ProbView: A Flexible Probabilistic Database System

LAKS V. S. LAKSHMANAN
Concordia University
NICOLA LEONE
Technical University of Vienna
ROBERT ROSS and V. S. SUBRAHMANIAN
University of Maryland

Probability theory is mathematically the best understood paradigm for modeling and manipulating uncertain information. Probabilities of complex events can be computed from those of basic events on which they depend, using any of a number of *strategies*. Which strategy is appropriate depends very much on the known interdependencies among the events involved. Previous work on probabilistic databases has assumed a *fixed* and *restrictive* combination strategy (e.g., assuming all events are pairwise independent). In this article, we characterize, using postulates, whole classes of strategies for conjunction, disjunction, and negation, meaningful from the viewpoint of probability theory. (1) We propose a probabilistic relational data model and a *generic* probabilistic relational algebra that neatly captures *various strategies* satisfying the postulates, within a *single unified framework*. (2) We show that as long as the chosen strategies can be computed in polynomial time, queries in the positive fragment of the probabilistic relational algebra have essentially the same data complexity as classical relational algebra. (3) We establish various containments and equivalences between algebraic expressions, similar in spirit to those in classical algebra. (4) We develop algorithms for maintaining materialized probabilistic views. (5) Based on these ideas, we have developed

Lakshmanan's research was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada and the Fonds Pour Formation De Chercheurs Et L'Aide À La Recherche of Quebec. Leone's research was supported in part by Christian Doppler Lab for Expert Systems, by Instituto per la Sistemistica e l'Informatica (ISI-CNR), and by FWF project P11580-MAT. Ross and Subrahmanian's research was supported in part by the Army Research Office under grants DAAH-04-95-10174 and DAAH-04-96-10297, by the Air Force Office of Scientific Research under grant F49620-93-1-0065, by ARPA/Rome Labs contract F4630602-93-C-0241 (Order A716), by an NSF Young Investigator award IRI-93-57756, and by the Army Research Laboratory under Cooperative Agreement DAAL01-96-2-0002 Federated Laboratory ATIRP Consortium.
Authors' addresses: L. V. S. Lakshmanan, Department of Computer Science, Concordia University, Montreal, Quebec H3G 1M8, Canada; email: ⟨laks@cs.concordia.ca⟩; N. Leone, Information Systems Department, Technical University of Vienna, Paniglgasse 16, A-1040 Vienna, Austria; email: ⟨leone@dbai.tuwien.ac.at⟩; R. Ross and V. S. Subrahmanian, Department of Computer Science, University of Maryland, College Park, MD 20742; email: ⟨robross@cs.umd.edu⟩; ⟨vs@cs.umd.edu⟩.

a prototype probabilistic database system called **ProbView** on top of Dbase V.0. We validate our complexity results with experiments and show that rewriting certain types of queries to other equivalent forms often yields substantial savings.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design—*data models*; H.2.3 [**Database Management**]: Languages—*query languages*; H.2.4 [**Database Management**]: Systems

General Terms: Algorithms, Languages, Performance, Theory

Additional Key Words and Phrases: Algebra, data complexity, performance evaluation, probabilistic databases, view maintenance, query optimization, view maintenance

1. INTRODUCTION

Although it is now four years since Silberschatz et al. [1991, pp. 116–117] stated that one of the important unsolved problems in databases is the ability to smoothly and efficiently integrate models of uncertainty, progress in this area has been surprisingly slow. Uncertainties occur in relational databases in many ways, four of which we list in the following four sections (1.1–1.4).

1.1 Uncertainty in Image Databases

In most image databases, image processing (IP) algorithms process surveillance images and feed the results into a relational database. Consider, for instance, a face database. In this case, given an image `im1.gif`, IP algorithms do two things: they attempt to locate faces in the image file `im1.gif`—this process is called *segmentation*—and they attempt to *match* the faces segmented in the previous step to images in a mugshot database. The result is placed in a relation:

`face`*(Filename,LeftCorner_X,LeftCorner_Y,RtCorner_X,RtCorner_Y,Person_Prob).*

Tuples in this relation are allowed to have multiple person-probability *triples* in the last argument. Three example tuples, t_p^1, t_p^2 and t_p^3 are found in Table I.

Tuple t_p^1 says that a face occurs in the image file `im1.gif` and that the rectangular box having the bottom-left corner at (5, 10) and top right corner at (35, 40) contains a face (these coordinates are relative to the bottom-left corner of the image file `im1.gif` according to a fixed pixel numbering scheme); the person whose face appears in this box is John (with 20–25% probability), Jim (with 35–40% probability), or Tom (with 40–45% probability).

The reason probabilities are not stated as points but rather as intervals is because they are computed as follows: the image processing package assigns a point probability p to the faces it identifies, but there is also a degree of error e possible in the image processing package itself; thus the

Table I. Three example tuples

t_p^1	`im1.gif`	5	10	35	40	john	0.2	0.25
						jim	0.35	0.4
						tom	0.4	0.45
t_p^2	`im1.gif`	35	40	60	40	john	0.6	0.65
						jim	0.2	0.25
						ed	0.1	0.15
t_p^3	`im2.gif`	10	10	25	25	john	0.3	0.35
						ed	0.6	0.65

actual probability lies within the interval [max(0, $p - e$), min(1, $p + e$)]. In the preceding example, assuming $e = 0.025$ (i.e., 2.5%), this means that the face recognition package said the face in the specified rectangle is John with 22.5% certainty. In addition, as stated in Silberschatz et al. [1991, p. 117], satellite image databases may contain uncertain data describing image features.

1.2 Sensor Data

In addition, there are a vast number of applications where analogue sensor data are digitized, and the resulting digitized information needs to be stored in a database. However, this task requires several steps. First, multiple sensors are often used to study the same phenomenon and the (analogue) results from these sensors are *fused* to provide an estimate of the probability p_O that the sensors have detected object *Obj*. For example, thermal sensors may be used to detect underground nuclear activity, airborne early warning (AWACS) systems are used by military agencies to monitor enemy air-defenses, and analogue image data may be digitized and subject to further analysis to pinpoint camouflaged enemy offensive and defensive missile sites. In all these cases, sensor data are processed as follows: analogue data from multiple sources are "fused" (the field is called *sensor fusion*, cf. Iyengar et al. [1995]), and the resulting fusion leads to a number of possible alternatives, each of which has an associated probability. For example, two alternatives may be: "The radar system used at Basra location (X_1, Y_1) is 999 KHz wideband radar," and a second possibility is "The radar system used at Basra location (X_1, Y_1) is 750 KHz wideband radar." The probability associated with these alternatives is the probability predicated by the sensors. However, in many cases, especially in cases concerning sensor data, the reliability of the sensors is often well studied as a result of exhaustive testing and validation procedures. Consequently, each of the preceding alternatives must be "adjusted" to reflect the margin of error e. Thus, if sensors predict alternative A is true with probability p, then the actual estimate is adjusted to be [$max(0, p - e)$, $min(1, p + e)$].

1.3 Temporal Indeterminacy

Dyreson and Snodgrass [1993] have noted that in many cases, when a tuple in a relation is time-stamped (this is called *valid-time*), the "stamp" is in fact a temporal interval *TI*. For example, if a tuple *t* in such a relation is time-stamped with the interval TI_0, then this is interpreted as "tuple *t* holds at some point in interval TI_0." For instance, suppose we have a tuple saying that "John Smith became Chairman of ABC Corp. sometime in Jan. 1996." Also, we may know that no new appointments start on weekends. Thus, given any weekday *d* in Jan. 1996, there is a probability of 1/23 that John Smith became (i.e., started as) Chairman on that day. Also, if there are *k* weekdays in January before and including day *d*, then the probability that John Smith was playing the role of Chairman on that day is *k*/23. Dyreson and Snodgrass study the handling of probabilistic temporal indeterminacy, assuming event independence.

1.4 Information Retrieval

The information retrieval community has long used probabilistic techniques for retrieval of document data based on "concepts." As database techniques expand to include document databases, we expect to see relational databases being extended to include relations with schemas such as (`DocId, Concept, Prob`) saying that a given document addresses a given topic with probability *p*. This is the basis of the well-known technique of latent semantic indexing [Dumais 1993].

This provides practical justification for the use of interval probabilities. Later in this article, it is shown that in any case, point probabilities cannot be used *unless* independence assumptions are made.

In general, probabilistic reasoning is notoriously tricky. To see this, consider two events *A* and *B*. Examples of these events are given in the following. Suppose we know that $\textbf{Prob}(A) = 0.8$ and $\textbf{Prob}(B) = 0.7$. What can we say about the probability of the complex event $(A \wedge B)$?

(1) $\textbf{Prob}(A \wedge B)$ *could be* $\textbf{Prob}(A) \times \textbf{Prob}(B)$ *if* *A* and *B* are independent events. For example, suppose *A* is the event, "The face occurring in `im1.gif` in the rectangle with (5, 10) and (35, 40) as its lower-left corner and top-right corner, respectively, is John." Suppose *B* is the event, "The face occurring in `im2.gif` in the rectangle with (10, 10) and (25, 25) as its lower-left corner and top-right corner, respectively, is John." *In this case*, Events *A* and *B* are independent, so we would be justified in multiplying their probabilities in order to compute the probability of the compound event $(A \wedge B)$. In general, however, there could be dependencies in databases that sometimes invalidate this assumption.

(2) On the other hand, suppose *B* is the event: "The face occurring in `im1.gif` in the rectangle with (35, 40) and (60, 40) as its lower-left corner and top-right corner, respectively, is John." Suppose further, that these are surveillance photos, so two faces in one photograph cannot be the same person. *In this case*, *events A and B are NOT independent*; in fact, in this case, they are mutually exclusive, and the probability of the compound event $(A \wedge B)$ is 0.

(3) The previous examples show two completely different extremes—one where the events are independent, and another where they are mutually exclusive. However, the *reality of life is that in the vast majority of cases, we are ignorant of the precise dependencies between events A and B*. In such cases, **Prob**$(A \wedge B)$ may not even be uniquely expressible by a number. For example, as shown by Ng and Subrahmanian [1993, 1995] (based on results of Fenstad [1980], which are in turn derived from Boole [1854] many years ago), if **Prob**$(A) = \alpha$ and **Prob**$(B) = \beta$, then in the case of complete ignorance of events A, B, the best we can say about **Prob**$(A \wedge B)$ is that its probability is *in the interval* $[\min(1, \alpha + \beta - 1), \min(\alpha, \beta)]$.

(4) A fourth possibility is that events A and B are such that one implies the other; this is called *positive correlation* and it can be shown [Lakshmanan and Sadri 1994a] that in this case again, the probability **Prob**$(A \wedge B)$ is equal to $\min(\alpha, \beta)$.

(5) By now, we hope the reader is convinced that there are many other possible dependencies (e.g., negative correlation $-$ A implies $\neg B$, conditional correlation $-$ A implies B when $\langle$*condition*$\rangle$, etc.) between A and B that could lead to other expressions that accurately capture the desired probability of $(A \wedge B)$.

The reader will not be surprised to know that *exactly the same situation* occurs when we consider computing **Prob**$(A \vee B)$ from **Prob**(A) and **Prob**(B), or **Prob**$(\neg A)$ from **Prob**(A), respectively.

We refer to the different ways of computing the probabilities of compound events from those of basic events as *combination strategies* (associated with the connectives $\wedge$, $\vee$, $\neg$). As the preceding examples show, the effective incorporation of probabilities in databases requires the ability to *smoothly switch* from one probabilistic combination strategy to another, depending upon the relationship among the events involved. Past work on extending relational DBMSs to handle probabilities have lacked this flexibility and hence, their results hinge critically upon strong assumptions about the dependencies between events—a factor that cannot be ignored if a *truly useful* probabilistic relational data model is to be designed.

The key contributions of this article are as follows.

(1) First, although associating probabilities with individual domain elements (as in the `face` example of Section 1) is natural and is close to the way probabilistic data are derived from sensors or image processing algorithms, they are not convenient for algebraic manipulation. We show that probabilistic data obtained from appropriate sources using an *element level* interface, can be converted into a representation that (i) associates probabilities with whole tuples, making them amenable to algebraic manipulation, (ii) "faithfully" represents the original probabilistic information, and (iii) can be computed from the original information in polynomial time.

(2) We propose *postulates* that any probabilistic conjunction and disjunction strategy must satisfy, in order to be considered "reasonable." In addition, we present various example strategies that satisfy these postulates and show that they capture intuitions such as "probabilistic independence," "positive correlation," and "ignorance." We show that once a strategy for conjunction and a strategy for disjunction are picked, these two jointly induce a notion of difference.

(3) Our intention is that *each and every* strategy characterized by the postulates mentioned in item 2 will be available to the user of a probabilistic DBMS. We propose a *generic probabilistic relational algebra* that allows the user to *choose* any strategy appropriate for his or her application as long the postulates are satisfied. Consequently, our approach applies to a wide variety of probabilistic reasoning systems. In particular, it captures as special cases, the probabilistic frameworks of Barbara et al. [1992], Dubois and Prade [1988], and Cavallo and Pitarelli [1987]. The reason is that each of these frameworks uses a specific way of combining probabilities that applies only under certain assumptions.

(4) We prove various *query equivalence* and *containment* results that apply to any choice of probabilistic conjunction (resp., disjunction, negation) strategies satisfying our postulates.

(5) We show (Theorem 3.1) that as long as negation is absent, the *complexity of answering probabilistic queries* remains essentially the same as that for standard relational algebra queries, provided that the probabilistic strategies chosen can be computed in polynomial time. (This is a valid assumption, since all well-known strategies based on assumptions such as independence, positive correlation, negative correlation, and ignorance, can be computed in constant time.)

(6) We show that the problem of *view maintenance* raises some unique challenges in the case of probabilistic databases. We develop incremental algorithms for view maintenance involving insertion and deletion both with and without modification of probabilities.

(7) Based on these ideas, we have built a prototype probabilistic database system called **ProbView** that supports querying in probabilistic databases. **ProbView** is built on top of Dbase V.0 and runs on the PC/Windows platform. It implements all the strategies described in this article. Building the implementation "on top" of Dbase allows us to use all the indexing and optimization schemes already present in Dbase. In addition, we have used **ProbView** to experiment with the equivalences referred to in item 4 for computational efficiency. In particular, the experiments seem to indicate, more or less conclusively, that certain ways of rewriting queries always leads to substantial savings in time. We have a special operation called *compaction* which is essentially a generalization of duplicate elimination for probabilistic databases. Our experiments shed light on optimization opportunities that exist for this normally expensive operation.

For lack of space, we suppress the proofs of some of our theorems. The interested reader may consult the technical report [Lakshmanan et al. 1996], available online.

2. BASIC DEFINITIONS

We use $\mathcal{U}$ to denote the unit interval [0, 1] and $\mathcal{C}[0, 1]$ to denote the set of closed subintervals of the unit interval. Let $\mathcal{A}$ be a nonempty set of symbols called *attributes*. Associated with each attribute $A \in \mathcal{A}$, there is a nonempty set, $dom(A)$, of values, called the *domain* of A. A *relation scheme* is any subset of $\mathcal{A}$. Let $R = \{A_1, \ldots, A_n\} \subseteq \mathcal{A}$ be a relation scheme. In this article, we deal with three kinds of tuples over R: *data-tuples* (Definition 2.1) denoted $s, t, \ldots$, *probabilistic tuples* (Definition 2.2) denoted $t_p, s_p, \ldots$, and *annotated tuples* (Definition 2.4) denoted $t_a, s_a, \ldots$.

Definition 2.1 (*Data tuples and relations*). *A* data tuple *over the relation scheme $R = \{A_1, \ldots, A_n\}$ is any n-tuple $t = (a_1, \ldots, a_n)$, where each $a_i \in dom(A_i)$. A* data relation *over R is a finite set of data tuples over R.* (*Thus, a data relation is just a classical relation.*)

Our notion of a probabilistic tuple is a simple adaptation of the notion proposed by Barbara et al. [1992]. In their model, they make the *restrictive* assumption that the tuples in a probabilistic database are pairwise independent. When such restrictive assumptions are *not* made, however, as Fenstad [1980] has shown, the probabilities of compound events can only be evaluated to within certain bounds. This suggests working with probability *intervals* associated with tuples. Several works have followed this approach.[1] Accordingly, for an event e, we let $\mathbf{Prob}(e)$ denote the *probability interval* associated with that event.

Definition 2.2 (*Probabilistic tuples and relations*). *A* probabilistic tuple *over the relation scheme $R = \{A_1, \ldots, A_n\}$ is an n-tuple $(\mathbf{v_1}, \ldots, \mathbf{v_n})$, where each $\mathbf{v_i}$ is of the form $\langle V_i, h_i \rangle$,* with *$V_i \subseteq dom(A_i)$ a finite set of values from A_i's domain, and h_i: $V_i \rightarrow \mathcal{C}[0, 1]$ a function that maps each value in V_i to a probability range. A probabilistic relation over the scheme R is a finite set of probabilistic tuples over R. A probabilistic database is a finite set of probabilistic relations with associated schemes.*

Consider a probabilistic tuple $t_p = ((V_1, h_1), \ldots, (V_n, h_n))$. The events associated with this tuple are equalities of the form $t_p.A_i = c$, where A_i is one of the attributes and $c \in V_i$. Such an event says that the value of the tuple corresponding to attribute A_i is c. Indeed, the preceding probabilistic tuple t_p says that:

$$\mathbf{Prob}(t_p.A_i = c) = \begin{cases} h_i(c) & \text{if } c \in V_i \\ [0,0] & \text{otherwise(i.e., if } c \in dom(A_i) - V_i). \end{cases}$$

[1]See, for example, Ng and Subrahmanian [1993, 1995], Lakshmanan and Sadri [1994b, 1994a], Lakshmanan [1994].

Table II. target with three attributes.

