Readings #9: Learning II: Structure Learning*

Daphne Koller
Stanford University

Nir Friedman
Hebrew University

1 The Big Picture

So far, we have assumed that the network structure is given in advance. As we discussed, in many cases we also want to learn the structure from the data.

There are many reasons why we want to learn the structure. First, by exploiting structure, we get more accurate models with less data. The more accurate our structure, the fewer parameters we have to estimate. (We discuss this point further below.) As we've seen, structured models are also better for inference. Second, by learning structure, we discover structural properties of the domain. We can see what a variable depends on directly, as opposed to many other standard approaches, which simply test for correlations (that may be indirect).

Let us first consider the most obvious variant of the structure learning task: Assume that the data D is sampled from some network structure G^*, which induces some associated distribution P^*. Can we reconstruct G^* from D? If we have only a little data, the answer is clearly "no". But if we have an unlimited amount of data, what is the best we can hope for? Given enough data, we can reconstruct P^*. Thus, this question reduces to one that we addressed earlier: Is there a unique structure consistent with a given distribution P^*? As we saw, the answer is no. Even if G^* is a minimal I-map for P^*, there are many minimal I-maps for a distribution, and they can be very different from each other. Let's make a stronger assumption. Assume that G^* is a perfect map for P^*. Can we recover G^*? Even here, the answer is "partly at best". Recall that there can be many perfect maps for a distribution P^*: all of the networks in the same equivalence class as G^*. All of these are equally good structures for P^*, and therefore we cannot distinguish between them. Thus, the best we can hope for is an algorithm that, asymptotically, recovers a structure that is equivalent to the original structure G^* from which the data was sampled.

This property is an important theoretical construct. But pragmatically, we never have enough data to reconstruction the (exponentially large) P^* explicitly and exactly. Hence, we have to deal with the real-world tradeoffs of a limited amount of data, which may or may not be a good reflection of P^*. At first glance, we might think that it's better to err on the side of safety: if we think that two variables might be correlated, it's better to have an edge than not to have an edge. Otherwise, we are risking making independence statements that are false, which would make our BN not an I-map of the underlying distribution. In other words, if we make independence assumptions that are wrong, there is no way that our BN can represent the true underlying distribution, not with any setting of the parameters.

However, there are also many valid reasons not to introduce edges. Consider an experiment where we toss two standard coins X_1 and X_2 independently. We are given a data set with 100 instances of this experiment. We would like to learn the probabilities of the coins and the fact that

*This is based on draft of a text book, some of the references and citations are missing they are independent. A "typical" data set may have 27 head/head, 22 head/tail, 25 tail/head, and 26 tail/tail. The two coins seem to be weakly correlated, and therefore an I-map of the empirical distribution $\hat{P}$ — the distribution appearing in the data, would have the two variables being correlated. We have just added a superfluous edge.

This superfluous edge comes at a price. Had we estimated the parmeters for the model in which the two coins are independent, then we would get that the probability of the first coming up with a head is 49/100 and the probability of the second coming up with a head is 52/100. Both estimates are quite close to the actual "true" probability. On the other hand, if we assume that the second coin depends on the first one, we have that the probability $P(X_2 = H \mid X_1 = H)$ is $27/49 \approx 0.55$, and the probability $P(X_2 = H \mid X_1 = T)$ is $25/51 \approx 0.49$. Although one of these conditional probabilities is quite accurate, the other is off the mark.

Recall that we expect to get more accurate parameter estimates when base our estimate on many examples. However, adding the dependency between the two coins divided a set of 100 coin tosses into two pools from which estimated two parameters. This is a general phenomena. As we add more edges, we have to split up our data cases into lots of little buckets in order to estimate parameters. For example, if X has k binary parents, we have to split up our data into 2^k buckets, one for each combination of the parents. Thus, the more edges we have, the noisier our parameter estimation. Conversely, the fewer parameters we have, the more robust our estimation.

This suggests that in terms of parameter estimation, erring on the side of caution would be to avoid adding edges. In many circumstances, we might have too little data to learn parameters for complex structures, so that a simpler network might perform better on test data. This is another classical example of the bias/variance tradeoff, or the overfitting phenomenon. One somewhat surprising conclusion is that, with a limited amount of data, the "true" structure might not be the best one to learn!

There are two main approaches to the structure learning task:

- **Constraint based:** Perform tests of conditional independence on the data. These attempt to determine, for various X, Y and $\mathbf{Z}$, whether the dependence between X and Y given Z is genuine or an artifact of sampling noise. Once these tests are done, search for a network that is consistent with the observed dependencies and independencies.
- **Score based:** Define a score that evaluates how well the (in)dependencies in a structure match the data, and search for a structure that maximizes the score.

Constraint-based methods are more intuitive: they decouple the problem of finding structure from the notion of independence, and they follow more closely the definition of a BN: we have a distribution, which satifies a set of independencies, and our goal is to find an I-map for this distribution. The constraint-based algorithms are very similar to the algorithms that we discussed earlier for constructing an I-map for a distribution. Unfortunately, these methods can be sensitive to failures in individual tests. It suffices that one of these tests returns a answer to mislead the network construction procedure.

Score-based methods, on the other hand, consider the whole structure at once. As such they are not as sensitive to individual failures and are better at making compromises between the extent to which variables are dependent in the data and the "cost" of adding the edge. The disadvantage of the score-based approaches is that they pose a search problem that does not have an elegant and efficient solution.

We will focus on score-based methods. Score-based methods all operate on the same principle: we define a hypothesis space — the set of possible BN structures — and a scoring function that

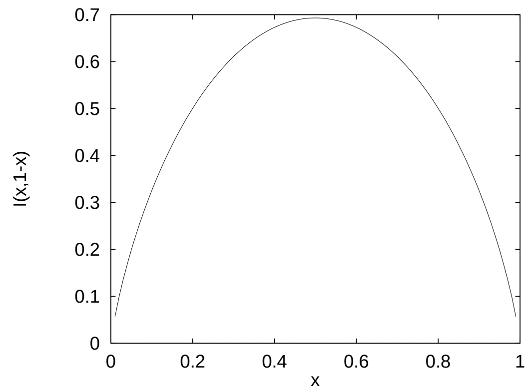

Figure 1: The entropy function over a Binomial distribution.

specifies a score for each network, based on how well it fits the data. Our goal is to find the highest-scoring network structure. The space of Bayesian networks is a combinatorial space, consisting of a superexponential number of structures — $2^{O(n^2 \log n)}$. Therefore, even with a scoring function, it is not clear how one can find the highest-scoring network. As we will see, there are very special cases where we can find the optimal network. In general, however, the problem is (as usual) NP-hard. AI has spent much of its effort on the problem of searching a combinatorial space with the goal of optimizing a function, so the framework is fairly well understood. Basically, the answer is to define a search space, and then do heuristic search.

Thus, we see that in order to define a BN structure learning algorithm, the following components need to be determined:

- Scoring function for different candidate network structures.
- The definition of the search space: operators that take one structure and modify it to produce another.
- A search algorithm that does optimization search.

We will discuss each component separately.

