Learning Bayesian Networks from Data: An Information-Theory Based Approach

Jie Cheng[1] , Russell Greiner and Jonathan Kelly

Department of Computing Science, University of Alberta

David Bell and Weiru Liu

Faculty of Informatics, University of Ulster

November 1, 2001

Abstract

This paper provides algorithms that use an information-theoretic analysis to learn Bayesian network structures from data. Based on our three-phase learning framework, we develop efficient algorithms that can effectively learn Bayesian networks, requiring only polynomial numbers of conditional independence (CI) tests in typical cases. We provide precise conditions that specify when these algorithms are guaranteed to be correct as well as empirical evidence (from real world applications and simulation tests) that demonstrates that these systems work efficiently and reliably in practice.

Keywords

Bayesian belief nets, learning, probabilistic model, knowledge discovery, data mining, conditional independence test, monotone DAG faithful, information theory

---

[1] Now at Global Analytics, Canadian Imperial Bank of Commerce, Toronto, Canada.

1 Introduction

Bayesian networks (BNs; defined below) are a powerful formalism for representing and reasoning under conditions of uncertainty. Their success has led to a recent furry of algorithms for *learning* Bayesian networks from data. Although many of these learners produce good results on some benchmark data sets, there are still several problems:

- *Node ordering requirement*. Many BN-learning algorithms require additional information – notably an ordering of the nodes to reduce the search space (see [Cooper and Herskovits, 1992; Heckerman *et al.*, 1994]). Unfortunately, this information is not always available. We therefore want a learner that can exploit such prior information if it is available, but which can still learn effectively if it is not.
- *Computational Complexity*. Practically all BN learners are slow, both in theory [Chickering *et al.*, 1994] and in practice – e.g., most dependency-analysis based algorithms (defined below) require an exponential numbers of "conditional independence" tests.
- *Lack of publicly available learning tools.* Although there are many algorithms for this learning task, very few systems for learning Bayesian networks systems are publicly available. Even fewer can be applied to real-world data-mining applications where the data sets often have hundreds of variables and millions of records.

This motivates us to develop more effective algorithms for learning Bayesian networks from training data. Using ideas from information theory, we developed a three-phase dependency analysis algorithm, *TPDA*.[2] This *TPDA* algorithm is correct (i.e., will produce the perfect model of the distribution) given a sufficient quantity of training data whenever the underlying model is *monotone DAG faithful* (see below). Moreover, it requires at most $O(N^4)$ CI tests to learn an N-variable BN. In the special case where a correct node ordering is given, we developed a related algorithm, *TPDA-Π,* that requires $O(N^2)$ CI tests and is correct whenever the underlying model is *DAG-faithful*. These algorithms employ various other heuristics that enable it to work well in practice, including the use of the Chow-Liu algorithm to produce an initial structure; see Section

[2] "TPDA" stands for Three-Phase Dependency Analysis and the suffix "-Π"indicates that this algorithm expects an ordering of the nodes. We will later use "SLA" for Simple Learning Algorithm.

7.1. A more general *TPDA** algorithm, which is correct under DAG-faithfulness assumption, is also briefly introduced. We show that the efficiency of *TPDA** is actually very similar to *TPDA* in most real world applications.

Both *TPDA* and *TPDA-Π* algorithms have been incorporated into the *Bayesian Network PowerConstructor* system, which has been freely available on the Internet since October 1997 and has already been downloaded over 2000 times, with very positive feedback from the user community. To make these algorithms more useful to practitioners, both algorithms can exploit other types of prior knowledge that human experts may supply – such as the claim that there must (or must not) be an arc between two nodes [Darken, 1999]. The user can also specify various special cases for the structures – e.g., that it must be a tree-augmented Naïve Bayesian net (TAN) structure, or a Bayesian network augmented Naïve Bayesian net (BAN) structure [Friedman *et al.*, 1997; Cheng and Greiner, 1999]. We have also used this system to win the KDD Cup 2001 datamining competition (task one) – the Bayesian network model we learned gives the best prediction accuracy among 114 submissions on a very challenging biological data set [Page and Hatzis, 2001].

The remainder of the paper is organized as follows. Section 2 introduces Bayesian network learning from an information theoretic perspective. Its Table 1 provides a succinct summary of the terms that will be used. The subsequent two sections present simplified versions of the algorithms, to help illustrate the basic ideas. Section 3 presents the simple *SLA-Π* learning algorithm, which applies when a correct node ordering is available. That section also describes when *SLA-Π* is provably correct and analyses its complexity. Section 4 presents a more general algorithm, *SLA*, to handle the situation when the user does *not* provide a node ordering. It also proves the correctness of this algorithm, and analyses its complexity. Section 5 presents our actual algorithms – called *TPDA* for the general case, and *TPDA-Π* for the algorithm that takes a node ordering as input – which incorporate several heuristics to be more efficient. Section 6 presents and analyses the experimental results of both algorithms on real-world data sets. In addition to showing that our algorithms work effectively, we also show that the heuristics incorporated within *TPDA* make the system more efficient. Section 7 relates our learning algorithms to other Bayesian network learning algorithms, and Section 8 lists our contributions and proposes some future research directions. The appendices provide proofs of the theorems, discuss our "monotone DAG-faithful" assumption, and quickly introduce our general Bayesian network learning system, called the *BN PowerConstructor*.

As a final comment, please note that our complexity results (eg, $O(N^2)$ or $O(N^4)$) refer to the number of CI tests required. These results say nothing about the *order* of each such test, and so do not necessarily bound the computational complexity of the algorithm. In practice, however, these quantities are very informative, as they indicate the number of times the algorithm must sweep through the dataset, and we have found that such sweeps are in fact the major cost of these algorithms; see Section 6.

2 Learning Bayesian Networks Using Information Theory

A Bayesian network is represented by $BN = \langle N, A, \Theta\rangle$, where $\langle N, A\rangle$ is a directed acyclic graph (DAG) – each node $n \in N$ represents a domain variable (corresponding perhaps to a database attribute), and each arc $a \in A$ between nodes represents a probabilistic dependency between the associated nodes. Associated with each node $n_i \in N$ is a conditional probability distribution (*CPtable*), collectively represented by $\Theta=\{\theta_i\}$, which quantifies how much a node depends on its parents (see [Pearl, 1988]).

Learning a Bayesian network from data involves two subtasks: Learning the *structure* of the network (i.e., determining what depends on what) and learning the *parameters* (i.e., the strength of these dependencies, as encoded by the entries in the CPtables). As it is trivial to learn the parameters for a given structure from a complete data set (the observed frequencies are optimal with respect to the maximum likelihood estimation [Cooper and Herskovits, 1992]), this paper therefore focuses on the task of learning the structure.

We view the BN structure as encoding a group of conditional independence relationships among the nodes, according to the concept of *d-separation* (defined below). This suggests learning the BN structure by identifying the conditional independence relationships among the nodes. Using some statistical tests (such as chi-squared or mutual information), we can find the conditional independence relationships among the nodes and use these relationships as constraints to construct a BN. These algorithms are referred as *dependency analysis* based algorithms or *constraint*-based algorithms [Spirtes *et al.* 1993; Cheng *et al.* 1997a]. Section 7 compares this approach with the other standard approach, based on maximizing some score.

2.1 Overview of the Learning Algorithms (*SLA, TPDA*)

Our goal is to find "what is connected to what" – that is, which nodes should be joined by arcs. As explained below, our algorithms each work incrementally: at each point, it has a current set of arcs, and is considering adding some new arc, or perhaps deleting an existing one. Such decisions are based on "information flow" between a pair of nodes, relative to the rest of the current network. To elaborate this "flow" metaphor:

We can view a Bayesian network as a network of information channels or pipelines, where each node is a ***valve*** *that is either* ***active*** *or* ***inactive*** *and the valves are connected by noisy* ***information channels*** *(arcs). Information can flow through an active valve but not an inactive one. Now suppose two nodes – say X and Y – are* not *directly connected within the current network structure. If this structure is correct, then there should be no information flow between these nodes after closing all of the existing indirect connections between X and Y. Our learning algorithms will therefore try to close off all of these connections, then ask if the dataset exhibits additional information flow between these nodes. If so, the learner will realize the current structure is not correct, and so will add a new arc (think "pipeline") between X and Y.*

To be more precise, a path between nodes X and Y is closed, given some evidence C, if X and Y are conditionally independent given C. Graphically, this is defined by the concept called **direction dependent separation** or **d-separation** [Pearl, 1988]. Based on this concept, all the valid conditional independence relations in a DAG-Faithful distribution can also be directly derived from the topology of the corresponding Bayesian network.

That is,

Definition 1: *Adjacency path, d-separation, collider, cut-set, d-connected*: For any two nodes $X,Y \in V$, an "adjacency path" $P = \langle a_1, a_2, ..., a_k \rangle$ between $a_1=X$ and $a_k=Y$ is a sequence of arcs that, if viewed as undirected edges, would connect X and Y.

For a DAG $G = (N,A)$, for any nodes $X,Y \in N$ where $X \neq Y$, and "evidence" $C \subseteq N \backslash \{X,Y\}$, we say that "$X$ and Y are **d-separated** given C in G" if and only if there exists no *open adjacency path* between X and Y, where any such adjacency path P is considered **open** iff

(i) every collider on P is in C or has a descendent in C and

(ii) no other nodes on path P is in C.

where a node v is a *collider* of the path $\langle a_1, \ldots a_{i-1}=(X, v), a_i=(Y,v), \ldots, a_k \rangle$ if the two directed arcs associated with that node, here $a_{i-1}=(X, v)$ and $a_i=(Y,v)$, 'collide' at v.[3]

Here we call this set C a **cut-set**. If X and Y are not d-separated given C we say that X and Y are **d-connected** given C. []

In Figure 1, C-E-D is an adjacency path connecting C and D, even though the arcs are in different directions; we also say that E is a collider in the path C-E-D. Given empty evidence (ie, the empty cut-set {}), C and D are d-separated.

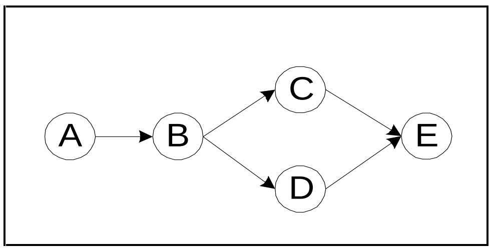

Figure 1 A simple multi-connected

In our analog, putting a node into the cut-set is equivalent to altering the status of the corresponding valves – hence, putting the collider E into the cut-set will open the path between C and D; while putting the non-collider B into the cut-set will close both the A-B-C-E and the A-B-D-E paths, thereby d-separating A and E.

Hence, to decide whether to add a new arc between nodes A and E, with respect to this graph, our learning algorithms (eg, *TPDA*) will try to block the information flow from every other indirect set of pipelines, by making inactive at least one valve in each path. Here, this can be accomplished by

[3] The other nodes are called *non-colliders* of the path. Note that the concept of collider is always related to a particular path, as a node can be a collider in one path and a non-collider in another path.

adding *B* to the cut-set. As noted above, if there is residual information flow between these nodes given this cut-set, TPDA will then add an arc directly connecting *A*-*E*.

In general, we measure the volume of information flow between two variables (read "nodes") *A* and *B* using *mutual information*

$$I(A\,,B) \;=\; \sum_{a,b} P(a,b)\,\log\frac{P(a,b)}{P(a)\,P(b)}\,, \tag{2.1}$$

and the *conditional mutual information*, with respect to the set of "evidence" variables (condition-set) ***C***,

$$I(A,B\,|\,C) \;=\; \sum_{a,b,c} P(a,b,c)\,\log\frac{P(a,b\,|\,c)}{P(a\,|\,c)\,P(b\,|\,c)}\,. \tag{2.2}$$

The mutual information between variables *A* and *B* measures the expected information gained about *B*, after observing the value of the variable *A*. In Bayesian networks, if two nodes are dependent, knowing the value of one node will give us some information about the value of the other node. Hence, the mutual information between two nodes can tell us if the two nodes are dependent and if so, how close their relationship is.

Given the actual probability distribution $P(x)$, we would claim that *A* and *B* are independent iff $I(A,B)=0$. Unfortunately, our learning algorithms do not have access to the true distribution $P(x)$, but instead use empirical estimates $\hat{P}_D(x)$, based on the dataset *D*. We therefore use $I_D(A,B)$, which approximates $I(A,B)$ but uses $\hat{P}_D(x)$ rather than $P(x)$. Our algorithms will therefore claim that *A* is independent of *B* whenever $I_D(A,B) < \varepsilon$, for some suitably small threshold $\varepsilon > 0$. We will similarly define $I_D(A,B|C)$, and declare conditional independence whenever $I_D(A,B|C) < \varepsilon$.

Note the computational cost of computing $I_D(A,B|C)$ is exponential in the size of *C* – ie, requires time proportional to the product of the sizes of the domains of *A*, *B*, and of all of the nodes of *C*. It is also linear in the size of the dataset *D*, as our learner will sweep through the entire dataset to gather the relevant statistics (to determine, in parallel, all necessary $\hat{P}_D(x)$ values).

Finally, we will later use the following two definitions:

Definition 2: *Ngbr, AdjPath*: For any graph $G = (V,A)$, let

$$Ngbr_G(a) = Ngbr(a) = \{ v \in V \mid (v,a) \in A \text{ or } (a,v) \in A \}$$

be the set of nodes connecting node $a \in V$ by an edge. Also, for any pair of nodes $A,B \in V$, let

$$AdjPath(A,B) = AdjPath_G(A,B)$$

be the set of *nodes* in the adjacency paths connecting A to B within G.

$\text{AdjPath}_G(\text{A,B})$	Nodes appears on *adjacency path* connecting nodes A to B within graph G, See Definition 2.
Adjacency path	An open path, ignoring the directions of the edges. See Definition 2
Arc	Directed edge (within a graph)
Bayesian network	A kind of graphical model of a joint distribution. See Section 2.
CI test	*Conditional independence* test --- Equations 2.1 and 2.2.
Collider	A node on a path where two arcs “collide”. The node “C” in A→C←B. See Section 2
Condition-set	The “C” in “P(A|C)”. See Equation 2.2
Cut-set	In graph G, if a set of nodes S can d-separate X and Y, we say that S is a cut-set between X and Y in G.
DAG-faithful	A dataset is *DAG-faithful* if its underlying probabilistic model is DAG structured. See Section 2.2
DAG-Isomorph	A distribution that can be represented by a DAG. See Section 2.2.
Dependency-map (D-map)	A graph that can express all the dependencies of the underlying model. See Definition 3 (Section 2.2).
Drafting	A phase in our Bayesian network learning algorithm TPDA (and TPDA-Π) where a draft graph is generated by using pair-wise statistics. See Section 5.
d-separation	Directed separation of two nodes in a graph according to a set of rules. See Definition 1 (Section 2).
Edge	Connection between a pair of nodes (not necessarily directed).
*EdgeNeeded**	An (potentially exponential-time) routine that determines if a direct edge between two nodes is needed. This routine is guaranteed correct given DAG-faithful assumption. See Section 4.2.1.
EdgeNeeded_H	A routine that determines if a direct edge between two nodes is needed. This routine uses a heuristic. See Section 4.2.2.
EdgeNeeded	A routine that determines if a direct edge between two nodes is needed. This routine is guaranteed correct given monotone DAG-faithful assumption. See Section 4.2.3.
Independency map (I-map)	A graph that can express all the independencies of the underlying model. Definition 3 (Section 2.2)
Monotone DAG-faithful	The underlying model satisfies a stronger assumption than DAG-faithful. See Definition 5 (Section 4.1)
Node ordering	The temporal or causal ordering of the nodes. See Definition 4 (Section 3.1)
OrientEdges	A routine is used in *TPDA* to orient edges. See Section 4.3.

$Path_G(X,Y)$	The set of adjacency paths between X and Y in G. See Definition 5 (Section 4.1).
Perfect map (P-map)	A graph that is both a D-map and an I-map. See Definition 3 (Section 2.2).
SLA	A simple learning algorithm for learning Bayesian networks when the node ordering is not given. See Section 4.
SLA-Π	A simple learning algorithm for learning Bayesian networks when node ordering is given. See Section 3.
Thickening	A phase in our Bayesian network learning *TPDA* (and *TPDA-Π*) that tries to add more edges as a result of CI tests. See Section 5.
Thinning	A phase in our Bayesian network learning *TPDA* (and *TPDA-Π*) that tries to remove edges from the current graph as a result of CI tests. See Section 5.
TPDA	A three phase learning algorithm for learning Bayesian networks when node ordering is not given. See Section 5.1.
TPDA-Π	A three phase learning algorithm for learning Bayesian networks when node ordering is given. See Section 5.3.
v-structure	A structure where two nodes are both connected to a third node and the two nodes are not directly connected. See Section 2.1 (aka "unshielded collider").

Table 1: Terms used

2.2 Input, Output and Assumptions

Our goal is to use the training data to learn an accurate model of the underlying distribution; here, this reduces to identifying exactly which arcs to include. We can state this more precisely using the following ideas:

Definition 3: *Dependency map, independency map and perfect map***:** A graph *G* is a **dependency map (D-map)** of a probabilistic distribution *P* if every dependence relationship derived from *G* is true in *P*; *G* is an **independency map (I-map)** of *P* if every independence relationship derived from *G* is true in *P*. If *G* is both a D-map and an I-map of *P*, we call it a **perfect map (P-map)** of *P*, and call *P* a **DAG-Isomorph** of *G* [Pearl, 1988]. Here we say that *P* and *G* are **faithful** to each other [Spirtes, *et al.*, 1993]. []

While our goal, in general, is a graph that is a P-map of the true distribution, this is not always possible; there are some distributions whose independence relations cannot all be represented. For instance, let *Z* stand for the sound of a bell that rings whenever the outcomes of two fair coins, *X* and *Y*, are the same [Pearl, 1988]. Clearly the only BN structures that can represent this domain must contain $X \rightarrow Z \leftarrow Y$. Notice these networks, however, are not perfect, as they do not represent the facts that *X* and *Z* (resp., *Y* and *Z*) are marginally independent.

While we can create distributions that have no P-maps (such as this $X \rightarrow Z \leftarrow Y$), Spirtes *et al.* (1993) argue that most real-world probabilistic models in social sciences are faithful to Bayesian networks. They also shows that in a strong measure-theoretic sense, almost all Gaussian distributions for a given network structure are faithful. Meek (1995b) proves that the same claim is also hold for discrete distributions. This paper, therefore, focuses on learning Bayesian networks from data sets that are drawn from distributions that have faithful probabilistic models.

