Learning Bayesian Belief Networks
An approach based on the MDL Principle

Wai Lam and Fahiem Bacchus
Department of Computer Science
University of Waterloo
Waterloo, Ontario,
Canada, N2L 3G1
{fbacchus,wlam1}@logos.uwaterloo.ca

To Appear in
Computational Intelligence, Vol 10:4, 1994

Abstract

A new approach for learning Bayesian belief networks from raw data is presented. The approach is based on Rissanen's Minimal Description Length (MDL) principle, which is particularly well suited for this task. Our approach does not require any prior assumptions about the distribution being learned. In particular, our method can learn *unrestricted multiply-connected* belief networks. Furthermore, unlike other approaches our method allows us to tradeoff accuracy and complexity in the learned model. This is important since if the learned model is very complex (highly connected) it can be conceptually and computationally intractable. In such a case it would be preferable to use a simpler model even if it is less accurate. The MDL principle offers a reasoned method for making this tradeoff. We also show that our method generalizes previous approaches based on Kullback cross-entropy. Experiments have been conducted to demonstrate the feasibility of the approach.

Keywords: Knowledge Acquisition; Bayes Nets; Uncertainty Reasoning.

1 Introduction

Bayesian belief networks, advanced by Pearl (1986), have become an important paradigm for representing and reasoning with uncertainty. Systems based on Bayesian networks have been constructed in a number of different application areas, ranging from medical diagnosis, e.g., (Beinlich et al., 1989), to reasoning about the oil market, e.g., (Abramson, 1991). Despite these successes, a major obstacle to using Bayesian networks lies in the difficulty of constructing them in complex domains. It can be a very time-consuming and error-prone task to specify a network that can serve as a useful probabilistic model of the problem domain; there is a knowledge engineering bottleneck. Clearly, any mechanism that can help automate this task would be beneficial. A promising approach to this problem is to try to automatically construct, or learn, such network representations from raw data. In many areas raw data can be provided from a database of records. If techniques can be constructed for automatically learning Bayesian networks from data, not only will this help address the knowledge engineering problem, but it will also facilitate the automatic refinement of the representation as new data is accumulated.

In this paper we present a new approach to learning Bayesian networks. Our method can discover arbitrary network structures from raw data without relying on *any* assumptions about the underlying probability distribution that generated the data. In particular, the method can learn unrestricted *multiply-connected networks*. Multiply-connected networks are more expressive than tree or polytree networks, and that extra expressiveness is sometimes essential if the network is to be a sufficiently accurate model of the underlying distribution.

Although multiply-connected networks allow us to more accurately model the underlying distribution, they have a number of disadvantages. Computationally they are much more difficult to deal with. It is well known that in the worst case it is intractable to compute posterior probabilities in multiply-connected Bayesian networks; to be precise this computation is NP-Hard (Cooper, 1990). Furthermore, the time complexity of the known algorithms increases with the degree of connectivity of the network. For large multiply-connected networks approximation algorithms are often used, either based on stochastic simulation, e.g., (Chavez and Cooper, 1990; Chavez, 1990; Dagum and Chavez, 1991; Fung and Chang, 1990; Henrion, 1987; Pearl, 1987; Shachter and Peot, 1990), or search through the space of alternative instantiations, e.g., (Cooper, 1984; Henrion, 1990; Henrion, 1991; Peng and Reggia, 1987a; Peng and Reggia, 1987b). In practice these algorithms allow one to reason with more complex networks than can be handled by the exact algorithms. However, it has recently been shown that in general computing approximations in multiply-connected networks is also NP-hard (Dagum and Luby, 1993). Besides the time complexity of reasoning with multiply-connected networks, such networks also present a space complexity problem. In particular, the space complexity of the network increases with its degree of connectivity. Bayesian networks with more connections between their nodes require the storage of more probability parameters; the number of probability parameters required at each node

increases exponentially with the number of its parents, i.e., with the number of incoming arcs. Hence, irrespective of the type of algorithm used there are computational, both time and space, benefits in using networks of low connectivity.

Besides computational advantages, networks of low connectivity also possess conceptual advantages. The topology of a Bayesian network expresses information about the underlying causal and probabilistic relationships in the domain. Networks with simpler topologies are easier to understand. Hence, when we learn a network from raw data, there is an advantage in constructing simpler networks: they are simpler to understand. This can be particularly important if we also wish to explain the results computed using the network.

Hence, we are faced with a tradeoff. More complex networks allow for more accurate models, but at the same time such models are computationally and conceptually more difficult to use. The approach we take is to construct Bayesian networks that balance accuracy and usefulness. Our method will learn a less complex network if that network is sufficiently accurate, and at the same time, unlike some previous methods, it is still capable of learning a complex network if no simpler network is sufficiently accurate. To make this tradeoff we use a well-studied formalism: Rissanen's Minimum Description Length (MDL) Principle (Rissanen, 1978).

Besides the reasons given above, making a tradeoff between accuracy and usefulness seems to be particularly important when learning from raw data. The raw data is itself only an approximate picture of the true underlying distribution. It is *highly* unlikely that the frequencies expressed in the raw data match the true frequencies of the underlying distribution. Since the raw data is only a sample of the population, the only guarantee we have is that the frequencies in the raw data are probably close to the true frequencies. Hence, any method that attempts to uncover the true underlying distribution can *at best* only uncover an approximation to the underlying distribution: the approximation that is expressed in the raw data. If all we can do is approximate the underlying distribution, then it seems only reasonable to prefer approximations that are more useful.

Example 1 [Approximately Equivalent Networks] To better illustrate this point consider the two networks in Figure 1 in which all the nodes take on binary values. In the graph G1, the node C has two parents, A and B, while in G2, C's only parent is B. G2 is a simpler singly-connected network. However, if we examine the conditional probability parameters associated with node C in graph G1 we find that C's value depends *mainly* on the value of B and only in a minor way on the value of A. Hence, the dependency relationships of the distribution described by G1 are almost the same as those in the distribution described by G2; these two Bayesian networks can be considered as approximately equivalent structures even though they have different topologies.

Bayesian networks are commonly used to manage belief update as some of the nodes become instantiated to particular values. Under this usage two networks can be regarded as being approximately equivalent if they exhibit close results after belief update. For example, the two networks in Figure 1 are approximately equivalent under this usage. Suppose there

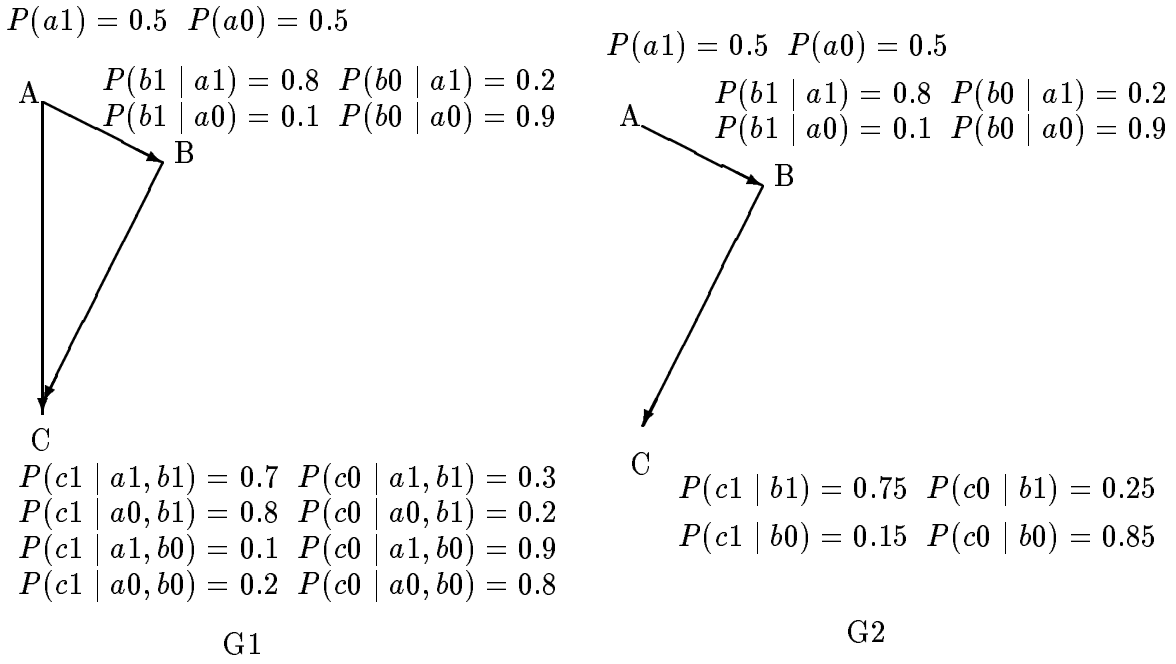


Figure 1: Approximately Equivalent Bayesian Belief Networks

is an initial evidence supporting $P(a1) = 0.3$. After performing belief update on these two networks, both of them lead to almost the same result: $P(b1) = 0.31$ in both networks, $P(c1) = 0.356$ in G1, and $P(c1) = 0.336$ in G2. In this case there is little loss in accuracy modeling the underlying distribution using the simpler network G2 instead of the more complex G1.

Furthermore, say that we learned the probability parameters, $P(c1|a1, b1), \ldots, P(c0|a0, b0)$, from frequencies taken over raw data where an error of 0.1 was possible[1]. Then it is quite possible that, e.g., $P(c1|a1, b1) = P(c1|a0, b1) = 0.75$ in the true distribution, even though this was not the case in the raw data. That is, it is quite possible that G2 is in fact a *more accurate* model of the underlying distribution than G1 (although it is not a more accurate model of the raw data). Given in addition the fact that it is a simpler, and thus more useful, model, the approach of learning the most accurate model of the raw data is moot.

