Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

INTERNATIONAL JOURNAL OF APPROXIMATE REASONING

ELSEVIER International Journal of Approximate Reasoning 33 (2003) 235–254
www.elsevier.com/locate/ijar

Comparison of multiagent inference methods in multiply sectioned Bayesian networks

Y. Xiang *

Department of Computing and Information Science, College of Physical and Engineering Science, University of Guelph, Guelph, Ont., Canada NIG 2W1

Received 1 August 2002; accepted 1 December 2002

Abstract

As intelligent systems are being applied to larger, open and more complex problem domains, many applications are found to be more suitably addressed by multiagent systems. Multiply sectioned Bayesian networks provide one framework for agents to estimate what is the true state of a domain so that the agents can act accordingly. Existing methods for multiagent inference in multiply sectioned Bayesian networks are based on linked junction forests. The methods are extensions of message passing in junction trees for inference in single-agent Bayesian networks.

Many methods other than message passing in junction trees have been proposed for inference in single-agent Bayesian networks. It is unclear whether these methods can also be extended for multiagent inference. This paper presents the first investigation on this issue. In particular, we consider extending loop cutset conditioning, forward sampling and Markov sampling to multiagent inference. They are compared with the linked junction forest method in terms of off-line compilation, inter-agent messages during communication, consistent local inference, and preservation of agent privacy. The results reveal issues to be considered in investigating other single-agent oriented inference methods. The analysis provides those who implement multiagent probabilistic inference systems with a guide on the pros and cons of alternative methods.

* Tel.: +1-519-824-4120x52824; fax: +1-519-837-0323.
E-mail address: yxiang@cis.uoguelph.ca (Y. Xiang).

0888-613X/03/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/S0888-613X(03)00017-3

Keywords: Multiagent uncertain reasoning; Exact and approximate reasoning; Bayesian networks; Agent autonomy; Agent privacy

1. Introduction

As intelligent systems are being applied to larger, open and more complex problem domains, many applications are found to be more suitably addressed by multiagent systems [14,16]. Consider a large uncertain problem domain populated by a set of agents. The agents can be charged with many possible tasks depending on the nature of the application. One common task is to estimate what is the true state of the domain so that they can act accordingly. In general, each agent has only a partial perspective of the domain (with only knowledge on a local subdomain and can only obtain local observations). Such a task, often referred to as *distributed interpretation* [6], arises in many applications including trouble-shooting complex equipment, building/area surveillance, battle field/disaster situation assessment, and distributed design. Multiply sectioned Bayesian networks (MSBNs) [17,23] provide one framework to conduct such a task. How to use the MSBN framework for equipment monitoring and diagnosis is illustrated in [21]. An MSBN consists of a set of interrelated Bayesian subnets each of which encodes an agent's knowledge on a subdomain. Probabilistic inference can be performed in a distributed fashion while answers to queries are exact with respect to probability theory.

Existing methods for multiagent inference in MSBNs are extensions of a class of methods for inference in single-agent Bayesian networks (BNs): message passing in junction trees [5,8,12]. The *linked junction forest* (LJF) method [19] compiles the subnet at each agent into a junction tree (JT). Inter-agent message passing is performed through a *linkage tree* between a pair of adjacent agents. The distributed Shafer–Shenoy propagation and distributed lazy propagation [22] compile the subnet at an agent into a set of JTs, one for each adjacent agent. The JT is then used for message passing with the agent.

Inference methods, other than message passing in JTs, have been proposed for reasoning in single-agent BNs. Loop cutset conditioning [9,13] converts a general BN into a number of tree-structured BNs. Efficient inference can then be performed in each of them and the results are combined. Direct factoring, such as [7], and variable elimination, such as [1], marginalize out variables not in the query, one by one, from a product of a small subset of probability distributions. Search based inference, such as [3,10], approximates the posteriors by summing a small subset of joint probability values that contains most of the probability mass. Simulation based inference, such as [2,9,11], uses Monte Carlo sampling techniques to simulate a sufficient number of cases and compute posteriors from them. Researchers have raised questions whether these

inference methods can also be extended for multiagent inference in MSBNs. However, to our knowledge, no such investigation has been attempted.

This paper presents the first investigation on this issue. Because of the wide variety of methods for single-agent inference in BNs, we do not attempt to consider them all. In particular, we consider extensions of an exact method, *loop cutset conditioning* [9], and two approximate methods, *forward sampling* [2] and *Markov sampling* [9] for multiagent inference in MSBNs. We compare their performance with the LJF method with a focus on agent autonomy and agent privacy.

The three chosen methods are some of the earliest methods proposed for inference in BNs. Many newly proposed methods are their variations. For instance, likelihood weighting [11] was developed to improve the performance of forward sampling. The two chosen approximate methods are sufficiently distinct in that forward sampling follows the topological order of the BN while Markov sampling relies on the Markov blanket. Investigation on how these methods may be extended to multiagent inference will provide valuable lessons to investigating the extensions of the variations of these methods. We have left out variable elimination methods as they are mostly used for answering a single probabilistic query and appear unsuited to multiagent applications, where agents would need to answer different queries in parallel. We have also left out search based inference methods. Since these methods tend to make use of topological order of the BN, we expect that our investigation of forward sampling will shed light on their extensions to MSBNs.

Section 2 introduces MSBNs and the LJF inference method. Section 3 first briefly reviews the method of loop cutset conditioning for single-agent BNs, and then presents a distributed version with its properties analyzed. Similarly, Sections 4 and 5 briefly review the single-agent forward sampling and Markov sampling, and then present and analyze their distributed versions, respectively. Conclusions are drawn in Section 6 based on these analyses.

