On the Use of Independence Relationships for Learning Simplified Belief Networks

Luis M. de Campos*
Dpto. Ciencias de la Computacion e Inteligencia Artificial, E.T.S.I. Informatica, Universidad de Granada, 18071, Granada, Spain

Juan F. Huete
Dpto. Ciencias de la Computacion e Inteligencia Artificial, E.T.S.I. Informatica, Universidad de Granada, 18071, Granada, Spain

Belief networks are graphic structures capable of representing dependence and independence relationships among variables in a given domain of knowledge. We focus on the problem of automatic learning of these structures from data, and restrict our study to a specific type of belief network: simple graphs, i.e., directed acyclic graphs where every pair of nodes with a common direct child has no common ancestor nor is one an ancestor of the other. Our study is based on an analysis of the independence relationships that may be represented by means of simple graphs. This theoretical study, which includes new characterizations of simple graphs in terms of independence relationships, is the basis of an efficient algorithm that recovers simple graphs using only zero and first-order conditional independence tests, thereby overcoming some of the practical difficulties of existing algorithms. © 1997 John Wiley & Sons, Inc.

I. INTRODUCTION

Belief Networks (also called Bayesian networks, causal networks or influence diagrams) are knowledge-based systems that represent uncertain knowledge by means of both graphical structures, namely directed acyclic graphs (dags), and numerical parameters associated to these graphs. In a belief network, the qualitative component (the graph) represents dependence/independence relationships: the absence of some arcs means the existence of certain conditional independence relationships between variables, and the presence of arcs may represent the existence of direct dependence relationships (if a causal interpretation is given, then the arcs signify the existence of direct causal influences between the linked variables). The quantitative component is a collection of uncertainty measures, which give idea of the strength of the dependencies. In the literature, we can find

*Author to whom correspondence should be addressed. e-mail: lci@decsai.ugr.es

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 12, 495–522 (1997)
 CCC 0884-8173/97/070495-28

different formalisms to manage the uncertainty in belief networks, for example probability[1] (which is the dominant approach), possibility[2] or probability intervals.[3]

Once a complete belief network has been built, it constitutes an efficient device to perform inferences.[1,4] However, there still remains the previous problem of building such a network, i.e., to provide the graph structure and the numerical parameters necessary for characterizing the network. So, an interesting task is then to develop automatic methods capable of learning the network directly from raw data, as an alternative or a complement[5] to the (difficult and time-consuming) method of eliciting opinions from experts.

A usual assumption is to consider the data as being a good representation of the problem, and then any learning algorithm tries to find the network that fits the data better, according to some specified criterion. We can find methods that recover the network by means of scoring metrics,[6–9] and methods based on independence tests.[1,10–13] The former compute a numerical scoring (which may be based on different principles, such as entropy, bayesian or minimum description length) reflecting the goodness-of-fit of the structure to the data, whereas the latter attempt to recover the network that represents most of the independences observed in the data. Anyway, general algorithms for learning belief networks involve a great computational cost: it has been reported that this problem is NP-Hard.[14]

In this article our interest is focused on studying the methods based on independence criteria. The main reason for selecting these methods is that we can obtain general procedures for learning belief networks, regardless of the formalism used for managing the uncertainty. In other words, we consider independence statements as abstract concepts, reflecting qualitative instead of quantitative properties of the problem, and therefore less dependent on the numerical parameters represented in the network. So, these methods could also be used for learning belief networks in cases where the underlying uncertainty model is different from probability theory (provided that we have an appropriate concept of independence within this formalism, see Refs. 3, 15, 16, and 17). However, learning methods based on independence criteria present two main drawbacks: (1) they usually need an exponential number of conditional independence tests, and (2) these tests involve a great number of variables. Note that testing a conditional independence statement between two variables requires a time which is exponential with the number of variables used in the conditioning set.

Considering that the problem of learning general belief networks belongs to the class of NP-Hard problems, and that any attempt for reducing the complexity is welcome, we focus on the following question: suppose that we have previous knowledge about the kind of model to be recovered. Then, is it possible to obtain an efficient learning algorithm using this information? To be more specific, if the given information limits the type of graph structure to be recovered, may this be used in order to design faster learning algorithms? This approach has been successfully used for singly connected networks (trees and polytrees), where efficient learning algorithms have been found.[18–20] Furthermore, since for singly connected networks we can find efficient inference algorithms (using purely local

propagation techniques), these structures have been extensively used in order to approximate general models; the price we have to pay is less expressive power, since the type of independence relationships that may be represented is much more restricted for singly connected networks than for general multiply connected networks. This idea of a priori limiting of the kind of graph to be used has also been applied for more general structures in different contexts: for example, in classification problems, automatic classifiers can be built by learning restricted networks,[21] and for healthcare problems we may find specific learning algorithms, too.[10]

In this article, our interest is focused on a specific class of directed acyclic graphs, those called *simple graphs.*[10] Simple graphs can represent a richer set of independence relationships than singly connected graphs; particularly, conditional independence statements of high order [even $O(n-2)$] can be represented. Moreover, there are algorithms that, using the independence relationships represented in a simple graph, give rise to fast inference procedures.[22] These properties make simple graphs appealing. Our main reasons for carrying out this study are the following: firstly, to show how specific independence properties for simple graphs can be used to obtain efficient learning algorithms; moreover, the methodology presented probably might be generalized for learning other kinds of simplified graphical structures. Secondly, since the problem of learning is computationally expensive, we consider that any attempt to obtain fast learning procedures is worth striving for. Additionally, considering that testing independence in different frameworks to represent the uncertainty may be quite expensive,[1,15,16,17] it would be interesting to obtain learning algorithms in which the use of high order independence tests is not necessary. These algorithms would allow us to efficiently learn more reliable belief networks.

The rest of the article is organized as follows: in the next section we briefly recall how independence statements can be represented in belief networks. Then, in Section III, a review of simple graphs, together with some basic concepts and results used throughout the article, are presented. Section IV studies specific independence properties that hold in simple graphs. These properties permit us to develop, in Section V, a learning algorithm that, whenever the model can be represented by a simple graph, recovers the correct structure using only zero- and first-order conditional independence tests. In Section VI, we discuss how to use the algorithm when we do not know in advance whether the model can really be represented by means of a simple graph. Section VII contains the concluding remarks and some proposals for future work. Finally, as the proof of the stated results is mainly technical, and somewhat long and/or cumbersome, we have preferred to group all of the proofs together into an Appendix, aiming to ease the reading of the article, but without sacrificing mathematical soundness.

II. PRELIMINARIES

Belief networks[1] allow us to represent our knowledge about a given domain of knowledge by means of directed acyclic graphs. In an abstract way, a belief network can be considered as a representation of a *Dependency Model*, i.e., a

pair $M = (U, I)$, where U is a set of variables and I is a rule which assigns truth values to the predicates $I(X|Z|Y)$, (read "X is independent of Y, given Z"), where X, Y, and Z are disjoint subsets of variables in U. The intended interpretation of $I(X|Z|Y)$ is that having observed Z, no additional information about X could be obtained by also observing Y. A criterion for testing independence statements is an essential component of any dependency model. For example, a probability distribution can be considered as a dependency model, where the predicate I is defined through the concept of stochastic independence, i.e., $I(X|Z|Y)$ holds if and only if $P(\mathbf{x}|\mathbf{y}, \mathbf{z}) = P(\mathbf{x}|\mathbf{z})$, whenever $P(\mathbf{y}, \mathbf{z}) > 0$, for every instantiation $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ of the subsets of variables X, Y, Z respectively. Throughout the article, subsets of variables will be denoted by capital letters, whereas single variables will be represented by lower case letters. The connection between belief networks and dependency models may be stated by means of the so-called d-separation criterion,[1,23] which may be considered as a graphical definition of conditional independence (which turns a belief network into a dependency model itself).

DEFINITION 1. *Given a directed acyclic graph G, a trial c (a trail in a directed graph is a sequence of adjacent nodes, the direction of the arrows does not matter) from node x to node y is said to be blocked by the set of nodes Z, if there is a node* $z \in c$ *such that, either*

- $z \in Z$ *and arrows of c do not meet head to head at z, or*
- $z \notin Z$, *nor has z any descendants in Z, and the arrows of c do not meet head to head at z.*

A trail that is not blocked by Z is said to be opened by Z.

DEFINITION 2 (d-separation). *Let X, Y, and Z be three disjoint subsets of nodes in a dag G. Then, X and Y are said to be d-separated (or graphically independent) by Z if all trails between the nodes in X and the nodes in Y are blocked by Z. Otherwise, X and Y are graphically dependent, given Z.*

The *skeleton* of a dag G is the undirected graph obtained from G by replacing the arrows by edges. When we speak about cycles in a dag, we always refer to undirected cycles in the skeleton of G (a dag cannot contain directed cycles). Note that in a dag every cycle must contain at least one *head to head connection,* i.e., a subgraph of the form $x \rightarrow z \leftarrow y$; the middle node in a head to head connection is called a *head to head node*.

We say that $I(X|Z|Y)$ is a conditional independence statement of *order* k if the cardinality of the conditioning set Z is equal to k. If it is also $\neg I(X|W|Y)$ for any set $W \subset Z$ of cardinality less than k, then we say that $I(X|Z|Y)$ is an *essential* conditional independence statement of order k.

A belief network may also represent quantitative knowledge. Let us suppose that this knowledge is probabilistic. In that case, for each variable, x_i, we need conditional probability distributions $P(\mathbf{x_i}|\mathbf{\Pi}(\mathbf{x_i}))$, where $\mathbf{x_i}$ represents any assignment of values to the variable x_i, $\Pi(x_i)$ is the parent set of x_i in the network,

and $\mathbf{\Pi}(\mathbf{x_i})$ denotes any assignment of values to the variables in the set $\Pi(x_i)$. Then, the joint probability distribution, P, can be obtained by means of the following expression:

$$P(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}) = \prod_{i=1}^{n} P(\mathbf{x_i}|\mathbf{\Pi}(\mathbf{x_i})).$$

If we start out from a joint probability distribution P (a dependency model), we can look for a belief network G that is a ''good'' graphical representation of P. In that case, we may find different situations:[1] If every d-separation condition in G corresponds to a valid conditional independence relationship in P, then G is said to be an *I-map* of P; if every conditional independence relationship in P corresponds to a d-separation condition in G, then G is a *D-map* of P; finally, if G is both an I-map and a D-map of P, then we say that G is a *perfect map* of P and that P is *dag-isomorphic*. Throughout this article, we shall always consider dag-isomorphic dependency models.

