DISTRIBUTED STRUCTURE VERIFICATION IN MULTIPLY SECTIONED BAYESIAN NETWORKS

Y. Xiang
Department of Computer Science, University of Regina
Regina, Saskatchewan, Canada S4S 0A2, yxiang@cs.uregina.ca

Abstract

Multiply sectioned Bayesian networks (MSBNs) provide a framework for probabilistic reasoning in a single user oriented system in a large problem domain or in a cooperative multi-agent distributed interpretation system. During the construction or dynamic formation of a MSBN, an automatic verification of the acyclicity of the overall structure is desired. Although algorithms for testing acyclicity are well known, they assume a centralized storage of the graphical structure to be tested. We discuss why a centralized representation of the overall structure is undesirable. We then propose a set of distributed operations that are performed by individual subnets/agents in the system to cooperatively verify the acyclicity of the overall structure.

1 INTRODUCTION

Multiply sectioned Bayesian networks (MSBNs) is an extension of Bayesian networks (BNs) [4, 3, 1]. A MSBN consists of a set of interrelated Bayesian subnets [12, 11]. Each subnet shares a non-empty set of variables with at least one other subnet. Subnets are organized into a hypertree structure such that probabilistic inference can be performed coherently in a modular and distributed fashion. The modularity improves inference efficiency in a single user oriented system in a large problem domain [10]. It also extends MSBNs into a framework for probabilistic reasoning in cooperative multi-agent distributed interpretation systems [7].

As the structure of a BN is a directed acyclic graph (DAG), the overall structure of a MSBN, the composition of subnet structures, is also a DAG. To ensure the correct composition, an automatic verification of the acyclicity of the composed structure is desired. Although algorithms for testing acyclicity are well known [6], they assume a centralized storage of the graph to be tested. In this paper, we propose a set of distributed operations used by individual subnets/agents to test cooperatively the acyclicity of the composed structure.

Section 2 briefly reviews the theory and applications of MSBNs. The concepts necessary for the rest of the paper are formally defined. Section 3 discusses reasons why a distributed verification of acyclicity is preferred over a centralized test. Section 4 shows that some 'obvious' solutions to the distributed verification do not solve the problem. Section 5 derives the graph-theoretic fundation for the proposed operations and Section 6 presents these operations for cooperative verification.

2 OVERVIEW OF MSBNS

In this section, we briefly overview the theory of MSBNs and their applications. More details on MSBNs can be found in [12, 10, 7, 8].

A BN S is a triplet (N, D, P) where N is a set of variables, D is a directed acyclic graph (DAG) whose nodes are labeled by elements of N, and P is a joint probability distribution (jpd) over N. We shall call N the *domain* of S, D the *structure* of S and P the *distribution* or jpd of S.

Let $G^i = (N^i, E^i)$ $(i = 1, 2)$ be two graphs (directed or undirected). We shall refer to the graph $G = (N^1 \cup N^2, E^1 \cup E^2)$ as the *union* of G^1 and G^2, denoted $G = G^1 \sqcup G^2$.

A MSBN M is a collection of Bayesian subnets that together define a BN. These subnets are required to satisfy certain conditions that permit the construction of distributed inference algorithms. One of these conditions requires that nodes shared by different subnets form a *d-sepset*, as defined below.

Definition 1 (d-sepset) *Let $D^i = (N^i, E^i)$ $(i = 1, 2)$ be two DAGs such that $D = D^1 \sqcup D^2$ is a DAG. The intersection $I = N^1 \cap N^2$ is a* `d-sepset` *between D^1 and D^2 if for every $A_i \in I$ with its parents π_i in D, either $\pi_i \subseteq N^1$ or $\pi_i \subseteq N^2$. Each node in a d-sepset is called a* `d-sepnode`.

It can be shown that when a pair of subnets are isolated from M, their d-sepset renders them conditionally independent. Figure 1 (left) shows the three DAGs D^i $(i = 1, 2, 3)$ of a MSBN for diagnosis of three neuromuscular diseases, Median nerve lesion (Medn), Carpal tunnel syndrome (Cts) and Plexus upper trunk lesion (Plut).[1] Each d-sepnode is highlighted by a dotted circle. The d-sepset between each pair of DAGs is $\{Medn, Cts, Plut\}$. In general, d-sepsets between different pairs of DAGs of M may be different.