2 Background: Information theory

For this topic, we need to introduce some basic concepts in information theory. Let $\mathbf{Y}$ be a set of random variables, and let P be a distribution over these variables. We define the *entropy* of $\mathbf{Y}$ relative to P as:

$$\boldsymbol{H}_P(\mathbf{Y}) = -\sum_{\mathbf{y}} P(\mathbf{y}) \log P(\mathbf{y})$$

Intuitively, the entropy corresponds inversely to the amount of noise in the distribution. The noisier the distribution, the higher the entropy. Figure 1 shows the entropy function for a distribution $P(X)$ for a single binary variable X. The graph has a peak at $1/2$. In general, the entropy is always maximal at the point where all the probabilities are equal.

There are many different ways to justify the use of entropy. One way is using information theory. We can view entropy as a measure of the amount of randomness in a distribution. The more randomness there is, the less information we have.

Why is entropy related to information? Assume that we want to efficiently encode a bunch of values $\mathbf{y}[1], \mathbf{y}[2], \ldots$. Furthermore, assume that we know that the probability that we encounter a particular value $\mathbf{y}_j$ is p_j. Our goal is to use as few bits as possible when encoding the information. However, my encoding has to be unambiguous. Hence, I cannot have the encoding for one value being a prefix of the encoding for another. (Such encodings are called *prefix-free*.) How do I do that? If I have four messages that are equally likely, I will encode each message using two bits: 00,01,10,11. On the other hand, if $\mathbf{y}_1$ has probability $1/2$, $\mathbf{y}_2$ probability $1/4$, and the last two probability $1/8$ each, then the best encoding is: 0, 10, 110, 111. In general, Shannon's coding theorem tells us that the best I can do is to encode the value $\mathbf{y}_j$ using $\log 1/p_j = -\log p_j$ bits. It turns out that, if we have a long enough set of values to encode, then we can, in fact, achieve very close to this number of bits. This is precisely the principle used in the various encoding schemes such as Huffman encoding or the Lempel-Ziv algorithm, which are the core techniques for all of the various compression programs.

If we use this type of encoding, and I get M values from the distribution P, then the expected total number of bits that we need for the encoding is $-M\sum_j p_j \log p_j$; i.e., the average number of information bits needed for encoding each value is precisely the entropy of this distribution: $-\sum p_j \log p_j$. Note that the more deterministic the distribution, the lower the entropy, and indeed, the fewer bits are needed for the encoding.

What happens if the samples come from a different distribution than the one based on which we designed our code? In other words, what if the samples come from a distribution Q? In this case, each value $\mathbf{y}_j$ will require $-\log p_j$ bits to encode, but the probability of encountering $\mathbf{y}_j$ is q_j. Therefore, the expected number of bits required is

$$M(-\sum_j q_j \log p_j)$$

Now, consider a distribution P. If we had correct knowledge of P, we would need $-\sum p_j \log p_j$ bits to encode our data. If we use an encoding based on Q, we would need $-\sum p_j \log q_j$. The extra number of bits we need in the two cases is a notion of distance between the two distribution:

$$\begin{aligned} -\sum p_j \log q_j - (-\sum p_j \log p_j) &= \sum p_j \log p_j - \sum p_j \log q_j \\ &= \sum_j p_j \log \frac{p_j}{q_j} \end{aligned}$$

This notion of distance is called *relative entropy* or *KL-distance* or *KL-divergence* (KL stands for Kullback-Liebler). We denote it $D_{KL}(P\|Q)$. It is the most commonly used notion of distance between distributions. It is zero iff $P = Q$ and positive otherwise. It has a variety of nice properties (see book on Information Theory by Cover and Thomas).

For two sets of variables $\mathbf{Y}$ and $\mathbf{Z}$, we can define the *mutual information* of $\mathbf{Y}$ and $\mathbf{Z}$ relative to P as:

$$\mathit{MI}_P(\mathbf{Y};\mathbf{Z}) = \sum_{\mathbf{y},\mathbf{z}} P(\mathbf{y},\mathbf{z}) \log \frac{P(\mathbf{y},\mathbf{z})}{P(\mathbf{y})P(\mathbf{z})}$$

This measure is precisely the relative entropy between $P(\mathbf{Y},\mathbf{Z})$ and the independent distribution $P(\mathbf{Y})P(\mathbf{Z})$. It is clear intuitively that the mutual information is measuring the extent to which $\mathbf{Y}$ and $\mathbf{Z}$ are correlated in P. If they are independent, then the mutual information is zero. Otherwise, it is guaranteed to be positive.

3 Scoring Functions

As we have mentioned several times, machine learning is often formulated by specifying a hypothesis space, and a scoring function that tells us how good different hypotheses are relative to our data. We then aim to choose the highest-scoring hypothesis.

3.1 Maximum likelihood

The most obvious choice of scoring function is the log-likelihood function, which we used in the parameter estimation case. Recall that this function measures the probability of the data given a model. Thus, it seems intuitive to find a model that would make the data as probable as possible

Assume that we want to maximize the log-likelihood of the model. In this case, our model is a pair $\langle G, \boldsymbol{\theta}_G \rangle$. We have already determined how to maximize the log-likelihood for a given structure G. In other words,

$$\begin{aligned} \max_{G, \boldsymbol{\theta}_G} \ell(\langle G, \boldsymbol{\theta}_G \rangle : D) &= \max_G [\max_{\boldsymbol{\theta}_G} \ell(\langle G, \boldsymbol{\theta}_G \rangle : D)] \\ &= \max_G [\ell(\langle G, \hat{\boldsymbol{\theta}}_G \rangle : D)] \end{aligned}$$

where $\hat{\boldsymbol{\theta}}_G$ is the maximum likelihood parameter instantiation for $\boldsymbol{\theta}_G$ given the data, as discussed in the previous chapter. We therefore define:

$$\text{score}_L(G \ : \ D) = \ell(\langle G, \hat{\boldsymbol{\theta}}_G \rangle : D)$$

Unfortunately, this scoring function does not provide an appropriate metric for scoring the quality of a network. To understand this, consider again the two coins experiment. Suppose that the coins are denoted by X and Y. We consider two models, G_0 where X and Y are independent, and G_1 where Y depends on X. Now consider some parameters θ_0 for G_0. We can "match" these parameters in the context of G_1 by setting θ_1 to be such that $P(Y \mid X, G_1, \theta_1) = P(Y \mid G_0, \theta_0)$ for both values of X. For this choice of θ_1, clearly, $P(X, Y \mid G_0, \theta_0) = P(X, Y \mid G_1, \theta_1)$, and thus the likelihood of both models is the same. This implies that $\text{score}_L(G_1 \ : \ D) \geq \text{score}_L(G_0 \ : \ D)$, since every parameter choice for G_0 has a matching choice for G_1, but the opposite is not true. Thus, regardless of the acutual dependencies in D, the likelihood of G_0 would never surpass the likelihood of G_1. Moreover, unless the frequencies in D *exactly* fit the probabilies of independent events, G_1 would have higher likelihood than G_0.

This does not mean that maximum likelihood is not useful. We can use it when we limit ourselves to a class of models that cannot be too complex (say networks with limited number of parents per variable). In addition, studying the properties of maximum likelihood is useful for the scores we discuss in the next sections.