III. SIMPLE GRAPHS: DEFINITIONS AND BASIC RESULTS

We consider a particular case of dependency models, those dag-isomorphic models that can be represented by means of *Simple Graphs.*[10] Formally, a simple graph is defined as follows:

DEFINITION 3 (Simple Graph). *A dag G is said to be simple if, and only if, every pair of nodes with a common direct child have no common ancestor nor is one an ancestor of the other.*

Graphically, in simple graphs, only a special kind of (undirected) cycles can appear, those containing at least two head to head connections. Considering the d-separation criterion, simple graphs are the class of dags where the common parents of any variable are always marginally independent (i.e., d-separated by the empty set) between each other. So, the following characterization of simple graphs is immediate:

THEOREM 1. *Let G be a dag. The following statements are equivalent:*

1. *G is a simple graph.*
2. *All the cycles in G contain at least two head to head connections.*
3. *Every pair of nodes in G with a common direct child are d-separated by the empty set.*

Singly connected graphs (trees and polytrees) are specific cases of simple graphs. Directed bipartite graphs are also examples of simple graphs. It is interesting to note that, in a simple graph, essential conditional independence statements of any order can be represented (observe that in the case of singly connected

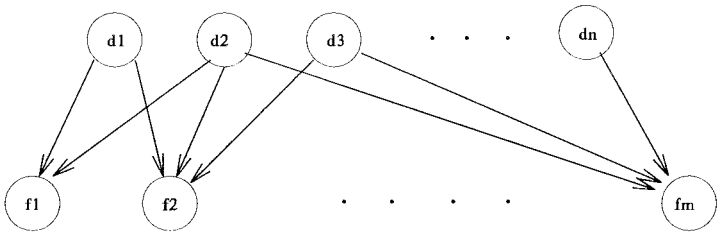

Figure 1. Simple graph representing diseases and findings.

graphs, only essential conditional independence assertions of order zero and one are permitted).

In daily life, we can find models that may be represented by means of simple graphs. In general, and considering the definition above, any model where there are no correlated sources of evidence can be represented by a simple graph. For example, simple graphs have been used to represent the relationships between diseases and findings[22] in clinical models. In that case, we represent the fact that diseases are marginally independent, and after knowing the diseases, the findings become conditionally independent (see Fig. 1). Simple graphs have also been used to represent independence relationships in genetics models.[24]

As our interest lies in studying learning algorithms, first we consider the algorithm proposed by Geiger et al.[10] for learning simple graphs. Their algorithm takes as the input a dependency model M, and gives as the output a simple graph that represents the model well.† If such a graph does not exist, the algorithm gives an error code as the output.

Geiger, Paz, and Pearl (GPP) Algorithm

1. Start with a complete undirected graph G.
2. Remove from G every edge $x - y$ such that $I(x|U\backslash\{x, y\}|y)$ holds.
3. Remove from G every edge $x - y$ such that $I(x|\varnothing|y)$ holds.
4. For every pair of adjacent edges $x - y$ and $y - z$ in G, if $I(x|\varnothing|z)$ holds, then direct the edges as $x \rightarrow y \leftarrow z$.
5. Direct the remaining edges without introducing new head-to-head connections. If the resultant graph is not simple, then give an error code as the output.
6. If the resultant graph does not represent the model well, then give an error code as the output. Otherwise, give the resultant graph as the output.

The GPP algorithm uses a polynomial number of conditional independence tests, but for each pair of variables, it needs a conditional independence test of order $n - 2$, with n being the number of variables in the model. Thus, the main

†A graph is said to represent a dependency model well if whenever two nodes x and y are connected by a trail without head to head connections, then x and y are marginally dependent, i.e., $\neg I(x|\varnothing|y)$. This concept could be called marginal D-mapness.

drawback of the algorithm is that if we have to learn the graph from statistical data, each conditional independence test of order $n - 2$ requires a time which is exponential with n. Therefore, although the algorithm is polynomial [order $O(n^2)$] in the number of independence tests, the overall complexity is still exponential. Moreover, to realiably test high order conditional independence statements we would need an enormous amount of data. Therefore, this algorithm would be appropriate only if the conditional independence tests were inexpensive; for example, this is the case if we can obtain the answer by asking an expert for the results of the tests.

Our purpose is to study specific independence properties of simple graphs, in order to avoid some of the practical problems presented in the algorithm above. Some preliminary concepts will be necessary. The following theorem[10,25] characterizes equivalent (or isomorphic) simple graphs:

THEOREM 2. *Two simple graphs G_1 and G_2 are isomorphic if, and only if, they share the same skeleton and the same head-to-head connections.*

Thus, isomorphism represents a theoretical limitation on the ability to identify the directionality of some links, using information about independences; for example, the following structures reflect the same independence assertion, i.e., x and z are marginally dependent, but given y they become conditionally independent:

$$x \leftarrow y \leftarrow z; \quad x \rightarrow y \rightarrow z; \quad x \leftarrow y \rightarrow z$$

Once the head-to-head connections play an important role in simple graphs, the following definition will be used.

DEFINITION 4 (Active Trail). *A trail c linking two nodes x and y in a graph G is said to be active if there is no head-to-head connection in c. Any trail with head-to-head connections is said to be nonactive.*

Using the d-separation criterion, it is clear that any active trail linking two variables x and y is open by the empty set, i.e., x and y are not d-separated by $\emptyset$. We can establish the following classification of the active trails linking any two nodes x and y:

$HT(x, y)$: Those active trails linking x and y with a head connection at x and a tail connection at y, i.e., directed paths from y to x, such as $x \leftarrow \ldots \leftarrow y$.

$TH(x, y)$: Those active trails linking x and y with a tail connection at x and a head connection at y, i.e., directed paths from x to y, such as $x \rightarrow \ldots \rightarrow y$.

$HH(x, y)$: Those active trails linking x and y with a head connection at x and a head connection at y, such as $x \leftarrow \ldots \rightarrow y$. In these trails we can always find a node, z, such that we have directed subpaths from z to x and from z y ($x \leftarrow \ldots \leftarrow z \rightarrow \ldots \rightarrow y$).

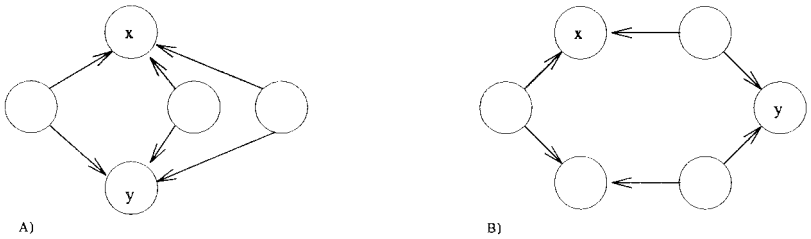

Figure 2. Active and nonactive cycles in a simple graph.

As interesting and useful property of simple graphs is that the existence of an active trail linking two nodes affects to the existence of other active trails connecting these nodes:

PROPOSITION 1. *Let G be a simple graph, and x, y two nodes in G. If there is an active trail c that belongs to either HT*(x, y) or *TH*(x, y), *then c is the only active trail linking x and y in G.*

Using the previous result, we immediately obtain the following corollary:

COROLLARY 1. *Let G be a simple graph, and x, y two nodes in G. If there is more than one active trail in G linking x and y, then all these trails necessarily are of the HH*(x, y) *type.*

Note that when we have more than one active trail of the *HH*(x, y) type, each pair of these trails forms a cycle. This kind of cycle will play an important role in the subsequent development (moreover, they also constitute the only way in which simple graphs can represent essential independence relationships of any order). Therefore, to distinguish them from the other cycles that may be present in a simple graph (see Fig. 2), we shall say that a cycle in a simple graph is an *active cycle* if it contains exactly two head-to-head connections. Any other cycle in a simple graph (which necessarily contains at least three head-to-head connections) is said to be a *nonactive cycle*. We say that there is an active cycle *between* two nodes x and y, if x and y are the only head-to-head nodes in the cycle.

IV. INDEPENDENCE RELATIONSHIPS IN SIMPLE GRAPHS

The aim of this section is to study specific independence properties that hold in simple graphs, which will allow us to develop an efficient learning algorithm in the next section. Remember that we are only considering dag-isomorphic models, so that we can talk about independence and d-separation statements interchangeably.