Just asthe structure of a BN is a DAG, the structure of a MSBN is a multiply sectioned DAG (MSDAG) of a hypertree structure, or simply a *hypertree MSDAG* defined as follows.

Definition 2 (Hypertree MSDAG) *A hypertree MSDAG $\mathcal{D} = \bigsqcup_i D^i$, where each D^i is a*

[1]The example is taken from a fraction of PAINULIM [10] with modification.

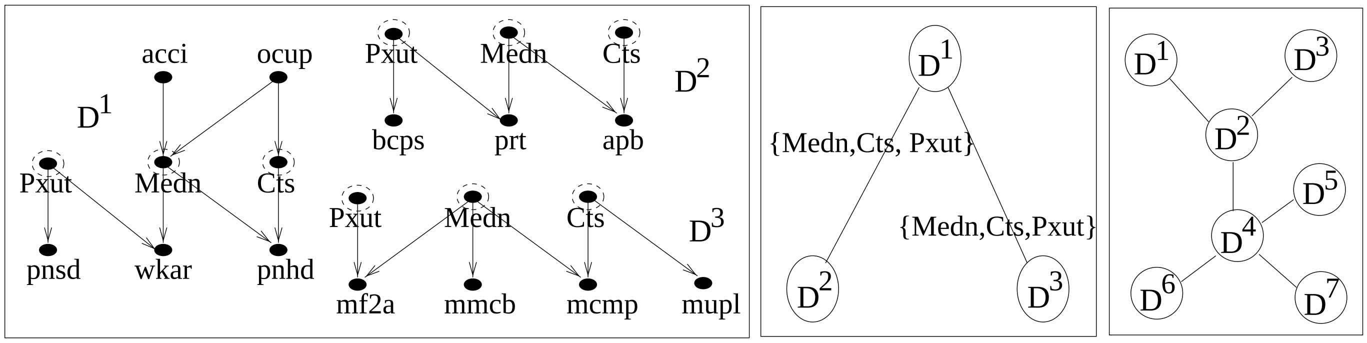

Figure 1: Left: The DAGs of an example MSBN for neural muscular diagnosis. Middle: The hypertree organization of the DAGs in the left. Right: A general hypertree MSDAG. Each d-sepnode is highlighted by a dotted circle.

connected DAG, is a DAG that is built by the following procedure:

Start with an empty graph (no node). Recursively add a DAG D^k, called a `hypernode`*, to the existing MSDAG $\bigsqcup_{i=1}^{k-1} D^i$ subject to the constraints:*

[d-sepset] For each D^j $(j < k)$, the intersection $I^{jk} = N^j \cap N^k$ is a d-sepset.

[Local covering] There exists D^i $(i < k)$ such that, for each D^j $(j < k; j \neq i)$, we have $I^{jk} \subseteq N^i$.

I^{jk} is called the `hyperlink` *between hypernodes D^j and D^k, and D^i and D^k are said to be* `adjacent`*.*

The DAGs in Figure 1 (left) can be organized into the hypertree MSDAG in Figure 1 (middle). Figure 1 (right) depicts a general hypertree MSDAG. Although DAGs of subnets of a MSBN M are organized into a tree as defined above, each DAG may be multiply connected (more than one path exist between a pair of nodes), e.g., D^1. Moreover, there can be multiple paths between a pair of nodes in different DAGs in a hypertree MSDAG. For instance, multiple paths are formed between *apb* and *mcmp* after D^2 and D^3 are unioned. A hypertree structured M ensures that each hyperlink render the two parts of M that it connects conditionally independent. An intuitive justification of this structure is given in [9].

A MSBN is defines as follows. Readers are referred to [12] for more details.

Definition 3 *A MSBN M is a triplet $(\mathcal{N}, \mathcal{D}, \mathcal{P})$. $\mathcal{N} = \bigcup_i N^i$ is the* `total universe` *where each N^i is a set of variables. $\mathcal{D} = \bigsqcup_i D^i$ (a hypertree MSDAG) is the* `structure` *where nodes of each DAG D^i are labeled by elements of N^i. $\mathcal{P} = \prod_i P^i(N^i) / \prod_k P^k(I^k)$ is the* `joint probability distribution` *(jpd). Each $P^i(N^i)$ is a probability distribution over N^i such that whenever D^i and D^j are adjacent in $\mathcal{D}$, the marginalizations of $P^i(N^i)$ and $P^j(N^j)$ onto the d-sepset I^{ij} are identical. Each $P^k(I^k)$ is such a marginal distribution over a hyperlink of $\mathcal{D}$. Each triplet $S^i = (N^i, D^i, P^i)$ is called a* `subnet` *of M.*

Without confusion, we shall say that two subnets S^i and S^j are adjacent if D^i and D^j are adjacent. We shall refer to the computational entity that holds the representation of a single subnet as an *agent*.

