Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Sciences 168 (2004) 111–132
INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL
www.elsevier.com/locate/ins

A formal approach to using data distributions for building causal polytree structures

M. Ouerd [a], B.J. Oommen [b,*], S. Matwin [a]

[a] *School of Information Technology and Engineering, University of Ottawa, Ottawa, Canada K1S 5B6*
[b] *School of Computer Science, IEEE, Carleton University, Ottawa, Canada K1S 5B6*

Received 10 February 2003; received in revised form 17 November 2003; accepted 13 January 2004

Abstract

We consider the problem of approximating an underlying distribution by one derived from a dependence polytree. We propose a formal and systematic algorithm, which traverses the undirected tree obtained by the Chow method [IEEE Trans. Inform. Theory 14 (1968) 462], and which subsequently processes the latter using the knowledge of inter-node independence tests. By using the tree structure and *these* independence tests, our scheme successfully orients the polytree using an application of the depth first search (DFS) strategy to multiple causal basins. The algorithm has been formally proven, and rigorously tested for synthetic and real-life data.
© 2004 Elsevier Inc. All rights reserved.

1. Introduction and background

Over the last decade Bayesian learning principles have received a fair amount of attention. Although they are elegant, they usually involve summations or integrals along all possible instantiations of the parameters and along all possible models. In the case of learning of Bayesian networks (which

* Corresponding author. Tel.: +1-613-520-4333; fax: +1-613-520-4334.
E-mail addresses: ouerd@site.uottawa.ca (M. Ouerd), oommen@scs.carleton.ca (B.J. Oommen), matwin@site.uottawa.ca (S. Matwin).

is distinct from Bayesian learning itself) this can be perceived as a discrete optimization problem [5]. Precise solutions of this can be obtained by using search techniques, if we assume that there are only a few relevant models. This has proven to be the method of choice in many real-life applications [1].

Many of the Bayesian models, which are studied, are intractable [3,10]. The challenge is to find general-purpose, tractable approximation algorithms for reasoning with these elegant and expressive stochastic models. [1] For example, if we are to use Bayesian learning to improve performance of distributed database applications, where there can be millions of transactions every day, we will need an efficient technique to build a model of the use of the database. The belief network that underlies the Bayesian learning is at the heart of the approach. The connection between Bayesian learning and belief networks is that one can *use* Bayesian techniques to induce a belief network referred to as a Bayesian belief network (BBN). Often, due to the lack of domain knowledge and in the interest of simplicity, it is assumed that the underlying structure is in a particularly simple form, representing reciprocal independence of variables involved.

The reader will observe that the learning, clearly, benefits if a more comprehensive and causal *model* of interaction between the variables is available. Such a model, represented as a Bayesian network, plays the role of a restricted hypothesis bias [9]. The method allows us to obtain the approximating probability distribution $P(X)$ by a well-defined and easily computable density function $P_a(X)$. Indeed, it is impractical to store all estimates of the joint function $P(X)$ for all possible values of the vector X. Our goal is to build a probabilistic network from the distribution of the data, which adequately represents it. Once constructed, such a network can provide insight into probabilistic dependencies that exist between the variables.

In order to measure the "goodness" of the approximation, an information theoretic measure can be specified in terms of the Kullback–Leibler [8] cross-entropy metric to compare joint probability distributions. Chow and Liu [2] used this measure to approximate discrete distributions by collecting the entire first and second order marginals. They derived a relationship between the measure of closeness between the probabilities and the measure of independence between all the pairs of the variables. A maximum weight-spanning tree (MWST) called the "Chow tree" [2,3,6,10,14,15,17,21–23] was built using the information measure between the variables forming the nodes of the tree. An alternative method of obtaining such a tree using the χ^2 metric was later

[1] Our algorithm is not an *approximation* algorithm. By virtue of Chow's result and the independence tests, it will yield the MLE of the true underlying polytree if the underlying distribution obeys a polytree nature. But if the underlying distribution does not obey a polytree structure, our algorithm turns out to be an *approximation* algorithm, which approximates the true structure with a polytree.

proposed by Valiveti and Oommen [17]. A subsequent work due to Rebane and Pearl [15] used the Chow tree as the starting point of an algorithm which builds a polytree (singly connected network) from a probability distribution. This algorithm orients the Chow tree by assuming the availability of independence tests on various multiple parent nodes.

The works of Rebane and Pearl [15] are commendable. Although, as we shall see, they did not answer all the questions regarding polytrees, their inference and their characterization, in our opinion their work was pioneering (i.e., with regard to polytree representations) and represented a quantum jump since the work done on trees in the late 1960s. In our opinion, their most fundamental contribution was to discover and utilize the edge "orienting" principle [18] referred to later.

Numerous authors have built on the foundation of the work of Rebane and Pearl. Noteworthy are the results of Srinivas et al. [16] who worked with independence, and the recent results of Dasgupta [3,10] which explicitly specifies the complexity of the underlying problem. Friedman has also worked in this area and has modified the traditional EM algorithm to devise the "structural" EM algorithm [4] to learn BBNs, and also demonstrated how one can learn Bayesian networks from massive data sets using the "sparse candidate algorithm" [16].

Much of the current work has made substantial progress on learning the structure of multiply connected networks and dynamic Bayesian networks, and this has even been achieved in the presence of hidden variables and for real data sets for which perfect independence tests are not realistic.

1.1. Why polytrees

The most simplistic model for random vectors is the one that assumes that the components of the vector (the individual random variables) are statistically independent. At the other end of the spectrum, we have the model in which every variable is assumed to be dependent on every other variable, making the model both cumbersome, and often intractable. The first attempt to strike a happy medium was the one due to Chow [2,3,8,14,17,21,22] in which the model assumed that there was a tree-based dependence between the variables. This dependence has been exhaustively studied for discrete and continuous vectors using various error norms including the entropy and Chi-square norms.

