Cooperative Verification of Agent Interface

Y. Xiang and **X. Chen**
Dept. Computing and Information Science, University of Guelph, Canada

Abstract

MSBNs support representation of probabilistic knowledge in multiagent systems (MAS). To ensure exact, distributed reasoning, agent interfaces must satisfy the d-sepset condition. Otherwise, the system will behave incorrectly. We present a method that allows agents to verify cooperatively the d-sepset condition through message passing. Each message reveals only partial information on the adjacency of a shared node in an agent's local network. Hence, the method respects agent's privacy, protects agent vendors' know-how, and promotes integration of MAS from independently developed agents. **keywords:** Bayesian network, MSBN, multiagent, verification, distributed computation.

1 Introduction

As the cost of computers and networking continues to drop and distributed systems are widely deployed, users are expecting more intelligent behaviors from such systems – multiagent systems (MAS) (Sycara, 1998). Agents in an MAS perform a set of tasks depending on the particular application domain. A common task is for a set of cooperative agents to determine what is the current state of the domain so that they can act accordingly. Agents monitoring a piece of equipment need to determine whether the equipment is functioning normally and, if not, which components have failed. Agents populating a smart house should recognize the current need of inhabitants and adjust the appliances accordingly. Similar situations arise in other domains such as cooperative design, battle field assessment, and surveillance. Often agents have only uncertain knowledge about the domain and must perform the task based on partial observations. Such a task has been termed *distributed interpretation* (Lesser and Erman, 1980) by some authors. We shall refer to it as multiagent situation assessment.

Different approaches have been proposed to tackle multiagent situation assessment. *Blackboard* (Nii, 1986) offers a framework for multiagent inference and cooperation. It does not dictate how uncertain knowledge should be represented nor offers any guarantee of inference coherence. DATMS (Mason and Johnson, 1989) and DTMS (Huhns and Bridgeland, 1991) offer inference frameworks based on default reasoning. Relation between BDI model and decision-tree is studies in (Rao and Georgeff, 1991). Reasoning about the mental state of an agent from the received communication is considered by (Dragoni et al., 2001). Monitoring whether a multiagent system is functioning normally by focusing on agent-relation is investigated in (Kaminka and Tambe, 1999). Emotions of agents are studied using decision theory in (Gmytrasiewicz and Lisetti, 2000). Proving hypotheses by agents with distributed knowledge using dialectical argumentation is proposed in (McBurney and Parsons, 2001). Multiply sectioned Bayesian networks (MSBNs) (Xiang and Lesser, 2000) provide a framework where agents' knowledge can be encoded with graphical models and agent's belief can be updated by distributed, exact probabilistic reasoning. Multiagent MSBNs (MAMSBNs) are the focus of this work.

Distributed and exact inference requires that an MAMSBN observes a set of constraints (Xiang and Lesser, 2000). When building an MAMSBN, these constraints on the knowledge representation need to be verified before inference for situation assessment takes place. Otherwise, garbage-in-garbage-out may occur and the resultant MAS will not reason correctly. When agents are autonomous and may be constructed by independent vendors (hence privacy of agents becomes an issue), verification of these constraints raises a challenge. In this work, we study verification of agent interface. We present a method that verifies the correctness of agent interfaces in an MAMSBN without compromising agent autonomy and privacy.

Section 2 briefly overviews MAMSBNs and introduces formal background necessary for the remainder of the paper.

2 Overview of MAMSBNs

A BN (Pearl, 1988) S is a triplet (N, G, P), where N is a set of domain variables, D is a DAG whose nodes are labeled by elements of N, and P is a joint probability distribution (jpd) over N. In an MAMSBN, a set of $n > 1$ agents $A_0, ..., A_{n-1}$ populates a *to-*

tal universe V of variables. Each A_i has knowledge over a *subdomain* $V_i \subset V$ encoded as a Bayesian subnet (V_i, G_i, P_i). The collection of local DAGs $\{G_i\}$ encodes agents' knowledge of domain dependency. Distributed and exact reasoning require these local DAGs to satisfy some constraints (Xiang and Lesser, 2000) described below:

Let $G_i = (V_i, E_i)$ $(i = 0, 1)$ be two graphs. The graph $G = (V_0 \cup V_1, E_0 \cup E_1)$ is referred to as the *union* of G_0 and G_1, denoted by $G = G_0 \sqcup G_1$. If each G_i is the subgraph of G spanned by V_i, we say that G is *sectioned* into G_i $(i = 0, 1)$. Local DAGs of an MAMSBN should overlap and be organized into a hypertree.

Definition 1 *Let $G = (V, E)$ be a connected graph sectioned into subgraphs $\{G_i = (V_i, E_i)\}$. Let the G_is be organized as a connected tree Ψ, where each node is labeled by a G_i and each link between G_k and G_m is labeled by the interface $V_k \cap V_m$ such that for each i and j, $V_i \cap V_j$ is contained in each subgraph on the path between G_i and G_j in Ψ. Then Ψ is a* ***hypertree*** *over G. Each G_i is a* ***hypernode*** *and each interface is a* ***hyperlink****.*

Each hyperlink serves as the information channel between agents connected and is referred to as an *agent interface*. To allow efficient and exact inference, each hyperlink should render the subdomains connected conditionally independent. It can be shown (by extending results in (Xiang and Lesser, 2000)) that this implies the following structural condition.

