Recovery Protocols in Multi-Agent Probabilistic Reasoning Systems

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

Abstract

In this paper, we introduce a probabilistic relational data model as the basis for developing multi-agent probabilistic reasoning systems. Since our model subsumes the traditional relational data model, it immediately follows that we can take full advantage of the existing distributed and concurrency control techniques to address the undesirable characteristics exhibited by current multi-agent probabilistic reasoning systems. Thereby, our probabilistic relational data model has important theoretical and practical ramifications. One unified model allows the cross-fertilization of techniques, and serves as a basis for implementing one system for both of these similar domains.

1. Introduction

The relational data model [5] was proposed for *data management* in transaction processing systems. Data dependencies, such as functional dependencies and multivalued dependencies (MVD), were extensively studied as they played a key role in the normalization process. A main result [1] of this research was that the class of acyclic database schemes, i.e., the universal relation satisfies an acyclic join dependency (AJD), possess a number of desirable properties in database applications.

On the other hand, probabilistic networks [4, 6] have proven to be a useful tool for uncertainty management in many AI applications. It has been shown [8, 11] how the traditional relational data model can be generalized for developing probabilistic reasoning systems. This *probabilistic relational data model* consists of a probabilistic relation as its data structure and the addition of the two new relational operators *product join* $\times$ and *marginalization* $\downarrow$. The notion of probabilistic conditional independence can then be equivalently expressed as a probabilistic relation satisfying a *probabilistic multivalued dependency* (PMVD). Furthermore, the representation of probabilistic knowledge in practice can be expressed as a probabilistic relation satisfying a *probabilistic acyclic join dependency* (PAJD).

There is increasing interest in developing *multi-agent* probabilistic models for uncertainty management in distributed AI applications. The first model proposed [12], while innovative, exhibits several undesirable characteristics including: (i) each agent processes transactions *serially*; (ii) the multi-agent network is allowed to be *inconsistent*; (iii) the entire system has to be brought *off-line* to restore consistency; (iv) there is no concept of a *distributed* query; (v) no provisions are made for transaction or agent *failure*.

Once it is acknowledged that a probabilistic network is a generalized relational database, however, it immediately follows that relational database techniques can be modified for probabilistic reasoning purposes. In [10], the well known database concurrency control technique *two-phase locking* was extended to ensure *consistency* at all times. Maintaining global consistency permits the introduction of *distributed* queries into the multi-agent environment [10]. In this case, the query optimization techniques used in distributed databases can be applied to reduce the amount of data transmitted between the agents in answering this type of request. In this paper, we incorporate recovery protocols into our model [10] in order to ensure consistency in recovering from transaction and agent *failure*. While it is acknowledged that the actual extension to the relational database recovery technique is not significant in itself, the main contribution of this paper is to demonstrate multi-agent uncertainty management as an area of distributed relational database research and implementation.

A rigorous generalization of the traditional relational model has important theoretical and practical ramifications. In practice, one intelligent system can be designed for both of these problems by including the appropriate relational operators. On the theoretical side, a unified model provides the opportunity for the cross-fertilization of techniques between Bayesian networks and relational databases.

This paper is organized as follows. Section 2 contains background knowledge including our *probabilistic relational data model* and multi-agent probabilistic reasoning. Section 3 motivates the need for recovery protocols by explicitly demonstrating the inconsistency problems that may arise from agent failure. Recovery protocols to ensure consistency in a multi-agent environment are presented in Section 4. The conclusion is given in Section 5.

2. Basic notions

We first define the basic notions of hypergraphs, probabilistic networks, a probabilistic relational database model, and multi-agent probabilistic reasoning.

2.1. Hypergraphs

Let $N = \{A_1, A_2, \ldots, A_m\}$ be a finite set of variables. A *hypergraph*, denoted $\mathcal{H}$, is a family of subsets of variables in N, i.e., $\mathcal{H} \subseteq 2^N$. An element in $\mathcal{H}$ is called a *hyperedge*. We call an element $t \in \mathcal{H}$ a *twig*, if there exists another distinct element $b \in \mathcal{H}$ such that

$$t \cap (\cup \{h \mid h \in \mathcal{H} \text{ and } h \neq t\}) = t \cap b.$$

(By this definition, the hyperedge in a hypergraph consisting of a single hyperedge is not a twig). This means that the intersection of t and the hypergraph $\cup(\mathcal{H} - \{t\})$ is contained in the hyperedge b. We call any such b a *branch* for the twig t, and note that a twig t may have many possible branches. A hypergraph $\mathcal{H} = \{h_1, h_2, \ldots, h_n\}$ is called an *acyclic* hypergraph (a hypertree) [1, 7] if its elements $h_1, h_2, \ldots, h_i$ can be ordered such that h_i is a twig in the sub-hypergraph $\{h_1, h_2, \ldots, h_i\}$, $i = 1, 2, \ldots, n$. We call any ordering satisfying this condition a *hypertree construction ordering* for $\mathcal{H}$. (A hypertree construction ordering can also be represented as a *join tree* [1].) Given a particular hypertree construction ordering, we can choose an integer $b(i)$, for $i = 2, \ldots, n$, such that $1 \leq b(i) \leq i - 1$ and $h_{b(i)}$ is a branch for h_i in $\{h_1, h_2, \ldots, h_i\}$. We call $b(i)$ a *branching function* for this ordering. It is possible that a hypertree construction ordering may have more than one branching function.

For example, consider the case where $N = \{A_1, A_2, \ldots, A_6\}$. Let $\mathcal{H} = \{h_1 = \{A_1, A_2, A_3\}, h_2 = \{A_2, A_3, A_4\}, h_3 = \{A_2, A_3, A_5\}, h_4 = \{A_5, A_6\}\}$ denote the hypergraph shown in Figure 1. Since we can define a hypertree construction ordering h_1, h_2, h_3, h_4, by definition this hypergraph is a hypertree. One possible branching function for this ordering h_1, h_2, h_3, h_4 is $b(2) = 1, b(3) = 1, b(4) = 3$.

