Comparing Alternative Methods for Inference in Multiply Sectioned Bayesian Networks

Y. Xiang
Dept. Computing & Information Science
University of Guelph, Ontario, Canada

Abstract

Multiply sectioned Bayesian networks (MSBNs) provide one framework for agents to estimate the state of a domain. Existing methods for multi-agent inference in MSBNs are based on linked junction forests (LJFs). The methods are extensions of message passing in junction trees for inference in single-agent Bayesian networks (BNs). We consider extending other inference methods in single-agent BNs to multi-agent inference in MSBNs. In particular, we consider distributed versions of loop cutset conditioning and forward sampling. They are compared with the LJF method in terms of off-line compilation, inter-agent messages during communication, consistent local inference, and preservation of agent privacy.

Introduction

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. Multiply sectioned Bayesian networks (MSBNs) (Xiang 1996) provide one framework to conduct such a task. 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 multi-agent inference in MSBNs are extensions of a class of methods for inference in single-agent Bayesian networks (BNs): message passing in junction trees (Jensen, Lauritzen, & Olesen 1990; Shafer 1996; Madsen & Jensen 1998). The *linked junction forest* (LJF) method (Xiang 1996) 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 (Xiang & Jensen 1999) 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. In this work, we consider extending two of them for multi-agent inference in MSBNs: *loop cutset conditioning* and *forward sampling*. We compare their performance with the LJF method with a focus on agent autonomy and agent privacy.

Section introduces MSBNs and the LJF inference method. Section presents a distributed loop cutset conditioning with its properties analyzed. Section analyzes distributed forward sampling.

MSBNs and inference with LJFs

An MSBN (Xiang 1996) 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 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. The subnets in an MSBN must satisfy a hypertree condition:

Definition 1 *Let $G = (V, E)$ be a connected graph sectioned into subgraphs $\{G_i = (V_i, E_i)\}$. Let the G_i's 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`.

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. 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.*

In a multi-agent 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.*

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* $(V, G, \mathcal{P})$*.* $V = \bigcup_i V_i$ *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.* $\mathcal{P} = \prod_i P_i$ *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.*

An MSBN supports knowledge representation of a cooperative multi-agent system, where each subnet encodes the partial knowledge of an agent on the domain. We denote by A_i the agent whose knowledge is encoded in subnet S_i. Each A_i can only observe locally. Once a multi-agent 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 we term as *communication*. It is performed by inter-agent message passing. Hence communication requires *system-wide inter-agent 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. 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. Local inference is performed by message passing in the local JT as for single-agent BNs. Inter-agent message passing during communication is performed using the linkage trees. A communication requires passing $O(2\ g)$ 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 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, its compilation into an LJF, and communication using LJF reveal *no* additional information regarding the internals of each agent.

In the following sections, we extend two common inference methods in single agent BNs to multi-agent 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 corresponding agent, and (3) that only peer-to-peer coordination is available.

Loop cutset conditioning

Single-agent oriented

Cutset conditioning (Pearl 1988) converts a multiply connected BN into multiple tree-structured BNs and performs the $\lambda - \pi$ message passing in each of them. The key step of the method is to hypothetically observe a set C of nodes, the *loop cutset*, so that all loops are broken. The posterior distribution of a variable x given observation e_1 is computed by

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

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

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

where $P(e_1|\mathtt{c})$ is obtained by $\lambda - \pi$. To obtain $P(\mathtt{c})$ in Eq. (2), an ancestral ordering of variables is used to compute the following factors (Suermondt & Cooper 1991) whose product is $P(\mathtt{c})$:

$$P(c_1), P(c_2|c_1), ..., P(c_n|c_1, ..., c_{n-1}). \tag{3}$$

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

$$P(e_1|\mathtt{c}), P(e_2|\mathtt{c}, e_1), ..., P(e_m|\mathtt{c}, e_1, ..., e_{m-1}) \tag{4}$$

to obtain

$$P(\mathtt{c}|e_1), P(\mathtt{c}|e_1, e_2), ..., P(\mathtt{c}|e_1, ..., e_m), \tag{5}$$

and compute in sequence

$$P(x|\mathtt{c}, e_1), P(x|\mathtt{c}, e_1, e_2), ..., P(x|\mathtt{c}, e_1, ..., e_m) \tag{6}$$

to obtain

$$P(x|e_1), P(x|e_1, e_2), ..., P(x|e_1, ..., e_m). \tag{7}$$

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 (Xiang 1998),

a polynomial algorithm for verifying acyclicity of the structure of an MSBN. A *simple* variation works as follows:

Start by recursively marking terminal nodes (with no more than one adjacent node) in each agent. After all such nodes have been marked, mark a tail-to-tail or head-to-tail node s in any agent and add s to C by propagation as there is no centralized control. At least one loop is now cut open. Mark recursively new terminal nodes until no more. Repeat the above until all nodes are marked. See (Xiang 1998) for details on multi-agent node marking.

In Figure 1, $C = \{f, h, j, o\}$ 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.

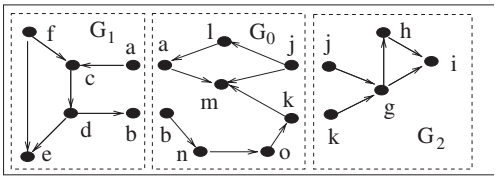

Figure 1: The structure of a trivial MSBN.

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 (3) through (7) where the observations $e_1, ..., e_m$ are those obtained since the last communication. Note that $e_1, ..., e_m$ as well as variables in C are distributed among agents.

First, consider the computation of sequence (3). 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 (Xiang 1998), described as follows:

Recursively mark root nodes in all g agents in multiple rounds. At most one node per agent can be marked at ith round and the node is given an index between i and $i+g-1$. The indexes then define an ancestral ordering.

Figure 2 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.

Using the ancestral ordering, sequence (3) can be obtained by extending the method of Suermondt and Cooper (Suermondt & Cooper 1991). For the example in Figure 2 and the loop cutset $C = \{f, h, j, o\}$, the sequence

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

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

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

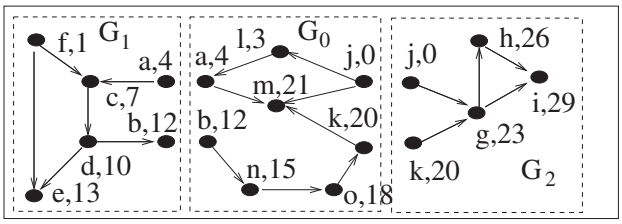

Figure 2: Ancestral ordering where the index of each node is shown beside the label.

First, for each configuration c arranged in the given ancestral ordering $(c_1, ..., c_n)$, the agent with c_i must instantiate the corresponding variable according to the order, since 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 since part of the DAG union still contains loops due to cutset variables yet to be instantiated. For instance (Figure 2), after instantiating variable f, A_1 can only propagate its impact to c and e but not beyond since the rest of the DAG union still contains loops. 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 (3) needs to be computed for each configuration 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 (4). Given c, sequence (4) can be obtained by m message propagations in each corresponding DAG union. Since $e_1, ..., e_m$ 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 $P(e_1|\mathsf{c})$. It then enters e_1, followed by the second propagation over the system. The agent with e_2 obtains $P(e_2|\mathsf{c}, e_1)$, and the process continues. Note that results for sequence (4) are distributed. Sequence (6) can be obtained through the same process with each agent selecting its local x.

From sequence (4), sequence (5) can be obtained similarly to Eq. (2). Since the results for sequence (4) are distributed, this computation needs to be *coordinated*. The agent with $P(e_1|\mathsf{c})$ computes $P(\mathsf{c}|e_1)$ through Eq. (2). Note that to derive the normalizing constant, the computation cannot be performed until sequence (4) has been computed for each c. It then sends $P(\mathsf{c}|e_1)$ to other agents. The agent with $P(e_2|\mathsf{c}, e_1)$ will then compute $P(\mathsf{c}|e_1, e_2)$ and sends it. The process continues then at the next agent.