Definition 2 *Let G be a directed graph such that a hypertree over G exists. A node x contained in more than one subgraph with its parents $\pi(x)$ in G is a* ***d−sepnode*** *if there exists one subgraph that contains $\pi(x)$. An interface I is a* ***d−sepset*** *if every $x \in I$ is a d-sepnode.*

The overall structure of an MAMSBN is a hypertree MSDAG:

Definition 3 *A* ***hypertree MSDAG*** *$\mathcal{D} = \bigsqcup_i D_i$, where each D_i is a DAG, is a connected DAG such that (1) there exists a hypertree ψ over $\mathcal{D}$, and (2) each hyperlink in ψ is a d-sepset.*

Table 1: Agent communication interfaces.

$I_{0,1}$	$\{a_0, b_0, c_0, e_0, f_0, g_1, g_2, x_3, z_2\}$
$I_{1,2}$	$\{g_7, g_8, g_9, i_0, k_0, n_0, o_0, p_0, q_0, r_0, t_2, y_2, z_4\}$
$I_{2,3}$	$\{a_2, b_2, d_1, d_2, d_3, s_0, u_0, w_0, x_0, y_0, z_0\}$
$I_{2,4}$	$\{e_2, h_2, i_2, j_2, t_4, t_5, t_7, w_2, x_4, y_4, z_5\}$

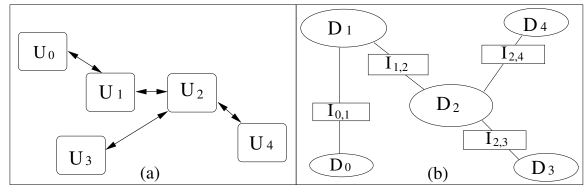

Figure 1: (a) A digital system of five components. (b) The hypertree modeling.

As a small example, Figure 1 (a) shows a digital system five components U_i $(i = 0, ..., 4)$. Although how components are interfaced, as shown in (a), and the set of interface variables, as shown in Table 1, are known to the system integrator, internal details of each component are proprietary. To give readers a concrete idea on the scenario, a centralized perspective of the digital system is shown in Figure 2. The

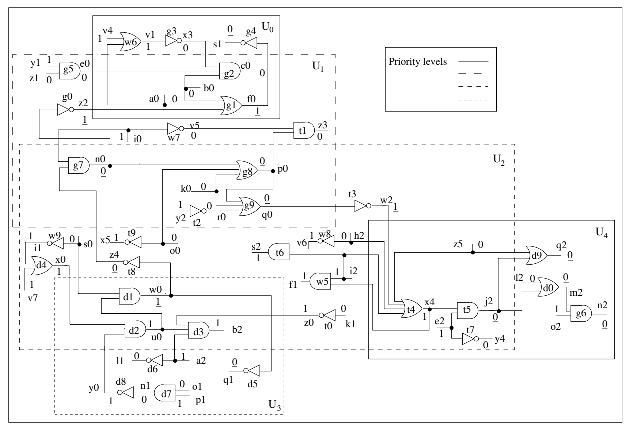

Figure 2: A digital system.

subnets for agents A_1 and A_2 are shown in Figures 3 and 4, where each node is labeled by the variable name and an index. The agent interface $I_{1,2}$ between them contains 13 variables and is a d-sepset. For instance, the parents of z_4 are all contained in D_2, while those of n_0 are contained in both D_1 and D_2.

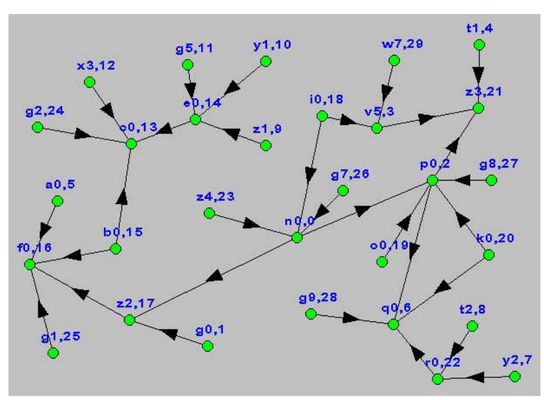

Figure 3: The subnet D_1 for U_1.

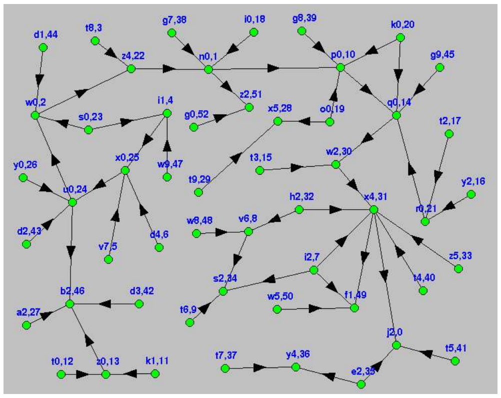

Figure 4: The subnet D_2 for U_2.

