Then we fill-in the data using the current parameters — in practice, this means when we ask for the expected counts for any particular family, we perform inference using our current model. (If we know which counts we will need, we can precompute all of them, which is much faster.) We then evaluate the BIC score of all of our neighbors using the filled-in data, and we pick the best neighbor. We then refit the model parameters, fill-in the data again, and repeat. For increased speed, we may choose to only refit the model every few steps, since small changes to the structure hopefully won't invalidate the parameter estimates and the filled-in data too much.

One interesting application is to learn a phylogenetic tree structure. Here the observed leaves are the DNA or protein sequences of currently alive species, and the goal is to infer the topology of the tree and the values of the missing internal nodes. There are many classical algorithms for this task (see e.g., (Durbin et al. 1998)), but one that uses SEM is discussed in (Friedman et al. 2002).

Another interesting application of this method is to learn sparse mixture models (Barash and Friedman 2002). The idea is that we have one hidden variable C specifying the cluster, and we have to choose whether to add edges $C \rightarrow X_t$ for each possible feature X_t. Thus some features will be dependent on the cluster id, and some will be independent. (See also (Law et al. 2004)

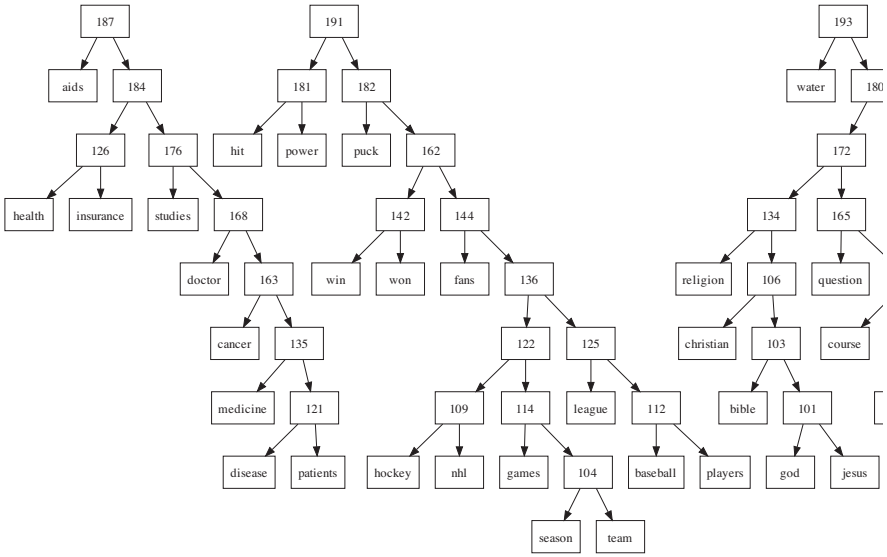


Figure 26.10 Part of a hierarchical latent tree learned from the 20-newsgroup data. From Figure 2 of (Harmeling and Williams 2011). Used with kind permission of Stefan Harmeling.

for a different way to perform this task, using regular EM and a set of bits, one per feature, that are free to change across data cases.)

26.5.3 Discovering hidden variables

In Section 26.5.1.4, we introduced a hidden variable "by hand", and then figured out the local topology by fitting a series of different models and computing the one with the best marginal likelihood. How can we automate this process?

Figure 11.1 provides one useful intuition: if there is a hidden variable in the "true model", then its children are likely to be densely connected. This suggest the following heuristic (Elidan et al. 2000): perform structure learning in the visible domain, and then look for **structural signatures**, such as sets of densely connected nodes (near-cliques); introduce a hidden variable and connect it to all nodes in this near-clique; and then let structural EM sort out the details. Unfortunately, this technique does not work too well, since structure learning algorithms are biased against fitting models with densely connected cliques.

Another useful intuition comes from clustering. In a flat mixture model, also called a **latent class model**, the discrete latent variable provides a compressed representation of its children. Thus we want to create hidden variables with high mutual information with their children.

One way to do this is to create a tree-structured hierarchy of latent variables, each of which only has to explain a small set of children. (Zhang 2004) calls this a **hierarchical latent class model**. They propose a greedy local search algorithm to learn such structures, based on adding or deleting hidden nodes, adding or deleting edges, etc. (Note that learning the optimal latent

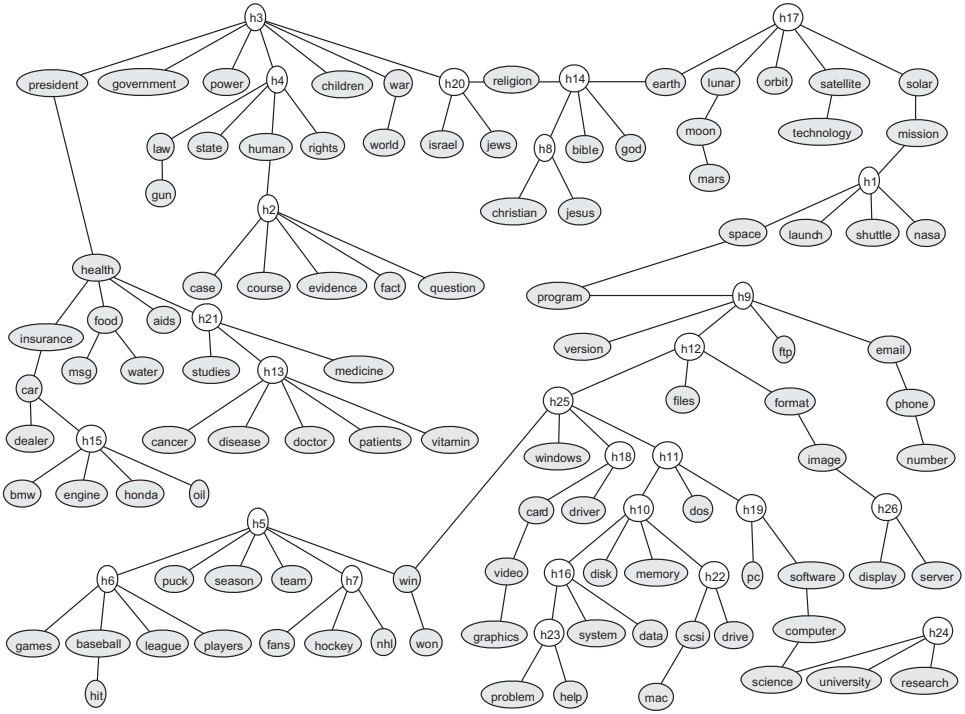


Figure 26.11 A partially latent tree learned from the 20-newsgroup data. Note that some words can have multiple meanings, and get connected to different latent variables, representing different "topics". For example, the word "win" can refer to a sports context (represented by h5) or the Microsoft Windows context (represented by h25). From Figure 12 of (Choi et al. 2011). Used with kind permission of Jin Choi.

tree is NP-hard (Roch 2006).)

Recently (Harmeling and Williams 2011) proposed a faster greedy algorithm for learning such models based on agglomerative hierarchical clustering. Rather than go into details, we just give an example of what this system can learn. Figure 26.10 shows part of a latent forest learned from the 20-newsgroup data. The algorithm imposes the constraint that each latent node has exactly two children, for speed reasons. Nevertheless, we see interpretable clusters arising. For example, Figure 26.10 shows separate clusters concerning medicine, sports and religion. This provides an alternative to LDA and other topic models (Section 4.2.2), with the added advantage that inference in latent trees is exact and takes time linear in the number of nodes.

