Uncertain Data: Representations, Query Processing, and Applications

Tingjian Ge, Alex Dekhtyar, and Judy Goldsmith

Abstract. Uncertain data is common in many emerging applications. In this chapter, we start by surveying a few applications in sensor networks, ubiquitous computing, and scientific databases that require managing uncertain and probabilistic data. We then present two approaches to meeting this requirement. In the first approach, we propose a rich treatment of probability distributions in the system, in particular the SPO framework and the SP-algebra. In the second approach, we stay closer to a traditional DBMS, extended with tuple probabilities or attribute probability distributions, and study the semantics and efficient processing of queries.

1 Probabilistic Databases and Their Applications

There is a wide range of emerging applications that produce uncertain data and demand new techniques to manage such data. The mature, industry-standard relational database management systems have a history of about 40 years, but they do not have the capability of managing uncertain or probabilistic data. The applications that are discussed in this chapter are mainly in the areas of sensor networks, ubiquitous computing, bioinformatics and scientific databases. There are many applications (often related to the Internet) that also fall in this domain, such as information extraction and information integration.

In this section, we present several applications where large collections of probabilistic data are acquired, stored, and used. We divide those applications into two categories: sensor networks and ubiquitous computing, and scientific databases.

Tingjian Ge
University of Massachusetts, Lowell, MA, USA
e-mail: ge@cs.uml.edu

Alex Dekhtyar
California Polytechnic State University, San Luis Obispo, CA, USA
e-mail: dekhtyar@calpoly.edu

Judy Goldsmith
University of Kentucky, Lexington, KY, USA
e-mail: goldsmith@cs.uky.edu

DOI: 10.1007/978-3-642-37509-5_4

Within sensor networks, we consider a person-based application, namely monitoring of individual soldiers' physical status in the field, and the larger-grained examples of monitoring and controlling traffic in a large city, and monitoring and controlling power use in a house or community. Within scientific databases, we look at storing and managing astronomical data, microarrays, and proteomics. These are meant to be illustrative examples of a few hot research areas that depend on intelligent handling of probabilistic data, not a comprehensive catalogue of probabilistic database applications.

1.1 Sensor Networks and Ubiquitous Computing

Sensor networks and ubiquitous computing are major trends in modern computing. For example, many smartphones provide location estimates using a variety of sensors, such as GPS, WiFi, and/or cellular triangulation. However, the correctness of the triangulation depends on the proximity of cell towers, and on the local interference. It is thus important to handle any new data management issues that arise from uncertain data. Let us look at some concrete application scenarios.

Soldier Physiologic Status Monitoring

In the Soldier Physiologic Status Monitoring application (Tatbul et al. 2004), sensors are embedded in a "smart uniform" that monitors key biological parameters to determine the physiological status of a soldier. Under the harsh environment of the battlefield, it is crucial that sufficient medical resources reach wounded soldiers in a timely manner. Sensors in a smart uniform monitor thermal signals, hydration levels, cognitive and life signs, and wound levels.

There are a few ways the soldier's physiological states can be estimated with different sensors and with different confidence. An algorithm computes an overall score indicating how much medical attention the soldier needs and how urgent his or her condition is.

Tuple ID	Soldier ID	Time	Location	Score for Medical Needs	Conf.
T1	1	10:50	(10, 20)	49	0.4
T2	2	10:49	(10, 19)	60	0.4
T3	3	10:51	(9, 25)	110	0.4
T4	2	10:50	(10, 19)	80	0.3
T5	4	10:49	(12, 7)	56	1.0
T6	3	10:50	(9, 25)	58	0.5
T7	2	10:50	(11, 19)	125	0.3

Fig. 1 A table generated by sensors monitoring soldiers' needs for medical attention. The Conf. (confidence) attribute is the probability of existence of the tuple.