Let $\hat{P}$ be the empirical distribution observed in the data, i.e., $\hat{P}(\mathbf{x})$ for an instantiation $\mathbf{x}$ to our network variables $\mathbf{X}$ is simply the empirical frequency of $\mathbf{x}$ in D.

We have already seen that

$$\ell(\langle G, \boldsymbol{\theta} \rangle : D) = \sum_{i=1}^{n} \sum_{\mathbf{u} \in \mathit{Val}(\mathbf{Pa}_{\mathbf{X_i}})} [\sum_{x_i \in \mathit{Val}(X_i)} M[x_i, \mathbf{u}] \log \theta_{x_i|\mathbf{u}}].$$

The log-likelihood score of a graph is simply its log-likelihood relative to the maximum likelihood parameters $\boldsymbol{\theta}$. Therefore,

$$\text{score}_L(G \ : \ D) = \sum_{i=1}^{n} \text{FamScore}_L(X_i, \mathbf{Pa}_{\mathbf{X_i}} \ : \ D)$$

where

$$\mathrm{FamScore}_L(X_i, \mathbf{Pa_{X_i}} \ : \ D) = \sum_{\mathbf{u} \in Val(\mathbf{Pa_{X_i}})} [\sum_{x_i \in Val(X_i)} M[x_i, \mathbf{u}] \log \hat{\theta}_{x_i|\mathbf{u}}].$$

Note that the *family score* $\mathrm{FamScore}_L(X_i, \mathbf{Pa_{X_i}} \ : \ D)$ depends only on the sufficient statistics relating to X_i's family $M[x_i, \mathbf{u}]$.

Next, observe that $M[x_i, \mathbf{u}]$ is simply $M \cdot \hat{P}(x_i, \mathbf{u})$ and that $\theta_{x_i|\mathbf{u}}$ is $\hat{P}(x_i \mid \mathbf{u})$. Hence, we have that

$$\mathrm{FamScore}_L(X_i, \mathbf{Pa_{X_i}} \ : \ D) = M \sum_{\mathbf{u}} \sum_{x_i} \hat{P}(x_i, \mathbf{u}) \log \hat{P}(x_i \mid \mathbf{u}).$$

Lemma 3.1:

$$\mathrm{FamScore}_L(X_i, \mathbf{Pa_{X_i}} \ : \ D) = M(\boldsymbol{MI}_{\hat{P}}(X_i; \mathbf{Pa_{X_i}}) - \boldsymbol{H}_P(X_i))$$

Proof: Let $\mathbf{U} = \mathbf{Pa_{X_i}}$.

$$\begin{aligned}
\frac{1}{M}\mathrm{FamScore}_L(X_i, \mathbf{Pa_{X_i}} \ : \ D) &= \sum_{\mathbf{u}} \sum_{x_i} \hat{P}(x_i, \mathbf{u}) \log \hat{P}(x_i \mid \mathbf{u}) \\
&= \sum_{\mathbf{u}} \sum_{x_i} \hat{P}(x_i, \mathbf{u}) \log \left(\frac{\hat{P}(x_i, \mathbf{u})}{\hat{P}(\mathbf{u})} \frac{\hat{P}(x_i)}{\hat{P}(x_i)} \right) \\
&= \sum_{\mathbf{u}} \sum_{x_i} \hat{P}(x_i, \mathbf{u}) \log \frac{\hat{P}(x_i, \mathbf{u})}{\hat{P}(\mathbf{u})\hat{P}(x_i)} + \sum_{x_i} \left(\sum_{\mathbf{u}} \hat{P}(x_i, \mathbf{u}) \right) \log \hat{P}(x_i) \\
&= \boldsymbol{MI}_{\hat{P}}(X_i; \mathbf{U}) - \sum_{x_i} \hat{P}(x_i) \log \frac{1}{\hat{P}(x_i)} \\
&= \boldsymbol{MI}_{\hat{P}}(X_i; \mathbf{U}) - \boldsymbol{H}_{\hat{P}}(X_i)
\end{aligned}$$

where (as implied by the definition) the mutual information $\boldsymbol{MI}_{\hat{P}}(X_i; \mathbf{Pa_{X_i}})$ is zero if $\mathbf{Pa_{X_i}} = \emptyset$. ∎

Hence,

$$\mathrm{score}_L(G \ : \ D) = M \sum_{i=1}^{n} \boldsymbol{MI}_{\hat{P}}(X_i; \mathbf{Pa_{X_i}}) - M \sum_{i=1}^{n} \boldsymbol{H}_{\hat{P}}(X_i) \qquad (1)$$

Note that the last term does not depend on the Bayesian network structure. Recall, that we can interpret $\boldsymbol{MI}_P(X; Y)$ to the strength of the dependence between X and Y in P. Thus, the likelihood of a network measures how strong are the dependencies between variables and their parents. A network in which the dependencies are stronger, will be prefered. This is an appealing interpretation. It shows that if we are forced to choose between two networks, then maximum likelihood will choose the one where the dependencies are stronger (although both networks might capture different dependnecies that are true in the generating distribution).

We now can reexamine the effect of optimizing the likelihood score. Recall that $\boldsymbol{MI}_{\hat{P}}(X; Y)$ is non-negative, and is only zero if X and Y are independnet. Thus, in a network with two nodes X and Y, it is always better to connect the two nodes unless they are independent in the empirical distribution. Note that it is extremely unlikely for two nodes to be exactly independent. (In fact, for many sizes of data sets, it is impossible to get exact independence simply because the numbers don't work out right.) In general, we can show (fairly easily) that for any $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ and any distribution P,

$$\boldsymbol{MI}_P(\mathbf{X}; \mathbf{Y} \cup \mathbf{Z}) \geq \boldsymbol{MI}_P(\mathbf{X}; \mathbf{Y})$$

with equality only holding if $\mathbf{Z}$ is conditionally independent of $\mathbf{X}$ given $\mathbf{Y}$. This is fairly intuitive: if $\mathbf{Y}$ gives me a certain amount of information about $\mathbf{X}$, adding $\mathbf{Z}$ can only give me more information. Hence, the mutual information between a node and its parents can only go up if we add a parent, and it will go up except in those few cases where we get a conditional independence assertion holding exactly *in the empirical distribution*. Hence, in most circumstances, the maximum likelihood network will be a fully connected one.

3.2 MDL score

As the previous discussion shows, we need a better scoring function. The obvious solution is to somehow trade off fit to data with complexity. Essentially, we want to penalize networks based on how complicated they are. This is an application of Occam's razor: use a simpler hypothesis if there is one that explains the data well. How do we make this intuition concrete? We now describe one attempt to answer this question.

The *mininum description length* (MDL) principle is motivated by *universal coding*. Suppose that we are given a set D of instances, which we would like to store in our records. Naturally, we would like to conserve space and save a compressed version of D. One way of compressing the data is to find a suitable model for D that the encoder can use to produce a compact version of D. Moreover, as we want to be able to recover D, we must also store the model used by the encoder to compress D. The *total description length* is then defined as the sum of the length of the compressed version of D and the length of the description of the model used in the compression. The MDL principle dictates that the optimal model is the one (from a particular class of interest) that minimizes the total description length.