2. MSBNs and inference with LJFs

An MSBN [17] M is a collection of Bayesian subnets that together defines a BN. To ensure exact inference, subnets are required to satisfy certain conditions. First we introduce terminologies to describe these conditions. Let $G_i = (V_i, E_i)$ $(i = 0, 1)$ be two distinct graphs (directed or undirected). G_0 and G_1 are said to be *graph-consistent* if the subgraphs of G_0 and G_1 spanned by $V_0 \cap V_1$ are identical. Given consistent graphs $G_i = (V_i, E_i)$ $(i = 0, 1)$, the graph $G = (V_0 \cup V_1, E_0 \cup E_1)$ is called the *union* of G_0 and G_1, denoted by $G = G_0 \sqcup G_1$. Given a graph $G = (V, E)$, V_0, and V_1 such that $V_0 \cup V_1 = V$ and $V_0 \cap V_1 \neq \emptyset$, and subgraphs G_i of G spanned by V_i $(i = 0, 1)$, we say that G is *sectioned* into G_0 and G_1. Fig. 1 shows an example.

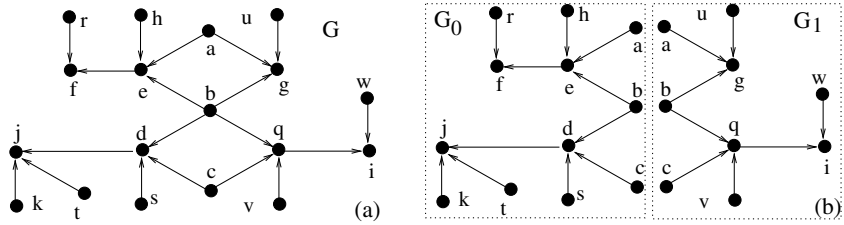

Fig. 1. The graph G in (a) is sectioned into G_0 and G_1 in (b).

The subnets in an MSBN must satisfy a hypertree condition:

Definition 1. Let $G = (V, E)$ be a connected graph sectioned into subgraphs $\{G_i = (V_i, E_i)\}$ such that the G_i's can be associated with a tree Ψ with the following property: Each node in Ψ 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`.

Fig. 2 shows an example. For this example, $V_2 \cap V_1 = \emptyset$ (hence the hypertree condition is trivially satisfied). But in general, $V_i \cap V_j$ can be non-empty.

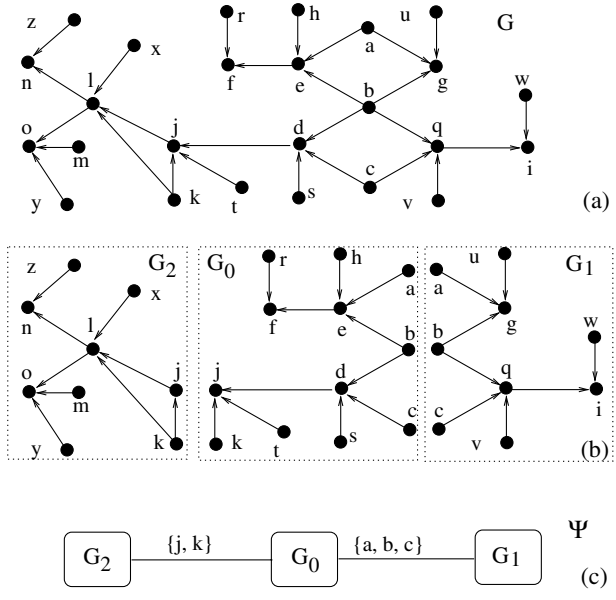

Fig. 2. The graph G in (a) is sectioned into G_0, G_1 and G_2 in (b). Ψ in (c) is a hypertree over G.

The interface between subnets in an MSBN must form a *d-sepset*:

Definition 2. Let G be a directed graph such that a hypertree over G exists. Let x be a node that is contained in more than one subgraph and $\pi(x)$ be its parents in G. Then x 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.

Each of a, b, c, j, k in the interfaces of Fig. 2(b) is a d-sepnode. Hence, the interfaces $\{a, b, c\}$ and $\{j, k\}$ are d-sepsets. If the arc from j to l were reversed, however, the node j would no longer be a d-sepnode and $\{j, k\}$ would no longer be a d-sepset. The hypertree and d-sepset conditions together ensures *syntactically* that agents can inform each other by passing their believes on interfaces only.

In a multiagent system, a d-sepnode is shared by more than one agent and is called a *public* node. A node internal to a single-agent is called a *private* node. The structure of an MSBN is a multiply sectioned DAG (MSDAG) with a hypertree organization:

Definition 3. A `hypertree MSDAG` $G = \bigsqcup_i G_i$, where each G_i is a DAG, is a connected DAG such that (1) there exists a hypertree Ψ over G, and (2) each hyperlink in Ψ is a d-sepset.

Fig. 2(b) and (c) show a hypertree MSDAG. An MSBN is then defined as follows, where a *potential* over a set of variables is a non-negative distribution of at least one positive parameter, and a *uniform potential* consists of 1's only.

Definition 4. An MSBN M is a triplet $(V, G, \mathscr{P})$. $V = \bigcup_i V_i$ is the `domain` where each V_i is a set of variables, called a `subdomain`. $G = \bigsqcup_i G_i$ (a hypertree MSDAG) is the `structure` where nodes of each DAG G_i are labeled by elements of V_i. Let x be a variable and $\pi(x)$ be all parents of x in G. For each x, exactly one of its occurrences (in a G_i containing $\{x\} \cup \pi(x)$) is assigned $P(x|\pi(x))$, and each occurrence in other DAGs is assigned a uniform potential. $\mathscr{P} = \prod_i P_i$ is the joint probability distribution (`jpd`), where each P_i is the product of the potentials associated with nodes in G_i. A triplet $S_i = (V_i, G_i, P_i)$ is called a `subnet` of M. Two subnets S_i and S_j are said to be adjacent if G_i and G_j are adjacent.

