SUPPORTING PROBABILISTIC DATA IN A RELATIONAL SYSTEM

Hector Garcia-Molina
Daryl Porter

CS-TR-147-88

February 1988

Supporting Probabilistic Data in a Relational System

Hector Garcia-Molina
Daryl Porter

Department of Computer Science
Princeton University
Princeton, New Jersey 08544

ABSTRACT

It is often desirable to represent in a database entities whose properties cannot be deterministically classified. We develop a new data model that includes probabilities or confidences associated with the values of the attributes. Thus we can think of the attributes as random variables with probability distributions dependent on the entity the tuple purportedly describes. This new model offers a richer descriptive language allowing the database to more accurately reflect the uncertain real world. It also offers a new interpretation of information incompleteness. We study three sets of issues: the proper model for probabilistic data, the semantics of probabilistic data, and the choice of operators and language necessary to manipulate such data.

February 14, 1988

Supporting Probabilistic Data in a Relational System

Hector Garcia-Molina
Daryl Porter

Department of Computer Science
Princeton University
Princeton, New Jersey 08544

I. Probabilistic Model

Entities with stochastic attributes abound in real database applications. For example, Toyota might have demographic information indicating that customers living in a certain region are likely to purchase a Corolla with probability 0.7 or a Celica with probability 0.3. An oil company might have a database of potential *Oil Sites* with probabilistic projections as to the *Type* and *Quantity* of oil at each site. Military applications generate many enticing probabilistic scenarios: Entities such as *Military Bases* where information about hardware at the camp is uncertain or an entity set like *Spies* whose *Location* attribute is the product of guesswork. In a traditional database application, *Employees*, the qualitative attributes *Work Quality*, or *Enthusiasm* could be introduced. The stock market features *Companies* with many attributes that are notoriously non-deterministic. Finally, Medical Records describing, say, a patient's *Susceptibility to Heart Attack* or the *Causes of a Disease* could be stochastic in nature.

In this paper we develop the Probabilistic Data Model (PDM), an extension of the information incompleteness model of Lipski in [Imi3, Lip, Lip2], that includes probabilities (or confidences) associated with the values of attributes. The PDM will offer a relational scheme well suited for representing the kinds of applications discussed above.

The PDM features relations whose key is deterministic in nature. That is, the relation describes a set of real entities such as EMPLOYEES where the key might be <NAME, ADDRESS>. The rest of the attributes describe properties of these entities and can be deterministic in nature (i.e., the DEPARTMENT they work for) or stochastic (i.e., the QUALITY of work they do assessed through a vote by ten people). These dependent attributes are necessarily conditional on the value of the key, but they may be independent of each other, jointly distributed, or a mixture of both.

EXAMPLE 1.1 *Probabilistic Relation COMPANY*

Key	Independent Deterministic	Interdependent Stochastic	Independent Stochastic
EMPLOYEE	DEPARTMENT	QUALITY BONUS	SALES
Jon Smith	Toy	0.4 [Great YES] 0.5 [Good YES] 0.1 [Fair NO]	0.3 $30,000 0.5 $35,000
Fred Jones	Houseware	1.0 [Good YES]	0.5 $20,000 0.5 $25,000

The relation in this example describes two entities, "Jon Smith" and "Fred Jones." Attribute DEPARTMENT is deterministic. e.g., it is certain that Jon Smith works in the Toy department. In this example, QUALITY and BONUS are probabilistic and jointly distributed. For instance,

P[QUALITY = "Great" AND BONUS = "YES" | EMPLOYEE = "Jon Smith"] = 0.4

This seems especially sensible if, as the example implies, QUALITY functionally determines BONUS. The last attribute, SALES, describes the expected sales by the employee. It is probabilistic but independent of the other non-key attributes. For instance,

P[SALES = "$35,000" | EMPLOYEE = "Jon Smith"] = 0.5.

Note that in Example 1.1 the SALES probabilities for Jon Smith do not add up to 1.0. This is an important feature of our model: we allow for *missing probability*. In our model we assume that the missing probability is distributed over the remaining domain values, but we make *no assumptions* as to how it is distributed. In our sales example, the missing 0.2 probability is distributed over all possible sales values except 30,000 and 35,000. All of the 0.2 could correspond to some value, say 10,000, or it could be uniformly distributed, or any other possibility.

Other interpretations of missing probability are possible but not considered here. For example, in some cases the missing probability may be distributed over *all* values, including those explicitly listed. (For instance, suppose that 10 people are polled to estimate Jon Smith's sales for the upcoming year. For simplicity, only two values are being considered: last year's sales of $30,000, or last year's sales plus inflation $35,000. Three people have estimated sales of 30,000; five of 35,000. Two people have not been reached and give rise to the 0.2 missing probability.) Under this interpretation, we estimate the Jon Smith's sales will be 30,000 with probability somewhere between 0.3 and 0.5. This uncertainty in the probabilities is one complication we wish to avoid, at least in this paper.

In other situations, we may wish to assume that the missing probability is *uniformly* distributed [Mey] over the remaining values. Since in this case the distribution is known, we represent it with the shorthand notation U[::] to be presented in Section V. In the example, 0.2 U[::] would be included as a third entry for the SALES distribution for tuple Jon Smith. The probabilities now add up to one and there is no missing probability.

We believe that missing probability is a powerful concept. It makes it easier to input data into a probabilistic relation, i.e., it is not necessary to have all information before some tuple can be entered. It also makes it possible to eliminate uninteresting information when displaying relations. For example, a user may only be interested in seeing values that have a probability greater than 0.5; the rest can be ignored. Finally, as we will see later on, missing probability arises naturally during relational operations, even when the base relations have no missing probability.

A central premise of our model is that keys are deterministic. This is not the only choice, but we feel that deterministic keys are very natural and lead to simple relational operators. Furthermore, it is still possible to represent stochastic entities with our model. For instance, suppose that we are not certain whether Jon Smith and Fred Jones are employees in company ACME of Example 1.1. Say the probability that Jon Smith "exists," i.e., works for ACME, is 0.7. We can add an attribute COMPANY to our relation, and add the value 0.7 ACME for Jon Smith. The 0.3 missing probability is the likelihood that Smith does not exist at ACME.