Pearl[1] identified a set of properties or axioms which have to be verified by any dag-isomorphic dependency model (although they do not constitute a

characterization of these models). These properties are the following (their semantic interpretation is to be found in Ref. 1):

Trivial Independence:
$I(X|Z|\varnothing)$

Symmetry:
$I(X|Z|Y) \Rightarrow I(Y|Z|X)$

Decomposition/Composition:
$I(X|Z|\ Y \cup W) \Leftrightarrow I(X|Z|Y)$ and $I(X|Z|W)$

Weak Union:
$I(X|Z|Y \cup W) \Rightarrow I(X|Z \cup Y|W)$

Contraction:
$I(X|Z|Y)$ and $I(X|Z \cup Y|W) \Rightarrow$
$I(X|Z|Y \cup W)$

Intersection:
$I(X|Z \cup W|Y)$ and $I(X|Z \cup Y|W) \Rightarrow$
$I(X|Z|Y \cup W)$

Weak Transitivity:
$I(X|Z|Y)$ and $I(X|Z \cup \gamma|Y) \Rightarrow$
$I(X|Z|\gamma)$ or $I(\gamma|Z|Y)$

Chordality:
$I(\alpha|\gamma \cup \delta|\beta)$ and $I(\gamma|\alpha \cup \beta|\delta) \Rightarrow$
$I(\alpha|\gamma|\beta)$ or $I(\alpha|\delta|\beta)$

Obviously, these properties, which hold for dags, must also hold for simple graphs. However, due to graphical restrictions imposed, we can find additional independence properties, specific for simple graphs, like for example Weak Semi-strong Union and Weak Atringularity,[18] defined by means of

Weak Semi-strong Union:
$I(\alpha|Z|\beta)$ and $\neg I(\alpha|\varnothing|\beta) \Rightarrow I(\alpha|U\backslash\{\alpha, \beta\}|\beta)$
Weak Atringularity:
$\neg I(\alpha|Z|\gamma), \forall Z \subseteq U\backslash\{\alpha, \gamma\}$ and $\neg I(\gamma|Z|\beta), \forall Z \subseteq U\backslash\{\gamma, \beta\} \Rightarrow$
$I(\alpha|\varnothing|\beta)$ or $\exists Z_0$ s.t. $I(\alpha|Z_0 \cup \gamma|\beta)$
being $Z_0 \subseteq U\backslash\{\alpha, \beta, \gamma\}$

It is interesting to note that these properties have been implicitly used by the GPP recovery algorithm. For example, using Weak Semi-strong Union, we can be sure that, after step 3, the GPP algorithm will have correctly identified the skeleton of the simple graph, and using Weak Atringularity, the head-to-head connections may be detected through marginal independence tests.

Because of our assumption of dag-isomorphy, it is clear that the existence of a conditional independence relationship between two variables x and y, $I(x|Z|y)$, is closely related with the presence or absence of active trails in the graph. Thus, in order to obtain independence statements, studying how an active trail can be blocked becomes an important question. For simple graphs, as happens in general dags, we know that whenever there is an active trail c linking two variables, then they are marginally dependent i.e., $\neg I(x|\varnothing|y)$. Moreover, using d-separation, we find that this trail is blocked after instantiating any node z in c. But, in contrast to what happens for general dags, the following proposition proves that there is no nonactive trail linking x and y which becomes open after instantiating z.

PROPOSITION 2. *Let G be a simple graph and let c be an active trail linking two nodes x and y in G. Then, c is blocked by any node z in this trail and there is no nonactive trail linking x and y which is opened by z.*

An important consequence of this proposition is that, in order to establish an independence (d-separation) relationship between two nodes, x and y, in a simple graph, it is sufficient to instantiate a node for each active trail connecting x and y:

PROPOSITION 3. *Let G be a simple graph, and let x, y be two nonadjacent nodes in G, such that there exists at least one active trail linking x and y. Let* $\rho_x(y)$ *be the set of nodes connected directly to x in any active trail linking x and y. Then* $I(x|\rho_x(y)|y)$ *holds in G.*

From Propositions 1 and 3 we can deduce an interesting property that relates independence statements with local topological relationships between nodes in the graph: given a simple graph, if node x is known, then we find that its parents are conditionally independent of its children. Moreover, the fulfilment of this property is sufficient to guarantee that a given structure is a simple graph. Therefore, this property characterizes simple graphs.

PROPOSITION 4. *Let G be a directed acyclic graph. Then G is a simple graph if, and only if, for every node x in G, and for every pair of nodes* p_x, h_x, *with* $p_x \in Parents_x$ *and* $h_x \in Children_x$, $I(p_x|x|h_x)$ *holds.*

This property allows us to test whether a given dag is a simple graph or not, using only independence tests among triplets of variables, i.e., independence tests of order one. A result similar to that of proposition 4, but using the ancestors and descendants of a node, instead of its parents and children, can also be established (the proof is similar, so we omit it from the appendix):

PROPOSITION 5. *Let G be a directed acyclic graph. Then G is a simple graph if, and only if, for every node x in G,* $I(Ancestors_x|x|Descendant_x)$ *holds.*

The fact that only one variable is necessary to establish a conditional independence relationship between the set of ancestors and the set of descendants of this variable, raises the question of the possibility of recovering a simple graph using only conditional independence relationships of order zero and one (as happens for singly connected graphs[18,20]). In the next section we show that this question has a positive answer.

V. THE LEARNING ALGORITHM

In this section, we consider models isomorphic to simple graphs, and develop an efficient recovery algorithm. From Theorem 2, we know that every simple graph isomorphic to the model must share the same skeleton and the same head-to-head connections. Using this property, a simple graph representing the model can be obtained by first recovering the skeleton and next detecting the head-to-head connections, using the fact that in a simple graph any two variables with a common direct descendant must be marginally independent.

A natural approach for recovering the skeleton would be to locally look for the set of nodes directly connected with each variable x in the model (its parents and its children) and then build the structure by fusing all these components. Thus, our first task is to detect whether there is a direct connection (or not) between any two variables x and y. A general rule, used by a number of algorithms for learning belief networks based on independence relationships, is the following: whenever given any two variables, x and y, we can find a set of variables Z such that the independence relationship $I(x|Z|y)$ holds, then x and y cannot be adjacent in any graph representing the model.[12,13] This gives rise to learning algorithms that, starting from a complete graph, look for independence relationships between variables which, if found, make it possible to remove the corresponding edge from the graph. The GPP algorithm uses this methodology, in combination with specific properties of simple graphs that allow us to identify the candidate conditioning (d-separating) sets Z (in this case either $\varnothing$ or $U\backslash\{x, y\}$) directly, without any search. Our approach for recovering the network will be slightly different: we shall not blindly search for d-separating sets, nor will we use the ''obvious'' (and computationally expensive) set $U\backslash\{x, y\}$. Instead, we shall determine a set of rules that allows us to identify that a set $Z \subseteq U\backslash\{x, y\}$ verifying that $I(x|Z|y)$, exists, i.e., that there is no direct connection between x and y in the simple graph.

From the discussion in the previous section, it is clear that zero- and first-order conditional independence statements play an important role in simple graphs, and then, in order to find the skeleton, the following approach will be used:

(i) For each node x, select from $U\backslash\{x\}$ the subset of variables, I_x^{0-1}, such that there are neither zero- nor first-order conditional independence relationships between x and each variable in I_x^{0-1}, i.e.,

$$I_x^{0-1} = \{y \in U\backslash\{x\} \mid \neg I(x|\varnothing|y) \text{ and } \neg I(x|z|y) \forall z \in U\backslash\{x, y\}\}$$

(ii) Then, remove from I_x^{0-1} those variables y such that there is a conditional independence relationship between x and y of an order greater than or equal to two.

Considering Weak Semi-strong Union, an immediate approach to perform the second step could be to test, for each variable y in I_x^{0-1}, whether $I(x|U\backslash\{x, y\}|y)$ holds (as the GPP algorithm does), but in this case we have to pay the high computational cost necessary to perform these tests. We shall use an alternative method that permits us to find these nodes more efficiently. The next proposition, which characterizes which nodes may belong to I_x^{0-1}, is useful for our purposes:

PROPOSITION 6. *Let G be a simple graph, and let x, y be any two nodes in G. Then, y belongs to I_x^{0-1} if, and only if, there is either an active cycle between x and y, or a direct connection between x and y in G.*

According to the proposed scheme for identifying the set of nodes adjacent to x, after performing the first step, we know that the set I_x^{0-1} contains only nodes adjacent to x and nodes that are conditionally independent from x given a set

of cardinality greater than or equal to two. Now, we have to exclude this second type of nodes from I_x^{0-1}, thus obtaining the correct set of nodes adjacent to x. So, for each node y in I_x^{0-1} we have to look for a *separator set*, $\Phi_x(y)$, $|\Phi_x(y)| \geq 2$, with minimal cardinality (and therefore requiring less computational cost), such that $I(x|\Phi_x(y)|y)$ holds. If we can find such a set, we can remove y from I_x^{0-1}, otherwise, y would really be a node adjacent to x. The result in Proposition 6 allows us to characterize the nodes that we want to exclude from I_x^{0-1} as those nodes y such that there is an active cycle between x and y. This property will make it possible to determine whether $\Phi_x(y)$ exists or not, without the need for effectively performing any test $I(x|\Phi_x(y)|y)$. The process is as follows: for each $y \in I_x^{0-1}$, we shall start from an initial set of nodes which are candidates to form part of the separator set $\Phi_x(y)$; next we shall refine this set in successive steps (by removing nodes from it). If some of these refined sets becomes empty, this will mean that there is no set of nodes capable of separating x and y [i.e., the separator set $\Phi_x(y)$ does not exist], hence the edge linking x and y is a true edge in the skeleton. However, if, at the end of this refining process, the final set is not empty, this will mean that we can find a set of nodes separating x and y, hence the edge between x and y will be eliminated. Now, let us describe this process more formally:

DEFINITION 5. *Let G be a simple graph, and let x, y be nodes in G such that $y \in I_x^{0-1}$.*

1. *The initial set $K_x(y)$ is:*

$$K_x(y) = \{w \in I_x^{0-1} \backslash \{y\} \mid \neg I(w|x|y)\}.$$

2. *The set of candidate nodes (to separate x and y), denoted by $\Omega_x^*(y)$, is:*

$$\Omega_x^*(y) = \{w_i \in K_x(y) \mid \exists w_j \in K_x(y) \text{ verifying } I(w_i|\varnothing|w_j) \text{ and } \neg I(w_i|y|w_j)\}.$$

3. *The final set of candidate nodes, denoted $\Omega_x(y)$, is:*

$$\begin{aligned}\Omega_x(y) = \Omega_x^*(y) \backslash \{&w_i \text{ such that } \exists \alpha \in I_x^{0-1} \text{ with } \neg I(\alpha|\varnothing|w_i) \text{ verifying either} \\ &a)\ I(\alpha|\varnothing|x) \text{ and } \neg I(\alpha|y|x), \text{ or} \\ &b)\ \neg I(\alpha|\varnothing|x) \text{ and } I(\alpha|y|x) \text{ and } \neg I(\alpha|y|w_i)\}\end{aligned}$$

Observe that all these sets, $K_x(y)$, $\Omega_x^*(y)$ and $\Omega_x(y)$ are defined using only independence statements of order zero and one, so that they can be calculated efficiently and reliably from data.