The tree-based model has been successfully generalized to specify the dependence using polytrees, which is the model of interest of this study. The question: "Why polytrees?" is thus not rhetorical, and deserves justification. Indeed, in his doctoral thesis, Dasgupta (see Section V.1 titled "*Why polytrees*" in [10]) argued, that while it is easy to learn branchings of *trees* in the structured learning problem, it is NP-Hard to learn probabilistic nets even if the degrees are bounded. He suggested, quite eloquently, that the first step in moving from simple branchings towards generalized networks was to consider the polytree structure. While Dasgupta [3,10] considered how an approximation algorithm for polytrees could be developed, de Campos and his colleagues showed how these structures could be learned from data in Heute and de Campos [21], and presented a solid experimental verification of their proposed techniques in Acid and de Campos [20]. They also describe a system [22], *CASTLE*, implementing these techniques. More recently, Meila and Jaakkola [23] have studied specific analytic problems associated with learning tree belief networks, although they have not specifically addressed the problem that we have solved here.

1.2. Problem statement and outline of solution

This paper deals with the problem of automatically building a belief network [2] in terms of a directed polytree, with the assumption that the observations have been presented to the system in terms of joint probability distributions [5,6,14,18,19]. We assume that the true underlying dependence obeys a polytree structure, and that the joint dependence relations represented by these observations is available. Pearl and his co-authors [14,15] discussed this process using a two-phase dependence learning scheme. Our aim is to find causal [3] polytree structures that fit the data presented in terms of joint probability distributions. The question of inferring the polytree structure from the data (as opposed to the data distribution) is the study of a subsequent paper presently being compiled and is described in [11,12].

Before we embark on discussing our contribution, we mention that in the context of polytrees, the work of Rabane and coworkers [14,15,18,19] can be perceived to be of a pioneering sort! Also, the reader must observe that polytrees represent much richer dependency models than undirected trees, because their joint probability density functions are products of higher order distributions. Consequently, (as argued in [3,10,23]) the problem itself can be shown to be a much harder problem than that of finding the best tree [14]. Indeed, first of all, the reported algorithms do not guarantee to yield a polytree structure if the underlying distribution is degenerate, and not of a polytree type distribution i.e., if the distributions do not fit into a polytree representation. Secondly, the algorithm relies on the repeated use of the independence tests

[2] We assume that the reader is reasonably familiar with the concepts and terms involved in the learning of belief networks such as "polytree", "articulation point", "causal basin" etc. Their formal definitions can be found in [14].

[3] The concept of causality is not derived from the semantics of the probabilistic graphical models, which is, in turn, related to the conditional independences between the triplets of variables. On the contrary, causality is a well-studied and "hot" topic of research for the community of researchers interested in issues related to "Uncertainty in Artificial Intelligence". It is found on principles that we do not discuss here—such a discussion would merely distract.

that determines categorically whether two random variables X_i and X_j are statistically independent. As shown in [12], even if the random variables are statistically independent, the experimental evaluation may never yield conclusive independence decisions.

Although the work in [15] is fundamental and pioneering, if we consider the polytree construction algorithm as given by [15], we observe that the order of traversing the tree, in order to render it oriented, is not formally specified. The implementation strategy of determining the order of dependence tests is unanswered and left to the reader. In this paper, we shall formally develop an algorithm which answers these questions. First of all, the algorithm determines the network structure or the tree using the MWST algorithm described earlier, and subsequently orients the tree by beginning with the assumption that we have marginal independence between *at least* two parents of any node. Thus if, there is no independence of any two parents of a node the algorithm will terminate by informing the user that the underlying tree structure cannot be oriented to yield a polytree.

The problem of orienting the tree is solved in two steps. The first step identifies all the independencies, where, two nodes X_i and X_j are independent if the following is satisfied: $P(X_i, X_j) = P(X_i) * P(X_j)$. Although, as mentioned above, this equality is not always satisfied with sample data, in this paper, we assume that such independence inferences are available. We also assume that we are provided with this information whenever it is requested. The second step is as follows: after inferring all the statistical independence between the pairs of variables, we use the following *Orienting principle (T)* due to Pearl [14] to completely orient the tree.

Orienting principle (T): For every unoriented triplet of variables X, Y and Z ordered as: $X–Z–Y$, we test for the independence of X and Y. If X and Y are independent then X is a parent of Z and Y is a parent of Z. For any triplet X, Y and Z such that: $X \rightarrow Z–Y$, we test if X and Y are independent, and if this is so Y is parent of Z otherwise Y is a child of Z.

The details of why this principle works is shown using an example taken directly from Verma and Pearl [18]. Given three variables X, Y and Z, the set of causal dependency models that could exist between these variables are shown in Fig. 1.

As explained in [19] (in what follows, we refer the reader to Fig. 1), the set of parameters portrayed by the dependency model (M_1) are: $P(X)$, $P(Z|X)$ and $P(Y|Z)$. The set of parameters required for model (M_2) are $P(X|Z)$, $P(Y|Z)$ and $P(Z)$. For model (M_3), the probabilities required for its structure are: $P(Y)$, $P(Z|Y)$ and $P(X|Z)$. It is easy to see that models (M_1) and (M_2) are equivalent because $P(X)P(Z|X) = P(XZ) = P(Z)P(X|Z)$. Furthermore, model $(M3)$ is also equivalent to these models since its parameters $P(Y)$, $P(Z|Y)$ and $P(X|Z)$ can be obtained from those of models (M_1) and (M_2) [19]. However, model (M_4) is completely different from the previous models. Its parameters are $P(X)$, $P(Y)$

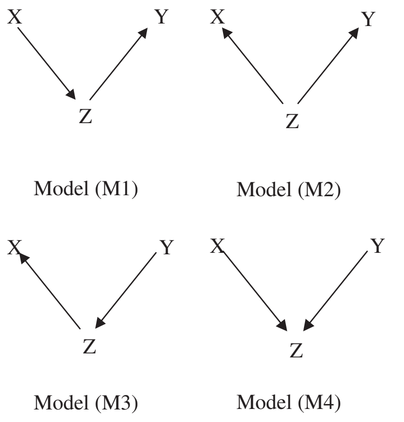

Fig. 1. The causal models that can be obtained with three variables X, Y and Z.