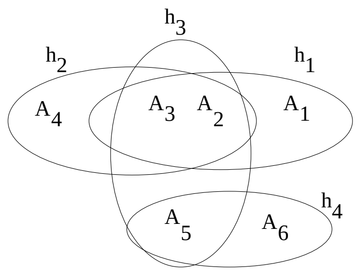

Figure 1. A graphical representation of the hypergraph $\mathcal{H} = \{h_1, h_2, h_3, h_4\}$.

2.2. Probabilistic Networks

Probabilistic networks [4, 6] have proven to be a useful tool for uncertainty management in many AI applications. A probabilistic network consists of a dependency (graphical) structure coupled with a corresponding set of probability tables. A *Bayesian* network [6] consists of a directed acyclic graph (DAG) and corresponding conditional probability tables. In practice, it is useful to transform a Bayesian network into a Markov network to facilitate probabilistic inference. A *Markov* network [4] consists of an acyclic hypergraph with a corresponding set of marginal distributions.

Formally, a *relation scheme* N is a finite set of *attributes* (attribute names) $N = \{A_1, A_2, \ldots, A_m\}$. Corresponding to each attribute A_i is a nonempty finite set D_i, $1 \leq i \leq m$, called the *domain* of A_i. Let $\mathbf{D} = D_1 \cup D_2 \ldots \cup D_m$. A *relation* r on the relation scheme N, written $r(N)$, is a finite set of mappings $\{t_1, t_2, \ldots, t_s\}$ from N to $\mathbf{D}$ with the restriction that for each mapping $t \in r$, $t(A_i)$ must be in D_i, $1 \leq i \leq k$, where $t(A_i)$ denotes the value obtained by restricting the mapping to A_i. The mappings are called *tuples* and $t(A)$ is called the A-value of t. We use $t(X)$ in the obvious way and call it the X-value of t. To simplify the notation, however, we will henceforth denote relations by writing the attributes in a certain order and the tuples as lists of values in the same order.

Let $r(N)$ be a fixed relation representing the domain of a finite set of attributes $N = A_1 A_2 \cdots A_m$. A *joint probability distribution* [4, 6] over $r(N)$ is a function p on $r(N)$ assigning to each tuple $t \in r(N)$ a real number

$0 \leq p(t) \leq 1$ such that $\sum_{t \in r(N)} p(t) = 1$. (We say the distribution is over N when the domain $r(N)$ is understood, and sometimes write p as $p(A_1, A_2, \ldots, A_m)$ or p_N.) By the chain rule, a joint probability distribution p over $N = A_1 A_2 \cdots A_m$ can always be written as:

$$\begin{aligned} p(N) &= p(A_1) \cdot p(A_2|A_1) \cdot p(A_3|A_1A_2) \cdot \ldots \cdot \\ &\quad p(A_m|A_1A_2 \ldots A_{m-1}). \end{aligned}$$

The above equation is an identity. However, one can use known conditional independencies to obtain a simpler representation of a jpd.

Let X, Y and Z be disjoint subsets of N. We say that Y and Z are *conditionally independent* given X if

$$p(Y|XZ) = p(Y|X), \qquad (1)$$

or equivalently

$$p(YXZ) = \frac{p(YX) \cdot p(XZ)}{p(X)}. \qquad (2)$$

For example, consider a jpd $p(N)$ on $N = A_1A_2A_3A_4A_5A_6$ and the following known conditional independencies:

$$\begin{aligned} p(A_3|A_1A_2) &= p(A_3|A_1), \\ p(A_4|A_1A_2A_3) &= p(A_4|A_2, A_3\}), \\ p(A_5|A_1A_2A_3A_4) &= p(A_5|\{A_2, A_3\}), \\ p(A_6|A_1A_2A_3A_4A_5) &= p(A_6|\{A_5\}). \end{aligned}$$

Utilizing these conditional independencies, the jpd p written using the chain rule can be expressed in a simpler form, namely:

$$\begin{aligned} p(N) &= p(A_1) \cdot p(A_2|A_1) \cdot p(A_3|A_1) \cdot p(A_4|A_2A_3) \\ &\quad \cdot p(A_5|A_2A_3) \cdot p(A_6|A_5). \qquad (3) \end{aligned}$$

We can represent the dependency structure of this jpd by a DAG as shown in Figure 2. A Bayesian network is defined by this DAG together with the conditional probability tables $p(A_1)$, $p(A_2|A_1)$, $p(A_3|A_1)$, $p(A_4|A_2A_3)$, $p(A_5|A_2A_3)$, and $p(A_6|A_5)$.

In practice, however, it is useful to transform a Bayesian network into a Markov network [4] to facilitate probabilistic inference. The DAG representing the dependency structure of a Bayesian network can be converted by the moralization and triangulation procedures [4, 6] into an acyclic hypergraph. As well, the conditional probability tables defined over the DAG are used to define marginal distributions over the hyperedges of the acyclic hypergraph.

For example, the DAG in Figure 2 may be transformed into the acyclic hypergraph $\mathcal{H} = \{h_1 = \{A_1, A_2, A_3\},\ h_2 = \{A_2, A_3, A_4\},\ h_3 = \{A_2, A_3, A_5\},\ h_4 = \{A_5, A_6\}\}$ depicted in Figure 1. Such an acyclic hypergraph represents the dependency structure of a Markov network. A Markov network can then be defined by specifying a marginal distribution $p(h)$ over each hyperedge h in $\mathcal{H}$. The jpd in equation (3) is then rewritten in terms of marginal distributions over the acyclic hypergraph $\mathcal{H}$ in Figure 1 as follows:

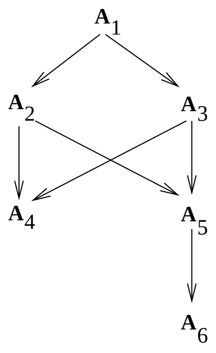

Figure 2. The DAG representing the conditional independencies in equation (3).

$$p = \frac{p(A_1A_2A_3) \cdot p(A_2A_3A_4) \cdot p(A_2A_3A_5) \cdot p(A_5A_6)}{p(A_2A_3) \cdot p(A_2A_3) \cdot p(A_5)}. \qquad (4)$$

Once the probabilistic knowledge is represented as a Markov network in terms of marginal distributions, the probabilistic reasoning system is ready for user interaction.

