Learning Belief Networks from Data: An Information Theory Based Approach

Jie Cheng, David A. Bell, Weiru Liu

School of Information and Software Engineering
University of Ulster at Jordanstown
United Kingdom, BT37 0QB
Email: {j.cheng, da.bell, w.liu}@ulst.ac.uk
Tel: 44 1232 366500   Fax: 44 1232 366068

Abstract

This paper presents an efficient algorithm for learning Bayesian belief networks from databases. The algorithm takes a database as input and constructs the belief network structure as output. The construction process is based on the computation of mutual information of attribute pairs. Given a data set that is large enough, this algorithm can generate a belief network very close to the underlying model, and at the same time, enjoys the time complexity of $O(N^4)$ on conditional independence (CI) tests. When the data set has a *normal DAG-Faithful* (see Section 3.2) probability distribution, the algorithm guarantees that the structure of a *perfect map* [Pearl, 1988] of the underlying dependency model is generated. To evaluate this algorithm, we present the experimental results on three versions of the well-known ALARM network database, which has 37 attributes and 10,000 records. The results show that this algorithm is accurate and efficient. The proof of correctness and the analysis of computational complexity are also presented.

1 Introduction

The Bayesian belief network is a powerful knowledge representation and reasoning tool under conditions of uncertainty. A Bayesian belief-network is a directed acyclic graph (DAG) with a conditional probability distribution for each node. The DAG structures of such networks contain nodes representing domain variables, and arcs between nodes representing probabilistic dependencies. On constructing Bayesian networks from databases, we use nodes to represent database attributes.

In recent years, many belief network construction algorithms have been developed. Generally, these algorithms can be grouped into two categories: one category of algorithms uses heuristic searching methods to construct a model and then evaluates it using a scoring method. This process continues until the score of the new model is not significantly better than the previous one. Different scoring criteria have been applied in these algorithms, such as, Bayesian scoring method [Cooper and Herskovits, 1992; Heckerman *et al*., 1994], entropy based method [Herskovits, 1991], and minimum description length method [Suzuki, 1996]. The other category of algorithms constructs Bayesian networks by analyzing dependency relationships among nodes. The dependency relationships are measured by using some kind of conditional independence (CI) test. The algorithms described in [Spirtes *et al*., 1991; Wermuth and Lauritzen, 1983; Srinivas *et al*., 1990] and the algorithm proposed in this paper belong to this category. Both of these two categories of algorithms have their advantage and disadvantage: Generally, the first category of algorithms has less time complexity in the worst case (when the underlying DAG is densely connected), but it may not find the best solution due to its heuristic nature; The second category of algorithms is usually asymptotically correct when the probability distribution of data satisfies certain assumption, but as Cooper *et al.* pointed out in [Cooper and Herskovits, 1992], CI tests with large condition-sets may be unreliable unless the volume of data is enormous.

On developing this algorithm, we take the following two facts into consideration. First of all, real world situations usually yield sparse networks, and densely connected networks reveal very few independence relationships and thus contain little valuable information. Therefore, the algorithm should be particularly efficient when the database has a sparse underlying network. Secondly, since CI tests with large condition-sets are computational expensive and may be unreliable, we try to avoid CI tests with large condition-sets and use as few CI tests as possible. Considering the above discussion, we developed a novel algorithm that can avoid many unnecessary CI tests with large condition-sets and also reduce the number of CI tests.

Generally, this algorithm has two major advantages over other CI test based algorithms. First of all, it constructs Bayesian networks through three phases, drafting, thickening and thinning. This is a natural extension of the well-known Chow and Liu's algorithm [Chow and Liu, 1968] to multi-connected network. In the case where the node ordering is known, this algorithm even preserves the merit of Chow and Liu's algorithm that only $O(N^2)$ times of CI tests are needed. Secondly, unlike other CI methods which use χ^2 tests [Spirtes *et al.*, 1991; Wermuth and Lauritzen, 1983] to check if two variables are dependent, we use mutual information as CI test which can tell us not only if two variables are dependent but also how close their relationship is. Therefore, our algorithm has the ability to compare the results of CI tests quantitatively. As showed later in this paper, this is the very reason that our algorithm requires CI tests only $O(N^4)$ times. (N is the number of nodes in the network.)

The remainder of this paper is organized as follows. In Section 2, we give the background information and introduce our information theory based algorithm. In Section 3, we present our algorithm in detail and give the proof of correctness and analysis of complexity. Section 4 contains the experimental results on three data sets of the alarm network. Finally, in Section 5, we discuss the related work and the important features of our work.

2 An Information Theory Based Approach

Our algorithm constructs belief networks by analyzing conditional independence relationships among nodes. To introduce our approach, we first review the concept of *d-separation* [Pearl, 1988], which plays an important role in our algorithm. For any three disjoint node sets *X*, *Y*, and *Z* in a belief network, *X* is said to be *d-separated* from *Y* by *Z* if there is no active *adjacency* path between *X* and *Y*. An *adjacency* path is a path between two nodes without considering the directionality of the arcs. A path between *X* and *Y* is active given *Z* if: (1) every *collider* [Spirtes *et al*., 1996] in the path is in *Z* or has a descendant in *Z*; (2) every other node in the path is outside *Z*. A *collider* of a path is a node where two arcs in the path meet at their endpoints. In a belief network, if there is an arc from *a* to *b*, we say that *a* is a parent of *b* and *b* is a child of *a*. We also say that *a* is in the neighborhood of *b* and *b* is in the neighborhood of *a*. To understand *d-Separation*, we can use an analogy, which is similar to the one suggested in [Spirtes *et al*., 1996]. We view a belief network as a network system of information channels, where each node is a valve that is either active or inactive and the valves are connected by noisy information channels. The information flow can pass an active valve but not an inactive one. When all the valves (nodes) on one *adjacency* path between two nodes are active, we say this path is **open**. If any one valve in the path is inactive, we say the path is **closed**. When all paths between two nodes are **closed** given the statuses of a set of valves (nodes), we say the two nodes are *d-separated* by the set of nodes. The statuses of valves can be changed through the instantiation of a set of nodes. The amount of information flow between two nodes can be measured by using mutual information when no nodes are instantiated, or conditional mutual information when some other nodes are instantiated.

