Probabilistic Reasoning in a Distributed Multi-Agent Environment

S.K.M. Wong and C.J. Butz
Department of Computer Science
University of Regina
Regina, Saskatchewan, Canada, S4S 0A2
{wong,butz}@cs.uregina.ca

Abstract

In this paper, a model is proposed for multi-agent probabilistic reasoning in a distributed environment. Unlike other methods, this model is capable of processing input in a truly asynchronous fashion. Asynchronous control protocols and a method for processing evidence are developed to ensure global consistency at all times. The proposed system then extends beyond an interpretive system since the now well-defined concept of a distributed request can be introduced. Techniques are also suggested to reduce data transmission in answering this type of request.

1. Introduction

Probabilistic reasoning has become an accepted formulism for managing uncertain knowledge in a centralized environment. There is, however, an emerging interest in extending the probabilistic formulism into a distributed multi-agent environment [8]. This environment consists of multiple agents perhaps located at distinct sites. In this paper, we assume that each agent's knowledge is represented by a Markov distribution [3]. A system which accepts evidence and produces a higher level description of the environment is called *interpretive*. One such cooperative, interpretive system which applies probabilistic techniques for managing uncertainty in a distributed multi-agent environment was proposed [8]. That system, while innovative, exhibits several undesirable characteristics including: (i) each agent processes evidence and answers local probabilistic requests *serially*; (ii) each agent's knowledge base is *private*. That is, the agents may share evidence entering their respective local sites, but they do not share their knowledge bases. Consequently, each agent can only answer *local* requests. There is no concept of a distributed request; (iii) the entire system has to be brought *off-line* periodically such that *global* consistency can be restored. Although techniques were suggested to reduce this off-line time, shutting down the system even for a relatively short time could be unacceptable to many multi-agent systems.

Aside from these undesirable characteristics, the claim was made in [8] that the proposed system processes evidence "asynchronously". While the claim is true, it should be somewhat qualified. Even though the agents process evidence "asynchronously" with respect to each other, each agent processes local evidence *serially*. More importantly, each agent only updates the *local* knowledge base and does not immediately share evidence with the other agents. Thereby, knowledge inconsistency of two agents having unique belief on common variables is allowed to occur. Thus, processing evidence in this "asynchronous" fashion avoids the fundamental problem of concurrent access to *one* knowledge base, as well as leads to an inconsistent global knowledge base.

In this paper, we propose a multi-agent system which maintains global consistency at all times. As it may be necessary in many cooperative multi-agent systems for one agent to access the knowledge base of another, we remove the restriction of private knowledge bases made in [8]. That is, we allow the agents to access the knowledge base of one another. The inconsistency problems associated with asynchronous access to a knowledge base *must* now be addressed. Asynchronous control protocols and a method for processing evidence are suggested to ensure global consistency at all times, without the need of ever bringing any agent in the system off-line. By assuming public knowledge bases and maintaining global consistency, the concept of a *distributed* request is now well defined extending our model beyond merely an interpretive one. Techniques are developed to reduce the amount of data transmitted between agents in answering distributed requests.

This paper is organized as follows. Section 2 contains background knowledge. Section 3 explicitly demonstrates the inconsistency problems that may arise from asynchronous access to a common knowledge base. Asynchronous control protocols are presented in Section 4. In Section 5, we suggest techniques for distributed request processing and a method for processing evidence which maintains global consistency at all times. The conclusion is given in Section 6.

2. Background

Let X be a frame. A *joint probability distribution* [3, 5] ϕ over X is a function on X assigning to each element $c \in X$ a real number $0 \leq \phi(c) \leq 1$ such that $\sum_{c \in X} \phi(c) = 1$. A joint distribution ϕ is called a *Markov* distribution [3] if ϕ can be factorized as:

$$\phi = \frac{\prod_{h \in \mathcal{H}} \phi^{\downarrow h}}{\prod_{l \in \mathcal{L}} \phi^{\downarrow l}},$$

where $\mathcal{H}$ is a *hypertree* (an acyclic hypergraph), $\mathcal{L}$ is the corresponding intersection set, and $\phi^{\downarrow h}$ denotes the *marginal* distribution of ϕ onto the subset of variables h.