Before continuing it will be useful to briefly discuss where the "probabilistic data" for our relations come from. There are actually many possibilities. One is to have users assign the probabilities according to their "confidence" or "belief" in the values. Thus a user, after reading recommendation letters for a job candidate, may say that he is 80% confident that this is a good candidate.

A second option is to compute the probabilities from an underlying sample. For example, a conventional deterministic relation may contain the evaluations of Fred Jones made by 100 people. Say 15 people find Jones "Very Good", 85 find him "Good". The relative frequencies can be interpreted as probabilities, leading to the following probabilistic relation:

EXAMPLE 1.2 *Relative Frequency Relation*

NAME	JUDGMENT
Jones	15/100 Very Good 85/100 Good

In Section IV we will present a new operator (Stochastic) that will automatically generate a probabilistic relation like the one in Example 1.2 out of an underlying deterministic relation containing samples.

Probabilities can also be assigned to data according to their timeliness. For instance, suppose that at 9:00am the position of a ship is reported as X. As time progresses, the likelihood that the ship is still at X decreases, so the user may provide a time decreasing function to generate the probabilities attached to positions. (Note that missing probability is useful once again.)

Finally, probabilities can also be generated out of conflicting data. Say two relations report the positions of ships. If one reports a given ship at X and the second one at Y, it may be desirable to report the ship at X with probability 0.5 and at Y with probability 0.5. In Section IV we will present another new operator, Combine, that allows us to perform this type of

operation. Combine takes two probabilistic (or deterministic) relations and merges them into a single one, computing new probabilities.

Incidentally, in this paper we will focus exclusively on discrete probability distribution functions, either explicitly enumerated (as in our examples so far), or via known distributions (e.g., uniform; see Section V.) However, it may be possible to extend our model to continuous probabilities, and this opens the door to other sources of probabilistic data. For example, we may know that a given radar has a normally distributed error with standard deviation of one mile. Thus, the position of an airplane will be given by a normal distribution.

In the rest of this paper we will discuss our model and its relational operators in more detail. In the next section we briefly look at prior work that is related to ours. Section III explores "conventional" relational operations (e.g., select, join) on probabilistic data, while Section IV deals with our proposed new operators (e.g., stochastic, combine). In Section V we study the use of discrete distributions, in particular the uniform one.

II. Related Work

Researchers have devoted much work to extending the traditional complete information model to more accurately reflect the real world. The following examples summarize the evolution of the incomplete information models.

EXAMPLE 2.1 *Null Value Model*

EMPLOYEE	SALES
Jon Smith	$30,000
Jim Jones	**???????**

Here, Jim Jones has been hired but no SALES estimate is known. NULL values are studied at length in [Imi, Imi2, Mor, Rei, Rot2, Vas].

EXAMPLE 2.2 *Null Value / Attribute Not Applicable Model*

EMPLOYEE	SALES	SPOUSE
Jon Smith	$30,000	*N/A*
Jim Jones	**???????**	Susan Jones

The database above represents the bachelor Jon Smith and Jim Jones who is married. The N/A model is studied in [Mor, Rot2].

EXAMPLE 2.3 *Lipski's Small Subset Model* from [Imi3, Lip, Lip2]

EMPLOYEE	SALES	SPOUSE
Fred Jones	**{$30K, $35K}**	N/A
Jon Smith	???????	Susan Jones

Here, it is expected that Fred Jones will have sales of either $30,000 or $35,000. Note how more information is present here than with the traditional NULL value in the Jon Smith tuple.

EXAMPLE 2.4 *Tuple Space as Sample Space* [*Cav*] A,B ∈ {0, 1}

A	B	p(Tuple = .)
0	0	0.35
0	1	0.25
1	0	0.01
1	1	0.39

In the model proposed in [Cav], each probability represents the likelihood that a tuple appears in a relation. For example, the probability that tuple <0,0> exists is 0.35

Fuzzy relations are discussed briefly in [Gar] and referenced as an area of future research in [Cav]. Each relation is viewed as a fuzzy set, with tuples that may or may not be in a relation. A related idea, fuzzy response, is described in [Mor]. Here, the response to a query is viewed as a fuzzy set, and the tuples are once again given a confidence rating as to whether they belong in the response or not. Two additional papers that discuss incomplete information are [Liu, Men].

In [Won], Wong views queries as statistical experiments in which information was incomplete because, for example, it was old. His method centered around computing the query response as those tuples which minimized the two types of statistical error. In another paper [Gho], Ghosh discusses statistical relational tables. Although there are similarities (see for instance the operator STOCHASTIC we define in Section IV), our main interest in this paper is on uncertain, not statistical information.

The PDM model we are proposing here was of course inspired by many of the prior ideas relating to uncertain information. In a way, it can be viewed as a generalization of many of these concepts. Clearly, conventional relations can be represented by making all attributes deterministic. Null values can be represented by making all probabilities missing (e.g., a blank entry for an attribute). (As an alternative, we may specify 1.0 U(::), indicating that all values are equally likely.) The N/A value could be represented, as is commonly done, by a distinguished and unique domain value. To model Lipski's sets, a uniform density function could be assigned to the members of the set. (For example, the sales of Fred Jones in Example 2.3 could be represented as it is in Example 1.1.) The existence of tuples in a relation (as done in [Cav]) could be simulated as we suggested in Section 1. We believe that the generalization we are proposing here is quite natural and allows representation of a broader set of applications.

Our work is also related to the current efforts with non-first normal form relations [Dad, Rot, Rot2]. As a matter of fact, our probabilistic relations are simply nested relations with depth two. Each attribute in the top level relation can be a relation that gives the probability distribution function for the attribute. For example, the relation of Example 1.1 can be described with the following schema, using the notation of [Rot]:

COMPANY → <EMPLOYEE, DEPARTMENT, QB, SALES>

DEPARTMENT → <P(.), D_NAME>

QB → <P(.), QUALITY, BONUS>

SALES → <P(.), C_AMOUNT>

If names do not appear on any left hand side then we have an atomic valued attribute. In the Appendix we describe a simpler schema description tailored more to our probabilistic application. Our focus in this paper will be on the probabilistic operators and not on the non-first normal form issues.