Note that all *embedded* conditional independencies are sacrificed in transforming a Bayesian network into a Markov network. For example, the embedded conditional independency of A_2 and A_3 given A_1 is reflected by the Bayesian network defined by equation (3), but not by the Markov network defined by equation (4).

2.3. A probabilistic relational data model

In this section, we extend the traditional relational data model [5] for transaction processing systems into a more general model for intelligent systems.

A jpd can be represented as a relation. The relation $\mathbf{r}$ representing the jpd p has attributes $N \cup \{A_p\}$, where the column labelled by A_p stores the probability value. The relation $\mathbf{r}$ representing a jpd p on the set of variables $N = A_1A_2 \cdots A_m$ is shown in Figure 3. Each tuple $\mathbf{t} \in \mathbf{r}$ is defined by $\mathbf{t}(N) = t \in \mathbf{D}$ and $\mathbf{t}(A_p) = p(t)$. That is, $\mathbf{t} = < t, p(t) >$. For convenience we will say relation $\mathbf{r}$ is on N with the attribute A_p understood by context. That is, relations denoted by boldface represent probability distributions. A relation $\mathbf{r}$ on scheme $N = A_1A_2A_3$ is depicted in Figure 4.

A_1	A_2	$\ldots$	A_m	A_p
$\mathbf{t}_1(A_1)$	$\mathbf{t}_1(A_2)$	$\ldots$	$\mathbf{t}_1(A_m)$	$\mathbf{t}_1(A_p) = p(t_1)$
$\mathbf{t}_2(A_1)$	$\mathbf{t}_2(A_2)$	$\ldots$	$\mathbf{t}_2(A_m)$	$\mathbf{t}_2(A_p) = p(t_2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathbf{t}_s(A_1)$	$\mathbf{t}_s(A_2)$	$\ldots$	$\mathbf{t}_s(A_m)$	$\mathbf{t}_s(A_p) = p(t_s)$

Figure 3. A joint distribution p expressed as a relation $\mathbf{r}$.

$\mathbf{r} =$

A_1	A_2	A_3	A_p
0	0	0	0.3
0	0	1	0.3
0	1	1	0.2
1	0	0	0.1
1	0	1	0.1

Figure 4. A relation $\mathbf{r}$ on $N = A_1A_2A_3$.

Let $\mathbf{r}(N)$ be a relation and X be a subset of N. The *marginalization of* $\mathbf{r}$ *onto* X, written $\tau_X(\mathbf{r})$, is defined as

$$\tau_X(\mathbf{r}) \;=\; \{\ \mathbf{t}(XA_{p(X)}) \mid \mathbf{t}(X) \in \pi_X(\mathbf{r}) \text{ and } \mathbf{t}(A_{p(X)}) = \sum_{\mathbf{t}' \in \mathbf{r}} \mathbf{t}'(A_p)\ \}, \qquad (5)$$

where $\mathbf{t}'(X) = \mathbf{t}(X)$, and π is the *project* operator [5]. That is, $\tau_X(\mathbf{r})$ represents a probability distribution on the traditional projection $\pi_X(\mathbf{r}) = \{\ \mathbf{t}(X) \mid \mathbf{t} \in \mathbf{r}\ \}$ of $\mathbf{r}$ onto X. In the literature [4, 6], the relation $\tau_X(\mathbf{r})$ is called the *marginal distribution* $p(X)$ of $p(N)$ onto X. For example, the relations $\tau_{A_1A_2}(\mathbf{r})$ and $\tau_{A_2A_3}(\mathbf{r})$ in Figure 5 depict the marginalization of relation $\mathbf{r}$ in Figure 4 onto A_1A_2 and A_2A_3, respectively.

A_1	A_2	$A_{p(A_1,A_2)}$
0	0	0.6
0	1	0.2
1	0	0.2

A_2	A_3	$A_{p(A_2,A_3)}$
0	0	0.4
0	1	0.4
1	1	0.2

Figure 5. The relations $\tau_{A_1A_2}(\mathbf{r})$ and $\tau_{A_2A_3}(\mathbf{r})$, where $\mathbf{r}$ is shown in Figure 4.

The *product join* of two relations $\mathbf{r}_1(X)$ and $\mathbf{r}_2(Y)$, written $\mathbf{r}_1(X) \times \mathbf{r}_2(Y)$, is defined as

$$\begin{aligned} & \mathbf{r}_1(X) \times \mathbf{r}_2(Y) \\ = \ & \{\mathbf{t}(XYA_{p_X \cdot p_Y}) \mid \mathbf{t}(XY) \in \pi_X(\mathbf{r}_1) \bowtie \pi_Y(\mathbf{r}_2) \\ & \text{and } \mathbf{t}(A_{p_X \cdot p_Y}) = \mathbf{t}(A_{p_X}) \cdot \mathbf{t}(A_{p_Y})\}, \qquad (6) \end{aligned}$$

where $\bowtie$ is the *natural join* operator [5]. Thus, $\mathbf{r}_1(X) \times \mathbf{r}_2(Y)$ denotes the product $p(X) \cdot p(Y)$ of the two distributions $p(X)$ and $p(Y)$.

Generalized relational data dependencies can now be introduced. The fundamental notion of probabilistic multivalued dependency is introduced first.