In our earlier papers [6, 7], we have demonstrated that the probabilistic model is a generalization of the traditional relational database. That is, a joint distribution ϕ can be represented as a *relation* Φ; the relational operator *product* join $\times$ can be used to multiply two relations representing distributions; the relational operator *marginalization* $\downarrow$ can be used to compute a marginal relation $\Phi^{\downarrow h}$ representing the marginal distribution $\phi^{\downarrow h}$. Most importantly, a Markov distribution ϕ can be expressed as a relation satisfying a *generalized acyclic join dependency* (GAJD) as follows:

$$\Phi = (\ldots((\Phi^{\downarrow h_1} \otimes \Phi^{\downarrow h_2}) \otimes \Phi^{\downarrow h_3}) \ldots \otimes \Phi^{\downarrow h_n}),$$

where $h_1, h_2, \ldots, h_n$ is a hypertree construction ordering [6, 7] for $\mathcal{H} = \{h_1, h_2, \ldots, h_n\}$, and $\otimes$ is the *generalized* join operator defined as

$$\Phi^{\downarrow h_1} \otimes \Phi^{\downarrow h_2} = \Phi^{\downarrow h_1} \times \Phi^{\downarrow h_2} \times (\Phi^{\downarrow h_1 \cap h_2})^{-1}.$$

Other important probabilistic notions such as local propagation and request processing were expressed as relational operations [7]. In this paper, we take full advantage of relational techniques to suggest methods for solving the inconsistency problems plaguing distributed probabilistic systems.

3. Asynchronous Access May Lead to Inconsistency

Initially, the local propagation techniques are applied to bring the multi-agent system into a globally consistent state. (Global consistency can be viewed as all agents having the same belief on all common variables, and will be formally defined in Section 5.) The system is then ready for user interaction and sensory input. Whereas in [8], each agent processes input from its *single* user and multiple sensors *serially*, our extended relational model is capable of processing input from *multiple* users and multiple sensors *asynchronously*. The situation is further complicated by assuming each agents knowledge base is *public*. The objective of this section is to explicitly demonstrate the types of problems that may arise once an agent extends from processing input *serially* to processing input *asynchronously*. In the next section, asynchronous control protocols will be developed which will maintain global consistency.

Sensory input or user interaction can be categorized into two cases, namely, collected evidence and probabilistic requests. Probabilistic requests involve only read operations on a knowledge base, while performing belief update with collected evidence may require read and write operations. In [8], processing evidence "asynchronously" means each agent processes evidence *serially* and does *not* immediately pass the evidence to the other private knowledge bases. In this paper, on the other hand, the focus is on processing both evidence and probabilistic requests in a truly *asynchronous* fashion. The following examples explicitly demonstrate the types of problems which may arise in meeting this objective.

It is important to note that if no asynchronous protocols are present, inconsistency can arise from the asynchronous execution of the input *only*. That is, this inconsistency is a result of executing the input asynchronously and is not a consequence of invalid input. Also, these examples introduce the concept of a *transaction*. We will formally define a transaction in the next section, but for now a transaction can be simply viewed as the necessary *read* and *write* operations required to execute the input.

Example Lost Evidence Consider the situation of a patient undergoing two medical examinations: one for his heart in the pulmonary ward and another for his lungs in the respiratory ward. Suppose we have a distribution $\Phi^{\downarrow h}$ on $h = \{heart, lung\}$ for this patient as shown in Figure 1. The values 0 and 1 represent a negative, positive result, respectively. Let the input of the pulmonary ward be represented by transaction T_1 and the

$\Phi^{\downarrow h}$ =

$heart$	$lung$	$f_{\phi\downarrow\{heart,lung\}}$
0	0	0.2
0	1	0.2
1	0	0.3
1	1	0.3

Figure 1. A probability distribution $\Phi^{\downarrow h}$ on $h = \{heart, lung\}$.

input of the respiratory ward by transaction T_2. Let T_1 have the evidence $heart = 0$ while T_2 has evidence $lung = 0$. If both transactions enter the system asynchronously, the following execution could occur:

$$Read_1(h); Read_2(h); Write_1(h); Write_2(h).$$

T_1 first reads $\Phi^{\downarrow h}$, then so does T_2. Transaction T_1 then enters its collected evidence and derives the distribution $(\Phi^{\downarrow h})'$ as shown in Figure 2 (left). It then writes $(\Phi^{\downarrow h})'$ back to memory overwriting $\Phi^{\downarrow h}$. At the same time, T_2 enters its collected evidence and derives the distribution $(\Phi^{\downarrow h})''$ shown in Figure 2 (right). Transaction T_2 then writes $(\Phi^{\downarrow h})''$ back to memory overwriting $(\Phi^{\downarrow h})'$. The result of this execution is as if only T_2 had entered evidence. However, the final result of all the collected evidence regarding the patient should be the distribution containing the single tuple $< 0, 0, 1.0 >$. That is, we have lost the evidence $heart = 0$ contained in T_1. □

$heart$	$lung$	$f_{(\phi\downarrow h)'}$
0	0	0.5
0	1	0.5

$heart$	$lung$	$f_{(\phi\downarrow h)''}$
0	0	0.4
1	0	0.6

Figure 2. Distributions $(\Phi^{\downarrow h})'$ (left) representing the evidence entered by T_1 only, and $(\Phi^{\downarrow h})''$ (right) representing the evidence entered by T_2 only.