In the context of learning Bayesian networks, the model is a network. Such a network, $\mathcal{B}$, describes a probability distribution, over the instances appearing in the data. Using this distribution, we can build an encoding scheme that assigns shorter code words to more probable instances (e.g., using Shannon encoding or a Huffman code. According to the MDL principle, we should choose a network, $\mathcal{B}$, such that the combined length of the network description and the encoded data is minimized. This implies that the learning procedure balances the complexity of the induced network with the degree of accuracy with which the network represents the frequencies in D.

We now describe in detail the representation length required for the storage of both the network and the coded data. The MDL score of a candidate network is defined as the total description length. To store a network $\mathcal{B}$ we need to describe the graph G, and the parameters $\boldsymbol{\theta}_G$. We assume that the description length for G is some function $DL(G)$ of the graph structure. (As we will see, the details of this description are less crucial.)

To describe the CPDs in $\mathcal{B}$, we must store the parameters in each conditional probability table. If a node X_i has q_i different parent combinations and r_i different values, then it has $q_i(r_i - 1)$ independent parameters. (This is the same notion we used in problem set #0 to determine whether two sets of independence assumptions were equivalent.) The representation length of these parameters depends on the number of bits we use for each numeric parameter. The usual choice in the literature is $1/2 \log M$. Thus, the encoding length of $\boldsymbol{\theta}$ is

$$\frac{\log M}{2}\mathrm{Dim}[G] = \frac{\log M}{2}\sum_i q_i(r_i - 1)$$

Finally, we have to encode the training data using the probability distribution defined by the network $\mathcal{B}$. The idea is to construct a Huffman code, based on $P(X_1, \ldots, X_n \mid \mathcal{B})$. In this code, the exact length of each codeword depends on the probability assigned to that particular instance.

There is no closed-form description of this length. However, it is known that we can approximate the optimal encoding length using $-\log P(x_1, \ldots, x_n \mid \mathcal{B})$ as the encoding length of each instance $x_1, \ldots, x_n$. Thus, the description length of the data is approximated by

$$-\sum_{m=1}^{M} \log P(\mathbf{x}[m] \mid \mathcal{B}) = -\mathrm{score}_L(\mathcal{B} \ : \ D)$$

Putting these components together, we see that the description length of the data D using the network $\mathcal{B}$ is

$$DL(G) + \frac{\log M}{2}\mathrm{Dim}[G] - \mathrm{score}_L(\mathcal{B} \ : \ D)$$

Recall that the MDL principle advocates that we should minimize this length. Since we usually examine maximizing the score of a network, we define the MDL score to be the negative of the description length. Also, since we can easily choose the maximum likelihood parameters for G, we focus only on the score the graph (and assume that the parameters are optimized for it). Thus, the resulting score is:

$$\mathrm{score}_{MDL}(G \ : \ D) = \mathrm{score}_L(G \ : \ D) - \frac{\log M}{2}\mathrm{Dim}[G] - DL(G) \tag{2}$$

Using our analysis from Eq. (1), we have that

$$\mathrm{score}_{MDL}(G \ : \ D) = M\sum_{i=1}^{n} \mathbb{MI}_{\hat{P}}(X_i; \mathbf{Pa}_{\mathbf{X_i}}) - M\sum_{i=1}^{n} \mathbb{H}_{\hat{P}}(X_i) - \frac{\log M}{2}\mathrm{Dim}[G] - DL(G) \tag{3}$$

We can now observe several things about the behavior of this scoring function:

- The entropy terms do not depend on the graph, so they do not influence the choice of graph.
- Also, notice that the mutual information term and the dimension term both grow with M, whereas the description length term is fixed. Hence, even for small values of M, the description length term is swamped by the other two.
- The MDL score exhibits a tradeoff between fit to data and model complexity: the stronger the dependence of a variable on its parents, the higher the score; the more complex the network, the lower the score.
- The mutual information term grows linearly in M, whereas the complexity term grows logarithmically. Therefore, the larger M is, the more emphasis will be given to the fit to data. Asymptotically, the model will exactly fit the dependencies in the data.

To make this last statement, we introduce the following definition, which formalized our earlier discussion.

Definition 3.2: Assume that our data is generated by some true model G^*. We say that a scoring function is *consistent* if the following properties hold as the amount of data $M \to \infty$, with probability that approaches 1 (over possible choices of data set D):

- the structure G^* will maximize the score.
- all structures G that are not equivalent to G^* will have strictly lower score.

∎

Theorem 3.3: *The MDL score is consistent.*

Proof: Our goal is to prove that if the MDL score asymptotically returns G then G implies the same independence assumtions as G^*. We prove each side separately. We sketch this proof.

Consider some graph G that implies an independence assumption that G^* does not support. Then G cannot be an I-map of the true underlying distribution P. Hence, G cannot be a maximum likelihood model for P, so that we must have:

$$\sum_i \mathbb{M}I_P(X_i; \mathbf{Pa}^{\mathbf{G}^*}_{\mathbf{X_i}}) > \sum_i \mathbb{M}I_P(X_i; \mathbf{Pa}^{\mathbf{G}}_{\mathbf{X_i}})$$

As $M \to \infty$, our empirical distribution $\hat{P}$ will converge to P with probability 1. Therefore, for large M,

$$\mathrm{score}_L(G^* : D) - \mathrm{score}_L(G : D) \approx \delta M$$

where $\delta = \sum_i \mathbb{M}I_P(X_i; \mathbf{Pa}^{\mathbf{G}^*}_{\mathbf{X_i}}) - \sum_i \mathbb{M}I_P(X_i; \mathbf{Pa}^{\mathbf{G}}_{\mathbf{X_i}})$. Therefore, asymptotically we have that

$$\mathrm{score}_{MDL}(G^* : D) - \mathrm{score}_{MDL}(G : D) \approx \delta M + \frac{1}{2}(\mathrm{Dim}[G] - \mathrm{Dim}[G^*]) \log M$$

The first term grows much faster than the second, so that eventually its effect will dominate, and the score of G^* will be better, contradicting our assumption.

Now, assume that G implies all the independence assumptions that G^*, but that G^* implies an independence assumption that G does not. (In other words, G is a superset of G^*.) In this case, G can represent any distribution that G^* can. In particular, it can represent $\mathcal{P}$. As $\hat{P}$ converges to P, we will have that:

$$\mathrm{score}_L(G^* : D) - \mathrm{score}_L(G : D) \to 0;$$

in fact, each term converges to 0 itself. Therefore, asymptotically we have that for

$$\mathrm{score}_{MDL}(G^* : D) - \mathrm{score}_{MDL}(G : D) \approx \frac{1}{2}(\mathrm{Dim}[G] - \mathrm{Dim}[G^*]) \log M$$

which means that G^* will be preferred to G. ∎

Note that consistency is an assymptotic property, and thus it does not imply much about the properties of networks learned with limited amounts of data. Nonetheless, the proof illustrate the tradeoffs that are playing a role in the definition of MDL.