LOC	OBJ	BAND
site1	radar_type1	{750, 800}
$h_1(site1) = [1, 1]$	$h_2(radar\_type1) = [1, 1]$	$h_3(750) = [0.4,0.7]$
		$h_3(800) = [0.5,0.9]$
site2	{radar_type1,radar_type2}	700
$h_4(site2) = [1, 1]$	$h_5(radar\_type1) = [0.8, 0.9]$	$h_6(700) = [1,1]$
	$h_5(radar\_type2) = [0.8, 0.3]$	
site3	{radar_type1,radar_type2}	{700, 750}
$h_7(site3) = [1, 1]$	$h_8(radar\_type1) = [0.4, 0.7]$	$h_9(700) = [0.6,0.9]$
	$h_8(radar\_type2) = [0.5, 0.6]$	$h_9(750) = [0,0.4]$

It is now easy to see that a conventional (classical) tuple is a probabilistic tuple where V_i is always required to be a singleton, say $\{d_i\}$, and $h_i(d_i) = [1, 1]$, $i = 1, \ldots, n$. Notice that the definition of probabilistic tuples allows for any combination of deterministic and probabilistic domains, in the sense of Barbara et al. [1992].

Suppose the available knowledge about a relationship $\hat{r}$ in some domain is incomplete and uncertain. A clean and concise way to model this knowledge is as a probabilistic relation, say r. We henceforth refer to $\hat{r}$ as the (unknown) *underlying (classical) data relation* modeled by r, or simply as the *underlying relation* associated with r.

Example 2.1 Table II gives a simple example of a probabilistic relation called target with three attributes LOC, OBJ, and BAND. In the table, we have explicitly shown the functions h for all attributes. A more concise notation can be obtained by adopting the convention that whenever $t_p.A = (V, h)$ is such that $V = \{v\}$ is a singleton, and $h(v) = [1, 1]$, we simply denote $t_p.A = v$. For example, the fact that the location in the second tuple is "site2," can be simply represented by entering the constant "site2" in that cell.

Let r be a probabilistic relation and let $\hat{r}$ be the underlying relation associated with r. Suppose $t_p = ((V_1, h_1), \ldots, (V_n, h_n)) \in r$. Then this tuple says *at most one* of the data tuples in $V_1 \times \cdots \times V_n$ belongs to $\hat{r}$. We can regard each of the data tuples in $V_1 \times \cdots \times V_n$ as a *world* and so, any of these data tuples is a *possible* world. A *tuple interpretation* can be viewed as an assignment of probabilities to the various worlds associated with a tuple. This intuition is formalized in the next definition.

Definition 2.3 (*Tuple worlds and interpretations*). Suppose $R = \{A_1, \ldots, A_n\}$ is a relation scheme and r is a probabilistic relation over scheme R. We can associate worlds and interpretations with each probabilistic tuple in r as follows. Let $t_p = ((V_1, h_1), \ldots, (V_n, h_n))$ be any probabilistic tuple in r.

—A t_p*-world* is any member w of $V_1 \times V_2 \times \cdots \times V_n$. Intuitively, a t_p-world is a data tuple that represents a possible "state" of the probabilistic tuple t_p. We let $\mathbf{W}(t_p)$ denote the set of all worlds associated with t_p.

—A t_p-*interpretation* is a map, $\wp_{t_p}:\mathbf{W}(t_p) \rightarrow [0, 1]$ such that $\Sigma_{w \in \mathbf{W}(t_p)} (\wp_{t_p}(w)) \leq 1$. Notice that when $\Sigma_{w \in \mathbf{W}(t_p)} (\wp_{t_p}(w)) < 1$, $\wp_{t_p}$ admits the possibility that *none* of the t_p-worlds is true with respect to the underlying relation $\hat{r}$ modeled by r. When the probabilistic tuple t_p is clear from context we may drop the subscript t_p in $\wp_{t_p}$.

—$\wp_{t_p}$ *satisfies* tuple t_p iff for all $1 \leq i \leq n$, and all $v \in V_i$,

$$\left(\sum_{w \in \mathbf{W}(t_p) \& w.A_i = v} \wp t_p(w) \right) \in h_i(v).$$

—An *interpretation* for a probabilistic relation r is a map I that associates, with each probabilistic tuple $t_p \in r$, a t_p-interpretation, $I(t_p)$, such that for any two tuples t_p^1, $t_p^2 \in r$, and for each $w \in \mathbf{W}(t_p^1) \cap \mathbf{W}(t_p^2)$, $I(t_p^1)(w) = I(t_p^2)(w)$.

—I *satisfies* a probabilistic tuple t_p provided $I(t_p)$ satisfies t_p.

—I *satisfies* a probabilistic relation r iff for every probabilistic tuple $t_p \in r$, I satisfies t_p.

Although the meaning of the classical relations is self-evident, the semantics of probabilistic relations is not straightforward. The inherent meaning behind such relations is captured by the preceding definition.

Example 2.2 Let us return to Example 2.1. Let us refer to the three probabilistic tuples[2] there as t_1, t_2, t_3, respectively. There are two t_1-worlds (w_1, w_2), two t_2-worlds (w_3, w_4), and four t_3-worlds ($w_5, \ldots, w_8$):

w_1 = (site1, radar_type1, 750). w_5 = (site3, radar_type1, 700).
w_2 = (site1, radar_type1, 800). w_6 = (site3, radar_type1, 750).
w_3 = (site2, radar_type1, 700). w_7 = (site3, radar_type2, 700).
w_4 = (site2, radar_type2, 700). w_8 = (site3, radar_type2, 750).

Consider the following two t_1-interpretations $\wp^1$, $\wp^2$,

$$\wp^1(w_1) = 0.5; \qquad \wp^1(w_2) = 0.5; \qquad \wp^2(w_1) = 0.7; \qquad \wp^2(w_2) = 0.3.$$

$\wp^1$ satisfies tuple t_1, but $\wp^2$ does not because

$$\left(\sum_{w_j.\mathrm{BAND}=800} \wp^2(w_j) \right) = 0.3 \notin h_3(800) = [0.5, 0.9].$$

Example 2.3. A slightly more complex situation occurs when we consider the probabilistic tuple t_3. Recall the four t_3-worlds $w_5, \ldots, w_8$ from

[2]Strictly speaking, we should use a notation such as $t_{p1}, \ldots$. We abuse the notation here for simplicity.

Example 2.2. Consider the t_3-interpretation $\wp$ defined as:

$$\wp(w_5) = 0.35; \qquad \wp(w_6) = 0.15; \qquad \wp(w_7) = 0.35; \qquad \wp(w_8) = 0.15.$$

$\wp$ satisfies tuple t_3. To see this, consider first the case when we look at the LOC field of t_3. The only value is site3, which thus appears in all t_3-worlds $w_5, \ldots, w_8$. The probability interval $h_7(site3)$ of site3 is [1, 1], and the criterion

$$\wp(w_5) + \wp(w_6) + \wp(w_7) + \wp(w_8) = 0.35 + 0.15 + 0.35 + 0.15 = 1 \in [1,1]$$

is satisfied. Consider now the case when we look at the OBJ field of t_3 and consider the value radar_type1. The criterion

$$\wp(w_5) + \wp(w_6) = 0.35 + 0.15 = 0.5 \in [0.4,0.7]$$

is satisfied. Similarly, consider the same field and the value radar_type2. In this case, the criterion

$$\wp(w_7) + \wp(w_8) = 0.35 + 0.15 = 0.5 \in [0.5,0.6]$$

is also satisfied. When we consider the field BAND and the value 700, we see that

$$\wp(w_5) + \wp(w_7) = 0.35 + 0.35 = 0.7 \in [0.7,0.9].$$

The final case arises when we wish to check the field BAND and the value 750. Here we need to ensure that

$$\wp(w_6) + \wp(w_8) = 0.15 + 0.15 = 0.3 \in [0,0.4],$$

which happens to be true.

Unlike classical relations, probabilistic relations can be inconsistent. For example, the tuple $(\{a, b\}, h)$ with $h(a) = [1, 1]$ and $h(b) = [1, 1]$ is inconsistent, due to incorrect probability assignment. A more subtle scenario is suggested by the following example. Consider the tuples $(\{a, b\}, h)$, with $h(a) = [0.1, 0.2]$ and $h(b) = [0.6, 0.7]$, and $(\{a, b\}, h')$, with $h'(a) = [0.3, 0.4]$ and $h'(b) = [0.4, 0.5]$. The same worlds $w_1 = (a)$, $w_2 = (b)$ are associated with both probabilistic tuples. Intuitively, the first tuple constrains the probability of w_1 to be in the range [0.1, 0.2] which is disjoint with the range [0.3, 0.4] imposed on w_1 by the second tuple. A similar remark applies w.r.t. w_2. This shows it is improbable that either of the worlds w_1, w_2 holds in reality.

Testing consistency of probabilistic relations can be done efficiently, as established by the following theorem.

THEOREM 2.1 (Complexity of consistency). *Let $R = \{A_1, \ldots, A_n\}$ be a relation scheme. Checking whether a given probabilistic relation r over R is consistent can be done in polynomial time.*

PROOF. A probabilistic relation is consistent iff the set of constraints imposed by the various tuples on the tuple-worlds is consistent. The idea is then to construct the linear program associated with the probabilistic relation and solve it. Define the size of r as the total number of symbol occurrences in r. The linear program $\mathsf{LP}(r)$ associated with the relation r can be constructed in time polynomial in the size of r, and the size of $\mathsf{LP}(r)$ is also polynomially bounded in the size of r. $\mathsf{LP}(r)$ is generated as follows. The set of worlds associated with the whole relation is $\mathbf{W}(r) = \cup_{t_p \in r} \mathbf{W}(t_p)$. *Note, in particular, that if two different tuples t_p^1, t_p^2 have a common world, then that world appears only once in* $\mathbf{W}(r)$. With each world $w \in \mathbf{W}(r)$, associate a linear programming variable z_w (which ranges over the real numbers). For each tuple $t_p \in r$, we have the following set of constraints.

Let $t_p = ((V_1, h_1), \ldots, (V_n, h_n))$.

1. For all $1 \leq i \leq n$ and for all $v \in V_i$, $\mathsf{LP}(r)$ contains the constraint:

$$\mathsf{lb}(h_i(v)) \leq \left(\sum_{w \in \mathbf{W}(t_p) \& w.A_i = v} z_w \right) \leq \mathsf{ub}(h_i(v)),$$

   where $h_i(v) = [\mathsf{lb}(h_i(v)), \mathsf{ub}(h_i(v))]$. *This constraint is said to be induced by v and V_i.*
2. For each $w \in \mathbf{W}(t_p)$, the constraint

$$0 \leq z_w \leq 1$$

   is present in $\mathsf{LP}(r)$.

We refer to the set of constraints corresponding to a tuple $t_p \in r$, as $\mathsf{LP}(t_p)$. Finally, add the constraint $(\Sigma_{w \in \mathbf{W}(r)} z_w) \leq 1$ to $\mathsf{LP}(r)$.

Observe that the total number of inequalities in $\mathsf{LP}(t_p)$ is $4 \times card(\mathbf{W}(t_p)) = 4 \times card(V_1 \times \cdots \times V_n)$, and the size of $\mathsf{LP}(t_p)$ is polynomial in the size of t_p. It follows that the size of $\mathsf{LP}(r)$ is also a polynomial in the size of r.

Now it is easy to see that there is a one-to-one correspondence between the admissible solutions of the linear program $\mathsf{LP}(r)$ and the interpretations of r that satisfy r. Therefore, r is consistent (i.e., there exists an interpretation I such that I satisfies r) if and only if the linear program $\mathsf{LP}(r)$ admits a (real) solution. Since linear programming is tractable (and the size of $\mathsf{LP}(r)$ is polynomial in the size of r), checking the consistency of r can be done in polynomial time as well.

Probabilistic relations are a convenient and expressive device for *modeling uncertainty and incompleteness*. However, probabilistic relations are not in "first normal form" and they are not directly amenable for algebraic manipulation. Also, it is unclear how a generic algebra not dependent on any specific probabilistic strategy can be defined based on probabilistic relations as defined in Definition 2.2. Thus, we need a *representation* for probabilistic tuples and hence relations and databases, that avoids these drawbacks. We propose a notion of *path-annotated* (or simply *annotated*) *tuples* to represent probabilistic tuples. Intuitively, given a data tuple $\vec{t}$, and probability bounds ℓ, u, a path p is a Boolean expression; $(\vec{t}, \ell, u, p)$ says that tuple $\vec{t}$ is in relation R with probability between ℓ and u (inclusive) if the Boolean condition encoded in p is satisfied. Basically, the Boolean conditions assume that each world has an associated *world id* (wid) w_i (which can easily be automatically generated by the database). A path in general is a Boolean expression over such world ids. The algebra defined in the next section manipulates annotated relations rather than the probabilistic relations that they represent. The idea is that probabilistic relations are only meant as an "interface" for "raw" probabilistic data coming from sources. Once the raw data are converted to the annotated representation, the database processes queries and presents answers to uses *only* in the form of annotated relations. Our operators manipulate data and probabilistic bounds, as well as paths, thus keeping track of the derivation history. *We henceforth assume any fixed standard enumeration* $w_1, w_2, \ldots$ *of all the worlds associated with all the tuples in a given probabilistic database.*

Definition 2.4 (*Paths and annotated tuples*). A *path* is a Boolean expression over wids. For a given database D, we use $\mathcal{P}_D$ to denote the set of all distinct paths over D. When the database is clear from the context, we drop the subscript D. A *path-annotated tuple* (or just *annotated tuple*, for short) over $\{A_1, \ldots, A_n\}$ is an element of $\text{dom}(A_1) \times \cdots \times dom(A_n) \times \mathcal{U} \times \mathcal{U} \times \mathcal{P}$. The definition of annotated relations and databases is an obvious extension of the preceding notion. The relation scheme associated with an annotated relation contains all the attributes $A_1, \ldots, A_n$ and three special attributes LB, UB, and PATH, corresponding to the domains $\mathcal{U}$, $\mathcal{U}$, and $\mathcal{P}$.

A path-annotated tuple $(a_1, \ldots, a_\ell, u, p)$ is *consistent* provided $\ell \leq u$, and p is a consistent Boolean expression over world-ids. We only consider consistent tuples in the following, unless otherwise specified. Such a tuple intuitively says the probability that $(a_1, \ldots, a_n)$ belongs to the underlying classical relation lies in the range $[\ell, u]$, and the justification for this belief is the path p associated with the tuple. Two path-annotated tuples $(a_1, \ldots, a_\ell, u, p)$ and $(a'_1, \ldots, a'_\ell, u', p')$ are said to be *data-identical* iff for all $1 \leq i \leq n$, $a_i = a'_i$.

We first formalize the notion of an *annotated representation* of probabilistic relations and databases. We next show (Theorem 2.2) that annotated relations can be used to "faithfully" represent probabilistic relations. This has the advantage that we can start with probabilistic relations as a natural model of uncertain knowledge and convert them into their corresponding annotated representations, which can then be manipulated by our algebra (defined in the next section).

When probabilistic tuples are converted to path-annotated tuples, the resulting tuples only have wids in their path field. No complex Boolean expressions occur in the path fields. However, when views are defined and materialized, the annotated relations representing these views may contain complex Boolean expressions in their path field. The following definition specifies what it means for an interpretation to satisfy a path-annotated tuple. As interpretations contain no path information, paths do not influence the definition of satisfaction for now. Later, in Section 3, we see how paths play an important role in manipulating path-annotated relations.

Definition 2.5 Suppose $(a_1, \ldots, a_n, \ell, u, p)$ is a path-annotated tuple. Suppose t_p is any probabilistic tuple. A t_p-interpretation $\wp_{t_p}$ satisfies $(a_1, \ldots, a_n, \ell, u, p)$ iff:

(1) $(a_1, \ldots, a_n)$ is a data tuple associated with one of the t_p-worlds, with world id, say w, and
(2) $\ell \le \wp_{t_p}(w) \le u$.

A t_p-interpretation satisfies a set of annotated tuples exactly when it satisfies each tuple in the set.

For a given t_p-world $w = (a_1, \ldots, a_n)$, $\wp_{t_p}(w)$ gives the probability that the data-tuple $(a_1, \ldots, a_n)$ is in the underlying classical relation. We say $\wp_{t_p}$ satisfies the annotated tuple $t_a = (a_1, \ldots, a_n, \ell, u, p)$ exactly when $\wp_{t_p}(w) \in [\ell, u]$. The next definition formalizes the notion of annotated representations of probabilistic tuples.

Definition 2.6 Let $t_p = ((V_1, h_1), \ldots, (V_n, h_n))$ be a probabilistic tuple and S be a set of annotated tuples such that (1) the projection of S on its data attributes coincides with $\mathbf{W}(t_p)$ and (2) if $(a_1, \ldots, a_n, \ell, u, p) \in S$, then p is the wid of the world $(a_1, \ldots, a_n)$. In such a case, we say that S represents t_p provided every t_p-interpretation satisfying t_p also satisfies S.

In general, more than one set of annotated tuples can represent a probabilistic tuple. For example, consider the probabilistic tuple t_1 in Example 2.1. The sets

$$S_1 = \{(site1, radar\_type1, 750, 0, 1, w_1), (site1, radar\_type1, 800, 0, 1, w_2)\} \text{ and}$$

$$S_2 = \{(site1, radar\_type1, 750, 0.4, 0.7, w_1), (site1, radar\_type1, 800, 0.5, 0.9, w_2)\}$$

both represent t_1. However, notice that the probability intervals in S_2 are *sharper* than those in S_1 indicating that S_2 is a *more accurate* representation of t_1. Indeed, every probabilistic tuple $t_p = ((V_1, h_1), \ldots, (V_n, h_n))$ has a trivial representation $S_0 = \{(a_1, \ldots, a_n, 0, 1, w) | (a_1, \ldots, a_n) \in V_1 \times \cdots V_n$ is a t_p-world with wid $w\}$. Let S_1, S_2 be any two representations of a tuple t_p. We say that S_1 is *no less accurate than* S_2, $S_1 \ge S_2$, provided for every t_p-world $(a_1, \ldots, a_n)$ with wid w, S_i contains the annotated tuple $(a_1, \ldots, a_n, \ell_i, u_i, w)$, $i = 1, 2$ with $[\ell_1, u_1] \subseteq [\ell_2, u_2]$. Intuitively, this says that for each t_p-world, S_1 associates a probability interval sharper than that associated by S_2. The following theorem shows that there is a maximally accurate representation for every probabilistic tuple and establishes its properties.

THEOREM 2.2 (Representation theorem). *Every consistent probabilistic tuple has a unique maximally accurate representation. More precisely, if t_p is a probabilistic tuple, then there is a unique set of annotated tuples, denoted* $\mathsf{AS}(t_p)$*, satisfying the following properties:*

(1) $\mathsf{AS}(t_p)$ *represents* t_p.