■

As mentioned above we use the MDL principle to make a tradeoff between accuracy and usefulness. The MDL principle says that the best model of a collection of data is the one that minimizes the sum of the encoding lengths of the data and the model itself. That is, with the aid of the model we can represent, or encode, the data more compactly, by exploiting the probabilistic regularities described by the model. However, the model itself will require

[1]If this amount of error in the raw data seems excessive, then it should be noted that we could easily alter this example to accommodate smaller errors.

some representation. The MDL principle specifies that both components should be taken into consideration. However, finding the network (model) that minimizes the sum of these two components is a computationally intractable task: there are simply too many networks to search. Hence, our realization of the MDL principle is based on a heuristic search algorithm that tries to find a network that has low, but not necessarily minimum, description length. We have conducted a number of experiments that successfully demonstrate the feasibility of our method.

In the sequel we first discuss related work on learning Bayesian Networks. Then we discuss in more detail the MDL principle and the manner in which it can be applied to the task at hand. A discussion of our heuristic search algorithm follows along with a presentation of the experimental results. We conclude with a discussion of future work.

2 Related Work

The earliest work that can be viewed as learning network models was that of Chow and Liu (1968). Their approach was able to recover simple tree structured belief networks from a database of records. If the database was generated by a distribution that had a tree structure, it could be exactly recovered, given sufficient raw data. Otherwise their method guaranteed that the probability distribution of the learned tree network was the closest of all tree networks to the underlying distribution of the raw data. The criterion of "closeness" they used was the well-known Kullback-Leibler cross-entropy measure (Kullback and Leibler, 1951). The main restriction of this work was that it could only learn tree structures. Hence, if the raw data was the result of a non-tree structured distribution, the learned structure could be very inaccurate. In subsequent work Rebane and Pearl (1987) were able to extend Chow and Liu's methods to the recovery of networks of singly connected networks (polytrees). If the underlying distribution had a polytree structure, its topological structure could be exactly recovered (modulo the orientation of some of the arcs). But again if the raw data came from a non-polytree distribution, the learned structure could be very inaccurate.

If we have three random variables, X, Y, and Z, we can say that X is independent of Y given Z if for every value of X, Y, Z, say x, y, z, we have that $P(X = x|Z = z) = P(X = x|Z = z, Y = y)$. That is, the probability that X takes on value x given that Z takes on value z is unaffected by Y taking on value y, for all values x, y, z. We can denote this independence relationship by $I(X, Y, Z)$. Given a set of such independencies, Geiger et al. developed an approach (1990) that can discover a minimal-edge I-map. A network structure is an I-map of a probability distribution if every independence relation exhibited in the network holds also in the distribution (Pearl, 1988; Geiger and Pearl, 1990). However, their approach is again limited to polytrees; it is only guaranteed to work in the case where the underlying distribution has a polytree structure.

All of the above approaches fail to recover the richer and more realistic class of multiply-connected networks, which topologically are directed acyclic graphs (dags). Recently, Spirtes

et al. (1990) have developed an algorithm that can construct multiply-connected networks. And, more recently, Verma and Pearl (1990; 1991) have developed what they call an IC-Algorithm that can also recover these kinds of structures. However, both approaches require that the underlying distribution being learned be *dag-isomorphic*.[2] But, not all distributions are. As a result, both of these methods have the common drawback that they are not guaranteed to work when the underlying distribution fails to be dag-isomorphic. In such cases no conclusions can be drawn about the closeness of fit between the learned structure and the underlying distribution.

All of these methods share the common disadvantage that they make assumptions about the underlying distribution. Unfortunately, we are hardly ever in a position to know properties of the underlying distribution. This is what we are trying to learn! Hence, we have no assurance that these methods will work well in practice. These methods might produce very inaccurate models if the underlying distribution fails to fall into the category of distributions they can deal with. Nevertheless, these approaches have provided a great deal of information pertinent to learning Bayesian networks.

An interesting alternate approach which can also deal with multiply-connected networks is that of Cooper and Herskovits (1991). Their approach tries to find the most probable network using a Bayesian approach. As with all Bayesian approaches, they must assume a prior distribution over the space of all possible network structures. They have taken this prior to be uniform.[3] Unfortunately, it seems to us that this is the wrong choice. By choosing this prior their method would prefer a more accurate network, even if that network is *much* more complex and only slightly more accurate. Given that we must perform learning with only a limited amount of data, this insistence on accuracy is questionable.[4]

One way of viewing the MDL principle is as a Bayesian approach in which the prior distribution over the models is inversely related to their encoding length, i.e., their complexity. Hence, the MDL principle has a bias towards learning models that are as simple as possible. As we have argued in Section 1, this seems to us to be a far more reasonable approach.

Cooper and Herskovits face the same problem we do: the space of possible network structures is simply too large to explore exhaustively. Hence, they also develop a heuristic method that searches a constrained set of structures looking, in their case, for the one with highest posterior probability, and in our case for the one with minimal description length. The heuristic method they choose requires a user specified ordering of the variables, and the network that they learn respects this ordering (i.e., the parents of a node are always

[2] A distribution is dag-isomorphic if there is some dag that displays all of its dependencies and independencies (Pearl, 1988).

[3] Cooper and Herskovits have also considered other priors. However, an essential difficulty remains in justifying any particular choice. With the MDL principle there is a natural justification for preferring less complex networks.

[4] Interestingly, the method Cooper and Herskovits used to construct their "uniform" prior did not in fact assign equal prior probability to every network. Rather, their prior introduced what appears to be an entirely accidental (in so far as it was never pointed out in their paper) bias for simpler networks.

lower in this ordering). The heuristic method we develop, however, does not require such an ordering, which is an advantage in situations where there is insufficient information to generate a total ordering.

3 The MDL Principle

In this section we will discuss in greater detail Rissanen's Minimal Description Length (MDL) principle, a well studied formalism in learning theory, see e.g., (Gao and Li, 1989; Rissanen, 1978). The MDL principle is based on the idea that the best model of a collection of data items is the model that minimizes the sum of

1. the length of the encoding of the model, and
2. the length of the encoding of the data *given the model*,

both of which can be measured in bits.

Example 2 [Polynomials] Say that the data items consist of n points on the plane, each specified by a pair of real coordinates with fixed precision, $(x_1, y_1), \ldots, (x_n, y_n)$. Suppose we wish to find a function (model) that fits these points. If we use an n degree polynomial that passes precisely through these points we would need $n+1$ numbers to specify the coefficients of the polynomial (item 1). To store the data given this polynomial (item 2) we would need to store the n x-coordinates, $x_1, \ldots, x_n$. However, we would not need to store the y-coordinates, $y_1, \ldots, y_n$, as each y_i could be computed precisely from our polynomial and the respective x-coordinate x_i. Hence the sum of the description lengths would be $2n + 1$ times the number of bits required to store the numbers at the given precision.

On the other hand if we used a lower order polynomial, say order k, we would only need $k + 1$ numbers to store the coordinates (item 1). Once again we could store the data points by specifying the x-coordinates, n numbers. However, now we could not guarantee that our polynomial precisely fits the data, hence there would in general be some error δ_i between the y-value of the polynomial evaluated at x_i and the actual y-coordinate of the i-th data point, y_i. Hence, to encode the data points we would need to store these error factors along with the x-coordinates. However, if $\max(\delta_1, \ldots, \delta_n)$ was small, we would need less bits to store these error factors than an ordinary number. In particular, we might be able to store these error factors in less space than would be required to store the extra $n - k$ coordinates needed when using a n-degree polynomial. Hence, there might be some polynomial of degree $k < n$ that yields the minimal description length.

■

To apply the MDL principle to Bayesian networks we need to specify how we can perform the two encodings, the network itself (item 1) and the raw data given a network (item 2).

3.1 Encoding the Network

To represent a particular Bayesian network, the following information is necessary and sufficient:

- A list of the parents of each node.
- The set of conditional probabilities associated with each node. These are required to parameterize the network.

Suppose there are n nodes in the problem domain. For a node with k parents, we need $k \log_2(n)$ bits to list its parents. To represent the conditional probabilities, the encoding length will be the product of the number of bits required to store the numerical value of each conditional probability and the total number of conditional probabilities that are required. In a Bayesian network, a conditional probability is needed for every distinct instantiation of the parent nodes and node itself (except that one of these conditional probabilities can be computed from the others due to the fact that they all sum to 1). For example, if a node that can take on 5 distinct values has 4 parents each of which can take on 3 distinct values, we will need $3^4 \times (5-1)$ conditional probabilities. Hence, under this simple scheme the total description length for a particular network will be:

$$\sum_{i=1}^{n} [k_i \log_2(n) + d(s_i - 1) \prod_{j \in F_i} s_j], \tag{1}$$

where there are n nodes; and for node i, k_i is the number of its parent nodes, s_i is the number of values it can take on, and F_i is the set of its parents; and d represents the number of bits required to store a numerical value. For a particular problem domain, n and d will be constants. This is not the only encoding scheme possible, but it is simple and it performs well in our experiments.

By looking at this equation, we see that highly connected networks require longer encodings. First, for many nodes the list of parents will get larger, and hence the list of conditional probabilities we need to store for that node will also increase. In addition, networks in which nodes that have a larger number of values have parents with a large number of values, will require longer encodings. Hence, the MDL principle will tend to favor networks in which the nodes have a smaller number of parents (i.e., networks that are less connected) and also networks in which nodes taking on a large number of values are not parents of nodes that also take on a large number of values.

As discussed in the introduction, with Bayesian networks the degree of connectivity is closely related to the computational complexity, both space and time, of using the network. Hence, our encoding scheme generates a preference for more efficient networks. That is, since the encoding length of the model is included in our evaluation of description length, we are enforcing a preference for networks that require the storage of fewer probability parameters and on which computation is more efficient. The encoding length of the model is, however,

not the only factor in determining the description length; we also have to consider the encoding length of the data given the model.

3.2 Encoding the Data Using the Model

Let us first be more precise about the form of the raw data. The task is to learn the joint distribution of a collection of random variables $\vec{X} = \{X_1, \ldots, X_n\}$. Each variable X_i has an associated collection of values $\{x_i^1, \ldots, x_i^s\}$ that it can take on, where the number of values s will in general depend on i. Every distinct choice of values for all the variables in $\vec{X}$ defines an atomic event in the underlying joint distribution and is assigned a particular probability by that distribution.