A MSBN can be used as a framework for probabilistic reasoning in a single user oriented system in a large problem domain. Using a MSBN is most beneficial if subdomains of the problem domain are loosely coupled (the size of each d-sepset is reasonably small relative to the size of the subdomain) and evidence and queries are focused on one subdomain for a period of time before shifting to a different subdomain. For example in Figure 1 (right), the user may focuses attention on the subnet of the structure D^1, denoted by S^1. After several pieces of evidence are entered and queries are issued to this subnet, the user may shift attention to the subnet S^3. The inference operations of MSBNs will then propagate evidence from S^1 to S^2 and then to S^3. The user can then enter evidence on variables contained in S^3. It can be shown that with such a restricted belief propagation during attention shift, the answers to queries obtained in S^3 are always consistent to all evidence accumulated in the *entire* MSBN. Computational complexity, however, is reduced by not having to update any subnets not on the hyperpath from the current subnet to the next target subnet. Application domains of single-user MSBNs include diagnosis of natural systems [10] and model-based diagnosis of artificial systems [5].[2]

MSBNs can be extended into a framework for probabilistic reasoning in cooperative multi-agent distributed interpretation systems. Each agent holds its partial perspective of a large problem domain, accesses a local evidence source, consumes its own computation resource, communicates with other agents *infrequently*, and answers queries. It can be shown [8] that if all agents are cooperative (vs self-interested), and each pair of adjacent agents are conditionally independent given their shared variables and have common initial belief on the shared variables, then a joint system belief is well defined which is consistent with each agent's belief. Even though multiple agents may acquire evidence asynchronously in parallel (compare with the single

[2]Although MSBNs are not referenced directly, the representation formalism used is a special case of MSBNs. For example, the set of input nodes I, output node O, mode node M, and dummy node D [5], which forms an interface between a higher level and a lower level in the hierarchy, is a d-sepset [12]. The 'composite joint tree' [5] corresponds to the 'hypertree' [12]. The way in which inference is performed in the composite join tree corresponds to the operation *ShiftAttention* [12].

user case where evidence is always entered into the current subnet), the corresponding communication operations of MSBNs ensure that the answers to queries from each agent are consistent with evidence acquired in the entire system after each communication. Since communication is infrequent, the operations also ensure that between two successive communications, the answers to queries for each agent are consistent with all local evidence gathered so far and are consistent with all evidence gathered in the entire system up to the last communication. Therefore, a MSBN can be characterized as one of functionally accurate, cooperative distributed systems [2]. Potential applications include decision support to cooperative human users in uncertain domains and troubleshooting a complex system by multiple knowledge based subsystems [8].

3 WHY DISTRIBUTED VERIFICATION?

As defined in Section 2, the structure of a MSBN is a MSDAG which is a DAG. Automatic verification of acyclicity of this structure is desirable in the construction of large MSBNs. Algorithms that test whether a directed graph is a DAG based on topological sorting are well known [6]. These algorithms, however, assume a central representation of the graphical structure to be tested.

A central representation of all DAGs in a MSBN is not desirable for at least two reasons. First, the construction of a multi-agent MSBN requires only the knowledge of the interface (d-sepset) between subnets (BNs) and does not require the knowledge of the internal structure of each subnet. Therefore, each subnet may be developed by an independent vendor who may not be willing to disclose the structural details. The assumption of a central representation of all DAGs will rule out the possibility to cooperate agents built by such vendors.

Secondly, a MSBN can potentially be dynamic. That is, subnets may join or leave the MSBN as the system is functioning. It is desirable to verify the correctness of the structure of the system whenever the member subnets change. It is also desirable that the verification does not require the communication of all DAGs to a central location or does not depend upon a single agent to maintain a repository of all DAGs in the current system.

In this paper, we propose a distributed algorithm for verification of the acyclicity of a MSBN structure. During the verification process, each agent does not need to reveal its internal structure. We shall refer to the structure to be tested as a *DAG union* since it may not qualify to be a MSDAG.