and $P(Z|X,Y)$, and these parameters cannot be obtained from the sets of parameters of models $(M_1–M_3)$. The directed acyclic graphs (DAG) representing models $(M_1–M_3)$ are indistinguishable in the sense that they carry the same set of independence assertions; $I(X,Z,Y)$ (i.e., X and Y are conditionally independent given Z), while the DAG representing model (M_4) is distinguishable from the previous DAGS because it represents the independence $I(X,\emptyset,Y)$ which implies that X and Y are marginally independent which is a condition not represented in either of the former DAGs. In [14], the principle of model (M_4) is used to orient the skeleton of the polytree.

2. A depth first search algorithm for building polytrees

Our algorithm for inducing the polytree is an application of the depth first search (DFS) algorithm to causal components of the undirected tree. Let $T=(V,E)$ be a connected, undirected tree where V is the set of vertices, and E, the set of edges. A vertex Z is said to be an *articulation point* between vertices X and Y if we have independence between X and Y. As defined by Pearl [14] and used by all the researchers since, a causal basin starts with a multi-parent cluster (a child node and all of its direct parents) and continues in the direction of causal flow to include all of the child's descendants and all of the direct parents of those descendants. An example of this is given in Fig. 2.

2.1. Problems with Pearl's algorithm

Although the above definition is consistent, there are some unanswered questions which arise from the work of Rebane and co-workers [14,15]. In fact,

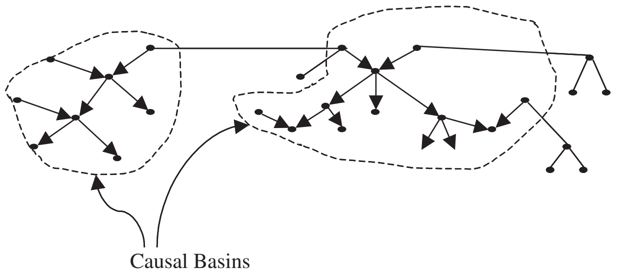

Fig. 2. A causal basin as defined by Pearl [14].

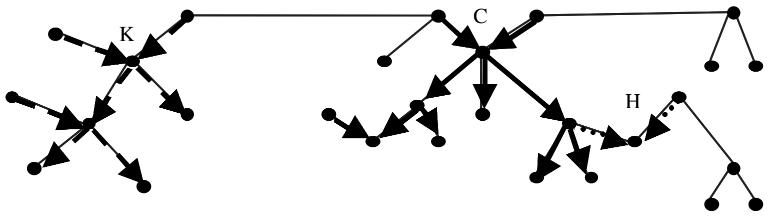

Fig. 3. Three causal basins starting respectively at nodes H, K and C.

although they specify a formal algorithm to compute the causal basins, they leave the following questions unanswered:

1. The question of what is meant by the outermost layer is not clear since it "depends on the tree" and its representation.
2. The question of how the traversal is done is not completely defined.
3. The algorithm introduces ambiguity regarding the edges that are already traversed.
4. The notion of causal basins depends on the starting point. [4]

The last of these issues can be seen from the following figure in which the Chow tree of this polytree is taken from [14].

Observe that the Chow tree of Figs. 2 and 3 is the same. In Fig. 2, the first articulation point (starting point) is node C and the second articulation point is

[4] Apparently the works of Pearl et al. do not highlight this salient point. We too do not solve the problem that results from the "starting point issue" in its entirety. However, having observed it, we have attempted to deliver a comprehensive solution based on this assumption.

node K. Having this order for the choice of the starting points we detect two causal basins as given in Fig. 2. In Fig. 3, we use node H as the starting point, node C as the second and node K as the third to be able to complete the same orientation of the Chow tree as in Fig. 2 using three causal basins instead of two. From this it is easy to see that the starting point determines the individual causal basins.

2.2. Motivation for a DFS strategy

Consider the process of visiting the vertices of an undirected tree in the following manner. We select and "visit" a starting vertex Z which is one of the articulation points in T and in particular, an articulation point between two nodes X and Y. First of all we orient the edges (X,Z) and (Y,Z) as pointing to node Z following the orienting principle since we have independence between them. Then we select any edge (Z,W) incident upon Z. We check for independence between nodes X and W to determine the orientation of edge (Z,W). We observe two possible scenarios: If there is no independence between X and W then edge (Z,W) is pointing to node W. We then visit node W and begin to search for a new edge starting at vertex W. After completing the search through all causal paths beginning at W, the search returns either to Z, the vertex from which W was first reached, to search through all nodes in the adjacency list of Z, or to another non-visited articulation point. If there is independence between X and W the edge (Z,W) is pointing to node Z, and the search returns either to Z, the vertex from which W was first reached, to search through all the nodes in the adjacency list of Z, or to another non-visited articulation point. The process of selecting unexplored edges incident on Z is continued until this list is exhausted. This is formalized in the algorithm Polytree-Depth-First-Search.

The input to the algorithm is the set of nodes, and for every node X_i we specify its "adjacency" list, namely, a list of nodes X_j such that arc X_i–X_j exists in the underlying tree structure. Also provided are the independence tests between nodes whenever required. The algorithm is formally given below.