For example, we might have three random variables X_1, X_2, and X_3, with X_1 having $\{1,2\}$, X_2 having $\{1,2,3\}$, and X_3 having $\{1,2\}$ as possible values. There are $2 \times 3 \times 2$ different complete instantiations of the variables. Each of these is an atomic event in the underlying joint distribution, and has a particular probability of occurring. For example, the event in which $\{X_1 = 1, X_2 = 3, X_3 = 1\}$ is one of these atomic events.

We assume that the data points in the raw data are all atomic events. That is, each data point specifies a value for every random variable in $\vec{X}$. Furthermore, we assume that the data points are the result of independent random trials. Hence, we would expect, via the central limit theorem, that each particular instantiation of the variables would eventually appear in the database with a relative frequency approximately equal to its probability. These assumptions are standard.

Given a collection of N data points we want to encode, or store, the data as a binary string. There are various ways in which this encoding can be done, but here we are only interested in using the length of the encoding as a metric, via item 2 in the MDL principle, for comparing the merit of candidate Bayesian Networks. Hence, we can limit our attention to *character codes* (Cormen, Leiserson and Rivest, 1989, pp. 337). With character codes each atomic event is assigned a unique binary string. Each of the data points, which are all atomic events, is converted to its character code, and the N points are represented by the string formed by concatenating these character codes together. For example, say that we assign the code 0000 to the event $e_{111} = \{X_1 = 1, X_2 = 1, X_3 = 1\}$ and the code 1011 to the event $e_{232} = \{X_1 = 2, X_2 = 3, X_3 = 2\}$. Then if the raw data consists of the sequence of atomic events, $e_{111}, e_{111}, e_{232}$ then it would be encoded as the binary string 000000001011 using these character codes.

It is well known that for character codes we can minimize the length of the final binary string by taking into account the frequency of occurrence of the different atomic events. In fact, there is an algorithm for generating optimal, i.e., minimal length, character codes. This is Huffman's algorithm for generating Huffman codes (Cormen, Leiserson and Rivest, 1989). Intuitively, what we do is assign events that occur more frequently shorter codes so that the total length of the string representing the data becomes shorter. For example, if we have 1000 data points and the 12 atomic events specified above, then with a fixed length code

for each event we would need 4 bits to encode each data point, and 4000 bits to encode the entire database. On the other hand, say that event e_{111} occurs 500 times, event e_{232} 300 times, and all the other 10 events occur 20 times each. Then if we assign the code words 0 to e_{111}, 10 to e_{232} and the code words 11111, 11110, 11101, 11100, 110111, 110110, 110101, 110100, 11001, 11000, to the remaining 10 events, we will need

$$500 + 300 \times 2 + 6 \times (20 \times 5) + 4 \times (20 \times 6) = 2180$$

bits to encode the entire database. This is the minimal number of bits required using a character code.[5]

Huffman's algorithm requires as input the frequency of occurrence of each event in the database. In fact, its operation only depends on the relative frequency of occurrence. That is, the numbers 500, 300, and 20 used above could be have been replaced by 1/2, 3/10, and 2/100, where the total number of data points, 1000, has been factored out. Now say that we are expecting to accumulate more and more data points, and we wish to design a code that will be optimal in the long run as the database becomes larger and larger. Since we are assuming that the data points are generated by a fixed underlying probability distribution, we know that in the long run the relative frequency of each different atomic event in the database will tend to its probability as determined by the underlying distribution. Hence, we could use those probabilities in Huffman's algorithm to design an code that will be optimal in the long run.

Say that in the underlying distribution each atomic event e_i has probability p_i. Then Huffman's algorithm, when run using these probabilities, will assign event e_i a codeword of length approximately $-log_2(p_i)$ (Lelewer and Hirschberg, 1987). When we have N data points, where N is large, we would expect that there will be Np_i occurrences of event e_i. Hence, the length of the string encoding the database will be approximately

$$-N \sum_i p_i \log_2(p_i), \tag{2}$$

where we are summing over all possible atomic events.

Of course we don't have these probabilities p_i: if we did we could construct our Bayesian network directly from this information. Say instead that we construct, via some learning scheme, a particular Bayesian network from the raw data. This Bayesian network acts as a model of the underlying distribution and it also assigns a probability, say q_i, to every atomic event e_i. Of course, in general q_i will not be equal to p_i, as the learning scheme cannot guarantee that it will construct a perfectly accurate network. Nevertheless, the aim is for q_i to be close to p_i, and the closer it is the more accurate is our model.

The constructed Bayesian network is intended as our best "guess" representation of the underlying distribution. Hence, given that the probabilities q_i determined by the network

[5]These codes were computed using Huffman's simple algorithm. If the number of atomic events is some power of two and all events occur with equal frequency then Huffman's algorithm would generate a fixed length code. This is one of the few times when a fixed length code can be optimal.

are our best guess of the true values p_i, it makes sense to design our Huffman code using these probabilities. This means that each event e_i will be assigned a codeword of length approximately $-log_2(q_i)$ instead of its optimal value of $-log_2(p_i)$. Despite our use of the values q_i in assigning codewords, the raw data will continue to be determined by the true probabilities p_i. That is, we still expect that for large N we will have Np_i occurrences of event e_i, as p_i is the true probability of e_i occurring. Therefore, when we use the learned Bayesian network to encode the data the length of the string encoding the database will be approximately

$$-N\sum_i p_i \log_2(q_i), \tag{3}$$

where again we are summing over all atomic events. How does this encoding length compare to the encoding length if we had access to the true probabilities p_i? An old theorem due originally to Gibbs gives us the answer.

Theorem 3.1 (Gibbs) *Let p_i and q_i, $i = 1, \ldots, t$, be non-negative real numbers that sum to 1. Then*

$$-\sum_{i=1}^{t} p_i \log_2(p_i) \leq -\sum_{i=1}^{t} p_i \log_2(q_i),$$

with equality holding if and only if $\forall i. p_i = q_i$. In the summation we take $0 \log_2(0)$ to be 0.

In other words, this theorem shows that the encoding using the estimated probabilities q_i will be longer than the encoding using the true probabilities p_i. It also says that the true probabilities achieve the minimal encoding length possible.

The MDL principle says that we must choose a network that minimizes the sum of its own encoding length, which we have seen depends on the complexity of the network, and the encoding length of the data given the model, which we have seen depends on the closeness of the probabilities q_i determined by the network to the true probabilities p_i, i.e., it depends on the accuracy of the model.

We can use Equation 3 to evaluate the second item required by the MDL principle, the encoding length of the data given the model. However, there are two problems with using this equation directly. First, we do not know the values of p_i. In some cases, however, this problem can be overcome. By the law of large numbers we would expect that the event e_i will appear in the database of N points approximately Np_i times, if N is large. Hence, we can use the actual number of occurrences of e_i divided by the number of data points as an estimator for p_i. The second problem, however, is more difficult. Equation 3 involves a summation over all the atomic events, and the number of atomic events is exponential in the number of variables.[6]

[6]This also points out that we often will not be able to use the relative frequency of occurrence of the event e_i as an estimator for its probability p_i. With an exponential number of atomic events e_i some of the probabilities p_i will be so small that our database will not be able to offer reliable estimates. The database might only be large enough to estimate low-order marginals which are the union of many different atomic events.

Instead of trying to use Equation 3 directly we can use Gibbs's theorem to relate the encoding length of the data to another well known measure: Kullback-Leibler cross-entropy. Cross-entropy is an important technique in previous work on learning Bayesian networks.

Definition 3.2 [Kullback-Leibler Cross-Entropy] Let P and Q be distributions defined over the same event space, $e_1, \ldots, e_t$. Let event e_i be assigned probability p_i by P and probability q_i by Q. The Kullback-Leibler cross-entropy is a measure of closeness between two different distributions defined over the same event space. In particular, the cross-entropy between P and Q, $C(P, Q)$, is given by the equation

$$C(P, Q) = \sum_{i=1}^{t} p_i(\log_2(p_i) - \log_2(q_i)). \tag{4}$$

It follows from Gibbs's theorem that this quantity is always non-negative and that it is zero if and only if $P \equiv Q$, i.e., $\forall i.p_i = q_i$.

From Equation 2 we know that the minimal possible encoding length of the data will be $-N \sum_i p_i \log_2(p_i)$. Hence, from Equation 3 when using a model that assigns probabilities q_i the encoding length will increase by $N(\sum_i p_i(\log_2(p_i) - \log_2(q_i)))$. That is, we have the following theorem relating the encoding length of the data to the cross-entropy measure.

Theorem 3.3 *The encoding length of the data is a monotonically increasing function of the cross-entropy between the distribution defined by the model and the true distribution.*

This theorem shows that instead of using the data encoding length, Equation 3, to evaluate candidate models, we can equivalently use the cross-entropy measure, Equation 4. Furthermore, this can be accomplished in a computationally feasible manner. That is, although Equation 4 also involves a summation over an exponentially number of atomic events, we can develop an approach to evaluating cross-entropy that uses local computation over low-order marginals. This approach is an extension of previous work due to Chow and Liu (1968).

Chow and Liu (Chow and Liu, 1968) developed a method for finding a tree structure that minimized the cross-entropy, and their method was extended by Rebane and Pearl (1987) to finding polytrees with minimal cross-entropy.

Theorem 3.3 also shows that in a certain sense the MDL principle can be viewed as a generalization of the previous work of Chow and Liu (as well as that of Rebane and Pearl (1987)). If we were to ignore the complexity (encoding length) of the model and were to restrict the class of models being examined, the MDL principle would duplicate these methods. The advantage of considering both the data and the model (i.e., the sum of Equations 1 and 3) is that we can tradeoff accuracy and complexity when learning a suitable model of the underlying distribution.

4 Applying the MDL Principle

In theory the MDL principle can be applied by simply examining every possible Bayesian network that can be constructed over our set of random variables $\vec{X}$. For each of these networks we could evaluate the encoding length of the data and of the network searching for the network that minimized the sum of these encodings.