III. Relational Algebra

In this section we look at the conventional relational algebra operators project, select, join, and cartesian product. On deterministic relations, these operators work exactly as one would expect. On probabilistic relations, the results are not conventional although similar in character to the deterministic case. Here we will only study the more interesting case, probabilistic relations.

Our discussion will center on a set of representative examples. We believe this is the most effective way of explaining our ideas. A more formal description of our proposed syntax can be found in the Appendix. We have no space to present detailed algorithms for performing each operator. However, we feel that, given the examples, developing the algorithms is straightforward.

Our relational operators can manipulate both the top level relation and the second level attribute relations. To refer to a top level relation we use its name; for an inner relation we use R.A_B_C where R is the top level name and A, B, C, ... are the jointly distributed attributes. We start by discussing the project operator. For our examples we use the following relation.

PROJECT EXAMPLE *Relation R*

KEY	A B	C
k1	0.3 [a1 b1] 0.5 [a2 b1] 0.2 [a2 b2]	0.4 c1 0.6 c2
k2	1.0 [a2 b2]	0.5 c2 0.5 c3

EXAMPLE 3.1.1 *Relation Level Project*
QUERY: PROJECT onto KEY, B attributes
SYNTAX: $\Pi_{KEY,B}(R)$

This amounts to computing the marginal probability distribution. The result is:

KEY	B
k1	0.8 b1 0.2 b2
k2	1.0 b2

EXAMPLE 3.1.2 *Relation within Relation Project*
QUERY: PROJECT out A portion of A_B attribute group.
SYNTAX: $\Pi_B(R.A\_B)$

This is a project within the second level relation R.A_B. This leaves the rest of the tuple intact, affecting only the attribute group named. The result is:

KEY	B	C
k1	0.8 b1 0.2 b2	0.4 c1 0.6 c2
k2	1.0 b2	0.5 c2 0.5 c3

EXAMPLE 3.1.3 *Projecting out the KEY*
QUERY: PROJECT onto A, B, C attributes
SYNTAX: $\Pi_{A,B,C}(R)$

In our two previous examples the relation key was not projected out. In this example, it is. In our model all tuples must have deterministic keys, so the result will contain a new key: the name of the relation, R. The one tuple in the result will contain the "expected" distributions for each attribute group in R. For our example, the result is:

KEY	A B	C
R	0.15 [a1 b1] 0.25 [a2 b1] 0.60 [a2 b2]	0.20 c1 0.55 c2 0.25 c3

To illustrate, consider the 0.55 probability associated with attribute value c2. This means that if we get a random tuple from R, it will have a c2 attribute value (for C) with probability 0.55. (computed as 0.6*0.5 plus 0.5*0.5). Thus, the resulting relation provides a "probabilistic summary" of the original relation. There are actually several other types of summaries that may be computed, but we will discuss these in Section IV when we present the COMBINE operator.

We now turn our attention to selects. As with projects, a select can operate on the top level relation or on an inner relation. The select condition can refer to the attribute values, their probabilities, or both. A condition on a single attribute group A_B_C will be of the form

A_B_C: value condition, probability condition

In the value condition, **V** refers to the value of the attribute, and "*" represents "do not care." In the probability condition, the probability of the value is **P**. Conditions on more than one attributes can be combined with logical ANDs, ORs, etc. in a conventional fashion. (If an attribute is deterministic, a condition can be written in the standard way, e.g., A: **V** > 5 can be written as A > 5.) (A more elegant but "harder to explain in limited space" syntax for select conditions is defined in [Gar]. Recall that our simpler syntax is summarized in the Appendix.)

For our examples we use the following relation:

SELECT EXAMPLE *Relation R*

KEY	A B C	D
k1	0.3 [a1 b1 c1] 0.5 [a2 b1 c1] 0.2 [a2 b2 c1]	0.4 d1 0.6 d2
k2	0.7 [a2 b2 c1] 0.3 [a3 b2 c2]	0.5 d2 0.5 d3

EXAMPLE 3.2.1 *Probability Range Select*
QUERY: SELECT all tuples that have a value of d2 for their D attribute with probability greater than 0.3.
SYNTAX: $\sigma_{D:\ \mathbf{V} = d1,\ \mathbf{P} > 0.3}(R)$
Since this is a select over the top level relation (select over R), the result contains complete R tuples. In our example, only tuple k1 satisfies the condition and is in the result.

EXAMPLE 3.2.2 *Relation Within Relation Select*
QUERY: Display only those portions of A_B_C with B = b1.
SYNTAX: $\sigma_{A\_B\_C:\ \mathbf{V} = [*, b1, *]}(R.A\_B\_C)$
The result is:

KEY	A B C	D
k1	0.3 [a1 b1 c1] 0.5 [a2 b1 c1]	0.4 d1 0.6 d2
k2		0.5 d2 0.5 d3

Since this is a select over R.A_B_C, all tuples originally in R appear in the result. In the distributions for the A_B_C group, the lines that do not satisfy the condition are dropped. Also note that in the condition **V** = [*, b1, *] the "*" is used for attributes where there is no condition.

As the example shows, selects can generate missing probability. In this case, the user is not interested in probabilities of tuples without a b1 value. We have lost information in the process. For instance, looking at the result we now have no information for the A_B_C group

of the k2 tuple. However, this loss may be desirable so the user can concentrate of the data that he is interested in.

EXAMPLE 3.2.3 *Lipski Upper Bound Select*
QUERY: SELECT all tuples such that we can't rule out that A = a3.
SYNTAX: $\sigma_{A\_B\_C:\ \mathbf{V} = [a3,*,*],\ \mathbf{P} > 0}(R)$
The result is the k2 tuple.