Lost evidence occurs when two transactions both read the same current distribution and subsequently both update the distribution.

Example Single-agent Inconsistent Retrieval Consider a distribution $\Phi^{\downarrow h}$ on $h = \{x_1, x_2\}$ as depicted in Figure 3. Suppose one transaction T_1 is computing the marginal of $\Phi^{\downarrow h}$ onto a subset of variables $\{x_1\}$, namely $\Phi^{\downarrow\{x_1\}}$. After T_1 starts reading tuples from $\Phi^{\downarrow h}$, an asynchronous transaction T_2 updates $\Phi^{\downarrow h}$ with evidence, say $x_1 = 1$ and $x_2 = 1$, by writing $< 1, 1, 1.0 >$ and then writing the probability value for all other tuples to 0.0. That is, we have the detailed execution (we omit the last three writes of T_2):

$$R_1(< 0, 0, 0.1 >); R_1(< 0, 1, 0.2 >); R_1(< 1, 0, 0.3 >); W_2(< 1, 1, 1.0 >); R_1(< 1, 1, 1.0 >),$$

where R and W denote $Read$ and $Write$, respectively.

$\Phi^{\downarrow h} = \Phi^{\downarrow\{x_1,x_2\}}$ =

x_1	x_2	$f_{\phi\downarrow\{x_1,x_2\}}$
0	0	0.1
0	1	0.2
1	0	0.3
1	1	0.4

Figure 3. A probability distribution $\Phi^{\downarrow\{x_1,x_2\}}$.

That is, if T_1 reads the first three tuples of $\Phi^{\downarrow h}$, then T_2 updates the fourth tuple, then it is possible for T_1 to compute the incorrect marginal distribution $\Phi^{\downarrow\{x_1\}} = \{< 0, 0.3 >, < 1, 1.3 >\}$. □

Thus, an inconsistent retrieval may occur when a retrieval transaction reads some tuples in a distribution $\Phi^{\downarrow h}$ (before another transaction containing collected evidence modifies the distribution) and subsequently reads other tuples which have been modified.

Example Global Inconsistent Retrieval Consider two agents A_1 and A_2 in a multi-agent environment $A_1, A_2, \ldots, A_n$. The knowledge bases of agents A_1 and A_2 represent the distributions $\Phi^{\downarrow h_1}$ and $\Phi^{\downarrow h_2}$ as shown in Figure 4.

x_1	x_2	$f_{\phi\downarrow h_1}$
0	0	0.1
1	1	0.9

x_2	x_3	$f_{\phi\downarrow h_2}$
0	1	0.1
1	0	0.4
1	1	0.5

Figure 4. Probability distributions $\Phi^{\downarrow h_1}$ (left) and $\Phi^{\downarrow h_2}$ (right) of agents A_1 and A_2.

Suppose evidence $x_1 = 0, x_2 = 0$ is collected at A_1, and belief update is performed by agent A_1 on the local knowledge base *only*. Thus, the knowledge base of the multi-agent system is inconsistent. Suppose further that the request $p(x_2 = 1)$ is issued by a user in the multi-agent system. If the request is submitted to A_1, then the answer $p(x_2 = 1) = 0.0$ will be returned. On the other hand, if the request is submitted to A_2, then the answer $p(x_2 = 1) = 0.9$ will be returned. The user will not be confident that the answer to the request is correct. □

In the next section, asynchronous control protocols will be developed to resolve the above problems. These

protocols maintain a consistent global knowledge base and return correct answers to probabilistic requests.

4 Asynchronous Control Protocols for a Multi-Agent System

Before introducing protocols to ensure global consistency, let us first clarify the key concept of a *transaction* in our discussion. A transaction is a mathematical structure which allows us to model asynchronous knowledge base operations for a given input. Asynchronous execution of several transactions are then modelled in a mathematical structure termed a *history*. Properties which a history must satisfy in order to ensure consistency are stated. In short, the effects of an asynchronous execution of transactions must be equivalent to some serial execution of the same transactions. Finally, a practical method to implement this theory is described. While vast amounts of research on concurrency control exist in relational theory, the discussion in this paper draws from Bernstein et al. [1].

4.1. Transactions and Histories

The central notion in our discussion on asynchronous access to a shared knowledge base is the *transaction*. Each input, whether a probabilistic request or collected evidence, is represented as a transaction. The transaction is translated into the necessary read and write operations to be performed on the knowledge base. The agent must support *Read* and *Write* operations to facilitate access to the knowledge base. $Read(X)$ (or $R(X)$) returns the distribution, denoted Φ_X, consisting of all tuple(s) and the associated marginalized probability value(s). $Write(X, val)$ (or $W(X, val)$) will overwrite the distribution Φ_X with the tuple(s) and the associated probability values in the new distribution Φ_X). (We will write $W(X, val)$ as $W(X)$ if the actual distribution Φ_X is unimportant.)