However, this approach is impractical as there are an exponential number of networks over n variables.[7] Furthermore, evaluating the encoding length, or equivalently the cross-entropy, directly also involves an exponential amount of work. Hence, we must resort to a heuristic search through the space of possible networks trying to find one that yields a low, albeit not necessarily minimal, sum of Equations 1 and 3, and we must develop a more efficient method for evaluating the cross-entropy of a candidate network.

We accomplish the heuristic search by dividing the problem into two. There can be between 0 and $n(n-1)/2$ arcs in a dag. For each possible number of different arcs we search heuristically for networks with that many arcs and low cross-entropy. By Theorem 3.3 we know that these networks will yield relatively low encoding lengths for the data. We then examine these different networks, with different numbers of arcs, and find the one that minimizes the sum of Equations 1 and 3. That is, from these low cross-entropy networks we select the one that is best according to the MDL principle.

To perform the first part of the search, i.e., to find networks with low cross-entropy, we develop some additional results based on the work of Chow and Liu (1968). These results allow us to develop a more efficient method for evaluating cross-entropy, rather than using Equation 4 which involves a sum over an exponential number of items.

4.1 Evaluating Cross-Entropy

The underlying distribution P is a joint distribution over the variables $\vec{X} = \{X_1, \ldots, X_n\}$, and any Bayesian network model will also define a joint distribution Q over these variables. If the atomic events in this joint distribution, i.e., each distinct instantiation of the variables $X_1, \ldots, X_n$, are numbered $e_1, \ldots, e_t$, and each e_i is assigned probability p_i by distribution P, and probability q_i by Q, then the cross-entropy between P and Q becomes

$$C(P,Q) = \sum_{i=1}^{t} p_i \log_2 \frac{p_i}{q_i},$$

where the sum extends over all atomic events.

To develop a method for evaluating the cross-entropy of the distribution Q specified by the Bayesian network that avoids summing over the exponentially many atomic events, we follow Chow and Liu (1968) and take advantage of the fact that Q has a special form. In

[7] Robinson (1976) gives a recurrence that can be used to calculate this number.

particular, since Q is specified by a Bayesian network it can be decomposed into a product of lower-order marginals.

In an arbitrary Bayesian network $Q(\vec{X})$ will take the form (Pearl, 1988):

$$\begin{aligned} Q(\vec{X}) &= Q(X_1 \mid F_{X_1})Q(X_2 \mid F_{X_2}) \ldots Q(X_n \mid F_{X_n}), \\ &\approx P(X_1 \mid F_{X_1})P(X_2 \mid F_{X_2}) \ldots P(X_n \mid F_{X_n}), \end{aligned} \tag{5}$$

where F_{X_i} is the, possibly empty, set of parents of X_i.

Every variable X_i corresponds to a node in our network. This node will have some set of parents, F_{X_i}, determined by the network's topology. To parameterize the network we need to know the probability of X_i taking on a value v given that its parents take on the values $\vec{u}$, for every possible v and $\vec{u}$. An unbiased estimator for this probability is $N_{v,\vec{u}}/N_{\vec{u}}$, where $N_{v,\vec{u}}$ is the number of data points in which X_i has value v and its parents have the values $\vec{u}$, and $N_{\vec{u}}$ is the number of data points in which X_i's parents have the values $\vec{u}$. We use these frequency counts as estimates for the low-order marginals $Q(X_i|F_{X_i})$ that appear in Q's product decomposition. However, since the raw data was generated by the underlying distribution P, by the law of large numbers we would expect that the raw data frequency counts will be close to the low-order marginals over P. That is, we could expect $N_{v,\vec{u}}/N_{\vec{u}}$ to be close to $P(X_i = v|F_{X_i} = \vec{u})$. Hence, the values we use for $Q(X_i|F_{X_i})$ should be close to the values for $P(X_i|F_{X_i})$, and we can make the substitution above.[8]

Note, however, that just because these low-order marginals for Q and P are close does not mean that the joint probabilities $Q(\vec{X})$ and $P(\vec{X})$ are also close. That is, as distributions, Q and P need not be close. In particular, although $Q(\vec{X})$ is equal to a product of low-order marginals involving P, it might not be the case that $P(\vec{X})$ is also equal to this product. $Q(\vec{X})$ is an *estimate* of $P(\vec{X})$ precisely because it is assuming that the joint distribution can be written as a product of lower-order terms, and the accuracy of our estimate will depend on how well we chose the lower-order terms.

Chow and Liu (1968) proved the following theorem:

Theorem 4.1 (Chow and Liu) *If the* mutual information *between any two nodes X_i and X_j is defined as*

$$W(X_i, X_j) = \sum_{X_i, X_j} P(X_i, X_j) \log_2 \frac{P(X_i, X_j)}{P(X_i)P(X_j)} \tag{6}$$

where we are summing over all possible values of X_i and X_j, then by assigning to every arc between two nodes X_i and X_j a weight equal to $W(X_i, X_j)$, cross-entropy $C(P, Q)$ over all tree structured *distributions Q is minimized when the structure representing $Q(\vec{X})$ is a maximum weight spanning tree.*

[8]Lower-order marginals like $P(X_i|X_j)$ can be estimated fairly accurately from a reasonably sized database. Such marginal include an exponential number of atomic events. Hence, we would expect that a statistically useful sample could be accumulated from a reasonably sized database.

Using this theorem they developed an algorithm for learning the best tree model by constructing the maximum weight spanning tree. Note that by using low cost local computation they can evaluate the mutual information weight and use it to find a network of minimal cross-entropy without ever computing the actual cross-entropy. As we pointed out above it is very expensive to compute cross-entropy directly. Hence, we want to deal with arbitrary Bayesian networks using a similar technique, and we accomplish this by extending Chow and Liu's method.

We can extend Chow and Liu's approach to the general case by defining a new weight measure for a node, X_i, with respect to an arbitrary set of parents as follows:

$$W(X_i, F_{X_i}) = \sum_{X_i, F_{X_i}} P(X_i, F_{X_i}) \log_2 \frac{P(X_i, F_{X_i})}{P(X_i)P(F_{X_i})} \tag{7}$$

where we are summing over all possible values that X_i and its parents F_{X_i} can take.[9] Note that if the network is a tree each F_{X_i} will either be empty of a singleton, and our formula will reduce to that of Chow and Liu's.

The following theorem holds.

Theorem 4.2 *$C(P, Q)$ is a monotonically decreasing function of*

$$\sum_{i=1, F_{X_i} \neq \emptyset}^{n} W(X_i, F_{X_i}). \tag{8}$$

Hence, it will be minimized if and only if the sum is maximized.

The proof is given in the Appendix. The summation gives the total weight of the directed acyclic graph according to the node by node weight measure defined in Equation 7.

In conclusion, given probabilities computed from the raw data, we can calculate the weight of various candidate network structures using local computation at each node. Our theorem shows that structures with greater weight are closer, in terms of cross-entropy, to the underlying distribution. If we can find a directed acyclic graph with maximum total weight, then the probability distribution of this structure will be closest to the underlying distribution of the raw data with respect to the cross-entropy measure. Hence, by Theorem 3.3 it will yield the shortest encoding of the data.

It should be noted that we cannot simply use the encoding length of the data without considering the encoding length of the network. In fact, for every probability distribution P, if we let

$$Q(\vec{X}) = P(X_1 \mid X_2, \ldots, X_n)P(X_2 \mid X_3, \ldots, X_n) \ldots P(X_n), \tag{9}$$

[9]Note that as the number of X_i's parents increase the number of terms in this summation still suffers from an exponential growth. Hence, when we use this measure to search through the space of candidate networks we are still limited to networks of "tractable" connectivity. Nevertheless, this measure is still a significant improvement over the direct computation of cross-entropy, which is exponential regardless of the network's topology.

then $Q \equiv P$. In other words, if we construct the multiply-connected network corresponding to the structure on the right side of the above expression, the probability distribution defined by this structure will absolutely coincide with the underlying distribution of the raw data, and hence it will have lowest possible cross-entropy and highest possible weight. However, this structure is a complete graph, and worse still, it does not convey any information since it can represent any distribution. This indicates that when we allow structures of arbitrarily complex topology, we can obtain a trivial match with the underlying distribution.

The MDL principle, however, allows us to avoid this difficulty. It considers not only the accuracy of the network but also its complexity. The totally connected network given in this equation will require a very large encoding. For example, to encode node X_1 we will need to store an exponential number of probability parameters. Hence, there will probably be a less complex network with a shorter encoding that is still able to produce a reasonably short encoding of the data, i.e., that is still reasonably accurate. When we evaluate the total description length, the sum of the encoding lengths for the model and the data, this less complex network will be preferred.

The next theorem provides some additional information about the relationship between accuracy and complexity of the networks.

Theorem 4.3 *Let M_i be the maximum weight of all networks that have i arcs, then*

$$i > j \Rightarrow M_i \geq M_j .$$

For the proof of this theorem see the Appendix. That is, we can always increase the quality of the learned network, i.e., decrease the error in the sense of decreasing the cross-entropy, by increasing the topological complexity, i.e., by learning networks with more arcs. Again, it is only through our use of the MDL principle that we avoid the difficulty of always preferring more complex networks.

4.2 Searching for Low Cross-Entropy Networks

Given our ability to evaluate the cross-entropy of a network through an evaluation of its weight, we proceed to describe a heuristic search routine that searches for a good network model.

Many different heuristic search procedures are possible, but we have developed one that is based on the notion of ensuring that we spend an equal amount of time searching among the simpler networks, i.e., ones with fewer arcs, as we do searching among the more complex ones, i.e., ones with more arcs. To do this we maintain separate sets of candidate graphs, one set for each possible number of arcs, and we time-share the search between these sets.

Say that we are searching for a network defined over the variables $\vec{X} = \{X_1, \ldots, X_n\}$. These variables become the nodes in the network. For directed acyclic networks we can have between 0 and $n(n-1)/2$ arcs between these nodes. Hence, we maintain $n(n-1)/2 + 1$

separate search sets which we denote as the sets S_i, for $0 \leq i \leq n(n-1)/2$. Each search sets S_i contains candidate i-arc networks.[10]

Before we start the search, however, we calculate the mutual information $W(X_i, X_j)$, Equation 6, between every pair of distinct nodes X_i, $X_j \in \vec{X}$, $i \neq j$. These weights give us a rough idea of the interdependency between every pair of nodes. Using these weights we generate a list of all pairs of distinct nodes, PAIRS, in which the pairs are sorted by their mutual information weight. That is, the first pair of nodes on PAIRS has highest mutual information.