Let N be a relation scheme, X and Y be disjoint subsets of N, and $Z = N - XY$. A relation $\mathbf{r}(N)$ satisfies the *probabilistic multivalued dependency* (PMVD) $X \Rightarrow\Rightarrow Y$ if

$$\mathbf{r}(N) \;=\; \tau_{XY}(\mathbf{r}) \times \tau_{XZ}(\mathbf{r}) \times \tau_X(\mathbf{r})^{-1}, \qquad (7)$$

where the relation $\tau_X(\mathbf{r})^{-1}$ is defined from $\tau_X(\mathbf{r})$ by renaming attribute $A_{p(X)}$ as $A_{1/p(X)}$, and setting $\mathbf{t}(A_{1/p(X)}) = 1/\mathbf{t}(A_{p(X)})$, if $\mathbf{t}(A_{p(X)}) > 0$, for every $\mathbf{t} \in \tau_X(\mathbf{r})^{-1}$. Saying that Y and Z are conditionally independent given X in equation (2) is the same as saying that relation $\mathbf{r}$ satisfies the PMVD $X \Rightarrow\Rightarrow Y$.

We introduce shorthand notation for the right hand side of equation (7) as follows:

$$\tau_{XY}(\mathbf{r}) \otimes \tau_{XZ}(\mathbf{r}) \equiv \tau_{XY}(\mathbf{r}) \times \tau_{XZ}(\mathbf{r}) \times \tau_X(\mathbf{r})^{-1}.$$

We call the binary operator $\otimes$ *Markov join*. Hence, a relation $\mathbf{r}(N)$ satisfies the PMVD $X \Rightarrow\Rightarrow Y$ if and only if

$$\mathbf{r}(N) = \tau_{XY}(\mathbf{r}) \otimes \tau_{XZ}(\mathbf{r}). \qquad (8)$$

Example 2.1 Relation $\mathbf{r}(A_1A_2A_3)$ in Figure 4 satisfies the PMVD $A_2 \Rightarrow\Rightarrow A_1$ since

$$\mathbf{r}(A_1A_2A_3) = \tau_{A_1A_2}(\mathbf{r}) \otimes \tau_{A_2A_3}(\mathbf{r}),$$

where $\tau_{A_1A_2}(\mathbf{r})$ and $\tau_{A_2A_3}(\mathbf{r})$ are shown in Figure 5.

PMVD is a special case of a more general kind of data dependency, called probabilistic acyclic join dependency. Let $\mathcal{N} = \{N_1, N_2, \ldots, N_n\}$ and $N = N_1 \cup N_2 \cup \ldots \cup N_n$. A relation $\mathbf{r}(N)$ satisfies the *probabilistic acyclic join dependency* (PAJD) $\otimes\mathcal{N}$, if

$$\mathbf{r}(N) = ((\tau_{N_1}(\mathbf{r}) \otimes \tau_{N_2}(\mathbf{r})) \otimes \ldots) \otimes \tau_{N_n}(\mathbf{r}), \qquad (9)$$

where the sequence $N_1, N_2, \ldots, N_n$ is a hypertree construction ordering for $\mathcal{N}$. Thus, a Markov network can be expressed as a PAJD. The Markov network defined by equation (4), for example, can be written as the PAJD

$$\mathbf{r}(N) = ((\tau_{h_1}(\mathbf{r}) \otimes \tau_{h_2}(\mathbf{r})) \otimes \tau_{h_3}(\mathbf{r})) \otimes \tau_{h_4}(\mathbf{r}). \qquad (10)$$

The important point to realize is that the probabilistic knowledge is represented as a generalized relational database. It is straightforward to simulate probabilistic inference using simple SQL queries [11]. Thereby, uncertainty management in AI can be processed as typical relational applications by replacing natural join $\bowtie$ and projection π with product join $\times$ and marginalization τ , respectively.

2.4. Multi-Agent probabilistic reasoning

In this section, we briefly review related research on multi-agent probabilistic reasoning. The first task is to construct the probabilistic network. Once the network is constructed, the system is ready for user interaction.

There is increasing interest in extending the probabilistic formulism for uncertainty management into a distributed *multi-agent* environment [9, 10, 12, 13]. This environment consists of multiple agents, denoted $S_1, S_2, \ldots, S_n$, perhaps located at distinct sites. We assume that each agent's knowledge is represented by a marginal distribution of the joint probability distribution. As in the single agent environment the joint probability distribution $\mathbf{r}$ satisfies a PAJD. However, in the multi-agent environment the following two conditions are imposed:

(i) the marginal distribution at each site satisfies a PAJD, and

(ii) the marginal distribution on the intersection attributes between any two sites satisfies a PAJD.

Conditions (i) and (ii) ensure that the joint probability distribution is well-defined [12].

There are at least three methods for constructing a multi-agent probabilistic network. The *multiply-sectioned Bayesian network* [14] technique can be applied to section a large Bayesian network into multiple local Bayesian networks. These local Bayesian networks are then transformed into a PAJD satisfying conditions (i) and (ii). Another method [13] involves the situation where the agents are cooperative yet desire to conceal their internal attributes. The solution here is simply to check if the combination of the local DAGs form a multi-agent DAG under the imposed condition of privacy. If so, the agents can work together to transform the representation of the probabilistic knowledge into a PAJD satisfying conditions (i) and (ii). A more robust method [9] for constructing a multi-agent probabilistic network is to allow each domain expert to supply any known PMVDs and not necessarily an explicit DAG. The method determines a minimal cover of all the supplied independency information by detecting all inconsistent information, and removing all redundant information. A unique acyclic hypergraph of the multi-agent probabilistic network can be constructed directly from this minimal cover. In fact, it was shown that the constructed acyclic hypergraph is a *perfect-map* [6] of the minimal cover. That is, every PMVD logically implied by the minimal cover can be inferred from the acyclic hypergraph, and every PMVD inferred from the acyclic hypergraph is logically implied by the minimal cover. The constructed acyclic hypergraph can be sectioned to satisfy conditions (i) and (ii).