An agent must execute the read and write operations atomically. From this point of view, it seems as if the read and write operations are executed serially. In the previously proposed system [8], the operations must be executed in a *serial* fashion. However, in our model we can execute knowledge base operations *asynchronously* to improve computer resource utilization and provide faster user response time. To ensure correctness, however, the result of this asynchronous execution of the read and write operations must be the *same* as one from some serial execution of the same operations.

The goal of controlling asynchronous access to a shared knowledge base is to ensure that transactions execute *atomically*. That is, the transactions must not interfere with one another. In our asynchronous system, as read and write operations may execute in parallel, we model transactions as partial orders. That is, a transaction need not specify the relative order of every pair of knowledge base operations that appear in it. We denote the Read (or Write) issued by transaction T_i on the distribution Φ_X by $R_i(X)$ (or $W_i(X)$). For simplified notation, we assume that no transaction reads or writes the same distribution Φ_X more than once. (Note that none of the results in this discussion depend on this assumption.) We now formalize the concept of a transaction.

Definition 1 A *transaction* T_i is a partial order with an ordering relation $<_i$, where

(i) $T_i \subseteq \{R_i(X), W_i(X) \mid \Phi_X \text{ is a distribution } \}$;

(ii) If $R_i(X), W_i(X) \in T_i$ then either $R_i(X) <_i W_i(X)$ or $W_i(X) <_i R_i(X)$.

To illustrate asynchronous control concepts, we can naturally depict a transaction T_i as a directed acyclic graph (DAG) with the arrows indicating the ordering defined by $<_i$. That is, the arrows indicate the order in which the knowledge base operations execute.

The concept of a transaction allows us to model asynchronous execution of the operations of a *single* transaction. We now extend this idea to modeling asynchronous transactions. In other words, we can now model asynchronous access of several transactions to a shared knowledge base. Similar to depicting the operations of a transaction, we represent a history as a DAG and do not depict arrows implied by transitivity. Consider the following three transactions:

$$\begin{aligned} T_1 &= R_1(X) \rightarrow W_1(X), \\ T_2 &= R_2(X) \rightarrow W_2(Y) \rightarrow W_2(X), \\ T_3 &= R_3(Y) \rightarrow W_3(X) \rightarrow W_3(Y) \rightarrow W_3(Z). \end{aligned}$$

A *history* H over $\{T_1, T_2, T_3\}$ may be depicted as the DAG shown in Figure 5.

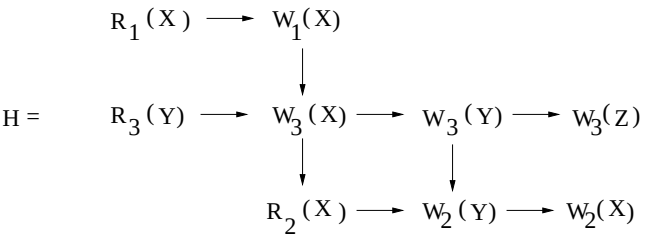

Figure 5. A history over three transactions $\{T_1, T_2, T_3\}$.

4.2. Serializability

Our goal here is to define when an asynchronous execution of transactions is correct. Executing transactions asynchronously may lead to inconsistency (see Section 3). These undesirable characteristics are entirely the result of the interleaving of the knowledge base operations and are not the result of incorrect transactions. The simplest solution to this problem is not to allow any interleaving of transactions and merely execute the transactions serially as in [8]. A better solution is to permit asynchronous execution of transactions and ensure that the result of this asynchronous execution of transactions is identical to the one obtained from some serial execution of the same transactions. An execution of transactions is *serializable* if it returns the same distribution to the user and has the same effect on the knowledge base as some serial execution of the same transactions. Since serial executions are correct [8], and serializable executions of asynchronous transactions have the same effect as some serial execution, we conclude that serializable executions are also correct. We now formalize these notions in terms of histories.

In order to define serializable histories, we first define the concept of equivalent histories as well as a serial history. We then introduce the notion of a serializable history.

Definition 2 Two histories H and H' are *equivalent*, denoted $H \equiv H'$, if

(i) H and H' are defined over the same set of transactions and have the same knowledge base operations; and

(ii) For any conflicting knowledge base operations P_i and Q_j belonging to transactions T_i and T_j, respectively, if $P_i <_H Q_j$ then $P_i <_{H'} Q_j$.

Definition 3 A history H is *serial* if for every two transactions T_i and T_j that appear in H, either all knowledge base operations of T_i appear before all knowledge base operations of T_j or vice versa.