An alternative approach is proposed in (Choi et al. 2011), in which the observed data is not constrained to be at the leaves. This method starts with the Chow-Liu tree on the observed data, and then adds hidden variables to capture higher-order dependencies between internal nodes. This results in much more compact models, as shown in Figure 26.11. This model also has better predictive accuracy than other approaches, such as mixture models, or trees where all the observed data is forced to be at the leaves. Interestingly, one can show that this method can recover the exact latent tree structure, providing the data is generated from a tree. See

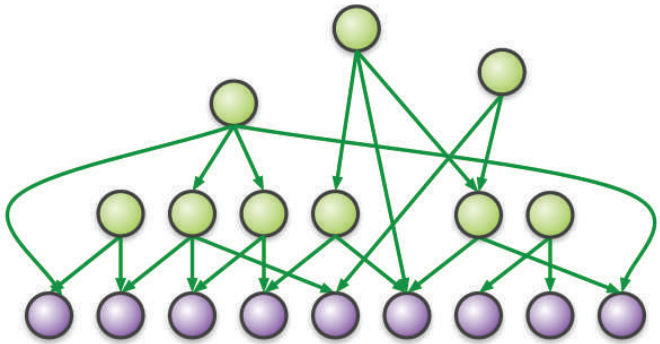


Figure 26.12 Google's rephil model. Leaves represent presence or absence of words. Internal nodes represent clusters of co-occuring words, or "concepts". All nodes are binary, and all CPDs are noisy-OR. The model contains 12 million word nodes, 1 million latent cluster nodes, and 350 million edges. Used with kind permission of Brian Milch.

(Choi et al. 2011) for details. Note, however, that this approach, unlike (Zhang 2004; Harmeling and Williams 2011), requires that the cardinality of all the variables, hidden and observed, be the same. Furthermore, if the observed variables are Gaussian, the hidden variables must be Gaussian also.

26.5.4 Case study: Google's Rephil

In this section, we describe a huge DGM called **Rephil**, which was automatically learned from data.[3] The model is widely used inside Google for various purposes, including their famous AdSense system.[4]

The model structure is shown in Figure 26.12. The leaves are binary nodes, and represent the presence or absence of words or compounds (such as "New York City") in a text document or query. The latent variables are also binary, and represent clusters of co-occuring words. All CPDs are noisy-OR, since some leaf nodes (representing words) can have many parents. This means each edge can be augmented with a hidden variable specifying if the link was activated or not; if the link is not active, then the parent cannot turn the child on. (A very similar model was proposed independently in (Singliar and Hauskrecht 2006).)

Parameter learning is based on EM, where the hidden activation status of each edge needs to be inferred (Meek and Heckerman 1997). Structure learning is based on the old neuroscience

3. The original system, called "Phil", was developed by Georges Harik and Noam Shazeer,. It has been published as US Patent #8024372, "Method and apparatus for learning a probabilistic generative model for text", filed in 2004. Rephil is a more probabilistically sound version of the method, developed by Uri Lerner et al. The summary below is based on notes by Brian Milch (who also works at Google).

4. AdSense is Google's system for matching web pages with content-appropriate ads in an automatic way, by extracting semantic keywords from web pages. These keywords play a role analogous to the words that users type in when searching; this latter form of information is used by Google's AdWords system. The details are secret, but (Levy 2011) gives an overview.

idea that "**nodes that fire together should wire together**". To implement this, we run inference and check for cluster-word and cluster-cluster pairs that frequently turn on together. We then add an edge from parent to child if the link can significantly increase the probability of the child. Links that are not activated very often are pruned out. We initialize with one cluster per "document" (corresponding to a set of semantically related phrases). We then merge clusters A and B if A explains B's top words and vice versa. We can also discard clusters that are used too rarely.

The model was trained on about 100 billion text snippets or search queries; this takes several weeks, even on a parallel distributed computing architecture. The resulting model contains 12 million word nodes and about 1 million latent cluster nodes. There are about 350 million links in the model, including many cluster-cluster dependencies. The longest path in the graph has length 555, so the model is quite deep.

Exact inference in this model is obviously infeasible. However note that most leaves will be off, since most words do not occur in a given query; such leaves can be analytically removed, as shown in Exercise 10.7. We an also prune out unlikely hidden nodes by following the strongest links from the words that are on up to their parents to get a candidate set of concepts. We then perform iterative conditional modes to find a good set of local maxima. At each step of ICM, each node sets its value to its most probable state given the values of its neighbors in its Markov blanket. This continues until it reaches a local maximum. We can repeat this process a few times from random starting configurations. At Google, this can be made to run in 15 milliseconds!

26.5.5 Structural equation models *

A **structural equation model** (Bollen 1989) is a special kind of directed mixed graph (Section 19.4.4.1), possibly cyclic, in which all CPDs are linear Gaussian, and in which all bidirected edges represent correlated Gaussian noise. Such models are also called **path diagrams**. SEMs are widely used, especially in economics and social science. It is common to interpret the edge directions in terms of causality, where directed cycles are interpreted is in terms of **feedback loops** (see e.g., (Pearl 2000, Ch.5)). However, the model is really just a way of specifying a joint Gaussian, as we show below. There is nothing inherently "causal" about it at all. (We discuss causality in Section 26.6.)

We can define an SEM as a series of full conditionals as follows:

$$x_i = \mu_i + \sum_{j \neq i} w_{ij} x_j + \epsilon_i \tag{26.44}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Psi})$. We can rewrite the model in matrix form as follows:

$$\mathbf{x} = \mathbf{W}\mathbf{x} + \boldsymbol{\mu} + \boldsymbol{\epsilon} \Rightarrow \mathbf{x} = (\mathbf{I} - \mathbf{W})^{-1}(\boldsymbol{\epsilon} + \boldsymbol{\mu}) \tag{26.45}$$

Hence the joint distribution is given by $p(\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where

$$\boldsymbol{\Sigma} = (\mathbf{I} - \mathbf{W})^{-1} \boldsymbol{\Psi} (\mathbf{I} - \mathbf{W})^{-T} \tag{26.46}$$

We draw an arc $X_i \leftarrow X_j$ if $|w_{ij}| > 0$. If $\mathbf{W}$ is lower triangular then the graph is acyclic. If, in addition, $\boldsymbol{\Psi}$ is diagonal, then the model is equivalent to a Gaussian DGM, as discussed in Section 10.2.5; such models are called **recursive**. If $\boldsymbol{\Psi}$ is not diagonal, then we draw a bidirected

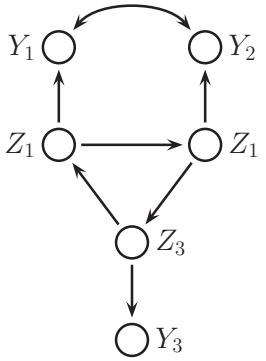


Figure 26.13 A cyclic directed mixed graphical model (non-recursive SEM). Note the $Z_1 \to Z_2 \to Z_3 \to Z_1$ feedback loop.

arc $X_i \leftrightarrow X_j$ for each non-zero off-diagonal term. Such edges represent correlation, possibly due to a hidden common cause.