The set of conditional independence relations implied in P may not be sufficient to define a single faithful BN model; for example, every distribution can be represented by the graph $\boxed{A \rightarrow B}$ can also be represented by $\boxed{A \leftarrow B}$. The independence relationships, however, are sufficient to define the **essential graph** (also called “pattern”, see [Spirtes *et al.* 1993]) of the underlying BN, where the essential graph of a BN is a graph that has the same edges of the BN and the same “v-structures”. (A triple of nodes X, Y, Z forms a **v-structure** if $X \rightarrow Z \leftarrow Y$ and X is not adjacent to Y.[4]) Note the essential graph *does* specify the direction of the arcs that lead into the collider (here, Z), and also constrains the directions of the other edges to avoid forcing the non-colliders to appear as colliders. We will later use this fact to orient edges when node ordering is not given. Moreover,

Theorem 1: Every DAG-faithful distribution has a unique essential graph [Spirtes *et al.* 1993, Chickering, 1996].

Our algorithms require the assumptions listed in Table 2 about the input data. In addition, the *SLA-Π* and *TPDA-Π* algorithms assume the appropriate node ordering; and the *SLA* and *TPDA* algorithms require a stronger first assumption – monotone DAG-faithfulness assumption.

[4] This is also called an “unshielded collider” [Dash and Druzdzel, 1999].

1.	The records occur independently given the underlying probabilistic model of the data (that is, the dataset is “independent and identically distributed”, iid).
2.	The cases in the data are drawn iid from a DAG-faithful distribution.
3.	The attributes of a table have discrete values and there are no missing values in any of the records.
4.	The quantity of data is large enough for the CI tests used in our algorithms to be reliable; that is $I_D(\ldots) \approx I(\ldots)$.

Table 2 Assumptions

3 Simple Learning Algorithm (given ordering): *SLA-Π*

This section presents the simple *SLA-Π* algorithm, which takes a data set and a (correct) node ordering as input and constructs a Bayesian network structure as output. (Recall that filling in the CPtable parameters is trivial.) Section 4 then provides a general algorithm, *SLA*, that does not require a node ordering, and Section 5 then presents more efficient versions of these algorithms, that use the “three phase” idea.

Section 3.1 first provides a formal specification of our task, which requires specifying the node ordering. It also specifies when this algorithm is guaranteed to perform correctly, and gives its complexity. (Appendix A.3 proves the associated theorem.) Section 3.2 then presents the actual *SLA-Π* algorithm.

3.1 Formal Description

Like many other Bayesian network learning algorithms [Cooper and Herskovits, 1992; Heckerman *et al.*, 1994], our *SLA-Π* system takes as input both a table of database entries and a node ordering.

Definition 4: *Node ordering***: A node ordering** is a total ordering of the nodes of the graph (variables of the domain) – specifying perhaps a causal or temporal ordering. []

This information can specify a causal or temporal order of the nodes of the graph (variables of the domain), in that any node cannot be a cause or happen earlier than the nodes appearing earlier in the order. If a node ordering is consistent with the underlying model of a data set, we say it is a *correct* ordering. For example, in Figure 1, A-B-C-D-E and A-B-D-C-E are two correct orderings.

Of course, we can represent any distribution using a graph that is consistent with *any* node ordering. However, only some such graphs will be DAG-faithful. (As an example, consider a naïve-bayes distribution, with a classification node C pointing to a set of attribute nodes $\{A_i\}$. In the ordering $\langle C, A_1, A_2, \ldots\rangle$, the obvious structure will be the standard naïve-bayes structure, which is a P-map. However, the ordering $\langle A_1, A_2, \ldots, C\rangle$ will typically produce a much larger structure – perhaps even one that is completely connected – which is not a P-map. Here, the first ordering would be considered correct, but the second would not.)

The next section provides an algorithm that can recover the underlying structure, given such a correct ordering together with a dataset that satisfies the conditions listed in Section 2.2.

3.2 Actual *SLA-Π* Algorithm

The *SLA-Π* algorithm, shown in Figure 2, incrementally grows, then shrinks, the graph structure: It first computes a list of all node-pairs that have sufficient mutual information to be considered, $L = \{ (X,Y) \mid I(X,Y) > \varepsilon \}$. As the underlying model is assumed to be DAG-faithful, the resulting graph should reflect all (and only) these dependencies, by including some (possibly indirect) path connecting each such X to Y. *SLA-Π* therefore first determines, for each pair of nodes X,Y, whether there is already sufficient information flow between X and Y in the current structure. This is done in Step 2, "Thickening", which first finds a cut-set C= MinCutSet($A, B; (V,A), \Pi$) separating X from Y in the graph.[5] Note this depends on both the graph structure (V,A) (which changes as new edges

---

[5] Of course, we would like to find as small a cut-set as possible – as that makes the code more efficient, and also means (given a limited number of training instances) that the results will be more reliable. As finding a minimum cut-set (a cut-set with minimum number of nodes) is NP-hard [Acid and Campos, 1996a], we use greedy search to find a small cut-set. The basic idea is to repeatedly add a node to the cut-set that can close the maximum number of paths, until all paths are closed. See [Cheng *et al.* 1997a] for details.

are added to A), and also on the node ordering Π, as that ordering determines the directions of the edges, which identifies which nodes in the paths between X and Y are colliders and which are not.

If the current structure is correct – and in particular, if no arc is needed connecting X to Y – this cut-set C should turn off all of the paths, stopping all information flow between X and Y. This means the CI test: "Is $I_D(X,Y \mid C)$ greater than ε?" should fail. (We typically use the threshold $\varepsilon \approx 0.01$ here.[6]) Otherwise, the data D suggests that an arc is needed, and so *SLA-*Π will add this $\langle X,Y \rangle$ arc.

[6] We regard this as a constant (see Section 5.4.3). Others, including [Fung and Crawford, 1990; Dash and Druzdzel, 1999; Cheng and Greiner, 2000], have used learning techniques to obtain the value of ε that works best for each dataset.

subroutine **SLA-Π(** ***D*: Dataset,** **Π: node ordering,** *ε*: threshold):

returns G = *(V, A):* graph structure

1. Let $V := \{attributes\ in\ \mathbf{D}\}$, $A := \{\}$

   $L := \{ (X,Y) \mid I(X,Y) > \varepsilon \}$ be the list of all pairs of distinct nodes *(X,Y)*

   where X, $Y \in V$ and $X \prec Y$ *in Π,* with at least ε mutual information.

   Begin [Thickening]

2. For each ⟨*X,Y*⟩ in *L*:

   C := MinCutSet(*X, Y; (V,A), Π)*

   If $I_D(X,Y \mid C) > \varepsilon$

   Add *(X,Y)* to *A*

   Begin [Thinning]

3. For each *(X,Y)* in *A*:

   If there are other paths, besides this arc, connecting *X* and *Y*,

   A' := A - (X, Y) *% ie, temporarily remove this edge from A*

   C := MinCutSet(*X, Y; (V,A'), Π)*

   If $I_D(X,Y \mid C) < \varepsilon$

   % ie, if X can be separated from Y in current "reduced" graph

   A := *A'* *% then remove ⟨X,Y⟩ from A*

4. Return *(V,A)*

Figure 2 The *SLA-Π* algorithm

After this sweep (ie, after step 2 in Figure 2), we know that that resulting graph $G_2 = (V, A_2)$ will include a (possibly indirect) connection between each pair of nodes that have a non-trivial dependence. (This is sufficient, thanks to the DAG-faithful assumption.) G_2 may, however, also include other unnecessary arcs – included only because the more appropriate (indirect) connection was not yet been included within the graph when this *X-Y* pair was considered. *SLA-Π* therefore makes another sweep over the arcs produced, again using a CI test to determine if each arc is superfluous; if so, *SLA-Π* removes that arc from the current network. Here, *SLA-Π* first identifies each arc connecting a pair of nodes that is also connected by one or more other paths. Since it is possible that the other arcs already explain the *X-Y* dependency, *SLA-Π* temporarily removes this arc, then computes a cut-set C that should separate X from Y in the reduced graph. If there is no information flow wrt this C – i.e., if $I_D(X,Y \mid C) < \varepsilon$ – then the $\langle X,Y \rangle$ arc was not needed, and so is eliminated.

We prove, in Appendix A.3:

Theorem 2: Given the four assumptions listed in Table 2 (i.e., a "sufficiently large" database of complete instances that are drawn, iid, from a DAG-faithful probability model), together with a correct node ordering, the *SLA-Π* algorithm will recover the correct underlying network. Moreover, this algorithm will require $O(N^2)$ CI tests.

4 Simple (order-free) Learning Algorithm: *SLA*

4.1 Overall *SLA* Algorithm

The *SLA* algorithm, shown in Figure 3, has the same structure as the *SLA-Π* algorithm, as it too incrementally grows, then shrinks, the graph structure. However, as *SLA* does not have access to the node orderings, these growing and shrinking processes are more complex. As with *SLA-Π*, we want to first determine whether there is additional information flow connecting X to Y, beyond the flow implied by the current graph (V,E). Once again, we first seek a cut-set C that should separate X and Y, then add in the $\langle X,Y \rangle$ arc if $I_D(X,Y \mid C) > \varepsilon$. The challenge here is finding an appropriate cut-set: As *SLA-Π* knew the node ordering, it could determine the direction of the edges, and so identify which nodes (in the paths connecting X to Y) are colliders versus non-colliders, and then determine whether to exclude or include them within the cut-set C. *SLA* does not have this information. It therefore uses the *EdgeNeeded* subroutine, defined in Section 4.2 below, to first find an appropriate cut-set and then make this determination. (Note *SLA* uses *EdgeNeeded* in both places that *SLA-Π* had used a single CI test – ie, in both adding new arcs, and also in deleting superfluous ones.[7]) SLA

---

[7] Step-2 of *SLA*, like *SLA-Π*'s Step-2, may produce extra edges as some of the eventual arcs were not included when $\langle X,Y \rangle$ was being considered. In addition, *SLA* may also include extra arcs as it may be unable to find the appropriate cut-set, as it does not know the directions of the arcs.

needs one additional step, beyond the steps used by *SLA-Π:* it must also find the appropriate direction for (some of) the edges.

Subroutine ***SLA(*** ***D*: Dataset,** ε: threshold): returns **G = *(V, E):* graph structure**

1. Let *V= {attributes in **D**}, E= {}*

   $L = \{ (X,Y) \mid I(X,Y) > \varepsilon \}$ be the list of all pairs of distinct nodes *(X,Y)* where $X,Y \in V$ and

   $X \neq Y$, with at least ε mutual information (Equation 2.1)

   Begin [Thickening]

2. For each *(X,Y)* in *L*:

   If *EdgeNeeded((V,E), X, Y; D,* ε *)*

   Add ⟨*X,Y*⟩ to *E*

   Begin [Thinning]

3. For each ⟨*X,Y*⟩ in *E*:

   If there are other paths, besides this arc, connecting *X* and *Y*,

   E' = E - ⟨*X,Y*⟩ *% ie, temporarily remove this edge from E*

   If ¬*EdgeNeeded((V,E'), X, Y; D,* ε *)* then

   % ie, if X can be separated from Y in current "reduced" graph

   E = E' *% then remove* ⟨*X,Y*⟩ *from E*

4. Return[*OrientEdges((V,E), D)*]

Figure 3 SLA: Simple Bayesian Net Structure Learner w/o Ordering

The next two subsections address the two challenges: (1) deciding whether an edge is required, given the rest of the current graph; and (2) orienting the edges.

4.2 Determine if an Edge is Needed

In general, *EdgeNeeded(G, X,Y, …)* tries to determine if there is additional "information flow" between *X* and *Y*, once every known "path" between them has been blocked. Here, it tries to form a cut-set *C* that blocks each *X*-*Y* path, then returns "Yes (ie, an edge is needed)" if the conditional mutual information $I_D(X,Y \mid C)$ exceeds a threshold ε for this cut-set. (We continue to use ε≈0.01.)

An appropriate cut-set *C* should block every known path. *If the node ordering is known*, we can determine *C* immediately (as *SLA-Π* does), and therefore only one CI test is required to check if two nodes are independent. Unfortunately, as *SLA* does know this ordering information, it must use a group of CI tests to find this *C*.

We show below three procedures for this task: Section 4.2.1 shows the straightforward exponential procedure *EdgeNeeded**, which is correct given the assumptions of Table 2 but not efficient. Sections 4.2.2 and 4.2.3 use the idea of quantitative measurements to improve the efficiency, assuming that the data is monotone DAG-Faithful. Section 4.2.2 illustrate the basic ideas using the heuristic procedure *EdgeNeeded_H*, which is very efficient but not always correct. We then use this as a basis for describing *EdgeNeeded* in Section 4.2.3, which is guaranteed to be correct given the monotone DAG-Faithful assumption. (While *SLA* uses only the correct *EdgeNeeded*, the actual *TDPA* algorithm will gain some efficiency by using *EdgeNeeded_H* in some situations; see Section 5.) We first need some definitions:

Definition 5: *paths, open, Monotone DAG Faithful***:**

* $paths_G(X,Y)$ is the set of all adjacency paths between *X* to *Y* in graph *G*.

* $open_G(X,Y|C)$ is the subset of $paths_G(X,Y)$ that are open by cut-set *C*.

* A DAG-Faithful model $G=\langle V,E,\Theta\rangle$ is *Monotone DAG-faithful* iff

For all nodes A, B $\in V$, if $Open_G(X, Y \mid C') \subseteq Open_G(X, Y \mid C)$, then $I(X,Y|C') \le I(X,Y|C)$

[]

Appendix B discusses these notions in more detail.

Using these subroutines, we prove (in Appendix A):

Theorem 3: Given a “sufficiently large” database of complete instances that are drawn, iid, from a monotone DAG-faithful probability model, the *SLA* algorithm will recover the correct underlying essential network. Moreover, this algorithm requires $O(N^4)$ conditional independence tests. []

4.2.1 Subroutine *EdgeNeeded** (Exponential)

Consider any two nodes *X* and *Y*; we assume that *Y* is not an ancestor of *X*. From the Markov condition, we know that we can close all the indirect pipelines between these nodes by setting the cut-set *C* to be all the parents of *Y*, but not any children. Since we do not know which of these Y-neighbor nodes are parents, one approach is to sequentially consider every subset. That is, let C_1

through C_{2^k} be the 2^k subsets of *Y*'s k neighbors. As one of these sets – say C_i – must include exactly the parents of *Y*, $I(X,Y \mid C_i)$ will be effectively 0 if *X* and *Y* are independent. This basic "try each subset" algorithm is used by essentially all other dependency-analysis based algorithms, including the SGS algorithm [Spirtes *et al.*, 1990], the Verma-Pearl algorithm [Verma and Pearl, 1992] and the PC algorithm [Spirtes and Glymour, 1991]. Of course, this can require an exponential number of CI tests.

subroutine ***EdgeNeeded* (****G: graph, X, Y: node, D*: Dataset, ε : threshold***): boolean***

% Returns true iff the dataset D requires an arc between X and Y,

% in addition to the links currently present in G

% Also sets global CutSet

1. Let S_X = Ngbr(X) ∩ AdjPath(X,Y) be the neighbors of *X* that are on an *adjacency* path between *X* and *Y;* similarly S_Y = Ngbr(Y) ∩ AdjPath(X,Y). *CutSet* := {}
2. Remove from S_X any currently known child-nodes of X*;* and from S_Y any child-nodes of Y.
3. **3.** For each condition-set $C \in \{S_X, S_Y\}$ do

   For each subset $C' \subseteq C$ do

   Let $s := I_D(X, Y|C')$. [Equation 2.2]

   If $s < \varepsilon$,

   Let $CutSet := CutSet \cup \{ \langle\{X,Y\},C'\rangle \}$;

   return ('false'). *% ie, data does NOT require an arc between these nodes*
4. Return ('true') % *ie, there is significant flow from X to Y*

Figure 4 *EdgeNeeded** subroutine

In Step 3, the procedure tries every subset *C'* of *C*. If one of the *C'* can successfully block the information flow between X and Y, then we consider *C*' as a proper cut-set that can separate X and Y. The procedure then returns 'false' since no extra edge is needed between X and Y. The cut-set information is stored in a global structure *CutSet* , which is used later in the procedure *OrientEdges*.

By replacing *EdgeNeeded* with procedure *EdgeNeeded** in *SLA*, we can define an algorithm *SLA**, which is guaranteed to be correct given DAG-Faithfulness assumption. The correctness proof is omitted since it is very straightforward.

4.2.2 Subroutine *EdgeNeeded_H* (Heuristic)

As there is no way to avoid an exponential number on CI tests *if the result of each trial is only a binary 'yes' or 'no'*, we therefore developed a novel method that uses *quantitative* measurements -- measuring the *amount* of information flow between nodes *X* and *Y*, for a given cut-set *C*. For a given structure *G*, and pair of nodes *X* and *Y*, *EdgeNeeded_H* begins with a certain set *C* that is guaranteed to be a superset of a proper cut-set. It then tries to identify and remove the inappropriate nodes from *C* one at a time, by using a group of mutual information tests. This entire process requires only $O(k^2)$ CI tests (as opposed to the *EdgeNeeded**, which must consider testing each of the 2^k subsets of *Y*'s neighbors). In Appendix A.1, we prove that this quantitative CI test method is correct whenever the underlying model is *monotone DAG-faithful*.

From the above discussion we know that if *X* and *Y* are not adjacent, then either the parents of *X* or the parents of *Y* will form a proper cut-set. Therefore, we can try to find a cut-set by identifying the parents of *X* from *X*'s neighborhood (or parents of *Y* from *Y*'s neighbors). From the definition of monotone DAG-faithfulness, we know that if we do not close any path then the information flow will not decrease. Given the assumption that removing a parent node of *X* (or *Y*) from the condition-set containing all the neighbors of *X* (or *Y*) will seldom close any path, we conclude that we will not make the information flow decrease if we remove a parent node. Therefore, we can find a proper cut-set by distinguishing the parent nodes versus child nodes in the neighborhood of *X* (or *Y*) using mutual information tests. To illustrate the working mechanism of this separating procedure, we use the simple Bayesian network whose true structure is shown in Figure 5.