4 ISSUES IN DISTRIBUTED VERIFICATION

Recall that a MSDAG is built subject to the d-sepset and local covering conditions. It should be noted that these two conditions do not rule out the possibility of a directed cycle in the resultant DAG union.

Figure 2 shows two DAGs D^1 and D^2 with their d-sepset being $\{a, b\}$. If we union the two DAGs, it clearly satisfies the local covering condition. However, the union contains the directed cycle (a, c, d, b, g, a) and thus is not a DAG.

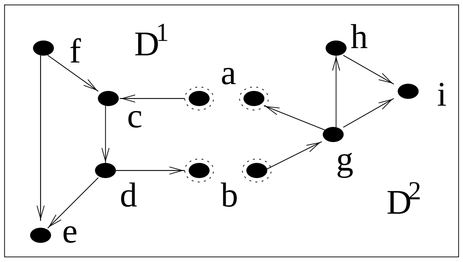

Figure 2: Two DAGs whose union is not a DAG.

The above cycle can be detected if we union the pair of DAGs and test the acyclicity. Although the pairwise verification may detect some directed cycles, pairwise acyclicity in a DAG union does not guarantee the global acyclicity.

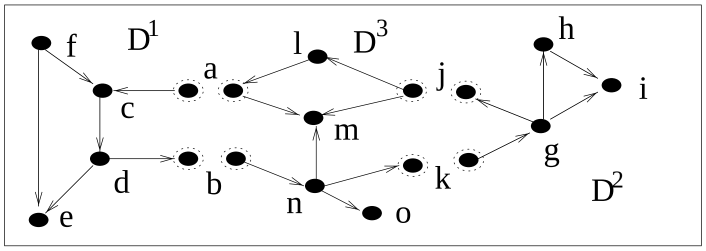

Figure 3: Three DAGs which are pairwise acyclic but whose union is cyclic.

Consider the three DAGs in Figure 3. The union of D^1 and D^3 is acyclic and so is the union of D^3 and D^2. However, when the three DAGs are unioned, a directed cycle $\{a, c, d, b, n, k, g, j, l, a\}$ is formed. Clearly, a distributed verification of acyclicity requires cooperation beyond pairs.

5 VERIFICATION BY MARKING NODES

In this section, we show that acyclicity of a DAG union can be verified by marking non-d-sepnodes and d-sepnodes separately and recursively. Once it is established, we can mark non-d-sepnodes locally and mark d-sepnodes by cooperation as presented in the next section. A node x is *marked* if x and arcs connected to x are ignored from further verification process. The following two propositions show that marking of non-d-sepnodes can be performed separately.

Proposition 4 *Let D be a DAG in a DAG union U. Let x be a root in D and be not in any d-sepset. Then the acyclicity of U remains the same after x is marked.*

Proof:

If U is acyclic, then marking x cannot create a directed cycle in U. Suppose U is cyclic. Then there exists a non-empty set O of cycles in U. Since x is a root in D, it does not have any incoming arc in D. Since x is not a d-sepnode, it cannot have any incoming arc from any other DAGs in U. Therefore, x cannot participate in any cycles in O, which implies that none of the cycles in O will be changed after x is marked. □

For example, the node f in Figure 3 is such a root node. Note that the condition that the node must not be in a d-sepset is necessary. For example, the node j

is a root node in D^3, but it should not be marked since it is part of an inter-DAG cycle.

Proposition 5 *Let D be a DAG in a DAG union U. Let x be a leaf in D and be not in any d-sepset. Then the acyclicity of U remains the same after x is marked.*

The proof is similar to that of Proposition 4. For example, the node o in Figure 3 is such a leaf node. Again, the non-d-sepnode condition is necessary. For instance, the leaf node j in D^2 should not be marked.

Once a non-d-sepnode root or leaf is marked, other non-d-sepnodes may become roots or leaves. For example, after the leaf i is marked, the node h becomes a leaf and can also be marked. Hence marking of non-d-sepnode roots and leaves can be performed recursively.

The following two propositions show that marking of d-sepnodes can also be performed separately. Their proof are similar to that of Proposition 4.

Proposition 6 *Let x be a d-sepnode in a DAG union U. If x is a root in every DAG that it participates, Then the acyclicity of U remains the same after x is marked.*

Proposition 7 *Let x be a d-sepnode in a DAG union U. If x is a leaf in every DAG that it participates, Then the acyclicity of U remains the same after x is marked.*

Similar to non-d-sepnodes, d-sepnode roots and leaves can also be marked recursively.