The intuition behind Definition 5 is the following: the initial set $K_x(y)$ excludes from consideration those nodes w which clearly cannot be part of any minimal set d-separating x from y: if $I(w|x|y)$ were true and w were in a set d-separating x and y, i.e., if $I(x|Z \cup w|y)$ were true, then $I(x|Z'|y)$, for some $Z' \subseteq Z$, would also be true, and therefore w does not need to be considered. On the other hand, taking into account the independence relationships defining the set $\Omega_x^*(y)$, it may be easily found (see Lemma 1 in the appendix) that whenever an active cycle exists between x and y, the candidate set $\Omega_x^*(y)$ is not empty [in

fact the parents of x in the cycle always belong to $\Omega_x^*(y)$]. However, it is also possible to obtain a nonempty set $\Omega_x^*(y)$ even if there is a direct connection between x and y. So, if $\Omega_x^*(y)$ [or $K_x(y)$] becomes empty, we can be sure that x and y are adjacent, although we do not know what happens if $\Omega_x^*(y)$ is not empty. But in that case, the independence relationships considered and the fact that the model can be represented by a simple graph, limit the structures that might be considered [see Lemmas 2 and 3 in the appendix]. The more refined set $\Omega_x(y)$ allows us to discriminate unambiguously between nodes adjacent to x and nodes forming an active cycle with x, as the following proposition shows:

PROPOSITION 7. *Let G be a simple graph, and let x, y be nodes in G such that $y \in I_x^{0-1}$. Then, there exists an active cycle between x and y in G if, and only if, $\Omega_x(y) \neq \varnothing$.*

Now, we have the tools necessary to recover the skeleton of the model: for each node we can determine the set of nodes directly connected to it, i.e., its *Neighbors,* and then fuse these components. The proposed algorithm is given below.

Recovery Algorithm: CH1

1. For each node x:
   (a) Calculate I_x^{0-1}.
   (b) Neighbor$(x) = \varnothing$.
   (c) For each $y \in I_x^{0-1}$:
      i. Calculate $K_x(y)$. If $K_x(y) = \varnothing$ go to step 1.c.iv.
      ii. Calculate $\Omega_x^*(y)$. If $\Omega_x^*(y) = \varnothing$ go to step 1.c.iv.
      iii. Calculate $\Omega_x(y)$. If $\Omega_x(y) = \varnothing$ go to step 1.c.iv. Else go to step 1.c.
      iv. Neighbor(x) = Neighbor$(x) \cup \{y\}$.
2. Fuse every Neighbor(x), to obtain G.
3. For each node x:
   (a) For each pair $y, z \in$ Neighbor(x), If $I(y|\varnothing|z)$ holds, put the nodes y, z as parents of x.
4. Direct the remaining edges without introducing new head-to-head connections.

Taking into account the previous results, it is easy to show that the algorithm gives as the output a simple graph isomorphic to the dependency model:

PROPOSITION 8. *Let M be a dependency model isomorphic to a simple graph, and let $\mathcal{L}$ be a list of marginal and first-order conditional independence relationships obtained from M. Let G be the graph obtained by the Algorithm CH1. Then M is isomorphic to G.*

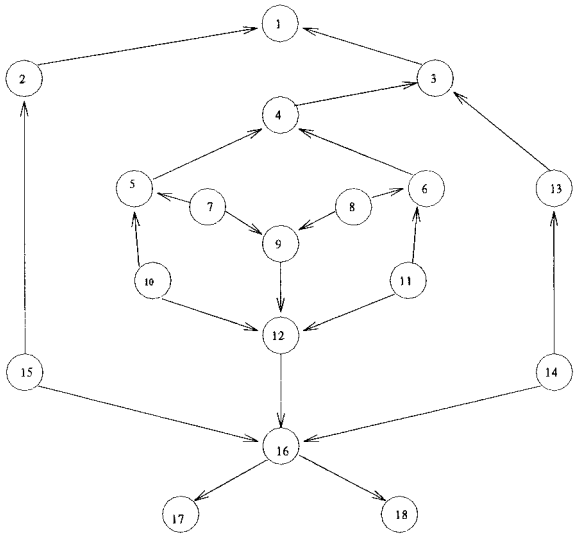

Figure 3. Simple graph.

The following example shows how the recovery algorithm works. We use the simple graph in Figure 3 as a hidden model, where the set of variables is $U = \{1, \ldots, 18\}$ and we study how the algorithm constructs the set of neighbors for nodes 12 and 16. For note 16, we find that $I_{16}^{0-1} = \{1, 3, 12, 14, 15, 17, 18\}$. We can construct $K_{16}(1) = \{3, 12, 14, 15\}$ and then, by considering that $3 \in K_{16}(1)$, $I(3|\varnothing|15)$ and $\neg I(3|1|15)$, we conclude that $3 \in \Omega_{16}^{*}(1)$. Using analogous reasoning, we obtain the set $\Omega_{16}^{*}(1) = \{3, 12, 14, 15\}$. Similarly, $\Omega_{16}^{*}(3) = \{12, 14\}$. The other sets $\Omega_{16}^{*}(.)$ are equal to the empty set. The next step is to construct the sets $\Omega_{16}(1)$ and $\Omega_{16}(3)$; in this case, we find that there is no element that may be excluded from $\Omega_{16}(1)$ and $\Omega_{16}(3)$ and then, nodes 1, 3 are eliminated, so Neighbor(16) = {12, 14, 15, 17, 18}.

To conclude the example, we now consider node 12. We find that $I_{12}^{0-1} = \{4, 5, 6, 9, 10, 11, 16\}$ and obtain the sets $\Omega_{12}^{*}(4) = \{5, 6, 9, 10, 11\}$, $\Omega_{12}^{*}(5) = \{9, 10\}$, $\Omega_{12}^{*}(6) = \{9, 11\}$ and $\Omega_{12}^{*}(9) = \{5, 6\}$, with the other sets $\Omega_{12}^{*}(.)$ being empty. Again, we find that the sets $\Omega_{12}(.)$ are all nonempty, except for $\Omega_{12}(9)$, in which nodes 7 and 8 verify the conditions that exclude nodes 5 and 6, respectively. Finally, we find that neighbor(12) = {9, 10, 11 16}.

As a direct consequence of Proposition 8, we can deduce the following theoretical result, which gives a new condition of isomorphy for simple graphs.

THEOREM 3. *Let G_1, G_2 be two simple graphs. Then, the following conditions are equivalent:*

1. *G_1 and G_2 are isomorphic.*
2. *G_1 and G_2 have the same marginal and first-order conditional independence relationships.*

We conclude this section with some comments about the efficiency of the recovery algorithm:

- The algorithm can be implemented locally in part, the calculations are independent for each variable: only the transition from $\Omega_x^*(y)$ to $\Omega_x(y)$ cannot be made in a purely local way. This would make it possible to use distributed computation.
- The algorithm needs a polynomial number of independence tests, $O(n^3)$.
- The independence tests needed are marginal independence test and first-order conditional independence tests. Therefore, we can calculate the tests in polynomial time.
- Knowing the results of the tests, the algorithm obtains a simple graph in polynomial time, $O(n^4)$.

VI. DETECTING SIMPLE GRAPH ISOMORPHISMS

In this section we study the following problem: ''Given a dependency model isomorphic to a general directed acyclic graph, but not necessarily isomorphic to a simple graph, can we efficiently detect whether a simple graph that fits the model exists?'' We propose an algorithm that, by reducing the number of conditional independence tests of order greater than one to the minimum (which may be zero), answers this question. If the model may be represented by a simple graph, the algorithm gives its graphical structure as the output, otherwise the output is an error code. Moreover, the algorithm is still polynomial in the number of independence tests needed.

If we know in advance that the model can be represented by a simple graph, the previous algorithm CH1 recovers the structure (and does so efficiently). The problem arises when we do not know whether or not the underlying model is isomorphic to a simple graph, although we assume that it is dag-isomorphic. In that case, we have to check (1) whether the output of the algorithm is a simple graph and also (2) whether this graph is isomorphic to the model.

We propose the following modified recovery algorithm, CH2:

Recovery Algorithm: CH2

1. For each x:
   (a) Calculate I_x^{0-1}.
   (b) Neighbor(x) = $\varnothing$.
   (c) For each $y \in I_x^{0-1}$:
      i. Calculate $K_x(y)$. If $K_x(y) = \varnothing$ go to step 1.c.iv.
      ii. Calculate $\Omega_x^*(y)$. If $\Omega_x^*(y) = \varnothing$ go to step 1.c.iv.
      iii. Calculate $\Omega_x(y)$. If $\Omega_x(y) = \varnothing$ go to step 1.c.iv. Else go to step 1.c.
      iv. Neighbor(x) = Neighbor(x) $\cup$ $\{y\}$.
2. Fuse every Neighbor(x), to obtain G.
3. For each x:
   (a) For each pair $y, z \in$ Neighbor(x), If $I(y|\varnothing|z)$ holds, put the nodes y, z as parents of x.

4. Checking 0-1 Independence Statements:
   (a) For each x, y, z in G such that the edges x—y and y—z belong to G:
      i. If $(x \rightarrow y \leftarrow z) \in G$ and $\neg I(x|\emptyset|z)$ in the model, then give as the output an error code.
      ii. If $(x \leftarrow y \rightarrow z)$, $(x \rightarrow y \leftarrow z) \notin G$ and $\neg I(x|y|z)$ in the model, then give as the output an error code.
5. Direct the remaining links without introducing new head-to-head connections. If the orientation is not feasible, then give as the output an error code.
6. For each $\Omega_x(y) \neq \emptyset$:
   If $\neg I(x|\Omega_x(y) \cap \text{Parents}_x|y)$ in the model, then give an error code as the output.