Thus, a serial history can be seen as an execution of transactions in some order [8].

Definition 4 A history H is *serializable*, if it is equivalent to a serial history H_S.

Determining whether an asynchronous execution of transactions is correct means determining whether the history representing the asynchronous execution of transactions is serializable. The history is deemed serializable or not by examining a *serialization graph* derived from the history.

Definition 5 Let H be a history over $T = \{T_1, T_2, \ldots, T_n\}$. The *serialization graph* (SG) for H, denoted $SG(H)$, is a directed graph whose nodes are the transactions in T that are in H and whose edges are all $T_i \rightarrow T_j$ $(i \neq j)$ such that one of T_i's knowledge base operations precedes and conflicts with one of T_j's knowledge base operations in H.

Theorem 1 [1] A history H is *serializable* iff $SG(H)$ is acyclic.

4.3. A Locking Protocol for Implementing Serializability

A practical solution to multi-user access on an agent's knowledge base is to use locking. With this tool, each distribution (not necessarily marginal) has a lock associated with it. Before a transaction T_j can execute operations on the distribution, it must first control the associated lock. If another transaction T_i currently controls the lock, then T_j must wait until T_i releases the lock. The concept of locking is used to ensure serializable executions.

Each distribution Φ_X has two locks associated with it, namely a read lock operation $Rl(X)$ and a write lock operation $Wl(X)$. We denote transaction T_i obtaining a read lock (or write lock) on the distribution Φ_X by $Rl_i(X)$ (or $Wl_i(X)$). In the situation where the type of lock is not important, we will denote the lock operation with O, P or Q. Transaction T_i obtaining an arbitrary lock on distribution Φ_X is then denoted by $Ol_i(X)$. (Note that we will sometimes write $Rl_i(X)$ (or $Wl_i(X)$) to denote the operation of setting the lock. The meaning will always be clear from the context.)

An important concept to be used in locking is that of two locks conflicting. Two locks $Ol_i(X)$ and $Pl_j(Y)$ *conflict* if $i \neq j$, $X = Y$, and at least one of Ol and Pl is a write lock. Thereby, two locks on distinct distributions do not conflict, nor do two Read locks by distinct transactions on one distribution, nor a read and a write lock on the same distribution held by the same transaction.

After a transaction T has performed its Read (or Write) operation on the distribution Φ_X, it then *releases* the lock, denoted by the *unlock* operation $Ru_i(X)$ (or $Wu_i(X)$).

It is the responsibility of the agent to obtain and release the locks on behalf of the transactions. We are now ready to describe a locking protocol called *two phase locking* which the agent can abide by to ensure consistency.

Two Phase Locking

(i) When a transaction T_i intends to perform an operation $P_i(X)$ on a distribution Φ_X, the agent tests if $Pl_i(X)$ conflicts with another lock $Ql_j(X)$ that

is already held. If $Pl_i(X)$ conflicts with any other lock, then the agent delays $P_i(X)$ until the conflicting lock is released. On the other hand, if $Pl_i(X)$ does not conflict with any other lock, then the agent sets $Pl_i(X)$ on behalf of T_i and executes the operation $P_i(X)$.

(ii) Once a transaction T_i has released a lock, it may not subsequently obtain any more locks on any distribution.

Part (i) of the two phase locking protocol ensures that conflicting operations on a distribution are executed in the same order as the locks are obtained. Part (ii) is required to ensure that all pairs of conflicting operations of two transactions are executed in the same order. Otherwise, transaction T_j may appear to precede transaction T_i with respect to distribution Φ_X, yet follow transaction T_i with respect to another distribution Φ_Y. Such an execution is not serializable.

Theorem 2 [1] Every two phase locking history H is serializable.

5 Processing Distributed Requests and Evidence

In this section, we discuss some performance issues on answering probabilistic requests by stressing their importance in a distributed environment. In distributed systems, the necessary data transmission between agents should be minimized. We assume here that transmission speeds may be orders of magnitude lower than the transfer rates between disks and main memory (i.e., local I/O operations). Therefore, it is important to reduce the data transmission costs in distributed request processing. We conclude by introducing a method for processing evidence which maintains global consistency at all times.

There are two classes of probabilistic requests, namely, unconditional and conditional. Conditional requests involve selection of certain configurations whereas unconditional requests do not. It is straightforward to adopt the centralized processing techniques for unconditional requests $p(x_a, \ldots, x_d)$ [7]. Hence, we focus the discussion on the more interesting techniques for conditional requests.

5.1. Conditional Request Processing

Conditional requests involve selection of certain tuples in computing the answer. In this section we will present a request processing technique to reduce the amount of data transmitted between agents in answering conditional probabilistic requests.