Within each search set S_i we perform best-first search. The individual elements of S_i have two components: a candidate network with i arcs, and a pair of nodes between which a new arc could be added in the candidate network without causing a cycle. The elements of S_i are separated into an OPEN and a CLOSED list, and we perform best-first search using these lists. The elements on the OPEN list are ordered by heuristic value, which is calculated as the weight of the element's network, calculated using Equation 8, plus the mutual information weight between the element's pair of nodes, calculated using Equation 6. When we run the search procedure on the set S_i we choose the element on OPEN with highest heuristic value and expand it via the following procedure.

1. Remove the element with greatest heuristic value from the S_i's OPEN list and copy it onto the CLOSED list. Let the element's network be G_{old} and the element's pair of nodes be (X_i, X_j).

2. Invoke the PD-procedure on G_{old} and (X_i, X_j) to get a new network G_{new}. The PD-procedure, described in detail below, adds an arc between the nodes X_i and X_j creating a new network G_{new}. It decides on the direction of this new arc, i.e., if it should be $X_i \Rightarrow X_j$ or $X_j \Rightarrow X_i$, picking the direction that most increases the network's accuracy. In the process it might also reverse the direction of other arcs in G_{old}. Note that G_{new} is a network with $i+1$ arcs, so it must be placed inside the search set S_{i+1} not back into S_i.

3. If G_{new} is fully connected, we place a copy of it into a list of final candidate networks, FINAL.

4. Next we make a new search element consisting of G_{new} and the first pair of nodes from PAIRS that appear after the old pair (X_i, X_j) and between which an arc could be added without generating a cycle in G_{new}. Then, we insert this element into the OPEN list of the search set S_{i+1}, placed in correct order according to the heuristic function.

5. Finally, we make a new search element consisting of G_{old} and the first pair of nodes from PAIRS that appear after the old pair (X_i, X_j) and between which an arc could be

[10] These search sets are dynamically generated as search progresses, so search may stop before all $n(n-1)/2$ search sets are generated.

added without generating a cycle in G_{old}. This element is inserted into the OPEN list of search set S_i, placed in the correct order according to the heuristic function.

Note that the expansion of an element in S_i generates two new elements, one placed on the OPEN list of S_i and the other placed on the OPEN list of S_{i+1}. This means that we can start the search by creating just the list S_0 containing one element on its OPEN list: the unique network with no arcs and the first pair of nodes on PAIRS.

Example 3 Say that we have the nodes X_1, X_2, and X_3, and say that PAIRS is in the order $\{(X_1, X_2), (X_2, X_3), (X_1, X_3)\}$. Initially, we would create the search set S_0 containing a single element on its OPEN list: $(\{\}, (X_1, X_2))$, where $\{\}$ indicates that this element's network has no arcs.

This element would be expanded when we run the search procedure on S_0. It would be removed from OPEN and the new element $(\{\}, (X_2, X_3))$ would be placed on S_0's OPEN list. Also, the new search set S_1 would be created and the element $(\{X_1 \Rightarrow X_2\}, (X_2, X_3))$ would be placed on its OPEN list (for now we ignore how the direction of the arcs is determined, these details are given below).

Now we have two search sets to run our search on. There are various schedules for timesharing the search between these sets, but say that we once again search inside S_0. This would convert S_0's OPEN list to $((\{\}, (X_1, X_3))$ and the element $(\{X_2 \Rightarrow X_3\}, (X_1, X_3))$ would be added to S_1's OPEN list. Now search would be performed in S_1. Assuming that higher heuristic value was given to the element $(\{X_1 \Rightarrow X_2\}, (X_2, X_3))$, this would result in the new element $(\{X_1 \Rightarrow X_2\}, (X_1, X_3))$ being added to S_1's OPEN list. Also a new search set S_2 would be created containing the element $(\{X_1 \Rightarrow X_2, X_2 \Rightarrow X_3\}, (X_1, X_3))$. Since, the network in this new element is connected it would also be added to FINAL.

■

To control the search as a whole we allocate a fixed amount of resources for searching within each search set. Once those resources have been consumed in all the search sets, search is terminated. At termination there will be some collection of connected networks in FINAL. These networks are then evaluated to find the one that is best according to the MDL principle, i.e., the one that minimizes the sum of the description length of the model and of the data given the model. Since our heuristics tries to grow networks by adding higher weight arcs, the completed networks that are found first, i.e., that are placed on FINAL before search terminates, tend to be of high weight. That is, they tend to be fairly accurate models. The more complex networks will generally be more accurate, as demonstrated by Theorem 4.3, but the MDL principle will trade this off against their complexity. Thus, our selection from FINAL will be the simplest possible model of reasonable accuracy.

There are $O(n^2)$ search sets, since there are at most $n(n-1)/2$ directed arcs in any acyclic network. Furthermore, the most complex task each time an element is expanded is the PD-procedure (described below). In practice complexity of this procedure is $O(N)$, where N is the number of raw data points. In practice we have found that a resource

limit of $O(n^2)$ node expansions within each search set yields very satisfactory performance. Hence, the complexity of the overall searching algorithm is $O(Nn^4)$. Various refinements are possible for dealing with networks containing a larger number of nodes, i.e., when n is very large. However, the aim of our current work has been to demonstrate the viability of our method. For this we have found our current approach to be successful, and hence have left such refinements for future work.

The only thing left to describe is the PD, or parents-detection, procedure. When we expand an element on one of the search lists we grow the element's network by adding an arc between the pair of nodes specified in the element's second component. Since the arcs are directed, we have to decide upon a direction for the new arc. The PD procedure decides upon a locally optimal way of placing a new arc into an existing network so as to maximize the weight of the resulting network. It considers the direction the new arc should have, and it also searches the parents of the nodes connected by the new arc to determine if the direction of other arcs already in the network should be reversed.

Input : A network G_{old}.
: An pair of nodes (X_i, X_j) between which an arc is to be added.
Output : A new network G_{new} with the arc added and some other arcs possibly reversed.

1. Create a new network by adding the arc $(X_i \Rightarrow X_j)$ to G_{old}. In this new network we then search locally to determine if we can increase its weight by reversing the direction of some of its arcs. This is accomplished via the following steps.

   (a) Determine the optimal directionality of the arcs attached directly to X_j by examining which directions maximize the weight measure. Some of these arcs may be reversed by this process.

   (b) If the direction of an existing arc is reversed then perform the above directionality determination step on the other node affected.

2. Repeat the above steps except this time with the new network formed by adding the arc $(X_j \Rightarrow X_i)$ to G_{old}.

3. Select the network of greatest weight from the two networks found in the above steps. This network is the output.

The complexity of the PD-procedure will depend on how many arcs are attached to X_i and X_j (step 1(a) above), and in the worst case if the directions of a number of these arcs are reversed, it will depend on the size of the set of arcs reachable from X_i and X_j (step 1(b) above). However, in practice we have never found more than a small number of arcs being examined by the procedure. Generally, changes in arc direction do not propagate very far. Hence, we can treat this factor as a constant. Each time an arc's direction is examined in step 1, we must evaluate the weight measure for each direction, Equation 8. This involves

the computation of marginal probabilities estimated from the raw data. In practice, we have found that this computation has complexity $O(N)$, where N is the number of cases in the database. Equation 8 involves a summation over all instantiations of the parent nodes of every node, and the number of instantiations is exponential in the number of parents. However, in the networks examined by our search algorithm the number of parents any node possesses never exceeds a constant bound. Hence, the complexity of evaluating Equation 8 is dominated by the number of data points, N. We have also found that the PD-procedure is drastically speeded up by hashing some of the probabilities computed from the raw data: some of these probabilities are used repeatedly in different weight computations.

5 Evaluating The Experimental Results

A common approach to evaluating various learning algorithms has been to generate raw data from a predetermined network and then to compare the network learned from that data with the original, the aim being to recapture the original. For example, this is the technique used by Cooper and Herskovits (1991). An implicit assumption of this approach is that the aim of learning is to reconstruct the true distribution. However, if one takes the aim of learning to be the construction of a *useful* model, i.e., one that is a good tradeoff between accuracy and complexity, as we have argued for, then this approach is not suitable. In particular, the aim of our approach is not to recapture the original distribution.

Hence, to evaluate our experimental results we use a different approach. Since Bayesian networks are commonly used to manage belief update, two networks can be regarded as being approximately equivalent, in a practical sense, if they exhibit close results after belief update. Belief update occurs when one of the nodes in the network is instantiated to a particular value, and all of the other nodes have their probabilities updated to the posterior probabilities that arise from this instantiation. Each node's new probability is determined by the probabilistic influences of the nodes in its local neighborhood. This local neighborhood shields the node from the rest of the nodes in the network.

When two different networks are defined over the same set of nodes, i.e., they differ only in terms of the connections between the nodes, a node, say X^*, can have a different local neighborhood in each of the networks. Since it is this local neighborhood that determines X^*'s posterior probabilities after update, we can compare the "closeness after update" of these two networks by comparing X^*'s local neighborhood in these two networks. If we do this for every node we will obtain a point-to-point measure of the closeness of the two networks. That is, we will obtain for each node a measure of how belief update will differ between the two networks at that node. Taking the average of this node-to-node distance we obtain an estimate of how close the two networks are in terms of their total belief updating behavior.

In a Bayesian network the neighborhood of a node has the property that given an instantiation of the nodes in the neighborhood, the node is independent of the rest of the network.