In information theory, the mutual information of two nodes X_i, X_j is defined as

$$I(X_i, X_j) = \sum_{x_i, x_j} P(x_i, x_j) \log \frac{P(x_i, x_j)}{P(x_i)P(x_j)}; \qquad \textbf{(1)}$$

and the conditional mutual information is defined as

$$I(X_i, X_j | C) = \sum_{x_i, x_j, c} P(x_i, x_j, c) \log \frac{P(x_i, x_j | c)}{P(x_i | c), P(x_j | c)}, \qquad \textbf{(2)}$$

where X_i, X_j are two nodes and C is a set of nodes. In our algorithm, we use conditional mutual information as CI tests to measure the average information between two nodes when the statuses of some valves are changed by the condition-set *C*. When $I(X_i, X_j | C)$ is smaller than a certain threshold value ε, we say that X_i, X_j are *d-separated* by the condition-set *C*, and they are conditionally independent.

Using mutual information in probabilistic model construction can be traced back to Chow and Liu's tree construction algorithm [Chow and Liu, 1968]. In 1987, Rebane and Pearl extend Chow and Liu's algorithm to causal polytree construction [Pearl, 1988]. Our algorithm extends those algorithms further to Bayesian belief network construction.

This algorithm also makes the following two assumptions: (1) The database attributes have discrete values and there are no missing values in all the records. (2) The volume of data is large enough for reliable CI tests.

3 The Construction Algorithm

This algorithm has three phases: drafting, thickening and thinning. The first phase of our algorithm is essentially Chow and Liu's tree construction algorithm; the second and the third phase are designed to extend tree construction to general Bayesian belief network construction. In the first phase, this algorithm computes mutual information of each pair of nodes as a measure of closeness, and creates a draft based on this information. The draft is a singly connected graph (a graph without loops). In a special case when the Bayesian network is a tree or polytree, this phase can construct the network correctly and the second and third phase will not change anything. So, Chow and Liu's algorithm and Rebane and Pearl's algorithm can be viewed as special cases of our algorithm. In the second phase, the algorithm adds edges when the pairs of nodes cannot be *d-separated*. The result of Phase II has the structure of an *independence map* (*I-map*) [Pearl, 1988] of the underlying dependency model given the underlying model is *normal DAG-Faithful*. In the third phase, each edge of the *I-map* is examined using CI tests and will be removed if the two nodes of the edge can be *d-separated*. The result of Phase III has the structure of a *perfect map* [Pearl, 1988] when the underlying model is *normal DAG-Faithful*. At the end of the third phase, our algorithm also carries out a procedure to orient the edges of the graph.

3.1 The Algorithm

Phase I: (Drafting)

1. Initiate a graph $G(V, E)$ where *V*={all the nodes of a data set}, *E*={ }. Initiate an empty list *L*.
2. For each pair of nodes (v_i, v_j) where $v_i, v_j \in V$, compute mutual information $I(v_i, v_j)$ using equation (1). For all the pairs of nodes that have mutual information greater than a certain small value ε, sort them by their mutual information and put these pairs of nodes into list *L* from large to small. Create a pointer *p* that points to the first pair of nodes in *L*.
3. Get the first two pairs of nodes of list *L* and remove them from it. Add the corresponding edges to *E*. Move the pointer *p* to the next pair of nodes.
4. Get the pair of nodes from *L* at the position of the pointer *p*. If there is no *adjacency* path between the two nodes, add the corresponding edge to *E* and remove this pair of nodes from *L*.
5. Move the pointer *p* to the next pair of nodes and go back to step 4 unless *p* is pointing to the end of *L* or *G* contains *n*-1 edges. (*n* is the number of nodes in G.)

In order to illustrate this algorithm's working mechanism, we use a simple multi-connected network example borrowed from [Spirtes *et al*., 1996]. Suppose we have a database that has underlying Bayesian network as Figure 1.a, our task is to rediscover the underlying network structure from data. After step 2, we can get the mutual information of all 10 pair of nodes. Suppose we have $I(B,D) \geq I(C,E) \geq I(B,E) \geq I(A,B) \geq I(B,C) \geq I(C,D) \geq I(D,E) \geq I(A,D) \geq I(A,E) \geq I(A,C)$, and all the mutual information is greater than ε, we can construct a draft shown in Figure 1.b after step 5. Please note that the order of mutual information between nodes can not be arbitrary. For example, from information theory, we have $I(A,C) < Min(I(A,B),I(B,C))$. This is also the reason why Phase I can construct a graph close to the original graph to some extent. In fact, if the underlying graph is a singly connected graph, Phase I of this algorithm is essentially the algorithm of [Chow and Liu, 1968], and it guarantees the constructed network structure is the same as the original one. In this example, *(B,E)* is wrongly added and *(D,E)* and *(B,C)* are missing because of the existing *adjacency* paths *(D-B-E)* and *(B-E-C)*. The draft created in this phase is the base for next phase.