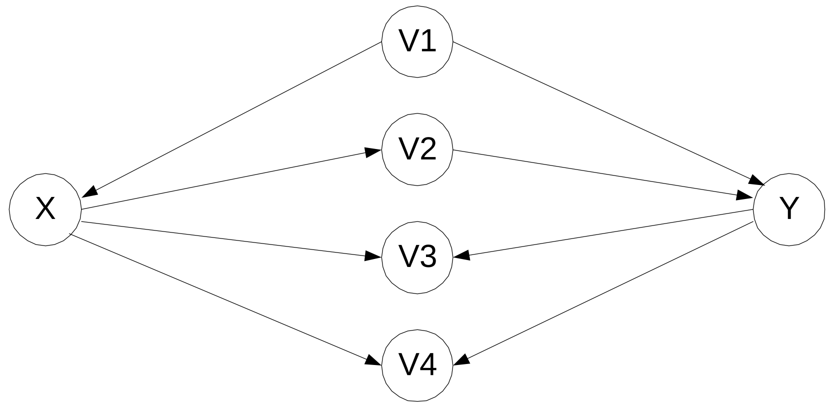

Figure 5: Bayesian net to illustrate the *EdgeNeeded* algorithm

Suppose we trying to determine whether there should be a direct edge between *X* and *Y*, where (we assume) we know all of the other relevant edges of the true structure; see Figure 5. If the node ordering is given, i.e., the directions of the edges are known, we easily see that *V1* and *V2* are parents of *Y*, and that *Y* is not an ancestor of *X*. So *P*={*V1*,*V2*} is a proper cut-set that can separate *X* and *Y*. However, as we do not know the directions of edges, we do not know whether a node is a parent of *Y* or not. Our *EdgeNeeded_H* procedure (Figure 6), therefore, first gets S_X and S_Y, which both happen to be {*V1,V2,V3,V4*}.[8] In step "3 1)", we use *C*={*V1,V2,V3,V4*} as the condition-set and perform a CI test – determining if $I_D(X,Y|C) > \varepsilon$ for this *C*. While this condition-set does close the paths *X*-*V1*-*Y* and *X*-*V2*-*Y*, it also opens *X*-*V3*-*Y* and *X*-*V4*-*Y* (see the definition of d-separation in Section 2.2). This means it does not separate *X* and *Y*, and so the CI test will fail, meaning the algorithm will go to step "3 2)". This step considers each 3-node subsets of {*V1*,*V2*,*V3*,*V4*} as a possible condition-set: viz., {*V1*,*V2*,*V3*}, {*V1*,*V2*,*V4*}, {*V1*,*V3*,*V4*} and {*V2*,*V3*,*V4*}. As the data is monotone DAG-faithful (Definition 5 in Section 2.2), either {*V1*,*V2*,*V3*} or {*V1*,*V2*,*V4*} will give the smallest value on CI tests. This is because they each leave only one path open (path *X*-*V3*-*Y* or path *X*-*V4*-*Y* respectively) while each of the other condition-sets leave open three paths. Assuming {*V1*,*V2*,*V3*} gives the smallest value, we will conclude that *V4* is a collider, and so will remove *V4* from the condition-set, and will never again consider including this node again (in this *X*-*Y* context).

In the next iteration, *EdgeNeeded_H* considers each 2-node subset of {*V1,V2,V3*} as a possible condition-sets; viz., {*V1,V2*}, {*V1,V3*} and {*V2,V3*}. After three CI tests, we see that the cut-set {*V1,V2*} can separate *X* and *Y* , as $I_D(X,Y \mid \{V1,V2\}) \approx 0$. *EdgeNeeded_H* therefore returns “false”, which means SLA will not add a new arc here, as is appropriate.

(Given an alternative dataset, drawn from a distribution where there *was* an additional dependency between *X* and *Y*, this $I_D(X,Y \mid \{V1,V2\})$ quantity would remain large, and *EdgeNeeded_H* would continue seeking subsets. Eventually it would find that there was significant flow between *X* and *Y* for all of the condition-sets considered, which means *EdgeNeeded_H* would return “true”, which would cause *SLA* to add a direct *X*-*Y* link.)

As *EdgeNeeded_H* will “permanently” exclude a node on each iteration, it will not have to consider every subset of S_Y as a condition-set, and thus it avoids the need for an exponential number of CI tests.

---

[8] In general, these sets will be different; in that case, we will consider each of them, as we know at least a subset of one of them should work if the two nodes are not connected. This is because at least one of the two nodes must be a non-ancestor of the other.

subroutine ***EdgeNeeded_H*** ***(****G: graph, X, Y: node, D*: Dataset, ε : threshold***): boolean***

% Returns true iff the dataset D requires an arc between X and Y,

% in addition to the links currently present in G

% Also sets global CutSet

1 Let S_X = Ngbr(X) ∩ AdjPath(X,Y) be the neighbors of *X* that are on an *adjacency* path between *X* and *Y;* similarly S_Y = Ngbr(Y) ∩ AdjPath(X,Y).

2 Remove from S_X any currently known child-nodes of X*;* and from S_Y any child-nodes of Y.

3 For each condition-set $C \in \{S_X, S_Y\}$ do

1) Let $s := I_D(X, Y|C)$. [Equation 2.2]

If $s < \varepsilon$, let $CutSet := CutSet \cup \{ \langle\{X,Y\},C'\rangle \}$; return ('false').

% ie, data does NOT require an arc between these nodes

2) While $|C| > 1$ do

a. For each *i*, let $C := C \setminus \{the\ i^{th}\ node\ of\ C\}$, $s_i = I(X, Y\,|\,C_i)$.

b. Let $m = argmin_i \{ s_1, s_2, \ldots \}$

c. If $s_m < \varepsilon$, % $s_m = \min(s_1, s_2, \ldots)$

Then return ('false');

Else If $s_m > s$ THEN BREAK (get next C, in Step 3)

Else Let s := s_m, $C := C_m$, CONTINUE (go to Step "3.2)").

4. Return ('true') % ie, there is significant flow from X to Y

Figure 6 *EdgeNeeded_H* subroutine

4.2.3 Subroutine *EdgeNeeded* (Guaranteed)

This *EdgeNeeded_H* procedure uses the heuristic that removing a parent of *X (or Y)* from the condition-set will seldom close any path between *X* and *Y*. However, such a closing can happen in some unusual structures. In particular, it may not be able to separate nodes *X* and *Y* when the structure satisfies both of the following conditions.

(1) There exists at least one path from *X* to *Y* through a child of *Y* and this child-node is a collider on the path.

(2) In such paths, there is one or more colliders besides the child node and all of these colliders are the ancestors of *Y*.

In such structures, *EdgeNeeded_H* may incorrectly think a parent of *Y* is a child of *Y* and so erroneously remove it from the conditioning. As a result, the procedure will fail to separate two nodes that can be separated. Figure 7 shows an example of such a structure.

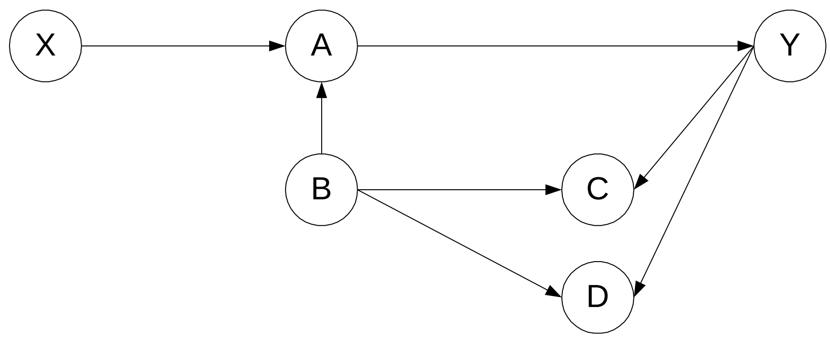

Figure 7: Problematic Case

Here we may try to separate *X* and *Y* using a certain subset of the neighbor-set of *Y*, N_2={*A*,*C*,*D*}. The *EdgeNeeded_H* procedure will first use {*A*,*C*,*D*} as the condition-set. As this leaves two paths open, *X*-*A*-*B*-*C*-*Y* and *X*-*A*-*B*-*D*-*Y*, it will therefore consider the 2-element subsets {*A*,*C*}, {*A*,*D*} and {*C*,*D*} as possible condition-sets. Each of these condition-sets leaves open one path --- viz., *X*-*A*-*B*-*C*-*Y*, *X*-*A*-*B*-*D*-*Y* and *X*-*A*-*Y* respectively. If the mutual information between *X* and *Y* is smallest when *X*-*A*-*Y* is open, the procedure will remove node *A* from further trials. Clearly, this will lead to a failure on separating *X* and *Y*. In this example, it happens that the neighbor-set of *X*, C={*A*}, can separate *X* and *Y*, but there are more complex models that this procedure will fail to find, from either S_X or S_Y. However, such structures are rare in real world situations and we have found this heuristic method works very well in most cases.

The procedure, *EdgeNeeded*, defined in Figure 8, is correct, even for such problematic situations. We prove (in Appendix A.1) that this algorithm will find the correct structures for all probabilistic models that are monotone DAG-faithful.

The major difference between *EdgeNeeded_H* and *EdgeNeeded* is that, in addition to including/excluding each neighbor of X – called S_x – *EdgeNeeded* will also consider including/excluding each neighbors of those neighbors (called S_X'). (Similarly S_Y and S_Y'). Notice also that *EdgeNeeded* only considers one of the sets – either $S_X \cup S_X'$ or $S_Y \cup S_Y'$; n.b., it does not have to consider *both* of these sets.

Since altering the statuses of two consecutive nodes in a path can always close the path (two consecutive nodes cannot both be colliders in a path), we know there is a subset of $S_X \cup S_X'$ (respectively, of $S_Y \cup S_Y'$) that can close all the paths that connect X and Y through two or more nodes. The only open paths are those connecting X and Y through one collider. Under this circumstance, we can remove all the colliders connecting X and Y without opening any previously closed paths. Thus, all paths between X and Y in the underlying model can be closed. For the example shown in Figure 7, this procedure will first use $\{A,B,C,D\}$ as the condition-set. Clearly, X and Y can be successfully separated using this cut-set.

As in *EdgeNeeded** (Figure 4) and *EdgeNeeded_H*, the *EdgeNeeded* procedure also uses the global structure *CutSet* to store the cut-sets between pairs of nodes.

subroutine ***EdgeNeeded (****G: graph, X, Y: node, D*: Dataset, ε : threshold***): boolean***

% Returns true iff the dataset D requires an edge between X and Y,

% in addition to the links currently present in G

% Also sets global CutSet

1. Let S_X = Ngbr(X) ∩ AdjPath(X,Y) be the neighbors of *X* that are on an *adjacency* path between *X* and *Y;* similarly S_Y = Ngbr(Y) ∩ AdjPath(X,Y).

2. Let $S_X' = AdjPath(X,Y) \bigcap (\bigcup_{x \in S_X} Ngbr_G(x) - S_X)$ be the neighbors of the nodes in S_X that are on the *adjacency* paths between *X* and *Y*, and do not belong to S_X; similarly $S_Y' = AdjPath(X,Y) \bigcap (\bigcup_{y \in S_y} Ngbr_G(y) - S_Y)$

3. Let *C* be smaller of { $S_X \cup S_X'$, $S_Y \cup S_Y'$ }
   (ie, if $|S_X \cup S_X'| < |S_Y \cup S_Y'|$ then $C = S_X \cup S_X'$ else $C = S_Y \cup S_Y'$.)

4. Let $s = I_D(X,Y|C)$. [Equation 2.2].

5. If $s < \varepsilon$, let $CutSet := CutSet \cup \{\langle \{X,Y\},C \rangle\}$; return ('false') *% no edge is needed*

6. While $|C| > 1$ do

   (a) For each *i*, let $C_i = C \setminus \{the\ i^{th} node\ of\ C\}$, $s_i = I_D(X, Y \mid C_i)$.

   (b) Let $m = argmin_i \{ s_1, s_2, \ldots \}$

   (c) If $s_m < \varepsilon$, % $s_m = \min(s_1, s_2, \ldots)$

   Then let $CutSet := CutSet \cup \{\langle \{X,Y\}, C_m \rangle\}$; return ('false');

   Else If $s_m > s$ THEN BREAK (go to step "7")

   ELSE let $s := s_m$, $C := C_m$, CONTINUE (go to step "6").

7. Return ('true') % ∃ significant flow from X to Y

Figure 8 *EdgeNeeded* subroutine

4.3 Orienting Edges

Among the nodes in Bayesian networks, only colliders can let information flow pass through them when they are instantiated. The working mechanism for identifying colliders is described as follows. For any three nodes *X*, *Y* and *Z* of a Bayesian network of the form X-Y-Z (ie, *X* and *Y*, and

Y and Z, are directly connected; and X and Z are not directly connected), there are only three possible structures, (1) $X \rightarrow Y \rightarrow Z$, (2) $X \leftarrow Y \rightarrow Z$ and (3) $X \rightarrow Y \leftarrow Z$. Among them, only the third type (called a *v-structure*) can let information pass from X to Z when Y is instantiated. In other words, only the v-structure makes X and Z dependent conditional on $\{Y\}$ – ie, only here is $I(X,Z|\{Y\}) > 0$.

Using this characteristic of Bayesian networks, we can identify all the v-structures in a network and orient the edges in such structures using CI tests. Then, we can try to orient as many edges as possible using these identified colliders.

As noted above, the methods based on identifying colliders will not be able to orient all the edges in a network. The actual number of arcs that can be oriented is limited by the structure of the network. (In an extreme case, when the network does not contain any v-structures, these methods may not be able to orient any edges at all.) However, this method is quite popular among Bayesian network learning algorithms due to its efficiency and reliability [Spirtes and Glymour, 1991; Verma and Pearl, 1992]. There are a few algorithms [Lam and Bacchus, 1994; Friedman and Goldszmidt, 1996] that use pure search & scoring methods to search for the correct directions of the edges. But these methods are generally slower than collider identification based methods since the search space is much larger when node ordering is not given [Dash and Druzdzel, 1999].

Procedure *OrientEdges(G=⟨V,E⟩: graph)*

% Modifies the graph G by adding directions to some of the edges

% Uses global variable *CutSet*, defined by *EdgeNeeded*

1. For any three nodes *X*, *Y* and *Z* that *X* and *Y*, and *Y* and *Z*, are directly connected; and *X* and *Z* are not directly connected

   if $\langle\{X,Z\},C\rangle \in$ *CutSet* and $Y \notin C$, or $\langle\{X,Z\},C\rangle \notin$ *CutSet*

   let *X* be a parent of *Y* and let *Z* be a parent of *Y*.
2. For any three nodes *X*,*Y*,*Z*, in *V*

   if (i) *X* is a parent of *Y*, (ii) *Y* and *Z* are adjacent,

   (iii) *X* and *Z* are not adjacent, and (iv) edge (*Y*, *Z*) is not oriented,

   let *Y* be a parent of *Z*.
3. For any edge (*X*,*Y*) that is not oriented,

   If there is a directed path from *X* to *Y*, let *X* be a parent of *Y*.

Figure 9: *OrientEdges* subroutine

In step 1, procedure *OrientEdges* tries to find a pair of nodes that may be the endpoints of a v-structure. It then tries to check whether Y is a collider by searching the global structure *CutSet* – if *Y* is a collider in the path *X*-*Y*-*Z*, *Y* should not be in the cut-set that separate *X* and *Z*. This process continues until all triples of nodes have been examined. Step 2 uses the identified colliders to infer the directions of other edges. The inference procedure applies two rules: (1) If an undirected edge belongs to a v-structure and the other edge in the structure is pointing to the mid-node, we orient the undirected edge from the mid-node to the end node. (Otherwise, the mid-node would be a collider and this should have been identified earlier.) (2) For an undirected edge, if there is a directed path between the two nodes, we can orient the edge according to the direction of that path. These latter two rules are the same as those used in all of the other collider-identification based methods mentioned above.

Appendices A.1 and A.2 prove that this overall *SLA* procedure is both correct and efficient (in terms of the number of CI tests).

5 The Three-Phase Dependency Analysis Algorithm

While the algorithm sketched above is guaranteed to work correctly, there are several ways to make it more efficient. We have incorporated two of these ideas into the *TPDA* algorithm. First, rather than start from empty graph (with no arcs), *TPDA* instead uses an efficient technique to produce a graph that we expect will be close to the correct one. Second, rather than call the full *EdgeNeeded* procedure for each check, *TPDA* instead uses the approximation *EdgeNeeded_H* is some places, and only calls the correct *EdgeNeeded* procedure at the end of the third phase.

After Section 5.1 presents the general *TPDA* algorithm, Section 5.2 uses an example to illustrate the ideas, and Section 5.3 then overviews the *TPDA-Π* algorithm that can use a node ordering.

5.1 *TPDA* Algorithm for Learning, without Ordering

The three phases of the *TPDA* algorithm are *drafting*, *thickening* and *thinning*. Unlike the simpler *SLA* algorithm (Figure 3), which uses *EdgeNeeded* to decide whether to add an edge *starting from the empty graph*, *TPDA* begins with a "drafting" phase, which produces an initial set of edges based on a simpler test – basically just having sufficient pair-wise mutual information; see Figure 10. The draft is a singly-connected graph (a graph without loops), found using (essentially) the Chow-Liu [1968] algorithm (see Section 7.1). The other two phases correspond directly to steps in the *SLA* algorithm. The second phase, "thickening", corresponds to *SLA*'s Step 2: here *TPDA* adds edges to the current graph when the pairs of nodes cannot be separated using a set of relevant CI tests. The graph produced by this phase will contain all the edges of the underlying dependency model when the underlying model is DAG-faithful. The third "thinning" phase corresponds to Step 3: here each edge is examined and it will be removed if the two nodes of the edge are found to be conditionally independent. As before, the result of this phase contains exactly the same edges as those in the underlying model, given the earlier assumptions. *TPDA* then runs the O*rientEdges* procedure to orient the essential arcs of the learned graph, to produce an essential graph. (Recall that the direction of some of the arcs is irrelevant, in that a structure is correct under any appropriate assignment of directions.)

subroutine ***TPDA(*** ***D*: Dataset,** ε: threshold): returns **G = *(V, E):* graph structure**

Begin [Drafting].

1. Let $V = \{attributes\ in\ \mathbf{D}\}$, $E = \{\}$

   $L = \{\langle X, Y \rangle \mid I(X, Y) > \varepsilon\}$ be the list of all pairs of distinct nodes $\langle X,Y \rangle$ where $X,Y \in V$ and $X \neq Y$, with at least ε mutual information.

2. Sort L into decreasing order, wrt $(I(X,Y)$
3. For each $\langle X,Y \rangle$ in L:

   If there is no adjacency path between X and Y in current graph (V,E)

   add $\langle X,Y \rangle$ to E and

   remove $\langle X,Y \rangle$ from L.

Begin [Thickening]

4. For each $\langle X,Y \rangle$ in L:

   If ~~*EdgeNeeded((V,E), X, Y; D, ε)*~~ *EdgeNeeded_H((V,E), X, Y; D, ε)*

   Add $\langle X,Y \rangle$ to E

Begin [Thinning].