Observe that the first three steps in CH2 are the same as in the algorithm CH1, i.e., we are using the same methodology, but now we need additional steps, because we do not have the additional information to ensure us that the model may be represented by a simple graph. In that case, after executing steps 1, 2, and 3 in the algorithm CH2, we cannot be sure that the resultant graph is simple, so we have to check it. Step 4 does this implicitly: note that if the model can be represented by a simple graph, then all the head-to-head connections in the output structure must represent true marginal independence statements in the model, and any conditional independence relationship between parents and children for a node x, i.e., $I(p_x|x|h_x)$, must also be a valid independence relationship in the model. These are the reasons for including steps 4(a.i) and 4(a.ii). Proposition 9 states that we can guarantee that the output structure, obtained after executing step 5 in the algorithm CH2, is a simple graph (and we can check this by using only independence tests of order zero and one).

PROPOSITION 9. *Let M be a dag-isomorphic dependency model. If after executing step 5 of the algorithm CH2 we do not obtain an error code, then the output structure, G, is a simple graph.*

Let us suppose that, after executing step 5, the algorithm does not fail, i.e., we get a simple graph G. This fact still does not guarantee that the model is isomorphic to a simple graph. For example, for the dag G_M displayed in Figure 4, the graph, G, obtained by the algorithm after step 5 (also displayed in the same Figure) is simple, but obviously G_M and G are not isomorphic. In order to guarantee that G represents the model, we need an additional check. If the model cannot be represented by a simple graph, then, in the nonsimple dag isomorphic to the model M, G_M, there has to exist a cycle c_M having only one head-to-head node; so, every pair of nodes x, $y \in c_M$ are marginally dependent, i.e., $\neg I(x|\emptyset|y)$. If all the direct connections in G_M, involving nodes from c_M, would correspond to arcs in G, then we would also have a cycle c in G, involving the same nodes as c_M. However, this cycle c cannot contain only one head-to-head node (because we suppose that the algorithm did not fail, hence G is simple),

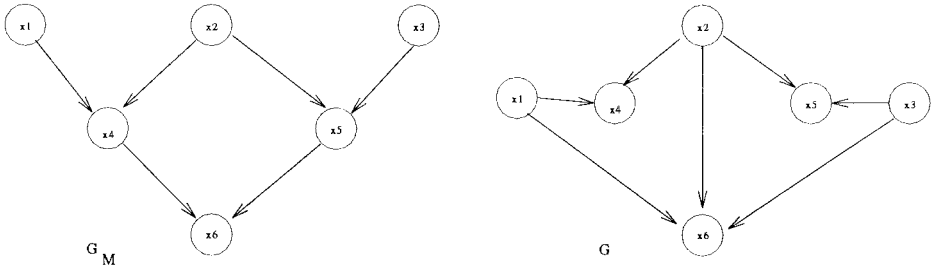

Figure 4. 0-1 Isomorphic structures G_M and G.

it has to contain at least one additional head-to-head node; but the algorithm only introduces head-to-head nodes after checking the corresponding marginal independence (step 3.a), and we know that no marginal independence exists. Therefore, if the model is not isomorphic to a simple graph, we conclude that some link has been eliminated improperly by the algorithm, i.e., a node y has been incorrectly excluded from the set of neighbors of some node x. Looking at the algorithm, we see that links are eliminated at steps 1(a) or 1(c.iii). In the first case, the independence relationships are tested, and so the edges are eliminated correctly. However, in step 1(c.iii), a link x—y is eliminated [because $\Omega_x(y) \neq \emptyset$] assuming that we are considering simple graphs. The problem arises because of this assumption. Proposition 10 shows that, by testing the independence statement $I(x|\Omega_x(y) \cap \text{Parents}_x|y)$, with Parents_x being the set of parents for node x in the output graph G (step 6 in the CH2 algorithm), we can determine whether G represents the model or not. Observe that if the independence relationship $I(x|\Omega_x(y) \cap \text{Parents}_x|y)$ holds, the set $\Omega_x(y) \cap \text{Parents}_x$ is a set of minimal cardinality d-separating x and y, i.e., this set coincides with the *separator set* $\Phi_x(y)$. Although we could test this relationship before eliminating the edge, taking into account that higher order independence tests must be used in this process, they are delayed until the end.

PROPOSITION 10. *Let M be a dag-isomorphic dependency model. Then, M is isomorphic to a simple graph if, and only if, the algorithm CH2 gives a simple graph as its output.*

Finally, since step 6 of the algorithm CH2 needs to use higher order independence tests, we consider the following question: what independence properties can be obtained if we do not execute step 6? For example, consider the dag-isomorphic dependency model G_M, represented in Figure 4, and the simple graph G (also displayed in Fig. 4) obtained after executing step 5 of the algorithm.

In this case, we can easily see that the graph G obtained by the first five steps of CH2 (supposing that it does not fail) is neither an I-map nor a D-map of the model: for example, for the model in Figure 4, we find that $I(x_6|x_4, x_5|x_2)$ is true in the model, but is not true in the graph G, hence G is not a D-map;

moreover, we know that $I(x_4|x_1, x_2|x_6)$ is not true in the model, but is true in G, so G is not an I-map.

Thus, we conclude, as was to be expected, that using only zero- and first-order independence relationships, we cannot always ensure that higher order independences are preserved. The next question is to consider what happens with the relationships used by the algorithm, i.e., the zero- and first-order independence statements? We can prove that the simple graph given as the output by the first five steps of CH2 has the same zero- and first-order independence relationships as the model, and therefore we can say that these structures are *0-1 Isomorphics* (although this result is quite intuitive, its formal proof is very long and complicated, so we do not include it; a complete proof is to be found in Ref. 26).

Finally, let us briefly summarize the different possibilities that we may face:

- If we know that the input model can be represented by a simple graph, then the algorithm CH1 (steps 1..3 and 5 of algorithm CH2) recovers its structure using only zero- and first-order conditional independence tests.
- If, on the other hand, we do not know whether the model may be represented by a simple graph, then the algorithm CH2 (steps 1..5) efficiently recovers a simple representation of the model, whenever it exists, i.e., a simple graph that has the same zero- and first-order independence relationships as the model.
- If step 6 of the algorithm CH2 is also executed, then we can distinguish whether the given model is isomorphic to a simple graph (and in this case the simple graph representing the model is obtained as the output) or not by using some additional independence tests of an order greater than one.

VII. CONCLUDING REMARKS

In this article we have presented an efficient algorithm for recovering simple graphs, based on a detailed study of the properties of the independence relationships that these structures can support. When we know that the hidden graph is in fact a simple graph, the algorithm recovers a graph isomorphic to it, using only zero- and first-order conditional independence tests. We have also discussed how the algorithm can be used for learning when no previous information about the hidden graph is known. In that case, it can be detected whether this graph is simple or not by using a polynomial number of independence tests, some of them of an order greater than one. The properties of the simple graphs obtained without carrying out higher order independence tests have been studied too. Our algorithm improves the complexity and reliability of previous results, at the cost of adding the assumption of dag-isomorphism. As the algorithm is based on an abstract concept of independence, it can be applied for learning belief networks using formalisms for representing the uncertainty which is not necessarily probabilistic[1,16,17] In some of these uncertainty models, independence tests can be computationally very expensive, so the use of zero- and first-order conditional independence tests allows us to obtain faster and more reliable results.

In future work, we should study how simple graphs can be used to approximate general dags. Since a richer set of independence relationships may be

represented in simple graphs, we expect to obtain better approximate models than by using singly connected graphs. An axiomatic characterization of simple graphs, in terms of independence properties, might be worth studying: our purpose would be to find a finite set of independence properties that, as for other kinds of graphs (such as singly connected networks,[18] chordal graphs[27] or undirected graphs[1]), ensures that a dependency model is equivalent to a simple graph. Another direction for research would be to design algorithms that, also using previous information to restrict the type of underlying structure, would allow us to recover more general graphs.

This work has been supported by the DGICYT under Project No. PB92-0939.

APPENDIX

In this appendix, we include the proofs of all the results stated in the main body of the article.

PROPOSITION 1. *Let G be a simple graph, and x, y two nodes in G. If there is an active trail c that belongs to either $HT(x, y)$ or $TH(x, y)$, then c is the only active trail linking x and y in G.*

Proof. Let us suppose that there are at least two active trails linking x and y, c_1 and c_2, and assume that $c_1 \in TH(x, y)$ (the case $c_1 \in HT(x, y)$ is similar).

(a) Suppose that c_1 and c_2 have no node in common, except x and y. In this case, we find that: (1) if $c_2 \in TH(x, y)$ or $c_2 \in HH(x, y)$, then there are two direct parents for y with either a common ancestor or one parent is an ancestor of the other, which is a structure forbidden in simple graphs; (2) if $c_2 \in HT(x, y)$, we obtain a directed cycle, also forbidden because G is a dag. In any case, we obtain a contradiction.

(b) Suppose that there is at least a node α, ($\alpha \neq x$ and $\alpha \neq y$) belonging to both c_1 and c_2. In that case, the active trail c_2 can be obtained by combining trails like:

- $c_2'(\alpha_i, \alpha_j)$, with α_i and α_j being the only nodes in c_2' that belong to c_1, and at least one of α_i, α_j being different from x and y.
- $c_2''(\beta_i, \beta_j)$ with all the nodes in c_2'' belonging to c_1.

Note also that at least one trail of type c_2' must exist. Then, since $c_1 \in TH(x, y)$, we find that the subtrail of c_1 linking α_i and α_j belongs to $TH(\alpha_i, \alpha_j)$. So, we are in the same position as before [case (a)], and therefore we also obtain a contradiction.