When using structural equation models, it is common to partition the variables into latent variables, Z_t, and observed or **manifest** variables Y_t. For example, Figure 26.13 illustrates the following model:

$$\begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \end{pmatrix} = \begin{pmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} = \begin{pmatrix} 0 & 0 & w_{13} & 0 & 0 & 0 \\ w_{21} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{32} & 0 & 0 & 0 & 0 \\ w_{41} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{52} & 0 & 0 & 0 & 0 \\ 0 & 0 & w_{63} & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \end{pmatrix}, \tag{26.47}$$

where

$$\mathbf{\Psi} = \begin{pmatrix} \Psi_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & \Psi_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & \Psi_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & \Psi_{44} & \Psi_{45} & 0 \\ 0 & 0 & 0 & \Psi_{54} & \Psi_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & \Psi_{66} \end{pmatrix} \tag{26.48}$$

The presence of a feedback loop $Z_1 \to Z_2 \to Z_3$ is evident from the fact that $\mathbf{W}$ is not lower triangular. Also the presence of confounding between Y_1 and Y_2 is evident in the off-diagonal terms in $\mathbf{\Psi}$.

Often we assume there are multiple observations for each latent variable. To ensure identifiability, we can set the mean of the latent variables Z_t to 0, and we can set the regression weights of $Z_t \to Y_t$ to 1. This essentially defines the scale of each latent variable. (In addition to the Z's, there are the extra hidden variables implied by the presence of the bidirected edges.)

The standard practice in the SEM community, as exemplified by the popular commercial software package called **LISREL** (available from `http://www.ssicentral.com/lisrel/`), is to

build the structure by hand, to estimate the parameters by maximum likelihood, and then to test if any of the regression weights are significantly different from 0, using standard frequentist methods. However, one can also use Bayesian inference for the parameters (see e.g., (Dunson et al. 2005)). Structure learning in SEMs is rare, but since recursive SEMs are equivalent to Gaussian DAGs, many of the techniques we have been discussing in this section can be applied.

SEMs are closely related to factor analysis (FA) models (Chapter 12). The basic difference is that in an FA model, the latent Gaussian has a low-rank covariance matrix, and the observed noise has a diagonal covariance (hence no bidirected edges). In an SEM, the covariance of the latent Gaussian has a sparse Cholesky decomposition (at least if $\mathbf{W}$ is acyclic), and the observed noise might have a full covariance matrix.

Note that SEMs can be extended in many ways. For example, we can add covariates/ input variables (possibly noisily observed), we can make some of the observations be discrete (e.g., by using probit links), and so on.

26.6 Learning causal DAGs

Causal models are models which can predict the effects of interventions to, or manipulations of, a system. For example, an electronic circuit diagram implicitly provides a compact encoding of what will happen if one removes any given component, or cuts any wire. A causal medical model might predict that if I continue to smoke, I am likely to get lung cancer (and hence if I cease smoking, I am less likely to get lung cancer). Causal claims are inherently stronger, yet more useful, than purely **associative** claims, such as "people who smoke often have lung cancer".

Causal models are often represented by DAGs (Pearl 2000), although this is somewhat controversial (Dawid 2010). We explain this causal interpretation of DAGs below. We then show how to use a DAG to do causal reasoning. Finally, we briefly discuss how to learn the structure of causal DAGs. A more detailed description of this topic can be found in (Pearl 2000) and (Koller and Friedman 2009, Ch.21).

26.6.1 Causal interpretation of DAGs

In this section, we define a directed edge $A \to B$ in a DAG to mean that "A directly causes B", so if we manipulate A, then B will change. This is known as the **causal Markov assumption**. (Of course, we have not defined the word "causes", and we cannot do that by appealing to a DAG, lest we end up with a cyclic definition; see (Dawid 2010) for further disussion of this point.)

We will also assume that all relevant variables are included in the model, i.e., there are no unknown **confounders**, reflecting hidden common causes. This is called the **causal sufficiency** assumption. (If there are known to be confounders, they should be added to the model, although one can sometimes use mixed directed graphs (Section 26.5.5) as a way to avoid having to model confounders explicitly.)

Assuming we are willing to make the causal Markov and causal sufficiency assumptions, we can use DAGs to answer causal questions. The key abstraction is that of a **perfect intervention**; this represents the act of setting a variable to some known value, say setting X_i to x_i. A real world example of such a perfect intervention is a gene knockout experiment, in which a gene is "silenced". We need some notational convention to distinguish this from observing that X_i

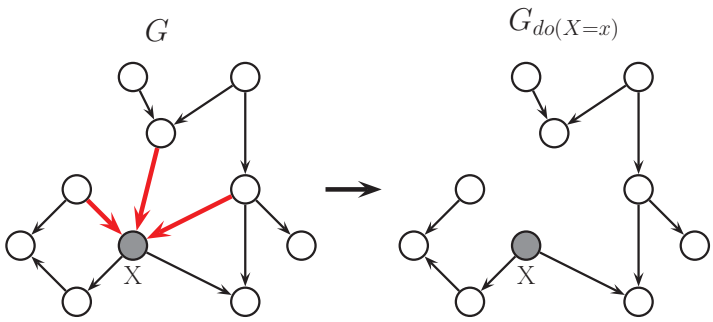


Figure 26.14 Surgical intervention on X. Based on (Pe'er 2005).

happens to have value x_i. We use Pearl's **do calculus** notation (as in the verb "to do") and write $\text{do}(X_i = x_i)$ to denote the event that we set X_i to x_i. A causal model can be used to make inferences of the form $p(\mathbf{x}|\text{do}(X_i = x_i))$, which is different from making inferences of the form $p(\mathbf{x}|X_i = x_i)$.

To understand the difference between conditioning on interventions and conditioning on observations (i.e., the difference between doing and seeing), consider a 2 node DGM $S \to Y$, in which $S = 1$ if you smoke and $S = 0$ otherwise, and $Y = 1$ if you have yellow-stained fingers, and $Y = 0$ otherwise. If I observe you have yellow fingers, I am licensed to infer that you are probably a smoker (since nicotine causes yellow stains):

$$p(S = 1|Y = 1) > p(S = 1) \tag{26.49}$$

However, if I intervene and *paint* your fingers yellow, I am no longer licensed to infer this, since I have disrupted the normal causal mechanism. Thus

$$p(S = 1|\text{do}(Y = 1)) = p(S = 1) \tag{26.50}$$

One way to model perfect interventions is to use **graph surgery**: represent the joint distribution by a DGM, and then cut the arcs coming into any nodes that were set by intervention. See Figure 26.14 for an example. This prevents any information flow from the nodes that were intervened on from being sent back up to their parents. Having perform this surgery, we can then perform probabilistic inference in the resulting "mutilated" graph in the usual way to reason about the effects of interventions. We state this formally as follows.

Theorem 26.6.1 (Manipulation theorem (Pearl 2000; Spirtes et al. 2000)). *. To compute $p(X_i|\text{do}(X_j))$ for sets of nodes i, j, we can perform surgical intervention on the X_j nodes and then use standard probabilistic inference in the mutilated graph.*

We can generalize the notion of a perfect intervention by adding interventions as explicit action nodes to the graph. The result is like an influence diagram, except there are no utility nodes (Lauritzen 2000; Dawid 2002). This has been called the **augmented DAG** (Pearl 2000). We

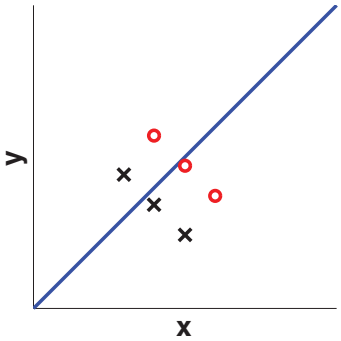


Figure 26.15 Illustration of Simpson's paradox. Figure generated by `simpsonsParadoxGraph`.

can then define the CPD $p(X_i|\text{do}(X_i))$ to be anything we want. We can also allow an action to affect multiple nodes. This is called a **fat hand** intervention, a reference to someone trying to change a single component of some system (e.g., an electronic circuit), but accidently touching multiple components and thereby causing various side effects (see (Eaton and Murphy 2007) for a way to model this using augmented DAGs).