5. For each $\langle X,Y \rangle$ in E:

   If there are other paths, besides this arc, connecting X and Y,

   $E' = E - \langle X,Y \rangle$ *% ie, temporarily remove this edge from E*

   If $\neg$*EdgeNeeded_H((V,E'), X, Y; D, ε)* *% ie, if X can be separated from Y*

   % in current "reduced" graph

   $E = E'$ *% then remove* $\langle X,Y \rangle$ *from E*

6. For each $\langle X,Y \rangle$ in E:

   If X has at least three neighbors other than Y, or Y has at least three neighbors other than X,

   $E' = E - \langle X,Y \rangle$ *% ie, temporarily remove this edge from E*

   If $\neg$*EdgeNeeded((V,E'), X, Y; D, ε)*

% ie, if X can be separated from Y in current "reduced" graph

   $E = E'$ *% then remove* $\langle X,Y \rangle$ *from E*

7. Return[*OrientEdges*((V,E), D)]

Figure 10 *TPDA* Algorithm

5.2 Example

Here we illustrate this algorithm using a multi-connected network, borrowed from Spirtes *et al.* (1993). Our data set is drawn from the Bayesian network shown in Figure 11(a). Of course, our learning algorithm does not know this network, nor even the node ordering. Our task is to recover the underlying network structure from this data. We first compute the mutual information of all 10 pairs of nodes (Step 2). Suppose the mutual information is ordered $I(B,D) \geq I(C,E) \geq I(B,E) \geq I(A,B) \geq I(B,C) \geq I(C,D) \geq I(D,E) \geq I(A,D) \geq I(A,E) \geq I(A,C)$, and all the mutual information is greater than ε (i.e., $I(A,C) > \varepsilon$).

In step 3, *TPDA* iteratively examines a pair of nodes from *L*, and connects the two nodes by an edge and removes the node-pair from *L* if there is no existing adjacency path between them. At the end of this phase, $L = [\langle B,C\rangle, \langle C,D\rangle, \langle D,E\rangle, \langle A,D\rangle, \langle A,E\rangle, \langle A,C\rangle]$ contains the pairs of nodes that are not directly connected in Phase I but have mutual information greater than ε. The draft is shown in Figure 11(b). We can see that the draft already resembles the true underlying graph; the only discrepancies are that the edge $\langle B,E\rangle$ is wrongly added and $\langle D,E\rangle$ and $\langle B,C\rangle$ are missing because of the existing adjacency paths (*D*-*B*-*E*) and (*B*-*E*-*C*).

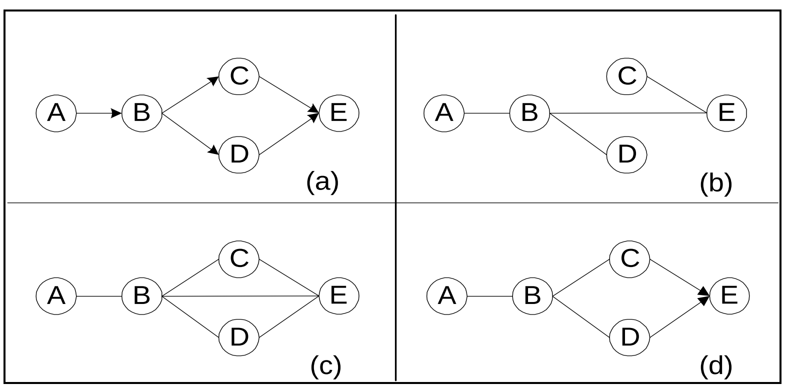

Figure 11: A simple multi-connected network and the results after Phase I, II, III of TPDA.

When creating the draft, we try to minimize the number of missing arcs and the number of wrongly added arcs compared to the (unknown) real model. Since we use only pair-wise statistics, reducing one kind of errors will often increase the other. Our stopping condition reflects a trade-off between the two types of errors – the draft-learning procedure stops when every pair-wise dependency is expressed by an adjacency path in the draft. As an adjacency path may not be a true open path, some pair-wise dependencies may not be really expressed in the draft --- for example, as the dependency between B and C appears to be explained by B-E-C we will not add an B-C arc. Note however that B-E-C is not a true open path.

Sorting the mutual information from large to small in *L* is a heuristic, justified by the intuition that a pair with larger mutual information is more likely to represent a direct connection (an edge) than a pair with smaller mutual information, which may represent an indirect connection. In fact, this intuition is provably correct when the underlying graph is a singly connected graph (a graph without loops). In this case, Phase I of this algorithm is essentially the Chow–Liu algorithm, which is guaranteed to produce a network that is correct (as DAG-Faithful here means the true distribution will have a tree structure); here the second and the third phases will not change anything. Therefore, the Chow-Liu algorithm can be viewed as a special case of our algorithm for the Bayesian networks that have tree structures.

Although the draft can be anything from an empty graph to a complete graph,[9] without affecting the correctness of the final outcome of the algorithm, the closer the draft is to the real underlying model, the more efficient the algorithm will be

The edge-set *E* produced by Phase I may omit some of *L*'s node-pairs only because there were other adjacency paths between the pairs of nodes. The second phase, "Thickening", therefore uses a more elaborate test, *EdgeNeeded_H*, to determine if we should connect those pairs of nodes.

This phase corresponds exactly to *SLA*'s Step2, except (1) rather than the entire list *L*, Thickening only uses the subset of "correlated" node-pairs that have not already been included in *E*, and (2) it uses the heuristic *EdgeNeeded_H* rather than the guaranteed *EdgeNeeded*; see Figure 6.

In more detail, here TPDA examines all pairs $\langle X,Y \rangle$ of nodes that remain in *L* --- i.e., the pairs of nodes that have mutual information greater than ε and are not directly connected. It then adds an edge between $\langle X,Y \rangle$ unless *EdgeNeeded_H* states that these two nodes are found to be independent conditional on some relevant cut-set.

[9] Note that our draft will always be a tree structure.

In our example, Figure 11(c) shows the graph after the Thickening Phase. Arcs ⟨*B*,*C*⟩ and ⟨*D*,*E*⟩ are added because *EdgeNeeded_H* cannot separate these pairs of nodes using CI tests. Arc ⟨*A*,*C*⟩ is not added because the CI tests reveal that *A* and *C* are independent given cut-set {*B*}. Edges ⟨*A*,*D*⟩, ⟨*C*,*D*⟩ and ⟨*A*,*E*⟩ are not added for similar reasons.

Appendix A.1 proves that this phase will find all of the edges of the underlying model – ie, no edge of the underlying model is missing after this phase. The resulting graph may, however, include some extra edges --- ie, it may fail to separate some pairs of nodes that are actually conditionally independent. This is because:

1. Some real edges may be missing until the end of this phase, and these missing edges can prevent *EdgeNeeded_H* from finding the correct cut-set.
2. As *EdgeNeeded_H* uses a heuristic method, it may not be able to find the correct cut-set for some classes of structures; see Section 4.2.3.

Since both of the first two phases (drafting and thickening) can add unnecessary edges, this third phase, "Thinning", attempts to identify these wrongly-added edges and remove them. This phase corresponds to SLA-Step3. While *EdgeNeeded_H* and *EdgeNeeded* have the same functionality and require the same $O(N^4)$ CI tests, in practice we have found that *EdgeNeeded_H* usually uses fewer CI tests and requires smaller condition-sets. *TPDA* therefore does a preliminary sweep using the heuristic *EdgeNeeded_H* routine, as an initial filter. To ensure that *TPDA* will always generate a correct structure, *TPDA* then double-checks the remaining edges using the slower, but correct *EdgeNeeded*; see Figure 8. Note that the second sweep does not examine every edges – it examines an edge X-Y only if X has at least three other neighbors besides Y or Y has at least three other neighbors besides X. This is safe because if both X and Y have at most two neighbors, *EdgeNeeded_H* will actually try every subset of the neighbors and make the correct decision. So, in the real-world situations when the underlying models are sparse, the correct procedure *EdgeNeeded* is seldom called. This also makes it affordable to define a correct algorithm *TPDA** that does not require the monotone DAG-Faithful assumption, by simply replacing *EdgeNeeded* with the exponential *EdgeNeeded**. The algorithm should still be efficient in most cases since the expensive *EdgeNeeded** will seldom be called.

The 'thinned' graph of our example, shown in Figure 11(d), has the same structure as the original graph. Arc $\langle B,E\rangle$ is removed because B and E are independent given $\{C,D\}$. Given that the underlying dependency model has a monotone DAG-faithful probability distribution (and that we have sufficient quantity of data, etc), the structure generated by this procedure contains exactly the same edges as those of the underlying model.

Finally, *TPDA* orients the edges. Here, it can orient only two out of five arcs, viz., $\langle C,E\rangle$ and $\langle D,E\rangle$. Note these are the only arcs that need to be oriented; any distribution that can be represented with the other arcs oriented one way, can be represented with those arcs oriented in any other consistent fashion.[10] Hence, this is the best possible – i.e., no other learning method can do better for this structure. Note the number of "direction"-able arcs depends on the number of edges and colliders in the true distribution. For instance, 42 out of 46 edges can be oriented in the ALARM network (see Section 6.1).

5.3 *TPDA-Π* Algorithm for Learning, given Ordering

Our deployed PowerConstructor system actually used a variant of *SLA-Π,* called *TPDA-Π,* to learn from data together with a node order. As the name suggests, *TPDA-Π* is quite similar to *TPDA*, and in particular, uses the same three phases – *viz.* drafting, thickening and thinning. In the first phase, *TPDA-Π* computes mutual information of each pair of nodes as a measure of closeness, and creates a draft based on this information. The only way this phase differs from *TPDA*'s is that, while *TPDA* adds an edge if there is no *adjacency* path between the two nodes, *TPDA-Π* adds an arc if there is no *open* path. The difference means that the stopping condition of *TPDA-Π* is finer – it stops when every pair-wise dependency is expressed by an open path in the draft. To illustrate the differences, we use the same multi-connected network as in Section 5.2. For instance, the $B \rightarrow C$ arc is added because we know that the adjacency path *B-E-C* is not an open path. The result of Phase I is shown in Figure 12(b). As before, *TPDA-Π* may miss some edges that do belong in the model.

---

[10] Assuming we avoid any node ordering that introduce inappropriate v-structures.

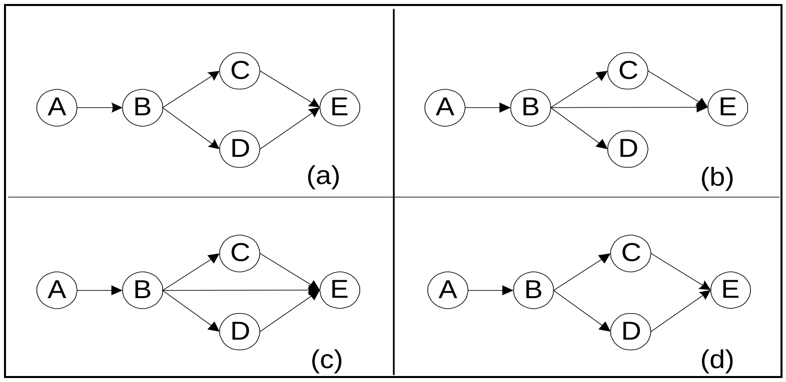

Figure 12 A simple multi-connected network and the results after Phase I, II, III of *TPDA-Π.*

Phase II therefore uses a more definitive test to determine if we should connect each such pair: by first finding a cut set *C* that should separate *A* and *B* (if the current graph is correct), and then connecting these nodes if there is additional information flow between these nodes, after using *C*. As *TPDA-Π* knows the direction of the edges, it can easily determine a sufficient cut-set. For example, to separate two nodes *X* and *Y*, where *X* appears earlier in the ordering, we can just set *C* to be the parents of *Y*. This means Phase II of *TPDA-Π* is much simpler that Phase II of *TPDA*, as the latter, not knowing the direction of the edges, has to use a *set* of CI tests for each such decision.

Since these phases will include an edge between two nodes except when they are independent, the resulting graph is guaranteed to include all of the correct edges whenever the underlying model is DAG-faithful; see the proof in Appendix A.3. In our example, the graph after Phase II is shown in Figure 12(c). We can see that the graph induced after Phase II contains all the real arcs of the underlying model, that is, it is an I-map of the underlying model. This graph may, however, include some additional, incorrect edges. (This is because some edges will not be added until the end of this phase, and these missing edges might have prevented a proper cut-set from being found.) Similar to our discussion in Section 5.2, if we use *SLA-Π* instead, the structure after "thickening" may be a complete graph – where each node is connected to every other node.

Since both Phase I and Phase II can add some incorrect arcs, we use a third phase, "Thinning", to identify those wrongly-added arcs and remove them. As in Phase II, we need only one CI test to make this decision. However, this time we can be sure that the decision is correct, as we know that the current graph is an I-map of the underlying model. (This was not true until the end of Phase II: thickening.) Since we can remove all the wrongly-added arcs, the result after Phase III is guaranteed to be a perfect map (see Section 2.2); see proof in Appendix A.3. The 'thinned' graph of our example is shown in Figure 12(d), which has the structure of the original graph. Hence, after the three phases, the correct Bayesian network is rediscovered. (Recall that *TPDA-Π* does not need to orient the edges as it already knows the direction of the arc when connecting two nodes, as that comes from the node ordering.)

5.4 Discussion

5.4.1 The Three-Phase Mechanism

Virtually all dependency-analysis-based algorithms have to determine whether there should be an edge or not between every pair of nodes in a network, and $O(N^2)$ such decisions will allow us to determine the correct network structure. However, if we require every such decision to be correct from the beginning, each single decision can require an exponential number of CI tests. Unlike the other algorithms, we divide the structure learning process into three phases. In the first and second phases, we will allow some decisions to be incorrect (although of course we try to minimize such bad decisions). In the general *TPDA* version (without ordering), for each pair of nodes, each decision in Phase I requires one CI test and each decision in Phase II requires $O(N^2)$ CI tests, where N is the number of nodes. Phase III then requires $O(N^2)$ CI tests to verify each edge proposed in these two phases. Hence, we must make $O(N^2)$ correct decisions to produce the BN structure, and each such decision requires $O(N^2)$ CI tests. This means *TPDA* requires at most $O(N^4)$ CI tests to discover the edges.

5.4.2 Quantitative Conditional Independence Tests

Although the three-phase mechanism alone is enough to avoid the exponential number of CI tests in the special case when the node ordering is given (*TPDA-Π*), it must work with the quantitative CI test method to avoid exponential complexity in the general case.

Our algorithms use conditional mutual information tests as quantitative CI tests. However, it is also possible to use other possible quantitative CI tests, such as the likelihood-ratio chi-squared tests or the Pearson chi-squared test [Agresti, 1990]. We view Bayesian network learning from an information-theoretic perspective as we think it provides a natural and convenient to present our algorithms. Moreover, by using information theoretic measures, we can easily relate our algorithms to entropy scoring and MDL-based algorithms like the BENEDICT Algorithm [Acid and Campos, 1996b] and the Lam-Bacchus Algorithm [Lam and Bacchus, 1994]; see Section 7 below. One of our further research directions is to combine our approach with the cross entropy or MDL based scoring approach.

5.4.3 Threshold in Conditional Independence Tests

Like all other conditional independence test based algorithms, our algorithms rely on the effectiveness of the conditional independence tests to learn the accurate structures. Unfortunately, these tests are sensitive to the noise when sample sizes are not large enough. The common practice to overcome this problem is to use some technique to adjust the threshold ε according to the sample size and the underlying distribution of the data.

Because people may also need to learn Bayesian networks of different complexities, our system allows users to change the threshold value from the default value. However, we also try to make the threshold value less sensitive to the sample size. Instead of automatically adjusting the threshold according to the sample size, we developed an empirical formula to filter out the noise in the mutual information tests. We consider a high dimensional mutual information test as the sum of many individual low dimensional mutual information tests. The individual mutual information will only contribute to the sum if it meets certain criterion, which takes the degrees of freedom of these tests into consideration. As a result, we found our mutual information tests to be quite reliable even when the date sets are small. When the data sets are larger, the empirical formula has very little effect. Therefore, our algorithm can achieve good accuracy using the same threshold when sample sizes are different. In practice, we find that adjusting threshold in our system according to sample size is not necessary and the default threshold value is good for most of the real-world data sets, where the underlying models are sparse. Our experimental results on the three benchmark data sets presented in Section 6 are all based on the default threshold.

In our work of Bayesian network *classifier* learning, we also search for the best threshold using a wrapper approach, but here attempting to maximize the prediction accuracy [Cheng and Greiner, 2001]; see also the results in [Page and Hatzis, 2001].

5.4.4 Incorporating Domain Knowledge

A major advantage of Bayesian networks over many other formalisms (such as artificial neural networks) is that they represent knowledge in a “semantic” way, in that the individual components (such as specific nodes, or arcs, or even CPtable values) have some meaning in isolation – which can be understood independent of the “meaning” of Bayesian network as a whole [Greiner *et al.*, 2001]. This makes the network structure relatively easy to understand and hence to build.

As *TPDA* is based on dependency analysis, it can be viewed as a constraint based algorithm, which uses CI test results as constraints. Therefore, domain knowledge can naturally be incorporated as constraints. For instance, when direct cause and effect relations are available, we can use them as a basis for generating a draft in Phase I. In Phase II, the learning algorithm will try to add an arc only if it agrees with the domain knowledge. In Phase III, the algorithm will not try to remove an arc if that arc is required by domain experts. Partial node ordering, which specify the ordering of a subset of the node-pairs, can also be used to improve the learner’s efficiency. Each such pair declares which of the two nodes should appear earlier than the other in a correct ordering. Obviously, these relations can help us to orient some edges in the first and second phases so that the edge orientation procedure at the end of the third phase can be finished more quickly. These relations can also be used in several other parts of the algorithm to improve performance. For example, in *EdgeNeeded_H*, we need to find S_X and S_Y, which are the neighbor-sets of X and Y respectively. Since the procedure tries to separate the two nodes using only the parents of X (Y) in S_X (S_Y), if we know that some nodes in S_X (S_Y) that are actually the children of X (Y), we can remove them immediately without using any CI tests. This improves both the efficiency and accuracy of this procedure.

5.4.5 Improving the Efficiency

We have found that over 95% of the running time of the *TPDA* algorithms is consumed in database queries, which are required by the CI tests. Therefore, one obvious way to make the algorithm more efficient is to reduce the number of database queries. There are two ways to achieve this: by reducing the number of CI tests and by using one (more complex) database query to provide information for more than one CI test.