EXAMPLE 3.2.4 *Lipski Lower Bound Select*
QUERY: Find all KEY values such that it is certain that C = c1.
SYNTAX: $\Pi_{KEY}(\sigma_{C:\ \mathbf{V} = c1,\ \mathbf{P} = 1.0}(\Pi_{KEY,C}(R)))$
The intermediate and final results are:

1. PROJECT onto KEY, C

KEY	C
k1	1.0 c1
k2	0.7 c1 0.3 c2

2. SELECT all tuples with P[C=c1|KEY=?]=1.0

KEY	C
k1	1.0 c1

3. PROJECT onto KEY

KEY
k1

All of our select examples have used a relation with no missing probability. If there is missing probability, the select operator works in the same fashion (only considering the existing probabilities); however, we have to be careful in interpreting the results. Consider once again example 3.2.3, "Select all tuples where we cannot rule out that A = a3." Suppose that we change R so that tuple k1 now has some missing probability for A_B_C (e.g., delete the 0.3 [a1,b1,c1] entry). The select operation of example 3.2.3 would not select tuple k1, even though it is now conceivable that the value [a3,*,*] does exist with some probability greater than 0.

If we wish to force the select to consider missing probability, we use **MP** instead of **P** in the query, i.e.,

$$\sigma_{A\_B\_C:\ \mathbf{V} = [a3,*,*],\ \mathbf{MP} > 0}(R)$$

This would select both tuples in the modified R.

In general, we would expect users to ignore missing probability in their selects (**P**) where the number of values in the domain is very large and the probability of a nonappearing value is "insignificant." If the domain is small and the amount of missing probability is significant, users may prefer to consider the missing probability (**MP**).

Next we consider natural joins. We restrict our attention to the case where join is most natural, i.e., when the join attribute is the key of one relation, a foreign key of the other relation, and the condition is equality. To motivate the need for this type of join, consider the following example.

EXAMPLE 3.3.1 *Natural Join*

Consider the following two relations:

SHIPS

NAME	TYPE
Maria	0.6 Frigate 0.4 Tugboat

DESCR

TYPE	MAX-SPEED
Frigate	0.7 20-knots 0.3 30-knots
Tugboat	1.0 15-knots

We are 40% sure the Maria is a Tugboat. We know that Tugboats have a maximum speed of 15 knots, so it makes sense to say that we are 40% sure the Maria's maximum speed is 15 knots. Similarly, we are 60% sure she is a Frigate, and Frigates have a maximum speed of 20 knots with probability 0.7, so it is reasonable to expect the Maria to have a maximum speed of 20 knots with probability 0.6 times 0.7. This motivates the natural join operator we now illustrate.

QUERY: JOIN relations SHIPS and DESCR over the common attribute TYPE (TYPE must be the key of one of the relations.)
SYNTAX: SHIPS >< DESCR

The resulting relation is:

NAME	TYPE MAX-SPEED
Maria	0.6*0.7 [Frigate 20-knots] 0.6*0.3 [Frigate 30-knots] 0.4*1.0 [Tugboat 15-knots]

The resulting probabilities are obtained by multiplying the corresponding probabilities in SHIPS and DESCR. This operation can be justified as follows. To simplify our expressions, we use the attribute initials, i.e., N will be NAME, T will be TYPE, and M will be MAX-SPEED.

- The T attribute of SHIPS gives $P[T \mid N]$.
- The M attribute of DESCR gives $P[M \mid T]$.
- In the resulting relation, the T_M attribute is computed as $P[T \mid N] \cdot P[M \mid T]$.
- This product can be rewritten as:

$$P[T \mid N] \cdot P[M \mid T]$$

$$= P[T \mid N] \cdot P[M \mid T,N] \qquad \text{(assuming N and M independent)}$$

$$= (P[T,N]/P[N]) \cdot (P[T,M,N]/P[T,N]) \qquad \text{(Bayes Rule)}$$

$$= P[T,M,N]/P[N] \qquad \text{(cancelling } P[T,N] \text{ term)}$$

$= P[T,M \mid N]$ 1

Thus our natural join operation computes what we expected, the probability that T and M jointly take the values given in SHIPS >< DESCR. The operation is valid as long as N and M are independent. This is implicit in the example relations.

Although not illustrated by our example, join also works if the relations have more than two attributes. For example, if the DESCR relation had an additional probabilistic attribute LENGTH, the result of the join would contain a second group TYPE_LENGTH.

EXAMPLE 3.3.2 *Natural Join that generates missing probability*
Suppose that the DESCR relation given earlier does not contain a tuple for Tugboat. The result of the join in this case would be:

NAME	TYPE MAX-SPEED
Maria	0.6*0.7 [Frigate 20-knots] 0.6*0.3 [Frigate 30-knots]

The resulting relation has 0.4 missing probability. The interpretation is quite natural. With probability 0.4 the TYPE_MAX-SPEED attribute takes on a value [Tugboat, ?], where "?" is some speed value. The distribution of this probability over all possible [Tugboat, ?] values is unknown and is not represented in the resulting join.

The last operator we present is Cartesian Product. It combines two relations representing *independent* entities into a single relation giving their joint probability distribution. We again use an example to illustrate.

EXAMPLE 3.4.1 *Cartesian Product*
QUERY: Cartesian product of R1 and R2 (displayed below).
SYNTAX: R1 X R2

Relation R1

KEY	A B	C
k1	0.3 [a1 b1] 0.7 [a1 b2]	0.4 c1 0.6 c2
k2	0.5 [a2 b2] 0.5 [a1 b3]	1.0 c1

Relation R2

KEY'	D E	F
K1'	0.4 [d2 e1] 0.6 [d1 e3]	1.0 f2
K2'	1.0 [d2 e1]	0.2 f1 0.8 f3

The result is as follows. (Attributes KEY and KEY' are the key of the result relation.)