Phase II: (Thickening)

6. Move the pointer *p* to the first pair of node in *L*.
7. Get the pair of nodes from *L* at the position of the pointer *p*. Call procedure **try_to_separate_A (current graph, node1, node2)** to see if this pair of nodes can be separated in current graph. If so, go to next step; otherwise, connect the pair of nodes by adding a corresponding edge to *E*. (Procedure **try_to_separate_A** will be presented later in this subsection.)
8. Move the pointer *p* to the next pair of nodes and go back to step 7 unless *p* is pointing to the end of *L*.

In our example, the graph after Phase II is shown in Figure 1.c. Edge *(B,C)* and *(D,E)* are added because procedure **try_to_separate_A** cannot separate these pairs of nodes using CI tests. Edge *(A,C)* is not added because CI test can reveal that A and C are independent given block set {*B*}. Edge *(A,D)*, *(C,D)* and *(A,E)* are not added for the same reason.

In this phase, the algorithm examines all pairs of nodes that have mutual information greater than ε and are not directly connected. An edge is not added only when the two nodes are independent given certain block set. However, it is possible that some edges are wrongly added in this phase. The reasons are as follows. 1. Some real edges may be still missing until the end of this phase, and these missing edges can prevent procedure **try_to_separate_A** from finding the correct condition-set. 2. Because Procedure **try_to_separate_A** uses a heuristic method, it may not be able to find the correct condition-set for a special group of structures. (The detail is discussed later in this section.)

Phase III: (Thinning)

9. For each edge in *E*, if there are other paths besides this edge between the two nodes, remove this edge from *E* temporarily and call procedure **try_to_separate_A (current graph, node1, node2)**. If the two nodes are dependent, add this edge back to *E*; otherwise remove the edge permanently.
10. For each edge in *E*, if there are other paths besides this edge between the two nodes, remove this edge from *E* temporarily and call procedure **try_to_separate_B (current graph, node1, node2)**. If the two nodes are dependent, add this edge back to *E*; otherwise remove the edge permanently. (Procedure **try_to_separate_B** will be presented later in this subsection.)
11. Call procedure **orient_edges (current graph)**. (This procedure will be presented later in this subsection.)

The 'thinned' graph of our example is shown in Figure 1.d, which has the same structure of the original graph. Edge *(B,E)* is removed because *B* and *E* are independent given {*C,D*}. Given the underlying dependency model has a *normal DAG-faithful* probability distribution, the structure generated by this procedure is exactly the same as the structure of the underlying model. This phase can also orient edge *(C,E)* and edge *(D,E)* correctly.

Since procedure **try_to_separate_A** uses heuristic method to find the condition-set, it may not always be able to separate two *d-separated* nodes. In order to guarantee that a correct structure can always be generated, we have to use a correct procedure **try_to_separate_B** at step 10 to re-examine the current edges. Theoretically, we can use procedure **try_to_separate_B** to replace procedure **try_to_separate_A** in Phase II and remove step 9 in Phase III since they do the same thing and both of them have complexity $O(N^4)$ on CI test. But in practice, procedure **try_to_separate_A** usually uses fewer CI tests and requires smaller condition-sets. Therefore we try to avoid using procedure **try_to_separate_B** whenever it is possible.

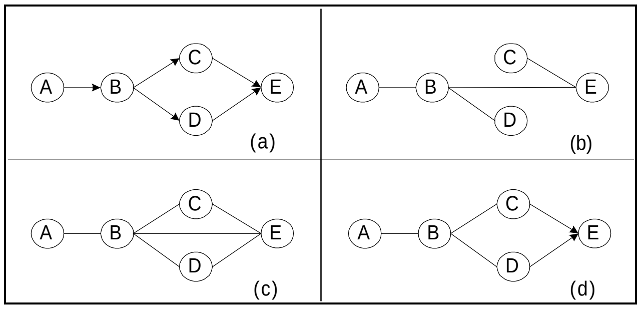

Figure 1. A simple multi-connected network.

Procedure try_to_separate_A (current graph, node1, node2)

1. Find the neighbors of *node1* and *node2* that are on the *adjacency* paths between *node1* and *node2*. Put them into two sets *N1* and *N2* respectively.
2. Remove the currently known child-nodes of *node1* from *N1* and child-nodes of *node2* from *N2*.
3. If the cardinality of *N1* is greater than that of *N2*, swap *N1* and *N2*.
4. Use *N1* as condition-set *C*.
5. Conduct a CI test using equation (2). Let $v = I(node1,node2|C)$. If $v < \varepsilon$, return ('d-separated').
6. If *C* contains only one node, go to step 8; otherwise, for each *i*, let $C_i = C \setminus \{the\ i^{th}\ node\ of\ C\}$, $v_i = I(node1,node2|\ C_i)$. Find the smallest value v_m of $v_1, v_2, \ldots$
7. If $v_m < \varepsilon$, return ('d-separated'); otherwise, if $v_m > v$ go to step 8 else let $v = v_m$, $C = C_m$, go to step 6.
8. If *N2* has not been used, use *N2* as condition-set *C* and go to step 5; otherwise, return ('failed').