The neighborhood of a node X^* consists of three types of nodes: the direct parents of X^*, the direct successors of X^*, and the direct parents of X^*'s direct successors. Let F_1 and F_2 be the set of neighborhood nodes of X^* in two different Bayesian networks G_1 and G_2 respectively. Furthermore, let P_1 and P_2 be the probability distributions defined by G_1 and G_2, respectively. We evaluate the distance between X^* in G_1 and X^* in G_2 by the formula

$$\frac{1}{\|F_1\|}\sum_{F_1} D(P_1(X^* \mid F_1), P_2(X^* \mid F_1)) + \frac{1}{\|F_2\|}\sum_{F_2} D(P_2(X^* \mid F_2), P_1(X^* \mid F_2)) \qquad (10)$$

where $\|F_i\|$ is the total number of all possible instantiations of the neighborhood nodes F_i, we are summing over all instantiations of F_i, and D is a function measuring the "distance" between two probability distributions. Note that once we have chosen a particular instantiation for F_i the probabilities $P_j(X^*|F_i)$ define a distribution over the possible values of X^*. That is, we are in fact applying D to two distributions.

Since in network G_1 instantiating F_1 renders X^* independent of all other nodes in G_1, we can examine the behavior of node X^* in G_1 without considering nodes not in F_1. Every distinct instantiation of F_1 will yield a particular distribution over the values of X^*. In G_2 however, F_1 will not be X^*'s neighborhood. Nevertheless, we can still examine the distribution over X^*'s values when the nodes in F_1 are instantiated in G_2 using various belief updating algorithms. The distance between these two distributions, over X^*'s values in G_1 and over X^*'s values in G_2, gives a measure of how divergent belief update will be in these two networks for this particular instantiation of F_1. The formula above simply takes the average of this distance over all possible instantiations of X^*'s neighborhood F_1. Finally, we can apply the same reasoning with X^*'s neighborhood in G_2, F_2. Hence, the second summation measures the distance "in the other direction." This makes our formula symmetric in its arguments.

To apply Equation 10 we simply have to compute the probability distributions $P_j(X^*|F_i)$. This can be done, given the networks G_1 and G_2, with well known belief updating algorithms like (Jensen, Lauritzen and Olesen, 1990). Then we have to determine a suitable distance function between distributions, D. We have used two implementations of D. For the first we simply take the average of the absolute difference in the probabilities at each point in the distribution. For example, if we have two distributions over a coin toss, with $P_1 = \{p, 1-p\}$, and $P_2 = \{q, 1-q\}$, then this distance function would yield $|p-q|$ as the distance between P_1 and P_2. For the second we use the Kullback-Leibler cross-entropy measure given in Equation 4.

Finally, to calculate the total distance between the two networks we simply average the node to node distance (Equation 10) over all of the nodes. We use this method, with the two distribution distance functions D described, to evaluate our learning mechanism. In particular, following Cooper and Herskovits (1991), we construct sample original networks. Then we generate raw data using Henrion's logic sampling technique (1987), and apply our learning mechanism to the data to generate a learned network. We can then compare

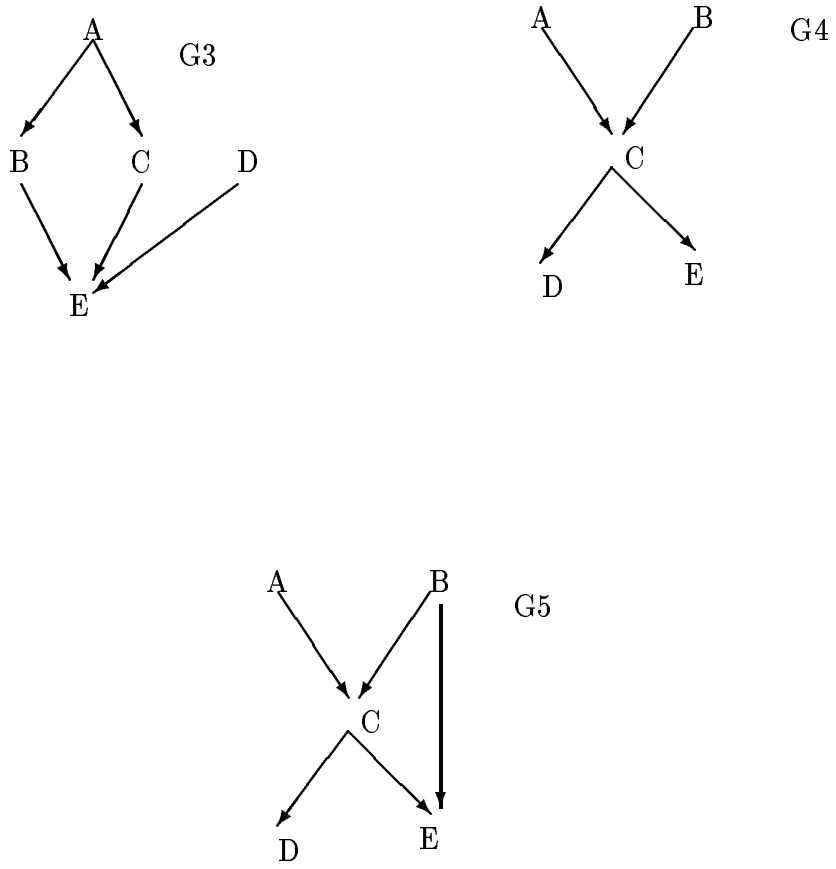


Figure 2: Small Bayesian Belief Networks

the learned network with the original by examining any structural differences and by also evaluating the distance between them.

6 Experimental Results

We have done three sets of experiments to demonstrate the feasibility of our approach. The first set of experiments consisted of a number of Bayesian networks that were composed of small number of variables (5) as shown in Figure 2. Some of these structures are multiply-connected networks.

The second experiment consisted of learning a Bayesian network with a fairly large number of variables (37 nodes and 46 arcs). This network was derived from a real-world application in medical diagnosis (Beinlich et al., 1989) and is known as the ALARM network (Figure 3).

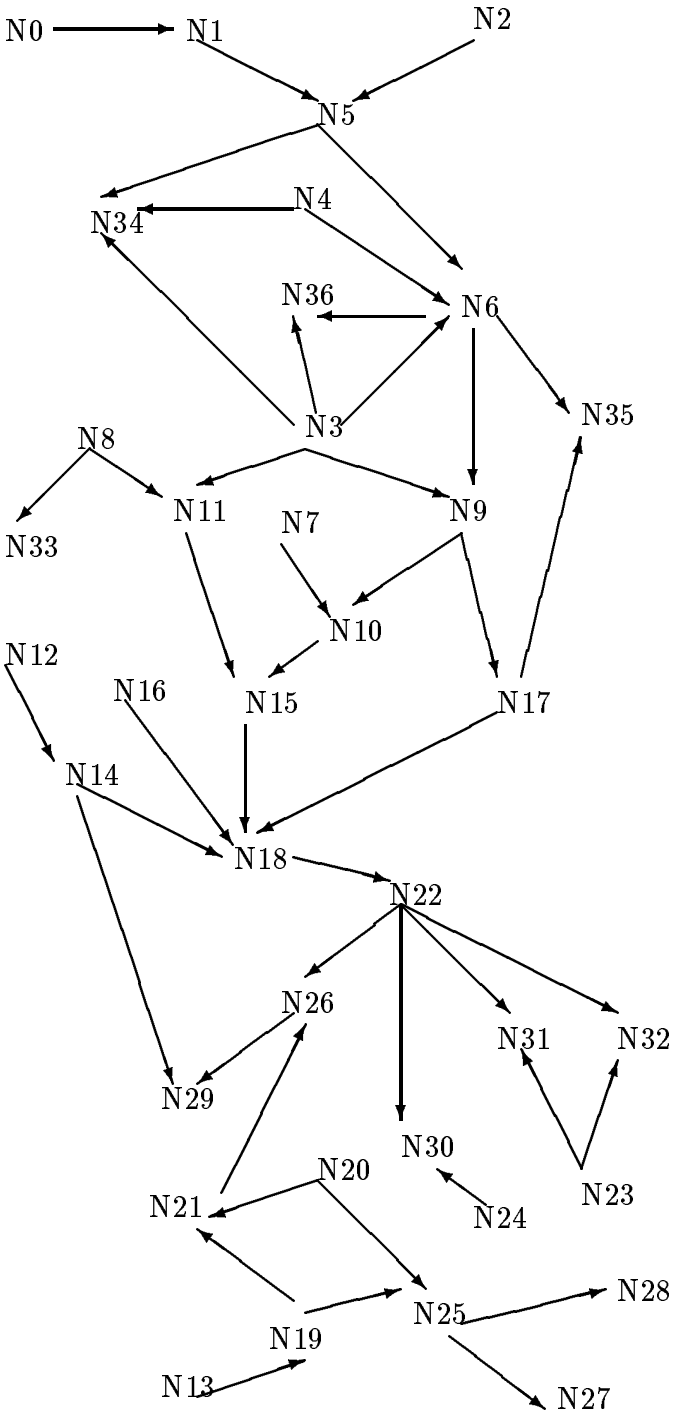


Figure 3: The ALARM Belief Network Structure

The third experiment consisted of learning a small Bayesian network, as shown in Figure 4. We experimented by varying the conditional probability parameters of this network. Here the aim was to demonstrate that our procedure could often learn a simpler network that was very close to the original.

In the first set of experiments, since the networks are small we were able to perform an exhaustive search of the possible acyclic networks with different numbers of arcs. That is, we did not need the heuristic search approximation. Thus, the maximum weight network structure for each different number of arcs was found. In all cases, we observed that the weight was non-decreasing as the number of arcs becomes higher, as predicted by Theorem 4.3. After applying the MDL principle by calculating the description lengths of the networks, the network with the minimum description length was selected. In all these cases we found that the learned network was exactly the same as the one used to generate the raw data.

In the second experiment, we used the described heuristic search algorithm to find the network structure. The Bayesian network recovered by the algorithm was found to be close to the original network structure. Two different arcs and three missing arcs were found. The "distance" between this learned structure and the original structure was small. They were 0.03 and 0.016 obtained by the average difference and the Kullback-Leibler measure for the function D respectively. One additional feature of our approach, in particular a feature of our heuristic search algorithm, is that we did not require a user supplied ordering of variables, cf. (Cooper and Herskovits, 1991). We feel that this experiment demonstrates that our approach is feasible for recovering Bayesian networks of practical size.