Fig. 3 shows a trivial MSBN. It is *trivial* because it does not reflect the large scale of a practical MSBN and only serves for the purpose of illustration. In (a), a trivial digital system of three units is shown. It is modeled into an MSBN with the hypertree MSDAG in (b) (a duplication of Fig. 2(b) for reader's convenience), where the hypertree has the topology $G_2 - G_0 - G_1$. Ignore (c) for the moment.

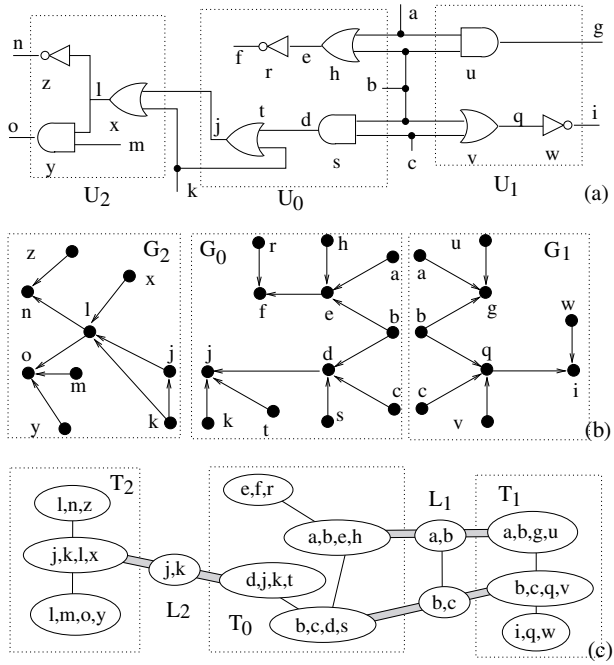

Fig. 3. A trivial MSBN.

An MSBN can be used as the knowledge representation of a cooperative multiagent system, where each subnet encodes the partial knowledge of an agent on the domain. How to use the MSBN framework for equipment monitoring and diagnosis is illustrated in [21]. We denote by A_i the agent whose knowledge is encoded in subnet S_i. Each A_i can only observe locally. Once a multiagent MSBN is constructed, agents may perform probabilistic inference by computing the query $P(x|e)$, where x is any variable within the subdomain of an agent, and e denotes the observations made by all agents. The key computation is to propagate the impact of observations to all agents, which is termed as *communication*. It is performed by inter-agent message passing. Hence communication requires *system-wide inter-agent message passing* or simply *system-wide message passing*. As agents are autonomous, constant system-wide message passing is either unavailable or undesirable. Most of the time, each agent A_i computes the query $P(x|e_i, \overline{e_i}')$, where e_i is the local observations made by A_i and $\overline{e_i}'$ is the observations made by other agents up to the latest communication. Note that $\overline{e_i}'$ is not explicitly available to A_i and only its impact is propagated to A_i. This computation is termed *local inference*.

Each subnet may be multiply connected. Multiple undirected paths may also exist across different subnets (e.g., those between d and q in Fig. 3(b)).

To facilitate exact inference with message passing, the LJF method compiles each subnet into a JT, called a local JT, and converts each d-sepset into a JT, called a linkage tree. Fig. 3(c) illustrates the three local JTs and two linkage trees of the monitoring system. Each oval in a JT or a linkage tree represents a subset of variables. For instance, $\{l, m, o, y\}$ is a cluster in the JT T_2 and $\{b, c\}$ is a cluster in the linkage tree L_1. Local inference is performed by message passing in the local JT as in the case of a single-agent BN. Inter-agent message passing during communication is performed using the linkage trees. A communication requires passing $O(2g)$ inter-agent messages, where g is the number of agents. The size of the message is linear on the number of clusters in the linkage tree and is exponential on the cardinality of the largest cluster.

During construction [18] of an MSBN, whether a public node has any parent or child in a subnet (but *not* how many or what they are) needs to be revealed. The conditional probability distribution of a public node may be negotiated among relevant agents to pool the diverse expertise together. Other than these, construction of an MSBN [18], its compilation into an LJF [20], and communication using LJF [19] reveal *no* additional information regarding the internals of each agent. Respect of agent privacy promotes agent autonomy, eases independent agent development, and facilitates integration of agents from different vendors into a coherent system.

In the following sections, we consider extending three common inference methods in single-agent BNs to inference in multiagent MSBNs. Based on agent autonomy and privacy, we assume (1) that constant system-wide message passing is not available, (2) that the d-sepnodes are *public*, but all other variables in each subnet and their associated distributions are *private* to the corresponding agent, and (3) that only peer-to-peer coordination is available with no centralized control.

3. Loop cutset conditioning

3.1. Single-agent oriented

Given observed values of some variables in a BN, the posterior probabilities of other variables may be computed. Pearl [9] proposed a method, called $\lambda - \pi$ message passing, that performs this computation efficiently. Local probability distributions are propagated through the BN from the leaf nodes towards the root nodes (called λ messages) as well as from the root nodes towards the leaf nodes (called π messages). Each node in the BN receives a λ message from each child node and a π message from each parent node. It computes its own λ and π messages based on its local distribution and messages received. After $\lambda - \pi$ message passing, the probability distribution of each variable conditioned on

the observation is available at the corresponding node. Unfortunately, the method is not applicable (with the same effect) directly to a non-tree BN.

The method of cutset conditioning [9] converts a multiply connected BN into multiple tree-structured BNs, which allows $\lambda - \pi$ message passing to be applied to each of them. The results are then combined to obtain the correct posteriors. The key step of the method is to hypothetically observe a set C of nodes, the *loop cutset*, in the multiply connected BN so that all loops are broken.