26.6.2 Using causal DAGs to resolve Simpson's paradox

In this section, we assume we know the causal DAG. We can then do causal reasoning by applying d-separation to the mutilated graph. In this section, we give an example of this, and show how causal reasoning can help resolve a famous paradox, known as **Simpon's paradox**.

Simpson's paradox says that any statistical relationship between two variables can be reversed by including additional factors in the analysis. For example, suppose some cause C (say, taking a drug) makes some effect E (say getting better) more likely

$$P(E|C) > P(E|\neg C)$$

and yet, when we condition on the gender of the patient, we find that taking the drug makes the effect less likely in both females (F) and males ($\neg F$):

$$\begin{aligned} P(E|C,F) &< P(E|\neg C,F) \\ P(E|C,\neg F) &< P(E|\neg C,\neg F) \end{aligned}$$

This seems impossible, but by the rules of probability, this is perfectly possible, because the event space where we condition on $(\neg C, F)$ or $(\neg C, \neg F)$ can be completely different to the event space when we just condition on $\neg C$. The table of numbers below shows a concrete example (from (Pearl 2000, p175)):

	Combined				Male				Female			
	E	$\neg E$	Total	Rate	E	$\neg E$	Total	Rate	E	$\neg E$	Total	Rate
C	20	20	40	50%	18	12	30	60%	2	8	10	20%
$\neg C$	16	24	40	40%	7	3	10	70%	9	21	30	30%
Total	36	44	80		25	15	40		11	29	40	

From this table of numbers, we see that

$$p(E|C) = 20/40 = 0.5 \quad > \quad p(E|\neg C) = 16/40 = 0.4 \tag{26.51}$$

$$p(E|C,F) = 2/10 = 0.2 \quad < \quad p(E|\neg C,F) = 9/30 = 0.3 \tag{26.52}$$

$$p(E|C,\neg F) = 18/30 = 0.6 \quad < \quad p(E|\neg,\neg F) = 7/10 = 0.7 \tag{26.53}$$

A visual representation of the paradox is given in in Figure 26.15. The line which goes up and to the right shows that the effect (y-axis) increases as the cause (x-axis) increases. However, the dots represent the data for females, and the crosses represent the data for males. Within each subgroup, we see that the effect decreases as we increase the cause.

It is clear that the effect is real, but it is still very counter-intuitive. The reason the paradox arises is that we are interpreting the statements causally, but we are not using proper causal reasoning when performing our calculations. The statement that the drug C causes recovery E is

$$P(E|\text{do}(C)) \quad > \quad P(E|\text{do}(\neg C)) \tag{26.54}$$

whereas the data merely tell us

$$P(E|C) \quad > \quad P(E|\neg C) \tag{26.55}$$

This is not a contradiction. Observing C is positive evidence for E, since more males than females take the drug, and the male recovery rate is higher (regardless of the drug). Thus Equation 26.55 does not imply Equation 26.54.

Nevertheless, we are left with a practical question: should we use the drug or not? It seems like if we don't know the patient's gender, we should use the drug, but as soon as we discover if they are male or female, we should stop using it. Obviously this conclusion is ridiculous.

To answer the question, we need to make our assumptions more explicit. Suppose reality can be modeled by the causal DAG in Figure 26.16(a). To compute the causal effect of C on E, we need to **adjust for** (i.e., condition on) the **confounding variable** F. This is necessary because there is a **backdoor path** from C to E via F, so we need to check the $C \rightarrow E$ relationship for each value of F separately, to make sure the relationship between C and E is not affected by any value of F.

Suppose that for each value of F, taking the drug is harmful, that is,

$$p(E|\text{do}(C),F) \quad < \quad p(E|\text{do}(\neg C),F) \tag{26.56}$$

$$p(E|\text{do}(C),\neg F) \quad < \quad p(E|\text{do}(\neg C),\neg F) \tag{26.57}$$

Then we can show that taking the drug is harmful overall:

$$p(E|\text{do}(C)) < p(E|\text{do}(\neg C)) \tag{26.58}$$

The proof is as follows (Pearl 2000, p181). First, from our assumptions in Figure 26.16(a), we see that drugs have no effect on gender

$$p(F|\text{do}(C)) = p(F|\text{do}(\neg C)) = p(F) \tag{26.59}$$

Now using the law of total probability,

$$p(E|\text{do}(C)) \quad = \quad p(E|\text{do}(C),F)p(F|\text{do}(C)) + p(E|\text{do}(C),\neg F)p(\neg F|\text{do}(C)) \tag{26.60}$$

$$\quad = \quad p(E|\text{do}(C),F)p(F) + p(E|\text{do}(C),\neg F)p(\neg F) \tag{26.61}$$

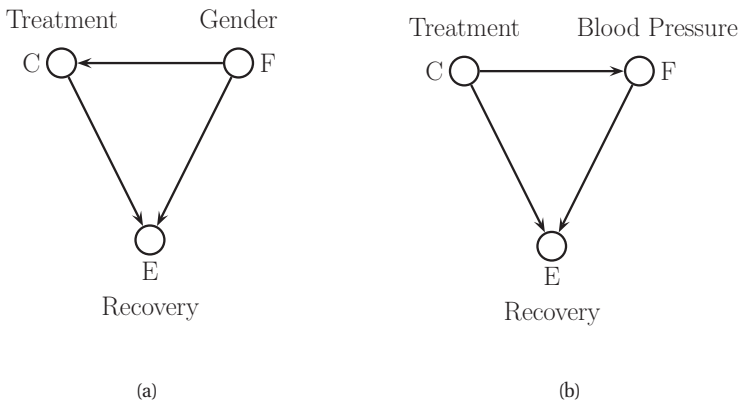


Figure 26.16 Two different models uses to illustrate Simpson's paradox. (a) F is gender and is a confounder for C and E. (b) F is blood pressure and is caused by C.

Similarly,

$$p(E|\text{do}(\neg C)) \quad = \quad p(E|\text{do}(\neg C), F)p(F) + p(E|\text{do}(\neg C), \neg F)p(\neg F) \tag{26.62}$$

Since every term in Equation 26.61 is less than the corresponding term in Equation 26.62, we conclude that

$$p(E|\text{do}(C)) < p(E|\text{do}(\neg C)) \tag{26.63}$$

So if the model in Figure 26.16(a) is correct, we should not administer the drug, since it reduces the probability of the effect.

Now consider a different version of this example. Suppose we keep the data the same but interpret F as something that is affected by C, such as blood pressure. See Figure 26.16(b). In this case, we can no longer assume

$$p(F|\text{do}(C)) = p(F|\text{do}(\neg C)) = p(F) \tag{26.64}$$

and the above proof breaks down. So $p(E|\text{do}(C)) - p(E|\text{do}(\neg C))$ may be positive or negaitve.