In a central database, as shown in Figure 1, a table records the information sent out by the sensors in the soldiers' uniforms. Each tuple in the table is one estimate with its related confidence. Sensors might be broken in harsh environments. For high availability, there can be two sets of sensors in a soldier's uniform in case one of them breaks down or loses precision. When each sends out an estimate at about the same time and they are inconsistent, at most one of them can be correct (together they form a discrete distribution with the confidence indicating the weight of each). These estimates may differ considerably due to variations in sensors, possible lost network messages, and different estimation algorithms.

Dynamic Traffic Routing

A few projects in both academia and industry (e.g., the CarTel project at MIT[1] and a product at INRIX[2]) provide traffic-aware routing and traffic mitigation. The idea is that various sensing devices are embedded in the cars that travel on roads and highways in urban areas. Some of the sensors measure the location of the cars (e.g., GPS or WiFi (Thiagarajan et al. 2009)), while others estimate travel delays. A large number of sensors from many cars continuously send data to a server.

The server uses this data to give real-time route planning decisions to drivers (e.g., *what is the quickest way to travel from A to B right now*?). Compared to alternatives such as using a standard online map, the dynamic routing also considers real-time factors such as road accidents and rush-hour traffic.

Such a system uses the travel delays reported in a recent time window to infer the probability distribution of current delay at a road. Due to random factors, the best we can get is a distribution. Similar to Figure 1, a central database contains a relational table with a number of attributes such as *road_ID*, *road_length*, *date*, *time*, *speed_limit*, and *current_delay*. Here, the *current_delay* attribute of a road can be modeled as a probability distribution, which is learned from a set of delay readings sent out from that road. To answer a routing query as given above, the system may need to run a shortest path algorithm over road delays that are probability distributions.

Smart Energy Grids

There is increased interest in monitoring, predicting, and even generating energy from multiple sources. Consider a system that integrates gas, coal, nuclear power, solar, hydro, and wind power, that has chips on all electric devices that communicate with central power company servers, local servers, and weather stations. Power-intensive tasks, from washing machines to automated factories, could be set to run when the solar cells are likely to be charged, the windmills are likely to be active, or the demand for heating or air conditioning is expected to be low. South Korea is testing such a system (McDonald, 2011), as are other countries. Power agents are being developed in situ, and in the context of a Trading Agents Competition (Block et al. 2010).

A power agent needs to be able to reason about likely weather conditions and power demands over the immediate and near future. It needs to condition such reasoning on location, time of year, and recent power demands, and to know about the tasks it is assigned to schedule.

[1] `http://cartel.csail.mit.edu/doku.php`

[2] `http://www.inrix.com/`

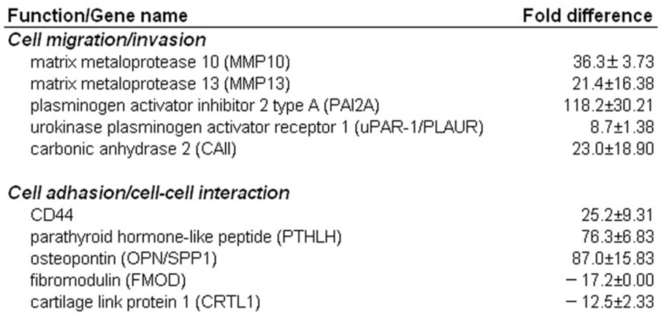
1.2 Scientific Databases

Scientific observations are fundamentally uncertain. No measurement is exact. When a quantity is observed and measured, the outcome depends on the measuring system, the experimental procedure, the skill of the person conducting the experiment, the environment, and other effects. Even if the quantity were to be measured several times, in the same way and in the same circumstances, a different measured value would in general be obtained each time, assuming that the measuring system has sufficient resolution to distinguish between the values.

Furthermore, as observed by domain scientists (e.g., (Burton et al. 2009)), due to unknown complex factors, contemporary scientific problems (e.g., associations of genetic variants and chronic diseases) often demand vast sample sizes and it is much needed to synthesize data across many studies and to undertake a pooled analysis. Below, we will look at a few concrete examples.