The conclusion is that c_1 has to be the only active trail linking x and y. ■

PROPOSITION 2. *Let G be a simple graph and let c be an active trail linking two nodes x and y in G. Then, c is blocked by any node z in this trail and there is no nonactive trail linking x and y which is opened by z.*

Proof. From the d-separation criterion we directly obtain that c is blocked by z. Now, in order to prove that z does not open any nonactive trail, let us suppose that there is a nonactive trail e linking x and y which is opened by z. In this case, z has to be the only head-to-head node in e (or z is a descendant of every head-to-head node in e). Consider two different cases:

1. Suppose that $c \in TH(x, y)$ [the case $c \in HT(x, y)$ is completely analogous]. Since z opens the nonactive trail e, we can find a node p, $p \notin c$, $p \in e$, which is a parent of the head-to-head node nearest y in the trail e (and therefore p is an ancestor of z). Then, we have an active trail $c_1 \in TH(p, y)$ linking p and y (this trail goes through z). On the other hand, as p is the parent of the head-to-head node nearest to y in e, then another active trail c_2, must exist connecting p and y. Therefore, we find two active trails, c_1 and c_2, linking p and y, and $c_1 \in TH(p, y)$, in contradiction with Proposition 1.
2. Suppose that $c \in HH(x, y)$. Since z is a node in c, we can find an active trail belonging to either $TH(z, y)$ or $HT(z, x)$ (which is a subtrail of c). Now, we can apply the same reasoning used above, also obtaining a contradiction with Proposition 1.

Therefore, there is no nonactive trail linking x and y which is opened by z. ■

PROPOSITION 3. *Let G be a simple graph, and let x, y be two nonadjacent nodes in G, such that there exists at least one active trail linking x and y. Let $\rho_x(y)$ be the set of nodes connected directly to x in any active trail linking x and y. Then $I(x|\rho_x(y)|y)$ holds in G.*

Proof. As $\rho_x(y)$ contains a node from each active trail linking x and y, then all these active trails are blocked by $\rho_x(y)$. Moreover, from Proposition 2, we know that no nonactive trail is opened by instantiating the nodes in $\rho_x(y)$. Therefore, all the trails linking x and y are blocked by $\rho_x(y)$, hence x and y are d-separated by (are independent given) $\rho_x(y)$. ■

PROPOSITION 4. *Let G be a directed acyclic graph. Then G is a simple graph if, and only if, for every node x in G, and for every pair of nodes p_x, h_x, with $p_x \in Parents_x$ and $h_x \in Children_x$, $I(p_x|x|h_x)$ holds.*

Proof. Necessary condition:
Let G be a simple graph and let x be any node in G. Since, from Proposition 1, x belongs to the single active trail connecting any node $p_x \in \text{Parents}_x$ to any node $h_x \in \text{Children}_x$ then, using Proposition 3, we obtain $I(p_x|x|h_x)$.

Sufficient condition:
Let us suppose that G is not a simple graph. Then, we can find a cycle having only one head-to-head node. Let u be this node, and let p_u, q_p be nodes such that the trail $q_p \rightarrow p_u \rightarrow u$ belongs to the cycle (we can always find this trail, because u is the only head-to-head node in the cycle). Therefore, there is an active trail linking q_p and u not including p_u (the other part of the cycle which

does not pass through p_u). Then, using d-separation, we find $\neg I(q_p|p_u|u)$, which contradicts the hypothesis. Thus, the graph must be simple. ■

PROPOSITION 6. *Let G be a simple graph, and let x, y be any two nodes in G. Then, y belongs to I_x^{0-1} if, and only if, there is either an active cycle between x and y, or a direct connection between x and y in G.*

Proof. Necessary condition:

Let us suppose that $y \in I_x^{0-1}$, i.e., there are neither zero nor first-order conditional independence statements between x and y, and assume that x and y are not adjacent. Since x and y are marginally dependent, we can find at least one active trail, c, connecting x and y. Two different alternatives may be considered:

1. Suppose that c is the only active trail linking x and y. Let z be the node of c adjacent to x. Now, using Proposition 3, we obtain $I(x|z|y)$, contradicting the fact that there are no first-order conditional independence statements between x and y.
2. Suppose that there are at least two active trails connecting x and y. Then, by using Proposition 1, we know that these trails must belong to $HH(x, y)$, and therefore each pair of trails forms an active cycle. If x is not a head-to-head node in any of these cycles (the same reasoning applies to y), then all the active trails linking x and y must share a common node z, which is a parent of x. Once again using Proposition 3, we obtain $I(x|z|y)$ (because $\rho_x(y) = \{z\}$), contradicting the fact that there is no first-order conditional independence statement between x and y. Therefore, we conclude that x and y are the head-to-head nodes in every active cycle containing x and y, hence there is an active cycle *between* x and y in G.

The sufficient condition follows immediately from the d-separation criterion. ■

Now, we shall prove Proposition 7. First, we prove some preliminary lemmas.

LEMMA 1. *Let G be a simple graph, and let x, y be two nodes in G such that there is at least one active cycle between x and y in G. Then $\Omega_x^*(y)$ is not empty.*

Proof. We shall see that $\Omega_x^*(y)$ always includes at least the parents of x in any active trail in the cycle connecting x and y.

Since there is an active cycle between x and y, then we can find at least two active trails in $HH(x, y)$. Let w_i, w_j be the parents of x in these trails. It is clear that w_i and w_j belong to I_x^{0-1}. Moreover, considering the active trail connecting w_i and y, we find that $\neg I(w_i|x|y)$ holds, and therefore $w_i \in K_x(y)$ (and $w_j \in K_x(y)$, too). On the other hand, as G is simple, we know that $I(w_i|\varnothing|w_j)$ has to be true, and taking into account that there is a head-to-head connection at y, we deduce that $\neg I(w_i|y|w_j)$ holds. Therefore, w_i and w_j are in $\Omega_x^*(y)$, hence $\Omega_x^*(y) \neq \varnothing$. ■

The following lemmas identify graphically the prototypical skeletons that we can find whenever there is a direct connection between x and y and $\Omega_x^*(y)$ is not empty. In the figures, a dashed line represents a particular kind of active

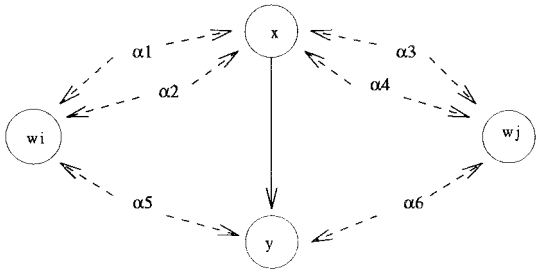

Figure 5. Active trail $TH(x, y)$.

trail, instead of the trail itself; for example, $x \longleftarrow\!\!-\!\!-\alpha_i\!\!-\!\!-\!\!\longrightarrow y$ represents any trail in $HH(x, y)$:

LEMMA 2. *Let G be a simple graph, and let x, y be nodes in G such that y is a child of x, i.e., a direct connection in* $TH(x, y)$ *exists. Then* $\Omega_x^*(y)$ *is not empty if and only if the graph in Figure 5 is a subgraph of G.*

Proof. Sufficient condition:

Using the d-separation criterion on the graph in Figure 5, we can easily see that w_i and w_j belong to $\Omega_x^*(y)$.

Necessary condition:

Suppose that $\Omega_x^*(y)$ is not empty and let w_i be any node in $\Omega_x^*(y)$. In this case, we can find at least some other node w_j in $\Omega_x^*(y)$. As w_i, w_j are also in I_x^{0-1} then, according to Proposition 6, they are parents, children or nodes forming an active cycle with x. Moreoever, since $I(w_i|\varnothing|w_j)$ holds, neither w_i nor w_j can be children of x. Thus, suppose that w_i is a parent of x (the same reasoning applies for w_j). In that case, there is an active trail $TH(w_i, y)$, which includes node x. Then, by using Proposition 4, we would deduce that $I(w_i|x|y)$ holds, contradicting the fact that $w_i \in K_x(y)$. So, w_i (and also w_j) cannot be a parent of x. Therefore, the only possibility is that there are active cycles between w_i and x and between w_j and x. So, at least two active trails connecting w_i (and w_j) with x exist and then, we find that x belongs to some active trails $HH(y, w_i)$. Moreover, since w_i belongs to $K_x(y)$, we know that $\neg I(w_i|x|y)$ holds. Therefore, from Proposition 3, at least an active trail, c, connecting w_i to y, not including node x, must exist. Finally, by using Proposition 1, this active trail c has to be of the $HH(w_i, y)$ type. So, we obtain the subgraph in Figure 5. ■

LEMMA 3. *Let G be a simple graph, and let x, y be nodes in G such that y is a parent of x, i.e., a direct connection in* $HT(x, y)$ *exists. Then* $\Omega_x^*(y)$ *is not empty if and only if one of the graphs in Figure 6 is a subgraph of G.*

Proof. Sufficient condition:

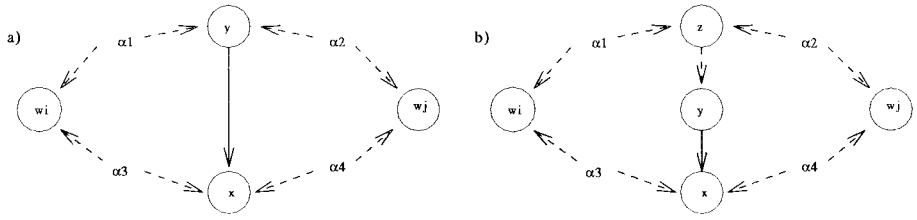

Figure 6. Active trail $HT(x, y)$.

It follows directly, by applying the d-separation criterion to the graphs in Figure 6, that w_i and w_j belong to $\Omega_x^*(y)$.
Necessary condition:

As $\Omega_x^*(y)$ is not empty, we can find at least two nodes w_i and w_j belonging to $\Omega_x^*(y)$. Once again, we shall see that w_i and w_j are neither parents nor children of node x:

1. Suppose that w_i (analogous for w_j) is a child of x: in this case, as y is a parent of x, using Proposition 4, $I(w_i|x|y)$ holds, contradicting the fact that $w_i \in K_x(y)$.
2. Suppose that w_i (analogous for w_j) is a parent of x: in this case, w_i and y have a common child, x, and therefore (because we are considering simple graphs), $I(w_i|\varnothing|y)$ holds, i.e., there is no active trail between w_i and y. But we also know that $I(w_i|\varnothing|w_j)$ and $\neg I(w_i|y|w_j)$ hold, and this implies, using d-separation, that there is an active trail between w_i and y, which is a contradiction.