3.3 Bayesian scoring

Our final scoring function is Bayesian scoring. Recall that the main principle of the Bayesian approach was that, whenever we have uncertainty over anything, we should place a distribution over it. In this case, we have uncertainty both over structure and over parameters. We therefore define a structure prior $P(G)$ and a parameter prior $P(\boldsymbol{\theta}_G \mid G)$. By Bayes rule, we have

$$P(G \mid D) = \frac{P(D \mid G)P(G)}{P(D)}$$

where, as usual, the denominator is simply a normalizing factor that does not help distinguish between different structures. Thus, we can define the *Bayesian score* as:

$$\mathrm{score}_B(G : D) = \log P(D \mid G) + \log P(G) \tag{4}$$

The ability to ascribe a prior over structures gives us a way of preferring some structures over others. For example, we can penalize dense structures more than sparse ones. It turns out, however, that this term in the score is almost irrelevant compared to the other. To understand this point, consider the first term, $\log P(D \mid G)$. It is important to distinguish this expression from the log-likelihood score $\text{score}_L(G : D)$. The likelihood score measures the log probability of the data relative to G and *the maximum likelihood parameters* $\hat{\boldsymbol{\theta}}_G$. The Bayesian score measures the log probability relative only to the graph. More precisely, $P(D \mid G)$ takes into consideration our uncertainty over the parameters:

$$P(D \mid G) = \int_0^1 P(\boldsymbol{\theta}_G) P(D \mid \boldsymbol{\theta}_G) d\boldsymbol{\theta}_G \tag{5}$$

Thus, we call $P(D \mid G)$ the *marginal likelihood of the data given the structure* (where we use the word "marginal" because we marginalize out the parameters).

To understand the distinction, consider a single binary random variable X, and assume we have a distribution $Dir(\alpha_h, \alpha_t)$ over X. Consider a data set D that has M_h heads and M_t tails. Then

$$P(D \mid \hat{\theta}) = \left(\frac{M_h}{M}\right)^{M_h} \cdot \left(\frac{M_t}{M}\right)^{M_t}$$

Now, consider the marginal likelihood. Here, we are not conditioning on the parameter. Rather, we are computing

$$P(x[1], \ldots, x[M]) = P(x[1]) \cdot P(x[2] \mid x[1]) \cdot \ldots \cdot P(x[M] \mid x[1], \ldots, x[M-1])$$

Recall that

$$P(x[m+1] \mid x[1], \ldots, x[m]) = \frac{M_h^m + \alpha_h}{m + \alpha}$$

where M_h^m is the number of heads in the first m examples. For example, if $D = \langle H, T, T, H, H \rangle$, we have

$$\begin{aligned} P(x[1], \ldots, x[5]) &= \frac{\alpha_h}{\alpha} \frac{\alpha_t}{\alpha+1} \frac{\alpha_t+1}{\alpha+2} \frac{\alpha_h+1}{\alpha+3} \frac{\alpha_h+2}{\alpha+4} \\ &= \frac{[\alpha_h(\alpha_h+1)(\alpha_h+2)][\alpha_t(\alpha_t+1)]}{\alpha \cdots (\alpha+4)} \end{aligned}$$

Picking $\alpha_h = \alpha_t = 1$, we get

$$\frac{[1 \cdot 2 \cdot 3] \cdot [1 \cdot 2]}{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5} = \frac{12}{120}$$

which is very different from the

$$\left(\frac{3}{5}\right)^3 \cdot \left(\frac{2}{5}\right)^2 = \frac{108}{3125}$$

that we have for the likelihood.

In general, we have

$$\frac{[\alpha_h \cdots (\alpha_h + M_h - 1)][\alpha_t \cdots (\alpha_t + M_t - 1)]}{\alpha \cdots (\alpha + M - 1)}$$

Recalling that $\Gamma(m) = (m-1)!$, we can rewrite

$$\alpha(\alpha+1) \cdots (\alpha + M - 1) = \frac{\Gamma(\alpha + M)}{\Gamma(\alpha)}$$

Thus,

$$P(x[m+1] \mid x[1], \ldots, x[m]) = \frac{\Gamma(\alpha)}{\Gamma(\alpha + M)} \frac{\Gamma(\alpha_h + M_h)}{\Gamma(\alpha_h)} \frac{\Gamma(\alpha_t + M_t)}{\Gamma(\alpha_t)}$$

Let's see what happens in (slightly) more interesting networks. Consider two possible structures over two binary random variables X and Y. G_1 is the graph with no edges. Here, we have:

$$P(D \mid G_1) = \int_0^1 P(\theta_X, \theta_Y) P(D \mid [\theta_X, \theta_Y]) d[\theta_X, \theta_Y].$$

G_2 is a graph with an edge $X \to Y$. Here, we have:

$$P(D \mid G_2) = \int_0^1 P(\theta_X, \theta_{Y|x^1}, \theta_{Y|x^0}) P(D \mid [\theta_X, \theta_{Y|x^1}, \theta_{Y|x^0}]) d[\theta_X, \theta_{Y|x^1}, \theta_{Y|x^0}]$$

As we discussed before, our parameter independence assumptions allow us to decompose each of these integrals into a product of smaller integrals, each over a single parameter. Each of these is associated with an expression very similar to the one we just saw. In other words, the probability for a structure with several parameters is simply the product of the probabilities for the individual parameters.

Thus, we will have that

$$P(D \mid G) = \prod_i \prod_{\mathbf{u} \in Val(\mathbf{Pa}^{\mathbf{G}}_{\mathbf{X_i}})} \frac{\Gamma(\alpha_{X_i|\mathbf{u}})}{\Gamma(\alpha_{X_i|\mathbf{u}}) + M[\mathbf{u}]} \prod_{x_i{}^j \in Val(X_i)} \left[\frac{\Gamma(\alpha_{x_i{}^j|\mathbf{u}})}{\Gamma(\alpha_{x_i{}^j|\mathbf{u}}) + M[x_i{}^j, \mathbf{u}]} \right]$$

where $\alpha_{x_i{}^j|\mathbf{u}}$ is the hyperparameter for $X_i = x_i{}^j$ and $\mathbf{Pa}^{\mathbf{G}}_{\mathbf{X_i}} = \mathbf{u}$, and $\alpha_{X_i|\mathbf{u}} = \sum_j \alpha_{x_i{}^j|\mathbf{u}}$.

The actual Bayesian score requires that we transform this formula by taking its logarithm. Note that the result is a summation over terms, one for each of the nodes in the network. In other words, we have that:

$$\text{score}_B(G \ : \ D) = \sum_i \text{FamScore}_B(X_i, \mathbf{Pa}^{\mathbf{G}}_{\mathbf{X_i}} \ : \ D).$$

We have left unspecified the actual choice of priors. For the structure prior, we often use a uniform prior over structures, although other priors have been proposed. But in order to use Bayesian scores, we also need to have parameter priors for the parameterization corresponding to every possible structure. The number of possible structures is super-exponential, which makes it difficult to elicit separate parameters for each one. How do we elicit priors for all these networks? One simplistic approach is simply to take some fixed Dirichlet distribution, e.g., $Dir(1, 1, 1, \ldots, 1)$, for every parameter. However, that is somewhat inconsistent. Consider a structure where the binary variable Y has no parents. If we take $Dir(1, 1)$ for θ_Y, we are in effect stating that our imaginary sample size is 2. But now, consider a different structure where Y has the parent X, which has 4 values. If we take $Dir(1, 1)$ as our prior for all parameters $\theta_{Y|x^i}$, we are effectively stating that we have seen 2 imaginary samples in each context x^i, for a total of 8. It seems that the number of imaginary samples we have seen for different events is a basic concept, and should not vary with different candidate structures.