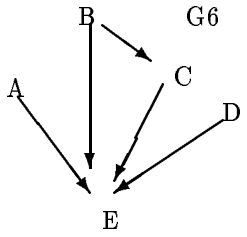
In the third set of experiments, the original Bayesian network $G6$ consisted of 5 nodes and 5 arcs. We varied the conditional probability parameters during the process of generating the raw data obtaining four different sets of raw data. Exhaustive searching was then carried out and the MDL learning algorithm was applied to each of these sets of raw data. Different learned structures were obtained, all of which were extremely close to the original network as measured by both of our distance formulas. In one case the original network was recovered.

This experiment demonstrates that our algorithm yields a tradeoff between accuracy and complexity of the learned structures: in all cases where the original network was not recovered a simpler network was learned. The type of structure learned depends on the parameters, as each set of parameters, in conjunction with the structure, defines a different probability distribution. Some of these distributions can be accurately modeled with simpler structures. In the first case, the distribution defined by the parameters did not have a simpler model of sufficient accuracy, but in the other cases it did.

7 Conclusions

We have argued in this paper that the purpose of learning a Bayesian network from raw data is not to recover the underlying distribution, as this distribution might be too complex to use. Rather, we should attempt to learn a useful model of the underlying phenomena. Hence,

Original Bayesian Network Structure :



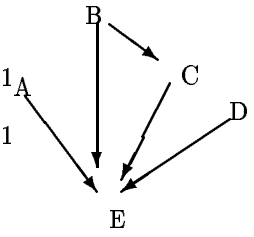
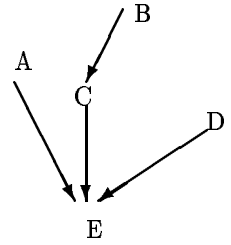
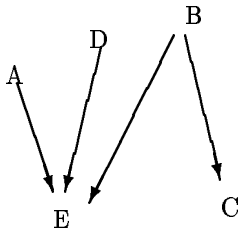
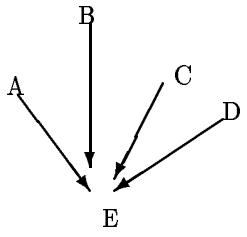
Original Conditional Parameters	Learned Structures	Overall Distance From Original Network: Kullback-Leibler	Average
$P(c1 \mid b1) = 0.8 \;\; P(c1 \mid b0) = 0.2$ $P(e1 \mid a1,b1,c1,d1) = 0.9 \;\; P(e1 \mid a0,b1,c1,d1) = 0.1$ $P(e1 \mid a1,b0,c1,d1) = 0.15 \;\; P(e1 \mid a0,b0,c1,d1) = 0.1$ $P(e1 \mid a1,b1,c0,d1) = 0.1 \;\; P(e1 \mid a0,b1,c0,d1) = 0.1$ $P(e1 \mid a1,b0,c0,d1) = 0.08 \;\; P(e1 \mid a0,b0,c0,d1) = 0.1$ $P(e1 \mid a1,b1,c1,d0) = 0.1 \;\; P(e1 \mid a0,b1,c1,d0) = 0.1$ $P(e1 \mid a1,b0,c1,d0) = 0.1 \;\; P(e1 \mid a0,b0,c1,d0) = 0.1$ $P(e1 \mid a1,b1,c0,d0) = 0.1 \;\; P(e1 \mid a0,b1,c0,d0) = 0.1$ $P(e1 \mid a1,b0,c0,d0) = 0.1 \;\; P(e1 \mid a0,b0,c0,d0) = 0.1$		0.0	0.0
$P(c1 \mid b1) = 0.85 \;\; P(c1 \mid b0) = 0.2$ $P(e1 \mid a1,b1,c1,d1) = 0.9 \;\; P(e1 \mid a0,b1,c1,d1) = 0.1$ $P(e1 \mid a1,b0,c1,d1) = 0.8 \;\; P(e1 \mid a0,b0,c1,d1) = 0.1$ $P(e1 \mid a1,b1,c0,d1) = 0.1 \;\; P(e1 \mid a0,b1,c0,d1) = 0.13$ $P(e1 \mid a1,b0,c0,d1) = 0.1 \;\; P(e1 \mid a0,b0,c0,d1) = 0.1$ $P(e1 \mid a1,b1,c1,d0) = 0.09 \;\; P(e1 \mid a0,b1,c1,d0) = 0.1$ $P(e1 \mid a1,b0,c1,d0) = 0.1 \;\; P(e1 \mid a0,b0,c1,d0) = 0.1$ $P(e1 \mid a1,b1,c0,d0) = 0.11 \;\; P(e1 \mid a0,b1,c0,d0) = 0.1$ $P(e1 \mid a1,b0,c0,d0) = 0.1 \;\; P(e1 \mid a0,b0,c0,d0) = 0.1$		0.004	0.02
$P(c1 \mid b1) = 0.8 \;\; P(c1 \mid b0) = 0.15$ $P(e1 \mid a1,b1,c1,d1) = 0.9 \;\; P(e1 \mid a0,b1,c1,d1) = 0.1$ $P(e1 \mid a1,b0,c1,d1) = 0.12 \;\; P(e1 \mid a0,b0,c1,d1) = 0.087$ $P(e1 \mid a1,b1,c0,d1) = 0.8 \;\; P(e1 \mid a0,b1,c0,d1) = 0.1$ $P(e1 \mid a1,b0,c0,d1) = 0.1 \;\; P(e1 \mid a0,b0,c0,d1) = 0.1$ $P(e1 \mid a1,b1,c1,d0) = 0.1 \;\; P(e1 \mid a0,b1,c1,d0) = 0.1$ $P(e1 \mid a1,b0,c1,d0) = 0.1 \;\; P(e1 \mid a0,b0,c1,d0) = 0.1$ $P(e1 \mid a1,b1,c0,d0) = 0.1 \;\; P(e1 \mid a0,b1,c0,d0) = 0.1$ $P(e1 \mid a1,b0,c0,d0) = 0.1 \;\; P(e1 \mid a0,b0,c0,d0) = 0.1$		0.0014	0.014
$P(c1 \mid b1) = 0.3 \;\; P(c1 \mid b0) = 0.2$ $P(e1 \mid a1,b1,c1,d1) = 0.9 \;\; P(e1 \mid a0,b1,c1,d1) = 0.1$ $P(e1 \mid a1,b0,c1,d1) = 0.1 \;\; P(e1 \mid a0,b0,c1,d1) = 0.1$ $P(e1 \mid a1,b1,c0,d1) = 0.1 \;\; P(e1 \mid a0,b1,c0,d1) = 0.1$ $P(e1 \mid a1,b0,c0,d1) = 0.1 \;\; P(e1 \mid a0,b0,c0,d1) = 0.1$ $P(e1 \mid a1,b1,c1,d0) = 0.1 \;\; P(e1 \mid a0,b1,c1,d0) = 0.1$ $P(e1 \mid a1,b0,c1,d0) = 0.1 \;\; P(e1 \mid a0,b0,c1,d0) = 0.1$ $P(e1 \mid a1,b1,c0,d0) = 0.1 \;\; P(e1 \mid a0,b1,c0,d0) = 0.1$ $P(e1 \mid a1,b0,c0,d0) = 0.1 \;\; P(e1 \mid a0,b0,c0,d0) = 0.1$		0.0003	0.01

Figure 4: The Quality of Learned Networks

there should be some tradeoff between accuracy and complexity. The MDL principle has as its rational this same tradeoff, and it can be naturally applied to this particular problem. We have discussed in detail how the MDL principle can be applied and have pointed out its relationship to the method of minimizing cross-entropy. Using this relationship we have extended the results of Chow and Liu relating cross-entropy to a weighing function on the nodes. This has allowed us to develop a heuristic search algorithm for networks that minimize cross-entropy. These networks minimize the encoding length of the data, and when we also consider the complexity of the network we can obtain models that are good under the MDL metric. Our experimental results demonstrate that our algorithm does in fact perform this tradeoff, and further that it can be applied to networks of reasonable size.

There are a number of issues that arise which require future research. One issue is the search mechanism. We are currently dividing the task into first searching for a network that minimizes the encoding length of the data and then searching through the resulting networks for one that minimizes the total description length. This method has been successful in practice, but we are also investigating other mechanisms. In particular, it seems reasonable to combine both phases into one search. Another important component that has not yet been addressed is the accuracy of the raw data. In general, there will be a limited quantity of raw data, and certain parameters can only be estimated with limited accuracy. We are investigating methods for taking into account the accuracy of the data in the construction. For example, nodes with many parents will require higher-order marginal probabilities as parameters. Estimates of such parameters from the raw data will in general be less accurate. Hence, there might be additional reasons to discourage the learning of complex networks. Finally, there might be partial information about the domain. For example, we might know of causal relationships in the domain that bias us towards making certain nodes parents of other nodes. The issue that arises is how can this information be used during learning. We are investigating some approaches to this problem.

Acknowledgements

This work was supported by NSERC under their Operating Grants Program and by the Institute for Robotics and Intelligent Systems. Wai Lam's work was additionally supported through an Ontario Government Scholarship. The authors' e-mail addresses are `wlam1@math.waterloo.edu` and `fbacchus@logos.waterloo.edu`. We also wish to thank the referees for many helpful comments that allowed us to make considerable improvements to the paper.