Therefore, since w_i, $w_j \in I_x^{0-1}$ and they are neither parents nor children of x, then there is at least one active cycle between x and w_i, i.e., two active trails in $HH(x, w_i)$ (and the same happens for w_j). Once again, from the fact that $I(w_i|\varnothing|w_j)$ and $\neg I(w_i|y|w_j)$ hold, there is at least one active trail between w_i and y, and, considering that x is a child of y, we find that the form of these trails is $c_1 = (w_i - \ldots - y \rightarrow x)$. But we know that between w_i and x there is an active trail belonging to $HH(w_i, x)$, so according to Proposition 1, we find that c_1 is also in $HH(w_i, x)$, i.e., $c_1 = (w_i \leftarrow \ldots - y \rightarrow x)$. Using a similar reasoning, we can find an active trail $c_2 \in HH(w_j, x)$, such that $y \in c_2$, i.e., $c_2 = (w_j \leftarrow \ldots - y \rightarrow x)$. As w_i and w_j are marginally independent, i.e., there is no active trail connecting w_i and w_j, we can find at least one head-to-head node in $c_1 \cap c_2$. If c_1 and c_2 intersect only at nodes x and y, we obtain the graphical representation in Figure 6(a); on the other hand, if they also intersect at another node, we obtain the graphical representation in Figure 6(b). ■

PROPOSITION 7. *Let G be a simple graph, and x, y two nodes in G such that $y \in I_x^{0-1}$. Then, there exists an active cycle between x and y in G if, and only if, $\Omega_x(y) \neq \varnothing$.*

Proof. In order to prove the result, we are going to demonstrate the equivalent assertion: there is a direct connection between x and y in G if, and only if, $y \in I_x^{0-1}$ and $\Omega_x(y) = \varnothing$.

Necessary condition:

Suppose that there is a direct connection between x and y. Then, we obtain directly that $y \in I_x^{0-1}$. Now, suppose that $\Omega_x^*(y)$ is not empty. In that case, considering Lemmas 2 and 3, we find that for any node w in $\Omega_x^*(y)$ there is an active cycle between x and w (like w_i and w_j in Figs. 5 and 6). We shall consider the two possible direct connections between x and y.

1. Suppose that the connection belongs to $HT(x, y)$, i.e., $y \to x \in G$. Let w be any node in $\Omega_x^*(y)$. Using Lemma 3, let α be a parent of node y belonging to an active trail connecting w and y. In this case, node α verifies that $\alpha \in I_x^{0-1}$ and that $\neg I(\alpha|\varnothing|w)$, Moreover, the independence statements defining $\Omega_x(y)$ (condition b) hold i.e., $\neg I(\alpha|\varnothing|x)$, $I(\alpha|y|x)$ and $\neg I(\alpha|y|w)$. Therefore, every node w in $\Omega_x^*(y)$ can be removed, hence $\Omega_x(y)$ is empty.
2. Suppose that the connection belongs to $TH(x, y)$, i.e., $y \leftarrow x \in G$. Using Lemma 2, for each node $w \in \Omega_x^*(y)$ we can find a node α that allows us to remove w from $\Omega_x(y)$: let α be a parent of y in an active trail connecting w and y. Then, we have $\alpha \in I_y^{0-1}$ and $\neg I(\alpha|\varnothing|w)$. Moreover, for this node, we can find that the relationships defining $\Omega_x(y)$ (condition a), $I(\alpha|\varnothing|x)$ and $\neg I(\alpha|y|x)$ are verified, and therefore w does not belong to $\Omega_x(y)$. Thus, we may also conclude that the set $\Omega_x(y)$ must be empty.

Sufficient condition:

Suppose that $y \in I_x^{0-1}$, $\Omega_x(y) = \varnothing$, but there is no direct connection between x and y. Using Proposition 6, we obtain that there is an active cycle between x and y. Then, from Lemma 1 we conclude that $\Omega_x^*(y)$ is not empty. We shall show that not all the nodes in $\Omega_x^*(y)$ can be removed from $\Omega_x(y)$, so that $\Omega_x(y)$ would be nonempty.

Let w be any parent of node x in the active cycle between x and y. Then, the proof of Lemma 1 shows that w is in $\Omega_x^*(y)$. In order to obtain an empty $\Omega_x(y)$ we would have to remove w from this set. Thus, it is necessary to find a node α in I_y^{0-1} satisfying the independence statements that define $\Omega_x(y)$. Considering Proposition 6, since $\alpha \in I_y^{0-1}$, the different alternatives for α are:

1. α is a child of y: in this case, we have $\neg I(\alpha|\varnothing|w)$ and $\neg I(\alpha|\varnothing|x)$. Let us see that the other independence statements (in case b) can not be verified simultaneously: if $\neg I(\alpha|y|w)$ is true, then there is an active trail linking α and w not including y, and therefore we deduce $\neg I(\alpha|y|x)$.
2. α is a parent of y: in order to verify $\neg I(\alpha|\varnothing|w)$, an active trail linking w and α must exist, and so we deduce $\neg I(\alpha|\varnothing|x)$. But then we can also easily deduce $\neg I(\alpha|y|x)$.
3. There is an active cycle between α and y: the same reasoning used in the previous case is still valid.

Thus, we conclude that, for node w, we cannot find another node α satisfying the conditions that would allow us to remove w from $\Omega_x(y)$, which contradict the hypothesis. Therefore, a direct connection between x and y must exists. ■

PROPOSITION 8. *Let M be a dependency model isomorphic to a simple graph, and let $\mathcal{L}$ be a list of marginal and first-order conditional independence relationships*

obtained from M. Let G be the graph obtained by the Algorithm CH1. Then M is isomorphic to G.

Proof. Let G_M be any simple graph isomorphic to M. In order to prove the proposition, it is suffice to see that G_M has the same skeleton and the same head-to-head connections as G. First, we shall see that for any node x in G we obtain the same set of neighbors as in G_M. Let y be any node in G_M, $y \neq x$. If between x and y there is neither zero- nor first-order independence relationships in G_M, i.e., $y \in I_x^{0-1}$ then, from Proposition 6, we know that there is a direct connection between x and y or there is an active cycle between x and y. By using Proposition 7 we find that if there is a direct edge connecting x and y in G_M, then $\Omega_x(y)$ is empty and the algorithm inserts node y as direct neighbor of x [step 1 (c.iv)]. If, on the other hand, there is an active cycle between x and y, we obtain a nonempty $\Omega_x(y)$, and in that case, node y is not included as a direct neighbor of x [step 1 (c.iii)]. Therefore, after running the algorithm, there is a direct edge between x and y in G, if and only if this connection exists in G_M.

Now, we shall see that G has the same head-to-head connections as G_M. We know that any head-to-head connection in G_M involves a marginal independence statement. Therefore, since the algorithm checks this property [step 3(a)] before including any head-to-head connection, we conclude that G has the same set of head-to-head connections as G_M. ■

PROPOSITION 9. *Let M be a dag-isomorphic dependency model. If after executing step 5 of the algorithm CH2 we do not obtain an error code, then the output structure, G, is a simple graph.*

Proof. Since M is dag-isomorphic, let G_M be any dag isomorphic to M, and G be the graph obtained by the algorithm CH2.

1. First, we shall see that G is a dag, i.e., there is no directed cycle in G:
   Suppose that c is a directed cycle in G, and let $x_1, \ldots, x_{n-1}, x_n(= x_1)$ be the sequence of nodes in c, i.e., the substructure $x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_{n-1} \rightarrow x_1$ appears in G. Let x_i, x_{i+1}, x_{i+2} be any three consecutive nodes in c. We find that $\neg I(x_i|\varnothing|x_{i+2})$ holds in the model [this has been tested by the algorithm in step 3(a)]. Then, using the d-separation criterion, we find that there is at least one active connection between x_i and x_{i+2} in G_M. Now, using that $I(x_i|x_{i+1}|x_{i+2})$ holds [tested in step 4(a.ii) of the algorithm] and again the d-separation criterion, we find that node x_{i+1} belongs to this connection in G_M. Since these properties hold for any sequence of three nodes in c, we may conclude that, in G_M, there is a (undirected) cycle including nodes $x_1, \ldots, x_{n-1}$. In this case, as G_M is a dag, we know that at least there is a head-to-head node for this cycle in G_M. Suppose that x_j is such a head-to-head node. Then, we obtain $\neg I(x_{j-1}|x_j|x_{j+1})$, which is a contradiction since, in step 4(a.ii) it had been tested that this independence relationship was true. Therefore, there is no directed cycle in G.
2. Now, let us see that any cycle in G has to contain at least two head-to-head nodes:
   Suppose that there is a cycle c in G having only one head-to-head node, x. Let x_1 and x_n be the parents of x in c, $x_2, \ldots, x_{n-1}$ the other nodes in c, and let x_j be the (single) tail to tail node belonging to c. Thus, we have two active trails $c_1, c_2 \in TH(x_j, x)$ in G. Let x_k, x_{k+1}, x_{k+2} be any three consecutive nodes in c_1

(equivalently for c_2). In that case, we know that $\neg I(x_k|\varnothing|x_{k+2})$ and $I(x_k|x_{k+1}|x_{k+2})$ (otherwise, the algorithm would have produced an error code). Therefore, with a reasoning similar to the one used before, we find that, in G_M, there are at least two active connections between x_j and x, each one including those nodes in c_1 and c_2, respectively. We also know that $I(x_1|\varnothing|x_n)$ holds [tested in step 4(a.i)], so the node x_j must be a head-to-head node in G_M for these connections [otherwise, we would have an active connection linking x_1 and x_n in G_M, which contradicts $I(x_1|\varnothing|x_n)$], and therefore we obtain $\neg I(x_{j-1}|x_j|x_{j+1})$. Taking into account that the algorithm did not give an error code as the output in step 4(a.ii), then the trail $x_{j-1} \leftarrow x_j \rightarrow x_{j+1}$ had to be oriented before executing step 4(a.ii) [i.e., the algorithm oriented these edges in step 3(a)].