Astronomy

In astronomy, observations of the objects and phenomena in the sky are typically associated with "error bars" that indicate the estimated Gaussian distributions for the values being observed. Let us look at one of the most popular astronomical dataset, the Sloan Digital Sky Survey (SDSS). SDSS is one of the most ambitious and influential surveys in the history of astronomy[3]. It covers more than a quarter of the sky and contains more than 930,000 galaxies and more than 120,000 quasars.



Function/Gene name	Fold difference
Cell migration/invasion	
matrix metaloprotease 10 (MMP10)	36.3± 3.73
matrix metaloprotease 13 (MMP13)	21.4±16.38
plasminogen activator inhibitor 2 type A (PAI2A)	118.2±30.21
urokinase plasminogen activator receptor 1 (uPAR-1/PLAUR)	8.7±1.38
carbonic anhydrase 2 (CAII)	23.0±18.90
Cell adhasion/cell-cell interaction	
CD44	25.2±9.31
parathyroid hormone-like peptide (PTHLH)	76.3±6.83
osteopontin (OPN/SPP1)	87.0±15.83
fibromodulin (FMOD)	− 17.2±0.00
cartilage link protein 1 (CRTL1)	− 12.5±2.33

Fig. 2 Fold differences of two function groups of genes (among many) as measured by a microarray experiment (Komatsu, et al. 2006)

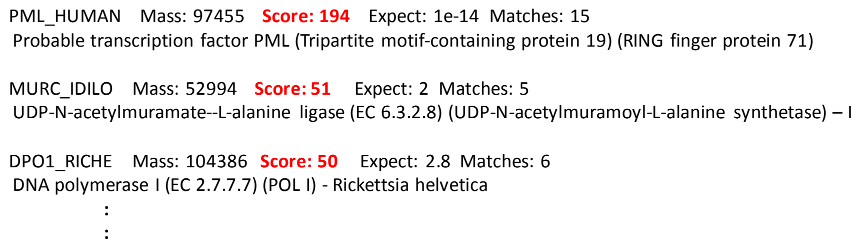
In the SDSS dataset, objects can have positional attributes: *right ascension* (ra) and *declination* (dec) in the J2000 coordinate system. Besides these two attributes, there are another two attributes, *ra_error* and *dec_error*, which are error bars. They indicate that the right ascension (declination, respectively) attribute is a random variable that has a Gaussian distribution with a standard deviation *ra_error* (*dec_error*, respectively) and a mean *ra* (*dec*, respectively).

[3] `http://www.sdss.org/`

Microarrays

DNA microarray analysis has been one of the most widely used sources of genome-scale data in the life sciences. Microarray expression studies are producing massive quantities of gene expression and other functional genomics data, which promise to provide key insights into gene function and interactions within and across metabolic pathways.

Figure 2 shows a snippet of the result from a microarray experiment performed by a research group (Komatsu, et al. 2006). It shows the fold differences of genes under two function groups. Here, a fold difference value indicates the difference between the gene’s expressed strength in a tissue sample (e.g., cancer cells) and that in a normal tissue being compared with. A positive (negative, resp.) value indicates that the gene is expressed more strongly (more weakly, resp.) in the tissue sample. Thus, scientists are interested in finding genes with large absolute fold differences, which are characteristic of the disease/tissue being studied. The ± range value (e.g., 3.73 in the first gene) is the standard deviation over a few repeated runs, each of which is called a replicate. We can see that the variance can be quite significant. Figure 2 only shows selected genes from two function groups among many.



```
PML_HUMAN   Mass: 97455   Score: 194   Expect: 1e-14  Matches: 15
 Probable transcription factor PML (Tripartite motif-containing protein 19) (RING finger protein 71)

MURC_IDILO   Mass: 52994   Score: 51   Expect: 2  Matches: 5
 UDP-N-acetylmuramate--L-alanine ligase (EC 6.3.2.8) (UDP-N-acetylmuramoyl-L-alanine synthetase) – I

DPO1_RICHE   Mass: 104386   Score: 50   Expect: 2.8  Matches: 6
 DNA polymerase I (EC 2.7.7.7) (POL I) - Rickettsia helvetica
              :
              :
```

Fig. 3 Sample output from the Mascot software that displays the proteins found in a sample. The scores indicate the confidence of the detections.