In an MAMSBN integrated from agents from different vendors, no agent has the perspective of Figure 2, nor the simultaneous knowledge of D_1 and D_2. Only the nodes in an agent interface are *public*. All other nodes in a subnet are *private* and known to the corresponding agent only. This forms the constraint of many operations in an MAMSBN, e.g., triangulation (Xiang, 2001) and communication (Xiang, 2000). Using these operations, agents can reason about their environment probabilistically based on local observations and limited communication. More formal details on MAMSBNs can be found in references noted above.

3 The issue of cooperative verification

Each agent interface in an MAMSBN should be a d-sepset (Def. 2). When an MAS is integrated from independently developed agents, there is no guarantee that this is the case. Blindly performing MAMSBN operations on the MAS would result in incorrect inference. Hence, agent interfaces need to be verified.

An agent interface is a d-sepset if every public node in the interface is a d-sepnode. However, whether a public node x in an interface I is a d-sepnode cannot be determined by the pair of local graphs interfaced with I. It depends on whether there exists a local DAG that contains all parents $\pi(x)$ of x in G. Any local DAG that shares x may potentially contain some parent nodes of x. Some parent nodes of x are public, but others are private. For agent privacy, it is desirable not to disclose parentship. Hence, we cannot send the parents of x in each agent to a single agent for d-sepnode verification. Cooperation among all agents whose subdomains contain x or parents of x is required to verify whether x is a d-sepnode. We refer to the unverified structure of an MAS as a *hypertree DAG union*.

In presenting our method, we will illustrate using examples. Although MAMSBNs are intended for large problem domains, many issues in this paper can be demonstrated using examples of much smaller scale. Hence, we will do so for both comprehensibility as well as space. Readers should keep in mind that these examples do not reflect the scales to which MAMSBNs are applicable.

A formal treatment of our method with proofs of properties has been worked out. However, full presentation of the formal treatment is beyond the space limit. Therefore, we only mention some formal results as necessary but reserve the details for an extended version.

4 Checking private parents

A public node x in a hypertree DAG union G may have public or private parents or both. Three cases regarding its private parents are possible: more than one local DAG (Case 1), exact one local DAG (Case 2), or no local DAG (Case 3) contains private parents of x.

The following proposition shows that the d-sepset condition is violated in Case (1).

Proposition 4 *Let a public node x in a hypertree DAG union G be a d-sepnode. Then no more than one local DAG of G contains private parent nodes of x.*

Proof: Assume that two or more local DAGs contain private parent nodes of x. Let y be a private parent of x contained in a local DAG G_i and z be a private parent of x contained in $G_j (i \neq j)$. Then there cannot be any one local DAG that contains both y and z. Hence no local DAG contains all parents of x, and x is not a d-sepnode by Def. 2, which is a contradiction. □

Figure 5 shows how this result can be used to detect non-d-sepnodes. We refer to the corresponding operation as **CollectPrivateParentInfo**. To verify if the public node j is a d-sepnode, suppose that agents perform a rooted message passing (shown by arrows in (a)). Agent A_4 sends a count 1 to A_3, signifying that it has private parents of j. A_3 has no private parents of j. It forms its own count 0, adds the count from A_4 to its own, and sends the result 1 to A_2. Because A_1 does not contain j, it does not participate in this operation. Hence, A_2 receives a message only from A_3. Because A_2 has only a public parent i of j, it forms its own count 0, adds the count from A_3 to its own, and sends the result 1 to A_0. Upon receiving the message, A_0 forms its own count 1, for it has a private parent p of j. It adds the count from A_2 to obtain 2 and the message passing halts. The final count signifies that there are two

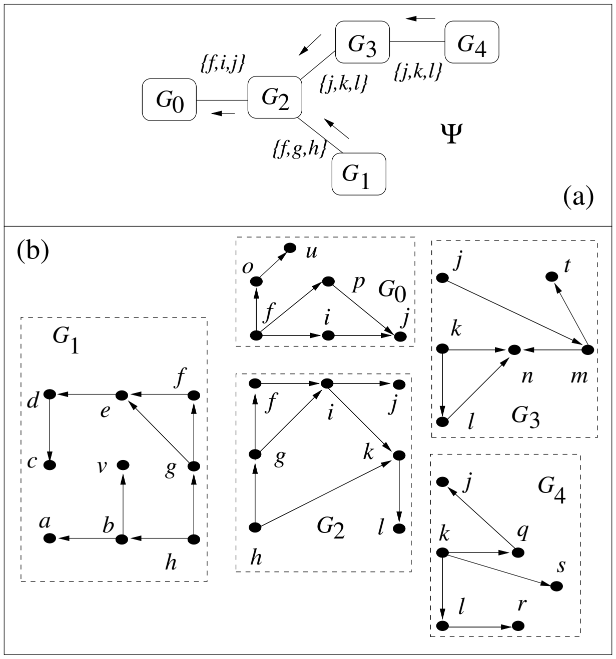

Figure 5: A hypertree DAG union with the hypertree in (a) and local DAGs in (b).

agents which contain private parents of j. Hence, j is a non-d-sepnode and the hypertree DAG union has violated the d-sepset condition.