In the true model is Figure 26.16(b), then we should not condition on F when assessing the effect of C on E, since there is no backdoor path in this case, because of the v-structure at F. That is, conditioning on F might block one of the causal pathways. In other words, by comparing patients with the same post-treatment blood pressure (value of F), we may mask the effect of one of the two pathways by which the drug operates to bring about recovery.

Thus we see that different causal assumptions lead to different causal conclusions, and hence different courses of action. This raises the question on whether we can learn the causal model from data. We discuss this issue below.

26.6.3 Learning causal DAG structures

In this section, we discuss some ways to learn causal DAG structures.

26.6.3.1 Learning from observational data

In Section 26.4, we discussed various methods for learning DAG structures from observational data. It is natural to ask whether these methods can recover the "true" DAG structure that was used to generate the data. Clearly, even if we have infinite data, an optimal method can only identify the DAG up to Markov equivalence (Section 26.4.1). That is, it can identify the PDAG (partially directed acylic graph), but not the complete DAG structure, because all DAGs which are Markov equivalent have the same likelihood.

There are several algorithms (e.g., the **greedy equivalence search** method of (Chickering 2002)) that are consistent estimators of PDAG structure, in the sense that they identify the true Markov equivalence class as the sample size goes to infinity, assuming we observe all the variables. However, we also have to assume that the generating distribution p is **faithful** to the generating DAG G. This means that all the conditional indepence (CI) properties of p are exactly captured by the graphical structure, so $I(p) = I(G)$; this means there cannot be any CI properties in p that are due to particular settings of the parameters (such as zeros in a regression matrix) that are not graphically explicit. For this reason, a faithful distribution is also called a **stable** distribution.

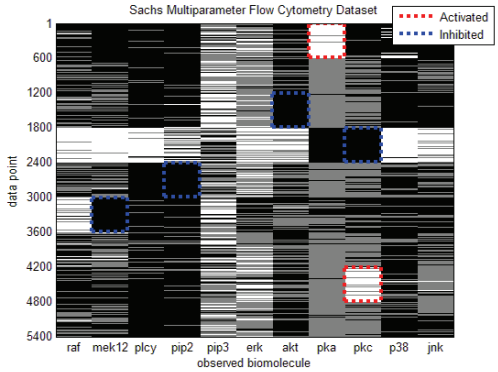
Suppose the assumptions hold and we learn a PDAG. What can we do with it? Instead of recovering the full graph, we can focus on the causal analog of edge marginals, by computing the magnitude of the causal effect of one node on another (say A on B). If we know the DAG, we can do this using techniques described in (Pearl 2000). If the DAG is unknown, we can compute a lower bound on the effect as follows (Maathuis et al. 2009): learn an equivalence class (PDAG) from data; enumerate all the DAGs in the equivalence class; apply Pearl's do-calculus to compute the magnitude of the causal effect of A on B in each DAG; finally, take the minimum of these effects as the lower bound. It is usually computationally infeasible to compute all DAGs in the equivalence class, but fortunately one only needs to be able to identify the local neighborhood of A and B, which can be esimated more efficiently, as described in (Maathuis et al. 2009). This technique is called **IDA**, which is short for "intervention-calculus when the DAG is absent".

In (Maathuis et al. 2010), this technique was applied to some yeast gene expression data. Gene knockout data was used to estimate the "ground truth" effect of each 234 single-gene deletions on the remaining 5,361 genes. Then the algorithm was applied to 63 unperturbed (wild-type) samples, and was used to rank order the likely targets of each of the 234 genes. The method had a precision of 66% when the recall was set to 10%; while low, this is substantially more than rival variable-selection methods, such as lasso and elastic net, which were only slightly above chance.

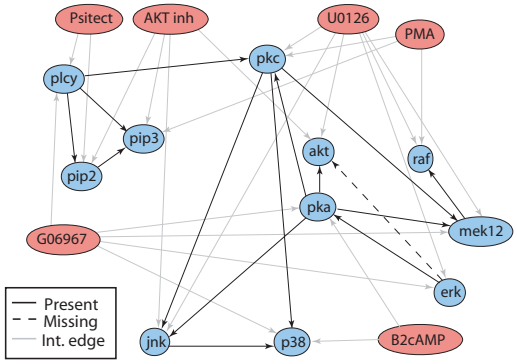
26.6.3.2 Learning from interventional data

If we want to distinguish between DAGs within the equivalence class, we need to use **interventional data**, where certain variables have been set, and the consequences have been measured. An example of this is the dataset in Figure 26.17(a), where proteins in a signalling pathway were perturbed, and their phosphorylation status was measured using a technique called flow cytometry (Sachs et al. 2005).

It is straightforward to modify the standard Bayesian scoring criteria, such as the marginal likelihood or BIC score, to handle learning from mixed observational and experimental data: we



(a)



(b)

Figure 26.17 (a) A design matrix consisting of 5400 data points (rows) measuring the status (using flow cytometry) of 11 proteins (columns) under different experimental conditions. The data has been discretized into 3 states: low (black), medium (grey) and high (white). Some proteins were explicitly controlled using activating or inhibiting chemicals. (b) A directed graphical model representing dependencies between various proteins (blue circles) and various experimental interventions (pink ovals), which was inferred from this data. We plot all edges for which $p(G_{st} = 1|\mathcal{D}) > 0.5$. Dotted edges are believed to exist in nature but were not discovered by the algorithm (1 false negative). Solid edges are true positives. The light colored edges represent the effects of intervention. Source: Figure 6d of (Eaton and Murphy 2007) . This figure can be reproduced using the code at `http://www.cs.ubc.ca/~murphyk/Software/BDAGL/index.html`.

just compute the sufficient statistics for a CPD's parameter by skipping over the cases where that node was set by intervention (Cooper and Yoo 1999). For example, when using tabular CPDs, we modify the counts as follows:

$$N_{tck} \triangleq \sum_{i: x_{it} \text{ not set}} \mathbb{I}(x_{i,t} = k, \mathbf{x}_{i,\text{pa}(t)} = c) \tag{26.65}$$

The justification for this is that in cases where node t is set by force, it is not sampled from its usual mechanism, so such cases should be ignored when inferring the parameter $\boldsymbol{\theta}_t$. The modified scoring criterion can be combined with any of the standard structure learning algorithms. (He and Geng 2009) discusses some methods for choosing which interventions to perform, so as to reduce the posterior uncertainty as quickly as possible (a form of active learning).

The preceeding method assumes the interventions are perfect. In reality, experimenters can rarely control the state of individual molecules. Instead, they inject various stimulant or inhibitor chemicals which are designed to target specific molecules, but which may have side effects. We can model this quite simply by adding the intervention nodes to the DAG, and then learning a larger augmented DAG structure, with the constraint that there are no edges between the intervention nodes, and no edges from the "regular" nodes back to the intervention nodes.

Figure 26.17(b) shows the augmented DAG that was learned from the interventional flow cytometry data depicted in Figure 26.17(a). In particular, we plot the median graph, which includes all edges for which $p(G_{ij} = 1|\mathcal{D}) > 0.5$. These were computed using the exact algorithm of (Koivisto 2006). It turns out that, in this example, the median model has exactly the same structure as the optimal MAP model, $\text{argmax}_G\, p(G|\mathcal{D})$, which was computed using the algorithm of (Koivisto and Sood 2004; Silander and Myllmaki 2006).