Proteomics

Proteomics is the large-scale study of proteins, particularly their structures and functions. Mass spectrometry has become a powerful tool in protein analysis and the key technology in proteomics (Mann et al. 2001). Proteomics experimental results contain information such as what proteins are in a tissue (either with a certain disease or normal), and their abundance, etc. Due to the many technical constraints in mass spectrometry (Mann et al. 2001), the experimental results have significant uncertainty.

Figure 3 shows a piece of the sample output from the widely used Mascot software[4] using Peptide Mass Fingerprint. Each possible protein is associated with a score, indicating the confidence of the detection. This can become more complicated when a tissue sample contains multiple proteins. A scientist would be

[4] `http://www.matrixscience.com/`

interested in knowing the abundance of a protein in a tissue, etc. Such information is often compared between a tissue sample (e.g., cancer cells) and a control (i.e., normal cells).

We have seen motivating applications in domains that require data management systems to handle uncertain and probabilistic data. In the rest of this chapter, we focus on two approaches of probabilistic databases. In the first approach (Section 2), we propose a rich treatment of probability distributions as data, in particular the SPO framework and the SP-algebra. In the second approach (Sections 3 and 4), we stay closer to a traditional DBMS, extend it with tuple probabilities or attribute probability distributions, and then study the semantics and efficient processing of queries in this model.

2 Semistructured Probabilistic Database Management Systems

One of the most common data structures for probabilistic reasoning is the Bayesian network, or Bayes net (Pearl 1988). A Bayes net is a directed acyclic graph, where nodes represent random variables and edges represent dependencies; each node has a probability table for the associated variable, conditioned on the values of its parents in the graph.

Example. Consider the Soldier Physiologic Status Monitoring application discussed in Section 1.1. Let us assume that a soldier has three sensors that, at set time intervals, send information about his/her body temperature, oxygen levels and pulse back to the home base server. Based on information supplied by the three sensors, which we refer to as T (temperature), O (oxygen) and P (pulse), the monitors (human or automatic) at the home base make decisions concerning the current state of the soldier. In this simplified scenario, suppose there are four states that a soldier can be in: N(ormal), W(eak), A(gitated) or S(ick). The reports of each of the three sensors are discretized into two values: H(igh) and N(ormal), with specific values of body temperature (e.g., 101F), blood oxygen levels (e.g., 90%) and pulse (e.g, 84) serving as the boundary values between them. The current state of the soldier is determined based on the information obtained from these three sensors. This can be represented graphically in the form of a Bayes net shown in Figure 4.

To complete the Bayes net, we supply the conditional probability distribution for the random variable C(ondition) based on the value of random variables T(emperature), O(xygen) and P(ulse), and provide marginal probability distributions for the other three random variables. Table **1** shows the former, while Table 2 shows a joint probability distribution of T, O and P.