KEY	KEY'	A B	C	D E	F
k1	K1'	0.3 [a1 b1] 0.7 [a1 b2]	0.4 c1 0.6 c2	0.4 [d2 e1] 0.6 [d1 e3]	1.0 f2
k1	K2'	0.3 [a1 b1] 0.7 [a1 b2]	0.4 c1 0.6 c2	1.0 [d2 e1]	0.2 f1 0.8 f3
k2	K1'	0.5 [a2 b2] 0.5 [a1 b3]	1.0 c1	0.4 [d2 e1] 0.6 [d1 e3]	1.0 f2
k2	K2'	0.5 [a2 b2] 0.5 [a1 b3]	1.0 c1	1.0 [d2 e1]	0.2 f1 0.8 f3

We emphasize that the cartesian product is only valid if the entities are independent. For example, P[A,B|KEY] must be equal to P[A,B|KEY, KEY']. The operation would not be valid with the relations of Example 3.3.1 (SHIPS, DESCR) where the attributes are dependent.

Our model also includes other relational operators like UNION, INTERSECT, and MINUS. They work just like their deterministic counterparts. In particular, tuples would only match for an intersection or a minus if they had identical values and distributions. In a Union, tuples with the same key but different distributions would be kept as two tuples (violating the key constraint). This is discussed further in the next section.

IV. New Operators

We have just finished exploring the implications of the PDM on the traditional relational algebra operators. In this section we present new operators that have no counterpart in conventional systems. The four operators are COMBINE, STOCHASTIC, DISCRETE, and GROUP.

The COMBINE operator is invoked in conjunction with another operator to eliminate tuples with duplicate key values. For example, suppose that two independent observers keep track of the distance of objects to a given site. Each observer generates a probabilistic relation for his data. If two such relations are available, it may be desirable to perform a union to combine the information. Say each relation contains distance data for an object k1. The resulting union relation will contain two k1 tuples (among others). This is illustrated on the left hand side table below. (For example, the first observer says the object is 2 miles away with probability 0.2. Both agree that the object is 3 miles away with probability 0.3.)

Original R

K	D
k1	0.2 2 0.3 3
k1	0.3 3 0.4 4

After COMBINE

	K	D
Average	k1	0.1 2 0.3 3 0.2 4
Sum	k1	0.06 5 0.17 6 0.12 7
Merge	k1	0.2 2 0.3 3 0.4 4

Since the distance information we have for k1 differs, the tuples are not quite duplicates so the standard union operator leaves both tuples. However, the key constraint has been violated so this is not a legal relation in our model. The solution is to combine or merge the two distributions into a single one. This is what COMBINE does. A COMBINE can be explicitly invoked with the command that requires it (union in our example), or an automatic COMBINE strategy can be defined when a relation is created.

The strategy for combining probability distributions must be given by the user since it depends on the application semantics. However, we can identify *at least* three common "library" strategies that may be provided by the system. Each of these is illustrated above on the right hand side. (The right hand side shows three possible resulting tuples, not a relation with three tuples.)

(a) *Average*. The probabilities for each value are added and normalized by the number of repeated tuples. This option makes sense in our example. The resulting tuple for k1 would contain the expected distribution if an observer were picked at random. (We may also have a weighted average where each observer is given a weight, but we do not discuss this option here.)

(b) *Sum*. Here we treat each attribute as an independent random variable and we compute the distribution of the sum of the random variables [May]. This may make sense if we are dealing with, say, deposited money. If with probability 0.2 one customer deposited 2 dollars, and with probability 0.3 the second customer deposited 3 dollars, we can expect 2+3 dollars with probability 0.2*0.3 = 0.06. The rest of the values are computed in a similar fashion. (If the attribute values are not numeric, i.e., cannot be added, a mapping function must be provided. See the DISCRETE operator later on in this section.)

(c) *Merge*. The distributions are assumed to be selections from the same underlying distribution. Thus, the distributions are simply merged. (An error occurs if the same value appears with differing probabilities in two tuples. An error also occurs if probability of the resulting relation is grater than 1.) The Merge option can be used to reconstruct

distributions that have been partially lost due to selects or other operations. Later on we will present an example to illustrate this.

In section III we discussed the problem of projecting out the key of a relation. Although we did not mention it there, we implicitly used an averaging combine to obtain the summary distribution. In other words, unless the default combine strategy is Average, the command for Example 3.1.3 should be:

$$\Pi_{A,B,C}(R);\ \textit{COMBINE Average}\,(A\_B),\ \textit{Average}\,(C)$$

(Note that we allow different combine options for each group.) The other combine options now give us the flexibility of computing our summary in different ways. For example, if attribute C of Example 3.1.3 represents the amount of usable grain (not spoiled) in a storage silo, then the summary will give, with the Sum option, the (distribution of the) total amount of usable grain in all silos.

STOCHASTIC is an operator that takes as input a deterministic relation and a probabilistic schema and returns a probabilistic relation based on that schema. The probabilistic schema describes what attributes comprise the KEY, and which of the dependent attributes are jointly distributed or independent. The probabilities for the new relation are given by the relative frequencies of the values in the original relation. Hence, one deterministic relation can generate many different probabilistic relations. EXAMPLE 4.3 demonstrates STOCHASTIC.

EXAMPLE 4.1 *STOCHASTIC*

Relation R

KEY	OBSERVER	A	B	C
k1	1	a1	b1	c1
k1	2	a1	b1	c1
k1	3	a1	b1	c2
k1	4	a2	b2	c2
k1	5	a2	b2	c3
k2	1	a3	b3	c3
k2	2	a3	b3	c3
k2	3	a3	b3	c3
k2	4	a3	b3	c3
k2	5	a1	b3	c3

STOCHASTIC(R;<KEY; A B; C>)

KEY	A B	C
k1	0.6 [a1 b1] 0.4 [a2 b2]	0.4 c1 0.4 c2 0.2 c3
k2	0.8 [a3 b3] 0.2 [a1 b3]	1.0 c3

For example, attributes A and B can be the QUALITY and BONUS for employees as suggested five managers (the observers). Since 3 out of 5 suggest a quality of a1 and a bonus of b1 for employee k1, the resulting relation states that the value [a1, b1] occurs with probability 0.6 for k1.