As noted above, we designed our algorithms to reduce the number of CI tests needed. However, there is a trade-off between using one query for more than one CI test and using one query for one CI test. While this latter method can reduce the total number of database queries, it also increases the overhead of the query and demands more memory. Our *TPDA* algorithm uses this method, but only in its first phase.

6 Empirical Study

This section demonstrates empirically that our *TPDA* and *TPDA-Π* algorithms work effectively, using three benchmark datasets and a group of datasets from randomly generated distributions. It also demonstrates that the heuristics used (to change from *SLA* to *TPDA*) do improve the performance. These tests were performed using our Bayesian network learning tool, *PowerConstructor*, which is described in Appendix C.

The three benchmark Bayesian networks are:

- ALARM: 37 nodes (each with 2-4 values); 46 arcs; 509 total parameters [Beinlich *et al.*, 1989]
- Hailfinder: 56 nodes (each with 2-11 values); 66 arcs; 2656 total parameters (http://www.sis.pitt.edu/~dsl/hailfinder/hailfinder25.dne)
- Chest-clinic: 8 nodes (each with 2 values); 8 arcs; 36 total parameters (http://www.norsys.com/netlib/Asia.dnet)

The first two networks are from moderate complex real-world domains and the third one is from a simple fictitious medical domain. In all cases, we generated synthesized data sets from their underlying probabilistic models using a *Monte Carlo* technique, called *probabilistic logic sampling* [Henrion, 1988].

Note that our PowerConstructor system has also been successfully applied in many real-world applications, both by ourselves and by other users who downloaded the system. However, since the underlying models of these real-world data sets are usually unknown, it is difficult to evaluate and analyze the learning accuracy. This is why almost all researchers in this area use synthesized data sets to evaluate their algorithms. We have also applied our system to learn predictive models (classifiers) from real-world data sets; in those cases we used performance (prediction accuracy) to evaluate our system. The results there were also very encouraging; see [Cheng and Greiner, 1999; Cheng and Greiner, 2001; Page and Hatzis, 2001].

Here, we evaluate each learned structure in two ways: First, based on the number of missing arcs and wrongly added arcs, as compared to the true structure. This measure is easy to determine; and clearly the score (0,0) (read “0 missing arcs and 0 wrongly added arcs”) means the learned structure is perfect. Of course, some arcs may be difficult, if not impossible, to find – eg, consider an arc between the binary variables *A* and *B*, when $P(B=1|A=1) = 0.3 = P(B=1|A=0)$. Here, this $A \rightarrow B$ arc is clearly superfluous. Similarly, it can be extremely difficult to detect an arc if the dependency is very slight – eg, if $P(B=1|A=1) = 0.3001$ and $P(B=1|A=0)= 0.3000$. Notice it will take thousands of instances to have a chance to see this very slight difference; moreover, a network that does *not* include this edge will only be slightly different to one that includes it. We therefore also report the mutual information associated with each missing link.

We also measure the number and “order” of the CI tests (ie, the cardinality of the conditioning set) that were performed; note that the number of “$p \ln p$” tests that are performed (2.1; 2.2) will be exponential in this quantity, as a *k*-ary CI test will involve $O(r^{k+2})$ such computations, for *r*-ary variables.

All the experiments in this paper were conducted on a Pentium II 300 MHz PC with 128 MB of RAM running under Windows NT 4.0. The data sets were stored in an MS-Access© database.

6.1 Experimental Results on the ALARM Network

ALARM, which stands for ’A Logical Alarm Reduction Mechanism’, is a medical diagnostic system for patient monitoring, which includes nodes for 8 diagnoses, 16 findings and 13 intermediate variables [Beinlich *et al.*, 1989]. Each variable has two to four possible values. The network structure is shown in Figure 13.

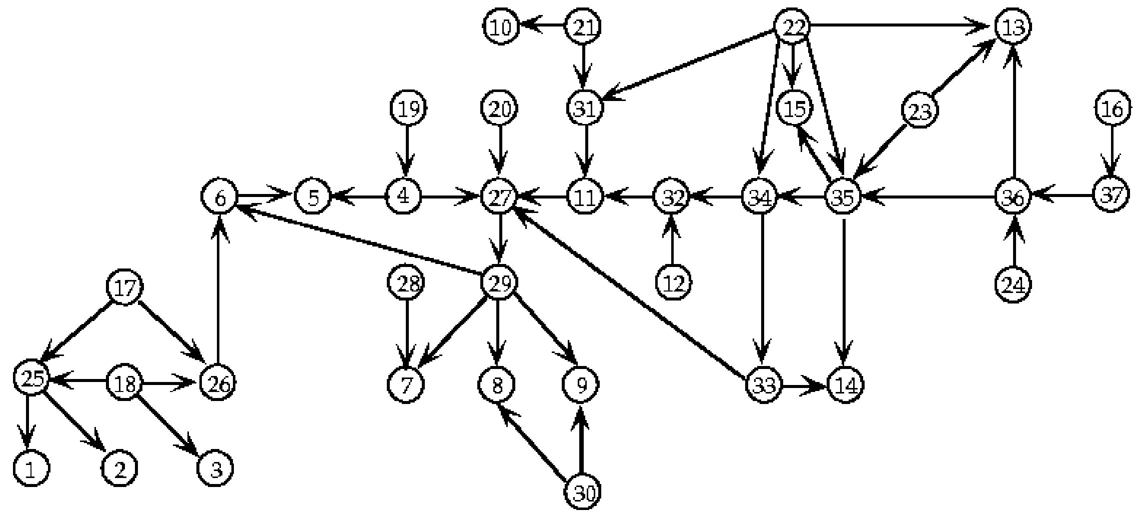

Figure 13 The ALARM network

The ALARM network is the most widely used benchmark in this area; many researchers have used this network to evaluate their algorithms.[11] Sections 6.1.1 and 6.1.2 provide detailed results on learning the ALARM network from a fixed number of cases (10,000) using *TPDA-Π* and *TPDA*, respectively. Section 6.1.3 gives our results on learning from different sample sizes of the ALARM network using both algorithms.

6.1.1 Use *TPDA-Π* to learn the ALARM Network Data

Here, we gave the *TPDA-Π* algorithm a set of 10,000 cases, drawn independently from the correct Alarm BN, as well as a correct ordering of the variables (as inferred from the structure). Table 3 summarizes our results, showing how each phase of *TPDA-Π* performed, in terms of the structure learned and the number of CI tests used. The CI tests are grouped by the cardinalities of their condition-sets.

---

[11] There are three different versions of it, which share the same structure but use slightly different CPtables. In this paper, we will focus on the probabilistic distribution described at the web site of Norsys Software Corporation http://www.norsys.com. We also tested our algorithms on the version presented in [Heckerman *et al.*, 1994] and the one in [Cooper and Herskovits, 1992], and obtained similar results; see [Cheng, 1998].

Phase	Results			No. of CI Tests (of each order)					
	Arcs	**M.A.**	**E.A.**	**0**	**1**	**2**	**3**	**4+**	**Total**
I	43	5	2	666	0	0	0	0	666
II	49	0	3(2+1)	0	116	54	22	10	202
III	45	1	0	0	12	1	1	3	17

M.A. = number of missing arcs; E.A. = number of extra arcs.

The CI tests are grouped by the cardinalities of their condition-sets.

Table 3: Results on the ALARM network (*TPDA-Π*)

Table 3 demonstrates that *TPDA-Π* can learn a nearly perfect structure (with only one missing arc) from 10,000 cases. The result after Phase I already resembles the true structure: this draft has 43 arcs, only 2 of which arcs are incorrect. The draft also missed 5 arcs of the true structure.

The result after the second phase (thickening) has 50 arcs, which continues to include all the arcs of Phase I as well as 4 out of 5 arcs that Phase I did not find. In addition, it also wrongly added another arc. This is understandable, as it is always possible to add some arcs wrongly before all the real arcs are discovered. In Phase III, *TPDA-Π* 'thinned' the structure successfully by removing all three previously wrongly added arcs. However, it also deleted a real arc 22→15 due to the fact that the connection between 22 and 15 is very weak given node 35 --- ie, $I(22, 15 | 35) = 0.0045$. The result after the third phase has 45 arcs, all of which belong to the true structure.

From the complexity analysis of Appendix A.4, we know that each of the three phases can require $O(N^2)$ CI tests, in the worst case. However, the number of CI tests used in Phase I is quite different from that of Phases II and III. This is because the ALARM network is a sparse network and Phases II and III only require a large number of CI tests when a network is densely connected, whereas Phase I always requires $O(N^2)$ CI tests. All our experiments on real-world data sets show similar patterns – i.e., most CI tests are used in Phase I. Table 3 also shows that most CI tests have small condition-sets. Here, the largest condition-set contains only five variables. (Notice no node in the true structure had more than 4 parents.)

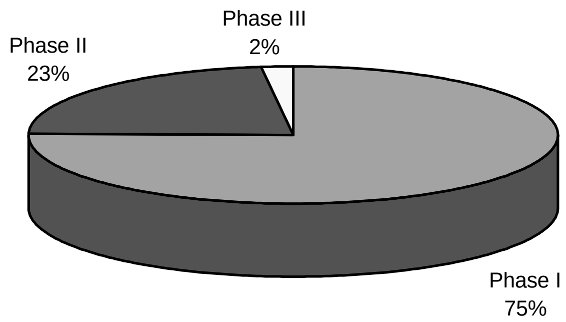

Figure 14: The number of CI tests used in each phase.

As we mentioned in Section 5.4.5, most of the running time is consumed in database queries while collecting information for CI tests. Therefore, we can improve the efficiency of our program by using high performance database query engines. To prove this, we moved this data set to an ODBC database server (SQL-server 6.5 running remotely under Windows NT Server 4) and repeated the experiment. Here we found that the experiment ran 13% faster. Note that our basic system was still running on the local PC as before.

6.1.2 Use TPDA to learn the ALARM Network Data

Phase	Results			No. of CI Tests (of each order)					
	Edges	M.A.	E.A.	0	1	2	3	4+	Total
I	36	12	2	666	0	0	0	0	666
II	49	2	2+3	0	127	61	22	7	217
III	44	2	0	0	86	6	8	3	103

M.A. = number of missing arcs; E.A. = number of extra arcs.

Table 4 Results on the ALARM network (TPDA)

We next consider *TPDA,* the version that does not have the node ordering. Table 4 shows that *TPDA* can get a very good result when using 10,000 cases; here only two missing edges. The result after Phase I (drafting) has 36 edges, among which 2 edges are wrongly added; the draft also missed 12 edges of the true structure. As *TPDA* does not know the node ordering, it is easy to understand why *TPDA*'s draft is significantly worse than *TPDA-Π*'s (Section 6.1.1), which only missed 5 edges.

The structure after *TPDA*'s second phase (thickening) has 49 edges, which includes all the arcs of Phase I and 10 out of 12 of the previously missing arcs. *TPDA* did not discover the other two edges (22-15, 33-27) as those two relationships are too weak: I(22,15|35)=0.0045; I(33,27|34,14)=0.0013. In addition, *TPDA* also wrongly added 3 edges. In Phase III, *TPDA* 'thinned' the structure successfully by removing all five wrongly added arcs. *TPDA* can also orient 40 of the 44 learned edges correctly. It cannot orient the other 4 edges due to the limitation of collider identification based method; of course, no other algorithm, given only this information, could do better. By comparing Table 3 to Table 4, we can see that the results of the three phases of *TPDA* are not as good as those of *TPDA-Π*, and Phase II and Phase III of *TPDA* require more CI tests than the two corresponding phases of *TPDA-Π*. This is not surprising since the *TPDA* does not have access to the node ordering, while *TPDA-Π* does.

From the complexity analysis of Appendix A.2, we know that the first phase requires $O(N^2)$ CI tests and the second and the third phases are of the complexity $O(N^4)$ and $O(N^5)$ respectively, in the worst case. However, since most real-world situations have sparse Bayesian networks, the numbers of CI tests used in the second and the third phases are usually much smaller than the number of CI tests used in the first phase, which is of complexity $O(N^2)$. As in Section 6.1.1, we use a pie chart (Figure 15) to show the percentages of the number of CI tests used.

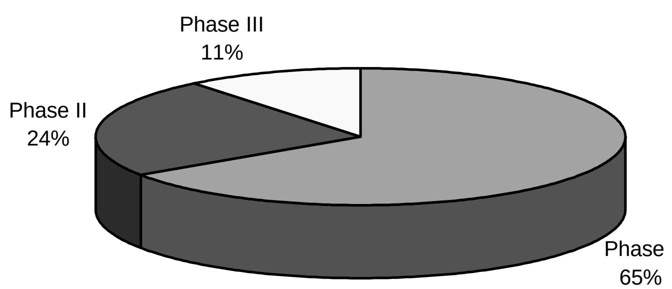

Figure 15 The number of CI tests used at each phase.

6.1.3 Experiments on Different Sample Sizes

In this section, we present our experimental results on 1,000, 3,000, 6,000 and 10,000 cases of the ALARM network data. The results are shown in the following table.

Cases	Ordering	Results				Time (Seconds)
		M.A	E.A.	M.O.	W.O.	
1,000	Yes	0	3	N/A	N/A	19
	No	3	2	3	2	19
3,000	Yes	1	3	N/A	N/A	43
	No	1	1	4	0	46
6,000	Yes	1	0	N/A	N/A	65
	No	2	0	4	0	75
10,000	Yes	1	0	N/A	N/A	100
	No	2	0	4	0	115

Table 5 Results on 1,000, 3,000, 6,000 and 10,000 cases

(M.O. and W.O. stand for missing orientation and wrongly oriented)

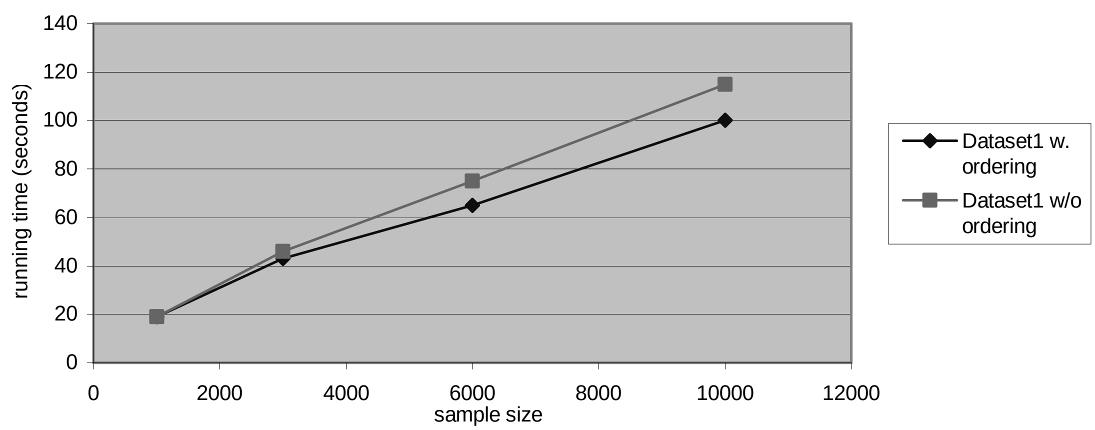

Figure 16 The relationship between the sample sizes and the running time

Figure 16 shows that the running time is roughly linear to the number of cases in the data set. This is what we expected, since most of the running time of the experiments is consumed by database queries; and response time of each database query is roughly linear to the number of records in the database table. The fact that the run-time increases so slowly suggests that our algorithms will be able to handle very large data sets. Table 5 also shows the learner typically produces more accurate networks, given larger data sets. Note the results on 3,000 cases are already quite acceptable; this suggests that our algorithms can give reliable results even when the data set is not large for its domain. This is because our algorithms can often avoid many high-order CI tests, which are unreliable when the data sets are not large enough.

6.1.4 Other Learning Algorithms

Many other learning algorithms have attempted to learn this network from a dataset (and sometimes, from a node ordering as well). The table below summarizes their performance. (Section 7 summarizes many of these learning systems.)

Algorithm	Node ordering	Sample size	Platform	Running time (Min.)	Results
K2	Yes	10,000	Macintosh II	17	1 M.A. 1 E.A.
Kutato	Yes	10,000	Macintosh II	1350	2 M.A. 2E.A.
Chu-Xiang	No. (Learn Markov net)	10,000	AVX-series2 12 processors	6	Unknown
Benedict	Yes	3,000	Unknown	10	4 M.A. 5 E.A.
CB	No	10,000	Dec Station 5000	7	2 E.A. 2 W.O.
Suzuki	Yes	1,000	Sun Sparc-2	306	5 M.A. 1 E.A.
Lam-Bacchus	No	10,000	Unknown	Unknown	3 M.A. 2 W.O.
Friedman-Goldszmidt	No	250-32,000	Unknown	Unknown	Only in term of scores
PC	No	10,000	Dec Station 3100	6	3 M.A. 2 E.A.
HGC	Using a prior net	10,000	PC	Unknown	2 E.A. 1 W.O.
Spirtes-Meek	No	10,000	Sun Sparc 20	150	1 M.A.
TPDA-Π	Yes	10,000	PC	2	1 M.A.
TPDA	No	10,000	PC	2	2 M.A.

Table 6 Experimental results of various algorithms on ALARM net data
(M.A., E.A. and W.O. stand for missing edges, extra edges and wrongly oriented edges.)

Note that the Chu-Xiang algorithm is used to learn a Markov network (undirected graph) so it does not need node ordering; and the HGC algorithm uses a prior network as domain knowledge rather than node ordering. When evaluating the results, Friedman and Goldszmidt use entropy distance rather than the direct comparison. By comparing the results of this table with our results, we can see that our results on the ALARM network data are among the best.

6.2 The Hailfinder Network

Hailfinder network is another real-world domain Bayesian belief network. It is a normative system that forecasts severe summer hail in northeastern Colorado. The network structure, shown in Figure 17, contains 56 nodes and 66 arcs. Each node (variable) has from two to eleven possible values. For detailed information about the Hailfinder project and various related documents, please visit the web page http://www.sis.pitt.edu/~dsl/hailfinder.

To evaluate our algorithms, we generated a data set of 20,000 cases from the underlying probabilistic model of Hailfinder network (version 2.5) using a probabilistic logic sampling method.