Next, we show that if a DAG union is acyclic, every node in it will be marked by recursive application of Propositions 4, 5, 6 and 7.

Proposition 8 *Let a DAG union U be acyclic. For each node x in U, x can be marked after a finite rounds of recursive marking of non-d-sepnode roots and leaves and d-sepnode roots and leaves.*

Proof sketch:

Without lossing generality, we assume that at each round of marking, all current non-d-sepnode roots and leaves are marked first, followed by the marking of all d-sepnode roots and leaves.

If x is either a root or a leaf, then x can be marked in the first round. Suppose x is neither a root nor a leaf. Since U is acyclic, there exists a longest directed path p started at a root r_1 at one end and a leaf v_1 at the other end such that x is in p. We denode p by $(r_1, r_2, \ldots, r_m, x, v_n, \ldots, v_2, v_1)$, where $m, n > 0$.

Since r_1 is a root, it can be marked either as a non-d-sepnode or as a d-sepnode in the first round. After the first round, r_2 must become a root. We show the opposite cannot be true. If r_2 does not become a root, it must be the case that r_2 has at least one other parent r not being marked in the first round. But this implies that r has at least one parent s, which in turn implies that there exists a directed path $t = (\ldots, s, r, r_2, \ldots, r_m, x, v_n, \ldots, v_2, v_1)$ that is longer than p. This contradicts the assumption that p is the longest path that contains x.

Using the same argument repeatedly, we conclude that x will be marked no later than the $m + 1$ rounds. □

Finally, we show that if a DAG union is cyclic, some nodes will remain unmarked.

Proposition 9 *Let a DAG union U be cyclic. At least three nodes cannot be marked after recursive marking of non-d-sepnode roots and leaves and d-sepnode roots and leaves.*

Proof sketch:

Since U is cyclic, there exists at least one directed cycle o in U. Since U is a DAG union, o consists of at least three nodes.

Let x be a node in o. We denote o by $(x, y_1, \ldots, y_n, x)$ where $n > 1$, y_1 is the parent of x and y_n is the child of x in o. Without lossing generality, we assume that at each round of marking, all current non-d-sepnode roots and leaves are marked first, followed by the marking of all d-sepnode roots and leaves.

We claim that x can never be marked as a root. Suppose x is marked in a particular round m as a root, then y_1 must be marked in round $m-1$ or earlier. But this implies that y_2 must be marked in round $m-2$ or earlier. Repeating this argument, y_n must be marked in round $m-n$ or earlier, which implies that x must be marked in round $m-n-1$ or earlier. Now the marking of x becomes the precondition for x to be marked, which is impossible. Hence we conclude that x can never be marked as a root. Similarly, x can never be marked as a leaf either.

Since the argument is applicable to any node x in o, none of the nodes in o can be marked by recursive marking of of non-d-sepnode roots and leaves and d-sepnode roots and leaves. □

6 COOPERATIVE VERIFICATION

As demonstrated in Section 4, in order to verify the acyclicity of a DAG union, agents must cooperate. Since cooperation requires communication which incurs overhead, it is desirable to simplify the task for cooperation as much as possible. According to Propositions 4 and 5, non-d-sepnode roots and leaves in a DAG union can be marked separately and recursively. We define a preprocessing operation to mark these nodes.

Let a DAG in the DAG union be arbitrarily chosen. If we treat this DAG as the root of the hypertree and direct the hyperlinks of the hypertree away from it, then the hypertree is converted into a *directed* tree. For each given DAG, we can then refer to each adjacent DAG as its *child* or its *parent* in the normal sense.

Operation 10 (PreProcess) *When* `PreProcess` *is called in a DAG D, the following are performed:*

1. *D recursively marks each non-d-sepnode root or leaf.*
2. *D calls* `PreProcess` *in each child DAG.*

After `PreProcess` is completed in a DAG union, nodes left unmarked in each DAG are either isolated d-sepnodes, or nodes that form directed pathes ended with d-sepnodes. Cooperation among DAGs is needed to further the verification process. Figure 4 shows the three DAGs in Figure 3 after `PreProcess` is initiated in any of them. Marked nodes are shown as grey. Only directed pathes are left in this case.