5 Processing public parents

If **CollectPrivateParentInfo** on a public node x results in a final count less than or equal to 1, then no more than one agent contains private parents of x (Cases (2) and (3) above). The hypertree DAG union G, however, may still violate the d-sepset condition. Consider the example in Figure 6. The public nodes are w, x, y, z. No local DAG has any private parent of x or z. Only G_0 has a private parent of y, and only G_2 has a private parent of w. Hence, **CollectPrivateParentInfo** will produce a final count ≤ 1 for each of w, x, y, z. However, no single local DAG contains all parents of x: $\pi(x) = \{w, y\}$. Therefore, x is not a d-sepnode according to Def. 2 and none of the agent interfaces is a d-sepset.

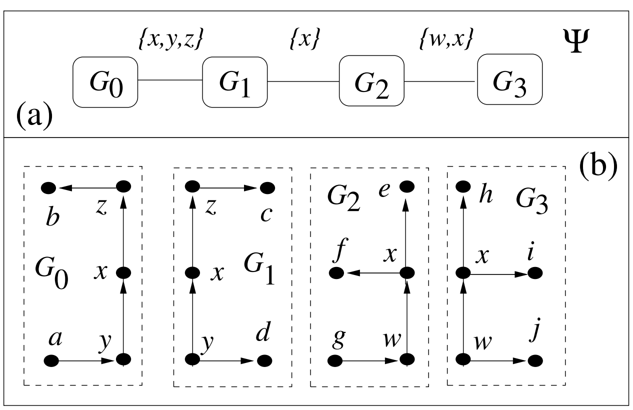

Figure 6: A hypertree DAG union G with the hypertree in (a) and local DAGs in (b).

The example illustrates that final counts from **CollectPrivateParentInfo** only provide a necessary condition for d-sepset verification. To determine if G satisfies the d-sepset condition conclusively, agents still need to further process the public parents of public nodes.

First, we consider Case 3, where no local DAG contains private parents of x. Case 2 will be considered in Section 6.

5.1 Public parent sequence

We propose the following concept called public parent sequence to describe the distribution of public parents $\pi(x)$ of a public node x on a hyperchain DAG union denoted as $\langle G_0, G_1, ..., G_m\rangle$. We use $X \bowtie Y$ to denote that sets X and Y are incomparable (neither is the subset of the other).

Definition 5 *Let $\langle G_0, G_1, ..., G_m\rangle$ ($m \geq 2$) be a hyperchain of local DAGs, where x is a public node, each G_i contains either x or some parents of x, and all parents of x are public. Denote the parents of x that G_i ($0 < i < m$) shares with G_{i-1} and G_{i+1} by $\pi_i^-(x)$ and $\pi_i^+(x)$, respectively. Denote the parents of x that G_m shares with G_{m-1} by $\pi_m^-(x)$. Then the sequence*

$$(\pi_1^-(x), \pi_2^-(x), ..., \pi_m^-(x))$$

*is the **public parent sequence** of x on the hyperchain. The sequence is classified into the following types, where $0 < i < m$:*

Identical *For each i, $\pi_i^-(x) = \pi_i^+(x)$.*

Increasing *For each i, $\pi_i^-(x) \subseteq \pi_i^+(x)$, and there exists i such that $\pi_i^-(x) \subset \pi_i^+(x)$.*

Decreasing *For each i, $\pi_i^-(x) \supseteq \pi_i^+(x)$, and there exists i such that $\pi_i^-(x) \supset \pi_i^+(x)$.*

Concave *One of the following holds:*

1. *For $m \geq 3$, there exists i such that the subsequence $(\pi_1^-(x), ..., \pi_i^-(x))$ is increasing and the subsequence $(\pi_i^-(x), ..., \pi_m^-(x))$ is decreasing.*
2. *There exists i such that $\pi_i^-(x) \bowtie \pi_{i+1}^-(x)$; the preceding subsequence $(\pi_1^-(x), ..., \pi_i^-(x))$ is trivial ($i = 1$), increasing, or identical; and the*

trailing subsequence $(\pi^-_{i+1}(x), ..., \pi^-_m(x))$ *is trivial* $(i = m-1)$*, decreasing, or identical.*

Wave *One of the following holds:*

1. *There exists* i *such that* $\pi^-_i(x) \supset \pi^+_i(x)$ *and* $j > i$ *such that either* $\pi^-_j(x) \subset \pi^+_j(x)$ *or* $\pi^-_j(x) \bowtie \pi^+_j(x)$.
2. *There exists* i *such that* $\pi^-_i(x) \bowtie \pi^+_i(x)$ *and* $j > i$ *such that either* $\pi^-_j(x) \subset \pi^+_j(x)$ *or* $\pi^-_j(x) \bowtie \pi^+_j(x)$.