Operator DISCRETE goes the opposite direction as STOCHASTIC. DISCRETE takes as input a probabilistic relation and yields a deterministic relation as output. This operator is similar in spirit to [Gel], where the "expected value" result to a query is discussed. Here,

however, we compute an *expected value relation*. When the attribute values are not numeric (or are not atomic values), a mapping must be given to translate the values into numbers.

EXAMPLE 4.2 *DISCRETE*

Suppose we have a relation R for students and their grades in a course, as shown below on the left. The grades are entered in a typical A+, A, A-, B+,... fashion To compute each student's expected grade, we need to assign numeric values to the grades. Let F be the function that translates the grades as follows: A+ = 4.3, A = 4.0, A- = 3.7, B+ = 3.3, ... With F, the expected grades can be computed by DISCRETE as shown on the right.

Relation R

NAME	GRADE
John	0.7 A 0.2 B 0.1 C
Tom	0.5 A- 0.5 B-

DISCRETE(R; GRADE, F)

NAME	GRADE
John	3.6
Tom	3.2

Expected value computations:
For John: 0.7(4.0) + 0.2(3.0) + 0.1(2.0) = 3.6
For Tom: 0.5(3.7) + 0.5(2.7) = 3.2

There are other options for making relations discrete (e.g., choose the most likely grade) but we do not discuss them here due to space limitations.

Our last operator is GROUP. It fuses together two or more attribute groups in a relation into a single group. In doing this, it computes the joint probability distribution for the new group.

EXAMPLE 4.3 *GROUP*

Relation R

NAME	TYPE	COUNTRY
Maria	.6 Frigate .4 Tugboat	.7 USA .3 CAN

GROUP(R; TYPE COUNTRY)

NAME	TYPE COUNTRY
Maria	.6*.7 [Frigate USA] .6*.3 [Frigate CAN] .4*.7 [Tugboat USA] .4*.3 [Tugboat CAN]

To see why this operator is useful, let us assume that we wish to perform a join of relation R above with a DESCRIPTIONS relation that gives the speed of each ship, as in Example 3.3.1. However, suppose that the speed of a ship is determined by both its type and its country (e.g.,

Canadian Frigates may be faster than USA Frigates). Before we can join R with DESCRIPTIONS, we must GROUP together TYPE and COUNTRY as in the example. Then the group TYPE_COUNTRY is a foreign key of the DESCRIPTIONS relation and the join can be performed.

Notice that we have not defined an inverse to the GROUP operation. We expect that in most cases attributes within a group will not be independent, so breaking up the group will not be valid. However, if the attributes are independent, the inverse GROUP operation can be performed with projects and a cross product. In the above example, if we let S = GROUP(R; TYPE, COUNTRY), then R could be reconstructed as (using initials for the attributes):

$$\Pi_{N,T,C} (\Pi_{N,T}(S) >< \Pi_{N,C}(S))$$

EXAMPLE 4.4 *A More Substantial Example*

In closing this section we present an example that shows how several of our operators can be used together. In Example 3.3.1 we showed the natural join of two relations, SHIPS(N,T) >< DESCR(T,M). (N was the NAME, T the TYPE of ship, and M the MAXIMUM-SPEED.) We now show how the natural join can be performed via a cross product, in a manner analogous to doing a conventional join with a cross product (followed by select and project). The steps are as follows:

(1) $T1 \leftarrow SHIPS\ X\ DESCR$

(2) $T2 \leftarrow \sigma_{SHIPS.T:\ V\ =\ DESCR.T}(T1)$

(3) $T3 \leftarrow GROUP\ (T2,\ SHIPS.T,\ M)$

(4) $T4 \leftarrow \Pi_{N,\ SHIPS.T\_M}(T3);\ COMBINE\ Merge\ (SHIPS.T\_M)$

Unfortunately we do not have space to show all the intermediate results, but here is a brief description. The first step performs a cross product that, strictly speaking, is not valid because the two relations are not independent. However, the next step deletes the invalid probabilities, leaving us with correct but partial distributions. These distributions are merged in the last step. (In relations T2 and T3 there are two attributes with name T. To distinguish them, we use the notation R.T, where R is the original relation where T appeared.)

Note that we are not advocating cross products between dependent relations here. We are only illustrating that our commands (without the natural join) are powerful enough to perform a join. If a user needs to do a join, he should of course use the operator that is provided.

V. The Uniform Distribution

Our model permits the use of some well known probability distribution functions as an alternative to explicitly enumerating the probabilities. Here we focus exclusively on the *uniform distribution*; however, the same principle can be used for other distributions such as the binomial or geometric.

A uniform distribution is specified with the notation

n U[range].

The value n is the total probability that is to be uniformly distributed over all values in the range. The range can be specified as a list of values (e.g., v1, v2, v3), or if the values in the domain have a sequence, as a starting and ending value (e.g., 12..17). No value in a range

should appear with an explicit probability or in another range within the same distribution. We also define the range notation :: to represent all possible domain values that do not appear elsewhere in the distribution.

EXAMPLE 5.1 *Use of the :: Notation*
Let the domain of attribute A be {a1, a2, a3, a4, a5, a6}. Then

KEY	A
k	0.3 a1 0.5 a2 0.2 U[::]

is equivalent to . . .

KEY	A
k	0.30 a1 0.50 a2 0.05 a3 0.05 a4 0.05 a5 0.05 a6

EXAMPLE 5.2 *Ranges with Attribute Groups*
Let the domain of attribute A be {a1, a2} and of B {b1, b2, b3}.

KEY	A B
k	0.3 U[a1,a2 b1] 0.5 U[a1 ::] 0.2 U[:: ::]

is equivalent to . . .

KEY	A B
k	0.15 [a1 b1] 0.15 [a2 b1] 0.25 [a1 b2] 0.25 [a1 b3] 0.10 [a2 b2] 0.10 [a2 b3]

Note that the order of the ranges in the left hand side distribution is important. For example, switching the 0.5 U[a1 ::] and the 0.2 U[:: ::] entries would create an error, i.e., 0.2 U[:: ::] would assign probabilities to all remaining values, and 0.5 U[a1 ::] could not reassign them.