Example 2.2 Consider the constructed multi-agent probabilistic network depicted in Figure 6. The PAJD $\otimes\mathcal{H}$ is satisfied by the jpd $\mathbf{r}$, namely,

$$\mathbf{r} = (\ (\tau_{h_1}(\mathbf{r}) \otimes \tau_{h_2}(\mathbf{r})) \otimes \ldots) \otimes \tau_{h_{12}}(\mathbf{r}),$$

where $\mathcal{H} = \{h_1, h_2, \ldots, h_{12}\}$ is an acyclic hypergraph. The marginal distribution at site S_1 satisfies the PAJD,

$$\tau_{S_1}(\mathbf{r}) \quad = \quad ((\tau_{h_1}(\mathbf{r}) \otimes \tau_{h_2}(\mathbf{r})) \otimes \ldots) \otimes \tau_{h_6}(\mathbf{r}), \quad (11)$$

where the subscript S_1 in $\tau_{S_1}(\mathbf{r})$ is an abbreviation for the set of attributes at S_1. Similarly, the marginal distribution at site S_2 satisfies the PAJD

$$\tau_{S_2}(\mathbf{r}) \quad = \quad ((\tau_{h_7}(\mathbf{r}) \otimes \tau_{h_8}(\mathbf{r})) \otimes \ldots) \otimes \tau_{h_{12}}(\mathbf{r}). \quad (12)$$

Equations (11) and (12) indicate that condition (i) is satisfied. Condition (ii) is also satisfied since the marginal distribution on the intersection attributes satisfies the PAJD

$$\tau_{A_1A_2\cdots A_9}(\mathbf{r}) = (\tau_{A_1A_2A_3}(\mathbf{r}) \otimes \tau_{A_3A_4A_5A_6}(\mathbf{r})) \otimes \tau_{A_6A_7A_8A_9}(\mathbf{r}). \quad (13)$$

Agents S_1 and S_2 can take advantage of the respective PAJDs in equations (11) and (12) to optimize local processing. Both agents can take advantage of the PAJD in equation (13) to reduce the amount of data transmitted in distributed processing.

The first cooperative, interpretive system which applies probabilistic techniques for managing uncertainty in a distributed multi-agent environment was proposed in [12]. That system, while innovative, exhibits several undesirable characteristics including: (i) each agent processes evidence and answers local probabilistic queries *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* queries. There is no concept of a distributed query; (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; (iv) no provisions are made for transaction or agent *failure*.

Aside from these undesirable characteristics, the claim was made in [12] 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

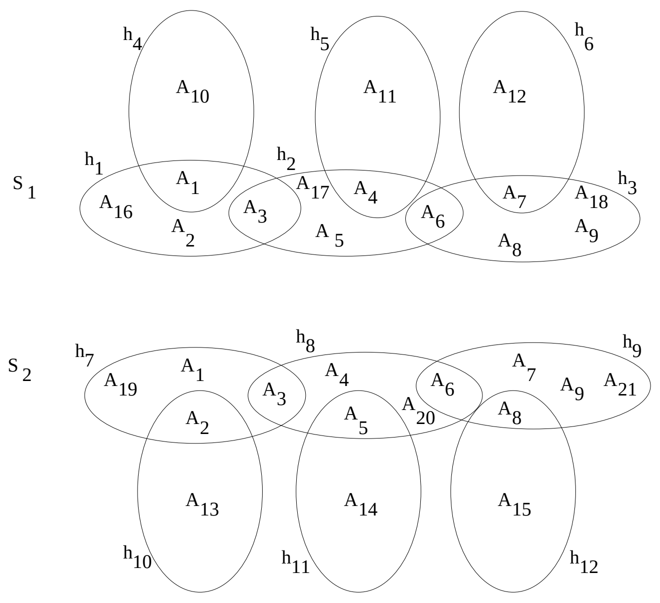

Figure 6. A multi-agent system consisting of two agents S_1 and S_2.

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 attributes 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.

Section 2.3 explicitly demonstrates that a probabilistic network is simply a generalized relational database. It immediately follows that relational database techniques can be modified where necessary for probabilistic reasoning purposes. In [10], two-phase locking results were applied to ensure *consistency* at all times. Maintaining global consistency permits the introduction of *distributed* queries into the multi-agent environment [10]. In this case, the *semi-join* operator used in distributed databases can be applied to reduce the amount of data transmitted between the agents in answering this type of request. In this paper, the problems of transaction and agent failure are discussed.

3. Problems Managing Agent Failure and Recovery

In this section, we use examples from [2] to demonstrate the problems involved in managing agent failure and recovery. (For convenience, we will use the terms agent and site interchangeably.)

There is a simple method to manage replicated attributes if agents never fail. Process Read(X) (or $R(X)$) by reading any copy of $\tau_X(\mathbf{r})$ and process Write(X) (or $W(X)$) by writing all copies of $\tau_X(\mathbf{r})$. Asynchronous control is achieved by *distributed two-phase locking* [3].

In practice, however, agents will fail and subsequently recover. Agent failure can be managed by extending the simple method as follows. Process $R(X)$ by reading any copy of $\tau_X(\mathbf{r})$ at an "up" agent. Process $W(X)$ by updating all copies of $\tau_X(\mathbf{r})$ at "up" agents and ignoring copies at "down" agents. This simple method of managing agent failure is not necessarily correct.