Based on Propositions 6 and 7, the operation `CollectFamilyInfo` is used by a DAG to find out if a d-sepnode x can be marked. The operation passes a triple (x, p, c) around all child DAGs which contain x.

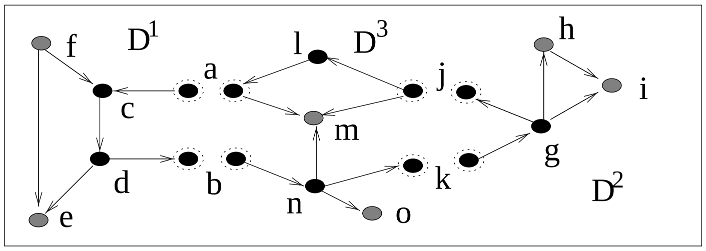

Figure 4: Three DAGs in Figure 3 after local preprocessing. Marked nodes are shown as grey.

The purpose is to record the parent/child information for x, where p is a count of the number of DAGs that contain parents of x and c is a count of the number of DAGs that contain children of x. The caller in the following definition refers to either a parent DAG or the next highter level of operation which initiated this operation.

Operation 11 (CollectFamilyInfo) *When* `CollectFamilyInfo(x)` *is called in a DAG D, the following are performed:*

1. *D forms a triple $t_0 = (x, p_0, c_0)$, where $p_0 = 1$ if D contains a parent of x and $p_0 = 0$ otherwise, and $c_0 = 1$ if D contains a child of x and $c_0 = 0$ otherwise.*
2. *If D has no child DAG to which x is a d-sepnode, then D returns t_0 to* `caller`*.*
3. *Otherwise, D calls* `CollectFamilyInfo(x)` *in each child DAG to which x is a d-sepnode.*
4. *After each child DAG being called has returned their triples (assuming k DAGs are called), $t_1, t_2, \ldots, t_k$, D returns a triple $t = (x, \sum_{i=0}^{k} p_i, \sum_{i=0}^{k} c_i)$ to* `caller`*.*

Once a d-sepnode is determined to be a root or a leaf, the operation `DistributeMark` is used to mark it in every DAG that it participates.

Operation 12 (DistributeMark) *When* `DistributeMark(x)` *is called in a DAG D, the following are performed:*

1. *D marks the node x.*
2. *D recursively marks any non-d-sepnode root or leaf.*
3. *If D has any adjacent DAG to which x is a d-sepnode except* `caller`*, then D calls* `DistributeMark(x)` *in each of them.*

The operation `MarkNode` combines `CollectFamilyInfo` and `DistributeMark` to recursively check and mark all d-sepnodes that are markable down the hypertree.

Operation 13 (MarkNode) *When* `MarkNode` *is called in a DAG D, the following are performed:*

1. *D returns* `false` *if it has no child DAG, otherwise continues.*
2. *For each unmarked d-sepnode x with a child DAG of D, D calls* `CollectFamilyInfo(x)` *in itself. When the trible (x, p, c) is returned to D, D calls* `DistributeMark(x)` *in itself if $p = 0$ or $c = 0$.*
3. *D calls* `MarkNode` *in each child DAG.*
4. *If any child DAG returns* `true` *or* `DistributeMark(x)` *was called in D, then D returns* `true` *to* `caller`*. Otherwise, D returns* `false` *(no node is marked).*

The operation `MarkedAll` checks if all nodes in a DAG union have been marked after roots and leaves have been recursively marked. The DAG union is acyclic if it returns `true`.

Operation 14 (MarkedAll) *When* `MarkedAll` *is called in a DAG D, the following are performed:*

1. *If there exists a node in D that has not been marked, then D returns* `false`*.*
2. *Otherwise, if D has no child DAG, it returns* `true`*. If D has children DAGs, D calls* `MarkedAll` *in each child DAG.*
3. *If any child DAG returns* `false` *(with unmarked nodes), then D returns* `false`*. Otherwise, D returns* `true`*.*

Finally, `TestAcyclicity` combines three previously defined operations to provide the top level operation for the verification of acyclicity of a DAG union.