Algorithm Polytree-Depth-First-Search
Input: (a) A tree $T = (V,E)$.
(b) Independence test available for every pair of nodes when it is required.
(c) For every node, a list of all its direct neighbours specified as a connected list specified as ConList. We assume that the test for a node being an articulation point is a straightforward operation. Also, a node W is in the causal basin of X if there is a path from X to W.
Output: A directed polytree if the orientation exists. It returns T, the undirected tree if any orientation is not possible. Even if T cannot be fully

oriented, edges in every causal basin will be oriented, and the other edges can be oriented if additional information is provided.

```
**Method**
**Begin**
  **For** (all $X$ in $V$) **Do**
    Visited [$X$] $=$ false /* Visited is an array holding the nodes */
  **EndFor**
  **For** (all $X$ in $V$) **Do**
    /* always start with an articulation point */
    **If** (!Visited[$X$] and $X$ is an articulation point) **Then**
  **Call** Processing ($X$)
    **EndIf**
  **EndFor**
**End Algorithm Polytree-Depth-First-Search**
**Procedure Processing** ($X$)
**Begin**
  /* node $X$ is not a leaf */
  **If** ((Visited[$X$] $=$ false) and (ConList($X$)) $>$ 1) **Then**
    /* orient the adjacent edges */
  **Call** IndepOrient($X$)
    Visited[$X$] $=$ true
    /* traverse the adjacency list of $X$ */
    **For** (all W in the ConList of $X$ and $W$ is in causal basin of $X$) **Do**
      /* Processing is recursive because of Depth-First Search */
  **Call** Processing ($W$)
    **EndFor**
  **EndIf**
**End Algorithm Processing**
**Procedure IndepOrient** ($X$)
**Begin**
**For** (every distinct $N_1$ and $N_2$ in ConList($X$)) **Do**
  **If** Indep($N_1$,$N_2$) $=$ True **Then**
    print arcs from ($N_1$ to $X$) and ($N_2$ to $X$)
  **EndIf**
**EndFor**
**For** (every distinct $N_1$ and $N_2$ in ConList($X$)) **Do**
  **If** (arc from $N_1$ to $X$ exists and edge from $N_2$ to $X$ is unoriented) **Then**
    print arc from $X$ to $N_2$
  **Else If** (arc from $N_2$ to $X$ exists and edge from $N_1$ to $X$ is unoriented) **Then**
    print arc from $X$ to $N_1$
  **EndIf**
**EndFor**
**End Algorithm IndepOrient**
```

2.3. Analytic properties of the algorithm

The formal proof that the above algorithm works follows the arguments of the DFS traversal of a graph. To clarify how this is achieved we now present a straightforward example. [5]

Example 1. Consider the following undirected tree given in Fig. 4. Here, we assume that the independence information given in Table 1. This information may either be given a priori or obtained "on request". Also, for the sake of simplicity, we assume that the tree is formed of one causal basin.

As stated, the starting point is always an articulation point. Here, we assume that the DFS algorithm takes node 1 as the starting point to orient the tree since we have independence between nodes 2 and 3 (Fig. 5).

Step 1: $I(2,3) = T$. Therefore the edges are directed as:

$$\begin{tikzcd} 2 \arrow[r] & 1 & 3 \arrow[l] \end{tikzcd}$$

Step 2: The next independence tests between neighbors of 1 are: $I(3,4) = F$ and $I(2,4) = F$. Since edge (2, 1) is already directed as pointing to 1, edge (1, 4) is directed in the causal basin of the edge (2, 1) as:

$$\begin{tikzcd} 2 \arrow[r] & 1 \arrow[r] & 4 \end{tikzcd}$$

After directing all the edges starting at node 1, we mark it as a visited node and we progress in the DFS manner to the nodes belonging to its adjacency list.

Step 3: Both nodes 2 and 3 are marked as visited because they are leaves. We therefore move to node 4 and start the same process of testing independence between its neighbors as we did for node 1. We check for independence between every pair of its neighbors, which are: $I(6,5)$, $I(1,5)$ and $I(1,6)$. These tests fail and we know already that edge (1, 4) is pointing to node 4. Thus edges (4, 5) and (4, 6) are directed in the same causal flow as with edge (1, 4). The direction of the edges for the nodes processed till now is:

Step 4—End: The same process will continue for all the nodes of the tree. The final figure is as below, and all the edges will be directed in the same flow as they are in the same causal basin. The result of orienting the whole tree is given in Fig. 6 below.

[5] This example (and the one given later) are a little tedious. We include them here *only* for the sake of illustration—to highlight the salient points of the algorithm. They were included on the recommendation of the reviewers.

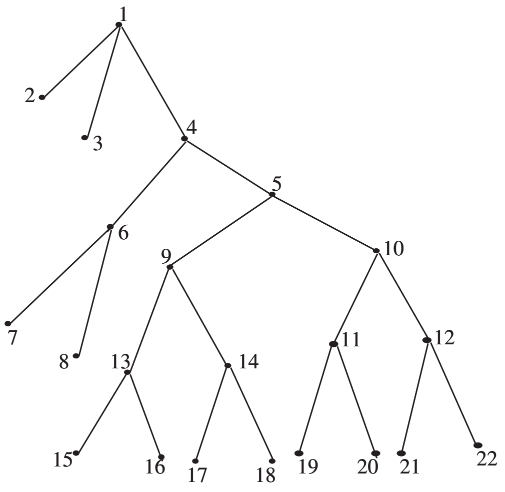

Fig. 4. A tree example.

Table 1
Independence information about the nodes of the tree above

$I(2,3)=T$	$I(2,4)=F$	$I(3,4)=F$	$I(5,1)=F$	$I(1,6)=F$
$I(5,6)=F$	$I(4,8)=F$	$I(4,7)=F$	$I(7,8)=F$	$I(4,9)=F$
$I(4,10)=F$	$I(9,10)=F$	$I(9,16)=F$	$I(15,9)=F$	$I(15,16)=F$
$I(9,18)=F$	$I(17,9)=F$	$I(17,18)=F$	$I(12,5)=F$	$I(11,5)=F$
$I(11,12)=F$	$I(10,19)=F$	$I(10,20)=F$	$I(19,20)=F$	$I(10,22)=F$
$I(10,21)=F$	$I(21,22)=F$			

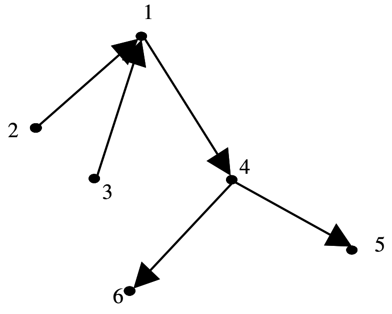

Fig. 5. Portion of polytree oriented at the end of Step 3.