Consider what is involved in computing $\Phi_h \otimes \Phi_k$ to answer a conditional probabilistic request where Φ_h and Φ_k represent the knowledge bases of two different agents. One agent could transmit the *entire* distribution to the other, which could then compute the answer. However, since the generalized join operator $\otimes$ is defined in terms of the product join operator $\times$ (which is partly defined [7] in terms of the natural join operator $\bowtie$), we can take advantage of the query processing techniques developed for standard relational databases to answer conditional probabilistic requests.

In the following definition, $\Phi_X[Y]$ denotes the projection [4] of the distribution Φ_X onto the set of attributes $Y \subseteq (X \cup \{f_{\phi_X}\})$.

Definition 6 Let Φ_h and Φ_k be two distributions on h and k respectively. The *semijoin* of Φ_h with Φ_k, denoted $\Phi_h \rhd \Phi_k$, is the distribution $(\Phi_h \bowtie \Phi_k)[h \cup \{f_{\phi_h}\}]$. That is, $\Phi_h \rhd \Phi_k$ are the tuples in the distribution Φ_h that are joinable with tuples in the distribution Φ_k.

Note that the entire distribution Φ_k is not required in computing $\Phi_h \rhd \Phi_k$ because:

$$\Phi_h \rhd \Phi_k = (\Phi_h \bowtie \Phi_k)[h \cup \{f_{\phi_h}\}] = \Phi_h \bowtie \Phi_k[h \cap k].$$

Now the useful property of semijoin is that $\Phi_h \bowtie \Phi_k = (\Phi_h \rhd \Phi_k) \bowtie \Phi_k$. Thereby, computing $\Phi_h \bowtie \Phi_k$ as $(\Phi_h \rhd \Phi_k) \bowtie \Phi_k$ in step (i) of product join [7] will reduce transmission costs between agents whenever Φ_h has a greater cardinality than that of $\Phi_k[h \cap k]$ and $\Phi_h \rhd \Phi_k$ together.

Example Consider a Markov distribution Φ defined on a hypertree $\mathcal{H} = \{h_1 = \{x_1, x_2, x_3\}, h_2 = \{x_3, x_4, x_5\}\}$, namely, $\Phi = \Phi^{\downarrow h_1} \otimes \Phi^{\downarrow h_2}$, where $\Phi^{\downarrow h_1}$ and $\Phi^{\downarrow h_2}$ are the marginal distributions represented by the knowledge bases of agents A_1 and A_2 as shown in Figure 6. Suppose that the probabilistic request $p(x_1, x_3, x_5 \mid x_2 = 0, x_4 = 0)$ is issued by a user at A_2. Agent A_1 could transmit the entire distribution $\Phi^{\downarrow h_1}$ to A_2 which would then compute the answer to the probabilistic request. However, the request can be answered more efficiently using the semijoin operation. First each agent performs local processing to reduce the tuples involved in answering the request. The resultant distributions, called *views* [4] in the relational model, are shown in Figure 7.

Agent A_2 next transmits $(\Phi^{\downarrow h_2})'[h_1 \cap h_2] = (\Phi^{\downarrow h_2})'[x_3]$ to A_1. Agent A_1 computes the distribution $(\Phi^{\downarrow h_1})''$ which is the semijoin of $(\Phi^{\downarrow h_1})'$ and $(\Phi^{\downarrow h_2})'$, namely, $((\Phi^{\downarrow h_1})' \rhd (\Phi^{\downarrow h_2})')$ shown in Figure 8, and transmits it to A_2. Agent A_2 computes $(\Phi^{\downarrow h_2})''$ which is the semijoin of $(\Phi^{\downarrow h_2})'$ and $(\Phi^{\downarrow h_1})''$, namely, $(\Phi^{\downarrow h_2})' \rhd (\Phi^{\downarrow h_1})''$ as shown in Figure 9. Finally, agent A_2 answers the conditional probabilistic request with

x_1	x_2	x_3	$f_{\phi^{\downarrow h_1}}$
0	0	0	0.1
1	0	1	0.2
2	1	2	0.3
3	2	3	0.4

x_3	x_4	x_5	$f_{\phi^{\downarrow h_2}}$
0	0	1	0.1
1	1	2	0.2
2	0	3	0.3
3	1	4	0.4

Figure 6. Distributions $\Phi^{\downarrow h_1}$ (left) and $\Phi^{\downarrow h_2}$ (right) of agents A_1 and A_2.

x_1	x_2	x_3	$f_{(\phi^{\downarrow h_1})'}$
0	0	0	0.1
1	0	1	0.2

x_3	x_4	x_5	$f_{(\phi^{\downarrow h_2})'}$
0	0	1	0.1
2	0	3	0.3

Figure 7. Applying local processing on distributions $\Phi^{\downarrow h_1}$ (left) and $\Phi^{\downarrow h_2}$ (right) in Figure 6.