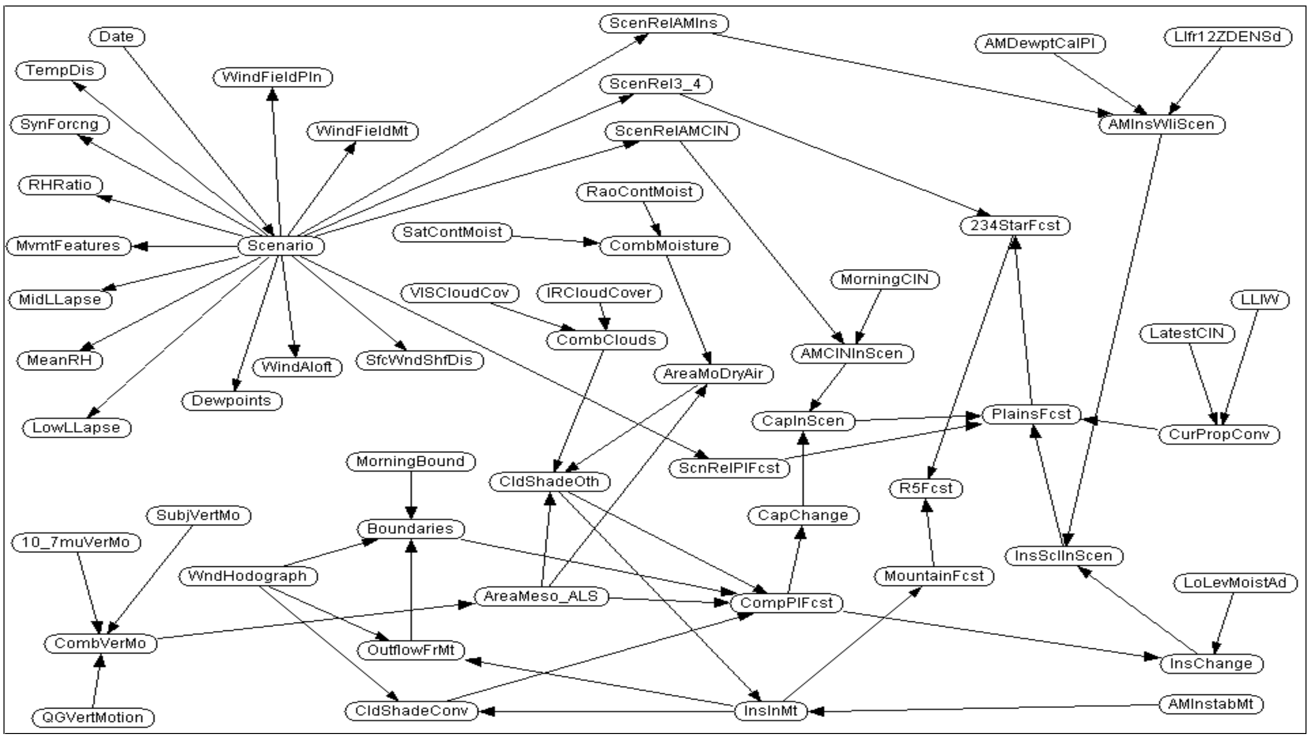

Figure 17 The HailFinder network

6.2.1 Experiments on 10,000 cases

In this section, we give the detailed experimental results using the first 10,000 cases of the Hailfinder data. The results are from two runs of our system, one with node ordering (*TPDA-Π*), and the other without node ordering (*TPDA*). The node ordering we used for Hailfinder network is the ordering described in the file http://www.sis.pitt.edu/~dsl/hailfinder/hailfinder25.dne.

Node ordering	No. of CI tests (of each order)					Results				Time (seconds)
	0	1	2	3+	Total	M.A.	E.A.	M.O.	W.O.	
Yes	1540	163	33	7	1743	3	0	N/A	N/A	227
No	1540	290	18	1	1849	4	1	1	5	245

Table 7 Running time and the CI tests used on 10,000 cases of Hailfinder data

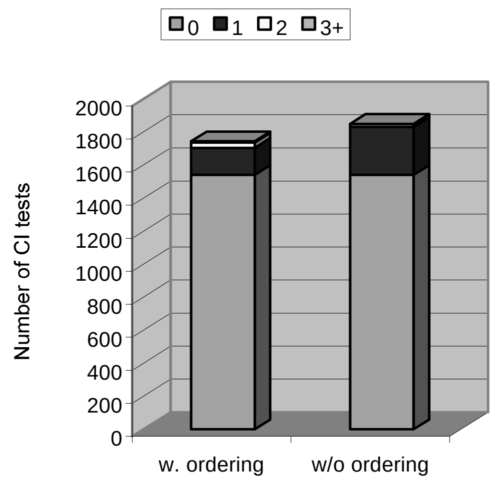

Figure 18 The bar charts of CI tests and running time.
(The number 0, 1, 2, 3+ in the bar chart represent the cardinalities of the condition-sets of CI tests.)

Table 7 shows that the running time is about 4 minutes and that the learned networks are very close to the underlying BN. Figure 18 shows that most CI tests are of the low orders. Now compare the number of CI tests used in both *TPDA* and *TPDA-Π* to learn the Hailfinder network versus learning the ALARM network. While in theory TPDA may require $O(N^5)$ CI tests, its actual speed appears similar to that of *TPDA-Π*, which is $O(N^2)$. This suggests that in real-world situation, where the underlying networks are sparse, the actual time complexity on CI tests is close to $O(N^2)$ even when node ordering is not given.

6.2.2 Experiments on Different Sample Sizes

In this section, we present our experimental results on 2,500, 5,000, 10,000, 20,000 cases of the Hailfinder data. The results are shown in the table below.

Cases	Ordering	Results				Time (Seconds)
		M.A.	E.A.	M.O.	W.O.	
2,500	Yes	2	6	N/A	N/A	132
	No	5	4	1	5	133
5,000	Yes	3	3	N/A	N/A	172
	No	3	2	0	2	174
10,000	Yes	3	0	N/A	N/A	227
	No	4	1	1	5	245
20,000	Yes	3	0	N/A	N/A	369
	No	4	1	1	5	403

Table 8 Results on different sample sizes of Hailfinder data

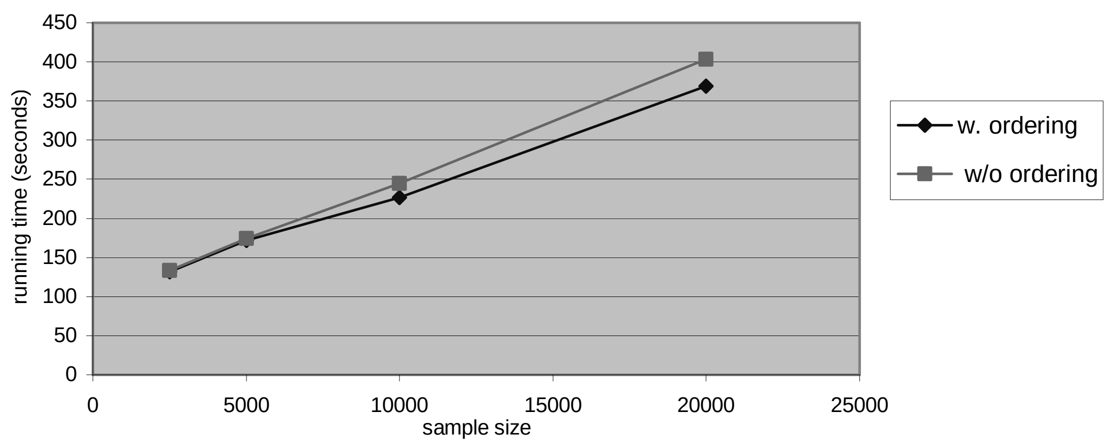

Figure 19 The relationship between the sample sizes and the running time

These results on Hailfinder data repeat the trends we found on ALARM data – that is, the growth of running time is roughly linear to the number of cases in the data set, and in general, the number of errors decreases as the sample size increases. Table 8 shows that we get the same results using 10,000 cases as we get using 20,000 cases; this suggests that 10,000 cases of Hailfinder data is already large enough for our algorithms.

6.3 The Chest-clinic Network

The Chest-clinic network (also known as the “Asia network”) is a very small Bayesian network for a fictitious medical domain, relating whether a patient has tuberculosis, lung cancer or bronchitis, to their X-ray, dyspnea, visit-to-Asia and smoking status. The structure of this network is shown in Figure 20, which contains 8 arcs connecting 8 nodes, each of which has exactly two possible values. The underlying probabilistic distribution of this network is described in http://www.norsys.com/netlib/Asia.dnet. We generated a data set of 1,000 cases using the probabilistic logic sampling method. We use this simple Bayesian network to show the performance of our algorithms on small domains.

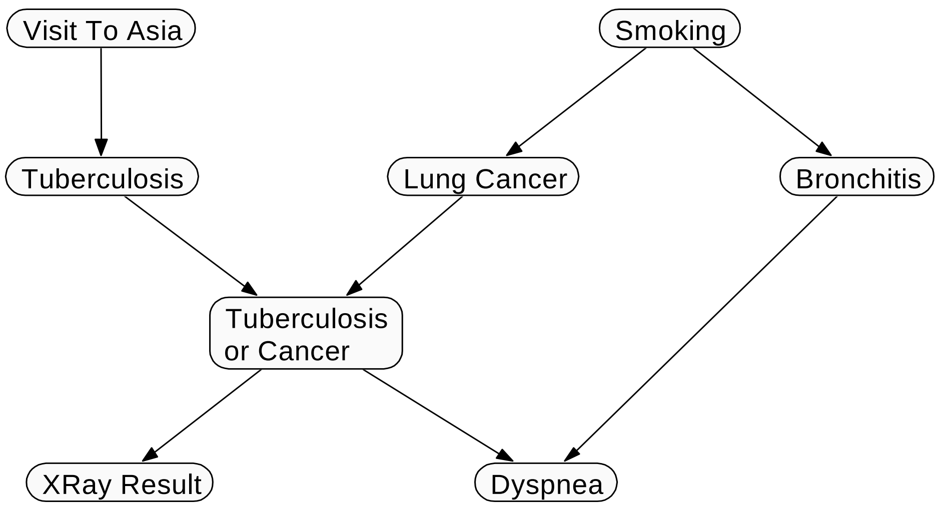

Figure 20 The Chest-clinic network

Our results are summarized in Table 9 below.

Node Ordering	Results				Time (seconds)
	M.A.	E.A.	M.O.	W.O.	
Yes	1	0	N/A	N/A	1
No	1	0	2	0	1

Table 9 Results on 1,000 cases of Chest-clinic data

The node ordering we use is [Visit to Asia, Tuberculosis, Smoking, Lung Cancer, Tuberculosis or Cancer, X-ray results, Bronchitis, Dyspnea]. In both experiments, the system could not find the arc from 'Visit to Asia' to 'Tuberculosis', as that dependency is extremely weak; *I(VisitToAsia, Tuberculosis) = 6.05E-5*. There are also two edges (Smoking → Lung Cancer and Smoking → Bronchitis) that *TPDA* cannot orient.

6.4 Simulation Tests Using Random DAGs

Our *TPDA* and *TPDA-Π* attempt to gain efficiency by minimizing the number of CI tests used. When the node ordering is given, *TPDA-Π* also minimizes the complexity of each CI test by using a small cut-set, so that the CI tests are reliable even when the sample size is small. However, when node ordering is not given, *TPDA* may use more complex CI tests. For example, when compared to the PC algorithm (which also uses dependency analysis to learn Bayesian net structures; [Spirtes and Glymour, 1991], see Section 7), we found that *TPDA* typically uses fewer CI tests than PC uses, but these tests are often more complex.

To investigate the reliability issue of *TPDA* when the data sets are noisy, we use simulation tests to compare the *TPDA* algorithm and the PC algorithm. The overall setting of this experiment is very similar to the one described in [Spirtes and Meek, 1995]. We randomly generated 10 DAGs with 10 nodes and 10 arcs, and another 10 DAGs with 10 nodes and 15 arcs. For each of the 20 DAGs, we randomly generated a single parameterization, using the Tetrad system [Scheines *et al.*, 1994]. From each of these 20 Bayesian networks, we created three data sets, of size 300, 1000 and 3000 respectively. This produced 60 data sets in total. As Spirtes and Meek (1995) did in their experiment, before measuring the performance, we also used preliminary tests to determine a reasonable threshold value for these artificial data sets. Here, we found $\varepsilon \approx 0.0025$ worked best.

The results appear in Table 10. We can see that the error rates here are higher than those in the experiments on benchmark data sets, shown in previous sessions. This is because there is more noise in these data sets, which make the learning task more difficult. The performances of the two algorithms on these data sets are quite similar. *TPDA* generates structures with more extra edges than PC especially when the sample size is small, which shows that minimizing the complexity of CI tests when data sets are small and noisy is a good strategy. It also seems that *TPDA* performs a little better than PC on orienting edges. Both algorithms are very efficient – it took less than a minute to learn all 60 structures from the data sets on our computer.

Algorithm	# of Edges in true DAG	Size	Missing Edges %	Extra Edges %	Missing Orientation %	Wrong Orientation %
PC	10	300	39.0	2.0	42.0	7.0
TPDA	10	300	32.0	27.0	33.0	7.0
PC	10	1,000	27.0	1.0	39.0	10.0
TPDA	10	1,000	23.0	9.0	36.0	5.0
PC	10	3,000	18.0	0.0	34.0	9.0
TPDA	10	3,000	23.0	1.0	37.0	8.0
PC	15	300	50.0	6.0	21.33	12.67
TPDA	15	300	38.0	12.67	15.33	16.0
PC	15	1,000	30.67	2.67	26.0	15.33
TPDA	15	1,000	25.33	6.67	10.67	19.33
PC	15	3,000	19.33	4.0	14.67	20.67
TPDA	15	3,000	24.0	4.0	9.33	19.33

Table 10 Simulation test results

6.5 How *TPDA*'s Heuristics Improve Its Efficiency

As noted above, *TPDA* differs from *SLA* by incorporating several heuristics, such as starting with a quickly-computed draft, and using the faster (but less accurate) *EdgeNeeded_H* before *EdgeNeeded*. While these heuristics are intuitive, they still do not *have* to work. To find out, we run some tests on ALARM data and HailFinder data using 10,000 data points, to see how *SLA* really compares with *TPDA*. We found that, in general, *TPDA* and *SLA* return the same answers, but *SLA* is about 2.5 times slower. *SLA* also requires 65% *more* CI tests, and many of the CI tests are of high order.

7 Related Work

In recent years, graphical probabilistic models, including Bayesian networks and Markov networks, have become very popular. *Learning* such graphical model has become a very active research topic and many algorithms have been developed for it. For survey papers and introductory papers on probabilistic network learning, please refer to [Buntine, 1996; Chrisman, 1996; Heckerman, 1995; Krause, 1996].

As noted above, we divide learning a BN into two subtasks: first learn the structure, and then find the parameters (read "CPtables") for that structure. We continue to focus on the first subtask. There are two ways to view a BN, each suggesting a particular approach to the structure learning. (1) A BN structure encodes the joint distribution of the attributes. This suggests that the best BN is the one that best fits the data, and leads to the *scoring-based* learning algorithms, which each seek a structure that maximizes the Bayesian, MDL or Kullback-Leibler (KL) entropy scoring function (Heckerman 1995; Cooper and Herskovits 1992). (2) Each arc in a BN structure specifies a dependency between the two associated nodes. This suggests learning structures that captures these dependencies; and more importantly, leaving unconnected nodes that are *independent* of each other. This leads to the "*dependency based*" methods – which include the *TPDA* and *TPDA-*Π algorithms presented here.

Heckerman *et al.* (1997) compare these two general approaches for learning BNs, and show that the scoring-based methods often have certain advantages over the dependency analysis based methods.[12] Recently, Cowell (2001) proves that for every scoring-based algorithm, there is an equivalent dependency based algorithm and vice versa. So the major difference between the two approaches is actually not the different measures used, but whether or not an algorithm utilizes the d-separation concept to constrain the model space. When the number of variables are large, the constraint based methods are usually much more efficient. However, when the sample size is small and the data is noisy, the scoring-based algorithms can often give more accurate results since they (potentially) search the whole model space to find the optimal model.

This section provides a synopsis of a few relevant BN-learning systems, providing any details only about the systems that are related to our *TPDA* system. Sections 7.1 and 7.2 introduce some

---

[12] Here we consider only the task of modeling a distribution. See Friedman *et al.* (1997), Cheng and Greiner (1999,2001) and Greiner *et al.* (1997) for a discussion of learning Bayesian-net based *classifiers.*

representative algorithms of each group, and Section 7.3 briefly introduces other related learning algorithms.

7.1 Search & scoring Based Methods

Algorithm	Resulting models	Node ordering required	Scoring method	Main features
Chow-Liu [Chow and Liu, 1968]	Trees (a special kind of Markov network)	No	Entropy	only needs $O(N^2)$ pair-wise dependency calculations
Rebane-Pearl [Rebane and Pearl, 1987]	Polytrees (a special kind of Bayesian network)	No	Entropy	only needs $O(N^2)$ pair-wise dependency calculations; can orient edges
K2 [Cooper and Herskovits, 1992]	General Bayesian nets	Yes	Bayesian	Efficient, uses heuristic search
HGC [Heckerman *et al.*, 1994]	General Bayesian nets	No (requires prior net)	Bayesian	Uses prior net as domain knowledge
Kutato [Herskovits and Cooper, 1991]	General Bayesian nets	Yes	Entropy	Uses CI tests to speed up entropy calculations
Wong-Xiang [Wong and Xiang, 1994]	General Markov nets	No	Entropy	The results are I-maps of the underlying models
BENEDICT [Acid and Campos, 1996b]	General Bayesian nets	Yes	Entropy	Heuristic search; uses the concept of d-separation
CB [Singh and Valtorta, 1995]	General Bayesian nets	No	Bayesian	Combines PC (see Section 2.2.2) and K2; can orient edges
Suzuki [Suzuki, 1996]	General Bayesian nets	Yes	MDL	Can learn the optimal structure but inefficient
Lam-Bacchus [Lam and Bacchus, 1994]	General Bayesian nets	No	MDL	Can orient edges using a pure search & scoring method
Friedman-Goldszmidt [Friedman and Goldszmidt, 1996]	General Bayesian nets	No	MDL or Bayesian	Can orient edges using a pure search & scoring method

Table 11 Summary of the search-&-scoring algorithms

Table 11 summarizes the representative search-&-scoring algorithms. The algorithm most related to our *TPDA* algorithm is presented below.

Chow-Liu Tree Construction Algorithm

We say a network is "tree structured" if it is connected and each node has at most one parent. Chow and Liu (1968) developed an algorithm for learning the *optimal* tree-structured BN; this system has had a far-reaching influence throughout the area of graphical model learning. It takes as input a probability distribution *P(x)* over *N* variables (which of course could be an empirical distribution), and returns as output a tree-structured BN, *P**, and does so in only $O(N^2)$ time. The authors prove that the resulting tree-shaped distribution P^* is the best tree-structured approximation of *P*, in that it has minimum KL-divergence (Kullback and Leibler, 1951), over all possible tree-structured distributions. This means, in particular, that when the underlying structure of distribution *P* is actually a tree, this algorithm is guaranteed to recover the true model.