26.7 Learning undirected Gaussian graphical models

Learning the structured of undirected graphical models is easier than learning DAG structure because we don't need to worry about acyclicity. On the other hand, it is harder than learning DAG structure since the likelihood does not decompose (see Section 19.5). This precludes the kind of local search methods (both greedy search and MCMC sampling) we used to learn DAG structures, because the cost of evaluating each neighboring graph is too high, since we have to refit each model from scratch (there is no way to incrementally update the score of a model).

In this section, we discuss several solutions to this problem, in the context of **Gaussian random fields** or undirected Gaussian graphical models (GGM)s. We consider structure learning for discrete undirected models in Section 26.8.

26.7.1 MLE for a GGM

Before discussing structure learning, we need to discuss parameter estimation. The task of computing the MLE for a (non-decomposable) GGM is called **covariance selection** (Dempster 1972).

From Equation 4.19, the log likelihood can be written as

$$\ell(\boldsymbol{\Omega}) = \log\det\boldsymbol{\Omega} - \text{tr}(\mathbf{S}\boldsymbol{\Omega}) \tag{26.66}$$

where $\mathbf{\Omega} = \mathbf{\Sigma}^{-1}$ is the precision matrix, and $\mathbf{S} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})^T$ is the empirical covariance matrix. (For notational simplicity, we assume we have already estimated $\hat{\boldsymbol{\mu}} = \overline{\mathbf{x}}$.) One can show that the gradient of this is given by

$$\nabla \ell(\mathbf{\Omega}) = \mathbf{\Omega}^{-1} - \mathbf{S} \tag{26.67}$$

However, we have to enforce the constraints that $\Omega_{st} = 0$ if $G_{st} = 0$ (structural zeros), and that $\mathbf{\Omega}$ is positive definite. The former constraint is easy to enforce, but the latter is somewhat challenging (albeit still a convex constraint). One approach is to add a penalty term to the objective if $\mathbf{\Omega}$ leaves the positive definite cone; this is the approach used in `ggmFitMinfunc` (see also (Dahl et al. 2008)). Another approach is to use a coordinate descent method, described in (Hastie et al. 2009, p633), and implemented in `ggmFitHtf`. Yet another approach is to use iterative proportional fitting, described in Section 19.5.7. However, IPF requires identifying the cliques of the graph, which is NP-hard in general.

Interestingly, one can show that the MLE must satisfy the following property: $\Sigma_{st} = S_{st}$ if $G_{st} = 1$ or $s = t$, i.e., the covariance of a pair that are connected by an edge must match the empirical covariance. In addition, we have $\Omega_{st} = 0$ if $G_{st} = 0$, by definition of a GGM, i.e., the precision of a pair that are not connected must be 0. We say that $\mathbf{\Sigma}$ is a positive definite **matrix completion** of $\mathbf{S}$, since it retains as many of the entries in $\mathbf{S}$ as possible, corresponding to the edges in the graph, subject to the required sparsity pattern on $\mathbf{\Sigma}^{-1}$, corresponding to the absent edges; the remaining entries in $\mathbf{\Sigma}$ are filled in so as to maximize the likelihood.

Let us consider a worked example from (Hastie et al. 2009, p652). We will use the following adjacency matrix, representing the cyclic structure, $X_1 - X_2 - X_3 - X_4 - X_1$, and the following empirical covariance matrix:

$$\mathbf{G} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \quad \mathbf{S} = \begin{pmatrix} 10 & 1 & 5 & 4 \\ 1 & 10 & 2 & 6 \\ 5 & 2 & 10 & 3 \\ 4 & 6 & 3 & 10 \end{pmatrix} \tag{26.68}$$

The MLE is given by

$$\mathbf{\Sigma} = \begin{pmatrix} 10.00 & 1.00 & \mathbf{1.31} & 4.00 \\ 1.00 & 10.00 & 2.00 & \mathbf{0.87} \\ \mathbf{1.31} & 2.00 & 10.00 & 3.00 \\ 4.00 & \mathbf{0.87} & 3.00 & 10.00 \end{pmatrix}, \quad \mathbf{\Omega} = \begin{pmatrix} 0.12 & -0.01 & \mathbf{0} & -0.05 \\ -0.01 & 0.11 & -0.02 & \mathbf{0} \\ \mathbf{0} & -0.02 & 0.11 & -0.03 \\ -0.05 & \mathbf{0} & -0.03 & 0.13 \end{pmatrix} \tag{26.69}$$

(See `ggmFitDemo` for the code to reproduce these numbers.) The constrained elements in $\mathbf{\Omega}$, and the free elements in $\mathbf{\Sigma}$, both of which correspond to absent edges, have been highlighted.

26.7.2 Graphical lasso

We now discuss one way to learn a sparse GRF structure, which exploits the fact that there is a 1:1 correspondence between zeros in the precision matrix and absent edges in the graph. This suggests that we can learn a sparse graph structure by using an objective that encourages zeros in the precision matrix. By analogy to lasso (see Section 13.3), one can define the following ℓ_1 penalized NLL:

$$J(\mathbf{\Omega}) = -\log\det\mathbf{\Omega} + \mathrm{tr}(\mathbf{S}\mathbf{\Omega}) + \lambda||\mathbf{\Omega}||_1 \tag{26.70}$$

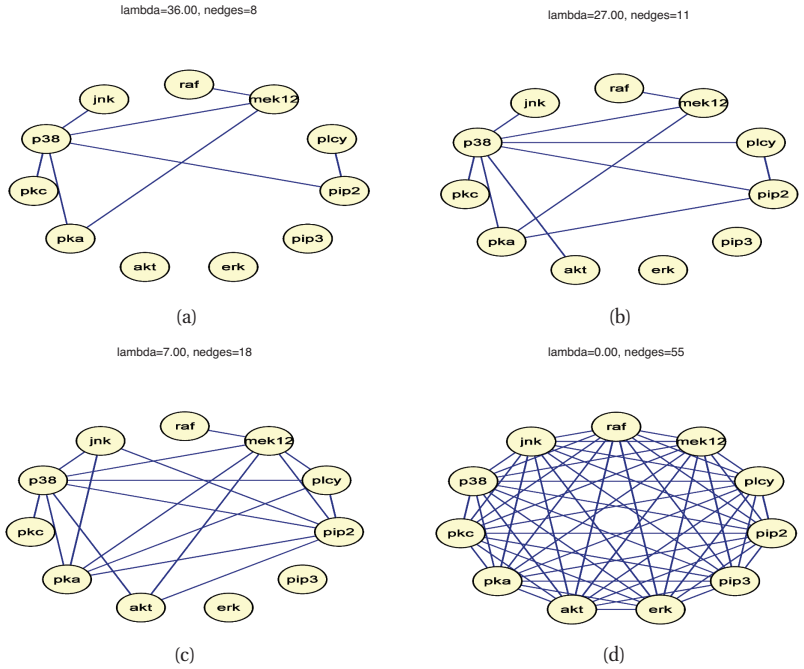


Figure 26.18 Sparse GGMs learned using graphical lasso applied to the flow cytometry data. (a) $\lambda = 36$. (b) $\lambda = 27$. (c) $\lambda = 7$. (d) $\lambda = 0$. Figure generated by `ggmLassoDemo`.

where $||\mathbf{\Omega}||_1 = \sum_{j,k} |\omega_{jk}|$ is the 1-norm of the matrix. This is called the **graphical lasso** or **Glasso**.