Theorem 1. *The algorithm Polytree-Depth-First-Search correctly computes the polytree given the skeleton tree structure and the underlying independence relationships.*

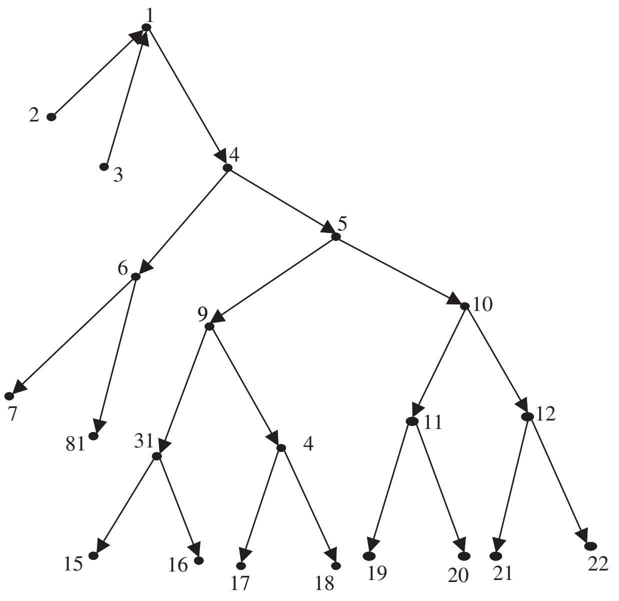

Fig. 6. The polytree obtained from the tree in Fig. 4 and the independence tests of Table 1.

Proof. The proof of the correctness of the algorithm is achieved by induction on the number of the nodes. We shall prove the following statements:
(i) Every node of the tree is visited, and,
(ii) Every edge in the tree is directed by the DFS as imposed by the ordering given by the independence tests.

Without loss of generality, we assume that we are processing one causal basin from the starting node. Thus if a node is not visited it will be a starting point for another causal basin. The first statement is trivial as a direct result of the DFS traversal to the tree. The more crucial argument involves the *orientation* of its edges.

Basis step: We use an articulation point as the starting point for the algorithm since it is clear that the algorithm will not lead to any orientation without such a starting point. The basis step involves the first articulation point X. Let the variables $\{X_i\}$ denote the neighbors of X and in particular, let X_n and X_m be the neighbors of X that satisfy $I(X_n, X_m) = T$ for indexes n and m because X is an articulation point. Thus edges (X_n, X) and (X, X_m) are oriented as pointing to node X since X_n and X_m are independent. We then proceed to check for independence between all pairs of the neighbors of X to orient the corresponding edges. If a test $I(X, X_j) = T$ for some j, it implies that the edge (X, X_j) points to node X; otherwise it points to node X_j. This achieves the orientation of all the neighbors of X.

Inductive hypothesis: Suppose that at a certain stage of the algorithm we are visiting node Y, where node Y can be any node of the tree. Suppose that Y_c is one of the children of Y, and Y_p is one of the parents of Y where the edge (Y, Y_c) is unoriented. When visiting node Y, we check for all the independence tests between the neighbors of Y which include Y_c and Y_p. We know that edge (Y_p, Y) is already oriented as pointing to Y since we have earlier visited Y_p. Two scenarios can happen:

The first scenario is that the independence test $I(Y_p, Y_c)$ is false. In such a case, as per the orienting principle, we orient edge (Y, Y_c) as pointing to Y_c.

The second scenario is when the independence test $I(Y_p, Y_c)$ is true. In this case both edges (Y_p, Y) and (Y, Y_c) point to node Y.

By the same reasoning it is clear that we can keep orienting all the neighbors of Y, and using the DFS procedure we traverse the whole tree and orient all the edges. The result is thus true for all edges in one causal basin.

The result is also true if we have more than one causal basin, because once we have exhausted the nodes in the first causal basin, we would invoke the same DFS strategy from a new starting point in the next causal basin. Hence the theorem. □

The above algorithm uses a DFS strategy. It is easy to devise an analogous algorithm, which uses a Breadth-First search strategy, or for that matter, any systematic search scheme. Also, observe that this is a worst-case scenario. Thus, it may be possible to optimize our algorithm to require less independence tests when *other* independence relationships can be inferred from tests that have already been done. This is an open problem.

We shall now state results regarding the complexity of the above algorithm. This is done by considering the number of independence tests that need to be performed to be able to orient the tree. To clarify issues, we first present a straightforward example.

Example 2. Consider the case of Fig. 7 in which n, the number of nodes is 22. The root (node 1) has three children, and all the other nodes have only two children. Note that this exactly fits our model since every internal node has three dependent nodes—its parents and its two children, and thus, as mentioned in this case, $n = 22$ and $k = 3$.

We list below the number of independence tests, which need to be done for this tree in order to orient it.

Depth $=$ **0**: At this level *three* tests which must be done, i.e., $I(2,3)$, $I(2,4)$, $I(3,4)$ which is $\binom{3}{2} = 3$ tests.

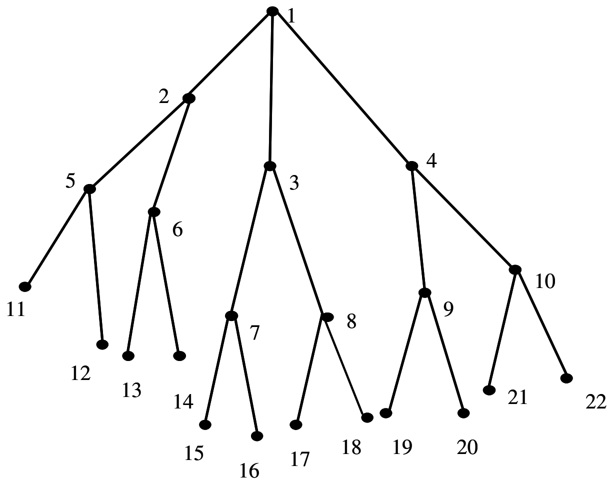

Fig. 7. A tree example.