Now, we focus on the arc $x_j \rightarrow x_{j+1}$ in G (the same reasoning applies to the arc $x_{j-1} \leftarrow x_j$). Since this arc was oriented in step 3(a), we can find a node y, adjacent to x_{j+1} in G, such that $x_j \rightarrow x_{j+1} \leftarrow y$ belongs to G. So, the algorithm found $I(x_j|\varnothing|y)$ true, and then x_{j+1} is a head-to-head node for the connections between x_j and y in G_M. Therefore, we can conclude that all the active connections between x_j and x_{j+1} in G_M must be $HH(x_j, x_{j+1})$. This implies that the active connection between x_j and x, which passes through x_{j+1}, and x_n is head on x_{j+1}, so that we have an active connection $TH(x_{j+1}, x_n)$ in G_M. The same reasoning allows us to deduce also that there is an active connection $TH(x_{j-1}, x_1)$ in G_M. On the other hand, we know that $\neg I(x_{j-1}|\varnothing|x_{j+1})$, otherwise we would have directed the arcs as $x_{j-1} \rightarrow x_j \leftarrow x_{j+1}$ at step 3(a). So, we can ensure that, in G_M, there is an active trail linking x_{j-1} and x_{j+1}. Then, by combining this trail with the trails $TH(x_{j-1}, x_1)$ and $TH(x_{j+1}, x_n)$, we obtain an active trail linking x_1 and x_n in G_M, which contradicts the statement $I(x_1|\varnothing|x_n)$.

Therefore, we conclude that it is not possible that a cycle in G includes only one head-to-head node, hence the graph G has to be simple. ■

PROPOSITION 10. *Let M be a dag-isomorphic dependency model. Then, M is isomorphic to a simple graph if, and only if, the algorithm CH2 gives a simple graph as its output.*

Proof. Necessary condition:

If the model can be represented by a simple graph, we know (Proposition 8) that the first three steps in CH2, which are the same as in algorithm CH1, correctly identify the skeleton and all the head-to-head nodes of a simple graph isomorphic to M. Then, the independence checkings in step 4 cannot produce an error code. Moreover, in step 6, if $\Omega_x(y)$ is not empty, then there is an active cycle between x and y (Proposition 7), and the independence relationship $I(x|\Omega_x(y) \cap \text{Parents}_x|y)$ has to be true, hence the algorithm does not fail, and gives a simple graph isomorphic to M as the output.

Sufficient condition:

Let G be the simple graph obtained by the algorithm. Suppose that the model is not isomorphic to a simple graph, and let G_M be any dag isomorphic with M. As G_M is not simple, then there exists a cycle in G_M with only one head-to-head node, x. Let y be a parent of x in this cycle, and z a node adjacent to y, also in the cycle. Note that there are neither zero- nor first-order independence relationships between any pair of x, y and z. In that case, if all $\Omega_\alpha(\beta)$ (with α, β taking values in x, y, and z) were empty, then the edges x—y, y—z, z—x

would belong to the output graph G, contradicting the fact that this graph is simple (and therefore it is atriangular). Then, some set $\Omega_\alpha(\beta)$ has to be non empty, and therefore, as the algorithm did not fail, the corresponding independence statement $I(\alpha|\Omega_\alpha(\beta) \cap \text{Parents}_\alpha|\beta)$ must be ture.

Consider $Q_\alpha(\beta) = \Omega_\alpha(\beta) \cap \text{Parents}_\alpha$, that is to say, the variables in $\Omega_\alpha(\beta)$ which are adjacent to α in G. At least one of the following relationships hold: (1) $I(x|Q_x(y)|y)$, (2) $I(y|Q_y(z)|z)$, or (3) $I(x|Q_x(z)|z)$.

As x, y, and y, z are adjacent in G_M, statements 1 and 2 do not hold. Thus, $I(x|Q_x(z)|z)$ is the only alternative, but in this case, using the d-separation criterion, we find that all the active connections between x and z in G_M (and we know that there are at least two of them) are blocked by $Q_x(z)$. Then, at least y and some node, v, belonging to the other active connection between x and z in G_M, must belong to $Q_x(z)$. In these circumstances, since y and v belong to $\Omega_x(z) \cap \text{Parents}_x$, we deduce that $I(y|\varnothing|v)$ is true [the algorithm has verified this assertion in step 3(a), which represents a contradiction because there is an active trail in the cycle connecting y and v in G_M. Therefore, we conclude that the model must be isomorphic to a simple graph. ■

References

1. J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference,* Morgan Kaufmann, San Mateo, 1988.
2. J. Gebhardt and R. Kruse, ''Learning possibilistic graphical models,'' in *Proceeding of Third European Congress on Intelligent Techniques and Soft Computing (EUFIT'95),* 1995, pp. 57–64.
3. L.M. de Campos and J.F. Huete, ''Learning non probabilistic belief networks,'' in *Symbolic and Quantitative Approaches to Reasoning and Uncertainty, Lecture Notes in Computer Science* **747,** M. Clarke, R. Kruse, and S. Moral, Eds., Springer-Verlag, Berlin, 1993, pp. 57–64.
4. S.L. Lauritzen and D.J. Spiegelhalter, ''Local computations with probabilities on graphical structures and their applications to expert systems'' (with discussion), *J. Roy. Statist. Soc. (Ser B),* **50,** 157–224 (1988).
5. D. Heckerman, D. Geiger, and D. Chickering, *Learning Bayesian Networks: The Combination of Knowledge and Statistical Data,* Technical Report MSR-TR-94-09, Microsoft Research, 1995.
6. G.F. Cooper and E. Herskovits, ''A bayesian method for the induction of probabilistic networks from data,'' *Mach. Learn.* **9,** 309–347 (1992).
7. D. Heckerman, D. Geiger, and D.M. Chickering, ''Learning bayesian networks: The combination of knowledge and statistical data,'' in *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence,* R. Lopez de Mantaras and D. Poole, Eds., Morgan Kaufmann, 1994, pp. 293–301.
8. W. Lam and F. Bacchus, ''Using causal information and local measures to learn bayesian belief networks,'' in *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence,* D. Heckerman and A. Mamdani, Eds., Morgan Kaufmann, 1993, pp. 243–250.
9. D. Spiegelhalter, A. Dawid, S. Lauritzen, and R. Cowell, ''Bayesian analysis in expert systems,'' *Statist. Sci.,* **8,** 219–283 (1993).
10. D. Geiger, A. Paz, and J. Pearl, ''Learning simple causal structures,'' *Int. J. Intell. Syst.,* **8,** 231–247 (1993).
11. J. Pearl and T. Verma, ''A theory of inferred causation,'' in *Principles of Knowledge*

Representation and Reasoning: Proceedings of the Second International Conference, J.A. Allen, R. Fikes, and E. Sandewall, Eds., Morgan Kaufmann, San Mateo, 1991, pp. 441–452.
12. P. Spirtes, C. Glymour, and R. Scheines, ''An algorithm for fast recovery of sparse causal graphs,'' *Social Sci. Computer Rev.,* **9,** 62–72 (1991).
13. P. Spirtes, C. Glymour, and R. Scheines, *Causation, Prediction and Search,* Lecture Notes in Statistics **81,** Springer-Verlag, New York, 1993.
14. D. Chickering, D. Geiger, and D. Heckerman, *Learning Bayesian Networks is NP-Hard,* Technical Report MSR-TR-94-17, Microsoft Research, 1994.
15. L.M. de Campos and J.F. Huete, ''Independence concepts in upper and lower probabilities,'' in *Uncertainty in Intelligent Systems,* B. Bouchon-Meunier, L. Valverde, and R.R. Yager, Eds., North-Holland, Amsterdam, 1993, pp. 49–59.
16. L.M. de Campos and J.F. Huete, ''Possibilistic Independence,'' in *Proceeding of Third European Congress on Intelligent Techniques and Soft Computing (EUFIT'95),* 1995, pp. 69–74.
17. L.M. de Campos and S. Moral, ''Independence concepts for convex sets of probabilities,'' in *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence,* P. Besnard and S. Hanks, Eds., Morgan Kauffman, 1995, pp. 108–115.
18. L.M. de Campos, *Independency Relationships and Learning Algorithms for Singly Connected Networks,* Technical Report DECSAI-96-02-04, Dept. of Computer Science and Artificial Intelligence, Universidad de Granada, 1996.
19. D. Geiger, A. Paz, and J. Pearl, ''Learning causal trees from dependence information,'' in *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI 90),* 1990, pp. 770–776.
20. J.F. Huete and L.M. de Campos, ''Learning causal polytrees,'' in *Symbolic and Quantitative Approaches to Reasoning and Uncertainty, Lecture Notes in Computer Science* **747,** M. Clarke, R. Kruse, and S. Moral, Eds., Springer-Verlag, Berlin, 1993, pp. 180–185.
21. N. Friedman and M. Goldszmidt, ''Building classifiers using Bayesian networks'' *Proceedings of the National Conference on Artificial Intelligence (AAAI 96),* to appear.
22. D. Heckerman, ''A tractable inference algorithm for diagnosing multiple diseases,'' in *Uncertainty in Artificial Intelligence* **5,** R.D. Shachter, T.S. Levitt, L.N. Kanal, and J.F. Lemmer, Eds., Eselvier Science Publishers B.V. North-Holland, 1990, pp. 163–171.
23. T. Verma and J. Pearl, ''Causal networks: Semantics and expressiveness,'' in *Uncertainty in Artificial Intelligence* **4,** R.D. Shachter, T.S. Lewitt, L.N. Kanal, and J.F. Lemmer, Eds., North-Holland, 1990, pp. 69–76.
24. L.K. Rasmussen, *Blood group determination of Danish Jersey cattle in F-blood group system,* Dina Research Report No. 8, 1992.
25. M. Frydenberg, ''The chain graph Markov property,'' *Scand. J. Statistic,* **17,** 333–353 (1990).
26. J.F. Huete, *Aprendizaje de Redes de Creencia mediante la deteccion de independencias: Modelos no probabilısticos,* Ph.D. Thesis, Universidad de Granada, 1995.
27. L.M. de Campos, ''Characterizations of decomposable dependency models,'' *J. Artifi. Intell. Res.,* **5,** 289–300 (1996).