Operation 15 (TestAcyclicity) *When* `TestAcyclicity` *is initiated in a DAG union, the following are performed:*

1. *A DAG D is arbitrarily chosen as the root of the hypertree.*
2. *D calls* `PreProcess` *in itself.*
3. *D calls* `MarkNode` *in itself repeatedly until* `false` *is returned (no node is marked in the last call).*
4. *D calls* `MarkedAll` *in itself. If* `true` *is returned, then* `TestAcyclicity` *is terminated with* `acyclic` *returned (the union is acyclic). Otherwise, the operation is terminated with* `cyclic` *returned.*

The following theorem establishes the correctness of the above operations.

Theorem 16 *The operation* `TestAcyclicity` *determines correctly if a DAG union U is acyclic or not.*

Proof sketch:
According to Propositions 4 and 5, non-d-sepnode roots and leaves can be marked separately and recursively. `PreProcess` does the first round of such marking and `DistributeMark` (step 2) performs the recursive marking.

According to Propositions 6 and 7, d-sepnode roots and leaves can be marked separately and recursively. `MarkNode` identifies these nodes by `CollectFamilyInfo` (either $p = 0$ or $c = 0$) and then mark them by `DistributeMark` (steps 1 and 3).

According to Proposition 8, if U is acyclic, all nodes can be marked by recursive marking of non-d-sepnode roots and leaves and d-sepnode roots and leaves. Each call of `MarkNode` in `TestAcyclicity` marks at least one root or one leaf, until all nodes are marked at which time `false` is returned. `MarkedAll` in step 4 will identify that all nodes have been marked and return `true` to signify that U is acyclic.

According to Proposition 9, if U is acyclic, at least three nodes are unmarked when `MarkNode` returns `false`. `MarkedAll` will then identify this and return `false` to signify that U is cyclic. □

7 REMARKS

In this paper, we presented a distributed algorithm, in terms of a set of distributed operations, for verification of acyclicity of the overall structure of a MSBN. The algorithm does not require each agent in the system to reveal its internal structure. It only provides information as whether a d-sepnode has a parent or a child in the DAG that the agent is responsible for. Therefore, the algorithm supports the construction of MSBNs constructed from multiple computational agents built by multiple vendors while providing the automatic verification of the correctness of the overall structure.

Acknowledgements

This work is supported by the Research Grant OGP0155425 from NSERC. Helpful comments from anonymous reviewers are acknowledged.

References

[1] E. Charniak. Bayesian networks without tears. *AI Magazine*, 12(4):50–63, 1991.

[2] V.R. Lesser and D.D. Corkill. Functionally accurate, cooperative distributed systems. *IEEE Trans. on Systems, Man and Cybernetics*, SMC-11(1):81–96, 1981.

[3] R.E. Neapolitan. *Probabilistic Reasoning in Expert Systems*. John Wiley and Sons, 1990.

[4] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[5] S. Srinivas. A probabilistic approach to hierarchical model-based diagnosis. In *Proc. 10th Conf. Uncertainty in Artificial Intelligence*, pages 538–545, Seattle, Washington, 1994.

[6] D.F. Stubbs and N.W. Webre. *Data Structures with Abstract Data Types and Modula-2*. Brooks/Cole, 1987.

[7] Y. Xiang. Distributed multi-agent probabilistic reasoning with Bayesian networks. In Z.W. Ras and M. Zemankova, editors, *Methodologies for Intelligent Systems*, pages 285–294. Springer-Verlag, 1994.

[8] Y. Xiang. Optimization of inter-subnet belief updating in multiply sectioned Bayesian networks. In *Proc. 11th Conf. on Uncertainty in Artificial Intelligence*, pages 565–573, Montreal, 1995.

[9] Y. Xiang. Semantics of multiply sectioned Bayesian networks for cooperative multi-agent distributed interpretation. In G. McCalla, editor, *Advances in Artificial Intelligence*, pages 213–226. Springer, 1996.

[10] Y. Xiang, B. Pant, A. Eisen, M. P. Beddoes, and D. Poole. Multiply sectioned Bayesian networks for neuromuscular diagnosis. *Artificial Intelligence in Medicine*, 5:293–314, 1993.

[11] Y. Xiang, D. Poole, and M. P. Beddoes. Exploring localization in Bayesian networks for large expert systems. In *Proc. 8th Conf. on Uncertainty in Artificial Intelligence*, pages 344–351, Stanford, 1992.

[12] Y. Xiang, D. Poole, and M. P. Beddoes. Multiply sectioned Bayesian networks and junction forests for large knowledge based systems. *Computational Intelligence*, 9(2):171–220, 1993.