This algorithm has characteristics of both learning approaches presented earlier: Although the general idea behind this algorithm is to find a structure with the best score (Kullback-Leibler (1951) cross-entropy), it does this by analyzing the pair-wise dependencies, which is the method used in the dependency analysis approach.

This algorithm requires only $O(N^2)$ pair-wise dependency calculations and each calculation uses only second-order statistics. Unfortunately, an equally efficient dependency-analysis algorithm is not possible for constructing multiply connected graphs, since larger condition-sets are required, which means higher order statistics must be used.

7.2 Dependency analysis Based Methods

Algorithm	Resulting models	Node ordering required?	Number of CI tests	Main features
Wermuth-Lauritzen [Wermuth and Lauritzen, 1983]	General Bayesian nets	Yes	$O(N^2)$	Only needs $O(N^2)$ CI tests but highly impracticable
Boundary DAG [Pearl, 1988]	General Bayesian nets	Yes	Exponential	A simple algorithm
SRA [Srinivas *et al.*, 1990]	General Bayesian nets	Partial ordering	Exponential	Extension of Boundary DAG; only needs partial ordering; uses heuristic search
Constructor [Fung and Crawford, 1990]	General Markov nets	No	Exponential	Uses cross-validation technique to avoid over-fitting
SGS [Spirtes *et al.*, 1990]	General Bayesian nets	No	Exponential	Can orient edges
Verma-Pearl [Verma and Pearl, 1992]	General Bayesian nets	No	Exponential	A variation of SGS; can orient edges and detect conflicts in the edge orientations
PC [Spirtes and Glymour, 1991]	General Bayesian nets	No	$O(N^{k+2})$	K is the maximum degree of any node in the true structure; Can orient edges; enhanced from SGS algorithm; efficient

Table 12 Summary of the dependency analysis based algorithms

7.3 Other Algorithms

Several researchers use *model averaging* techniques, which we view as a variant of the search & scoring based approach. They argue that sometimes the data does not identify the underlying model of a data set. Therefore, instead of searching for a single best solution, their algorithms [Buntine, 1994; Madigan and Raftery, 1994; Madigan *et al.*, 1994] return several networks and use the 'average' of these networks to perform belief propagation.

All of the above algorithms assume that the data sets are causally sufficient – i.e., all the variables in the underlying models appear in the data sets. Sometimes, the values of some variables are never in the data sets, we call them *hidden variables* or *latent variables*. There has been a lot of progress in learning Bayesian networks with hidden variables; see [Spirtes *et al.*, 1993; Spirtes *et al.*, 1997;Verma and Pearl, 1990].

There are also algorithms that can handle data sets with missing values --- that is, some values of some variables are excluded --- see *Bound and Collapse* [Ramoni and Sebastiani, 1996; Ramoni and Sebastiani, 1997], [Friedman, 1998] and [Singh, 1997].

8 Future Work and Conclusion

8.1 Future Work

We plan to work in the following directions.

1. Each of the two general approaches to Bayesian network learning (i.e., based on score-&-search and on dependency-analysis) has its own advantages. We plan to explore ways to combine the two approaches, especially for the task of learning models from data with hidden variables.
2. *TPDA* is correct for monotone DAG-faithful models. We conjecture that the monotone DAG-faithful assumption is only slightly stronger than DAG-faithfulness, and that most DAG-faithful models are also monotone DAG-faithful. We plan to explore the properties of monotone DAG-faithful models and compare them with those of DAG-faithful models.
3. We noted that *TDPA* spends most of its running time performing CI tests, and that most running time of these CI tests is in turn consumed by database queries. This suggests that we can improve the efficiency of our algorithms by improving the efficiency of database queries. One method is to move the data set to a high performance database server. We believe this will speed up the Bayesian network learning by a large factor --- perhaps even several hundred times, depending on the speed of the database server. Another method is to use database engines that are specially designed for such queries, i.e., capable of quickly counting the number of records that satisfy certain criteria.
4. We are already beginning to explore the use of these constraint-based techniques in the context of learning *classifiers* --- that is, performance systems that assign labels to (unlabeled) instances [Cheng and Greiner, 1999]. Here the goal is a Bayesian net that produces the correct label as often as possible; a goal that differs from finding the best "model" of the underlying

distribution. We plan to continue seeking ways to modify our basic *TPDA* algorithm for this specialized task.

8.2 Conclusion

This paper addresses the task of learning Bayesian networks from data. We develop two related information theoretic algorithms: *TPDA*, which learns Bayesian networks when node ordering is not given, and *TPDA-Π*, which deals with the special case where the node ordering is given. These two algorithms have been implemented within a general Bayesian network learning system – *BN PowerConstructor*. Using the PowerConstructor system, we have empirically evaluated our algorithms using two moderately complex real-world examples and a class of simpler synthetic ones. These results show that our algorithms are accurate and efficient.

Our algorithms improve on the other dependency-analysis based algorithms by using *quantitative* information from CI tests to avoid the need to perform an exponential number of such CI tests; *n.b.*, our algorithms are still guaranteed to recover the correct distribution when the underlying model of the data set is monotone DAG-faithful (given enough data and other simple assumptions). When the correct node ordering is available, our *TPDA-Π* algorithm requires only standard DAG-faithfulness, and uses only $O(N^2)$ CI test to learn an N-node BN, as it needs to perform only $O(1)$ CI test to decide whether to include each of the $O(N^2)$ possible arcs. Moreover, thanks to its three-phase mechanism, which allows some wrong decisions to be made in first two phases, this overall algorithm is even more efficient in practice. (By contrast, most other dependency-analysis based learning algorithms require an exponential number of CI tests for each decision.)

When node ordering is not given, it is impossible to make each "is an edge needed?" decision using only one CI test. However, by using *quantitative* CI tests, which tells us not only whether a pair of nodes are dependent or not but also how close their relationship is, *TPDA* need to use only $O(N^2)$ CI tests for each decision. We prove that *TPDA* is correct when the underlying model is monotone DAG-faithful.

Experimental results show that our algorithms are capable of handling large real-world data sets since the running time is linear in the number of records in the data set and polynomial in the number N of attributes in the data set --- empirically $O(N^2)$ for sparse networks. The results also show that our algorithms are quite reliable since the accuracy of the result does not deteriorate very fast when the sample size decreases.

7 Acknowledgements

We gratefully acknowledge the financial support of NSERC, PIMS, and Siemens Corporate Research, for helping to sponsor this work. The first author would also like to thank Thomas Richardson of the University of Washington, for his valuable comments, which helped to improve the TPDA algorithm; to Clark Glymour, Peter Spirtes and Richard Scheines of Carnegie Mellon University, for their encouragement and comments.

References

[Acid and Campos, 1996a] Acid, S. and Campos, L.M., An algorithm for finding minimum d-Separating sets in belief networks, *Proceedings of the twelfth Conference of Uncertainty in Artificial Intelligence*, 1996.

[Acid and Campos, 1996b] Acid, S. and Campos, L.M., BENEDICT: An algorithm for learning probabilistic belief networks, *Proceedings of the sixth International Conference IPMU'96*, 1996.

[Agresti, 1990] Agresti, A., *Categorical data Analysis*, John Wiley & Sons, 1990.

[Badsberg, 1992] Badsberg, J., Model search in contingency tables in CoCo, *Computational Statistics*, Dodge, Y and Wittaker, J. (Ed.), Heidelberg: Physica Verlag, page 251-256, 1992.

[Baker and Boult, 1990] Baker, M. and Boult T.E., Pruning Bayesian networks for efficient computation, *Proceedings of the sixth international conference on uncertainty in artificial intelligence*, 1990.

[Becker and Geiger, 1996] Becker, A. and Geiger, D., A sufficiently fast algorithm for finding close to optimal junction trees, *Proceedings of the twelfth international conference on uncertainty in artificial intelligence*, 1996.

[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.

[Bouckaert, 1994] Bouckaert, R.R., Properties of Bayesian belief network learning algorithms. Mantaras, R. and Poole, D. (Ed.), Morgan Kaufmann, 1994.

[Buntine, 1994] Buntine, W., Operations for learning with graphical models, *Journal of Artificial Intellignece Research*, 2, page 159-225, 1994.

[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.*, 1997a] 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.

[Cheng *et. al.*, 1997b] Cheng, J., Bell, D.A., and Liu, W., Learning belief networks from data: An information theory based approach, *Proceeding of the sixth ACM International Conference on Information and Knowledge Management*, 1997.

[Cheng, 1998] Cheng, J., Learning Bayesian networks from data: An information theory based approach, *Doctoral Dissertation*, Faculty of Informatics, University of Ulster, U.K., 1998.

[Cheng and Greiner, 1999] Cheng, J and Greiner, R., Comparing Bayesian Network Classifiers, *Proceedings of the fifteenth international conference on uncertainty in artificial intelligence*, 1999.

[Cheng and Greiner, 2001] Cheng, J. and Greiner, R., Learning Bayesian Belief Network Classifiers: Algorithms and System. *Proceedings of 14th Biennial conference of the Canadian society for computational studies of intelligence*, 2001.

[Chickering *et. al*., 1994] Chickering, D.M., Geiger, D., and Heckerman, D., Learning Bayesian Networks is NP-Hard, *Technical Report MSR-TR-94-17*, Microsoft Research, Microsoft Corporation, 1994.

[Chickering, 1996] Chickering, D. M., Learning equivalence classes of Bayesian network structures. *Proceedings of the twelfth conference on uncertainty in artificial intelligence*, 1996.

[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.

[Chrisman, 1996] Chrisman, L., A roadmap to Research on Bayesian networks and other decomposable probabilistic models, technical report, School of Computer Science, CMU, 1996.

[Chu and Xiang, 1997] Chu, T. and Xiang, Y., Exploring parallelism in learning belief networks, *Proceedings of the thirteenth international conference on uncertainty in artificial intelligence*, 1997.

[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.

[Cowell, 2001] Cowell, R.G., When learning Bayesian networks from data, using conditional independence tests is equivalent to a local scoring metric, *Proceedings of the seventeenth international conference on uncertainty in artificial intelligence*, 2001.

[Darken, 1999] Darken, C., Personal communication.

[Darroch *et al.*, 1980] Darroch, J.N., Lauritzen, S.L. and Speed, T.P. Markov fields and loglinear interaction models for contingency tables. *Ann. Stat.*, 8, 522-539, 1980.

[Dash and Druzdzel, 1999] Dash, D. and Druzdzel, M., A hybrid anytime algorithm for the construction of causal models from sparse data, *Proceedings of the fifteenth international conference on uncertainty in artificial intelligence*, 1999.

[Dietterich, 1997] Dietterich, T.G., Machine learning research: four current directions, AI Magazine, 18 (4), 1997.

[Edwards, 1995] Edwards, D., *Introduction to Graphical Modelling,* Springer-Verlag, New York, 1995.

[Friedman, 1998] Friedman, N. The Bayesian Structural EM Algorithm, Proceedings of the fourteenth international conference on uncertainty in artificial intelligence, 1998.

[Friedman and Goldszmidt, 1996] Friedman, N. and Goldszmidt, M., Learning Bayesian networks with local structure, *Proceedings of the twelfth international conference on uncertainty in artificial intelligence*, 1996.

[Fung and Crawford, 1990] Fung, R.M. and Crawford, S.L., Constructor: a system for the induction of probabilistic models, *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1990.

[Geiger and Heckerman, 1990] Geiger, D. and Heckerman, D., Separable and transitive graphoids, *Proceedings of the sixth international conference on uncertainty in artificial intelligence*, 1990.

[Geiger and Pearl, 1988] Geiger, D. and Pearl, J., Logical and algorithmic properties of conditional independence, Technical Report R-97, Cognitive Systems Laboratory, UCLA, 1988.

[Glymour *et al.*, 1997] Glymour, C., Madigan, D., Pregibon, D.and Smyth, P., Statistical themes and lessons for data mining, *Data mining and knowledge discovery*,1, 11-28, 1997.

[Greiner *et al.*, 1997] Greiner, R., Grove, A., and Schuurmans, D., Learning Bayesian Nets that perform well, *Proceedings of the thirteenth international conference on uncertainty in artificial intelligence*, 1996. 198-207.

[Greiner *et al.*, 2001] Greiner, R., Darken, C., and Santoso, N. I., Efficient Reasoning, *Computing Surveys*, 33:1 (March 2001), p. 1—30.

[Heckerman, 1988] Heckerman, D., An empirical comparison of three inference methods, *Uncertainty in Artificial Intelligence 4*, Shachter *et al.* (Ed.), page 283-302, North-Holland, 1988.

[Heckerman, 1995] Heckerman, D., A tutorial on learning Bayesian networks, *Technical Report MSR-TR-95-06*, Microsoft Research, 1995.

[Heckerman *et al.*, 1994] Heckerman, D., Geiger, D. and Chickering, D.M., Learning Bayesian networks: the combination of knowledge and statistical data, *Machine Learning Journal*, 20(3), 1997.

[Henrion, 1988] Henrion, M., Propagating uncertainty in Bayesian networks by probabilistic logic sampling, *Uncertainty in Artificial Intelligence 2*, page 149-163, North-Holland, 1988.

[Henrion and Cooley, 1987] Henrion, M. and Cooley, D.R., An experimental comparison of knowledge engineering for expert systems and for decision analysis, *Proceedings AAAI-87 Sixth National Conference on Artificial Intelligence*, page 471-476, Morgan Kaufmann, 1987

[Herskovits and Cooper, 1990] Herskovits, E. and Cooper, G., Kutato: An entropy-driven system for construction of probabilistic expert systems from databases, *Proceedings of the sixth international conference on uncertainty in artificial intelligence*, 1990.

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

[Hojsgaard *et. al.*, 1994] Hojsgaard, S., Skjoth, F. and Thiesson, B., User's guide to BIOFROST, *Technical report*, Department of Mathematics and Computer Science, Aalborg, Denmark.

[Jordan *et al.*, 1997] Jordan, M. I., Ghahramani, Z., Jaakkola, T. S. and Saul, L. K., An introduction to variational methods for graphical models, *Learning in graphical models*, Jordan, M. I. (Ed.), Kluwer Academic, 1997.

[Kjarulff, 1994] Kjarulff, U., Reduction of computational complexity in Bayesian networks through removal of weak dependences, *Proceedings of the tenth international conference on uncertainty in artificial intelligence*, 1994.

[Krause, 1996] Krause, P., Learning probabilistic networks, *Technical Report*, Philips research laboratories, UK, 1996.

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R., On information and sufficiency, Ann. Math. Stat., Vol 22 (76—86), 1951.

[Lam and Bacchus, 1994] Lam, W. and Bacchus, F., Learning Bayesian belief networks: An approach based on the MDL principle, Computational Intelligence, Vol 10:4, 1994.

[Lauritzen and Spiegelhalter, 1988] Lauritzen, S.L. and Spiegelhalter, D.J., Local computations with probabilities on graphical structures and their application to expert systems, *Royal Statistics Society B*, 50-2, 157-194, 1988.

[Madigan and Raftery, 1994] Madigan, D. and Raftery, A.E., Model selection and accounting for model uncertainty in graphical models using Occam's window, *Journal of the American Statistical Association*, 89, page 1535-1546, 1994.

[Madigan *et al.*, 1994] Madigan, D., Raftery, A.E., York, J.C., Bradshaw, J.M., and Almond R.G., Strategies for graphical model selection, *Selecting Models from Data: Artificial Intelligence and Statistics IV*, Cheeseman, P. and Oldford, R.W. (Ed.), Springer-Verlag, 1994.

[Meek, 1995a] Meek, C., Causal inference and causal explanation with background knowledge, *Proceedings of the eleventh international conference on uncertainty in artificial intelligence*, 1995.

[Meek, 1995b] Meek, C., Strong completeness and faithfulness in Bayesian networks, *Proceedings of the eleventh international conference on uncertainty in artificial intelligence*, 1995.

[Morjaia *et al.*, 1993] Morjaia, M.A., Rink, F.J., Smith, W.D., Klempner, J., Burns, C. and Stein, J., Commercialization of EPRI's Generator Expert Monitoring System (GEMS), *Expert System Application for the Electric Power Industry*, EPRI, 1993.

[Musick, 1988] Musick, C. R., Belief network induction, *doctoral dissertation*, UC Berkeley, 1988.

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

[Page and Hatzis, 2001] Page, D. and Hatzis, C., ACM SIGKDD Cup 2001, http://www.cs.wisc.edu/~dpage/kddcup2001/, 2001.

[Pearl, 1988] Pearl, J., *Probabilistic reasoning in intelligent systems: networks of plausible inference*, Morgan Kaufmann, 1988.

[Pearl and Wermuth, 1993] Pearl, J. and Wermuth, N., When can association graphs admit a causal interpretation? *Technical report*, 1993.

[Ramoni and Sebastiani, 1996] Ramoni, M. and Sebastiani, P., Robust learing with missing data, *Technical report*, KMI-TR-28, 1996.

[Ramoni and Sebastiani, 1997] Ramoni, M. and Sebastiani, P., Discovering Bayesian networks in incomplete databases, Technical report KMI-TR-46, Knowledge Media Institute, The Open University, March 1997.

[Russell and Norvig, 1995] Russell, S. and Norvig, P., *Artificial Intelligence: A Modern Approach*, Prentice Hall, 1995.

[Sarkar and Murthy, 1996] Sarkar, S. and Murthy, I., Constructing efficient belief network structures with expert provided information, *IEEE Transactions on knowledge and data engineering*, 8-1, 1996.

[Sarkar *et al.*, 1996] Sarkar, S., Sriram, R.S., Joykutty, S. and Murhty, I., An information theoretic technique to design belief network based expert systems, Decision support systems 17, page 13-30, 1996.

[Scheines *et al.*, 1994] Scheines, R., Spirtes, P., Glymour, C., and Meek, C., *TETRAD II: Tools for Discovery*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1994.

[Singh, 1997] Singh, M. Learning Bayesian networks from incomplete data, *Proceedings of AAAI-97.*

[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.

[Spiegelhalter *et al.*, 1993] Spiegelhalter, D., Dawid, P., Lauritzen, S., and Cowell, R., Bayesian analysis in expert systems, *Statistical Science*, 8:219-282, 1993.

[Spirtes *et al.*, 1990] Spirtes, P., Glymour, C. and Scheines, R., causality from probability, *Proceedings of Advanced Computing for the Social Sciences*, Williamsburgh, VA.