From the definition of Bayesian belief network [Pearl, 1988] we know that if two nodes *a*, *b* in the network are not connected, they can be *d-separated* by the parent nodes of *b* which are in the paths between those two nodes. (We assume that node *a* appears earlier in the node ordering than *b*.) Those parent nodes form a set *P* which is a subset of *N1* or *N2* of the above procedure. If node ordering is known, we can get *P* immediately and only one CI test is required to check if two nodes are *d-separated*. Since this information is usually not given, we have to use a group of CI tests to find such *P*. By assuming that removing a parent node of *b* will not increase the mutual information between *a* and *b*, the above procedure try to find set *P* by identifying and removing the child-nodes and irrelevant nodes from *N1* and *N2* once a time using a group of computations and comparisons of conditional mutual information. However, this assumption may not be true when the underlying structure satisfies the following conditions: (1) There exists at least one path from *a* to *b* through a child-node of *b* and this child-node is a *collider* on the path. (2) In such paths, there are one or more *colliders* besides the child-node and all these *colliders* are the parents or ancestors of *b*. In such structures, procedure **try_to_separate_A** may identify a parent-node of *b* as a child-node of *b* and remove it erroneously. As a result, the procedure fails to separate two *d-separated* nodes. To deal with these structures, a correct procedure **try_to_separate_B** is introduced.

Procedure try_to_separate_B (current graph, node1, node2)

1. Find the neighbors of *node1* and *node2* that are on the *adjacency* paths between *node1* and *node2*. Put them into two sets *N1* and *N2* respectively.
2. Find the neighbors of the nodes in *N1* that are on the *adjacency* paths between *node1* and *node2*, and do not belong to *N1*. Put them into set *N1'*.
3. Find the neighbors of the nodes in *N2* that are on the *adjacency* paths between *node1* and *node2*, and do not belong to *N2*. Put them into set *N2'*.
4. If |*N1+N1'*| < |*N2+N2'*| let set *C=N1+N1'* else let *C=N2+N2'*.
5. Conduct a CI test using equation (2). Let v = *I(node1,node2|C)*. If $v < \varepsilon$, return ('d-separated').
6. Let *C'=C*. For each $i \in [1, |C|]$, let C_i =*C* \ {*the* i^{th} *node of C*}, v_i = *I(node1,node2|* C_i *)*. If $v_i < \varepsilon$ return ('d-separated') else if $v_i \leq v+e$ then *C'=C'*\{*the* i^{th} *node of C*}. (*e* is a small value.)
7. If |*C'*|<|*C*| then let *C=C'*, go to step 5; otherwise, return ('failed').

The major difference between procedure **try_to_separate_A** and **try_to_separate_B** is that in stead of blocking the nodes of *N1* or *N2* the latter procedure also blocks nodes of set *N1'* or *N2'*. Since blocking two consecutive nodes in a path can always close the path, blocking set *N1+N1'* or *N2+N2'* can close all the paths that connect *node1* and *node2* through two or more nodes. The only open paths are those connect *node1* and *node2* through one *collider*. Under this circumstance, we can remove all the *colliders* that connecting *node1* and *node2* without opening any previously closed paths. Thus, all paths between *node1* and *node2* in the underlying model can be closed. The correctness proof of this procedure is in section 3.2.

In both procedure **try_to_separate_A** and procedure **try_to_separate_B**, we have to compare the mutual information on different condition-sets. If we do not use quantitative CI tests, we can not compare the results of different CI tests and therefore can not remove irrelevant nodes. By reducing the cardinality of the condition-set after each iteration, we can avoid to test on every subset of the initial condition-set and thus avoid exponential number of CI tests. However, qualitative CI test based algorithms have to carry out the test on every subset of *C* in order to separate two nodes, so the number of CI tests in such algorithms must be exponential in the worst case.

Procedure orient_edges (current graph)

1. For any two nodes *s1* and *s2* that are not directly connected and there is at least one node that is the neighbor of both *s1* and *s2*, find the neighbors of *s1* and *s2* that are on the *adjacency* paths between *s1* and *s2*. Put them into two sets *N1* and *N2* respectively.
2. Find the neighbors of the nodes in *N1* that are on the *adjacency* paths between *s1* and *s2*, and do not belong to *N1*. Put them into set *N1'*.
3. Find the neighbors of the nodes in *N2* that are on the *adjacency* paths between *s1* and *s2*, and do not belong to *N2*. Put them into set *N2'*.
4. If |*N1+N1'*| < |*N2+N2'*| let set *C=N1+N1'* else let *C=N2+N2'*.
5. Conduct a CI test using equation (2). Let v = *I(node1,node2|C)*. If $v < \varepsilon$ go to step 8.
6. Let *C'=C*. For each $i \in [1, |C|]$, let C_i =*C* \ {*the* i^{th} *node of C*}, v_i = *I(s1,s2|* C_i *)*. If $v_i \leq v+e$ then *C'=C'*\{*the* i^{th} *node of C*}, let *s1* and *s2* be parents of the i^{th} node of *C*. If $v_i < \varepsilon$, go to step 8. (*e* is a small value.)
7. If |*C'*|<|*C*| then let *C=C'*, go to step 5.
8. Go back to step1 until all pairs of nodes are examined.
9. For any three nodes *a*, *b*, *c*, if *a* is a parent of *b*, *b* and *c* are adjacent, and *a* and *c* are not adjacent and edge *(b, c)* is not oriented, let *b* be a parent of *c*.
10. For any edge *(a, b)* that is not oriented, if there is a directed path from *a* to *b*, let *a* be a parent of *b*.
11. Go back to step 9 until no more edges can be oriented.

Among the nodes in Bayesian networks, only *colliders* can let information pass them when they are instantiated. We use this feature in step 6 to identify *colliders*. All other edge orientations are virtually based on these identified *colliders*. The idea behind this procedure is the same as that in procedure **try_to_separate_B**, i.e., by blocking all the paths that have length equal to or greater than three, we can always identify the *colliders* from *C* in a certain order. The correctness proof of this procedure is in section 3.2. The *collider* based edge orientation methods have also been studied in [Pearl, 1988; Spirtes *et al*., 1996].