Formally, the posterior probability distribution of a variable x given observation e_1 on another variable is computed by

$$P(x|e_1) = \sum_{\mathtt{c}} P(x|\mathtt{c}, e_1)P(\mathtt{c}|e_1), \tag{1}$$

where $\mathtt{c} = (c_1, \ldots, c_n)$ is any configuration of C. For each $\mathtt{c}$, $P(x|\mathtt{c}, e_1)$ is obtained by $\lambda - \pi$ message passing in a corresponding tree-structured BN. $P(\mathtt{c}|e_1)$ can be calculated as

$$P(\mathtt{c}|e_1) = \alpha P(e_1|\mathtt{c})P(\mathtt{c}), \tag{2}$$

where α is a normalizing constant and $P(e_1|\mathtt{c})$ is obtained by $\lambda - \pi$ message passing. If the observations consist of values e_1, e_2 of two variables, $P(x|e_1, e_2)$ can be obtained by

$$P(x|e_1, e_2) = \sum_{\mathtt{c}} P(x|\mathtt{c}, e_1, e_2)P(\mathtt{c}|e_1, e_2), \tag{3}$$

where $P(x|\mathtt{c}, e_1, e_2)$ can be obtained by $\lambda - \pi$ message passing, and

$$P(\mathtt{c}|e_1, e_2) = \alpha P(e_2|\mathtt{c}, e_1)P(\mathtt{c}|e_1), \tag{4}$$

with $P(\mathtt{c}|e_1)$ computed by Eq. (2) and $P(e_2|\mathtt{c}, e_1)$ by $\lambda - \pi$ message passing.

To obtain $P(\mathtt{c})$ in Eq. (2), an ancestral ordering of variables is used to compute the following factors [13] whose product is $P(\mathtt{c})$:

$$P(c_1), P(c_2|c_1), \ldots, P(c_n|c_1, \ldots, c_{n-1}). \tag{5}$$

If the observations consist of m values $e_1, \ldots, e_m$, the above can be generalized to compute in sequence

$$P(e_1|\mathtt{c}), P(e_2|\mathtt{c}, e_1), \ldots, P(e_m|\mathtt{c}, e_1, \ldots, e_{m-1}) \tag{6}$$

to obtain

$$P(\mathtt{c}|e_1), P(\mathtt{c}|e_1, e_2), \ldots, P(\mathtt{c}|e_1, \ldots, e_m), \tag{7}$$

and compute

$$P(x|\mathtt{c}, e_1, \ldots, e_m) \tag{8}$$

to obtain

$$P(x|e_1, \ldots, e_m). \tag{9}$$

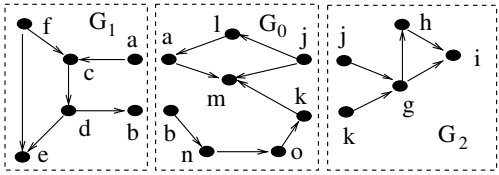

Fig. 4. The structure of a trivial MSBN.

3.2. Inference in MSBNs by distributed cutset conditioning

In an MSBN, loops can exist both within local DAGs and across multiple local DAGs. Finding a loop cutset C that can break all loops requires a distributed search. It can be performed using a variation of TestAcyclicity [18], a polynomial algorithm for verifying acyclicity of the structure of an MSBN. A *simple* variation works as follows:

Each agent starts by recursively marking *terminal* nodes (with no more than one adjacent node). After all such nodes have been marked, one agent marks a node s and adds s to C (through message passing as there is no centralized control). At least one loop is now cut open. Repeat the above until all nodes are marked. See [18] for details on multiagent node marking. Note that because a node s added to C could be a private node, communicating C to other agents reveals private information.

In Fig. 4, $C = \{f, h, j, o\}$ is a loop cutset. Note that f is private to A_1, h is private to A_2, o is private to A_0, and j is shared by A_2 and A_0.

After such a cutset is found, the multiply connected DAG union needs to be converted to $O(2^{|C|})$ tree-structured DAG unions, one for each configuration c of C, and distributed $\lambda - \pi$ message passing needs to be performed in each of them. [1] We consider the computation of sequences (5)–(9) where the observations $e_1, \ldots, e_m$ are those obtained since the last communication. Note that $e_1, \ldots, e_m$ as well as variables in C are distributed among agents. No single-agent has access to all of them.

First, consider the computation of sequence (5). This computation needs to be performed following an ancestral ordering of variables in the domain. The ordering can be defined through another simple variation of TestAcyclicity [18], described as follows. Recall from Section 2, the total number of agents is denoted by g and the agents are indexed $0, 1, \ldots, g-1$.

Recursively mark root nodes in all g agents in multiple rounds. At most one node per agent can be marked at each round. At the ith round, a node marked

[1] Equivalently, one could process the same DAG union $O(2^{|C|})$ times once for each distinct c. It is a matter of implementation.

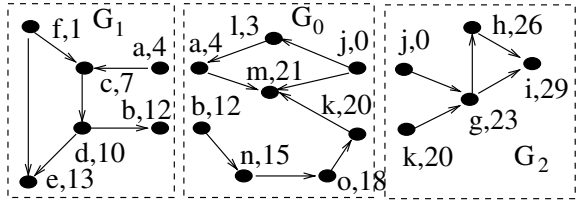

Fig. 5. Ancestral ordering where the index of each node is shown beside the label.

by the kth agent is assigned the index $i * g + k$. The indexes then define an ancestral ordering.

Fig. 5 shows an example. The available indexes for A_0 are 0, 3, 6, ... and those for A_1 are 1, 4, 7, ... In round 0, A_0, by cooperating with A_2, recognizes that the public node j is a root and indexes it with 0. A_1 indexes f with 1. A_2 is notified by A_0 with the index of j, but otherwise it has no root node to index. In round 1, A_0 indexes l with 3 and A_1 indexes a with 4. The process continues until all nodes are indexed.

The above method can be improved such that more than one node can be marked per agent per round. This will result in less number of inter-agent messages. Since the computation can be performed off-line and its complexity is polynomial, we will not pursue the refinement here.