[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.

[Spirtes *et al.*, 1993] Spirtes, P., Glymour, C. and Scheines, R., *Causation, Prediction, and Search*, Springer Lecture Notes in Statistics, 1993.

[Sprites and Meek, 1995] Spirtes, P. and Meek, C., Learning Bayesian Networks with Discrete Variables from data, *Proceedings of the first international conference on knowledge discovery and data mining*, 1995.

[Spirtes *et al.*, 1997] Spirtes, P., Richardson, T., and Meek, C., Heuristic greedy search algorithms for latent variable models, *Proceedings of AI & STAT'97* (pp.481-488), Ft. Lauderdale, Florida, 1997.

[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*, Bari, Italy, 1996.

[Thomas *et al.*, 1992] Thomas, A., Spiegelhalter, D.J., and Gilks, W.R., BUGS: A program to perform Bayesian inference using Gibbs sampling, Bayesian Statistics 4, Bernardo, J.M., Berger, J.O., Dawid, A.P. and Smith A.F. (eds), Oxford University Press, page 837-842, 1992.

[Verma and Pearl, 1990] Verma, T.S. and Pearl, J., Equivalence and synthesis of causal models, *Proceedings of the sixth international conference on uncertainty in artificial intelligence*, 1990.

[Verma and Pearl, 1992] Verma, T.S. and Pearl, J., An angorithm for deciding if a set of observed independencies has a causal explanation, *Proceedings of the Eighth international conference on uncertainty in artificial intelligence*, 1992.

[Walsh, 1990] Walsh, A., *Statistics for the social sciences*, Harper & Row, 1990

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

[Whittaker, 1989] Whittaker, J., *Graphical Models in Applied Multivariate Statistics*, John Wiley & Sons, 1989.

[Wong and Xiang, 1994] Wong, S.K.M. and Xiang, Y., Construction of a Markov network from data for probabilistic inference, *Third International Workshop on Rough Sets and Soft Computing*, pages 562-569, San Jose, CA, 1994.

[Xiang and Wong, 1994] Xiang, Y. and Wong, S.K.M., Learning conditional independence relations from a probabilistic model, *Technical report*, University of Regina, 1994.

Appendix A: Proofs

A.1 Correctness Proof of *SLA* (and *TPDA*)

Theorem 3: Given a "sufficiently large" database of complete instances that are drawn, iid, from a monotone DAG-faithful probability model, the *SLA* algorithm will recover the correct underlying essential network. Moreover, this algorithm requires only $O(N^4)$ conditional independence tests. []

We use the following propositions to prove that *SLA* is correct (i.e., the first part of Theorem 3). Throughout we will assume the assumptions stated in the theorem, and also that M is the true model of the distribution.

Proposition 3.1 The graph generated after Step2, G2, contains all the edges of M.

Proof: This step considers all the edges between any two nodes that are not independent. An edge is not added only if the two nodes are separated by a set of other nodes. Hence, any two nodes that are not directly connected in G2 are conditionally independent in M. []

Lemma 3.1 As *EdgeNeeded* starts with the initial condition-set $S_X \cup S_X'$, it can close all the paths of the underlying model M between nodes X and Y except the paths connecting X and Y by one collider.

Proof: By using, say, $S_X \cup S_X'$ as the condition-set, we instantiate the nodes in S_X (the neighbors of X on the paths between X and Y) and S_X' (the neighbors of nodes of S_X that are on the paths between X and Y).[13] Therefore, we can instantiate at least two consecutive nodes of any path that has length equal to or larger than three. Because two consecutive nodes of a path cannot both be colliders in 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 X and Y that have length equal to or larger than three. Hence, the only paths that could remain open are those connecting X and Y by one collider. []

Lemma 3.2 *EdgeNeeded* does not open any previously closed X-Y path by removing a node from condition-set C.

---

[13] The same claim holds for $S_Y \cup S_Y'$.

Proof: Given monotone DAG-faithfulness, *EdgeNeeded* will not remove a node that only opens some paths, as such a removal would necessarily increase the mutual information. We therefore need only prove that removing a node from C cannot simultaneously open some paths and close others. From Lemma 3.1 we know that initially the only open paths are those connecting X and Y by one collider. Now consider each node *v* in C. If *v* is not a child-node of both X and Y or a descendent of such a child-node, removing it may open some paths but cannot close the paths connecting X and Y by a collider. So suppose that *v* is a child-node of both X and Y or a descendent of such a child-node. Now if one of *v*'s descendents is in C, then removing *v* cannot close the path connecting X and Y by the child-node. If none of *v*'s descendents is in C, removing *v* may close the path connecting X and Y 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 *v*'s descendents is in C, such a path cannot be opened. []

Lemma 3.3 *EdgeNeeded* can remove all the descendents of both X and Y from condition-set C.

Proof: Toward a contradiction, suppose S is the subset of set C containing the descendents of both X and Y that cannot be removed. Then there must be a node $v \in S$ that is not an ancestor of any other nodes in S. The only reason we would not consider removing *v* is if removing it increases mutual information. From the assumption of monotone DAG-faithfulness and Lemma 3.2, we know that removing *v* will open at least one path. Therefore, node *v* is not a collider in such a path and so this path must go through at least one descendent of *v*. Because a descendent of *v* is also a descendent of both X and Y, there must exist at least one descendent of *v* which is a collider in such open path. To make such path open, this collider has to be in S. This contradicts our assumption that *v* is not an ancestor of any other nodes in S. []

Proposition 3.2 Given that graph G contains all the edges of a probabilistic model M, if two nodes X and Y are independent in M, *EdgeNeeded* can always separate them in G.

Proof: From Lemma 3.1 we know that initially the only open paths are those connecting node X and Y by one collider. From Lemmas 3.2 and 3.3, we know that the procedure does not open any path when removing nodes from the condition-set C and that it removes all the descendents of both

X and Y that are in C. Therefore, if nodes X and Y are independent in M, *EdgeNeeded* can separate them by closing all the open paths. []

Proposition 3.3 The graph generated after step 3, G3, contains the exact same edges as those of M.

Proof: Since G2 contains all the edges of M, and an edge is removed in step 3 only if the pair of nodes is conditionally independent in M, G3 also contains all the edges of M. From Proposition 2, we also know that if two nodes are independent in M, our algorithm can always separate them in G3. Hence, G3 contains exactly the same edges as those of M. []

Proposition 3.4 Given that graph G contains the exact same edges as those of the underlying model M, all the colliders that can be identified by *OrientEdges*(G) are the real colliders of M.

Proof: For any structure X-Y-Z where X and Z are not directly connected, *OrientEdges*(G) uses step 1 to check if Y is a collider on the path X-Y-Z. From Lemma 3.3 we know that the final cut-set between X and Z will never include Y if Y is a collider. So step 1 can identify a collider correctly. Since there are no extra-edges in G, step 2 of this procedure can never orient an edge wrongly. It is also easy to see that the inference of step 3 of the procedure is correct. []

A.2 Complexity Analysis for *SLA* (and for *TPDA*)

This appendix provides the worst-case time complexity of *TPDA* in terms of the number of CI tests. Please note that each CI test can require a large number of basic calculations – in fact, a number exponential in the size of the condition-set. However, the number of basic calculations is not a good index for comparing different algorithms because all algorithms are exponential in this sense. In practice, most of the running time of our algorithms is consumed in data queries from databases, which we have found often takes more than 95% of running time (see Section 6). Because the number of CI tests is directly related to the number of database queries, it is a relatively good criterion for judging an algorithm's performance. In fact, the number of CI tests is a widely used index for comparing different algorithms that are based on dependency analysis (Spirtes *et al.* 1993; Pearl 1988).

To prove that *TPDA-Π* requires $O(N^4)$ CI tests: Observe first that *EdgeNeeded* requires $O(N^2)$ CI tests, as it must, in the worst case, successively consider the one conditioning set of size N-2, then the N-2 possible subsets of this set of size N-3, then the N-3 size-N-4 subsets, and so forth, until considering 2 sets of size 1. This would require $\sum_{i=2..N-2} i = O(N^2)$ CI tests. Next note that Step 1 can call *EdgeNeeded* on at most every pair of nodes, as can Step 2; hence these steps require $O(N^2 * N^2)$ CI tests. Hence the final step requires $O(N^2 \times N^2) = O(N^4)$ CI tests, which is high-water complexity of this algorithm.

A.3 Correctness Proofs of *SLA-Π* (and *TPDA-Π*)

Theorem 2: Given a "sufficiently large" database of complete instances that are drawn, iid, from a DAG-faithful probability model, together with a correct node ordering, then the *SLA-Π* algorithm will recover the correct underlying network. Moreover, this algorithm will require only $O(N^2)$ CI tests. []

We use the following claims:

Proposition 2.1: The graph generated after Step2, G2, is an I-map of M.

Proof: As the first part of *SLA-Π* considers every pair of nodes that are not pair-wise independent, the only way that an arc can be excluded from the graph is if the two nodes of the arc are independent conditional on some appropriate condition-set. Hence, any two disconnected nodes in G2 are conditionally independent in M. []

Proposition 2.2: The graph generated after Step 3, G3, is a perfect map of M.

Proof: Because G2 is an I-map of M, and an arc is removed in Step 3 only if the pair of nodes is conditionally independent, it is easy to see that G3 and all the intermediate graphs of Step3 are I-maps of M. Now, we will prove that G3 is also a D-map of M. Suppose G3 is not a D-map, then there must exist an arc $\langle a, b \rangle$ which is in G3 and for which the two nodes *a* and *b* are actually independent in the underlying model M. Therefore, *a* and *b* can be d-separated by blocking all the real open paths **Pr** in M. In *SLA-Π,* the nodes *a* and *b* are connected in G3 only if *a* and *b* are still dependent after blocking all the open paths **P** in G3. Since all the intermediate graphs of Step3 are I-maps of M, **P** includes **Pr** and possibly some pseudo-paths. So blocking all the open paths in **P**

will block all the real open paths in **Pr**. Because information cannot pass along the pseudo-paths, the only reason for *a* and *b* to be dependent in G3 is that information can go through the paths in **Pr**. This contradicts our assumption that *a* and *b* are d-separated by blocking all the open paths of **Pr** in M. Thus, G3 is both a D-map and I-map and hence is a perfect map of M. []

The above propositions ensure that our algorithm can construct the perfect map of the underlying dependency model, i.e., the induced Bayesian networks are exactly the same as the real underlying probabilistic models of the data sets.

A.4 Complexity Analysis for *SLA-Π*

Since Step1 computes mutual information between any two nodes, it needs $O(N^2)$ mutual information computations. In Step2, the algorithm checks if it should add arcs to the graph. Each such decision requires one CI test. Therefore, Phase II needs at most $O(N^2)$ CI tests. In Step3, the algorithm sees if it can remove the arcs from the graph. Again, each such decision requires one CI test and so at most $O(N^2)$ CI tests are needed. Hence, the overall algorithm requires $O(N^2)$ CI tests in the worst case.

Appendix B: Monotone DAG-Faithfulness

In real world situations most DAG-faithful models are also monotone DAG-faithful. We conjecture that the violations of monotone DAG-faithfulness only happen when the probability distributions are ‘near’ the violations of DAG-faithfulness. In such situations, other algorithms also have difficulties in generating the true underlying model.

While *TPDA-Π* and other dependency analysis based algorithms require the assumption of DAG-faithfulness for its correctness proof, *TPDA* is only guaranteed to work correctly if the underlying probabilistic model of a data set is monotone DAG-faithful – which means it requires a stronger assumption.

From the definition of monotone DAG-faithful models we know that these models form a subset of DAG-faithful models. We have found that some models are DAG-faithful but not monotone DAG-faithful. To illustrate this, consider the probabilistic model shown in Figure B.1:

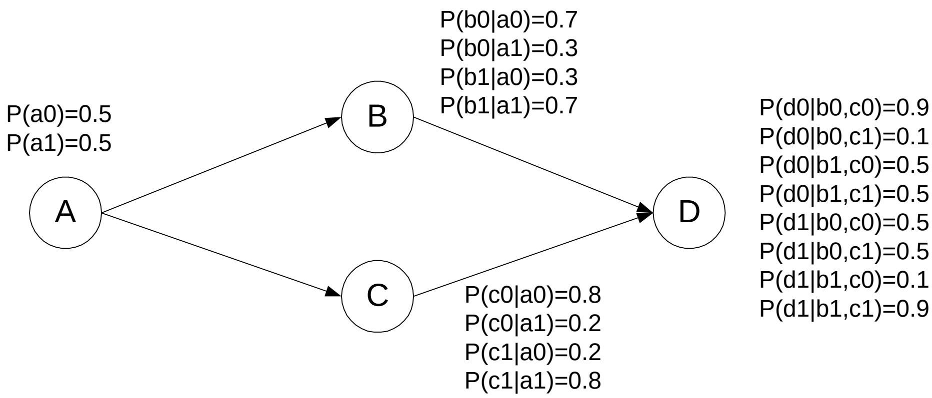

Figure B.1: Simple Bayesian Network

If the model was monotone-DAG-faithful, we expect $I(B,C \mid D)$ to be greater than $I(B,C \mid A,D)$. However, we find $I(B,C \mid A,D) = 0.018$ and $I(B,C \mid D) = 0$: when using {*A*,*D*} as the condition-set, there is one open path *B*-*D*-*C* and when using {*D*} as the condition-set, there are two open paths, *B*-*D*-*C* and *B*-*A*-*C*. Note that this model is not even DAG-faithful since the independence between *B* and *C* given {*D*} cannot be expressed by the DAG structure. However, if we change the parameters of the network a little, for instance, changing the CPtable of node *C* to the same as that of node *B*, we can make $I(B,C \mid D)$ greater than 0 but still smaller than $I(B,C \mid A,D)$. Now, we get a model that is DAG-faithful since *B* and *C* are not independent given {*D*}, but not monotone DAG-faithful.

From the above example, we can draw two conclusions. First, there are some models that are DAG-faithful but not monotone DAG-faithful. Secondly, the distinction between DAG-faithful models and non-DAG-faithful models is not black and white. In the above example, if the small value *I(B,C|D)* happens to be larger than the threshold used to separate ‘dependent’ and ‘independent’, then the model is DAG-faithful; otherwise, it is not DAG-faithful. This shows that there is a ‘gray’ area of the area of DAG-faithful models, in which the models are ‘close’ to being non-DAG-faithful. Although we do not have a formal proof, we conjecture that the non-monotone DAG-faithful models are all in the ‘gray’ area. In other words, if a model violates the monotone DAG-faithfulness assumption, we conjecture that it is also close to the violation of DAG-faithfulness. If so, then any such model may also be problematic for other learning algorithms.

Given the fact that qualitative CI test based learning methods, like *TPDA-Π,* require the qualitative DAG-faithfulness assumption for their correctness proof, it is reasonable to think that our quantitative CI test based method requires a quantitative assumption. We view the monotone DAG-faithfulness assumption used in *TPDA* as the quantitative counterpart of the DAG-faithfulness assumption. We believe that most real-world probabilistic models are actually monotone DAG-faithful.

Even when the underlying probability distribution is DAG-faithful but not monotone DAG-faithful, our algorithm may still be able to learn 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 connects the two nodes by a single collider and removing a node in the condition-set causes the violation of the monotone DAG-faithful assumption. However, since this will only cause one edge to be wrongly added to the current graph, the correctness of other edges in the graph will not be affected and the resulting graph can still be very close to the real model.

Appendix C: Introduction to BN *PowerSoft Package*

There are several commercial systems and research prototypes for learning Bayesian networks from data, including TETRAD II [Scheines *et al.*, 1994], Bayesian Knowledge Discoverer [Ramoni and Sebastiani, 1997], CoCo [Badsberg, 1992], BUGS [Thomas *et. al.*, 1992], BIFROST [Hojsgaard *et al.*, 1994] and MIM [Edwards, 1995]. (See also http://http.cs.berkeley.edu/~murphyk/Bayes/bnsoft.html .) However, as far as we know, only TETRAD II can handle a data set at the size of the ALARM network data we used, which contains 37 variables and 10,000 records. Considering that real-world data sets often contain hundreds of variables and millions of records, the size of the ALARM network data is actually quite moderate. The lack of practicable, easy-to-use learning systems that scale well hinders the real use of Bayesian networks in industry. As a result, most industry users are unaware of the current progress in this area. This is, at least partially, the reason that the Bayesian network method is not as popular as other methods, like neural networks and decision trees, in current data mining systems in industry.

To promote the real use of Bayesian networks and facilitate researchers in related fields, we implemented our algorithms into a Bayesian network learning system, named "Bayesian network

PowerConstructor". This system implements two learners (corresponding to *TPDA* and *TPDA-Π*). Since October 1997, over 4000 people have visited our web sites and over 2000 people have downloaded our system. We are also very glad to know that some users have used it successfully on real-world problems. In May 2000, we also extended the PowerConstructor system to a full-fledged data mining system – BN PowerPredictor, which has most of the features of PowerConstructor and additional features for data mining applications. In addition, our software package also includes a data pre-processing tool for data importing and data discretization. Both systems run under 32-bit windows systems (i.e., Windows 95, Windows 98, Windows NT and Windows 2000) on PCs. They are available for download from our web site (http://www.cs.ualberta.ca/~jcheng/bnsoft.htm).

Using this package, we obtained very encouraging results on a set of standard classification problems [Cheng and Greiner, 1999; Cheng and Greiner 2001]. We also won the ACM KDD Cup 2001 data mining competition "Task 1: prediction of molecular bioactivity for drug design", by learning, from training data, the classifier (here a Bayesian network) with the best prediction accuracy. There were 114 groups who participated in this task, using various data mining techniques [Page and Hatzis, 2001].

Summary of BN PowerSoft Package

This software package includes BN PowerConstructor, BN PowerPredictor and a data pre-processor. Besides its efficiency and scalability, our systems have the following features.

- User-friendly *interface* with online help.
- *Accessibility*. The system supports most of the popular desktop database and spreadsheet formats, including Ms-Access, dBase, Foxpro, Paradox, Excel and text file formats. It also supports remote database servers like Oracle, SQL-server through ODBC.
- *Reusability*. The engine is an ActiveX DLL, so it can be easily integrated into other Bayesian network, data mining or knowledge base systems for Windows 95/98/NT/2000.
- *Supporting domain knowledge*. Complete ordering, partial ordering and causes and effects can be used to constrain the search space and therefore speed up the construction process.
- Automatic feature subset selection and model selection in PowerPredictor by using a wrapper approach.

- Supporting misclassification cost function definition in PowerPredictor.