From the results of sequences (5) and (6), each agent will be able to compute sequence (7). Note that although the computation for sequences (4) through (7) can be performed in any order of $e_1, ..., e_m$, the order must be agreed and followed consistently by all agents for all four 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 as follows:

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 $e_1, ..., e_m$. Note that $P(C)$ must be computed on-line.
2. The computation of $P(C)$ for each c requires $O(|C|)$ rounds of system-wide message passing. If computations for all 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 c, may be batched, making $O(|C|)$ rounds of message passing sufficient with each message $O(2^{|C|})$ times long. The computation of sequences (4) through (7) requires one round of system-wide message passing for each element in sequences (4) and (6). Hence $O(m)$ rounds of message passing are needed, with message batching. Overall, $O(|C| + m)$ rounds of message passing are needed.
3. Local inference cannot be performed exactly.
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.

Inference in MSBNs by distributed forward sampling

According to forward sampling (Henrion 1988), 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 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 following scenario: An agent A_0 contains a variable x but its parents $\pi(x)$ are contained only in another agent A_1. Thus, to determine the value of x, A_0 needs to wait until A_1 sends the values of $\pi(x)$ to it. Furthermore, A_1 contains a variable y but its parents $\pi(y)$ are contained only in A_0, and x is an ancestor of y. 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 values of $\pi(y)$. Figure 3 illustrates the situation, where A_0 must wait for the value of a from A_1, and A_1 must wait for the value of b from A_0. Since the DAG union of an MSBN is acyclic, a deadlock is not possible. 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

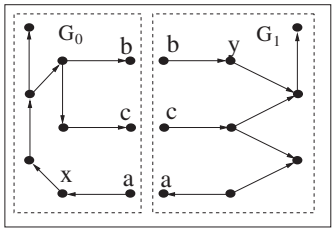

Figure 3: Two adjacent local DAGs in an MSBN.

agents are upper-bounded by the number of d-sepnodes between them. This implies that $O(g\ d)$ 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 Figure 3, A_1 may record down $P(b|\mathsf{e})$ and $P(c|\mathsf{e})$, where 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 as follows:

1. Distributed search for an ancestral ordering is needed.
2. With message batching, $O(d\ g)$ inter-agent messages are needed. The length of each message is in the order $O(K\ d)$, 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.

Conclusion

In this work, we compare three alternative methods for inference in multi-agent MSBNs 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.

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. The amount of off-line compilation compares as

$$LJF > DCC > DFS.$$

With g agents, communication by LJF can be performed with $O(2g)$ inter-agent messages. The order is $O((|C| + m)\ g)$ for DCC 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. DFS passes $O(d\ g)$ inter-agent messages where d is the cardinality of the maximum d-sepset. Commonly, we have $|C| + m > d$. The amount of inter-agent messages during communication compares as

$$DCC > DFS > LJF.$$

In this comparison, we have focused on the number of inter-agent messages (vs. 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 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 alternatives that supports consistent local inference. Local inference using the other 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.

This analysis and comparison not only provide insight to the issues involved in multi-agent probabilistic reasoning, but also serve to those who implement such 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.

References

Henrion, M. 1988. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In Lemmer, J., and Kanal, L., eds., *Uncertainty in Artificial Intelligence 2*. Elsevier Science Publishers. 149–163.

Jensen, F.; Lauritzen, S.; and Olesen, K. 1990. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly* (4):269–282.

Madsen, A., and Jensen, F. 1998. Lazy propagation in junction trees. In *Proc. 14th Conf. on Uncertainty in Artificial Intelligence*.

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

Shafer, G. 1996. *Probabilistic Expert Systems*. Society for Industrial and Applied Mathematics, Philadelphia.

Suermondt, J., and Cooper, G. 1991. Initialization for the method of conditioning in Bayesian belief networks. *Artificial Intelligence* 50:83–94.

Xiang, Y., and Jensen, F. 1999. Inference in multiply sectioned Bayesian networks with extended Shafer-Shenoy and lazy propagation. In *Proc. 15th Conf. on Uncertainty in Artificial Intelligence*, 680–687.

Xiang, Y. 1996. A probabilistic framework for cooperative multi-agent distributed interpretation and optimization of communication. *Artificial Intelligence* 87(1-2):295–342.

Xiang, Y. 1998. Verification of dag structures in cooperative belief network based multi-agent systems. *Networks* 31:183–191.