Using the ancestral ordering, sequence (5) can be obtained by extending the method of Suermondt and Cooper [13]. For the example in Fig. 5 and the loop cutset $C = \{f, h, j, o\}$, the sequence

$$P(j), P(f|j) = P(f), P(o|f, j), P(h|f, j, o)$$

can be computed. Because there is no centralized control, it is quite involved to *coordinate* the computation so that the ancestral ordering is followed.

First, for each configuration c arranged in the given ancestral ordering $(c_1, \ldots, c_n)$, the agent with c_i must instantiate the corresponding variable (effectively cutting one or more loops open) according to the order, and messages need to be passed among agents after the instantiation if c_{i+1} is contained in a distinct agent. Second, message passing must be performed partially within each agent because part of its local DAG may still belong to loops where cutset variables are yet to be instantiated. For instance (Fig. 5), after instantiating variable j, A_0 can only propagate its impact to l but not to m, because m still belongs to a loop in which no variable has been instantiated. Third, the message passing may involve each agent multiple times and hence activities of agents must be carefully coordinated. For example, after A_0 instantiated j, its impact needs to be propagated to l and a locally, then to c, d and b in A_1, and finally back to n and o within A_0 again.

The sequence (5) needs to be computed for each configuration $\mathtt{c}$ and the results need to be propagated to at least one agent to compute $P(C)$. The agent can then send $P(C)$ to every other agent for their local usage.

Next, consider the computation of sequence (6). Given $\mathtt{c}$, sequence (6) can be obtained by m message propagations in each corresponding DAG union. Since $e_1, \ldots, e_m$ are distributed and the sequence needs to be computed in order, agents must *coordinate* the computation. After the first propagation over the system, the agent with e_1 obtains $P(e_1|\mathtt{c})$. It then enters e_1, followed by the second propagation over the system. The agent with e_2 obtains $P(e_2|\mathtt{c}, e_1)$, and the process continues. Note that results for sequence (6) are distributed because each e_i may be contained in a different agent. For each $\mathtt{c}$, probability (8) can be obtained through one round coordinated computation with each agent selecting its local x.

From sequences (6) and (7) can be obtained similarly to Eq. (4). Since the results for sequence (6) are distributed, this computation needs to be *coordinated*. The agent with $P(e_1|\mathtt{c})$ computes $P(\mathtt{c}|e_1)$ through Eq. (2). Note that to derive the normalizing constant, the computation cannot be performed until sequence (6) has been computed for each $\mathtt{c}$. It then sends $P(\mathtt{c}|e_1)$ to other agents. The agent with $P(e_2|\mathtt{c}, e_1)$ will then compute $P(\mathtt{c}|e_1, e_2)$ and sends it. The process continues then at the next agent.

From the last probability of sequence (7) and probability (8), each agent will be able to compute probability (9). Note that although the computation for sequences (6) and (7) can be performed in any order of $e_1, \ldots, e_m$, the order must be agreed and followed consistently by all agents for both sequences.

Local evidential inference cannot be performed when the system-wide message passing is absent. For instance, A_0 cannot perform local cutset conditioning using its subnet only, since the dependence through subdomains in other agents cannot be counted for. Between communications, approximate local inference using only the local subnet is possible, but the posteriors obtained is *not* exact, and can be significantly different from what will be obtained after global communication.

We summarize the features of distributed loop cutset conditioning:

1. Distributed search for loop cutset and ancestral ordering can be performed off-line. The rest of the computation must be performed on-line since the computation depends on the observations $e_1, \ldots, e_m$. Note that $P(C)$ must be computed on-line as well since it is a different distribution at the beginning of each communication.
   Alternatively, because observations may cut some loops open, the cutset C may be redefined for each communication. Including observations in the cutset can cut down the amount of computation. The tradeoff is that computation for defining a new cutset must be performed on-line.
2. The computation of $P(C)$ for each $\mathtt{c}$ requires $\mathrm{O}(|C|)$ rounds of system-wide message passing since a message passing is needed after the instantiation of

each variable in C. If computations for all $\mathtt{c}$'s are performed sequentially, $O(|C|2^{|C|})$ rounds of message passing are needed. To reduce inter-agent message passing, the $O(2^{|C|})$ messages, one for each $\mathtt{c}$, may be batched, making $O(|C|)$ rounds of message passing sufficient with each message $O(2^{|C|})$ times long.

The computation of sequences (6) and (7) requires one round of system-wide message passing for each element in sequences (6) and (7). Hence $O(m)$ rounds of message passing are needed, with message batching.

Overall, $O(|C| + m)$ rounds of system-wide message passing are needed. Compared with the LJF method that requires two rounds of message passing, distributed cutset conditioning incurs much higher inter-agent transmission cost.

3. Local evidential inference cannot be performed exactly using cutset conditioning, although approximation using the local subnet provides a suboptimal alternative.
4. At least the number of nodes in the loop cutset and the number of variables observed system-wide must be revealed. Partial information regarding the ancestral ordering of domain variables is also revealed.

4. Forward sampling

4.1. Single-agent oriented

Forward sampling is also known as *logic sampling* [2]. The method of forward sampling proceeds in an ancestral ordering of nodes in a BN. For each node, its value is randomly simulated given the values of its parent nodes according to the conditional probability distribution stored in the node. A *case* is simulated when the value of each variable in the BN has been simulated. By repeating the simulation for a large number of cases, a *sample* is obtained according to the jpd of the BN. The cases that are incompatible with the given observations are discarded, and the posterior distributions for unobserved variables are approximated by frequency counting on the remaining cases. The larger the sample size, the more accurate the posteriors.

Case simulation in forward sampling does not consider the observations. When the observed events are rare, a large percentage of cases simulated may be discarded.