3.2 Correctness

Before we present the correctness proof, we give the definition of *normal DAG-faithful*.

Definition 1 In a *DAG-faithful* [Spirtes *et al*., 1996] probability model, for any two nodes that are connected by at least two adjacency paths, under an arbitrary situation where some paths between the two nodes are **closed** by a condition-set, if we can only increase the mutual information by **opening** any previously **closed** paths between the two nodes without **closing** any previously **opened** paths, we say that this probability model is ***normal DAG-faithful***.

In real world situations, most *DAG-faithful* models are also *normal DAG-faithful*. We conjecture that the violations of *normal DAG-faithfulness* only happen when the probability distributions are 'near' the violations of *DAG-faithfulness*. In such situations, other algorithms may also have difficulties to generate the true underlying model.

Given a data set that has a *normal DAG-faithful* underlying model and satisfies the two assumptions of Section 2, our algorithm presented in previous subsection can always generate the exact structure of the underlying dependency model *M*. We prove the correctness of our algorithm by the following propositions.

Proposition 1 G*raph G2 generated after Phase II has the structure of an I-map of M*.

Proof: Phase I and Phase II of our algorithm examined all the edges between any two nodes that are not independent. An edge is not added only if these two nodes are *d-separated* by a set of other nodes. Hence, any pair of not connected nodes of *G2* are conditional independent in *M*.   Q.E.D.

Lemma 1 *In procedure* ***try_to_separate_B*** *of PhaseIII, by using the initial condition-set N1+N1' or N2+N2', we can close all the paths of the underlying model M between node1 and node2 except the paths connecting node1 and node2 by one collider.*

Proof: By using *N1+N1'* or *N2+N2'* as the condition-set, we instantiate the nodes in *N1* or *N2* (the neighbors of *node1* or *node2* on the paths between *node1* and *node2*) and *N1'* or *N2'* (the neighbors of nodes of *N1 or N2* that are on the paths between *node1* and *node2*), therefore, at least two consecutive nodes of any path that has length equal to or larger than three are instantiated. Because instantiating two consecutive nodes of a path can close the path and all the paths of the underlying model *M* are in the current graph, we can close all the paths in *M* between *node1* and *node2* that have length equal to or larger than three. The only open paths are those connecting *node1* and *node2* by one *collider*.   Q.E.D.

Lemma 2 *Procedure* ***try_to_separate_B*** *does not open any previously closed path by removing a node from condition-set C.*

Proof: To prove this lemma, it is sufficient to prove that removing a node from *C* cannot open some paths while close other paths at the same time, since if removing a node can only open some paths, by the assumption of *normal DAG-faithful*, it must increase the mutual information and the procedure will not remove such node. Therefore, the paths cannot be opened. Now, we will prove that removing a node from *C* cannot open and close paths simultaneously. From lemma 1 we know that initially the only open paths are those connecting *node1* and *node2* by one *collider*. For a node in *C* that is not a child-node of both *node1* and *node2* or a descendent of such child-node, removing it may open some paths but cannot close the paths connecting *node1* and *node2* by a *collider*. For a node *v* in *C* that is a child-node of both *node1* and *node2* or a descendent of such child-node, if one of the descendents of *v* are in *C*, then removing *v* cannot close the path connecting *node1* and *node2* by the child-node. If none of its descendents are in *C*, removing *v* may close the path connecting *node1* and *node2* by the child-node but cannot open a path because the would-be opened path must go through a *collider* that is a descendent of *v*. Since none of the descendents of *v* is in *C*, such path cannot be opened.   Q.E.D.

Lemma 3 *Procedure* ***try_to_separate_B*** *can remove all the descendents of both node1 and node2 from C.*

Proof: Suppose *S* is a subset of set *C* and the all the nodes in *S* are the descendents of both *node1* and *node2* and cannot be removed, then there must exist a node $v \in S$ and *v* is not a ancestor of any other nodes in *S*. Since *v* cannot be remove, removing it must increase mutual information. From the assumption of *normal DAG-faithful*, we know that removing *v* will open at least one path. Therefore, node *v* is not a *collider* in such path and such path must go through at least one descendent of *v*. We also know that there must exist at least one descendent of *v* which is a *collider* in such path. To make such path open, this *collider* has to be in *S*. This contradicts our assumption that *v* is not a ancestor of any other nodes in *S*. Q.E.D.

Proposition 2 *Given that graph G has the structure of an I-map of the underlying model M, if two nodes a and b are independent in M, procedure* ***try_to_separate_B*** *can always d-separate them in G.*

Proof: From lemma 1 we know that initially the only open paths are those connecting node *a* and *b* by one *collider*. From lemma 2 and lemma 3 we know that the procedure does not open any path when removing nodes from *C* and all the descendents of both *a* and *b* in *C* are removed. Therefore, if node *a* and node *b* are independent in *M*, procedure **try_to_separate_B** can *d-separate* them by closing all the open paths.   Q.E.D.

Proposition 3 *Graph G3 generated after Phase III has the structure of a perfect map of M*.

Proof: Since *G2* has the structure of an *I-map* and an edge is removed in Phase III only if the pair of nodes are conditional independent in *M*, *G3* also has the structure of an *I-map* of *M*. From Proposition 2, we also know that if two nodes are independent in *M*, our algorithm can always *d-separate* them in *G3*. Hence, *G3* has the structure of a *perfect map* of *M*.   Q.E.D.