A much better approach is to elicit some sort of prior distribution P_0 over the entire probability space and a total size α for the set of imaginary samples. We then set the parameters in the corresponding way: the prior over θ_Y is

$$\begin{aligned} \alpha_h &= P_0(y^1) \cdot \alpha \\ \alpha_t &= P_0(y^0) \cdot \alpha \end{aligned}$$

The prior over $\theta_{Y|x^i}$ is

$$\alpha_{Y|x^i} = P_0(y^1, x^i) \cdot \alpha$$

We can elicit P_0 in a variety of ways. The simplest approach is to select P_0 to be uniform. A slightly more sophisticated choice is to select it to be a set of independent marginals, one for each variable. However, we can even choose P_0 to have the form of a Bayesian network. Note that this network structure is used *only* to provide parameter priors. It is not used to guide the structure search directly.

One can actually prove [Geiger & Heckerman, 1995] that this type of prior is the only one which is consistent with some set of reasonable assumptions. The Bayesian score that we get using a Dirichlet prior of this type is known as the *BDe score* (for Bayesian Dirichlet equivalent).

It is interesting to analyze the marginal likelihood as the amount of data we have grows asymptotically.

Theorem 3.4: *If we use a Dirichlet parameter prior (e.g., BDe), then, as $M \to \infty$, we have that:*

$$\log P(D \mid G) = \ell(\hat{\boldsymbol{\theta}}_G : D) - \frac{\log M}{2}\mathrm{Dim}[G] + O(1)$$

The proof is based on using *Stirling's approximation* for the various Gamma functions. Roughly speaking, Stirling's approximation tells us that

$$m! \approx \sqrt{2\pi m}\, m^m e^{-m} 1.$$

Recalling that $\Gamma(m) = (m-1)!$, we can replace the Γ functions in the equation with Stirling's approximation. If we do that, most things cancel. The $O(1)$ term is a shorthand for all of the constant terms — those that do not grow with M.

From this theorem, we see that, asymptotically, the MDL score and Bayesian score with Dirichlet priors are equivalent. Both have additional constant factors: the $O(1)$ and the $\log P(G)$ for the Bayesian score, and $DL(G)$ for the MDL score. However, these terms are vanishingly insignificant as M grows, so that, for large M, these two scores are equivalent. As a consequence, we obtain:

Corollary 3.5: *The Bayesian score with Dirichlet priors is consistent.*

This follows since the MDL score is consistent, and the two scores are asymptotically equivalent. However, for small M, their behavior is quite different. Empirically, the Bayesian score tends to learn better models from fewer instances.

3.4 Summary

To summarize, we have discussed three scoring functions: log-likelihood, MDL, and BDe (Bayesian). All three of them have the form

$$\mathrm{score}_X(G \; : \; D) = \sum i = 1^n \mathrm{FamScore}_X(X_i, \mathbf{Pa_{X_i}} \; : \; D) \tag{6}$$

where $\mathrm{FamScore}_X(X_i, \mathbf{Pa_{X_i}} \; : \; D)$ is a separate term in the score that depends only on the statistics for X_i and its parents in G. As we will see, this property turns out to be very important.

We also note that all three scores are *score-equivalent*: they assign the same score to networks in the same I-equivalence class. This property is a very compelling one: networks in the same I-equivalence class can represent exactly the same set of distributions. It does not make sense for our prior to distinguish between them. Of course, in certain cases, we might have additional prior knowledge that allows us to distinguish between equivalent networks. For example, we might believe that X precedes Y in time, in which case any edge between X and Y (if present) must be directed $X \to Y$. In the Bayesian framework, we can incorporate this type of bias into our structure prior $P(G)$. However, the marginal likelihood component remains the same for I-equivalent structures.

Finally, we saw that MDL and BDe are asymptotically equivalent and consistent. By contrast, log-likelihood overfits in almost all cases, regardless of the amount of data we have (even the tinyest digressions from true independence cause superfluous edges to be added).

4 Structure Search

In the previous section, we defined several ways of scoring a candidate network. Now, we have a well-defined optimization problem. Our input is:

- Training data D
- Scoring function (including priors, if needed)
- Set of possible structures (incorporating any prior knowledge)

Our desired output is a network that maximzes the score. It will turn out that the algorithms for this optimization rest on the decomposability properties of the score, i.e., Eq. (6).

It turns out that this problem partitions into two cases. The first is learning *tree-structured Bayesian networks*, i.e., ones where there is at most one parent per node. It turns out that finding a tree that optimizes the score (for any of the three scores) is quite easy. In the homework, you will show how this can be done for the log-likelihood score.

If we allow more than one parent, then the problem immediately becomes much harder. In fact, we can prove the following theorem:

Theorem 4.1: *Finding maximal scoring network structures where nodes are restricted to having at most k parents is NP-hard for any $k > 1$.*

We therefore resort to heuristic search. We define a search space, where the states in the space are possible structures, and the operators that take us from one state to the other denote adjacency of structures. We traverse this space looking for high-scoring functions.

4.1 The search space

Let's begin by discussing the search space. Most simply (and also most frequently), the search space is the set of network structures, with the obvious operators:

- Add an edge
- Delete an edge
- Reverse an edge

That is, we start out with some candidate network, which may be the empty one, or one which some expert gave us as a starting point. At each point, the "successor" candidate networks are the ones which are the result of applying one of these operators. A simple example is shown in Figure 2.

While this is the standard search space, it is rather inefficient. It contains a lot of network structures which are I-equivalent: they represent the same independence statements and therefore can