(2) *For any set S of annotated tuples that represents* t_p, $\mathsf{AS}(t_p) \geq S$*; that is,* $\mathsf{AS}(t_p)$ *is the maximally accurate representation of* t_p.

PROOF. We can generate a linear program $LP(t_p)$ with the property that there is a one-to-one correspondence between the solutions to $LP(t_p)$ and tuple-interpretations satisfying t_p. It can then be shown that $\mathsf{AS}(t_p)$ can be generated by extremizing certain variables, and that every tuple-interpretation satisfying t_p necessarily satisfies $\mathsf{AS}(t_p)$. Uniqueness of $\mathsf{AS}(t_p)$ trivially follows from this argument. The construction of $\mathsf{AS}(t_p)$ is given in the following.

Given a probabilistic tuple $t_p = ((V_1, h_1), \ldots, (V_n, h_n))$, let $\mathsf{LP}(t_p)$ be the linear program constructed in the proof of Theorem 2.1. For each t_p-world $w \in \mathbf{W}(t_p)$, set

$$\ell_w = \textbf{minimize } z_w \textbf{ subject to } \mathsf{LP}(t_p).$$

$$u_w = \textbf{maximize } z_w \textbf{ subject to } \mathsf{LP}(t_p).$$

(Observe that ℓ_w and u_w are well-defined, as $\mathsf{LP}(t_p)$ always admits solutions under the hypothesis that probabilistic tuple t_p is consistent.)

Finally, set

$$\mathsf{AS}(t_p) = \{(a_1, \ldots, a_n, \ell_w, u_w, id(w)) | w \in \mathbf{W}(t_p) \& \ w = (a_1, \ldots, a_n) \text{ and } id(w) \text{ is the wid of } w\}.$$

It is not hard to show that $\mathsf{AS}(t_p)$ satisfies condition (1) in the theorem. We next prove that it also satisfies condition (2).

Now let S be any set of annotated tuples representing t_p. We show that $\mathsf{AS}(t_p) \geq S$. Suppose not. This implies there are annotated tuples $(\vec{a}, \ell_1, u_1, p) \in S$ and $(\vec{a}, \ell_2, u_2, p) \in \mathsf{AS}(t_p)$ such that $[\ell_2, u_2] \not\subseteq [\ell_1, u_1]$; that is, either $\ell_2 < \ell_1$ or $u_1 < u_2$. Consider the first case, as the other one is symmetric. For this case, let $\wp$ be any t_p-interpretation that satisfies t_p such that $\wp(\vec{a}) = \ell_2$. Such an interpretation must exist, because there is at least one solution of $\mathsf{LP}(t_p)$ in which the tuple-world $\vec{a}$ obtains the value ℓ_2 and because, as stated in the proof of Theorem 2.1, there is a one-to-one correspondence between solutions of $\mathsf{LP}(t_p)$ and interpretations that satisfy t_p. However, this interpretation $\wp$ does not satisfy the annotated tuple $(\vec{a}, \ell_1, u_1, p) \in S$ because $\ell_2 < \ell_1$; this contradicts the assumption that S represents t_p as $\wp$ satisfies t_p but not S. □

It follows from Theorem 2.2, by a simple extension, that every probabilistic relation and hence database has a unique maximally accurate annotated

Table III. Annotated representation of `target`

LOC	OBJ	BAND	LB	UB	PATH
site1	radar_type1	750	0.4	0.7	w_1
site1	radar_type1	800	0.5	0.9	w_2
site2	radar_type1	700	0.8	0.9	w_3
site2	radar_type2	700	0.08	0.3	w_4
site3	radar_type1	700	0	0.5	w_5
site3	radar_type1	750	0	0.4	w_6
site3	radar_type2	700	0.1	0.6	w_7
site3	radar_type2	750	0	0.4	w_8

representation satisfying the conditions in the theorem. Furthermore, the set of interpretations that satisfy the original probabilistic tuple t_p coincides with the set of interpretations that satisfy the set $\mathsf{AS}(t_p)$ of annotated tuples representing t_p. In other words, t_p and $\mathsf{AS}(t_p)$ are equivalent. We use the notation $\mathsf{AS}(r_p)$ to denote the annotated representation of a probabilistic relation r_p. In the following, we refer to this unique representation as *the annotated representation* of a probabilistic relation (database).

Example 2.4 The annotated representation of the probabilistic relation `target` of Example 2.1 is shown in Table III. In Table III the path attributes correspond to the world-ids of Example 2.1. The lower and upper bounds of the probabilities associated with worlds are computed using the linear programming approach sketched earlier.

Example 2.5 A slightly different probabilistic relation, called `thermal`, may be used to record thermal emissions classifications detected by thermal sensors. The tuples in Table IV are in the `thermal` relation, where a and b are two classifications of thermal emissions. The annotated representation of this relation is very simple and is shown in Table V. Later, we use the relations `thermal` and `target` together in order to illustrate binary algebraic operations such as Cartesian product and Join.

The operations of our algebra are defined in the next section. In view of Theorem 2.2 and our remarks about the relative advantages of probabilistic and annotated relations, we only consider annotated relations in the following. Since annotated relations play a central role in our probabilistic model, it is worth pointing out both the size of annotated representations and the time needed to compute them.

THEOREM 2.3 (Complexity of annotated representations). *Let t_p be a consistent probabilistic tuple. Then:*

(1) *The size of* $\mathsf{AS}(t_p)$ *is polynomial in the size of* t_p, *and*

(2) $\mathsf{AS}(t_p)$ *is computable in time polynomial in the size of* t_p.

3. THE ALGEBRA

In this section, we define a set of operators that extend the relational algebra to handle annotated tuples.

Table IV. Tuples in the `thermal`

LOC	THERM
site1	$\{a, b\}$ $h(a) = [0.4, 0.7]; h(b) = [0.3, 0.6]$
site2	$\{a\}$ $h'(a) = [1, 1]$.

Table V. Annotation representation

LOC	THERM	LB	UB	PATH
site1	a	0.4	0.7	w_9
site1	b	0.3	0.6	w_{10}
site2	a	1	1	w_{11}

Convention. Let r be an annotated relation and $\vec{a}$ be a data-tuple such that $\vec{a}$ does not appear in r. We implicitly assume that the annotated tuple $(\vec{a}, 0, 0,$ **false**$)$ is in r for the purpose of our definitions.

3.1 Selection

Our first operation is selection. From a practical and expressive power point of view, we can allow selection on data attributes as well as probability attributes. For data selection, we allow any legal selection condition that is allowed in classical relational algebra. Since these conditions involve only the conventional (data) attributes of annotated relations, the notion of an annotated tuple satisfying a selection condition is identical to the classical case. By a data attribute of an annotated relation, we mean any attribute other than one of PATH, LB, and UB. Probabilistic selection requires a new definition.

Definition 3.1 (*Selection*). Let r be an annotated relation and $\mathcal{F}$ be any legal selection condition over the data attributes of r. Then $\sigma_{\mathcal{F}}(r) = \{t_a \in r | t_a$ satisfies $\mathcal{F}\}$.

For a probabilistic condition $\mathcal{C}$ and an annotated relation r, $\sigma_{\mathcal{C}}(r) = \{t_a \in r | t_a$ satisfies $\mathcal{C}\}$, where $\mathcal{C}$ is a probabilistic selection condition as defined below:

—every (data) selection condition $\mathcal{F}$ as previously defined is a (probabilistic selection) condition.

—for a real number $n \in [0, 1]$, and $\theta \in \{=, >, <, \geq, \leq, \neq\}$, $LB\ \theta\ n$ and $UB\ \theta\ n$ are both conditions.

—whenever $\mathcal{C}_1$, $\mathcal{C}_2$ are conditions, so are $(C_1 \wedge C_2)$, $(C_1 \vee C_2)$ and $\neg C_1$.

Example 3.1 Let us return to the `target` relation of Example 2.1, shown in Example 2.4. Let $r =$ AS(`target`) denote the annotated representation of the `target` relation, shown in Example 2.4. Then the query $\sigma_{\mathcal{F}}(r)$

Table VI. Type 1 radars

LOC	OBJ	BAND	LB	UB	PATH
site2	radar_type1	700	0.8	0.9	w_3

where

$$\mathcal{F} = (\texttt{OBJ} = \texttt{radar\_type1} \wedge \text{LB} \geq 0.75)$$

says: *find all tuples that identify objects to be Type 1 radars with over 75% probability*. The result of this query is shown in Table VI. Note that the definition of selection on *annotated relations* is identical to classical selection.

3.2 Projection

We now turn to projection. Let t_a be an annotated tuple over R and let $X \subseteq R$. Then the *restriction* of t_a to X, denoted $t_a[X]$, is obtained by deleting all components of t_a not corresponding to one of the attributes in X or to one of *LB*, *UB*, *PATH*. Notice that as a consequence of this criterion, the attributes *LB*, *UB*, *PATH* of an annotated tuple $(\vec{a}, \ell, u, p)$ *are carried through* projection; that is, these attributes are "inherited" by the resulting relation after the projection is performed.

Definition 3.2 (*Projection*). Let r be an annotated relation over the scheme R and let $X \subseteq R$. Then $\pi_X(r) = \{t_a[X] | t_a \in r\}$.

Although the definition of projection looks similar to classical projection, there is a subtle difference: the result of a probabilistic projection may contain tuples that are data-identical although there may be differences in the PATH (and perhaps even the LB and UB) attributes. Thus, probabilistic projection can be thought of as classical projection "without duplicate elimination.")

Example 3.2 Let r denote the annotated representation of the `target` relation of Example 2.1, shown in Example 2.4. Then, the *projection* of r on the *BAND* attribute yields Table VII. Note that in ordinary relational DBMSs, we would have had only one column and three tuples (corresponding to the BAND values 700, 750, and 800, respectively), if duplicates are eliminated. In contrast, in our probabilistic framework, the probabilistic attributes and the path attributes are carried through.

3.3 Cartesian Product

Unlike selection and projection, operations such as Cartesian product and join are not straightforward extensions of their classical counterparts, as they must take into account the *strategies* for combining probabilistic tuples. This is because, in general, if we know that tuple t_1 (resp., t_2) is in relation R_1 (resp., R_2) with probability range $[p_1, q_1]$ (resp., $[p_2, q_2]$), then the tuple t_1t_2 is in the Cartesian product $(R_1 \times R_2)$ with a probability that

Table VII. Projection of r on the BAND

BAND	LB	UB	PATH
750	0.4	0.7	w_1
800	0.5	0.9	w_2
700	0.8	0.9	w_3
700	0.08	0.03	w_4
700	0	0.5	w_5
750	0	0.4	w_6
700	0.1	0.6	w_7
750	0	0.4	w_8

depends upon the known relationship among tuples R_1, R_2 (and the tuples involved). In our framework, when performing a Cartesian product, the user has the opportunity to specify this information. To this end, we define a *generic concatenation* operation on tuples. The user can specify any probabilistic strategy to compute the Cartesian product, as long as the strategy satisfies the following *postulates* on the structure and semantics of computing concatenations of tuples (conjunctions of events).

Let $[\alpha_1, \beta_1]$, $[\alpha_2, \beta_2]$ be any two probability intervals associated with events e_1, e_2, respectively. Suppose we want to compute the probability range associated with the compound event $e_1 \wedge e_2$. We use the symbol $\otimes$ to denote some function that computes the probability range of $e_1 \wedge e_2$ given the probability ranges for e_1 and e_2, respectively. More precisely, we define a *generic probabilistic conjunction* as a function $\otimes : \mathcal{P} \times \mathcal{C}[0, 1] \times \mathcal{P} \times \mathcal{C}[0, 1] \rightarrow \mathcal{C}[0, 1]$. Any such generic conjunction function must satisfy the following postulates.

Postulates for Probabilistic Conjunction. In the following, $[\alpha_i, \beta_i]$ are arbitrary elements of $\mathcal{C}[0, 1]$ and p, q, r are arbitrary elements of $\mathcal{P}$. Recall that $\mathcal{P}$ is the set of all paths (i.e., Boolean expressions over world ids) over a probabilistic database $\mathcal{D}$.

(C1) (*Bottomline*) It is always the case that $(p, [\alpha_1, \beta_1]) \otimes (q, [\alpha_2, \beta_2]) \leq [min(\alpha_1, \alpha_2), min(\beta_1, \beta_2)]$. Here, $[a, b] \leq [a', b']$ iff $a \leq a'$ and $b \leq b'$.

(C2) (*Ignorance*) When nothing is known about the interdependence between the events, $(p, [\alpha_1, \beta_1]) \otimes (q, [\alpha_2, \beta_2]) = (p, [\alpha_1, \beta_1]) \otimes_{ig} (q, [\alpha_2, \beta_2]) = [\max(0, \alpha_1 + \alpha_2 - 1), \min(\beta_1, \beta_2)]$ which is the formula for conjunction under *total ignorance* and under no independence assumptions whatsoever [Ng and Subrahmanian 1993, 1995]. Here $\otimes_{ig}$ refers to this *specific* combination strategy. In general, we require that any conjunction strategy $\otimes$ satisfy the constraint that: $(p, [\alpha_1, \beta_1]) \otimes (q, [\alpha_2, \beta_2]) \subseteq (p, [\alpha_1, \beta_1]) \otimes_{ig} (q, [\alpha_2, \beta_2])$; that is, any probabilistic strategy assumes at least as much knowledge as being totally ignorant.

(C3) (*Identity*) $(p, [\alpha, \beta]) \otimes (q, [1, 1]) = [\alpha, \beta]$, as long as $p \wedge q$ is consistent.

(C4) (*Annihilator*) $(p, [\alpha, \beta]) \otimes (q, [0, 0]) = [0, 0]$.

(C5) (*Commutativity*) $(p, [\alpha_1, \beta_1]) \otimes (q, [\alpha_2, \beta_2]) = (q, [\alpha_2, \beta_2]) \otimes (p, [\alpha_1, \beta_1])$.

(C6) (*Associativity*) $((p, [\alpha_1, \beta_1]) \otimes (q, [\alpha_2, \beta_2])) \otimes (r, [\alpha_3, \beta_3]) = (p, [\alpha_1, \beta_1]) \otimes ((q, [\alpha_2, \beta_2]) \otimes (r, [\alpha_3, \beta_3]))$.

(C7) (*Monotonicity*) $(p, [\alpha_1, \beta_1]) \otimes (q, [\alpha, \beta]) \leq (p, [\alpha_2, \beta_2]) \otimes (q, [\alpha, \beta])$ if $[\alpha_1, \beta_1] \leq [\alpha_2, \beta_2]$.

Postulate (C1) expresses the intuition that the confidence associated with the conjunction of two events can be no more than that associated with the events themselves. Postulate (C2) says that under total ignorance about the interdependency of events, the probabilistic bounds on the conjunction are those given by Fenstad [1980]. Postulates (C3) and (C4) have an obvious intuition. Postulates (C5) and (C6) are imposed mainly with query optimization in mind. Indeed, in the absence of these postulates, no nontrivial optimization opportunities for the algebraic expressions may exist. Postulate (C7) specifies that conjunction is monotonic. Many conjunction strategies used in practice tend to satisfy these postulates. Note that these postulates are not intended to be exhaustive. They are a *minimal* set of postulates that any probabilistic conjunction operator must satisfy. Specific applications may require the addition of new axioms. However, any results derivable from the preceding set of postulates will also apply to expanded sets of postulates.

We now define generic concatenation of tuples.

Definition 3.3 (*Generic concatenation*). Let $\otimes$ be any conjunction strategy that satisfies the postulates (C1)–(C7). Let $t_r = (a_1, \ldots, a_m, \mu_1, \mu_2, p_1) \in r$ and $t_s = (b_1, \ldots, b_n, \mu_3, \mu_4, p_2) \in s$, where r, s are any annotated relations. Then the *generic concatenation* of t_r, t_s, denoted $t_r \odot t_s$, is defined as $t_r \odot t_s = (a_1, \ldots, a_m, b_1, \ldots, b_n, \alpha, \beta, p)$, where $[\alpha, \beta] = (p_1, [\mu_1, \mu_2]) \otimes (p_2, [\mu_3, \mu_4])$, and $p = p_1 \wedge p_2$.

The intuition is that generic concatenation extends ordinary concatenation by accounting for an appropriate strategy to combine the probability intervals associated with the operand tuples. In addition, it computes the path associated with the output as the (logical) conjunction of the input paths. The preceding postulates ensure that the strategies used respect the laws of probability. We are now ready to define Cartesian product.

Definition 3.4 (*Cartesian product*). Let $\odot$ be any generic notion of concatenation of annotated tuples. Let r and s be two annotated relations. Then the Cartesian product of r and s induced by $\odot$ is defined as $r \times s = \{t_r \odot t_s \mid t_r \in r, t_s \in s\}$.

Observe that since Cartesian product is defined in terms of $\odot$, which in turn could reflect different possible strategies for combining probability intervals, this definition is *robust* and facilitates a wide variety of probabilistic combinations. We next give some examples of some specific strategies for conjunction and hence concatenation.

Example 3.3 Let $t_r = (\vec{a}, \alpha_1, \beta_1, p_1)$, $t_s = (\vec{b}, \alpha_2, \beta_2, p_2)$ be annotated tuples as previously. Let $t_a = t_r \odot t_s$ and let $[\alpha, \beta]$ be the probability interval associated with t_a. Here are some examples for computing $\odot$.

(1) (*Ignorance*) The conjunction can be performed by ignoring the path information and computing under total ignorance. This gives rise to

$$[\alpha, \beta] = (p_1, [\alpha_1, \beta_1]) \otimes_{ig} (p_2, [\alpha_2, \beta_2]),$$

where $\otimes_{ig}$ is defined in the postulate (C2) for probabilistic conjunction.

(2) (*Positive Correlation*) The conjunction can be performed, again ignoring the path information, but computing under the conservative assumption that the overlap[3] between the two events is maximal, to the point where one of them implies the other. This leads to

$$[\alpha, \beta] = (p_1, [\alpha_1, \beta_1]) \otimes_{pc} (p_2, [\alpha_2, \beta_2]) =_{def} [min(\alpha_1, \alpha_2), min(\beta_1, \beta_2)],$$

where $\otimes_{pc}$ stands for probabilistic conjunction under the assumption of *positive correlation*, previously explained.