4.2. Inference in MSBNs by distributed forward sampling

According to forward sampling, simulation must be performed in an ancestral ordering. That is, the value of a child node is determined after the values of all its parents have been determined. The ordering can be obtained by a

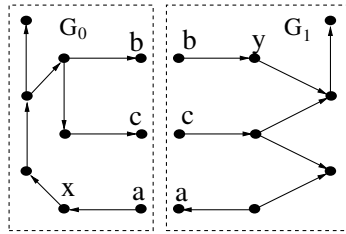

Fig. 6. Two adjacent local DAGs in an MSBN.

distributed search similar to that for distributed cutset conditioning. In an MSBN, the parents of a node may be located in an adjacent agent. To simulate the value of the node, an inter-agent message must be passed.

Consider the scenario illustrated in Fig. 6. Agent A_0 contains a private variable x. Its parent a is a public variable shared with agent A_1 and the parent $\pi(a)$ of a is private in A_1. Thus, to determine the value of x, A_0 needs to wait until A_1 sends the value of a to it. Furthermore, A_1 contains a private variable y, A_1 shares the parent b of y with A_0, the parent $\pi(b)$ of b is private in A_0 and x is an ancestor of $\pi(b)$. Hence, A_1 cannot determine the values for all variables it shares with A_0 and as a result cannot send them in one batch. Instead, A_1 must wait until A_0 sends the value of $\pi(y)$.

Although scenarios such as the above involve mutual wait among the two agents, it does not lead to a deadlock because the DAG union of an MSBN is acyclic. However, if a directed path crosses the interface between two agents k times, then messages must be passed back and forth k times between the two agents before values for all variables on the path are simulated. Note that a directed path may pass across multiple agent interfaces. Hence during simulation of each case, the number of messages between each pair of adjacent agents are upper-bounded by the number of d-sepnodes between them. This implies that $O(gd)$ inter-agent messages are needed to simulate one case, where g is the number of agents and d is the cardinality of the maximum d-sepset.

If the cases are generated one by one, the above mentioned cost for inter-agent message passing will be multiplied by the sample size. To reduce this cost, inter-agent messages can be batched. For example, A_1 may generate K values for a and pass all of them to A_0 and later receives K values for each of b and c. The price to be paid is that all K values for each variable must be saved in some way until the compatibility of each case with observations is resolved as discussed below and the contributions of the K cases to the posteriors are counted.

When the cases are generated one by one, the generation of a case can be terminated early as soon as one agent found the partial case to be incompatible with its local observations. Because batched sampling is intended to reduce inter-agent message passing, such finding by an agent cannot be communicated

to other agents immediately. After a sample of cases is simulated, it is necessary to determine which cases are compatible with the observations. This can be achieved by letting each agent label each case that is incompatible with its local observations. All such labels must then be passed among all agents to weed out each case that is labeled as incompatible by any agent.

Since parents of a variable may not be contained in an agent, local inference in the absence of system-wide message passing cannot be performed equivalently. An alternative is that, at the end of each communication, each agent records down the posterior distribution of each shared variable that is a root locally, and use the local subnet thus obtained for local inference. For the example in Fig. 6, A_1 may record down $P(b|\mathtt{e})$ and $P(c|\mathtt{e})$, where $\mathtt{e}$ stands for the observations made by all agents before the last communication. After new local observations are made, A_1 can then perform a local inference with a local forward sampling. The result, however, is *not* equivalent to what would be obtained with system-wide message passing, even when there is no observation other than that from A_0, since the dependence between b and c through the loops in A_0 is not counted for.

We summerize features of distributed forward sampling:

1. Distributed search for an ancestral ordering is needed.
2. With message batching, $O(dg)$ inter-agent messages are needed. The length of each message is in the order $O(Kd)$, where K is the sample size. Both values for shared variables and compatibility labels for cases need to be transmitted between agents. In comparison, communication using a LJF passes $O(2g)$ inter-agent messages, which requires $d/2$ times less inter-agent message passing.
3. Local inference in the absence of system-wide message passing does not converge to the correct posteriors in general.
4. Partial information regarding the ancestral ordering of domain variables is revealed.

5. Markov sampling

5.1. Single-agent oriented

Markov sampling [9] is also known as *Gibbs sampling* [4]. Unlike forward sampling, Markov sampling starts simulation of each case by instantiating observed variables to the observed values. Hence, no case simulated will be discarded before the frequency counting. The simulation begins with some initial case s^0. Each subsequent case s^i ($i > 0$) is simulated by conditioning on the previous case s^{i-1}. To simulate the value of variable x in s^i, only the values of s^{i-1} in the *Markov blanket* of x are needed. The Markov blanket of x includes the parents $\pi(x)$ of x, each child v of x and the parents of v, excluding x. The

value of x is simulated using the distributions $P(x|\pi(x))$ and $P(v|\pi(v))$ for each child v, where $\pi(x)$, v and $\pi(v)\backslash\{x\}$ are instantiated to the values in s^{i-1}. The simulation can be performed in arbitrary order of variables. A case is simulated after the value of each variable is simulated.

Markov sampling is more efficient than forward sampling but the posteriors may not converge to the true probabilities when local distributions contain extreme probability values.

5.2. Inference in MSBNs by distributed Markov sampling

In Markov sampling, simulations of any two non-adjacent variables in case s^i are *independent* given the values of their Markov blankets in s^{i-1}, assuming none of them is in the Markov blanket of the other. Therefore, simulation needs not to follow an ancestral ordering. This means that multiple passes through a single subnet during the simulation of each case (as in distributed forward sampling) is no longer required. In other words, simulation of each case requires only $O(g)$ inter-agent messages.