A software application for tracking the medical conditions of military personnel might have to operate with different conditional probability tables and different marginal probability distributions based on a variety of factors. For example, the distributions shown in Tables 1 and 2 may be based on performance of Army personnel in temperate, forest-covered hilly environment. A different set of probability distributions might cover mountainous or desert terrain and be constructed for

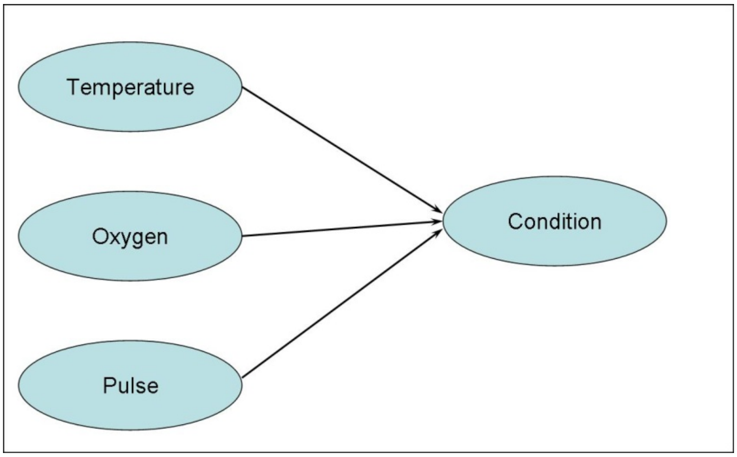


Fig. 4 A Bayes net for determining the physiological condition of a soldier in the field

other branches of the military (e.g., the Marine Corps) or specific military units. From joint probability distributions such as the one in Table 2, one can derive marginal probability distributions for subsets of parameters (e.g., Table 3 shows marginal probability distributions for (a) a pair of parameters T and P and (b) single parameter O) and can obtain conditional probability distributions (Table 3 (c) shows the distribution of T and O for personnel with high pulse rates).

Table 1 Conditional probability distribution for the Bayes net for determining the physiological condition of a soldier

T	O	P	Condition			
			Normal	Weak	Agitated	Sick
High	Normal	High	0.05	0.1	0.3	0.55
High	Normal	Normal	0.2	0.1	0.15	0.65
High	Low	High	0.05	0.05	0.1	0.8
High	Low	Normal	0.2	0.1	0.05	0.8
Normal	Normal	High	0.3	0.05	0.55	0.1
Normal	Normal	Normal	0.8	0.05	0.1	0.05
Normal	Low	High	0.2	0.2	0.45	0.15
Normal	Low	Normal	0.3	0.4	0.05	0.25

For decision-support software designed to work with this data, a data management mechanism is needed to deal with all such probability distributions. This is the underlying idea behind the Semistructured Probabilistic Objects (SPO) framework (Zhao et al. 2005). In this framework, diverse probability distributions, such as the ones depicted in Tables 1, 2, and 3 are stored as first-class database objects. A rich query algebra (the SP-algebra) is able to manipulate and retrieve the objects. The algebra incorporates traditional relational algebra operations of selection, projection, and

Cartesian product and join. It modifies their semantics to perform appropriate computations on the probability distributions, and adds a conditionalization operation that is unique to working with probability distributions. An SQL-style query language (SPOQL) has been implemented as a convenient syntax for querying databases of SPOs (Dekhtyar et al. 2006), though other implementations are certainly possible.

In what follows, we define the SPO framework formally, introduce the SP-algebra and discuss the semantics of its operations, and establish some key facts about the SP-algebra.

The SPO framework was originally introduced (Dekhtyar et al. 2001, Zhao et al. 2005) for exact (or "point") probabilities, i.e., for situations where the exact probabilities are known. However, it was observed that in many decision support applications, exact probabilities were not known. Rather, the probabilities of various situations/events were known to fall into probability intervals. The SPO framework was adapted to address such situations as well (Zhao et al. 2004). The notion of an Interval Semistructured Probabilistic Object (ISPO) is not too different than the notion of a SPO, but the Interval SP-algebra is significantly more complex (Zhao et. al 2003). We discuss this notion briefly at the end of Section 2.

Table 2 Joint marginal probability distribution of temperature, blood oxygen levels and pulse rate for the Bayes net for determining the physiological condition of a soldier.