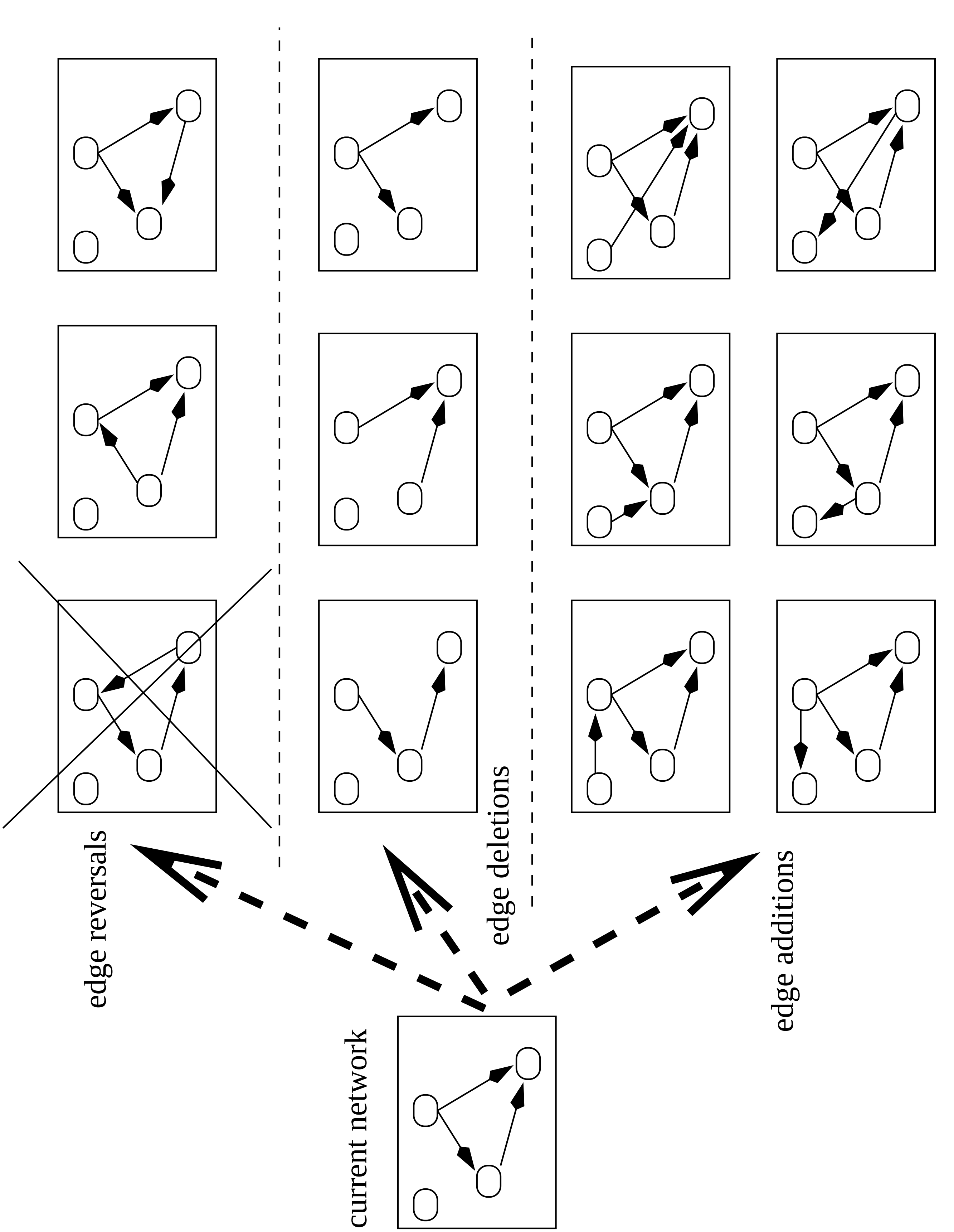

Figure 2: Possible successors of a single candidate BN

encode exactly the same class of distributions. All of our scoring functions would give I-equivalent networks the same score. Therefore, the search may spend much of its time just moving around between these equivalent networks before making any progress by making substantial changes. An alternative idea is to search directly over the space of I-equivalence classes rather than networks. This search takes fewer steps, but each is more expensive, because it is substantially more expensive to make local transformations on I-equivalence classes. Therefore, the tradeoff is often unclear.

4.2 search

Procedure Greedy-Structure-Search-X(
    G_0 over $X_1, \ldots, X_n$, // starting BN structure
    data set D, // Fully observed data set
    optionally $P(\theta_G)$ for all G // a set of parameter priors for the Bayesian score)
    Let $G' := G_0$
    Let $s' := \text{score}_X(G \ : \ D)$
    Repeat
        Let $G := G'$
        Let $s := s'$
        For each operator o
            Let G_o be the graph obtained by applying o to G
            If G_o is cyclic, continue
            If $\text{score}_X(G_o \ : \ D) > s$
                Set $G' := G_o$
                Set $s' := \text{score}_X(G_o \ : \ D)$
    Until $s' \leq s$
    Return (G')

Figure 3: Greedy structure search algorithm, with an arbitrary scoring function $\text{score}_X(G \ : \ D)$.

We are now in the realm of standard optimization search. The most obvious algorithm for optimization search is simple greedy hill-climbing, shown in Figure 3. The basic idea behind the algorithm is as follows: We pick an initial network structure G as a starting point; this network can be empty, random, the best tree, or a network obtained from some prior knowledge. We compute its score. We then consider all of the "neighbors" of G in the space — all of the legal networks obtained by applying a single operator to G — and compute the score for each of them. We then apply the change that leads to the best improvement in the score. We continue this process until no modification improves the score.

There are several key problems with greedy hill-climbing:

- local maxima: all one-edge changes reduce the score
- plateaux: a large set of neighboring networks that have the same score.

Both occur in this search space.

There is another type of problem that often comes up in optimization search: ridges. A ridge is a set of points in the space that looks locally like a local maximum. However, it is only a local maximum because our search algorithm cannot detect that there is a nearby point that is strictly better. We can often avoid ridges by introducing a new operator that jumps directly to the better point. In this case, the arc reversal operator was introduced for exactly this reason. If our set of operators had no arc reversal, we would still have a connected search space: you can simulate any arc reversal by an edge deletion and addition. However, that would introduce many many local maxima: we cannot introduce the reverse edge before deleting the original, and deleting the original often causes our score to go down, as it implies an undesirable independence assumptions. This problem can be avoided by the two-step "macro-operator" that corresponds to arc-reversal.

We can improve on the performance of greedy hill-climbing by using more clever search algorithms. Some common extensions are:

- **TABU-search:** keep list of K most recently visited structures and avoid them, i.e., apply best move that leads to a structure not on the list. This approach deals with local maxima whose "hill" has fewer than K structures.
- **Random restarts:** once stuck, apply some fixed number of random edge changes and then restart greedy search. At the end of the search, select the best structure encountered anywhere on the trajectory. This approach can escape from basin of one local maximum to another.
- **Simulated annealing:** Evaluate operators in random order. If the randomly selected operator induces an uphill step, move to the resulting structure. (Note, it does not have to be the best of the current neighbors.) If the operator induces a downhill step, apply it with probability inversely proportional to the reduction in score. A *temperature parameter* determines the probability of taking downhill steps. As the search progress, the temperature decreases, and the algorithm becomes less likely to take a downhill step.

In fact, it turns out that very cheap and simple algorithms work quite well. In many cases, greedy hill climbing with random restarts works better than the more clever and computationally expensive approaches. In another example, consider a two-pass greedy algorithm which, in the first pass, adds arcs in a greedy fashion until the model score does not improve. In the second pass, it deletes arcs in a greedy fashion until the model score does not improve. Surprisingly, that also works reasonably in some cases (although not always).

4.3 Score Decomposition

The dominant factor in the cost of the search algorithm is the evaluation of different successor networks at each stage. The number of successor networks is around n^2. To see this, observe that each pair of nodes is either connected by an edge or not. If they are, we can delete it (always OK) or reverse it (might introduce a cycle); if they are not, we can add an edge (might introduce a cycle).