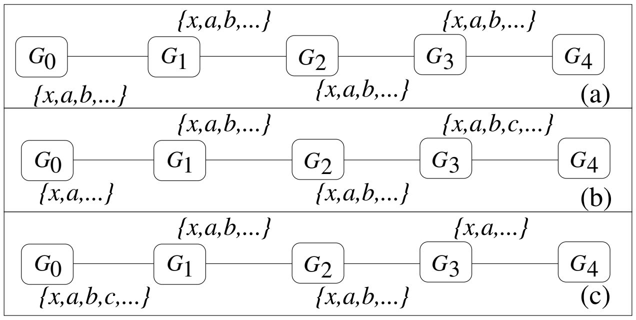

Figure 7: Public parent sequences. (a) An identical sequence. (b) An increasing sequence. (c) A decreasing sequence.

Figure 7 illustrates the first three sequence types, where only x and its parents are shown explicitly in each agent interface. **Identical** sequence is illustrated in (a). Each G_i contains $\pi(x) = \{a, b\}$, and hence x is a d-sepnode. **Increasing** sequence is exemplified in (b). From $i = 1$ to m, each G_i contains either the identical public parents of x or more. Because G_m contains $\pi(x)$, x is a d-sepnode. **Decreasing** sequence is exemplified in (c). It is symmetric to the increasing sequence; G_0 contains $\pi(x)$ and x is a d-sepnode.

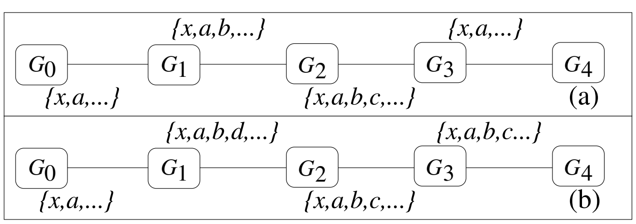

Figure 8: Concave parent sequences.

For **Concave** sequence, some parents of x appear in the middle of the hyperchain but not on either end. Figure 8 illustrates two possible cases. In (a), the parent b of x is contained in G_1, G_2, and G_3 but disappears in G_0 and G_4 and c is contained in G_2 and G_3 but disappears in G_0, G_1, and G_4. Two local DAGs (G_2 and G_3) in the middle of the hyperchain contain $\pi(x)$, and hence x is a d-sepnode. In (b), an increasing subsequence ends at $\pi^-_2(x)$, and a decreasing subsequence starts at $\pi^-_3(x)$ with $\pi^-_2(x)$ and $\pi^-_3(x)$ incomparable. Because G_2 contains $\pi(x)$, x is a d-sepnode.

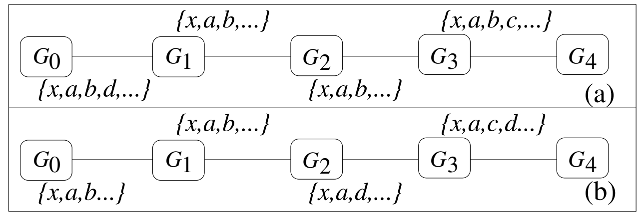

Figure 9: Wave parent sequences.

Figure 9 illustrates two possible cases of **Wave** sequence. In (a), a parent d of x appears at one end of the hyperchain, another parent c appears at the other end, and they disappear in the middle of the hyperchain. In other words, we have $\pi^-_1(x) \supset \pi^+_1(x)$ and $\pi^-_3(x) \subset \pi^+_3(x)$. No local DAG contains all parents of x, and hence x is not a d-sepnode. In (b), we have $\pi^-_2(x)$ and $\pi^+_2(x)$ being incomparable and $\pi^-_3(x) \subset \pi^+_3(x)$.

The following theorem states that the five parent sequences are exhaustive. They are also necessary and sufficient to identify d-sepnode.

Theorem 6 *Let* x *be a public node in a hyperchain* $\langle G_0, G_1, ..., G_m \rangle$ *of local DAGs with* $\pi(x)$ *being the parents of* x *in all DAGs, where no parent of* x *is private and each local DAG contains either* x *or some parents of* x*.*

1. *There exists one local DAG that contains* $\pi(x)$ *if and only if the public parent sequence of* x *on the hyperchain is identical, increasing, decreasing, or concave.*

2. *There exists no local DAG that contains* $\pi(x)$ *if and only if the public parent sequence of* x *on the hyperchain is of the wave type.*

5.2 Cooperative verification in hyperchain

To identify the sequence type by cooperation, agents on the hyperchain pass messages from one end to the other, say, from G_m to G_0. Each agent A_i passes a message to A_{i-1} formulated based on the message that A_i receives from A_{i+1} as well as on the result of comparison between $\pi^-_i(x)$ and $\pi^+_i(x)$. Note that A_{i+1} is undefined for A_m.

We partition the five public parent sequence types into three groups and associate each group with a message coded using an integer, as shown below:

type group	code
decreasing or **identical**	-1
increasing or **concave**	1
wave	0

Agents pass messages according to the algorithm **CollectPublicParentInfoOnChain** as defined below:

Algorithm 1 (CollectPublicParentInfoOnChain) *If A_{i+1} is undefined, agent A_i passes -1 to A_{i-1}. Otherwise, A_i receives a message from A_{i+1}, compares $\pi_i^-(x)$ with $\pi_i^+(x)$, and sends its own message according to one of the following cases:*

1. *The message received is -1:*

   If $\pi_i^-(x) \supseteq \pi_i^+(x)$, A_i passes -1 to A_{i-1}.