Although the objective is convex, it is non-smooth (because of the non-differentiable ℓ_1 penalty) and is constrained (because $\mathbf{\Omega}$ must be a positive definite matrix). Several algorithms have been proposed for optimizing this objective (Yuan and Lin 2007; Banerjee et al. 2008; Duchi et al. 2008), although arguably the simplest is the one in (Friedman et al. 2008), which uses a coordinate descent algorithm similar to the shooting algorithm for lasso. See `ggmLassoHtf` for an implementation. (See also (Mazumder and Hastie 2012) for a more recent version of this algorithm.)

As an example, let us apply the method to the flow cytometry dataset from (Sachs et al. 2005). A discretized version of the data is shown in Figure 26.17(a). Here we use the original continuous data. However, we are ignoring the fact that the data was sampled under intervention. In Figure 26.18, we illustrate the graph structures that are learned as we sweep λ from 0 to a large value. These represent a range of plausible hypotheses about the connectivity of these proteins.

It is worth comparing this with the DAG that was learned in Figure 26.17(b). The DAG has the advantage that it can easily model the interventional nature of the data, but the disadvantage that it cannot model the feedback loops that are known to exist in this biological pathway (see the discussion in (Schmidt and Murphy 2009)). Note that the fact that we show many UGMs and only one DAG is incidental: we could easily use BIC to pick the "best" UGM, and conversely, we

could easily display several DAG structures, sampled from the posterior.

26.7.3 Bayesian inference for GGM structure *

Although the graphical lasso is reasonably fast, it only gives a point estimate of the structure. Furthermore, it is not model-selection consistent (Meinshausen 2005), meaning it cannot recover the true graph even as $N \to \infty$. It would be preferable to integrate out the parameters, and perform posterior inference in the space of graphs, i.e., to compute $p(G|\mathcal{D})$. We can then extract summaries of the posterior, such as posterior edge marginals, $p(G_{ij} = 1|\mathcal{D})$, just as we did for DAGs. In this section, we discuss how to do this.

Note that the situation is analogous to Chapter 13, where we discussed variable selection. In Section 13.2, we discussed Bayesian variable selection, where we integrated out the regression weights and computed $p(\boldsymbol{\gamma}|\mathcal{D})$ and the marginal inclusion probabilities $p(\gamma_j = 1|\mathcal{D})$. Then in Section 13.3, we discussed methods based on ℓ_1 regularization. Here we have the same dichotomy, but we are presenting them in the opposite order.

If the graph is decomposable, and if we use conjugate priors, we can compute the marginal likelihood in closed form (Dawid and Lauritzen 1993). Furthermore, we can efficiently identify the decomposable neighbors of a graph (Thomas and Green 2009), i.e., the set of legal edge additions and removals. This means that we can perform relatively efficient stochastic local search to approximate the posterior (see e.g. (Giudici and Green 1999; Armstrong et al. 2008; Scott and Carvalho 2008)).

However, the restriction to decomposable graphs is rather limiting if one's goal is knowledge discovery, since the number of decomposable graphs is much less than the number of general undirected graphs.[5]

A few authors have looked at Bayesian inference for GGM structure in the non-decomposable case (e.g., (Dellaportas et al. 2003; Wong et al. 2003; Jones et al. 2005)), but such methods cannot scale to large models because they use an expensive Monte Carlo approximation to the marginal likelihood (Atay-Kayis and Massam 2005). (Lenkoski and Dobra 2008) suggested using a Laplace approxmation. This requires computing the MAP estimate of the parameters for $\boldsymbol{\Omega}$ under a G-Wishart prior (Roverato 2002). In (Lenkoski and Dobra 2008), they used the iterative proportional scaling algorithm (Speed and Kiiveri 1986; Hara and Takimura 2008) to find the mode. However, this is very slow, since it requires knowing the maximal cliques of the graph, which is NP-hard in general.

In (Moghaddam et al. 2009), a much faster method is proposed. In particular, they modify the gradient-based methods from Section 26.7.1 to find the MAP estimate; these algorithms do not need to know the cliques of the graph. A further speedup is obtained by just using a diagonal Laplace approximation, which is more accurate than BIC, but has essentially the same cost. This, plus the lack of restriction to decomposable graphs, enables fairly fast stochastic search methods to be used to approximate $p(G|\mathcal{D})$ and its mode. This approach significantly outperfomed graphical lasso, both in terms of predictive accuracy and structural recovery, for a comparable computational cost.

5. The number of decomposable graphs on V nodes, for $V = 2, \ldots, 8$, is as follows ((Armstrong 2005, p158)): 2; 8; 61; 822; 18,154; 61,7675; 30,888,596. If we divide these numbers by the number of undirected graphs, which is $2^{V(V-1)/2}$, we find the ratios are: 1, 1, 0.95, 0.8, 0.55, 0.29, 0.12. So we see that decomposable graphs form a vanishing fraction of the total hypothesis space.

26.7.4 Handling non-Gaussian data using copulas *

The graphical lasso and variants is inhertently limited to data that is jointly Gaussian, which is a rather severe restriction. Fortunately the method can be generalized to handle non-Gaussian, but still continuous, data in a fairly simple fashion. The basic idea is to estimate a set of D univariate monotonic transformations f_j, one per variable j, such that the resulting transformed data is jointly Gaussian. If this is possible, we say the data belongs to the nonparametric Normal distribution, or **nonparanormal** distribution (Liu et al. 2009). This is equivalent to the family of **Gaussian copulas** (Klaassen and Wellner 1997). Details on how to estimate the f_j transformations from the empirical cdf's of each variable can be found in (Liu et al. 2009). After transforming the data, we can compute the correlation matrix and then apply glasso in the usual way. One can show, under various assumptions, that this is a consistent estimator of the graph structure, representing the CI assumptions of the original distribution(Liu et al. 2009).

26.8 Learning undirected discrete graphical models

The problem of learning the structure for UGMs with discrete variables is harder than the Gaussian case, because computing the partition function $Z(\boldsymbol{\theta})$, which is needed for parameter estimation, has complexity comparable to computing the permanent of a matrix, which in general is intractable (Jerrum et al. 2004). By contrast, in the Gaussian case, computing Z only requires computing a matrix determinant, which is at most $O(V^3)$.

Since stochastic local search is not tractable for general discrete UGMs, below we mention some possible alternative approaches that have been tried.

26.8.1 Graphical lasso for MRFs/CRFs

It is possible to extend the graphical lasso idea to the discrete MRF and CRF case. However, now there is a set of parameters associated with each edge in the graph, so we have to use the graph analog of group lasso (see Section 13.5.1). For example, consider a pairwise CRF with ternary nodes, and node and edge potentials given by

$$\psi_t(y_t, \mathbf{x}) = \begin{pmatrix} \mathbf{v}_{t1}^T\mathbf{x} \\ \mathbf{v}_{t2}^T\mathbf{x} \\ \mathbf{v}_{t3}^T\mathbf{x} \end{pmatrix}, \; \psi_{st}(y_s, y_t, \mathbf{x}) = \begin{pmatrix} \mathbf{w}_{st11}^T\mathbf{x} & \mathbf{w}_{st12}^T\mathbf{x} & \mathbf{w}_{st13}^T\mathbf{x} \\ \mathbf{w}_{st21}^T\mathbf{x} & \mathbf{w}_{st22}^T\mathbf{x} & \mathbf{w}_{st23}^T\mathbf{x} \\ \mathbf{w}_{st31}^T\mathbf{x} & \mathbf{w}_{st32}^T\mathbf{x} & \mathbf{w}_{st33}^T\mathbf{x} \end{pmatrix} \tag{26.71}$$

where we assume $\mathbf{x}$ begins with a constant 1 term, to account for the offset. (If $\mathbf{x}$ only contains 1, the CRF reduces to an MRF.) Note that we may choose to set some of the $\mathbf{v}_{tk}$ and $\mathbf{w}_{stjk}$ weights to 0, to ensure identifiability, although this can also be taken care of by the prior, as shown in Exercise 8.5.