Appendix

Proof of Theorem 4.2 $C(P,Q)$ is a monotonically decreasing function of $\sum_{i=1,F_{X_i}\neq\emptyset}^{n} W(X_i, F_{X_i})$. Hence, it will be minimized if and only if the sum is maximized. (All logarithms are to the

base 2).
Proof:

$$\begin{aligned} C(P(\vec{X}),Q(\vec{X})) &= \sum_{\vec{X}} P(\vec{X})\log P(\vec{X}) - \sum_{\vec{X}} P(\vec{X})\log Q(\vec{X}) \\ &= \sum_{\vec{X}} P(\vec{X})\log P(\vec{X}) - \sum_{\vec{X}} P(\vec{X})\sum_{i=1}^{n}\log P(X_i \mid F_{X_i}) \\ &= \sum_{\vec{X}} P(\vec{X})\log P(\vec{X}) - \sum_{\vec{X}} P(\vec{X})\sum_{i=1,F_{X_i}\neq\emptyset}^{n}\log \frac{P(X_i,F_{X_i})}{P(X_i)P(F_{X_i})} \\ &\quad - \sum_{\vec{X}} P(\vec{X})\sum_{i=1}^{n}\log P(X_i) \end{aligned} \tag{11}$$

Since X_i is a component of $\vec{X}$, we have

$$\sum_{\vec{X}} P(\vec{X})\log P(X_i) = \sum_{X_i} P(X_i)\log P(X_i)$$

If we define $H(\vec{X}) = -\sum_{\vec{X}} P(\vec{X})\log P(\vec{X})$, the first term and the third term in Equation 11 can be expressed as $-H(\vec{X})$ and $\sum_{i=1}^{n} H(X_i)$ respectively. And since $P(X_i,F_{X_i})$ is a component of $P(\vec{X})$, we have:

$$\begin{aligned} \sum_{\vec{X}} P(\vec{X})\log \frac{P(X_i,F_{X_i})}{P(X_i)P(F_{X_i})} &= \sum_{X_i,F_{X_i}} P(X_i,F_{X_i})\log \frac{P(X_i,F_{X_i})}{P(X_i)P(F_{X_i})} \\ &= W(X_i,F_{X_i}) \end{aligned}$$

As a result, the closeness measure becomes:

$$C(P(\vec{X}),Q(\vec{X})) = -\sum_{i=1,F_{X_i}\neq\emptyset}^{n} W(X_i,F_{X_i}) + \sum_{i=1}^{n} H(X_i) - H(\vec{X}) \tag{12}$$

Since the last two terms in Equation 12 are independent of the structure of the network and the closeness measure $C(P(\vec{X}),Q(\vec{X}))$ is non-negative, $C(P(\vec{X}),Q(\vec{X}))$ decreases monotonically as the total weight (i.e., the first term) of the graph increases. Hence, it will be minimized if and only if the total weight is maximized. ■

Proof of Theorem 4.3 Let M_i be the maximum weight of all learned networks that have i arcs, then

$$i > j \Rightarrow M_i \geq M_j .$$

Proof: It is sufficient to prove the following inequality (All logarithms are to the base 2):

$$\sum_{X_i\vec{A}\vec{B}} P(X_i\vec{A}\vec{B})\log \frac{P(X_i\vec{A}\vec{B})}{P(X_i)P(\vec{A}\vec{B})} \geq \sum_{X_i\vec{A}} P(X_i\vec{A})\log \frac{P(X_i\vec{A})}{P(X_i)P(\vec{A})}$$

where X_i is an arbitrary node, and $\vec{A}$ and $\vec{B}$ are two arbitrary disjoint sets of nodes in a graph. Given the graph that has maximum weight M_i among all graphs with i arcs, we can find a node X_i such that we can add one more parent to X_i without creating cycle. Obviously, this X_i must exist for all acyclic graphs (except for complete graphs). This inequality shows that by increasing the number of parents of a node, its weight will not decrease. Hence, we can construct a graph with $i+1$ arcs that has weight greater or equal to M_i. By simple induction, the theorem holds.

To prove this inequality, consider:

$$\begin{aligned}
&\sum_{X_i\vec{A}} P(X_i\vec{A}) \log \frac{P(X_i\vec{A})}{P(X_i)P(\vec{A})} - \sum_{X_i\vec{A}\vec{B}} P(X_i\vec{A}\vec{B}) \log \frac{P(X_i\vec{A}\vec{B})}{P(X_i)P(\vec{A}\vec{B})} \\
= &\sum_{X_i\vec{A}\vec{B}} P(X_i\vec{A}\vec{B}) \log \frac{P(X_i\vec{A})}{P(X_i)P(\vec{A})} - \sum_{X_i\vec{A}\vec{B}} P(X_i\vec{A}\vec{B}) \log \frac{P(X_i\vec{A}\vec{B})}{P(X_i)P(\vec{A}\vec{B})}
\end{aligned}$$

since $P(X_i\vec{A})$ is a component of $P(X_i\vec{A}\vec{B})$,

$$\begin{aligned}
= &\sum_{X_i\vec{A}\vec{B}} P(X_i\vec{A}\vec{B}) \log \frac{P(X_i\vec{A})P(\vec{A}\vec{B})}{P(\vec{A})P(X_i\vec{A}\vec{B})} \\
\leq &\sum_{X_i\vec{A}\vec{B}} P(X_i\vec{A}\vec{B}) (\frac{P(X_i\vec{A})P(\vec{A}\vec{B})}{P(\vec{A})P(X_i\vec{A}\vec{B})} - 1) \log(e)
\end{aligned}$$

since $\log(Y) \leq (Y-1)\log(e)$ for all $Y \geq 0$,

where e is the base of the natural logarithms.

$$\begin{aligned}
= &\sum_{X_i\vec{A}\vec{B}} (\frac{P(X_i\vec{A})P(\vec{A}\vec{B})}{P(\vec{A})} - P(X_i\vec{A}\vec{B})) \log(e) \\
= &\sum_{\vec{A}\vec{B}} \frac{\sum_{X_i} P(X_i\vec{A})P(\vec{A}\vec{B}) - \sum_{X_i} P(X_i\vec{A}\vec{B})P(\vec{A})}{P(\vec{A})} \log(e) \\
= &\sum_{\vec{A}\vec{B}} \frac{P(\vec{A})P(\vec{A}\vec{B}) - P(\vec{A}\vec{B})P(\vec{A})}{P(\vec{A})} \log(e) \\
= &\ 0
\end{aligned}$$

Therefore, the inequality holds. ■

References

Abramson, B. (1991). ARCO1: An application of belief networks to the oil market. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 1–8.

Beinlich, I. A., Suermondt, H. J., Chavez, R. M., and Cooper, G. F. (1989). The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proceedings of the 2nd European Conference on Artificial Intelligence in Medicine*, pages 247–256.

Chavez, R. (1990). *Architectures and Approximation Algorithms for Probabilistic Expert Systems.* PhD thesis, Medical Computer Science Group, Stanford University, Stanford, CA.

Chavez, R. and Cooper, G. F. (1990). A randomized approximation algorithm for probabilistic inference on Bayesian belief networks. *Networks*, 20:661–685.

Chow, C. K. and Liu, C. N. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467.

Cooper, G. (1984). *NESTOR: a computer-based medical diagnosis aid that integrates causal and probabilistic knowledge.* PhD thesis, Medical Computer Science Group, Stanford University, Stanford, CA.

Cooper, G. F. (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405.

Cooper, G. F. and Herskovits, E. (1991). A Bayesian method for constructing Bayesian belief networks from databases. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 86–94.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1989). *Introduction to Algorithms.* MIT-Press, Cambridge, Massachusetts.

Dagum, P. and Chavez, R. (1991). Approximating probabilistic inference in Bayesian belief networks. Technical report, Knowledge Systems Laboratory, Stanford University, Stanford, CA.

Dagum, P. and Luby, M. (1993). Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60:141–153.

Fung, R. and Chang, K. (1990). Weighing and integrating evidence for stochastic simulation in Bayesian networks. In (Henrion et al., 1990), pages 209–219.

Gao, Q. and Li, M. (1989). The minimum description length principle and its application to online learning of handprinted characters. In *Procceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 843–848.

Geiger, D., Paz, A., and Pearl, J. (1990). Learning causal trees from dependence information. In *Proceedings of the AAAI National Conference*, pages 770–776.

Geiger, D. and Pearl, J. (1990). On the logic of causal models. In Shachter, R., Levitt, T. S., Kanal, L. N., and Lemmer, J. F., editors, *Uncertainty in Artificial Intelligence 4*, pages 3–14. North-Holland, Amsterdam.

Henrion, M. (1987). Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In Kanal, L. N. and Lemmer, J. F., editors, *Uncertainty in Artificial Intelligence Vol II*, pages 149–163. North-Holland, Amsterdam.

Henrion, M. (1990). Towards efficient inference in multiply connected belief networks. In *Influence Diagrams, Belief Nets, and Decision Analysis*, pages 385–407. Wiley, Chichester, England.

Henrion, M. (1991). Search-based methods to bound diagnostic probabilities in very large belief nets. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 142–150.

Henrion, M., Shacter, R., Kanal, L. N., and Lemmer, J. F., editors (1990). *Uncertainty in Artificial Intelligence 5*. North-Holland, Amsterdam.

Jensen, F., Lauritzen, S., and Olesen, K. (1990). Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4:269–282.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, 22:76–86.

Lelewer, D. A. and Hirschberg, D. S. (1987). Data compression. *ACM Computing Surveys*, 19(3):261–296.

Pearl, J. (1986). Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29:241–288.

Pearl, J. (1987). Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence*, 32:245–257.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California.

Pearl, J. and Verma, T. S. (1991). A theory of inferred causation. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 441–452.

Peng, Y. and Reggia, J. (1987a). A probabilistic causal model for diagnostic problem solving - part 1: integrating symbolic causal inference with numeric probabilistic inference. *IEEE Trans. on Systems, Man and Cybern.*, 17(2):146–162.

Peng, Y. and Reggia, J. (1987b). A probabilistic causal model for diagnostic problem solving - part 2: diagnostic strategy. *IEEE Trans. on Systems, Man and Cybern.*, 17(3):395–406.

Rebane, G. and Pearl, J. (1987). The recovery of causal poly-trees from statistical data. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 222–228.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14:465–471.

Robinson, R. W. (1976). Counting unlabeled acyclic digraphs. In *Proceedings of the 5th Australian Conference on Combinatorial Mathematics*, pages 28–43.

Shachter, R. D. and Peot, M. A. (1990). Simulation approaches to general probabilistic inference on belief networks. In (Henrion et al., 1990), pages 221–231.

Spirtes, P. Glymour, C. and Scheines, R. (1990). Causality from probability. In *Evolving Knowledge in Natural Science and Artificial Intelligence*, pages 181–199.

Verma, T. and Pearl, J. (1990). Equivalence and synthesis of causal models. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 220–227.