To evaluate each of these candidate successors, we need to score the network. This process requires that we traverse all the different data cases, computing sufficient statistics relative to our new structure. This computation can get quite expensive. This is where the decomposability property really turns out to be useful. Recall that all of our scores can be decomposed into a a sum of terms, one for each node X_i in the network. Each of these family scores is computed relative only to the nodes in the family of X_i. A local change — adding, deleting, or reversing an edge — leaves almost all of the families in the network unchanged. (Adding and deleting changes one family, and reversing changes two.) For families whose composition do not change, the associated component of the score also does not change.

To understand the importance of this observation, assume that our current candidate network is G. For each operator, we compute the improvement in the score that would result in making that change. We can do this computation relatively efficiently. For example, if we are considering an operator that adds a parent X_i to some node X_j, the difference in score is

$$\delta[a(X_i \to X_j)] = \text{FamScore}(X_i, \mathbf{Pa}^{\mathbf{G}}_{\mathbf{X_i}} \cup \{\mathbf{X_j}\} \ : \ D) - \text{FamScore}(X_i, \mathbf{Pa}^{\mathbf{G}}_{\mathbf{X_i}} \ : \ D)$$

Note that this computation involves only the sufficient statistics for $X_i, \mathbf{Pa}^{\mathbf{G}}_{\mathbf{X_i}}$ and for $X_i, \mathbf{Pa}^{\mathbf{G}}_{\mathbf{X_i}} \cup \{\mathbf{X_j}\}$. This requires a pass only over the appropriate columns in the table describing the training data.

Now, assume that we have decided to take some particular operator o that only changes the family of X_k (for $k \neq j$). The key observation is that $\delta[a(X_i \to X_j)]$ remains unchanged; we do not need to recompute it. We only need to recompute $\delta(o')$ for operators o' that involve X_k!

Thus, we can have an algorithm that maintains a list of potential operators o, and their associatd $\delta(o)$. We keep these sorted in a heap. When we change the network by using operator o, we need to recompute $\delta[o']$ only for operators o' whose family was modified by o. This turns out to be crucial to the functionality of the algorithm.

In general, the passes over the data in order to compute sufficient statistics turn out to be by far the most compute-intensive part of the algorithm. (Most cycles are spent here.) We can reduce the overhead of this step by caching previously computed counts, and by marginalizing counts such as $M[x, y]$ to compute $M[x]$. There has also been some interesting work recently on efficient data structures for maintaining sufficient statistics that can significantly improve the performance of the algorithm.

5 Exploiting local structure

We have seen that the global network structure helps us in learning, by parameterizing the network in a compact way. This reduces data fragmentation, and leads to more robust parameter estimation. Furthermore, when doing structure search, the parameterization biases our scoring function towards simpler structures: ones that have fewer parameters. This was the case for both MDL and BDe.

It turns out that local structure can be exploited in the same way. In particular, consider a case where we have a CPD-tree at some node. The parameterization of a CPD-tree has a multinomial distribution at each of the leaves of the CPD-tree. The total number of leaves is no more than (and typically much less than) the number of entries in a full table. For example, in our CPD-tree for the *Time* node in the Travel-time network, we have four leaves in the tree, and therefore four different distribution:

$$P(\mathit{Time} \mid \mathit{Road} = 101, {t_{101}}^1)$$
$$P(\mathit{Time} \mid \mathit{Road} = 101, {t_{101}}^0)$$
$$P(\mathit{Time} \mid \mathit{Road} = 280, {t_{280}}^1)$$
$$P(\mathit{Time} \mid \mathit{Road} = 280, {t_{280}}^0)$$

Hence, using CPD-trees reduces the number of parameters and thereby data fragmentation.

We can learn the tree structure automatically by using standard decision-tree learning algorithms. In fact, it turns out that the standard entropy-based splitting rule for decision trees is doing greedy local steps that maximize the log-likelihood. More precisely, consider a node X and a set of (potential) parents $Y_1, \ldots, Y_k$. By picking the entropy-maximizing split in the decision tree that tries to "classify" X using $Y_1, \ldots, Y_k$, we are taking the step that most increases

$$\text{FamScore}_L(X_i, \{Y_1, \ldots, Y_k\} \ : \ D)$$

It is straightforward to adapt the decision-tree splitting rule so that it takes greedy steps relative to our other scoring functions.

Even assume that this edge really ought to be there. However, It turns out that local structure also allows us to learn more accurate structures. Essentially, we can make finer-grained tradeoffs. For example, assume that we are considering adding an edge from X to Y. Even if X is only binary-valued, this operation would double the size of a table representation for the CPD of Y. If Y already has a lot of parents, the cost in terms of number of parameters can be very large. For small amounts of data, the cost in terms of number of parameters might dominate, even if the edge really "should" be there. However, if we represent our CPD as a tree, adding the dependency doesn't necessarily double the size of the CPD, since we can add the dependency only in certain circumstances. Thus, we can afford to be more generous with adding this dependency. It turns out that we can learn much more accurate structures by using this idea.

We can apply the same techniques to other types of local structure. For example, we can apply standard linear regression techniques to learn a linear Gaussian CPD for a continuous variable. In general, the decomposition of the likelihood function is a property that is independent of the form of the individual CPDs. Thus, to "plug in" a different CPD representation into the MDL score, all we need is an estimation procedure that finds a maximum likelihood parameterization for a conditional distribution. Given that, we can estimate the log-likelihood term in the MDL score; the model complexity term is simply a function of the number of parameters and the number of bits to represent them. For Bayesian estimation, we also need the ability to compute the marginal likelihood. As for the log-likelihood, the marginal likelihood decomposes. Hence, we only need a closed form for marginal likelihoods of individual nodes, which exists for various parametric families (e.g., linear Gaussian CPDs).

6 Bayesian model averaging

Finally, one might wonder why we are being inconsistent in our Bayesianism. In the case of parameters, when we had uncertainty over our parameters, we simply put a prior over them and averaged out by using expectation. Why not put a distribution over structures and average out over the set of all structures? This idea has many advantages. Assume that our goal is to learn the structure in order to understand the domain. If we end up with a particular structure, can we really assert that it is the "right one"? Our search is heuristic, so that it may have missed the best structure entirely. Furthermore, even if this structure is the best one, perhaps there are many others that are quite different, and are only a little bit worse. It is much more compelling to be able to say that, relative to the set of all structures, some edge exists with high probability. Note that we can, in principle make such a statement: The Bayesian approach defines a posterior distribution over structures given the data, which can be used to evaluate how likely each structure is relative to the data. The Bayesian approach can also be useful in other settings. It has been shown that, even purely for estimating the underlying probability distribution, one can get a better estimate by taking the distributions obtained over all structures, and constructing the weighted average $P(\mathbf{X} \mid G)P(G \mid d)$.

Attractive as this idea might be, at first glance it seems totally infeasible, due to the super-exponential number of structures. However, there are methods for getting around this problem, to some extent. For example, it turns out that we gain some of these benefits simply by considering a small number of very likely structures. There are heuristic search algorithms that explicitly aim to find all structures whose score is above a certain threshold. Another class of approaches is based on MCMC methods, where we sample structures from the distribution $P(G \mid D)$, and then use the resulting samples.