The semantics of all of our relational operators are unchanged by the inclusion of the uniform distribution. In other words, the result of a relational operator should be identical regardless of whether a distribution is explicitly enumerated or given via the uniform distribution.

However, in an actual implementation there are two issues to consider. First, it may be unnecessary to translate the uniform distributions to explicit probabilities for performing computations. A good system should be able to perform the computations directly with the uniform distributions. Second, it is probably convenient to represent uniform distributions in a result with the uniform notation "U", as opposed to listing all probabilities explicitly. The next two examples illustrate these points.

EXAMPLE 5.3 *PROJECT onto KEY, A, B with Uniform Distribution.*

Consider the operation $\Pi_{KEY,A,B}(R)$ where R is the relation:

KEY	A B C
k	0.2 [a1 b1 c1] 0.3 [a2 b2 c2] 0.5 U[:: :: ::]

and the domains are:

Domain of A is { a1, a2, a3, a4 }
Domain of B is { b1, b2, b3 }
Domain of C is { c1, c2 }

STEP 1: There are 4*3*2 = 24 possible A_B_C values, two of which are explicitly listed: [a1, b1, c1] and [a2, b2, c2]. This means that the remaining 24 - 2 values each occur with probability 0.5/22 = 0.023.

STEP 2: When the project is done, two of the uniformly distributed values, [a1, b1, c2] and [a2, b2, c1], will become identical to the explicitly enumerated values. Thus,

$$P(A\_B = [a1, b1] \mid KEY=k) = 0.200 + 0.023 = 0.223$$

$$P(A\_B = [a2, b2] \mid KEY=k) = 0.300 + 0.023 = 0.323$$

STEP 3: Determine what's left over to be uniformly distributed:

$$1 - (0.223 + 0.323) = 1 - 0.546 = 0.454$$

The resulting relation is:

KEY	A B
k	0.223 [a1 b1] 0.323 [a2 b2] 0.454 U[:: ::]

EXAMPLE 5.3 *Natural Join with Uniform Distribution*

Let the domains be:

Domain of A is { a1, a2, a3 }
Domain of B is { b1, b2, b3, b4 }

Relation P

K	A
k	0.8 a1 0.2 U[::]

Relation Q

A	B
a1	0.6 b1 0.4 b2
a2	1.0 U[::]

Relation P >< Q

K	A B
k	0.48 [a1 b1] 0.32 [a1 b2] 0.10 U[a2 ::]

The 0.2 U[::] in P assigns 0.1 probability to a2 and a3. The a3 value is not joined since there is no a3 in Q; thus, the result has 0.1 missing probability. The a2 value can be joined, yielding the 0.10 U[a2 ::] entry in P >< Q. Note that since the A value is known (= a2), the :: notation is only used for the B component.

VI. Conclusions

We have presented a probabilistic relational data model where tuples have deterministic keys and both probabilistic and deterministic attribute groups. The probability distribution function for a probabilistic group can be enumerated explicitly or given via a pre-defined distribution like the uniform one.

We believe that the major strengths of our model are its generality and naturalness. As we argued in Section I, our model encompasses most existing incomplete information models. The model leads to, in a very natural way, to some powerful new operators like Join, Combine, and Stochastic. As far as we can tell, these operators have no counterpart in other existing models.

Another important strength, in our opinion, is the concept of missing probability. Not only does this concept make it easier to input probabilistic data, but it also gives us more flexibility in computing selects, joins, and other operations.

In this paper we have relied heavily on examples for presenting our ideas. Given the number of ideas and issues to be discussed, we felt this was the most effective approach, both in terms of the readers time and the space available. But this of course means that we have not presented as many details as we would have liked to. The missing details range from the algorithms for computing probabilities under each operator, to an analysis of the complexity of such algorithms, to additional options available for each operator, to even additional operators and commands. Just to pick one example, we did not even present commands for inputting, updating, and deleting data in relations. However, it is our belief that from the basic ideas illustrated here, it is possible to derive most of the missing details relatively easily. (Some of the missing details can be found in [Gar].)

We also believe that our model and language raise some interesting and challenging future research issues. One is the use of continuous attributes and probability density functions. How are these attributes represented and manipulated? A second issue is user defined distributions. In a given application, a particular probability distribution function may arise frequently, so it may be useful to refer to it by name (as we did for the uniform distribution).

How would these distributions be defined and invoked?

A third issue is attribute dependencies. In discussing the Cross Product operator, we stated that it was only valid for independent entities. How can a user tell if two entities are independent? It may be desirable to have the user enter the elementary dependencies and then have the system enforce them, analogous to how a conventional system enforces functional dependencies. (Attributes A and B are dependent if P[A and B] is not equal to P[A] times P[B].) However, note that dependencies are not transitive (A,B dependant and B,C dependant, does not imply that A,C are dependant), so it hard for the system to infer additional dependencies. However, if the system is given conventional functional dependencies, it may use these to infer probabilistic dependencies. (If A → B, then A,B are dependant.)

It also possible to define a new type of dependency, a probabilistic functional dependency, PFD, $A \xrightarrow{p} B$. If $A \xrightarrow{p} B$, then a given value of attribute A determines *the probability distribution* of the possible B values. In our ships and descriptions example (3.3.1), $NAME \xrightarrow{p} TYPE$ and $TYPE \xrightarrow{p} MAX\text{-}SPEED$. A PFD is weaker than a functional or multivalued dependency, but still $A \xrightarrow{p} B$ implies that A and B are dependent random variables. Since PFD are transitive, they can be useful in a probabilistic database system. For example, $NAME \xrightarrow{p} TYPE$ and $TYPE \xrightarrow{p} MAX\text{-}SPEED$ implies $NAME \xrightarrow{p} MAX\text{-}SPEED$, so we know that NAME and MAX-SPEED are dependent attributes.