Depth = 1: At this depth the result of *nine* tests must be returned, namely, $I(5,6)$, $I(5,1)$, $I(6,1)$, $I(7,8)$, $I(7,1)$, $I(8,1)$, $I(9,10)$, $I(9,1)$, $I(10,1)$. The number of tests is seen to be: $3 * \binom{3}{2} = 9$ tests.

Depth = 2: The 18 tests to be done in this case are: $I(11,12), I(11,2), I(12,2), I(13,14), I(13,2), I(14,2), I(15,16), I(15,3), I(16,3)$, $I(17,18), I(17,3), I(18,3), I(19,20), I(19,4), I(20,4), I(21,22), I(21,4), I(22,4)$. The number of tests can be seen to be : $3 * (3–1)^1 * \binom{3}{2} = 18$ tests.

Thus the total number of tests to be done at all levels is 30. The above results are formalized below.

Remark. From a straightforward examination it is clear that the overall burden of the computation can be obtained by observing that for each node we have to do pairwise independence tests between its neighbors. Thus, in a straightforward manner the cost is $n * \binom{k}{2}$. However, to understand what happens at every level of the tree, a more detailed study is needed. This is done in Theorem 2.

Theorem 2. For a tree with n nodes, in which every node has k adjacent nodes, at depth d in the tree, the total number of independence tests that have to be done, is:

$$k * (k-1)^{d-1} * \binom{k}{2}.$$

Proof. The proof is done by induction.

Basis Step

The basis step considers all nodes of depths 0, 1 and 2.

For **Depth 0**, the root has k dependents, and so we have to test every pair of nodes for independence. This results in $\binom{k}{2}$ tests $= \frac{1}{2} * k * (k-1)$ tests.

For **Depth 1**, every node in the k nodes has $(k-1)$ dependents not counting the parent. The number of independence tests is $k * \binom{k-1}{2}$ tests for children and $k * (k-1)$ tests involving the nodes with the grandparent. This is equal to **Test(Depth 1)**, where,

$$\begin{aligned}\textbf{Test}(\textbf{Depth 1}) &= k * \binom{k-1}{2} + kc(k-1)\\ &= \left[\frac{1}{2} * k * (k-1) * (k-2) + k * (k-1)\right] = k * \binom{k}{2}.\end{aligned}$$

For **Depth 2**, every node in the $k * (k-1)$ nodes has $k-1$ dependents. The number of independence tests is **Test(Depth 2)** where

$$\begin{aligned}\textbf{Test}(\textbf{Depth 2}) &= \left[k * (k-1) * \binom{k-1}{2} + k * (k-1) * (k-1)\right]\\ &= k * (k-1) * \left[\frac{1}{2} * (k-1) * (k-2) + (k-1)\right]\\ &= k * (k-1) * (k-1)\left[\frac{1}{2} * (k-2) + 1\right] = k^2 * (k-1)^2\\ &= k * (k-1) * \binom{k}{2}.\end{aligned}$$

As mentioned before, the expressions for $d = 0$, $d = 1$ and $d = 2$ collectively constitute the basis case.

Inductive Hypothesis

We now assume that the property is true until level $d-1$, i.e., the number of independence tests at depth $d-1$ is equal to **Test(Depthd − 1)** where,

$$\textbf{Test}(\textbf{Depth d} - \textbf{1}) = k * (k-1)^{d-2} * \binom{k}{2}.$$

At depth d or layer d in the tree we have $k * (k-1)^{d-1}$ nodes each having $(k-1)$ dependents not counting the parent. Once we are at depth d, we have to perform $\binom{k-1}{2}$ tests for every node at this depth, which amounts, for $k * (k-1)^{d-1} * \binom{k-1}{2}$ tests. Subsequently, we have to check every child of the $k * (k-1)^{d-1}$ nodes with its grand parent, which amounts for $k * (k-1)^{d-1} * (k-1)$ tests. Thus, the total number of tests is **Test(Depth d)** where,

$$\begin{aligned}\mathbf{Test}(\mathbf{Depth\ d}) &= \left[k * (k-1)^{d-1} * \binom{k-1}{2} + k * (k-1)^{d-1} * (k-1)\right] \\ &= k * (k-1)^{d-1} * \left[\binom{k-1}{2} + (k-1)\right] \\ &= k * (k-1)^{d-1} * \left[\frac{1}{2} + k * (k-1)\right] = k * (k-1)^{d-1} * \binom{k}{2} \\ &= (k-1) * k * (k-1)^{d-2} * \binom{k}{2} \\ &= k * (k-1)^{d-1} * \binom{k}{2}.\end{aligned}$$

The theorem follows. □

Corollary to Theorem 2. As mentioned earlier, the total number of independence tests which needs to be done for a tree with n nodes of depth d and branching factor k is $n * \binom{k}{2}$. This follows from the above theorem since:

$$\mathbf{Total\ Tests} = \sum_{h=1}^{d} k(k-1)^{h-1}\binom{k}{2} = \binom{k}{2} * k * \sum_{h=1}^{d}(k-1)^{h-1}$$

But since we know that $[k * \sum_{h=1}^{d}(k-1)^{h-1}] = n$, we obtain, **Total Tests** $= \frac{1}{2} * k * (k-1) * n$.