Example 3.1 Consider a distributed multi-agent environment representing distributions $\tau_X(\mathbf{r})$ and $\tau_Y(\mathbf{r})$ with copies $\tau_{X_{S_1}}(\mathbf{r})$, $\tau_{X_{S_2}}(\mathbf{r})$, $\tau_{Y_{S_3}}(\mathbf{r})$ and $\tau_{Y_{S_4}}(\mathbf{r})$. Consider two *transactions* [10] T_1 and T_2, where T_1 reads $\tau_X(\mathbf{r})$ then writes $\tau_Y(\mathbf{r})$ and T_2 reads $\tau_Y(\mathbf{r})$ then writes $\tau_X(\mathbf{r})$. The simple method *allows* the following execution:

$$R_1(X_{S_1}), R_2(Y_{S_4}), S_1\sqcap, S_4\sqcap, W_1(Y_{S_3}), W_2(X_{S_2}).$$

where $S_i\sqcap$ denotes the failure of agent S_i.

T_1 and T_2 each set a read-lock on the corresponding copy and then perform their reads asynchronously. Next agents S_1 and S_4 fail. Finally, T_1 and T_2 perform their writes asynchronously. T_1 writes $\tau_Y(\mathbf{r})$ by locking all copies at "up" agents, namely, $\tau_{Y_{S_3}}(\mathbf{r})$. Note that T_1's write-lock on $\tau_{Y_{S_3}}(\mathbf{r})$ does not conflict with T_2's read-lock on $\tau_{Y_{S_4}}(\mathbf{r})$ because $\tau_{Y_{S_3}}(\mathbf{r})$ and $\tau_{Y_{S_4}}(\mathbf{r})$ are different copies at different agents. T_2 writes $\tau_X(\mathbf{r})$ similarly.

This execution is *not* correct since T_2 does not read the distribution $\tau_Y(\mathbf{r})$ written by T_1, and T_1 does not read the distribution $\tau_X(\mathbf{r})$ written by T_2. One of these two cases must occur in a serial execution of T_1 and T_2 in a single copy environment.

Managing agent recovery is slightly more complex. Consider a distribution $\tau_X(\mathbf{r})$ with copies $\tau_{X_{S_1}}(\mathbf{r})$ and $\tau_{X_{S_2}}(\mathbf{r})$ at agents S_1 and S_2, respectively. Suppose agent S_1 fails. Copy $\tau_{X_{S_1}}(\mathbf{r})$ may become outdated while S_1 is down since copy $\tau_{X_{S_2}}(\mathbf{r})$ may be updated. Copy $\tau_{X_{S_1}}(\mathbf{r})$ must be updated when agent S_1 recovers before users access it. A simple method to accomplish this is to copy the value of $\tau_{X_{S_2}}(\mathbf{r})$ into $\tau_{X_{S_1}}(\mathbf{r})$. This simple method of handling agent recovery is not necessarily correct.

Example 3.2 Consider a distributed multi-agent environment with four agents S_1, S_2, S_3, and S_4. Let $\tau_X(\mathbf{r})$ and $\tau_Y(\mathbf{r})$ be distributions with copies $\tau_{X_{S_1}}$, $\tau_{X_{S_2}}$, $\tau_{X_{S_3}}$, and $\tau_{Y_{S_4}}(\mathbf{r})$. Consider two transactions T_1 and T_2, where T_1 writes $\tau_X(\mathbf{r})$ and then reads $\tau_Y(\mathbf{r})$, and T_2 reads $\tau_X(\mathbf{r})$ and then writes $\tau_Y(\mathbf{r})$. The simple method for handling agent recovery *allows* the following execution:

$$W_1(X_{S_3}), R_{in}(X_{S_1}), W_{in}(X_{S_2}), W_1(X_{S_1}), R_2(X_{S_2}),$$
$$R_1(Y_{S_4}), W_2(Y_{S_4}),$$

where $Read_{in}(X_{S_1})$ and $Write_{in}(X_{S_2})$ are issued by agent S_2 to update copy $\tau_{X_{S_2}}(\mathbf{r})$.

Since agent S_2 has failed when the execution begins, T_1 only updates copies $\tau_{X_{S_1}}(\mathbf{r})$ and $\tau_{X_{S_3}}(\mathbf{r})$. However, S_2 recovers in parallel with T_1 by reading the outdated copy $\tau_{X_{S_1}}(\mathbf{r})$. Since S_2 has recovered by the time T_2 starts, T_2 subsequently reads the outdated copy $\tau_{X_{S_2}}(\mathbf{r})$.

This execution is *not* correct since it is not equivalent to a serial execution of T_1 and T_2 in a single copy environment. In a serial execution where T_1 is executed first, T_2 reads the distribution $\tau_X(\mathbf{r})$ written by T_1. However, T_2 reads a prior value of $\tau_X(\mathbf{r})$ in the above execution. A similar argument holds if T_2 is executed first in a serial execution.

In the next section, recovery protocols are presented which only produce *correct* executions.

4. Recovery Protocols in Multi-Agent Probabilistic Reasoning Systems

In Section 2.3, we presented a generalized relational data model as the basis for developing probabilistic reasoning systems. It immediately follows that we can take full advantage of the techniques already implemented in existing database systems. While vast amounts of distributed and currency control techniques have been developed for relational databases, the exposition here draws from Bernstein et al. [2].

There are two correctness criteria for multi-agent probabilistic environments, namely, replication control and asynchronous control. *Replication control* states that multiple copies of a distribution should behave as a single copy as far as the user can tell. *Asynchronous control* states that the asynchronous execution of transactions must be equivalent to some serial execution of those transactions. Executing transactions serially in a single copy environment is the correctness criteria, called *1-serializability*, for asynchronous access with replicated attributes. Unlike the model in [12] (see Section 2.4) which violates *both* correctness criteria, our goal here is to define a multi-agent probabilistic reasoning system that exhibits 1-serializability.

We begin by defining some mathematical structures pertinent to our discussion. A transaction is a mathematical structure used to model asynchronous read and write operations for one user. Asynchronous execution of transactions are then modelled in a *history.*

An agent *translates* transaction T's read and write operations on distributions into operations on the replicated copies of those distributions. Hence, we define a function h that maps $R(X)$ into $R(X_S)$, where $\tau_{X_S}(\mathbf{r})$ is a copy of $\tau_X(\mathbf{r})$; each $W(X)$ into $W(X_{S_1}), \ldots, W(X_{S_q})$ for some copies $\tau_{X_{S_i}}(\mathbf{r})$, $0 < i \leq q$; each transaction commit into c_i; and each transaction abort into a_i.