T	O	P	*Prob*
High	Normal	High	0.02
High	Normal	Normal	0.01
High	Low	High	0.03
High	Low	Normal	0.04
Normal	Normal	High	0.25
Normal	Normal	Normal	0.4
Normal	Low	High	0.2
Normal	Low	Normal	0.05

Table 3 Probability distributions computable in the physiological condition monitoring scenario

(a)

T	P	*Prob*
High	High	0.05
High	Normal	0.05
Normal	High	0.45
Normal	Normal	0.45

(b)

O	*Prob*
Normal	0.68
Low	0.32

(c)

T	O	*Prob*
High	Normal	0.44
High	Low	0.06
Normal	Normal	0.5
Normal	Low	0.4
Prob(T,O|P=high)		

The term "semistructured" in the name of the framework was chosen for two reasons. The probability distributions stored in a single "relation" inside a Semistructured Probabilistic Database can have diverse structures and contain different "attributes". In addition to that, originally, XML was chosen as the representation syntax for SP objects (Dekhtyar et al. 2001). As XML representation is essentially syntactical in nature, we omit it from this narrative, and instead, concentrate on the semantics of the proposed frameworks.

2.1 Semistructured Probabilistic Objects

Consider a universe $\boldsymbol{V}$ of discrete random variables $\{v'_1,\dots,v'_q\}$. With each random variable $v \in V$ we associate $dom(v)$, a finite set of its possible values. Given a set $V=\{v_1,\dots,v_q\} \subseteq \boldsymbol{V}$, $dom(V)$ denotes $dom(v_1) x \dots x\ dom(v_q)$.

Let $R=(A_1,\dots,A_n)$ be a collection of regular relational attributes. For $A \in R$, let $dom(A)$ denote the domain of A. We define a semistructured schema $\underline{R}^*$ over R as a multiset of attributes from R. For example, if R = *{Terrain, MilitaryBranch, Conditions}* the following are valid semistructured schemas over R: R^*_1 = *{Terrain, MilitaryBranch}*; R^*_2 = *{Terrain, Conditions, Conditions}*; R^*_3 = *{Terrain, Terrain, Terrain}*.

Let $\boldsymbol{P}$ denote a probability space used in the framework to represent probabilities of different events. We present the framework over two different probability spaces. The first probability space $\boldsymbol{P}_{point}=[0,1]$, is the unit interval. Values from this interval are called *exact* or *point probabilities.* The Semistructured Probabilistic Object (SPO) framework introduced below uses this probability space. Another possibility, leading to an extended SPO framework (Zhao et al. 2004), is based on the probability space $\boldsymbol{P}_{int}=C[0,1]$: the set of all subintervals of the unit interval. A probability value from this space is called an *interval probability.* The general definition of a Semistructured Probabilistic Object given below applies for any probability space, however, the query algebra for each of the two frameworks is substantially different. We describe the query algebra, the SP-algebra, over the point probability space $\boldsymbol{P}_{poin}$ in Section 2.2, and briefly discuss the query algebra (Extended SP-algebra) for the interval probability space $\boldsymbol{P}_{int}$ at the end of Section 2.

Definition 1. (Zhao et al. 2005) A *Semistructured Probabilistic Object (SPO)*} S is a tuple $S = (T, V, P, C, \omega)$ where

- T is a relational tuple over some semistructured schema R^* over R. We refer to T as the *context* of S.
- $V=\{v_1,\dots,v_q\} \subseteq \boldsymbol{V}$ is a set of random variables. We require that $V \neq \varnothing$, where V is called the set of *participating random variables.*
- $P: dom(V) \rightarrow \boldsymbol{P}$ is the *probability table* of S. Note that P need not be complete, but it must be *consistent* with respect to $\boldsymbol{P}$[5].

[5] Consistency criteria are probability-space dependent. For $\boldsymbol{P}_{point}$, the consistency criterion is that the sum of all probability values is less than or equal to 1. For $\boldsymbol{P}_{int}$ the consistency criterion is essentially equivalent to a requirement that the sum of lower bounds of each probability interval is less than or equal to 1.