(3) (*Path-based*) We can take the path information into account while computing probabilistic conjunction. This leads to

$$[\alpha, \beta] = \begin{cases} [0, 0], & \text{if } p_1 \wedge p_2 \text{ is inconsistent.} \\ (p_1, [\alpha_1, \beta_1]) \otimes_{ig} (p_2, [\alpha_2, \beta_2]), & \text{if } p_1 \wedge p_2 \text{ and } p_1 \wedge \neg p_2 \text{ are consistent.} \\ (p_1, [\alpha_1, \beta_1]) \otimes_{pc} (p_2, [\alpha_2, \beta_2]), & \text{otherwise.} \end{cases}$$

The rationale behind this last strategy is that when the two events are conflicting (as indicated by the inconsistency of $p_1 \wedge p_2$), the bounds associated with the conjunction are set to [0, 0]. When there is no overlap between the paths associated with the two tuples (as indicated by the consistency of $p_1 \wedge p_2$ and $p_1 \wedge \neg p_2$), the associated probability intervals are combined in the sense of total ignorance. This amounts to *not* assuming any a priori relationships among the basic wids. However, when there is some overlap between the two paths, the strategy is maximally conservative about their conjunction. It assumes the worst case, where one of the paths may (logically) imply the other and hence assigns the *min* of the probability intervals to the conjunction. There could be other viable strategies as well. In principle, it is possible to set up a system of linear constraints and solve them from first principles to obtain exact bounds for the conjunction. Although this can serve as the basis for the correctness of a strategy, this is clearly not an attractive strategy from the complexity viewpoint.

[3]We say two paths p_1, p_2 overlap if $(p_1 \wedge p_2)$ is consistent. Additional conditions may be added to describe different kinds of overlaps, but we do not describe these here.

Table VIII. Probability intervals

LOC	THERM	BAND	LB	UB	PATH
site1	a	750	0.4	0.7	$(w_9 \wedge w_1)$
site1	a	800	0.4	0.7	$(w_9 \wedge w_2)$
site1	a	700	0.4	0.7	$(w_9 \wedge w_3)$
site1	a	700	0.08	0.03	$(w_9 \wedge w_4)$
site1	a	700	0	0.5	$(w_9 \wedge w_5)$
site1	a	750	0	0.4	$(w_9 \wedge w_6)$
site1	a	700	0.1	0.6	$(w_9 \wedge w_7)$
site1	a	800	0	0.4	$(w_9 \wedge w_8)$
site1	b	750	0.3	0.6	$(w_{10} \wedge w_1)$
site1	b	800	0.3	0.6	$(w_{10} \wedge w_2)$
site1	b	700	0.3	0.6	$(w_{10} \wedge w_3)$
site1	b	700	0.08	0.03	$(w_{10} \wedge w_4)$
site1	b	700	0	0.5	$(w_{10} \wedge w_5)$
site1	b	750	0	0.4	$(w_{10} \wedge w_6)$
site1	b	700	0.1	0.6	$(w_{10} \wedge w_7)$
site1	b	800	0	0.4	$(w_{10} \wedge w_8)$
site2	a	750	0.4	0.7	$(w_{11} \wedge w_1)$
site2	a	800	0.5	0.9	$(w_{11} \wedge w_2)$
site2	a	700	0.8	0.9	$(w_{11} \wedge w_3)$
site2	a	700	0.08	0.03	$(w_{11} \wedge w_4)$
site2	a	700	0	0.5	$(w_{11} \wedge w_5)$
site2	a	750	0	0.4	$(w_{11} \wedge w_6)$
site2	a	700	0.1	0.6	$(w_{11} \wedge w_7)$
site2	a	800	0	0.4	$(w_{11} \wedge w_8)$

Example 3.4 Consider the annotated version of the probabilistic relation `thermal` (cf. Example 2.5) and consider taking the Cartesian product of this relation with π_{BAND} (AS(`target`)), the relation shown in Example 3.5. The result of the Cartesian product is shown in the following. Note that the *conjunction strategy* being used in the example is that of *positive correlation*. Had we chosen to use an alternative strategy (e.g., Ignorance, Independence, Negative Correlation, or Path-Based methods), then the probability intervals shown in Table VIII would be somewhat different.

Before concluding this example, we observe that several tuples in it are *data-identical*; that is, they coincide on all data attributes. Later, in Section 3.5.2, we show how we can summarize/compress these seemingly redundant tuples.

The reader will easily notice that the *join* of two annotated relations can be easily expressed in terms of the operators defined thus far (as is also the case in the classical relational algebra).

3.4 Union

Two annotated relations are union compatible exactly when their underlying classical relations are.

Definition 3.4 (*Union*). Let r, s be any union compatible annotated relations. Then $r \cup s = \{t | t \in r, \text{ or } t \in s\}$.

Union is analogous to classical union. Again, just as with projection, the subtlety is that the union of two annotated relations might well contain distinct tuples that are data-identical (and differ in the path component).

3.5 Compaction

Our final operation, called *compaction*, is the probabilistic counterpart of duplicate elimination. As already seen, union and projection can result in relations that contain data-identical tuples. Sometimes, it may be desirable to find out the overall probability that a certain data-tuple belongs to the underlying classical relation. For example, a given data-tuple may occur many times in an annotated relation with different paths justifying each such occurrence. In order to combine these different paths, as well as the probability bounds associated with each such derivation of a data-tuple, we need a generic disjunction between tuples, which we denote $\oplus$. Generic disjunction may then be used to define the compaction operation.

3.5.1 *Generic Disjunction.* Formally, *generic disjunction* is a function $\oplus:(\mathcal{P} \times \mathcal{C}[0, 1]) \times (\mathcal{P} \times \mathcal{C}[0, 1]) \rightarrow \mathcal{C}[0, 1]$. As in the case of probabilistic conjunction, we need specific axioms governing the use of such a disjunctive combination strategy. The following postulates capture these requirements.
Postulates for Generic Disjunction

In the following, $[\alpha_i, \beta_i]$ are arbitrary elements of $\mathcal{C}[0, 1]$ and p, q, r are arbitrary elements of $\mathcal{P}$.

(D1) (*Bottomline*) It is always the case that $(p, [\alpha_1, \beta_1]) \oplus (q, [\alpha_2, \beta_2]) \geq [max(\alpha_1, \alpha_2), max(\beta_1, \beta_2)]$. Here $[a, b] \leq [a', b']$ iff $a \leq a'$ and $b \leq b'$.

(D2): (*Ignorance*) When nothing is known about the interdependence between the events, $(p, [\alpha_1, \beta_1]) \oplus (q, [\alpha_2, \beta_2]) = (p, [\alpha_1, \beta_1]) \oplus_{ig} (q, [\alpha_2, \beta_2]) = [\max(\alpha_1, \alpha_2), \min(1, \beta_1 + \beta_2)]$ which is the formula for disjunction under *total ignorance* and under no independence assumptions whatsoever [Ng and Subrahmanian 1993, 1995]. Here $\oplus_{ig}$ refers to this *specific* disjunction strategy. In general, we require that any disjunction strategy $\otimes$ satisfy the constraint that: $(p, [\alpha_1, \beta_1]) \oplus (q, [\alpha_2, \beta_2]) \subseteq (p, [\alpha_1, \beta_1]) \oplus_{ig} (q, [\alpha_2, \beta_2])$, that is, any probabilistic strategy assumes at least as much knowledge as being totally ignorant.

(D3) (*Identity*) $(p, [\alpha, \beta] \oplus (q, [0, 0]) = [\alpha, \beta]$.

(D4) (*Annihilator*) $(p, [\alpha, \beta]) \oplus (q, [1, 1]) = [1, 1]$.

(D5) (*Commutativity*) $(p, [\alpha_1, \beta_1]) \oplus (q, [\alpha_2, \beta_2]) = (q, [\alpha_2, \beta_2]) \oplus (p, [\alpha_1, \beta_1])$.

(D6) (*Associativity*) $((p, [\alpha_1, \beta_1]) \oplus (q, [\alpha_2, \beta_2])) \oplus (r, [\alpha_3, \beta_3]) = (p, [\alpha_1, \beta_1]) \oplus ((q, [\alpha_2, \beta_2]) \oplus (r, [\alpha_3, \beta_3]))$.

(D7) (*Monotonicity*) $(p, [\alpha_1, \beta_1]) \oplus (q, [\alpha, \beta]) \leq (p, [\alpha_2, \beta_2]) \oplus (q, [\alpha, \beta])$ if $[\alpha_1, \beta_1] \leq [\alpha_2, \beta_2]$.

Let $\oplus$ be any strategy for computing the probability of the disjunction of events, which satisfies the preceding postulates. We can extend $\oplus$ to annotated tuples as follows. Let $(\vec{a}, \alpha_1, \beta_1, p_1)$ and $(\vec{a}, \alpha_2, \beta_2, p_2)$ be annotated tuples. Then $(\vec{a}, \alpha_1, \beta_1, p_1) \oplus (\vec{a}, \alpha_2, \beta_2, p_2) = (\vec{a}, \alpha, \beta, p)$, where $[\alpha, \beta] = (p_1, [\alpha_1, \beta_1]) \oplus (p_2, [\alpha_2, \beta_2])$, and $p = p_1 \vee p_2$.

We now give some example strategies for computing probabilistic disjunction. These strategies can be viewed as "duals" of those given for conjunction in Example 3.3.

Example 3.5 Let $t_r = (\vec{a}, \alpha_1, \beta_1, p_1)$, $t_s = (\vec{b}, \alpha_2, \beta_2, p_2)$ be any annotated tuples. Let $t_a = t_r \oplus t_s$ and let $[\alpha, \beta]$ be the probability interval associated with t_a. Here are some examples for computing $\oplus$.

(1) (*Ignorance*) The disjunction can be performed by ignoring the path information and computing under total ignorance. This gives rise to

$$[\alpha, \beta] = (p_1, [\alpha_1, \beta_1]) \oplus_{ig} (p_2, [\alpha_2, \beta_2]),$$

where $\oplus_{ig}$ is defined in the postulate (D2) for probabilistic disjunction.

(2) (*Positive Correlation*) The disjunction can be performed, again ignoring the path information, but computing under the conservative assumption that the overlap between the two events is maximal, to the point where one of them implies the other. This leads to

$$[\alpha, \beta] = (p_1, [\alpha_1, \beta_1]) \oplus_{pc} (p_2, [\alpha_2, \beta_2]) =_{def} [max(\alpha_1, \alpha_2),\ max(\beta_1, \beta_2)],$$

where $\oplus_{pc}$ stands for probabilistic disjunction under the assumption of *positive correlation*, previously explained.

(3) (*Path-Based*) We can take the path information into account while computing probabilistic disjunction. This leads to

$$[\alpha,\beta] = \begin{cases} (p_1, [\alpha_1, \beta_1]) \oplus_{nc} (p_2, [\alpha_2, \beta_2]) =_{def} [min(1, \alpha_1 + \alpha_2), min(1,\beta_1 + \beta_2)], & \text{if } p_1 \wedge p_2 \text{ is inconsistent.} \\ (p_1, [\alpha_1, \beta_1]) \oplus_{ig} (p_2, [\alpha_2, \beta_2]), & \text{if } p_1 \wedge p_2 \text{ and } p_1 \wedge \neg p_2 \text{ are consistent.} \\ (p_1, [\alpha_1, \beta_1]) \oplus_{pc} (p_2, [\alpha_2, \beta_2]), & \text{otherwise.} \end{cases}$$

The first two strategies are self-explanatory. The rationale for the last strategy is fairly similar to that for strategy 3 for conjunction in Example 3.3. The only difference is that when the paths are inconsistent, we deduce that the overlap between the events is minimal and apply the formula for negative correlation ($\oplus_{nc}$).

3.5.2 *Compaction.* Now that we have laid down the postulates for disjunctive combination of probabilities of data-identical tuples with various associated paths, we are in a position to "merge" data-identical tuples.

Table IX. Positive correlation

BAND	LB	UB	PATH
750	0.4	0.7	$(w_1 \vee w_6 \vee w_8)$
800	0.5	0.9	w_2
700	0.8	0.9	$(w_3 \vee w_4 \vee w_5 \vee w_7)$

This merge operation is called *compaction* and is formally defined later in this section. However, before proceeding any further, we give a quick example.

Example 3.6 Let us consider the relation π_{BAND}(AS(`target`)) shown in Example 3.2. Table VII contains several data-identical tuples. The aim of the *compaction* operation is to compactly represent the content of such data-identical tuples.

For example, there are two tuples in π_{BAND}(AS(`target`)) that have the value `BAND=750`. These two tuples can be "merged" into a new tuple whose path field is $(w_1 \vee w_6)$ and whose BAND field is 750. The probability of this merged tuple is obtained from the two original tuples by applying the desired generic disjunction operator. Thus, if we wish to use *positive correlation* as the desired disjunction strategy, then this procedure, when applied to Table VII, π_{BAND}(AS(`target`)), shown in Example 3.2, yields Table IX.

As the reader will notice, compaction summarizes the reasons to believe that a particular data tuple is in a relation. In the preceding example, the "reasons" for the BAND identified by the sensors being 750 KHz is that this is the case if either worlds w_1 or w_6 hold, and the probability of this, as computed by the desired disjunction strategy, is 40 to 70%.

It is also important to observe that *one cannot reconstruct the original* π_{BAND}AS(`target`)) *relation* (cf. preceding) *from the compacted Table 9*. Thus, in our framework, we leave the option of whether to compact to the discretion of the user. To facilitate this, we develop a special compaction operator κ. If the user wishes to compact the result of a query that he or she poses, then this must be explicitly stated by the user. (In our system, the user can set up compaction as the default if so desired.)

Definition 3.6 (*Compaction*). Let r be an annotated relation. Then its *compaction*, denoted $\kappa(r)$, is defined as $\kappa(r) = \{t | \{t_1, \ldots, t_k\} \subseteq r$ is a maximal subset of tuples in r that are data-identical and $t = \oplus\{t_1, \ldots, t_k\}\}$.

Intuitively, compaction works as follows.

(1) Partition the relation r into blocks of tuples that are data identical.
(2) For each block, compute the generic disjunction of the tuples in the block.
(3) Output the result.

From this intuition, it can be seen that the computation associated with compaction is very similar to aggregate computation in conventional relational databases.

3.6 Difference

Our last operation is difference. Just as with conjunction, we need to pay attention to the way the probability intervals of the operand tuples are combined. To this end, we let $\ominus$ be a generic difference strategy on annotated tuples. In other words, depending upon the precise interdependence between two events e_1, e_2, we may compute the probabilities of the compound event $(e_1 \wedge \neg e_2)$ corresponding to difference, using one of a number of strategies. Our approach is based on the following intuitions: (i) whenever the probability of an event is $\mathbf{Prob}(e) = [\alpha, \beta]$, the probability associated with its complement $\neg e$ is given by $\mathbf{Prob}(\neg e) = [1 - \beta, 1 - \alpha]$, and (ii) $\mathbf{Prob}(e_1) \ominus \mathbf{Prob}(e_2) = \mathbf{Prob}(e_1) \otimes \mathbf{Prob}(\neg e_2)$. The idea for difference $r_1 - r_2$, then, is to regard the tuples in the relations as events and appeal to the preceding intuitions. A complication arises when there are annotated tuples in r_1, r_2 that are data-identical. In the classical case, any tuple that belongs to both relations, by definition, does not belong to their difference. In the probabilistic case, the subtlety is that the effect of one tuple in r_1 can be captured by a *set* of tuples in r_2. For example, consider $r = \{(a, \alpha_1, \beta_1, p)\}$ and $s = \{(a, \alpha_2, \beta_2, p \wedge q), (a, \alpha_3, \beta_3, p \wedge \neg q)\}$. Notice that: (i) the tuples in r and s are data-identical, and (ii) the path associated with the tuple in r logically implies the disjunction of the paths associated with the two tuples in s. In general, we capture this phenomenon using the notion of path-subsumption, defined in the following way.

Definition 3.7 (*Path subsumption*). Let $t_1 = (\vec{t}, \ell_1, u_1, p_1)$ and $t_2 = (\vec{t}, \ell_2, u_2, p_2)$ be data-identical annotated tuples.

An annotated tuple $t_a = (\vec{a}, \alpha, \beta, p)$ is *path-subsumed* by a set of annotated tuples $S = \{(\vec{a}, \alpha_i, \beta_i, p_i) | 1 \leq i \leq n\}$, provided that (i) t_a is data-identical to every tuple in S, and (ii) p logically implies $p_1 \vee \cdots \vee p_n$.

It is important to note that probability intervals do *not* play any role in the definition of path-subsumption. The reason is that path-subsumption is used mainly as a means for determining which tuples should get into the output of the difference between two annotated relations. When defining the difference operator, we use these phases: in the first, we determine which tuples in relation r are *not* path-subsumed by any (sub)set of tuples in relation s, and in the second, we determine the probabilities associated with the output tuples.

Intuition behind Definition 3.7. We can regard each wid w_i as a proposition. Let $w_1, \ldots, w_k$ be all the tuple-worlds associated with a database. With each truth assignment on these propositions we can identify a *database-world*. For example, w.r.t. Example 2.4, the truth assignment d: $w_1 \mapsto$ **true**, $w_2 \mapsto$ **false**, $w_3 \mapsto$ **true**, $w_i \mapsto$ **false**, $4 \leq i \leq 8$ corresponds to a database-world where the tuple-worlds w_1 and w_3 (and hence the associated data-tuples; see Example 2.4) hold and all other tuple-worlds do not. Concretely, this corresponds to a classical database where exactly the data-tuples associated with w_1 and w_3 are true and all other data-tuples are false. In this perspective, the path p associated with a tuple $t_a = (\vec{a}, \ell, u, p)$ asserts that the data-tuple $\vec{a}$ holds in every database-world satisfying p, whereas the set of tuples S in the preceding definition says $\vec{a}$ is true in every database-world that satisfies $p_1 \vee \cdots \vee p_n$. Since p logically implies $p_1 \vee \cdots \vee p_n$, we see that the set of database-worlds in which $\vec{a}$ is true according to S, includes the worlds where it is true according to t_a.

We are now ready to define the difference operator.