On the other hand, in distributed forward sampling, simulations of any two cases are independent, which makes batching messages possible. In distributed Markov sampling, at the time the value of x in s^i is to be simulated, the value of its Markov blanket in s^{i-1} must be known. This renders message batching *impossible*. To see this, consider a directed chain u, v, x, y, z in a MSDAG where u has no parent and z has no child (the first line of Fig. 7). In the figure, solid arrows represent probabilistic causal dependence. Each subsequent line indexed by i represents the ith simulated case. The dashed arrows represent the dependence relation among variable values simulated. It can be seen that to simulate u in the fifth case, the value of all variables at various previous cases must be known. Hence, it is impossible to simulate, say, the value of u in the 1000th case without completing the simulation of the first 996 cases. The consequence of this limitation is serious. It implies that cases must be simulated

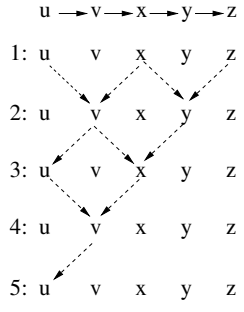

Fig. 7. The dependence relation among simulated variable values in different cases.

sequentially. The number of inter-agent messages will be in the order $O(Kg)$, where K is the intended sample size (normally a very large integer).

To simulate the value of a variable x, an agent must have access to factors in the following product from its *Markov blanket*:

$$P(x|\pi(x))\prod_{v_i(x)} P(v_i(x)|\pi(v_i(x))),$$

where each $v_i(x)$ is a child of x. The following proposition ensures that these factors are available to an agent if x is a private variable.

Proposition 5. *Let x be a private node of an agent A_i. Then the Markov blanket of x is contained in A_i.*

Proof. We consider a parent y of x, a child v of x and another parent z of v. Since x is private to A_i, y and v must be contained in A_i. The child v may be private or public to A_i. If v is private, then z must be contained in A_i as argued above. If v is public to A_i, then v is a d-sepnode. By definition, $\pi(v)$ (all parents of v in the domain) must be contained in at least one agent. Since the parent x of v is private, the only such agent is A_i. Hence, z is contained in A_i. □

If x is a public node (d-sepnode), the situation is different. Fig. 8 shows an example. Since x is a d-sepnode, all its parents must be contained in at least one agent. As is shown, A_0 is such an agent. It is possible, however, the members of Markov blanket of x are private to different agents. That is, $\{y, w\}$ is private to A_0, $\{c, z\}$ is private to A_1, and $\{d, v\}$ is private to A_2. At least one of them must be chosen to simulate the value for x, through a distributed election mechanism [15]. One such mechanism is the *bully* algorithm in which agents are given unique ids and the agent with the highest id among those in the election will be chosen through election message passing.

For the chosen agent to carry out the task, the other two agents must reveal some private information. For example, if A_0 is the chosen agent, then A_1 must

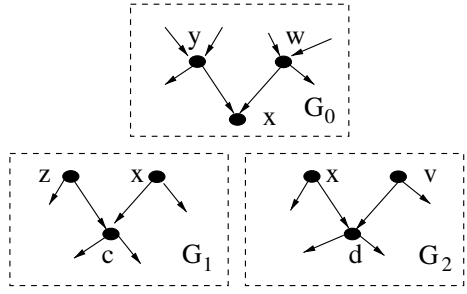

Fig. 8. Three adjacent local DAGs in an MSBN where x is public.

reveal to A_0: (1) that it has a private child c of x, (2) that the private child has a private parent z in addition to x in A_1, (3) the distribution $P(c|x,z)$, and (4) the simulated values of c and z for each case. A_2 must reveal the similar private information to A_0. Note that if any of these private variables are observed, then the observed values will have to be revealed as well.

Local inference without system-wide message passing suffers from the same problem as distributed forward sampling. Namely, dependence due to loops outside an agent cannot be counted for. Since Markov sampling may not converge even in the single-agent case if extreme probability values exist, this limitation of local inference may further increase the error bound.

We summerize features of the distributed Markov sampling:

1. The order in which agents conduct simulation is flexible. No simulated cases need to be discarded.
2. Batching messages is impossible, resulting in $O(Kg)$ inter-agent messages. Since K is a large integer, it requires $K/2$ times more inter-agent messages than the LJF method.
3. Local inference does not converge to the same posteriors as system-wide message passing.
4. Private information on the Markov blanket of each public variable must be revealed.

6. Conclusion

In this work, we consider extending three common inference methods in BNs to distributed multiagent inference in MSBNs. We outline how each method can be performed in a distributed fashion using only peer-to-peer coordination. We analyze how each method measures with respect to two important goals of multiagent systems, agent autonomy and privacy. We compare the three alternatives with the LJF method. In particular, we compare them along four dimensions: the complexity of off-line compilation, the amount of inter-agent messages during communication, the support of consistent local inference, and the private information revealed. Table 1 summarizes the comparison.

The LJF method requires off-line compilation of the LJF. Distributed cutset conditioning (DCC) requires off-line search for a loop cutset and an ancestral ordering. Distributed forward sampling (DFS) requires off-line search of an ancestral ordering. Distributed Markov sampling (DMS) requires no off-line compilation. The amount of off-line compilation can be compared as

$$\text{LJF} > \text{DCC} > \text{DFS} > \text{DMS},$$

where LJF has the most sophisticated off-line computation and DMS has none.

Table 1
Comparison of different multiagent inference methods

	Off-line compilation	Inter-agent messages	Consistent local inference	Reveal private info
LJF	****	*	Yes	No
DCC	***	***	No	Yes
DFS	**	**	No	Yes
DMS	*	****	No	Yes

The number of stars indicates the relative amount of computation. More stars signify a higher amount of computation.