$(\Phi^{\downarrow h_1})'' \otimes (\Phi^{\downarrow h_2})''$. $\square$

$(\Phi^{\downarrow h_1})'' =$

x_1	x_2	x_3	$f_{(\phi^{\downarrow h_1})''}$
0	0	0	0.1

Figure 8. The semijoin of $(\Phi^{\downarrow h_1})'$ and $(\Phi^{\downarrow h_2})'$, namely, $(\Phi^{\downarrow h_1})'' = (\Phi^{\downarrow h_1})' \rhd (\Phi^{\downarrow h_2})'$.

The important point is that no marginal distribution was transmitted in its entirety even though the multi-agent system is globally consistent.

5.2. Maintain Global Consistency While Processing Evidence

In [8], the method for processing evidence may lead to global inconsistency. In this section our goal is to develop a method for processing evidence which maintains global consistency at all times. The proposed method for maintaining global consistency takes advantage of the following theorem which relates *pairwise* consistency and *global* consistency in a hypertree.

Theorem 3 [2] Let X be a set of variables. Let $\mathcal{H} = \{h_1, h_2, \ldots, h_n\}$ be a hypertree such that $\bigcup_{i=1}^{n} h_i = X$. Let Φ_{h_i} be a probability distribution over h_i such that the distributions are pairwise consistent (i.e., $\Phi_{h_i}^{\downarrow h_i \cap h_j} = \Phi_{h_j}^{\downarrow h_i \cap h_j}$, for each pair h_i, h_j with $h_i \cap h_j \neq \emptyset$, $i \neq j$). Let $\mathcal{L} = \{l_2, l_3, \ldots, l_n\}$ be the intersection set of $\mathcal{H}$ and $\Phi_{h_i}^{\downarrow l_i}$ be the distribution over l_i computed from Φ_{h_i} for $i = 2, 3, \ldots, n$. Then there exists a unique joint probability distribution Φ such that

$(\Phi^{\downarrow h_2})'' =$

x_3	x_4	x_5	$f_{(\phi^{\downarrow h_2})''}$
0	0	1	0.1

Figure 9. The semijoin of $(\Phi^{\downarrow h_2})'$ and $(\Phi^{\downarrow h_1})''$, namely, $(\Phi^{\downarrow h_2})'' = (\Phi^{\downarrow h_2})' \rhd (\Phi^{\downarrow h_1})''$.

$\Phi^{\downarrow h_i} = \Phi_{h_i}$ for each h_i.

Theorem 3 states that maintaining pairwise consistency on a hypertree ensures global consistency. In other words, the distribution represented by the knowledge base of each agent is in fact a marginal distribution derived from the same joint probability distribution. Thereby, since our multi-agent system is defined on a hypertree, global consistency can be achieved by developing a method for processing evidence which maintains pairwise consistency.

The method for processing evidence which maintains pairwise consistency is now described. Let $\Phi^{\downarrow h_i}$ be the marginal distribution represented by the knowledge base of agent A_i. Let $\mathbf{C}$ be the set of configurations to be deleted from $\Phi^{\downarrow h_i}$ as a result of the collected evidence. Let $Y \subseteq h_i$ be the set of variables which are also represented in the knowledge base of at least one other agent. For each $\mathbf{c} \notin \mathbf{C}$:

$$\phi'_{h_i}(\mathbf{c}) = \phi_{h_i}(\mathbf{c}) + \Big(\sum_{\mathbf{c}' \in \mathbf{C}, (\mathbf{c}')^{\downarrow Y} = (\mathbf{c})^{\downarrow Y}} \phi_{h_i}(\mathbf{c}')\Big)\Big(\frac{\phi_{h_i}(\mathbf{c})}{\sum_{\mathbf{c}' \notin \mathbf{C}, (\mathbf{c}')^{\downarrow Y} = (\mathbf{c})^{\downarrow Y}} \phi_{h_i}(\mathbf{c}')}\Big),$$

$$\text{if} \sum_{\mathbf{c}' \notin \mathbf{C}, (\mathbf{c}')^{\downarrow Y} = (\mathbf{c})^{\downarrow Y}} \phi_{h_i}(\mathbf{c}') > 0,$$

where $(\mathbf{c})^{\downarrow Y}$ denotes the Y-tuple [4] obtained by deleting the values of the variables in h_i and not in Y. Delete the set of configurations $\mathbf{C}$ from $\Phi^{\downarrow h_i}$.