Definition 3.8 (*Difference*). Let r, s be any union-compatible annotated relations, and let $\oplus$ and $\otimes$ be any generic probabilistic disjunction and conjunction strategies, respectively. Then

$$r - s = \{(\vec{a}, \alpha, \beta, p) \mid \exists\, t_r = (\vec{a}, \alpha', \beta', p') \in r \text{ such that: } \oplus(\mathbf{S}_a)$$

$$= (\vec{a}, \alpha'', \beta'', p'') \text{ and } t_r \text{ is not path-subsumed by } \mathbf{S}_a \text{ and } [\alpha, \beta]$$

$$= (p', [\alpha', \beta']) \otimes (\neg p'', [1 - \beta'', 1 - \alpha'']) \text{ and } p = p' \wedge \neg p''\},$$

where $\mathbf{S}_{\vec{a}}$ denotes the set of tuples in s that are data-identical to $\vec{a}$ (i.e., the selection of s on $\vec{a}$).

The idea is that a tuple in r gets into the result of the difference operator, provided it is not path-subsumed by any set of tuples in s. Clearly, t_r is not path-subsumed by any set of tuples in s exactly when it is not path-subsumed by the set of all tuples in s that are data-identical to t_r. This explains the preceding definition. In general, testing path-subsumption can be expensive. This is reflected in our complexity results later.

Note that for each specific strategy chosen for probabilistic conjunction, we get a different strategy for difference. By definition, an empty disjunction corresponds to **false**. As a special case, when there is a tuple $t_r \in r$ such that no tuple in s is data-identical to t_r, we trivially have that $t_r \in r - s$, showing that the classical difference operator is a special case of probabilistic difference.

3.7 Queries and Views

Definition 3.9 Suppose $r_1, \ldots, r_n$ are the annotated relations in an annotated database. An *elementary query* is any legal algebraic expression constructed solely using the operators of selection, projection, difference, Cartesian product, and union on $r_1, \ldots, r_n$. An *elementary view definition* is any elementary query.

A (general) *view definition* is defined as follows.

(1) Any elementary view definition is a view definition.

(2) If E is a view definition, then $\kappa(E)$ is a view definition.

As in classical relational databases, the difference between queries and views is that queries can be ad hoc, whereas views are relatively permanent. In classical database applications, views were not materialized and only their definitions were stored (relatively) permanently. However, materialized views are popular in many newer database applications (e.g., see Gupta et al. [1993]).

Given a view definition VD, the *materialization* of this view definition is simply the set of all tuples obtained by executing the algebraic expression defining the view.

We conclude this section with an analysis of the data complexity of query computation and view materialization. Intuitively, data-complexity measures the complexity of evaluating a view (or query) E as a function of the size of the database D (see Vardi [1985]).

THEOREM 3.1 (Complexity of positive queries and views). *Let E be a query or view definition over an annotated database D. Then:*

(1) *if E involves only selection, projection, and union operators, then it can be computed in time polynomial in the database size $|D|$.*

(2) *In general, whenever E is an expression in the positive fragment of the algebra (i.e., it does not involve the difference operator), it can be computed in time polynomial in the database size $|D|$, as long as the adopted strategies for generic concatenation and generic disjunction are computable in time polynomial in $|D|$.*

Thus, the data-complexity of the positive fragment of probabilistic relational algebra is essentially the same as classical relational algebra, provided that polynomial time strategies for generic concatenation and disjunction are adopted. Observe that a relevant class of these strategies are polynomial time computable. For instance, both strategies (1) and (2) for generic concatenation described in Example 3.3 are polynomial time computable. Similarly, strategies (1) and (2) for generic disjunction of Example 3.5 are tractable.

Clearly, a probabilistic operator cannot be polynomial time computable if the underlying generic strategy is intractable. For instance, if we adopt strategy (3) of Example 3.3 for generic concatenation (which requires a consistency check), then the resulting Cartesian product operator is not tractable (unless P = NP). The same holds for compaction if we use strategy (3) of Example 3.5 for generic disjunction.

Whereas Theorem 3.1 shows that under reasonable assumptions on the strategies for generic conjunction and disjunction, queries expressible in positive probabilistic algebra are computable in polynomial time, our next theorem shows that when queries involve the difference operator, the complexity is higher, even when all strategies used are polynomial. This situation is similar to query processing with null values, where as long as queries do not involve difference, the complexity remains polynomial; when difference is involved, even testing membership of a tuple in a query is co-NP-complete [Vardi 1985].

THEOREM 3.2 (Complexity of difference). *Let r and s be two union-compatible annotated relations.*

(1) *Recognizing whether an annotated tuple t belongs to $r - s$, is NP-hard (even if compaction is not allowed).*
(2) *If polynomial strategies for generic disjunction and conjunction are adopted, then recognizing whether an annotated tuple t belongs to $r - s$, is NP-complete.*

The complexity results are interesting for the following reasons. (1) They apply to a broad range of strategies for the individual operations, where the *only* assumptions made are those made *explicit* by the *postulates*. As already discussed, these postulates are *necessary* for the operations to respect probability theory. (2) The results show that under the assumption that the combination strategies chosen are polynomial time computable, the positive algebra is tractable. This assumption is reasonable since many well-known strategies (e.g., positive correlation, negative correlation, independence, ignorance, etc.) are constant time computable. On the other hand, the expressive power of our algebra can be increased by "tuning" the strategies to be more sophisticated ones, which may, for example, test the consistency of path expressions. (3) Although the difference operator incurs a higher complexity, it should be noted that this is quite analogous to the higher complexity associated with difference in relational databases with null values. Indeed, both *incompleteness* and *uncertainty* are very important aspects of practical database applications. It is interesting that the computational complexity associated with these apparently different aspects is similar.

The preceding remarks show that our design of probabilistic algebra provides a nice balance between tractability and expressive power.

3.8 Query Equivalences and Containments

We now develop various kinds of algebraic identities involving containment and equivalence, with the motivation that they will be helpful in optimizing queries. The most important thing about these algebraic identities is that they will apply to any probabilistic conjunction, disjunction, and difference strategies as long as these strategies satisfy the postulates described earlier. Before proceeding, note that in classical algebra, two expressions E_1, E_2 are equivalent, provided for all input databases D, the relations $E_1(D)$ and $E_2(D)$ are *identical*. For any two expressions E_1, E_2 in our probabilistic algebra, we say E_1 is contained in E_2, denoted $E_1 \subseteq E_2$, provided that for all input databases D, $E_1(D) \subseteq \kappa(E_2(D))$, where the containment is classical set-theoretic. In principle, we could define containment based on compacting both $E_1(D)$ and $E_2(D)$. The reason we define as in the preceding is two-fold: compaction is a costly operation, and the notion of containment/equivalence and the identities derived w.r.t. it should be useful in query optimization; and a definition of containment based on compacting both the preceding relations would *not* be very useful in this respect. We say E_1 and E_2 are *equivalent*, $E_1 \cong E_2$, provided $E_1 \subseteq E_2$ and $E_2 \subseteq E_1$. The following result is easy to prove by manipulating the definitions.

THEOREM 3.3 *Suppose r_1, r_2 are annotated relations and $\oplus$, $\odot$, and $\ominus$ are any arbitrary generic disjunction, concatenation, and difference strategies satisfying the postulates for disjunction, concatenation, and difference listed earlier. Then:*

$$\kappa(r_1 \cup r_2) \subseteq \kappa(r_1) \cup \kappa(r_2). \tag{1}$$

$$\kappa(r_1 \cup r_2) \cong \kappa(\kappa(r_1) \cup \kappa(r_2)). \tag{2}$$

$$\kappa(r_1 \times r_2) \subseteq \kappa(r_1) \times \kappa(r_2). \tag{3}$$

$$\kappa(r_1 \times r_2) \cong \kappa(\kappa(r_1) \times \kappa(r_2)). \tag{4}$$

$$\kappa(\sigma_{\mathcal{F}}(r)) \cong \sigma_{\mathcal{F}}(\kappa(r)). \tag{5}$$

$$\kappa(\pi_X(r)) \cong \kappa(\pi_X(\kappa(r))). \tag{6}$$

In Equation (5) *we assume that the selection condition $\mathcal{F}$ does not involve probabilistic attributes.*

In order to investigate the relative efficiencies of evaluating queries using the algebraic identities of Theorem 3.3, we ran a number of experiments which we describe later in Section 5.

3.9 Presenting Answers as Probabilistic Relations

We have seen (Section 1) that probabilistic relations constitute an intuitive "front end" to the user, whereas annotated relations, which faithfully represent the former, are more convenient for algebraic manipulation. Thus, answers to queries are essentially annotated relations. A natural question, then, is whether query answers can be presented in the form of probabilistic relations to the user. The answer to this question is "yes."

In our model, unlike that of Barbara et al. [1992], we do *not* pin down attributes as deterministic or probabilistic a priori. The user has the flexibility of seeing attributes in different perspectives. Let r be an annotated relation, which could be the result of a query. By a user perspective, we mean an assignment of one of the two statuses—deterministic or probabilistic—to each of the attributes of r. Let $A_1, \ldots, A_n$ be the deterministic attributes and $B_1, \ldots, B_m$ be the probabilistic ones, according to the user perspective. Then, this perspective induces a *probabilistic view* of the annotated relation, defined as follows. For tuples $t_a^1, t_a^2 \in r$, define $t_a^1 \equiv t_a^2$ iff (i) t_a^1, t_a^2 have equivalent path fields and (ii) $t_a^1.A_i = t_a^2.A_i$ for all $1 \leq i \leq n$. Let $[t_a^1], \ldots, [t_a^k]$ be the set of all equivalence classes so generated. With each equivalence class $[t_a^j]$, we associate a single probabilistic tuple T_j, defined as follows.

(1) $T_j.A_i = (V_i, h_i)$, where $V_i = \{t^j_a.A_i\}$ and $h(t^j_a.A_i) = [1, 1]$ for all $1 \le i \le n$;

(2) $T_j.B_i = (V_i, h_i)$, where $V_i = \{t.B_i | t \in [t^j_a]\}$ and $\forall v \in V_i{:}h_i(v) = \cap\{[\ell, u] | t \in [t^j_a]$ and $t.B_i = v$ and $t.LB = \ell$ and $t.UB = u\}$, $1 \le i \le m$.

Finally, the probabilistic view of r induced by the preceding perspective is the set of all probabilistic tuples associated with the equivalence classes of tuples from $r¡$ as defined. It is easy to verify that $r = \cup_{i=1}^{k} \mathsf{AS}(T_i)$. Thus, we can see that it is feasible to present answers to queries using the intuitive interface of probabilistic relations. The previous method finds one probabilistic relation that can generate the annotated answer relation associated with query Q. Note that there could, in general, be other alternative, but equivalent representations.

4. MAINTENANCE OF MATERIALIZED VIEWS

In the preceding section, we formally defined the concept of a *view*. Even though views and queries have the same form, they serve different purposes. Views are generally *materialized*; that is, the query defining a view is computed and *stored*. In other words, a view is *persistent*, whereas the answer to a query disappears once the user has seen it. In traditional databases, view maintenance has been studied extensively in the context of relational databases [Blakeley et al. 1989; Gupta et al. 1993], deductive databases [Harrison and Dietrich 1992], and object-oriented databases [Kemper et al. 1994; Dayal 1989]. More recently, attempts have been made to build views on top of *heterogeneous* information sources [Dayal and Hwang 1984; Lu et al. 1995] leading to the emerging area of data warehousing.

One of the fundamental problems that arises when materialized views are created is that the base relations "on top" of which these views are defined may be updated from time to time. This, in turn, may cause the answer to the query defining the view to change, thus requiring that the materialized view be updated correspondingly. In contrast to relational, object-oriented, and deductive databases, the problem of view maintenance in a probabilistic database system such as ours raises fundamentally *new* types of updates which we discuss in the following.

4.1 What's New About Updating Probabilistic Databases?

As the reader would have observed by now, in our proposed framework, the user will "see" probabilistic tuples. These probabilistic tuples will be "internally" represented as annotated tuples (or sets of annotated tuples) that are then manipulated appropriately by the **ProbView** system. When an update is made to the probabilistic relation seen by the user, this update must be transformed to a corresponding set of updates on the annotated relations that represent that probabilistic relation. Such updates are *fundamentally* different from those encountered in ordinary relational databases. This is best illustrated through an example.

Table X(a).

im1.gif	5	10	35	40	john	0.2	0.25
					jim	0.35	0.4
					tom	0.4	0.45

Table X(b).

im1.gif	5	10	35	40	tony	0.4	0.65
					jim	0.45	0.50
					tom	0.1	0.15

Example 4.1 Let us return to the face database mentioned in the Introduction and consider the tuple relating to the file `im1.gif` (recall that this is a surveillance image). Suppose now that the mugshot database is updated by the insertion of a new face, viz. that of someone called Tony. Table X(a) shows the initial table, and Table X(b) shows the situation desired after this update. As can easily be seen by the reader, the new "replacement" probabilistic tuple is quite different from the original tuple. Not only is its "data" content different (e.g., Tony is introduced, and John has disappeared) but the probabilities have changed as well.

The change of probabilities also has profound implications. For example, suppose a user U had defined a view called `suspect(X)` which holds if there exists a surveillance image in which `X` and `Jim` both appear with certainty 40% or more. In the scenario described in the Introduction, no such `X` existed because both tuples (t_p^1 and t_p^2) associated with the file `im1.gif` had Jim identified with $LB < 0.4$. However, after the aforementioned update, *the change in the probabilities* causes `Tony` to be one such suspect.

The preceding example shows that insertions into a probabilistic relation can have a significant impact on the annotated representations of those probabilistic relations as well as on the views defined on top of those annotated relations.

4.2 Allowed Updates on Probabilistic Relations

We allow three kinds of updates to be made to probabilistic relations. These are as follows.

(1) *Probabilistic Tuple Insertion (PT-Insertion):* A user wishes to insert a probabilistic tuple $t_p = (\mathbf{v}_1, \ldots, \mathbf{v_n})$ into a probabilistic relation R where each $\mathbf{v_i} = \langle V_i, h_i \rangle$. Handling the insertion of a probabilistic tuple into relation R requires two steps:
   (a) Converting t_p into $\mathsf{AS}(t_p)$ and adding $\mathsf{AS}(t_p)$ into the annotated representation of relation R. This can be done using the technique specified in the proof of Theorem 2.2.
   (b) If V is a view that accesses relation R, then V's materialization must be updated to reflect the update to R. Algorithms to do this are developed in Section 4.4.

(2) *Probabilistic Tuple Deletion (PT-Deletion):* A user wishes to delete a probabilistic tuple $t_p = (\mathbf{v_1}, \ldots, \mathbf{v_n})$ from probabilistic relation R where each $\mathbf{v_i} = \langle V_i, h_i \rangle$. In this case, only Step (1b) needs to be performed after initially deleting AS(t_p) from the annotated representation of R.

(3) *Probabilistic Tuple Modification (PT-Modification):* Three kinds of modifications may be performed to a probabilistic tuple $t_p = (\mathbf{v_1}, \ldots, \mathbf{v_n})$ where each $\mathbf{v_i} = \langle V_i, h_i \rangle$.
   (a) *Plain Component Addition:* V_i may be *expanded* to $V_i^* \supseteq V_i$ and h_i is extended to a new function h_i^* in such a way that for all $x \in V_i$, $h_i^*(x) = h_i(x)$.
   (b) *Component Addition with Probability Update:* V_i may be *expanded* to $V_i^* \supseteq V_i$ and h_i is replaced by a new function h_i^*.
   (c) *Plain Component Deletion:* V_i may be *diminished* to $V_i^* \subseteq V_i$ and h_i^* is the restriction of h_i to V_i^*; that is, for all $x \in V_i^*$, $h_i^*(x) = h_i(x)$.
   (d) *Component Deletion with Probability Update:* V_i may be *diminished* to $V_i^* \subseteq V_i$ and h_i^* is *any* probability assignment to V_i^*.

The case where $V_i = V_i^*$, but a probability update is made is subsumed by the cases listed. As stated earlier, we assume that the user expresses his or her update on probabilistic relations, rather than on annotated relations. Suppose t_p^* represents the probabilistic tuple t_p *after* the preceding modifications have been made. Then, in each of the preceding cases, we need to do the following.

(a) Convert t_p^* into its annotated representation AS(t_p^*); we study *incremental* algorithms to do this in Section 4.3, and
(b) If V is a view that accesses relation R, then V's materialization must be updated to reflect the content of t_p^*. Algorithms to do this are developed in Section 4.4.

4.3 Converting Updates on Probabilistic Relations to Updates on Annotated Relations

In this section, we show how the annotated version of a probabilistic relation may be easily updated when the probabilistic relation is updated.

In the case of PT-insertion and PT-deletion operations, the answer is easy: we merely append AS(t_p) or delete AS(t_p) from the annotated version of the relation to/from which t_p is being added/deleted.

The situation is somewhat different in the case of probabilistic tuple modification and we now consider this.

4.3.1 *PT-Modification: Plain Component Addition.* Let $t_p = (\mathbf{v_1}, \ldots, \mathbf{v_n})$ where each $\mathbf{v_i} = \langle V_i, h_i \rangle$ and suppose t_p^* is given by $(\mathbf{v_1^*}, \ldots, \mathbf{v_n^*})$ where each $\mathbf{v_i^*} = \langle V_i^*, h_i^* \rangle$. Furthermore, suppose $V_i^* \supseteq V_i$ for all $1 \leq i \leq n$ and suppose that for all $x \in V_i$, $h_i^*(x) = h_i(x)$.

Algorithm NAIVE-INS:

(1) Compute the set W^* of all t_p^*-worlds.
(2) Construct LP(t_p^*).

(3) For each world $w_i \in W^*$, minimize and maximize z_i w.r.t. LP(t_p^*) where z_i is the linear programming variable associated with world w_i as specified in the Proof of Theorem 2.2. AS(t_p^*) = $\{(\vec{a}, \ell_i, u_i, w_i) | w_i \in W^*$ and ℓ_i, u_i are obtained by minimizing/maximizing z_i w.r.t. LP(t_p^*)$\}$.

The preceding algorithm is nonincremental in the sense that it completely recomputes AS(t_p^*) without ever using any of the information in AS(t_p). The question facing us is: How should we compute AS(t_p^*) from AS(t_p) without re-doing the whole linear program computation described in the proof of Theorem 2.2?