Proposition 4 *Given that graph G has the structure of an perfect map of the underlying model M, all the colliders that can be identified by procedure* ***orient_edges (G)*** *are the real colliders of M.*

Proof: For any structure *s1-a-s2* and *s1* and *s2* are not directly connected, procedure **orient_edges (*G*)** uses step 1 to 7 to check if *a* is a *collider* on the path *s1-a-s2*. Because step 1 to 7 of this procedure is virtually the same as procedure **try_to_separate_B**, from proposition 2 we know that it can identify a *collider* correctly. Since there are no pseudo-edges in *G*, step 6 of this procedure can never orient an edge wrongly. It is also easy to see that the inference of step 9 - 11 of procedure is correct.   Q.E.D.

When the underlying probability distribution is *DAG-faithful* but not *normal DAG-faithful*, this algorithm may also be able to generate the correct graph. In fact, this algorithm may not be able to separate two *d-separated* nodes only when there is at least one path that connect the two nodes by a single *collider* and removing a node in the condition-set causes violation of the *normal DAG-faithful* assumption. However, since this wrongly added edge does not change the *I-mapness* of the graph, the correctness of other edges in the graph will not be affected.

In order to get the correct graph for a *DAG-faithful* but not a *normal DAG-faithful* model, we can always replace procedure **try_to_separate_B(current graph, node1, node2)** with a procedure that try to separate the two nodes by using every subset of the neighbors of *node1* and then the subset of the neighbors of *node2*. By the definition of *DAG-Faithfulness*, this procedure is correct. Theoretically, this procedure requires exponential number of CI test in the worst case. However, for a sparse network, it only influences the performance slightly since this procedure is only used at the end of the third phase. From the experimental results shown in section 4 we can see that most running time is consumed in Phase I and Phase II.

3.3 Complexity Analysis

Suppose a data set has N attributes, the maximum number of possible values of any attribute is r, by equation (1), each computation of mutual information requires $O(r^2)$ times of basic operations such as logarithm, multiplication and division. By equation (2), each computation of conditional mutual information (CI test) in Phase II and III requires at most $O(r^N)$ basic operations in the worst case, when each CI test has a condition-set on all the other nodes. Therefore, this algorithm requires exponential number of basic operations in the worst case. However, the actual complexity is much less than that in real applications which usually have sparse networks. Since the number of CI tests required is a wildly accepted index for the efficiency of CI test based algorithms, we analyze the complexity of our algorithm as follows.

Phase I: Phase I computes mutual information between any two nodes; it needs N^2 mutual information computations. (A mutual information computation can be viewed as a CI test with an empty condition-set.) Therefore, this phase requires at most $O(N^2)$ times of CI tests.

Phase II: This phase tries to add each edge to the graph and call procedure **try_to_separate_A** at most N^2 times. An execution of procedure **try_to_separate_A** requires at most $2\sum_{i=1}^{N-2} i$ = (N-2)(N-1) times of CI tests. Therefore, Phase II requires at most $O(N^4)$ times of CI tests.

Phase III: This phase tries to remove each edge from the graph by using procedure **try_to_separate_A** and procedure **try_to_separate_B** at most N^2 times. An execution of procedure **try_to_separate_A** requires at most (N-2)(N-1) times of CI test; and an execution of procedure **try_to_separate_B** requires CI test at most *(N-2)(N-1)/2* times. Procedure **orient_edges** tries to find *colliders* by checking every pair of nodes using a mechanism similar to procedure **try_to_separate_B**, so it also requires CI test $O(N^4)$ times. Therefore, Phase III requires at most $O(N^4)$ times of CI tests.

Overall, this algorithm requires CI test at most $O(N^4)$ times.