2.4. Limitations of the algorithm

The limitations of the algorithm are primarily due to the lack of information about the independence between the variables. In order to orient the edges we used the independence tests between every two neighbors. This information may not be provided by the user, and it therefore prevents the algorithm from orienting the tree. The example below illustrates the case. In this example, let us suppose that it is not possible to orient edges (F, G) and (E, G) because the dependence information about nodes E and F is not available. Observe that this limitation is also relevant for the work of [15].

Elsewhere [13] we have presented a strategy for providing a temporal ordering of the variables and thus allowing an orientation following this ordering. Thus, if a variable A precedes a variable B in the "time ordering", we have shown how we will permit the orientation of A towards B (Fig. 8).

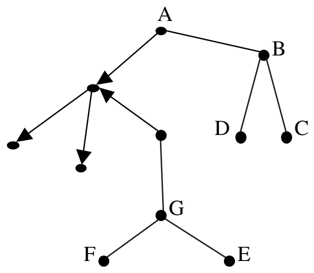

Fig. 8. An example of an incomplete oriented polytree.

3. Experimental results

In order to check the algorithm developed here we have done numerous experiments. We assumed that we were to learn an underlying polytree, which is unknown to the algorithm. One such polytree is given in Fig. 9. Also, we assumed that independence tests, which are consistent with the polytree orientation, were available whenever they were needed by the algorithm. These independence tests for the specific polytree in Fig. 9 are given in Table 2. The polytree learning algorithm specifically modified for discrete data (referred to as *Discrete-Polytree-Depth-First-Search*) was invoked using the skeleton and

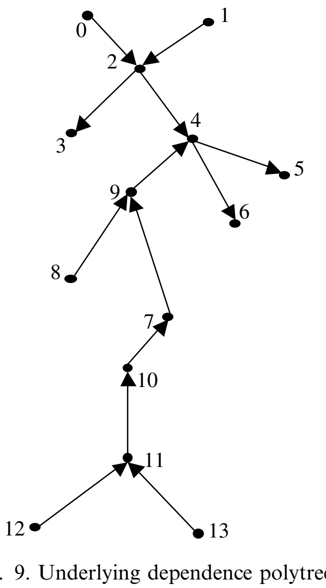

Fig. 9. Underlying dependence polytree.

Table 2
Independence information for the polytree given in Fig. 9

$I(0,1) = T$	$I(2,6) = F$	$I(7,11) = F$
$I(2,9) = T$	$I(2,5) = F$	$I(4,8) = F$
$I(7,8) = T$	$I(3,4) = F$	$I(4,7) = F$
$I(12,13) = T$	$I(5,6) = F$	$I(9,10) = F$
$I(1,3) = F$	$I(5,9) = F$	$I(10,13) = F$
$I(0,4) = F$	$I(9,6) = F$	$I(10,12) = F$

the sequence of independence tests. The results of running the polytree algorithm demonstrate its power.

Since we already know the distribution of the data, we first run the Chow algorithm on the given distribution. We then obtain the skeletal tree of the polytree, the tree given in Fig. 10. We are thus required to orient this tree using the independence tests given in Table 2. In order to orient it the algorithm detected the following articulation points: nodes 2, 4, 9 and 11 as starting points. The algorithm then attempted to orient the tree starting at articulation points, which represent nodes with multiple parents. After running the algorithm, the orientation of every edge of the "true" underlying polytree was compared with the orientation of the edge of the resultant "inferred" polytree.

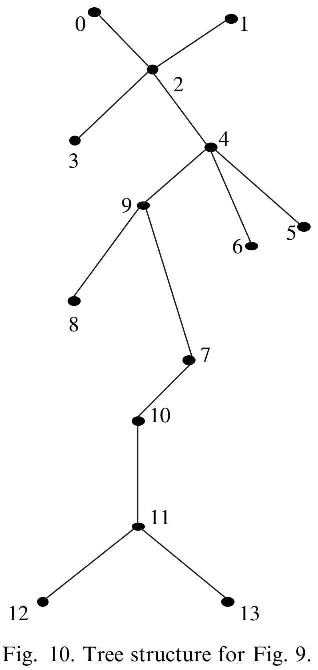

Fig. 10. Tree structure for Fig. 9.

In every case the results are exactly the same. These experiments were conducted for various kinds of polytrees, namely polytrees with one or multiple causal basins. In every case, the polytree was exactly inferred—this was true for every polytree tested.

A second example is given in Figs. 11 and 12. Its independence information is given in Table 3.

Our experimental results consistently demonstrate that the algorithm successfully orients the polytree. Again we observe that if the independence information for any pair of nodes is not provided to the algorithm, it will fail to orient the *entire* tree.

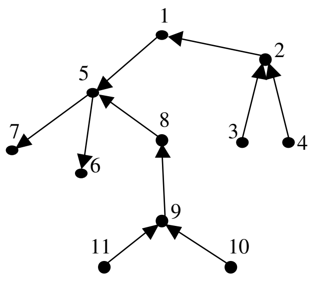

Fig. 11. Underlying dependence polytree.

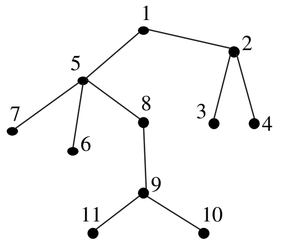

Fig. 12. The skeletal tree of the underlying polytree given in Fig. 11.

Table 3
Independence information for the polytree given in Fig. 11

$I(11,10) = T$	$I(3,4) = T$
$I(10,8) = F$	$I(3,1) = F$
$I(11,8) = F$	$I(4,1) = F$
$I(9,5) = F$	$I(2,5) = F$
$I(8,7) = F$	
$I(8,6) = F$	

We conclude this section by mentioning that the techniques presented here have also been used quite successfully in a real-life application where the problem is to improve performance in systems using repeated queries which access distributed databases [13], and for the ALARM data [7].