First, we assume without loss of generality that *exactly* one V_i is increased by *exactly one added element*. In other words, there is some $1 \leq i \leq n$ such that $V_i^* = V_i \cup \{e\}$ and for all $j \neq i$, $V_j^* = V_j$. Hence, e is the element being added in this case. We may now proceed as follows.

Incremental Annotated Representation Algorithm (IARA)

(1) Let $W = \{w_1, \ldots, w_m\}$ be the set of all t_p-worlds. Clearly, $m = card(V_1) \times \cdots \times card(V_n)$. Let $m' = card(V_1) \times \cdots \times card(V_{i-1}) \times card(V_{i+1}) \times \cdots \times card(V_n)$. Let $e_0 \in V_i$ be any arbitrary, but fixed element of V_i. Let $W_0 = \{w_1, \ldots, w_m\}$ be the set of all worlds in W having the ith component of the world equal to e_0.

(2) For each $1 \leq j \leq m'$, construct a *twin world* w_j^b which is exactly like w_j except that its ith component is e (the plain component being modified) instead of e_0. Let $W^* = W \cup \{w_1^b, w_2^b, \ldots, w_m^b\}$.

(3) If $h_i^*(e) = [\alpha, \beta]$, then construct a set LP* of constraints by *editing* LP(t_p) in the following way.

(a) For each constraint of the form $\alpha_0 \leq \Sigma_{r \in \mathscr{A}} Z_r \leq \beta_0$ generated by V_j and v in Step (1) of the proof of Theorem 2.2, where $j \neq i$, replace the preceding constraint by the constraint:

$$\alpha_0 \leq \left(\sum_{r \in \mathscr{A}} (z_j) + \left(\sum_{w_s^b\text{'s } j\text{th component is } v} z_s^b \right) \right) \leq \beta_0 .$$

(Recall that $\mathscr{A}$ varies over *world-ids*.) Note that in this step, constraints generated in Step (1) of the proof of Theorem 2.2 are *not* placed in LP* if they are generated by elements of V_i, the component being updated.

(b) Add the constraint: $\alpha \leq \Sigma_{j=1}^{m'} z_j^b \leq \beta$.

(c) Finally, replace the constraint generated in Step (3) of the construction of LP(t_p) in the proof of Theorem 2.2 by the constraint: $(\Sigma_{j=1}^{m} z_j) + (\Sigma_{j=1}^{m'} z_j^b) \leq 1$.

(d) All other constraints in $LP(t_p)$ remain unchanged. (These include the constraints specifying for instance, that $z_j^b \leq 1$, etc.)

(4) For each $1 \leq j \leq m'$: **minimize** z_j^b and **maximize** z_j^b w.r.t. the constraints in LP* to get the lower and upper bounds associated with the annotated type w_j^b.

Table XI. Simple probabilistic tuple t_1

a: [0.1, 0.5]	c: [0.3, 0.6]
b: [0.4, 0.8]	

(5) For each $w_j \in W$, **minimize** z_j and **maximize** z_j w.r.t. the constraints in LP* to get the bounds on the annotated tuple w_j.

The **IARA** algorithm is an improvement on the **NAIVE-INS** algorithm in a number of ways, as follows.

(1) First, instead of recomputing the set of worlds, **IARA** makes effective use of the existing worlds in W and merely modifies some of those worlds by substituting their ith data component by the new element e. This occurs in Steps 1 and 2 of the **IARA** algorithm.
(2) Second, we do not construct LP(t_p^*) from scratch. Instead, in Step 3 of the **IARA** algorithm, the existing linear program LP(t_p) is *edited* into a new linear program LP*.
(3) Third, the structure of the linear program LP* is hierarchical—one can first optimize the variables z_j^b using the constraints defined in Steps 3(b)–(c) and then optimize the variables z_j using the constraint defined in Step 3(a). It is well known (cf. Hiller and Liebermann [1974]) that computing linear programming problems with hierarchical structure is usually easier than those linear programs where the linear program cannot be broken up into hierarchies. The reason for this is that one subset of the linear program may "fix" or "determine" the values of certain variables, and those variables may then be eliminated from other constraints at higher levels. This is called "monotone variable elimination" and has proved very effective in linear programming computations of deductive databases [Bell et al. 1994].

We now present a simple example showing how the **IARA** algorithm works.

Example 4.2 Consider the simple probabilistic tuple t_1 shown in Table XI. Initially, $W = \{w_1, w_2\}$ is the set of t_1-worlds and they are defined as follows: $w_1 = (a, c)$, $w_2 = (b, c)$. AS(t_1) is given in Table XII. Suppose we now want to add an extra possibility; that is, the second argument of tuple t_1 could be *either* c (with probability [0.3, 0.6]) or d (with probability [0.2, 0.3]). Thus, t_1^* is given in Table XIII. In this case, we have four worlds altogether, including one twin for each of w_1, w_2. Note that these four worlds are not computed from scratch, just by modifying w_1, w_2. Here, $w_1^b = (a, d)$ and $w_2^b = (b, d)$.

In this case, LP* constructed by the **IARA** algorithm is as follows.

(a) Step 3(a) places the following constraints in LP*.

$$0.1 \le z_1 + z_1^b \le 0.5 \quad 0.4 \le z_2 + z_2^b \le 0.8$$

Table XII. t_1-worlds

Arg1	Arg2	ℓ	u	path
a	c	0.1	0.2	w_1
b	c	0.4	0.5	w_2

Table XIII. t_1^*

a: [0.1, 0.5]	c: [0.3, 0.6]
b: [0.4, 0.8]	d: [0.2, 0.3]

(b) Step 3(b) places the following constraints in LP*.

$$0.2 \leq (z_1^b + z_2^b) \leq 0.3.$$

(c) Step 3(c) places the following constraints in LP*.

$$z_1 + z_1^b + z_2 + z_2^b \leq 1.$$

(d) Note that in addition to the preceding LP* contains the constraint $0.3 \leq z_1 + z_2 \leq 0.6$. This constraint was present in LP(t_p) and is left unchanged; that is, it is not modified when the IARA algorithm performs its editing.

We now minimize and maximize z_1, z_2, z_1^b, and z_2^b to get the final result.

The **IARA** algorithm basically edits the linear program LP(t_p) into a new linear program LP* that is identical to LP(t_p^*) where t_p^* is the tuple obtained by plain component addition of a single component to the (old) tuple t_p. As a consequence of this identity, it follows immediately that:

(1) If w is the wid of a world t_w in W^* and **IARA** assigns $[\ell, u]$ to w, then the annotated tuple (t_w, ℓ, u, w) is in AS(t_p^*).

(2) If the annotated tuple (t, ℓ, u, w) is in AS(t_p^*), then w is a world in W^* and **IARA** assigns $[\ell, u]$ to w.

Before proceeding to develop techniques to handle other kinds of probabilistic updates (e.g., component addition/deletion with probability updates, plain component deletion, etc.), we revisit our assumption at the beginning of the section on plain component addition. We had assumed that *exactly* one V_i is increased by *exactly one added element*. In other words, there is some $1 \leq i \leq n$ such that $V_i^* = V_i \cup \{e\}$ and for all $j \neq i$, $V_j^* = V_j$. If more than one V_i is expanded, or more than one component is added to V_i, then this situation can be easily handled by calling the **IARA** algorithm iteratively, one at a time.

4.3.2 *PT-Modification: Component Addition with Probability Update.* As in the case of plain component addition, we assume, without loss of generality, that *exactly* one V_i is increased by *exactly one added element*.

Table XIV

a: [0.1, 0.5]	c: [0.4, 0.5]
b: [0.4, 0.8]	d: [0.2, 0.3]

However, unlike the situation with plain component addition, the probability assignments to the old elements of V_i may change.

Example 4.3 Let us return to the case of Example 4.2. Suppose we add the component d to the second field of the relation (as we did in Example 4.2) but, in addition, we change the probability of c (the one other element in this field) from [0.3, 0.6] to [0.4, 0.5]. Thus, the new probabilistic tuple in question is as shown in Table XIV. Suppose $w_1 = (a, c)$, $w_2 = (a, d)$, $w_3 = (b, c)$, and $w_4 = (b, d)$. In Step (1) in the construction of $\mathsf{LP}(t_p)$, $V_2 = \{c, d\}$ and c generated a constraint, viz. $0.4 \le w_1 + w_3 \le 0.5$. All that needs to be done is to edit this constraint to the new constraint: $0.3 \le w_1 + w_3 \le 0.6$ and re-optimize. Experimental evidence (Bell et al. [1994]) suggests that such algorithms are extremely efficient.

The observant reader will notice that now in order to handle the simultaneous insertion of a tuple and the adjustment of probabilities, we basically need to execute two steps:

—modify $\mathsf{LP}(t_p^*)$ to *first* incorporate all probability updates, and

—subsequently invoke the IARA algorithm to handle the plain component additions, working on the modified version of $\mathsf{LP}(t_p^*)$.

Modified IARA Algorithm (m-IARA).

(1) Let affected be the set of all elements e' in V_i such that $h(e') \neq h^*(e')$.

(2) If $e' \in$ affected causes the constraint $\alpha \le \Sigma_{j\in\mathcal{A}} z_j \le \beta$ to be added by Step (1) of the proof of Theorem 2.2, then edit this constraint to: $\alpha' \le \Sigma_{j\in\mathcal{A}} z_j \le \beta'$, where $h^*(e') = [\alpha', \beta']$.

(3) Iteratively execute the IARA algorithm to handle all plain component insertions.

Like **IARA**, the **m-IARA** algorithm judiciously edits the affected parts of $\mathrm{LP}(t_p)$ so that:

(1) if t_w is a world in W^* with wid w, and **m-IARA** assigns $[\ell, u]$ to w, then the annotated tuple (t_w, ℓ, u, w) is in $\mathsf{AS}(t_p^*)$;

(2) if the annotated tuple (t_w, ℓ, u, w) is in $\mathsf{AS}(t_p^*)$, then w is the id of a world in W^* and **m-IARA** assigns $[\ell, u]$ to w.

4.3.3 *Plain Component Deletion.* Plain component deletion corresponds to the situation where a component in a probabilistic relation is being deleted, but the probability assignments associated with the "diminished" probabilistic tuple remain unaffected by this deletion. In this section, we show how plain component deletions may be neatly implemented. We first provide a **NAIVE-DEL** algorithm, followed by a more sophisticated Incremental Annotated Representation Deletion Algorithm (**IARD** Algorithm).

Suppose $t_p = (\mathbf{v_1}, \ldots, \mathbf{v_n})$ is a probabilistic tuple where each $\mathbf{v_i} = \langle V_i, h_i \rangle$. Let $t_p^\bullet$ be the tuple defined as $t_p^\bullet = (\mathbf{v_1^\bullet}, \ldots, \mathbf{v_n^\bullet})$, where $\mathbf{v_j^\bullet} = \langle V_j^\bullet, h_j^\bullet \rangle$. As in the case of plain component addition, we assume, without loss of generality, that there exists *exactly one* V_i such that $V_i^\bullet = V_i - \{e\}$ for some $e \in V_i$; for all $j \neq i$, $V_j = V_j^\bullet$. Furthermore, for all j, and for all $x \in V_j^\bullet$, $h_j(x) = h_j^\bullet(x)$. Thus, the only case where one of the $h_j^\bullet$ assignment differs from that of j_j is when $j = i$ and the component in question is e.

NAIVE-DEL Algorithm: This algorithm is *identical* to the **NAIVE-INS** algorithm, the only difference being that they are called with different arguments. In the **NAIVE-INS** algorithm, t_p^* reflected a component insertion, whereas now, $t_p^\bullet$ reflects a component deletion.

The **NAIVE-DEL** algorithm corresponds to full recomputation of the annotated representation of $t_p^\bullet$ without using the existing annotated representation of t_p. However, making effective use of the current annotated representation of t_p may significantly speed up the computation of the annotated representation of $t_p^\bullet$.

The Incremental Annotated Representation Deletion (IARD) Algorithm.

(1) Let $W = \{w_1, \ldots, w_m\}$ be the set of all t_p-worlds. Clearly, $m = card(V_1) \times \cdots \times card(V_m)$. Let $W_0 = \{w_{r+1}, \ldots, w_m\}$ be the set of all worlds in W having the ith component equal to e, the component being deleted.

(2) Let $\mathsf{LP}^\bullet$ be the linear program obtained from $\mathsf{LP}(t_p)$ by:
   (a) deleting all constraints in $\mathsf{LP}(t_p)$ that were placed there when Step (1) in the proof of Theorem 2.2 was applied to V_i and the element e;
   (b) eliminating all occurrences of w_j for $j = (r + 1), \ldots, m$ from the remaining constraints.

(3) For each $1 \leq j \leq r$: **minimize** and **maximize** z_j subject to $\mathsf{LP}^\bullet$.

Unlike the case of insertion, deletion affords fewer possibilities for optimization. The reason for this is that the deletion of a constraint opens up possibilities for solutions, thus both further decreasing lower bounds as well as increasing upper bounds. The **IARD** algorithm may be extended to handle deletions combined with modifications in probability assignments in the obvious way.

4.4 Algorithms for View Maintenance on Annotated Relations

Suppose a probabilistic tuple t_p in probabilistic relation R is modified to t_p^*. Then we replace $\mathsf{AS}(t_p)$ by $\mathsf{AS}(t_p^*)$. This replacement may cause drastic changes in the views developed on top of $\mathsf{AS}(t_p)$.

Example 4.4 Consider the view V defined by the query: Find the locations and bands of all records associated with radars of type 1. This can be expressed as:

$$\pi_{\mathtt{LOC,BAND}}(\sigma_{\mathtt{Obj=radar\_type1}}(\mathtt{target})).$$

Table XV(a)

LOC	BAND	LB	UB	PATH
site1	750	0.4	0.7	w_1
site1	800	0.5	0.9	w_2
site2	700	0.8	0.9	w_3
site3	700	0	0.5	w_5
site3	750	0	0.4	w_6

Table XV(b)

LOC	BAND	LB	UB	PATH
site1	750	0.4	0.7	w_1
site1	800	0.5	0.9	w_2
site2	700	0.8	0.9	w_3
site3	700	0.4	0.7	w_5

Table XV(a) shows the *materialization* of this query. Now suppose that the original relation `target` was updated so that band of 750 KHz was deleted from the third tuple (denoted t_3) in the probabilistic relation, `target`, associated with Site 3. Table XV(b) shows the new annotated relation resulting from this deletion. It is very significant to note that the probability ranges associated with the tuple having wid w_5 has changed as a result of the update.

Suppose *Ins* is a set of annotated tuples being inserted into an annotated relation R, and *Del* is a set of annotated tuples being simultaneously deleted from the annotated relation R. Without loss of generality, we will assume that no two tuples in *Ins* (resp., *Del*) are data-identical. Let V be a view defined and $\mathcal{M}_{\text{old}}(V)$ be the materialization of that view prior to the update. The *Full Annotated View Maintenance* (*FAVM*) Algorithm provides a way of handling such updates. Before defining the full algorithm, we define two simpler algorithms. The first algorithm deals only with views of the form $\sigma_{\text{Cond}}(R)$; such views are called *selection-views* and the resulting algorithm is called the Annotated Selection View Maintenance Algorithm. The second algorithm deals with views of the form $\kappa(\sigma_{\text{Cond}}(R))$. These views are termed *selection-compaction views*.

Annotated Selection View Maintenance (ASVM) Algorithm.

(1) $\mathcal{M}_{\text{new}}(V) = \mathcal{M}_{\text{old}}(V)$.

(2) *Deletion Step*: For each annotated tuple $t_a \in Del$, find all tuples $t'_a \in \mathcal{M}_{\text{new}}(V)$ that are data-identical to t_a. Replace t'_a in $\mathcal{M}_{\text{new}}(V)$ by $(t'_a - t_a)$ where "$-$" denotes the generic difference between the relation containing the single tuple t_a and the relation containing the single tuple t'_a. In the future, when we apply the generic difference operator—to tuples, we do so with this interpretation.

(3) *Addition Step*: For each annotated tuple $t_a \in Ins$ do: If t_a satisfies the query V, then add it to M_{new}, else do nothing.
(4) *Return*: $\mathcal{M}_{\text{new}}(V)$ as the new materialized view.

Annotated Selection Compaction View Maintenance (ASCVM) Algorithm.

```
(1) $\mathcal{M}_{\text{new}}(V) = \mathcal{M}_{\text{old}}(V)$.
(2) Deletion Step: For each annotated tuple $t_a \in Del$, find all tuples $t'_a \in \mathcal{M}_{\text{new}}(V)$ that are data-identical to $t_a$. Replace $t'_a$ in $\mathcal{M}_{\text{new}}(V)$ by $(t'_a - t_a)$.
(3) Addition Step: For each annotated tuple $t_a \in Ins$ do:
    (a) If $t_a$ satisfies the query $V$, then:
        i. Replace all tuples $t'_a \in \mathcal{M}_{\text{new}}(V)$ that are data-identical to $t_a$ by $(t_a \oplus t'_a)$.
        ii. If no such data identical tuples exist, then $\mathcal{M}_{\text{new}}(V) := \mathcal{M}_{\text{new}}(V) \cup \{t_a\}$.
    (b) Otherwise, do nothing.
(4) Return: $\mathcal{M}_{\text{new}}(V)$ as the new materialized view.
```

The **ASCVM** algorithm is incremental in many ways (similar comments apply to the **ASVM** algorithm).

(1) When a deletion is performed, there is no need to explicitly check to see whether the tuples being deleted are in the view or if they satisfy the view criteria. Instead, deletions are incorporated by applying the generic difference operator to data-identical tuples in the materialization of the view.
(2) The query V is never re-evaluated against all relations; instead, we merely check if the annotated tuple being inserted satisfies the view definition.
(3) In cases when the preceding test succeeds, the materialized view is directly manipulated, using the $\oplus$ operator.
(4) The entire algorithm runs in time $O((card(Ins) + card(Del)) \times card(\mathcal{M}_{\text{old}}(V)))$, that is, in time proportional to the product of the materialized view size and the number of insertions/deletions.

The **ASVM** algorithm efficiently handles insertions, deletions, and modifications in a sound and complete manner.