   Otherwise, A_i passes 1 to A_{i-1}.

2. *The message received is 1:*

   If $\pi_i^-(x) \subseteq \pi_i^+(x)$, A_i passes 1 to A_{i-1}.

   Otherwise, A_i passes 0 to A_{i-1}.

3. *The message received is 0: A_i passes 0 to A_{i-1}.*

We demonstrate how agents cooperate using examples in Figures 7 through 9. In Figure 7 (a), -1 is sent from A_4 to A_3 and is passed along by each agent until A_0 receives it. Interpreting the message code, A_0 concludes that the parent sequence is either **identical** or **decreasing**. Because the actual sequence is **identical**, the conclusion is correct.

In (b), A_3 receives -1 from A_4 and sends 1 to A_2. Afterwards, 1 is passed all the way to A_0, which determines that the sequence is either **increasing** (actual type) or **concave**.

In (c), -1 is sent by each agent. The conclusion drawn by A_0 is to classify the type of sequence as either **identical** or **decreasing** (actual type).

In Figure 8 (a), A_3 receives -1 from A_4 and sends -1 to A_2. Agent A_2 sends 1 to A_1, which passes it to A_0. Agent A_0 then concludes that the sequence type is either **increasing** or **concave**, where **concave** is the actual type. In (b), -1 is sent from A_4 to A_3 and then to A_2. Agent A_2 sends 1 to A_1, which is passed to A_0.

In Figure 9 (a), A_3 receives -1 from A_4 and sends 1 to A_2. Agent A_2 passes 1 to A_1, which in turn sends 0 to A_0. Agent A_0 then interprets the sequence type as a **wave**, which matches the actual type. In (b), A_3 receives -1 from A_4 and sends 1 to A_2. Agent A_2 sends 0 to A_1, which passes 0 to A_0.

In summary, each agent on the hyperchain can pass a code message formulated based on the message it receives and the comparison of the public parents it shares with the adjacent agents. The message passing starts from one end of the hyperchain and the type of the public parent sequence can be determined by the agent in the other end. In this cooperation, no agent needs to disclose its internal structure.

5.3 Cooperative verification in hypertree

We investigate the issue in a general hypertree, and let agents to cooperate in a similar way as in a hyperchain. However, the message passing is directed towards an agent acting as the root of the hypertree.

Consider first the case in which the root agent A_i has exactly two adjacent agents A_1 and A_2. If an agent A_i has a downstream adjacent agent A_k, we denote the parents of x that A_i shares with A_k by $\pi_k(x)$. In Section 5.2, the only information that agent A_0 needs to process is the message received from A_1. Here, A_i has three pieces of information: two messages received from adjacent agents and a comparison between $\pi_1(x)$ and $\pi_2(x)$. The key to determine whether x is a d-sepnode is to detect whether its public parent sequence along any hyperchain, on the hypertree, is the **wave** type. A **wave** sequence can be detected based on one message received by A_i only (when the hyperchain from A_i to a terminal agent is a **wave**), or if not sufficient based on both messages received, or if still not sufficient based in addition on the comparison between $\pi_1(x)$ and $\pi_2(x)$.

The idea can be applied to a general hypertree where A_i has any finite number of adjacent agents. Now A_i must take into account the three pieces of information for each pair of adjacent agents. Consider

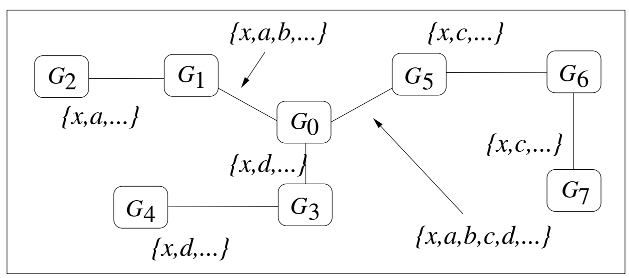

Figure 10: Parents $\pi(x)$ of a d-sepnode x shared by local DAGs in a hypertree.

the hypertree in Figure 10. If A_0 is the root, then messages will be passed towards A_0 from terminal agents A_2, A_4, and A_7. After agents send messages according to **CollectPublicParentInfoOnChain**, A_0 receives -1 from each of A_1, A_3, and A_5. This implies that the parent sequence type of each hyperchain from A_0 to a terminal agent is either **identical** or **decreasing**. Hence agent A_0 can conclude that itself contains $\pi(x)$ and x is a d-sepnode.

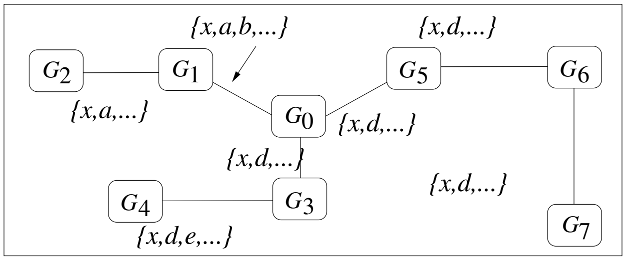

Figure 11: Parents $\pi(x)$ of a non-d-sepnode x shared by local DAGs in a hyperstar.