A *complete replicated data* (RD) history H over $T = \{T_1, \ldots, T_n\}$ is a partial order with ordering relation $<$ where

(i) $H = h(\cup_{i=1}^{l} T_i)$ for some translation function h;

(ii) For each T_i and all operations P_i, Q_i in T_i, if $P_i <_i Q_i$, then every operation in $h(P_i)$ is related by $<$ to every operation in $h(Q_i)$;

(iii) For every $R_j(X_A)$, there is at least one $W_i(X_A) < R_j(X_A)$;

(iv) All pairs of conflicting operations are related by $<$, where two operations *conflict* if they operate on the same *copy* and at least one of them is a write operation; and

(v) If $W_i(X) <_i R_i(X)$ and $h(R_i(X)) = R_i(X_A)$ then $W_i(X_A) \in h(W_i(X))$.

The nodes in a *serialization graph* (SG) [2] for an RD history correspond to committed transactions in the history, and there is an edge $T_i \rightarrow T_j$ if there are conflicting operations P_i in T_i and Q_j in T_j such that $P_i < Q_j$. Unfortunately, standard SGs for RD histories are too weak for our purposes.

Example 4.1 Consider a multi-agent environment representing distributions $\tau_X(\mathbf{r})$ and $\tau_Y(\mathbf{r})$ with copies $\tau_{X_{S_1}}(\mathbf{r})$, $\tau_{X_{S_2}}(\mathbf{r})$, $\tau_{Y_{S_3}}(\mathbf{r})$, and $\tau_{Y_{S_4}}(\mathbf{r})$. Suppose we have three transactions T_1, T_2 and T_3: T_1 wants to update $\tau_X(\mathbf{r})$ and $\tau_Y(\mathbf{r})$; T_2 wants to read $\tau_X(\mathbf{r})$ then update $\tau_Y(\mathbf{r})$; T_3 wants to read $\tau_Y(\mathbf{r})$ then update $\tau_X(\mathbf{r})$. If the agent uses two-phase locking and ignores failed copies, history H_1 depicted in Figure 7 is *allowed* to occur, where $X_{S_i}\sqcap$ denotes the failure of copy $\tau_{X_{S_i}}(\mathbf{r})$. (Here we find it convenient to speak of copies, rather than agents, failing.)

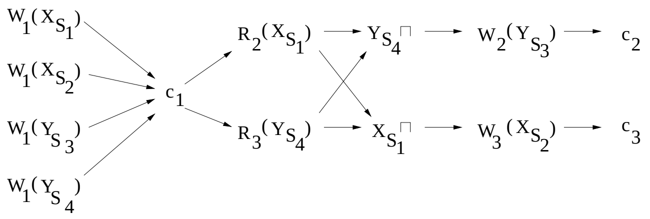

Figure 7. History H_1 models an *incorrect* execution even though $SG(H_1)$ is acyclic.

Transactions T_2 and T_3 begin by reading $\tau_{X_{S_1}}(\mathbf{r})$ and $\tau_{Y_{S_4}}(\mathbf{r})$. The copies that they read fail after they complete their reads. The agent translates $W_2(Y)$ into $W_2(Y_{S_3})$ since $\tau_{Y_{S_3}}(\mathbf{r})$ is the only available copy of $\tau_Y(\mathbf{r})$. Similarly, the agent translates $W_3(X)$ into $W_3(X_{S_2})$. The agent has no trouble locking each copy that T_2 and T_3 access because no two operations of these transactions access the same *copy* of any distribution. The important point is that H_1 is not correct since it is not 1-serializable, namely, H_1 is not equivalent to a serial history over $\{T_1, T_2, T_3\}$ on a single copy environment. A serial execution of T_1, T_2, and T_3 on a single copy environment would have either T_2 reading the value of $\tau_X(\mathbf{r})$ written by T_3, or T_3 reading the value of $\tau_Y(\mathbf{r})$ written by T_2. However, standard SGs are too weak for our purposes since it can be verified that $SG(H_1)$ is still *acyclic*. (Acyclicity in SGs is regarded as a correctness criteria [2].)

Example 4.1 explicitly demonstrates that simply translating $R(X)$ into $R(X_S)$ for some copy $\tau_{X_S}(\mathbf{r})$ and $W(X)$ into $W(X_S)$ for all available copies $\tau_{X_S}(\mathbf{r})$ may lead to an incorrect execution even if failed copies never recover. The problem here is that the logical conflicting access of T_2 and T_3 on $\tau_X(\mathbf{r})$ and $\tau_Y(\mathbf{r})$ was never manifested into conflicting access on *copies* of $\tau_X(\mathbf{r})$ and $\tau_Y(\mathbf{r})$. The task now is to develop a method to ensure that any two transactions that have conflicting access to the same distribution also have conflicting access to some copy of that distribution.

The *available copies algorithm* [2] handles replicated attributes by using a validation protocol, in conjunction with this read-any and write-all-available approach, to ensure correctness. The *validation protocol* consists of two steps:

(i) Missing writes validation: T_i makes sure that all copies it was unable to write are still unavailable, and

(ii) Access validation: T_i makes sure that all copies it read or wrote are still available.

This algorithm also assumes that the agent uses *strict two-phase locking*. The following two examples illustrate how the available copies algorithm ensures 1-serializability.

Let t_i denote the moment when transaction T_i *begins* its access validation step. By the available copies algorithm, t_i must follow all $R_i(X)$ and $W_i(X)$ as well as the missing writes validation step (if present), and must precede c_i.