For a multiagent system of g agents, communication with the LJF method can be performed with $O(2g)$ inter-agent messages. The order is $O((|C| + m)g)$ for distributed cutset conditioning where C is the cutset and m is the number of observed variables. Note that $|C|$ is upper-bounded by the number of loops in the MSDAG. Distributed forward sampling passes $O(dg)$ inter-agent messages where d is the cardinality of the maximum d-sepset. Commonly, we have $|C| + m > d$. Distributed Markov sampling passes $O(Kg)$ inter-agent messages where K is the sample size. The amount of inter-agent messages during communication can be compared as

$$\text{DMS} > \text{DCC} > \text{DFS} > \text{LJF},$$

where DMS has the most frequent inter-agent message passing while LJF has the least. In this comparison, we have focused on the number of inter-agent messages (versus the length of each message). This is based on the assumption that each message has a cost function $\alpha C_1 + C_2$, where C_1 is the connection cost between a given pair of agents, C_2 is the cost depending on the length of the message, and α is a parameter that quantifies how undesirable it is to pass messages between agents frequently. It is assumed that αC_1 is identical across methods (which is valid) and is much larger than C_2 no matter which method is used (which is a reasonable approximation).

LJF is the only method among the four that supports consistent local inference in the absence of system-wide message passing. Local inference using the other three methods may lead to errors of more or less arbitrary size.

The LJF method reveals no private information. DCC reveals the size of cutset and the number of observed variables, as well as partial information on ancestral ordering. DFS reveals partial information on ancestral ordering. DMS reveals private information on the Markov blanket of each d-sepnode.

Our investigation is the first on extending non-junction tree based methods from single-agent inference to multiagent inference. This analysis and comparison provide insight to the issues involved in multiagent probabilistic reasoning and facilitate investigation of extensions of many other single-agent inference methods to which the three chosen methods are representatives. Our

investigation also serves those who implement multiagent inference systems as a guide about the pros and cons of alternatives.

Acknowledgements

This work is supported by Research Grant OGP0155425 from NSERC of Canada. I thank anonymous reviewers of my past work for inspiring this investigation and reviewers of earlier drafts of this paper for helpful comments.

References

[1] R. Dechter, Bucket elimination: a unifying framework for probabilistic inference, in: E. Horvitz, F. Jensen (Eds.), Proc. 12th Conf. on Uncertainty in Artificial Intelligence, Portland, Oregon, 1996, pp. 211–219.
[2] M. Henrion, Propagating uncertainty in Bayesian networks by probabilistic logic sampling, in: J.F. Lemmer, L.N. Kanal (Eds.), Uncertainty in Artificial Intelligence, vol. 2, Elsevier Science Publishers, 1988, pp. 149–163.
[3] M. Henrion, Search-based methods to bound diagnostic probabilities in very large belief nets, in: Proc. 7th Conf. Uncertainty in Artificial Intelligence, San Francisco, 1991, pp. 142–150.
[4] F.V. Jensen, An Introduction To Bayesian Networks, UCL Press, 1996.
[5] F.V. Jensen, S.L. Lauritzen, K.G. Olesen, Bayesian updating in causal probabilistic networks by local computations, Computat. Stat. Quart. 4 (1990) 269–282.
[6] V.R. Lesser, L.D. Erman, Distributed interpretation: a model and experiment, IEEE Trans. Comput. C-29 (12) (1980) 1144–1163.
[7] Z. Li, B. D'Ambrosio, Efficient inference in Bayes' nets as a combinatorial optimization problem, Int. J. Approx. Reason. 5 (1994) 55–81.
[8] A.L. Madsen, F.V. Jensen, Lazy propagation in junction trees, in: Proc. 14th Conf. Uncertainty in Artificial Intelligence, 1998.
[9] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Morgan Kaufmann, Morgan Kaufmann, 1988.
[10] D. Poole, Probabilistic horn abduction and Bayesian networks, Artificial Intelligence 64 (1) (1993) 81–129.
[11] R.D. Shachter, M.A. Poet, Proc. 5th Workshop on Uncertainty in Artificial Intelligence, in: Simulation approaches to general probabilistic inference on belief networks, Windsor, Ontario, 1989, pp. 311–318.
[12] G. Shafer, Probabilistic Expert Systems, Society for Industrial and Applied Mathematics, Philadelphia, 1996.
[13] J. Suermondt, G. Cooper, Initialization for the method of conditioning in Bayesian belief networks, Artificial Intelligence 50 (1991) 83–94.
[14] K.P. Sycara, Multiagent systems, AI Magazine 19 (2) (1998) 79–92.
[15] A.S. Tanenbaum, Distributed Operating Systems, Prentice Hall, 1995.
[16] M. Wooldridge, N.R. Jennings, Intelligent agents: theory and practice, Knowledge Eng. Rev. 10 (2) (1995) 115–152.
[17] Y. Xiang, A probabilistic framework for cooperative multi-agent distributed interpretation and optimization of communication, Artificial Intelligence 87 (1–2) (1996) 295–342.
[18] Y. Xiang, Verification of dag structures in cooperative belief network based multi-agent systems, Networks 31 (1998) 183–191.

[19] Y. Xiang, Belief updating in multiply sectioned Bayesian networks without repeated local propagations, Int. J. Approx. Reason. 23 (2000) 1–21.
[20] Y. Xiang, Cooperative triangulation in MSBNs without revealing subnet structures, Networks 37 (1) (2001) 53–65.
[21] Y. Xiang, H. Geng, Distributed monitoring and diagnosis with multiply sectioned Bayesian networks, in: Proc. AAAI Spring Symp. on AI in Equipment Service, Maintenance and Support, Stanford, 1999, pp. 18–25.
[22] Y. Xiang, F.V. Jensen, Inference in multiply sectioned Bayesian networks with extended Shafer–Shenoy and lazy propagation, in: Proc. 15th Conf. Uncertainty in Artificial Intelligence, Stockholm, 1999, pp. 680–687.
[23] Y. Xiang, V. Lesser, Justifying multiply sectioned Bayesian networks, in: Proc. 6th Inter. Conf. Multi-agent Syst., Boston, 2000, pp. 349–356.