In Figure 11, suppose that A_5 is the root. Messages will be passed towards A_5 from terminal agents A_2, A_4, and A_7. Agent A_0 will receives -1 from A_1 and 1 from A_3. It realizes that each hyperchain from A_0 downstream through A_1 is either **identical** or **decreasing** and the hyperchain from A_0 downstream through A_3 is either **increasing** or **concave**. Because the messages are not sufficient to conclude, A_0 compares $\pi_1(x)$ with $\pi_3(x)$. It discovers that they are incomparable. This implies that there exist a hyperchain H_1 from A_0 downstream through A_1 and a hyperchain H_3 from A_0 downstream through A_3 such that when H_1 is joined with H_3 the resultant hyperchain has a **wave** parent sequence. Hence, A_0 will pass the code message 0 to A_5. Based on this message, the root agent A_5 concludes that x is not a d-sepnode. The conclusion is correct because no local DAG contains both a and e.

The following algorithm describes the actions a typical agent A_0 performs.

Algorithm 2 (CollectPublicParentInfo(x))

1. *Receive a message m_i from each downstream adjacent agent A_i.*
2. (a) *If any message is 0, A_0 sends 0 to the upstream agent A_c.*
   (b) *Otherwise, if any two messages are 1, A_0 sends 0 to A_c.*
   (c) *Otherwise, if a message m_i is 1, then A_0 compares $\pi_i(x)$ with $\pi_j(x)$ for each downstream adjacent agent A_j. If j is found such that $\pi_i(x) \not\supseteq \pi_j(x)$, A_0 sends 0. If not found, A_0 sends 1.*
   (d) *Otherwise, continue.*
3. *A_0 compares each $\pi_i(x)$ with the parents $\pi_c(x)$ shared with A_c. If there exists i such that $\pi_c(x) \not\supseteq \pi_i(x)$, then A_0 sends 1 to A_c. Otherwise, A_0 sends -1.*

The following theorem establishes that d-sepnode condition can be verified correctly by agent cooperation through **CollectPublicParentInfo**.

Theorem 7 *Let a hypertree of local DAGs $\{G_i\}$ be populated by a set of agents. Let x be a public node with only public parents in the hypertree. Let agents pass messages according to **CollectPublicParentInfo(x)**.*

Then x is a non-d-sepnode if and only if the root agent returns 0.

6 Cooperative verification in a general hypertree

We consider cooperative verification of the d-sepnode condition when both public and private parents of a public nodes are present. Agents who populate such a hypertree can first perform **CollectprivateParentInfo** to find out whether more than one local DAG contains private parents of x. If two or more agents are found to contain private parents of x, then agents can conclude, by Proposition 4, x is a non-d-sepnode. If no agent is found to contain private parents of x, then agents can perform **CollectPublicParentInfo** with any agent being the root to determine if x is a d-sepnode.

On the other hand, if one agent A_0 is found to contain private parents of x, then agents can perform **CollectPublicParentInfo** with A_0 being the root to determine if x is a d-sepnode. Note that it is necessary for A_0 to be the root. For instance, in Figure 10, if A_2 is the only agent that contains the private parents of x, when **CollectPublicParentInfo** is performed with the root A_0, agent A_0 cannot conclude as in Section 5.3. Clearly, although A_0 contains all public parents of x, it does not contain the private parents of x. Hence, it is unknown to A_0 whether there is an agent containing all parents of x. In this case, it depends on whether A_2 is such an agent.

The following algorithm summarizes the method.

Algorithm 3 (VerifyDsepset)

Let a hypertree DAG union G be populated by multiple agents with one at each hypernode. For each public node x, agents cooperate as follows:

1. *Agents perform **CollectprivateParentInfo**. If more than one agent is found to contain private parents of x, conclude that G violates the d-sepset condition.*
2. *If no agent is found to contain private parents of x, agents perform*

CollectPublicParentInfo *with any agent A_0 as the root. If A_0 generates the message 0, conclude that G violates the d-sepset condition. Otherwise, conclude that G satisfies the d-sepset condition.*

3. *If a single agent A_0 is found to contain private parents of x, then agents perform* ***CollectPublicParentInfo*** *with A_0 as the root. If A_0 generates the message -1, conclude that G satisfies the d-sepset condition. Otherwise, conclude that G violates the d-sepset condition.*

It can be proven that **VerifyDsepset** accomplishes the intended task correctly:

Theorem 8 *Let a hypertree DAG union G be populated by multiple agents. After VerifyDsepset is executed in G, it concludes correctly with respect to whether G satisfies the d-sepset condition.*

7 Complexity

We show that multiagent cooperative verification by VerifyDsepset is efficient. We denote the maximum cardinality of a node adjacency in a local DAG by t; the maximum number of nodes in an agent interface by k; the maximum number of agents adjacent to any given agent on the hypertree by s; and the total number of agents by n.