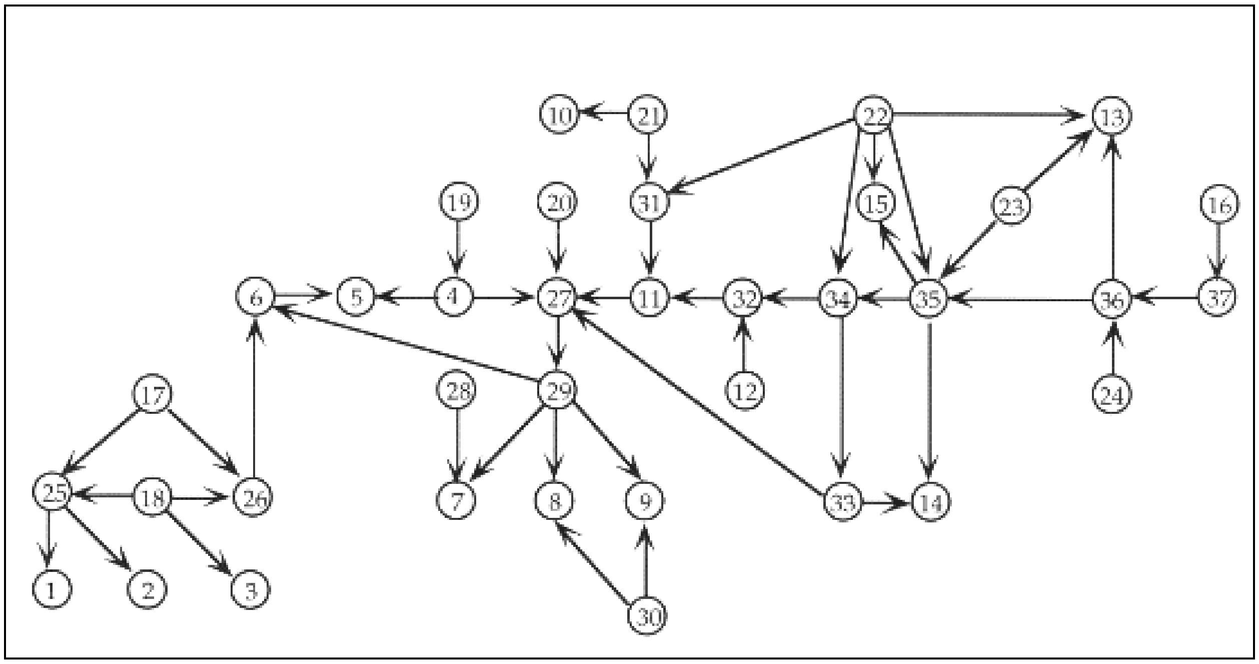

Figure 2. The ALARM belief network

Phase	Edges	Missing Edges	Extra Edges	No. of CI Tests[i]						Time
				0	1	2	3	4-7	Total	(Minute)
I	**36**	**13**	**3**	**666**	**48**	**0**	**0**	**0**	**714**	**22.0**
II	**51**	**1**	**6**	**0**	**128**	**81**	**25**	**7**	**241**	**11.0**
III	**46**	**1** {33-27}	**1** {32-27}	**0**	**27**	**43**	**31**	**12**	**113**	**5.0**

Table 1. Networks constructed at each phase for dataset1 of ALARM database.

Data sets	Results				No. of CI Tests						Time (Minute)			
	Missing Edges	Extra Edges	M.O.	W.O.	0	1	2	3	4+	Total	Phase I	Phase II	Phase III	Total
Dataset1	**1**	**1**	**4**	**0**	**666**	**203**	**124**	**56**	**19**	**1068**	**22.0**	**11.0**	**5.0**	**38.0**
Dataset2	**1**	**0**	**4**	**1**	**666**	**209**	**88**	**55**	**15**	**1033**	**22.0**	**8.5**	**4.7**	**35.2**
Dataset3	**3 (1)**	**0**	**5**	**0**	**666**	**181**	**84**	**43**	**12**	**986**	**22.0**	**5.7**	**4.6**	**32.3**

Table 2. Results on dataset1, dataset2 and dataset3.
(M.O. and W.O. in the table stand for 'missing orientation' and 'wrongly oriented'.)

4 Results on ALARM Network

ALARM network[Beinlich *et al.*, 1989] is a medical diagnostic alarm message system for patient monitoring, it contains 37 nodes and 46 arcs (see Figure 2). This belief network has become the *de facto* benchmark for evaluating algorithms on belief network construction. Researchers in this field use data sets generated from three versions of this belief network. To make a fair comparison to other algorithms, we test our algorithm on three data sets generated from each of the three versions of ALARM network which have the same structure and the different probability distributions. We call them dataset1, dataset2 and dataset3[ii] . Each of them has 10,000 cases. The experimental results show that our algorithm constructs the ALARM networks with fewer errors than other algorithms on all three data sets.

We summarize our experimental results into the following two tables. Table 1 shows the detailed testing results of each phase on dataset1, and table 2 compares the results of the three data sets. All these experiments were conducted on a Pentium 90MHz PC and the data sets are stored in a Microsoft Access database. To make CI tests more reliable when the volume of data is not large enough, we also modified equation (2) by taking the variable's *degree of freedom* into consideration. We use 0.003 as the value of ε.

From table 1, we can analyze the result of each phase on dataset1. In Phase I, the mutual information of all 666 ($= 37 \times (37-1)/2$) pairs of nodes were computed; and our algorithm uses this information to construct a draft. In Phase II, this algorithm conducted 241 CI tests and added 15 edges, among which, 12 of 13 real edges missed in the first phase were added. In Phase III, it removed 5 of 6 wrongly added edges. This algorithm also oriented 42 of 46 edges correctly. The four un-oriented edges are 18-3, 19-4, 21-10, 16-37. This is due to the limitation of *collider* based orientation method. In table 2, we compare the results on dataset1, dataset2 and dataset3. The difference in the running time is due to the difference in the obviousness of dependency relationships of the underlying probability distributions. On dataset1, our algorithm cannot find edge 33-27 because the value of $I(33,27|\{14,34\})$ is abnormally small. This missing edge causes that 32-27 be wrongly added. On dataset2 and dataset3, edge 11-27 is missing. This is because that this pair of nodes has very weak relationship when node 32 or 34 is instantiated. It may suggest that the relationship between this pair of nodes can be expressed through other dependency relationships. The result on dataset3 also missed two other edges; however, this is due to the fact that 12-32 and 21-31 are actually independent in dataset3 [Cheng *et al.*, 1997]. The independence between 21 and 31 also causes an extra un-oriented edge 22-31. From these tables, we can see that our results are very encouraging. The algorithm has also been successfully implemented in several real world applications. One of them is a telecommunication fault diagnosis system which has 34 attributes.

From table 1 and table 2, we can see that this algorithm is efficient and reliable. In the case of sparse networks like ALARM, the number of CI tests used is much fewer than N^4 times. In our experiments, we found that most running time is consumed by database engine on query preparation and data retrieving. The actual computations consume only about 5% of run-time.