In particular, suppose t_p is a probabilistic tuple in (probabilistic) relation R, and t_p^* is an updated version of t_p that we wish to construct. Suppose V is an annotated "selection" view defined on $\mathsf{AS}(R)$ and $\mathcal{M}_{\text{old}}(V)$ is its materialization. Then the materialized view, $\mathcal{M}_{\text{new}}(V)$, defined by the **ASVM** algorithm, coincides with the materialization of V with respect to $((\mathsf{AS}(R) - \mathsf{AS}(t_p)) \cup \mathsf{AS}(t_p^*)$.

Table XVI shows how we may extend the **ASVM** algorithm to view maintenance when the views are not just defined by Cartesian product operations. In each case, we have a relation of the form (R *op* S) where *op* is either $\times$, U, $-$. In each case (as indicated by the second column) one of

Table XVI.

View	Relation Updated	Algorithm
$R \times S$	R	1. $\Delta^+ = Ins \times S$.
		2. $\Delta^- = Del \times S$.
		3. Return $\mathcal{M}_{\text{new}}(V) = (\mathcal{M}_{\text{old}}(V) - \Delta^-) \cup \Delta^+$.
$(R - S)$	R	1. $\Delta^+ = Ins - S$.
		2. Return $\mathcal{M}_{\text{new}}(V) = (\mathcal{M}_{\text{old}}(V) \cup \Delta^+) - Del$.
$(R - S)$	S	1. $\mathcal{M}_{\text{new}}(V) = (\mathcal{M}_{\text{old}}(V) - Ins) \cup Del$.
$(R \cup S)$	R	1. $\mathcal{M}_{\text{new}}(V) = \mathcal{M}_{\text{old}}(V) \cup Ins$.
		2. $Temp = S \cap Del$.
		3. $\mathcal{M}_{\text{new}} = (\mathcal{M}_{\text{new}} - Del) \cup Temp$.

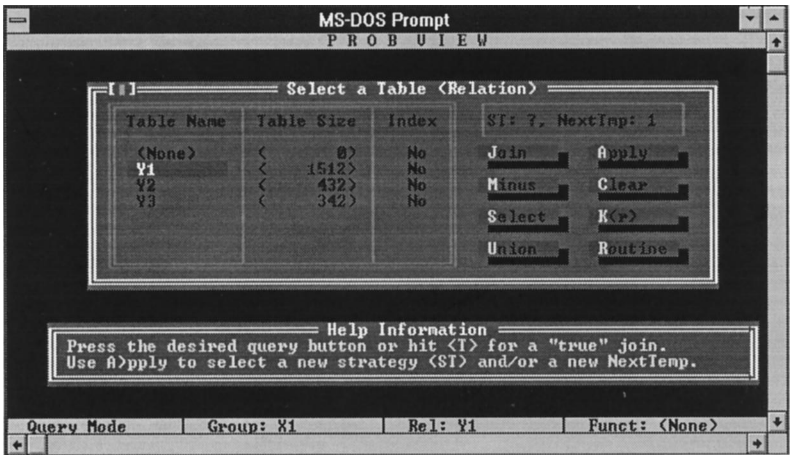
these two relations is being updated by the insertion of a set *Ins* of tuples, and the deletion of a set *Del* of tuples.

We have now defined how to maintain materialized views, for each possible algebraic operator that is the leading connective in the view definition.

5. IMPLEMENTATION AND EXPERIMENTS

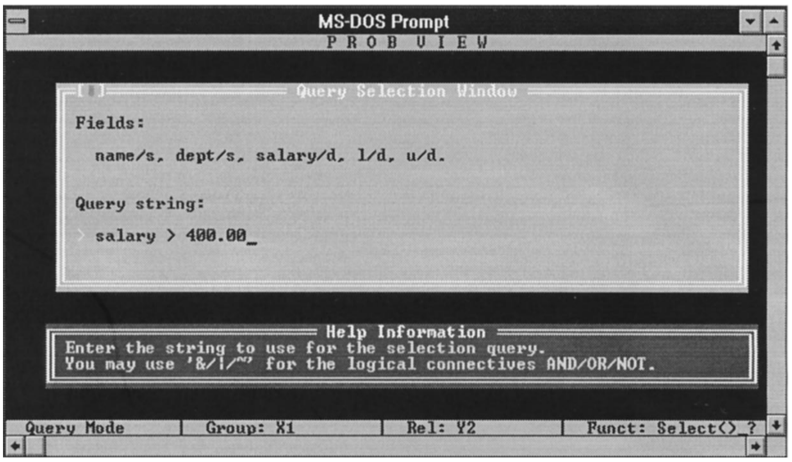
ProbView is built on top of DBASE V.0, and is implemented on the PC/Windows platform in C and comprises approximately 5100 lines of C code. Figure 1 shows one **ProbView** window where a user may select a table, select an operation, and then perform further operations depending on the choices made so far. For example, if the table Y1 is selected, and if the user subsequently clicks on union, he would be expected to select another table. The result is a materialized view that is stored in a temporary table that the user may rename. On the other hand, instead of a union, suppose the user wishes to perform a SELECT. Figure 2 shows the selection query window in which the user expresses her selection condition (in this case, Salary $>$ 400. Various other options exist in the implementation. The user can see these options in Figures 1 and 2. We are currently preparing documentation for the **ProbView** software and expect to make it available through the WWW soon.

We report on a number of experiments that we carried out. Unless explicitly mentioned otherwise, in all experiments the sizes of the relations were varied from 500 to 5,000 tuples. As the evaluation of traditional database operations is not the main focus of this experiment, we assumed (for the purposes of experimentation) that all tuples are of the form (d, ℓ, u, p), where d is a randomly generated integer. Note that the **ProbView** system itself supports arbitrary tuples (i.e., d can be any ordinary tuple). This allows us to focus on the unique aspects of the **ProbView** system, viz. its probabilistic components. When conducting experiments, we randomly generated the value of d from between 1 to 20,000. As d gets larger, the "duplication factor" (number of data-identical tuples) gets smaller.



```
Select a Table (Relation)

Table Name   Table Size   Index      ST: ?, NextTmp: 1

 (None)      (      0)     No
 Y1          (   1512)     No
 Y2          (    432)     No
 Y3          (    342)     No

Help Information
Press the desired query button or hit <T> for a "true" join.
Use A)pply to select a new strategy (ST) and/or a new NextTemp.

Query Mode | Group: X1 | Rel: Y1 | Funct: (None)
```

Figure 1



```
Query Selection Window
Fields:

  name/s, dept/s, salary/d, l/d, u/d.

Query string:

> salary > 400.00_

Help Information
Enter the string to use for the selection query.
You may use '&/|/~' for the logical connectives AND/OR/NOT.

Query Mode | Group: X1 | Rel: Y2 | Funct: Select()_?
```

Figure 2

5.1 Experiment: Interaction of Compaction and Union

We first attempted to evaluate the effect of increased duplication factors on the interaction of compaction and union. The graphs in Figure 3 show the computation of the queries in Table XVII when the data values range from [1, . . . , 10] to [1, . . . , 200]. The number of tuples in relations $r1$ and $r2$ are varied from 500 to 5000 each. Table XVII shows the legend used in the graphs of Figure 3. The indexing referred to is due to the fact that Dbase V.0 allows relations to be stored either in indexed or nonindexed form. In the former, we have options as to which attributes to index. The reader will easily observe the following points:

(1) In all cases, $\kappa(r_1 \cup r_2)$ took significantly longer to compute than the equivalent expression, $\kappa(\kappa(r_1) \cup \kappa(r^2))$.

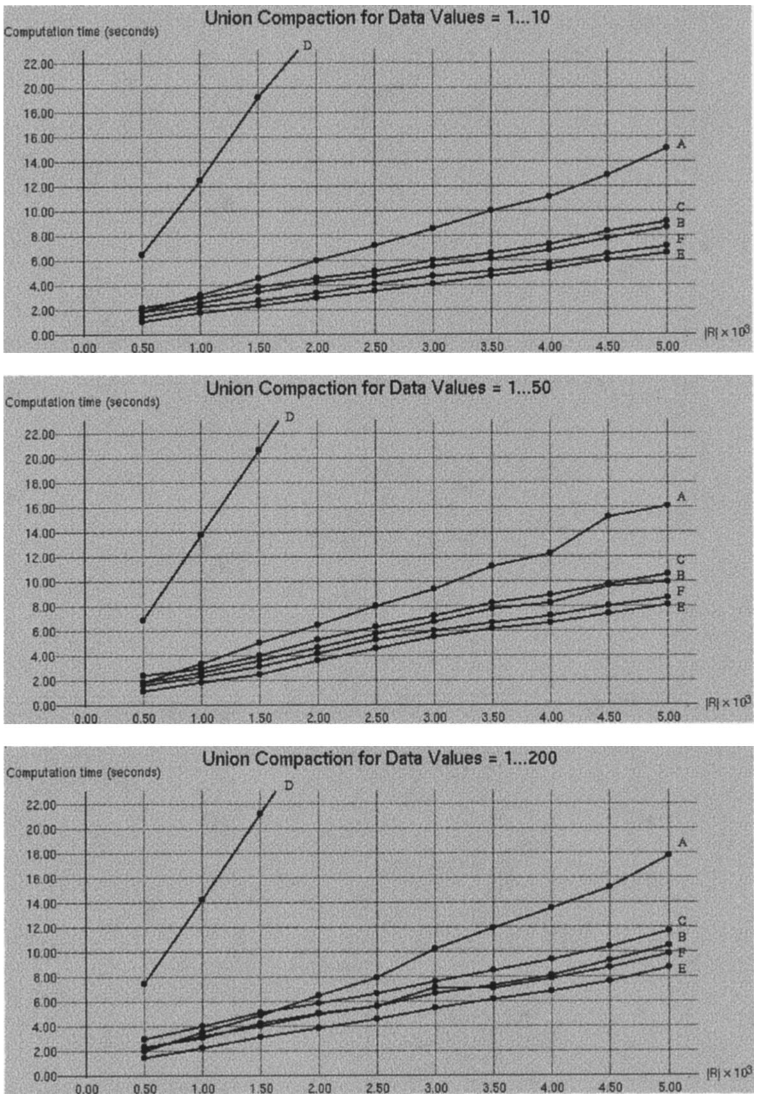


Fig. 3. Union compaction.

Table XVII

Expression	Legend without Indexing	Legend with Indexing
$\kappa(r_1 \cup r_2)$	A	D
$\kappa(r_1) \cup \kappa(r_2)$	B	E
$\kappa(\kappa(r_1) \cup \kappa(r_2))$	C	F

(2) The *approximation* $\kappa(r_1) \cup \kappa(r_2)$, in all cases, took the least time, but performing an extra round of compaction to compute the expression $\kappa(\kappa(r_1) \cup \kappa(r_2))$ seemed to only take marginally longer.

(3) As the space of data values increased (and hence the duplication of data-identical tuples in the relations decreased), the time taken to compute $\kappa(r_1 \cup r_2)$ increased in an exponential fashion. In contrast, the graphs of both $\kappa(\kappa(r_1) \cup \kappa(r_2))$ and the approximation $\kappa(r_1 \cup r_2)$ rose in a linear fashion with the number of tuples.

Conclusion. We conclude that any query of the form $\kappa(r_1 \cup r_2)$ should be rewritten to the equivalent query $\kappa(\kappa(r_1) \cup \kappa(r_2))$ because (a) this appears to be computationally much more efficient to compute, and (b) the fact that it is equivalent to the original query $\kappa(r_1 \cup r_2)$ seems to offset the minor efficiency gains realized by computing the approximation $\kappa(r_1) \cup \kappa(r_2)$ instead.

5.2 Experiment: Interaction of Compaction and Selection

In this experiment, we studied the effects of pushing selections through compaction. As in the previous case, we studied two query algebraic expressions, $\sigma_F(\kappa(r))$ and $\kappa(\sigma_F(r))$ where F is a selection condition. We studied the time taken to compute these expressions when three parameters were varied:

—The *space of data values* ranged from [1, . . . , 10] to [1, . . . , 200] and

—the *selectivity* of the selection criterion varied to return more and more tuples. Figure 4 shows the graphs associated with this experiment. The number T at the top of these graphs indicates the selection condition. For instance, $T = 10$ indicates that we selected all tuples where the data field was greater than 10. Similarly, if $T = 20$, then this means that we selected all tuples where the data field was greater than 20. Thus, as T is increased, the number of tuples returned by the selection monotonically decreases.

—Once the set of data values and the number T were fixed, the number of tuples in relation r was varied from 500 to 5,000 in each experimental run.

The legend used in the graph of Figure 4 is shown in Table XVIII. The graphs in Figure 4 indicate that when the data value space is held fixed and the selectivity is decreased (i.e., T is increased) $\sigma(\kappa(r))$ takes significantly more time to compute than $\kappa(\sigma(r))$; that is, *as selectivity decreases, pushing selection inside compaction seems to be a good idea.*

Conclusions. Pushing selection inside compaction seems to enhance the performance of the system. Thus, rewriting the query $\sigma(\kappa(r))$ into the equivalent query $\kappa(\sigma(r))$ seems to lead to performance improvements.

5.3 Experiment: Interaction of Compaction and Cartesian Product

In our third experiment, we attempted to study Cartesian products and their interaction with compaction. Cartesian product is a very important operation because as is well known, the all-important *join* operation can be defined in terms of Cartesian product and selection. In this experiment, we

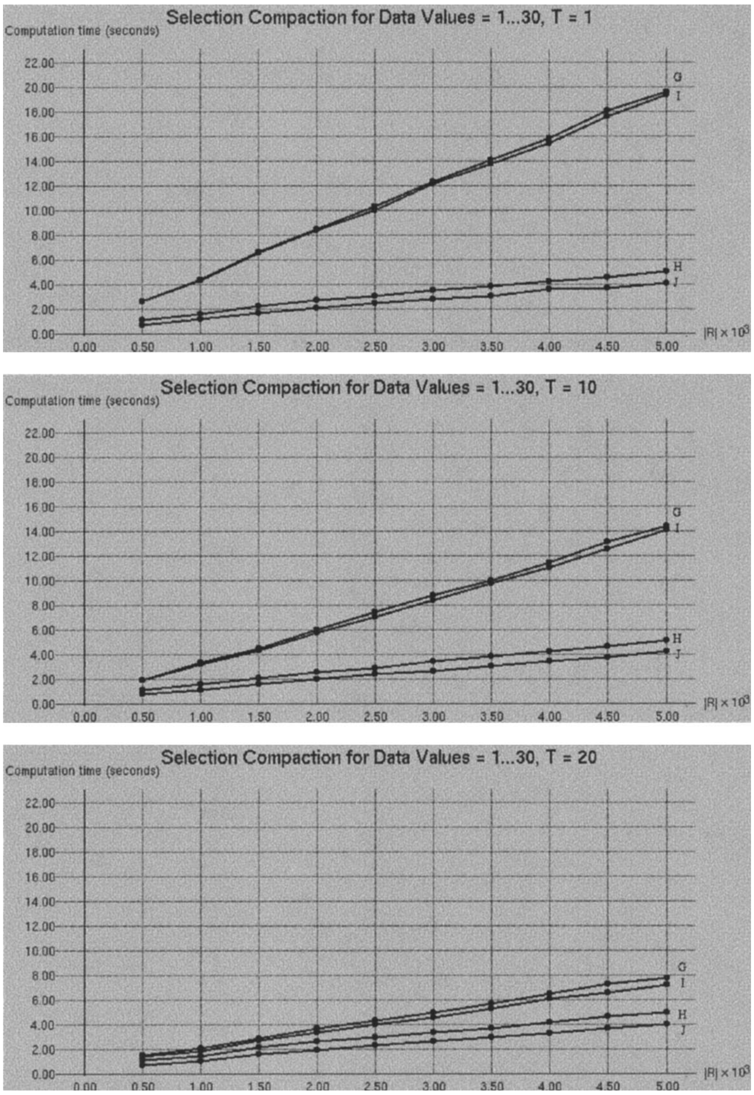


Fig. 4(a). Selection compaction for data values = 1 . . . 30.

Table XVIII

Expression	Legend without Indexing	Legend with Indexing
$\kappa(\sigma(r))$	G	I
$\sigma_F(\kappa(r))$	H	J

evaluated the three expressions listed in Table XIX. the graphs in Figure 5 use these legends. Furthermore, recall that the first and third expressions in the table are equivalent, whereas the second approximates the first and third. The graphs in Figure 5 clearly indicate the following trends.

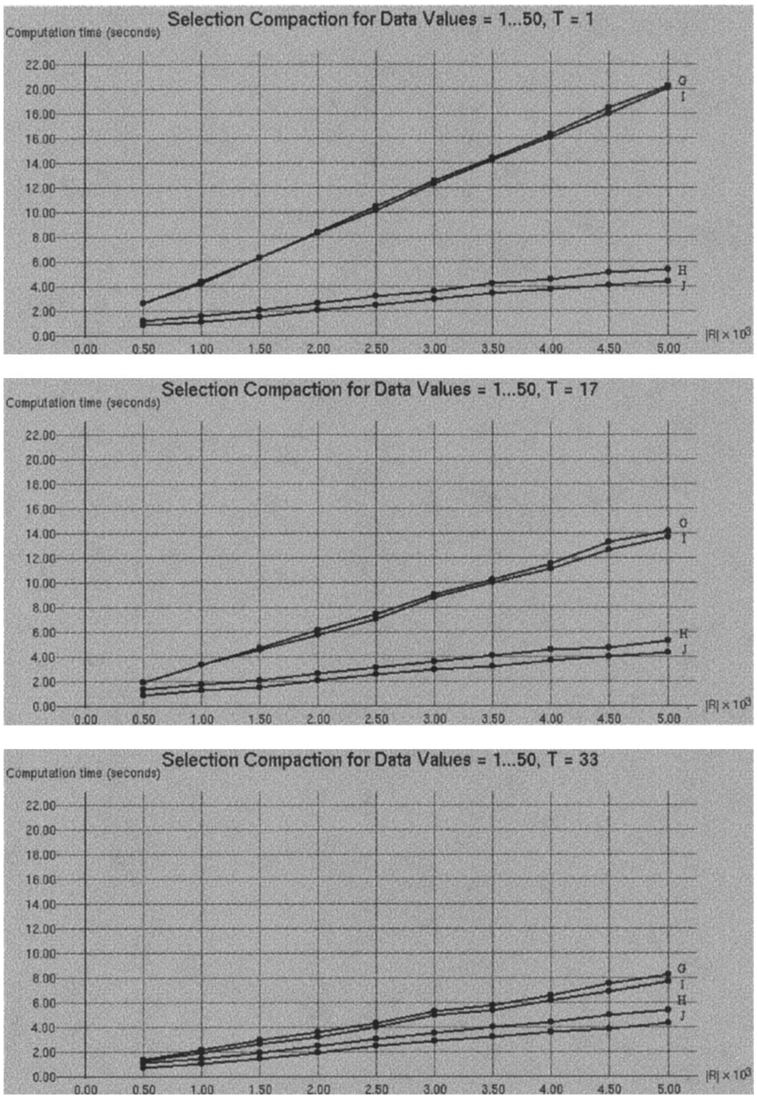


Fig. 4(b). Selection compaction for data values = 1 . . . 50.