Each agent may call **CollectPrivateParentInfo** $O(k\ s)$ times – one for each shared node. Each call may propagate to $O(n)$ agents. Examination of whether a shared node has private parents in a local DAG takes $O(t)$ time. Hence, the total time complexity for checking private parents is $O(n^2\ k\ s\ t)$.

Next, we consider processing of public parents after checking private parents succeeds positively. The computation time is dominated by **CollectPublicParentInfo**. Each agent may call **CollectPublicParentInfo** $O(k\ s)$ times. Each call may propagate to $O(n)$ agents. When processing public parent sequence information, an agent may compare $O(s)$ pairs of agent interfaces. Each comparison examines $O(k^2)$ pairs of shared nodes. Hence, the total time complexity for processing public parents is $O(n^2\ k^3\ s^2)$. The overall complexity of VerifyDsepset is $O(n^2\ (k^3\ s^2 + k\ s\ t))$ and the computation is efficient.

8 Conclusion

We present a method to verify agent interface in a MAS whose knowledge representation is based on MSBNs. To ensure exact, distributed probabilistic inference, agent interfaces must be d-sepsets. Using our verification method, agents only pass concise messages among them without centralized control. A message reveals only partial information about the parenthood of a public node without disclosing additional details on the agent's local DAG. Hence, the method respects agent's privacy, protects agent vendors' know-how, and promotes integration of MAS from independently developed agents.

MSBNs support both modular, exact probabilistic inference in single agent systems and exact, distributed probabilistic inference in MAS. The connection between MSBNs and OOBNs was explored by Koller and Pfeffer (Koller and Pfeffer, 1997). Although OOBNs are intended for single agent systems, the object interfaces also have to satisfy the d-sepset condition. The approach taken was to require all arcs from one network segment to another to follow the same direction. Owing to this requirement, the d-sepset condition is automatically satisfied in a hypertree DAG union. No verification is required. On the other hand, the requirement does restrict the dependency structures to a proper subset of general MSBNs. For instance, in the MSBN for monitoring the digital system (Figure 2), arcs may go either way between a pair of adjacent local DAGs. The method presented in this paper allows agent interfaces to be verified efficiently in a general MSBN.

References

A.F. Dragoni, P. Giorgini, and L. Serafini. 2001. Updating mental states from communication. In *Intelligent Agents VII: Agent Theories, Architectures and Languages*. Springer-Verlag.

P.J. Gmytrasiewicz and C.L. Lisetti. 2000. Using decision theory to formalize emotions for multiagent systems. In *Second ICMAS-2000 Workshop on Game Theoretic and Decision Theoretic Agents*, Boston.

M.N. Huhns and D.M. Bridgeland. 1991. Multiagent truth maintenance. *IEEE Trans. Sys., Man, and Cybernetics*, 21(6):1437–1445.

G. Kaminka and M. Tambe. 1999. I'm ok, you're ok, we're ok: experiments in centralized and distributed socially attentive monitoring. In *Proc. Inter. Conference on Automonomous Agents.*

D. Koller and A. Pfeffer. 1997. Object-oriented Bayesian networks. In D. Geiger and P.P. Shenoy, editors, *Proc. 13th Conf. on Uncertainty in Artificial Intelligence*, pages 302–313, Providence, Rhode Island.

V.R. Lesser and L.D. Erman. 1980. Distributed interpretation: a model and experiment. *IEEE Trans. on Computers*, C-29(12):1144–1163.

C.L. Mason and R.R. Johnson. 1989. DATMS: a framework for distributed assumption based reasoning. In L. Gasser and M.N. Huhns, editors, *Distributed Artificial Intelligence II*, pages 293–317. Pitman.

P. McBurney and S. Parsons. 2001. Chance discovery using dialectical argumentation. In T. Terano, T. Nishida, A. Namatame, S. Tsumoto, Y. Ohsawa, and T. Washio, editors, *New Frontiers in Artificial Intelligence, Lecture Notes in Artificial Intelligence Vol. 2253*, pages 414–424. Springer-Verlag.

H.P. Nii. 1986. Blackboard systems: the blackboard model of problem solving and the evolution of blackboard architectures. *AI Magazine*, 7(2):38–53.

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

A. Rao and M. Georgeff. 1991. Deliberation and its role in the formation of intentions. In B. D'Ambrosio, P. Smets, and P.P. Bonissone, editors, *Proc. 7th Conf. on Uncertainty in Artificial Intelligence*, pages 300–307. Morgan Kaufmann.

K.P. Sycara. 1998. Multiagent systems. *AI Magazine*, 19(2):79–92.

Y. Xiang and V. Lesser. 2000. Justifying multiply sectioned Bayesian networks. In *Proc. 6th Inter. Conf. on Multi-agent Systems*, pages 349–356, Boston.

Y. Xiang. 2000. Belief updating in multiply sectioned Bayesian networks without repeated local propagations. *Inter. J. Approximate Reasoning*, 23:1–21.

Y. Xiang. 2001. Cooperative triangulation in MSBNs without revealing subnet structures. *Networks*, 37(1):53–65.