To learn sparse structure, we can minimize the following objective:

$$\begin{aligned} J &= -\sum_{i=1}^{N}\left[\sum_t \log\psi_t(y_{it}, \mathbf{x}_i, \mathbf{v}_t) + \sum_{s=1}^{V}\sum_{t=s+1}^{V} \log\psi_{st}(y_{is}, y_{it}, \mathbf{x}_i, \mathbf{w}_{st})\right] \\ &\quad + \lambda_1 \sum_{s=1}^{V}\sum_{t=s+1}^{V} ||\mathbf{w}_{st}||_p + \lambda_2 \sum_{t=1}^{V} ||\mathbf{v}_t||_2^2 \end{aligned} \tag{26.72}$$

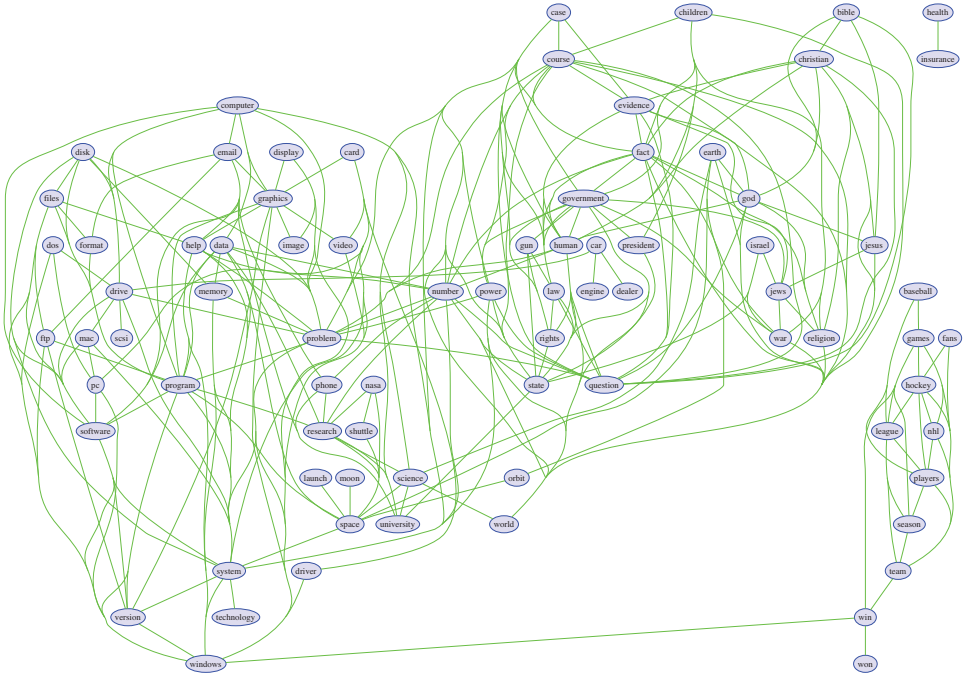


Figure 26.19 An MRF estimated from the 20-newsgroup data using group ℓ_1 regularization with $\lambda = 256$. Isolated nodes are not plotted. From Figure 5.9 of (Schmidt 2010). Used with kind permission of Mark Schmidt.

where $||\mathbf{w}_{st}||_p$ is the p-norm; common choices are $p = 2$ or $p = \infty$, as explained in Section 13.5.1. This method of CRF structure learning was first suggested in (Schmidt et al. 2008). (The use of ℓ_1 regularization for learning the structure of binary MRFs was proposed in (Lee et al. 2006).)

Although this objective is convex, it can be costly to evaluate, since we need to perform inference to compute its gradient, as explained in Section 19.6.3 (this is true also for MRFs). We should therefore use an optimizer that does not make too many calls to the objective function or its gradient, such as the projected quasi-Newton method in (Schmidt et al. 2009). In addition, we can use approximate inference, such as convex belief propagation (Section 22.4.2), to compute an approximate objective and gradient more quickly. Another approach is to apply the group lasso penalty to the pseudo-likelihood discussed in Section 19.5.4. This is much faster, since inference is no longer required (Hoefling and Tibshirani 2009). Figure 26.19 shows the result of applying this procedure to the 20-newsgroup data, where y_{it} indicates the presence of word t in document i, and $\mathbf{x}_i = 1$ (so the model is an MRF).

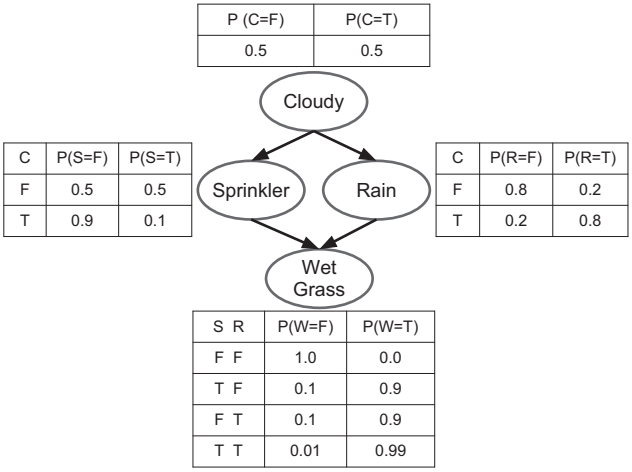


Figure 26.20 Water sprinkler DGM with corresponding binary CPTs. T and F stand for true and false.

26.8.2 Thin junction trees

So far, we have been concerned with learning "sparse" graphs, but these do not necessarily have low treewidth. For example, a $D \times D$ grid is sparse, but has treewidth $O(D)$. This means that the models we learn may be intractable to use for inference purposes, which defeats one of the two main reasons to learn graph structure in the first place (the other reason being "knowledge discovery"). There have been various attempts to learn graphical models with bounded treewidth (e.g., (Bach and Jordan 2001; Srebro 2001; Elidan and Gould 2008; Shahaf et al. 2009)), also known as **thin junction trees**, but the exact problem in general is hard.

An alternative approach is to learn a model with low **circuit complexity** (Gogate et al. 2010; Poon and Domingos 2011). Such models may have high treewidth, but they exploit context-specific independence and determinism to enable fast exact inference (see e.g., (Darwiche 2009)).

Exercises

Exercise 26.1 Causal reasoning in the sprinkler network

Consider the causal network in Figure 26.20. Let T represent true and F represent false.

a. Suppose I perform a perfect intervention and make the grass wet. What is the probability the sprinkler is on, $p(S = T|\mathrm{do}(W = T))$?

b. Suppose I perform a perfect intervention and make the grass dry. What is the probability the sprinkler is on, $p(S = T|\mathrm{do}(W = F))$?

c. Suppose I perform a perfect intervention and make the clouds "turn on" (e.g., by seeding them). What is the probability the sprinkler is on, $p(S = T|\mathrm{do}(C = T))$?