VII. References

[Cav] Cavallo, R. & Pittarelli, M. *The Theory of Probabilistic Databases,* Proceedings of the 13th Conference on Very Large Databases, 1987.

[Dad] Dadam, P., Kuespert, K. et al, *A DBMS Prototype to Support Non-First Normal Form Relations: An Integrated View on Flat Tables & Hierarchies,* SIGMOD Proceedings, 1986.

[Dat] Date, C. *An Introduction to Database Systems Vol. 1,* Addison-Wesley, 4th Edition, 1986.

[Gar] Garcia-Molina, H. & Porter, D. *Some Thoughts on Probabilistic Databases,* Princeton Technical Report CS-TR-090-87, April 1986.

[Gel] Gelenbe, E. & Hebrail, G. *A Probability Model of Uncertainty in Databases,* Proceedings of the International Conference on Data Engineering, Feb. 1986

[Gho] Ghosh, S. *Statistical Relational Tables for Statistical Database Management,* IEEE Transactions on Software Engineering, Dec. 1986.

[Imi] Imielinski, T. *Query Processing in Deductive Databases with Incomplete Information,* Rutgers Technical Report DCS-TR-177, Mar. 1986.

[Imi2] Imielinski, T. *Automated Deduction in Databases with Incomplete Information,* Rutgers Technical Report DCS-TR-181, Mar. 1986.

[Imi3] Imielinski, T. & Lipski, W. *Incomplete Information in Relational Databases,* Journal of the ACM, Vol. 31, No. 4, Oct. 1984.

[Lip] Lipski, W. *On Semantic Issues Connected with Incomplete Information Databases,* ACM Transactions on Database Systems, Vol. 4, No. 3, Sep. 1979.

[Lip2] Lipski, W. *On the Logic of Incomplete Information,* Proceedings of the 6th International Symposium of Mathematical Foundations of CS, Sep. 1977.

[Liu] Liu, Ken-Chih & Sunderraman, R. *On Representing Indefinite & Maybe Information in Relational Databases,* Proceedings of the 4th International Conference on Data Engineering, Feb. 1988.

[Men] Mendelson, H. & Saharia, A. *Incomplete Information Costs & Database Design,* ACM Transactions on Database Systems, Vol. 11, June 1986.

[Mey] Meyer, P. *Introductory Probability & Statistical Applications,* Addison-Wesley, 2nd Edition, 1970.

[Mor] Morrissey, J. & van Rijsbergen, C. *A Formal Treatment of Missing & Imprecise Information,* Proceedings of SIGIR, 1987.

[Rei] Reiter, R. *A Sound & Sometimes Complete Query Evaluation Algorithm for Relational Databases with Null Values,* Journal of the ACM, Vol. 33, No. 2, Apr. 1986.

[Rot] Roth, M. & Korth, H. & Silberschatz, A. *Extended Algebra & Calculus for Non-1NF Relational Databases,* University of Texas at Austin Technical Report TR-84-36, Jan. 1985.

[Rot2] Roth, M. & Korth, H. & Silberschatz, A. *Null Values in Non-1NF Relational Databases,* University of Texas at Austin Technical Report TR-85-32, Dec. 1985.

[Vas] Vassilou, Y. *Functional Dependencies & Incomplete Information,* Proceedings of the 6th International Conference on Very Large Databases, Montreal, Oct. 1980.

[Won] Wong, E. *A Statistical Approach to Incomplete Information in Database Systems,* ACM Transactions on Database Systems, Vol. 7, No. 3, Sep. 1982.

APPENDIX

In this appendix, we present a syntactic summary of the Data Definition Language (DDL) and the Data Manipulation Language (DML) of our *probabilistic database managment system* defined. The semantics of the languages are explained in the main body of the paper. The summary format is a pair of simplified *Context Free Grammars,* one for the DDL and one for the DML. Bold strings are terminals. All others are variables. The first production left hand side is the special start symbol.

1. Data Definition Language

Reln-Schema →	Base-Reln Schema
Schema →	< Key > | < Key : Groups : Option-Part >
Key →	A Key | A
Groups →	Group ; Groups | Group
Group →	A Group | A
Base-Reln →	"the possible base relation names; syntax not given here"
A →	"the possible attribute names; syntax not given here"
	(Option-Part represents the default Combine strategy for the relation. It is defined below.)

2. Data Manipulation Language

Relation →	Base-Reln | Reln-Expression | (Relation)
Base-Reln →	"the possible base relation names; syntax not given here"
Reln-Expression →	Reln-Part ; Option-Part | Reln-Part
Reln-Part →	Project-Expression | Select-Expression | Binary-Expression | Stochastic-Expression | Discrete-Expression | Group-Expression
Option-Part →	**Combine** Option
Option →	Simple-Option | Simple-Option , Option
Simple-Option →	**Average** (US-Group) | **Summary** (US-Group) | **Merge** (US-Group)
Project-Expression →	Π_Q (Relation) | Π_Q (Relation . US-Group)
Q →	A | A, Q
US-Group →	A | A_ US-Group
A →	"the possible attribute names; syntax not given here"
Select-Expression →	σ_{Pred} (Relation) | σ_{Pred} (Relation . US-Group)
Pred →	US-Group: Rest | US-Group Operator [Vector]
Rest →	**V** Operator [Vector] , Prob Operator Prob-val | **V** Operator [Vector] | Prob Operator Prob-val
Operator →	< | > | ≤ | ≥ | ≠ | =
Vector →	Val | Val , Vector
Val →	* | Attrib-Val
Prob →	**P** | **MP**
Prob-Val →	"the possible probability values; syntax not given here"

Attrib-Val →	"the possible attribute values; syntax not given here"
Binary-Expression →	Relation Binary Relation
Binary →	>< | **X** | ∪ | ∩ | -
Stochastic-Expression →	**Stochastic** (Relation ; Schema)
Discrete-Expression →	**Discrete** (Relation ; Group , Function)
Function →	"the possible mapping functions; syntax not given here"
Group-Expression →	**Group** (Relation ; Group)