(1) Indexing reduces the cost of computing each and every one of these expressions, and
(2) When the degree of duplication is higher (i.e., as the range of data values is "smaller"), $\kappa(r_1) \times \kappa(r_2)$ and $\kappa(\kappa(r_1) \times \kappa(r_2))$ significantly outperform $\kappa(r_1 \times r_2)$. Furthermore, the difference between $\kappa(r_1 \times r_2)$ and $\kappa(\kappa(r_1) \times \kappa(r_2))$ seems to stay relatively small.
(3) However, as the degree of duplication decreases (see the graphs where Data Values = [1, . . . , 5,000] and bigger), the situation is *flipped around* and $\kappa(r_1 \times r_2)$ significantly outperforms both $\kappa(r_1) \times \kappa(r_2)$ and $\kappa(\kappa(r_1) \times \kappa(r_2))$.

Table XIX.

Expression	Legend without Indexing	Legend with Indexing
$\kappa(r_1 \times r_2)$	K	N
$\kappa(r_1) \times \kappa(r_2)$	L	O
$\kappa(\kappa(r_1) \times \kappa(r_2))$	M	P

Conclusion. We conclude that any query of the form $\kappa(r_1 \times r_2)$ should be rewritten to the equivalent query $\kappa(\kappa(r_1) \times \kappa(r_2))$ when the duplication factor in the relations r_1, r_2 is relatively high. However, when this is not the case, then we should not do so.

Before closing this section, we remark that we also experimented on the interaction of compaction and difference. The conclusion of the experiment is that performing compaction *before* computing difference leads to substantial performance gains. The full details of the experiment can be found in Lakshmanan et al. [1996].

6. RELATED WORK

Though there has been extensive work on probabilities in the AI community, relatively little work has been done on probabilistic databases.

Kiessling and his group [Kiessling et al. 1992; Thone et al. 1995; Schmidt et al. 1987] have developed a framework called DUCK for reasoning with uncertainty. They provide an elegant logical axiomatic theory for uncertain reasoning in the presence of rules. In the same spirit as Kiessling et al., Ng and Subrahmanian [1993, 1995] have provided a probabilistic semantics for deductive databases; they assume absolute ignorance, and furthermore, assume that rules are present in the system. In contrast, in our framework, rules are not present, rather our interest is in extending the relational algebra to capture probabilistic information. In addition, our approach provides a single unified way of handling multiple probabilistic combination strategies, whereas their work assumes either probabilistic independence [Kiessling et al. 1992, p. 421][4] or total ignorance [Ng and Subrahmanian 1993, 1995]. Lakshmanan and Sadri [1994a] show how a few selected probabilistic strategies extend the previous probabilistic models. In contrast, in this article, we only specify *axioms* that such strategies should satisfy—the user may then pick strategies that accurately reflect the application domain. Furthermore, we treat difference, an operation not treated in Lakshmanan and Sadri [1994a], and introduce a unique notion of *path*. All our query equivalences, and view maintenance results are new. Finally, our **ProbView** system is the first of its kind.

Barbara et al. [1992] develop a probabilistic data model and propose probabilistic operators. Their work is based on the following assumptions, none of which is needed by us.

[4]To be precise, some deduction rules of the DUCK calculus rely on independence, whereas others do not.

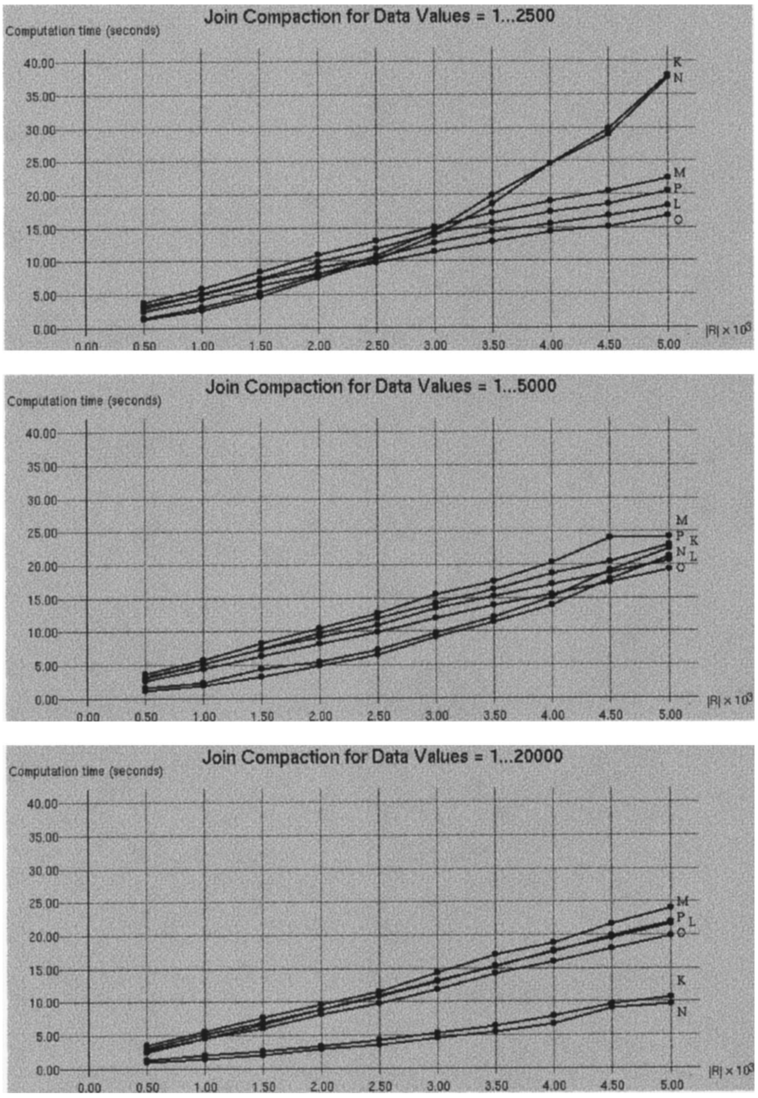


Fig. 5. Join compaction.

(1) A central premise in their model is that every probabilistic relation has a set of *deterministic* attributes forming the key of the relation. By contrast, we make no such assumption, although it can be realized as a special case of our framework.
(2) They use only discrete probabilities, which in effect amounts to assuming that probabilities of compound events can always be precisely determined, an assumption valid for few combination strategies. In contrast, we allow interval probabilities, allowing for margins of error in the probability data.

(3) When performing joins, they assume that Bayes' rule applies (and hence, as they admit up front, they make the assumption that all events are independent).

Also, as they point out, unfortunately their definition leads to a "lossy" join. In contrast, as we have demonstrated, we have removed unnecessary assumptions, supported a wide class of combination strategies for conjunction (and disjunction and negation), while avoiding the problem of lossy joins.

Cavallo and Pittarelli [1987] propose a model for probabilistic relational databases. In their model, tuples in a probabilistic relation are interpreted using an exclusive or, meaning at most one, data-tuple is assumed to be present in the underlying classical relation. This is a rather restrictive assumption, and we make no such assumptions.

Several other authors have handled uncertainty in databases through the use of fuzzy sets [Dubois and Prade 1988; Kifer and Li 1988; Raju and Majumdar 1988]; as the differences between probabilities and fuzzy sets are well known, we do not address these works extensively here.

Finally, the idea of compaction is similar in spirit to some analogous operations defined in quite different settings. First, in the context of relational databases with null values, Imielinski and Lipski [1984] consider a number of alternate representation systems for databases with nulls. Among others, they introduce C-tables. C-tables have an extra attribute named Con representing conditions associated with nulls. Each tuple in a C-table can have entries corresponding to marked nulls, and its value under column Con represents conditions associated with that tuple. The possibility of "collapsing" several tuples which are "data-identical" by taking the disjunction of their Con-value has been noticed by the authors. They also show that under closed world assumption, C-tables lead to a faithful representation system for the complete relational algebra. There are several important differences (1) the algebraic operators are fixed, not generic as in our framework; (2) whereas the algebra defined in Imielinski and Lipski [1984] does not manipulate any numerical measures, our algebra manipulates probabilities associated with tuples; and (3) as the authors themselves point out, the framework of C-tables is mainly of theoretical interest owing to the high complexity associated with it. By contrast, we have proposed a practical framework for probabilistic databases, established that the complexity of query processing is tractable under most practical circumstances, and also established the practicality of our framework with a concrete implementation and with performance results. Second, in the area of temporal databases (e.g., Gadia [1988]), we may often want to merge multiple tuples, for instance, if we have one tuple t_1 valid (at all times) during the interval Jan.–June 1995, and another data-identical tuple t_2 valid from July–Dec. 1995, then we may want to merge these tuples into a single tuple labeled with the interval 1995. Our framework is obviously very different from this.

7. CONCLUSIONS

In conclusion, we reiterate a simple point: *probabilities are very complex, but very powerful and useful for modeling uncertainty*. Computing the probabilities of complex events from the probabilities of elementary events depends critically on the interdependencies between the events, leading to a variety of combination strategies. Past treatments of probabilities in databases have, by and large, made the optimistic assumption that all events are independent—something that is rarely true in the real world and even more rarely so in databases where the presence of various integrity constraints may explicitly encode interdependencies between attributes and hence tuples.

Towards this end, we have proposed a single unified framework within which probabilities occurring in databases can be combined according to the known interdependencies between those events. We have proposed axioms characterizing reasonable probabilistic conjunction and disjunction (and hence negation) strategies. The **ProbView** system we have developed allows the user to pick a different strategy for each operator, per session, or even per query. The default choice is the strategy based on the principle of ignorance. Once specific conjunction and disjunction (and hence negation) strategies are chosen by the user, this automatically induces a probabilistic select, project, Cartesian product, difference, and union and intersection operators (and hence join as well, in terms of Cartesian product and select). The user may construct queries, secure in the knowledge that these queries will be processed according to the semantics of the strategies chosen for conjunction and disjunction (and hence negation). In addition, we have proved equivalence/containment results that hold for *any* probabilistic conjunction and disjunction (and hence negation) strategies that the user might choose, as long as those strategies satisfy our postulates. Consequently, the resulting system is very flexible and robust, and can accurately capture the probabilities in the context of complex queries without making unnecessary assumptions about independence and the like.

Finally, our previous querying operators have been implemented within the **ProbView** system that has been built on top of Dbase V.0, with the help of CodeBase to access the underlying functions in Dbase. Based on this, we have conducted experiments and have established that certain queries may be optimized substantially by rewriting them to a different, although equivalent, form.

ACKNOWLEDGMENTS

We are very grateful to Laura Drake for converting all the graphs to Postscript form. Thanks are due to Rick Snodgrass for pointing out the applicability of probabilistic reasoning for handling temporal indeterminacy.

REFERENCES

BARBARA, D., GARCIA-MOLINA, H., AND PORTER, D. 1992. The management of probabilistic data. *IEEE Trans. Knowl. Data Eng. 4*, 487–502.

BELL, C., NERODE, A., NG, R., AND SUBRAHMANIAN, V. S. 1994. Mixed integer programming methods for computing non-monotonic deductive databases. *J. ACM 41*, 6 (Nov.), 1178–1215.

BLAKELEY, J., COBURN, N., AND LARSON, P.-A. 1989. Updating derived relations: Detecting irrelevant and autonomously computable updates. *ACM Trans. Database Syst. 14*, 3, 369–400.

BOOLE, G. 1854. *The Laws of Thought*. Macmillan, London.

CAVALLO, R. AND PITTARELLI, M. 1987. The theory of probabilistic databases. In *Proceedings of the Conference on Very Large Data Bases* (Brighton, England, Sept. 1–4), 71–81.

DAYAL, U. 1989. Queries and views in a object-oriented databases. In *International Workshop on Database Programming Languages* (Glenden Beach, OR, June 4–8), 80–102.

DAYAL, U. AND HWANG, H. 1984. View definition and generalization for database integration in a multi-database system. *IEEE Trans. Softw. Eng. SE-10*, 6, 628–644.

DUBOIS, D. AND PRADE, H. 1988. Default reasoning and possibility theory. *Artif. Intell. 35*, 243–257.

DUMAIS, S. 1993. LSI meets TREC: A status report. In *Proceedings First Text Retrieval Conference* (Gaithersburg, MD), NIST Special Publication 500-207, 137–152.

DYRESON, C. E. AND SNODGRASS, R. T. 1993. Valid-time indeterminacy. In *Proceedings of the International Conference on Data Engineering* (Vienna, April), 335–343.

FENSTAD, J. E. 1980. The structure of probabilities defined on first-order languages. In *Studies in Inductive Logic and Probabilities*, Vol. 2, R. C. Jeffrey, Ed. University of California Press, Berkeley, CA, 251–262.

GADIA, S. 1988. A homogeneous relational model and query languages for temporal databases. *ACM Trans. Database Syst. 13*, 4, 418–448.

GAREY, M. R. AND JOHNSON, D. S. 1979. *A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York.

GUNTZER, U., KIESLING, W., AND THONE, H. 1991. New directions for uncertainty reasoning in deductive databases. In *Proceedings 1991 ACM SIGMOD* (Denver, CO, May), 178–187.

GUPTA, A., MUMICK, I. S., AND SUBRAHMANIAN, V. S. 1993. Maintaining views incrementally. In *Proceedings 1993 ACM SIGMOD Conference on Management of Data* (Washington, DC), 157–166.

HILLIER, F. AND LIEBERMAN, G. 1974. *Operations Research*. Holden-Day, San Francisco, CA.

IMIELINSKI, T. AND LIPSKI, W. 1984. Incomplete information in relational databases. *J. ACM 31*, 4 (Oct.).

IYENGAR, S. S., PRASAD, L., AND MIN, H. 1995. *Advances in Distributed Sensor Technology*. Prentice-Hall, Englewood Cliffs, NJ.

HAN, J., CAI, Y., AND CERCONE, N. 1992. Knowledge discovery in databases: An attribute-oriented approach. In *Proceedings of the 18th Conference on Very Large Data Bases* (Vancouver, Canada, Aug. 23–27), 547–559.

HARRISON, J. V. AND DIETRICH, S. 1992. Maintenance of materialized views in a deductive database: An update propagation approach. In *Workshop on Deductive Databases, JICSLP* (Washington, D.C., Nov. 1992).

KEMPER, A., KILGER, C., AND MOERKOTTE, G. 1994. Function materialization in object bases: Design, realization, and evaluation. *IEEE Trans. Knowl. Data Eng. 6*, 4 (Aug.).

KIESSLING, W., THONE, H., AND GUNTZER, U. 1992. Database support for problematic knowledge. In *Proceedings EDBT-92* (Vienna), Springer LNCS Vol. 580, 421–436.

KIFER, M. AND LI, A. 1988. On the semantics of rule-based expert systems with uncertainty. In *Second International Conference on Database Theory*, M. Gyssens, J. Paredaens, D. Van Gucht, Eds., Springer Verlag, (LNCS 326), 102–117.

KOLMOGOROV, A. N. 1956. *Foundations of the Theory of Probability*. Chelsea Publishing Co., New York.

LAKSHMANAN, L. V. S. 1994. An epistemic foundation for logic programming with uncertainty. In *Proceedings of the 14th Conference on the Foundations of Software Technology and Theoretical Computer Science* (Madras, India, Dec. 1994).

LAKSHMANAN, L. V. S. AND SADRI, F. 1994a. Probabilistic deductive databases. In *Proceedings of the International Logic Programming Symposium*, (Ithaca, NY, Nov.), MIT Press, Cambridge, MA, 254–268.

LAKSHMANAN, L. V. S. AND SADRI, F. 1994b. Modeling uncertainty in deductive databases. In *Proceedings International Conference on Database Expert Systems and Applications (DEXA '94)* (Athens, Greece, Sept.), LNCS 856, Springer-Verlag.

LAKSHMANAN, L. V. S., LEONE, N., ROSS, R., AND SUBRAHMANIAN, V. S. 1996. ProbView: A flexible probabilistic database system. Tech. Rep. Concordia University and University of Maryland, Dec. (Available by `http://www.cs.umd.edu/users/vs/papers/probpapers.html.`).

LU, J., MOERKOTTE, G., SCHUE, J., AND SUBRAHMANIAN, V. S. 1995. Efficient maintenance of materialized mediated views. In *Proceedings 1995 ACM SIGMOD Conference on Management of Data* (San Jose, CA, May).

NG, R. AND SUBRAHMANIAN, V. S. 1993. Probabilistic logic programming. *Inf. Comput. 101*, 2, 150–201.

NG, R. AND SUBRAHMANIAN, V. S. 1995. Stable semantics for probabilistic deductive databases. *Inf. Comput. 110*, 1, 42–83.

RAJU, K. S. V. S. V. N. AND MAJUMDAR, A. 1988. Fuzzy functional dependencies and lossless join decomposition of fuzzy relational database systems. *ACM Trans. Database Syst. 13*, 2 (June).

SCHMIDT, H., KIESSLING, W., GUNTZER, U., AND BAYER, R. 1987. Combining deduction by uncertainty with the power of magic. In *Proceedings DOOD-89* (Kyoto, Japan), 205–224.

SILBERSCHATZ, A., STONEBRAKER, M., AND ULLMAN, J. D. 1991. Database systems: Achievements and opportunities. *Commun. ACM 34*, 10, 110–119.

THONE, H., KIESSLING, W., AND GUNTZER, U. 1995. On cautious probabilistic inference and default detachment. *Ann. Oper. Res. 55*, 195–224.

VARDI, M. Y. 1985. Querying logical databases. In *Proceedings of the Fourth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, 57–65.

Received May 1996; revised January 1997; accepted January 1997