4. Conclusion

In this paper we have considered the problem of approximating an underlying distribution by one derived from a dependence polytree. The skeletal form of the polytree is known to be the MWST of a complete graph, with $I_f(X_i, X_j)$, the information theoretic metric, as the edge weight between the pair of nodes X_i and X_j. Once the tree is derived, Rebane and Pearl [15], proposed to use an independence test to determine if a variable has multiple parents. They dictated that every two node neighbors X and Y of a node Z must be tested for marginal independence to decide if Z has parents X and Y.

This paper proposes a formal and systematic algorithm to traverse the tree obtained by the Chow method. It uses an application of the DFS strategy to multiple causal basins. Experimental results clearly demonstrate that when the required independence tests are available to the algorithm the orientation of the polytree is completed, and always correct. The algorithm has also been used in two real-life applications [13] involving distributed databases and the ALARM data.

Acknowledgements

Partially supported by the Natural Sciences and Engineering Research Council of Canada. A preliminary version of some of the results in this paper can be found in the *Proceedings of the 2002 Int'l Symposium on Intelligent Systems,* Charlotte, NC. The latter did not contain the formal theorems or proofs, nor did it contain a detailed description of the actual algorithm.

References

[1] G.F. Cooper, E.H. Herskovits, A bayesian method for the induction of probabilistic networks from data, Machine Learning 9 (1992) 309–347.

[2] C.K. Chow, C.N. Liu, Approximating discrete probability distributions with dependence trees, IEEE Transactions on Information Theory 14 (1968) 462–467.

[3] S. Dasputa, Learning polytrees, in: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm Sweden, July/August 1999, pp. 134–141. The paper can be downloaded from http://www2.sis.pitt.edu/~dsl/UAI/UAI99/Dasgupta.UAI99.html.

[4] N. Friedman, The Bayesian structural EM algorithm, in: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, University of Wisconsin Business School Madison, Wisconsin, USA, July 1998, pp. 129–138. The paper can be downloaded from http://www.cs.huji.ac.il/~nirf/Papers/Fr2.pdf.

[5] N. Friedman, I. Nachman, D. Pe'er, Learning Bayesian network structure from massive datasets: the "Sparse Candidate" algorithm, in: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm Sweden, July/August 1999, pp. 206–215. The paper can be downloaded from http://www2.sis.pitt.edu/~dsl/UAI/UAI99/Friedman.UAI99.html.

[6] D. Geiger, A. Paz, J. Pearl, Learning causal trees from dependence information, in: Proceedings of 1990 AAAI Conference, Boston, MA, MIT Press, 1990, pp. 770–776.

[7] E.H. Herskovits, Computer-based probabilistic-network construction, Doctoral Dissertation, Medical Information Sciences, Stanford University, CA, 1991.

[8] S. Kullback, R.A. Leibler, On information and sufficiency, Annals Mathematics Statistics 22 (1951) 79–86.

[9] T.M. Mitchell, Machine Learning, McGraw-Hill, 1997.

[10] S. Dasputa, Learning probability distributions, Doctoral Dissertation, University of California at Berkeley, 2000.

[11] M. Ouerd, B.J. Oommen, S. Matwin, Generation of random vectors for underlying DAG structures given first order marginals, in preparation.

[12] M. Ouerd, B.J. Oommen, S. Matwin, Inferring polytree dependencies from sampled data, in: Proceedings of the 2002 IEEE International Conference on Systems, Man and Cybernetics, Hammamet, Tunisia, October 2002, in press.

[13] M. Ouerd, Building probabilistic networks and its application to distributed databases, Doctoral Dissertation, SITE, University of Ottawa, Ottawa, Canada, 2000.

[14] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, San Mateo, CA, 1988.

[15] G. Rebane, J. Pearl, The recovery of causal polytrees from statistical data, in: Proceedings of the Third Conference on Uncertainty in Artificial Intelligence, Seattle, Washington, July 1987, pp. 222–228.

[16] S. Srinivas, S. Russell, A. Agogino, Automated construction of sparse bayesian networks from unstructured probabilistic models and domain information, in: Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence, Windsor, Canada, August 1989, pp. 343–350. The paper can be downloaded from http://www2.sis.pitt.edu/~dsl/UAI/uai89.html.

[17] R.S. Valiveti, B.J. Oommen, On using the chi-squared metric for determining stochastic dependence, Pattern Recognition 25 (11) (1992) 1389–1400.

[18] T. Verma, J. Pearl, An algorithm for deciding if a set of observed independencies has a causal explanation, in: Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence, San Mateo, CA, July 1992, pp. 323–330. The paper can be downloaded from http://singapore.cs.ucla.edu/csl_papers.html.

[19] T. Verma, J. Pearl, Equivalence and synthesis of causal models, in: Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence, Cambridge, MA, July 1990, pp. 220–227.

[20] S. Acid, L.M. de Campos, Approximations of causal networks by polytrees: an empirical study, in: B. Bouchon-Meunier, R.R. Yager, L.A. Zadeh (Eds.), Proceedings of the Fifth IPMU Conference, 972–977. Also found in Advances in Intelligent Computing, Springer Verlag, 1994, pp. 149–158.

[21] J.F. Huete, L.M. de Campos, Learning causal polytrees, in: M. Clarke, R. Kruse, S. Moral (Eds.), Proceedings of the 1993 Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty, Lecture Notes in Computer Science, vol. 747, Springer Verlag, 1993, pp. 180–185.

[22] A. Acid, L.M. de Campos, A. Gonzales, R. Molina, N. Perez de la Blanca, Learning with CASTLE, in: R. Kruse, P. Siegel (Eds.), Proceedings of the 1991 Conference on Symbolic and Quantitative Approaches to Uncertainty, Lecture Notes in Computer Science, vol. 548, Springer-Verlag, 1991, pp. 99–106.

[23] M. Meila, T. Jaakkola, Tractable Bayesian learning of tree belief networks, in: Proceedings of the Sixteenth Workshop on Uncertainty in Artificial Intelligence. The paper can be downloaded from http://www.stat.washington.edu/mmp/#publications.