Example 4.2 Consider the incorrect history H_1 in Figure 7 redrawn as history H_1' in Figure 8. Since T_2 read $\tau_{X_{S_1}}(\mathbf{r})$, the failure of $\tau_{X_{S_1}}(\mathbf{r})$ must have occurred after T_2 started its access validation. Otherwise, T_2 would have found $\tau_{X_{S_1}}(\mathbf{r})$ to be unavailable and would therefore abort. Therefore, $t_2 < X_{S_1}\sqcap$ and, similarly, $t_3 < Y_{S_4}\sqcap$. Since T_2 wrote $\tau_{Y_{S_3}}(\mathbf{r})$ but not $\tau_{Y_{S_4}}(\mathbf{r})$, it must have carried out the missing writes validation and found that $\tau_{Y_{S_4}}(\mathbf{r})$ is still unavailable. Since $\tau_{Y_{S_4}}(\mathbf{r})$ had already been initialized, it must be that $\tau_{Y_{S_4}}(\mathbf{r})$ failed before the completion of T_i's missing writes validation and thus before the beginning of access validation. Hence $Y_{S_4}\sqcap < t_2$. The precedence $X_{S_1}\sqcap < t_3$ is justified on similar grounds. Given these precedences we have a cycle in H_1': $t_2 < X_{S_1}\sqcap < t_3 < Y_{S_4}\sqcap < t_2$. This is impossible since H_1', being a history, is supposed to be a partial order. This means that H_1' could not have happened.

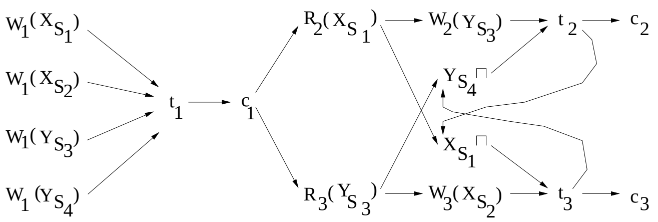

Figure 8. The incorrect history H_1 in Figure 7 redrawn as history H_1'.

The next example illustrates the significance that the agent uses strict two-phase locking.

Example 4.3 Let T_1, T_2 and T_3 be the same transactions as in H_1 except where there is only one copy $\tau_{Y_{S_3}}(\mathbf{r})$ of $\tau_Y(\mathbf{r})$. Consider history H_2 in Figure 9 ignoring for the moment the broken arrow from c_3 to $W_2(Y_{S_3})$. H_2 is not 1-serializable since T_2 does not read the distribution $\tau_X(\mathbf{r})$ written by T_3, and T_3 does not read the distribution $\tau_Y(\mathbf{r})$ written by T_2. One of these two cases must occur in a serial single copy environment. However, the validation protocol in the available copies algorithm does not prevent H_2 from occurring. Instead the assumption that the agent uses two-phase locking prohibits history H_2 from happening. Since T_3 read Y_{S_3} before T_2 updated it, T_3 must have locked $\tau_{Y_{S_3}}(\mathbf{r})$ before T_2 did. By strict two-phase locking, T_2 won't lock $\tau_{Y_{S_3}}(\mathbf{r})$ until after T_3 has committed. That is, we have the precedence $c_3 < W_2(Y_{S_3})$. Given these precedences we have a cycle in H_2: $t_3 < c_3 < W_2(Y_{S_3}) < t_2 < X_{S_1}\sqcap < t_3$. This is impossible since H_2, being a history, is supposed to be a partial order. This means that H_2 could not have happened.

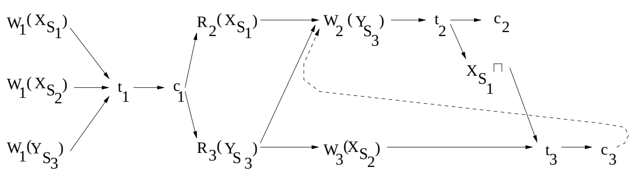

Figure 9. History H_1 in Figure 7 with only one copy of $\tau_Y(\mathbf{r})$ redrawn as history H_2.

Theorem 4.1 [2] The available copies algorithm only produces 1-serializable histories.

The model proposed in [12] for probabilistic reasoning in a multi-agent environment does not include provisions for transaction or agent failure, nor exhibit 1-serializability. In fact, it violates *both* correctness criteria, namely, replication control and asynchronous control. On the other hand, Theorem 4.1 indicates that our generalized relational data model will only produce 1-serializable histories even in light of transaction and agent failure.

5. Conclusion

In this paper, we have shown that probabilistic networks are generalized relational databases. Our *probabilistic relational data model* then serves as a basis for developing multi-agent probabilistic reasoning systems. Since our model subsumes the traditional relational data model, it immediately follows that we can take full advantage of the existing distributed and concurrency control techniques to address the undesirable characteristics exhibited by current multi-agent probabilistic reasoning systems.

A rigorous generalization of the traditional relational model has important theoretical and practical ramifications. In practice, one intelligent system can be designed for both of these problems by including the appropriate relational operators. On the theoretical side, a unified model provides the opportunity for the cross-fertilization of techniques between Bayesian networks and relational databases.

References

[1] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, July 1983.

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

[3] K. Eswaran, J. Gray, R. Lorie, and I. Traiger. The notions of consistency and predicate locks in a database system. *Communication of ACM*, 19(11):624–633, November 1976.

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

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

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

[7] G. Shafer. An axiomatic study of computation in hypertrees. School of Business Working Papers 232, University of Kansas, 1991.

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

[9] S. Wong and C. Butz. Constructing the dependency structure of a multi-agent probabilistic network. *Submitted for publication*, 1998.

[10] S. Wong and C. Butz. Probabilistic reasoning in a distributed multi-agent environment. In *Third International Conference on Multi-Agent Systems*, 1998.

[11] 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.

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

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

[14] Y. Xiang, D. Poole, and M. Beddoes. Multiply sectioned bayesian networks and junction forests for large knowledge based systems. *Computational Intelligence*, 9:171–220, 1993.