5 Discussion

Mutual information has been used in probabilistic model construction since 1968 [Chow and Liu, 1968; Pearl, 1988]. However, their works are limited to tree and polytree construction. We apply the idea to Bayesian network construction. Like our Bayesian network construction algorithm, algorithms presented in [Spirtes *et al.*, 1991] and [Singh and Valtorta, 1995] do not require node ordering. Other algorithms [Cooper and Herskovits, 1992; Herskovits, 1991; Suzuki, 1996; Wermuth and Lauritzen, 1983; Srinivas *et al.*, 1990] deal with a rather special case where node ordering is known. In this special case, our simplified algorithm [Cheng *et al.*, 1997] require $O(N^2)$ times of CI tests and is correct when the underlying model is *DAG-faithful*. On the three data sets of ALARM network used in this paper, the simplified algorithm in this special case runs 25% faster and constructs the network with fewer errors than that in the general algorithm presented in this paper, because more information is given.

Algorithms described in [Spirtes *et al.*, 1991; Wermuth and Lauritzen, 1983; Srinivas *et al*., 1990] can also construct a belief network whose structure is a *perfect map* of the underlying dependency model when the underlying model is *DAG-faithful*. However, as we discussed in section 3.1, these χ^2 CI test based algorithms must require exponential number of CI tests. Comparing to these algorithms, our algorithms have the following two features. (1) By generating a draft in Phase I, our algorithms prevent many pseudo-edges from being added in Phase II, and therefore avoid high order CI tests in Phase II and III, and also reduce the number of CI tests needed in Phase III. (2) By using mutual information and conditional mutual information as a measure of dependency relationship, our algorithms can compare two relationships quantitatively, and therefore avoid exponential complexity.

Our subsequent research will focus on handling continuous variable nodes and missing values. We also plan to develop a commercial software based on our algorithms.

Acknowledgments

We wish to thank T. S. Richardson for his valuable comments that helped us to improve our work greatly. We would like to thank G. Cooper and E. Herskovits for providing their ALARM network data and D. Heckerman and Norsys Software Corp. for providing their ALARM network probability distributions. We would also like to thank C. Glymour, P. Spirtes and R. Scheines for their encouragement and comments on our work.

References

[Beinlich *et al*., 1989] Beinlich, I.A., Suermondt, H.J., Chavez, R.M. and Cooper, G.F., The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. *Proceedings of the Second European Conference on Artificial Intelligence in Medicine* (pp.247-256), London, England, 1989.

[Buntine, 1996] Buntine, W., A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on Knowledge and Data Engineering*, 8(2), 195-210, 1996.

[Cheng et al., 1997] Cheng, J., Bell, D.A. and Liu, W., An algorithm for Bayesian belief network construction from data, *Proceedings of AI & STAT'97* (pp.83-90), Ft. Lauderdale, Florida, 1997.

[Chow and Liu, 1968] Chow, C.K. and Liu, C.N., Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14, 462-467, 1968.

[Cooper and Herskovits, 1992] Cooper, G.F., Herskovits, E., A Bayesian Method for the induction of probabilistic networks from data, *Machine Learning*, 9, 309-347, 1992.

[Heckerman *et al*., 1994] Heckerman, D., Geiger, D. and Chickering, D.M., Learning Bayesian networks: the combination of knowledge and statistical data, *Technical Report MSR-TR-94-09*, Microsoft Research, 1994.[Pearl, 1988] Pearl, J. , *Probabilistic reasoning in intelligent systems: networks of plausible inference*, Morgan Kaufmann, 1988.

[Herskovits, 1991] Herskovits, E.H., Computer-based probabilistic network construction, *Doctoral dissertation*, Medical information sciences, Stanford University, Stanford, CA, 1991.

[Madigan *et al*., 1994] Madigan, D., Mosurski, K. and Almond R.G., Explanation in belief networks. *Technical Report*, Department of Statistics, University of Washington, 1994.

[Neapolitan, 1990] Neapolitan, R.E. , *Probabilistic reasoning in expert systems: theory and algorithms*, John Wiley & Sons, 1990.

[Singh and Valtorta, 1995] Singh, M. and Valtorta, M. Construction of Bayesian network structures from data: a brief survey and an efficient algorithm, *International Journal of Approximate Reasoning*, 12, 111-131, 1995.

[Spirtes *et al.*, 1996] Spirtes, P., Glymour, C. and Scheines, R., *Causation, Prediction, and Search* (Book), http://hss.cmu.edu/html/departments/philosophy/TETRAD.BOOK/book.html,1996.

[Spirtes *et al.*, 1991] Spirtes, P., Glymour, C. and Scheines, R., An algorithm for fast recovery of sparse causal graphs, *Social Science Computer Review*, 9, 62-72, 1991.

[Srinivas *et al*., 1990] Srinivas, S. Russell, S. and Agogino, A., Automated construction of sparse Bayesian networks from unstructured probabilistic models and domain information, In Henrion, M., Shachter, R.D., Kanal, L.N. and Lemmer, J.F. (Eds.), *Uncertainty in artificial intelligence 5*, Amsterdam: North-Holland, 1990.

[Suzuki, 1996] Suzuki, J., Learning Bayesian belief networks based on the MDL principle: An efficient algorithm using the branch and bound technique, *Proceedings of the international conference on machine learning*, Bally, Italy, 1996.

[Wermuth and Lauritzen, 1983] Wermuth, N. and Lauritzen, S., Graphical and recursive models for contingency tables. *Biometrika*, 72, 537-552, 1983.

[i] The CI tests are grouped by the cardinalities of condition-sets.

[ii] Dataset1 has the underlying belief network described in the web page of Norsys Software Corp. Dataset2 has the underlying belief network used by David Heckerman. Dataset3 is generated by Gregory F. Cooper and Edward Herskovits. We generated dataset1 and dataset2 using a Monte Carlo technique.