Let us now demonstrate that performing belief update in this fashion maintains a pairwise consistent knowledge base. By the above construction, it can be shown that:

$$\frac{\phi'_{h_i}(\mathbf{c})}{\sum_{\mathbf{c}' \notin \mathbf{C}} \phi'_{h_i}(\mathbf{c}')} = \frac{\phi_{h_i}(\mathbf{c})}{\sum_{\mathbf{c}' \notin \mathbf{C}} \phi_{h_i}(\mathbf{c}')}.$$

Hence, pairwise consistency is maintained after the evidence has been processed by agent A_i. That is, for any knowledge base h_j with $h_i \cap h_j \neq \emptyset$, the equality $((\Phi^{\downarrow h_i})')^{\downarrow h_i \cap h_j} = (\Phi^{\downarrow h_j})^{\downarrow h_i \cap h_j}$ still holds. By Theorem 3 then, the knowledge base of the entire multi-agent system remains *globally* consistent.

Example Consider a multi-agent system consisting of three agents A_1, A_2, and A_3 with respective knowledge

bases depicted in Figure 10. Suppose A_2 collects evidence and consequently must delete the configurations $\mathbf{C} = \{< 0,1,0,1 >, < 0,1,1,1 >\}$. The set of variables that are also represented in the knowledge bases of at least one other agent is $Y = \{x_2, x_5\}$. The resulting marginal distribution $(\Phi^{\downarrow h_2})'$ obtained by the proposed method for processing evidence is shown in Figure 11.

x_1	x_2	$f_{\phi^{\downarrow h_1}}$
0	0	0.1
0	1	0.2
1	0	0.3
1	1	0.4

x_2	x_3	x_4	x_5	$f_{\phi^{\downarrow h_2}}$
0	0	0	0	0.05
0	0	0	1	0.025
0	0	1	0	0.05
0	0	1	1	0.075
0	1	0	0	0.05
0	1	0	1	0.05
0	1	1	0	0.05
0	1	1	1	0.05
1	0	0	0	0.1
1	0	0	1	0.1
1	0	1	0	0.1
1	0	1	1	0.1
1	1	0	0	0.05
1	1	0	1	0.05
1	1	1	0	0.05
1	1	1	1	0.05

x_5	x_6	$f_{\phi^{\downarrow h_3}}$
0	0	0.2
0	1	0.3
1	0	0.3
1	1	0.2

Figure 10. Marginal distributions representing the knowledge bases of agents A_1 (top left), A_2 (right) and A_3 (bottom left).

It can easily verified that the proposed method for processing evidence maintains pairwise consistency. As our multi-agent system is defined on a hypertree, by Theorem 3 the knowledge base is also globally consistency. Assuming public knowledge bases and maintaining global consistency at all times allows the introduction of *distributed* requests into the distributed probabilistic reasoning framework, as well as ensuring that all agents have consistent belief on all common variables.

6 Conclusion

In this paper, a model was proposed for multi-agent probabilistic reasoning in a distributed environment. Unlike other methods [8], this model is capable of processing input in a *truly* asynchronous fashion. Asynchronous control protocols and a method for processing evidence are developed to ensure global consistency at *all* times. The proposed system then extends beyond an interpretive system since the now well-defined concept of a *distributed* request can be introduced. Techniques are also suggested to reduce data transmission in answering this type of request.

x_2	x_3	x_4	x_5	$f_{(\phi^{\downarrow h_2})'}$
0	0	0	0	0.05
0	0	0	1	0.050
0	0	1	0	0.05
0	0	1	1	0.150
0	1	0	0	0.05
0	1	1	0	0.05
1	0	0	0	0.1
1	0	0	1	0.1
1	0	1	0	0.1
1	0	1	1	0.1
1	1	0	0	0.05
1	1	0	1	0.05
1	1	1	0	0.05
1	1	1	1	0.05

Figure 11. The marginal distribution $(\Phi^{\downarrow h_2})'$ representing the knowledge base of agent A_2 after the evidence has been processed.

References

[1] P. A. Berstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Don Mills, Ontario, 1987.

[2] A. Dawid and S. Lauritzen. Hyper markov laws in the statistical analysis of decomposable graphical models. *Ann. Stat.*, 21:1272–1317, 1993.

[3] P. Hajek, T. Havranek, and R. Jirousek. *Uncertain Information Processing in Expert Systems.* CRC Press, 1992.

[4] D. Maier. *The Theory of Relational Databases.* Principles of Computer Science. Computer Science Press, Rockville, Maryland, 1983.

[5] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers, San Francisco, California, 1988.

[6] S. Wong. An extended relational data model for probabilistic reasoning. *Journal of Intelligent Information Systems*, 9:181–202, 1997.

[7] S. Wong, C. Butz, and Y. Xiang. A method for implementing a probabilistic model as a relational database. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 556–564. Morgan Kaufmann Publishers, 1995.

[8] Y. Xiang. A probabilistic framework for cooperative multi-agent distributed interpretation and optimization of communication. *Artificial Intelligence*, 87:295–342, 1996.