- $C = \{(u_1,X_1),\ldots,(u_s,X_s)\}$, where $U = \{u_1,\ldots,u_s\} \subset \boldsymbol{V}$ and $X_i \subseteq dom(u_i)$, $1 \leq i \leq n$, such that $V \cap U = \emptyset$. We refer to C as the *conditional* of S.
- ω, called a *path,* is an expression of the probabilistic query algebra over $\boldsymbol{P}$. We define two different query algebras below.

An explanation of this definition is in order. For the SPO data model to possess the ability to store all the probability distributions described in Tables **1**, **2,** and **3 a--c**, the following information needs to be stored in a single object.

1. **Participating random variables.** These variables determine the probability distribution described in an SPO.
2. **Probability Table.** If only one random variable participates, it is a simple probability distribution table; otherwise the distribution will be joint. Probability table may be complete, when the information about the probability of every instance is supplied, or incomplete. In either case, it must be consistent, i.e., truly represent a probability distribution. It is convenient to visualize the probability table P as a table of rows of the form $(\boldsymbol{x},\alpha)$, where $\boldsymbol{x} \in dom(V)$ and $\alpha = Prob(\boldsymbol{x}) \in \boldsymbol{P}$. Thus, we speak about rows and columns of the probability table when that makes explanations more convenient.
3. **Conditional.** A probability table may represent a distribution, conditioned by some prior information. The conditional part of its SPO stores the prior information in one of two forms: "`random variable` *u* `has value` *x*" or "`the value of random variable` *u* `is restricted to a subset` *X* `of its values`". In our definition, this is represented as a pair (u,X). When X is a singleton set, we get the first type of the condition.
4. **Context** provides supporting information for a probability distribution, information about the known values of certain parameters, which *are not considered to be random variables* by the application.
5. **Path.** Participating variables, probability table, conditional and context combined form the content of an SPO. Path, the fifth component, documents the object's history in the database in which it is stored. Objects inserted into the database receive unique object identifiers (OIDs) upon insertion. When a new SPO is constructed out of one or more existing SPOs as a result of a query algebra expression, the path of the new object will contain that expression.

2.2 The SP-algebra for Point Probabilities

Let us fix the universe of random variables $\boldsymbol{V}$, the universe of context attributes $\boldsymbol{R}$ and set the probability space $\boldsymbol{P}=\boldsymbol{P}_{point}= [0,1]$. A finite collection $SP = \{ S_1,\ldots ,S_n\}$ of semistructured probabilistic objects over $\boldsymbol{V}$, $\boldsymbol{R}$ and $\boldsymbol{P}$ is called a *semistructured probabilistic relation (SP-relation).* A finite collection $D = \{SP_1,\ldots,SP_r\}$ is called a *semistructured probabilistic database (SP-database).*

One important difference between semistructured probabilistic databases and traditional relational or relational probabilistic databases is that each table in a relational database has a specified schema, whereas all SP-relations are "schema-less": any collection of SPOs can form an SP-relation. This means that the division of a semistructured probabilistic database into relations is a matter of the logic of a particular application. For example, if the SP-database is built from the information supplied by three different experts, this information can be arranged into three semistructured probabilistic relations according to the origin of each object inserted in the database. Alternatively, the information can be arranged in SP-relations by the date it was obtained.

Manipulation of SPOs stored in SP-databases is done by the means of a query algebra, called the semistructured probabilistic algebra (SP-algebra). The SP-algebra contains three standard set operations: *union*, *intersection* and *difference;* it extends the definitions of standard relational operations *selection*, *projection*, *Cartesian product*, and *join* to account for the appropriate management and maintenance of probabilistic information within SPOs; in addition, it contains a new operation, *conditionalization*. The latter operation is specific to the probabilistic databases and results in the construction of SPOs that represent conditional probability distributions of the input SPOs.

Before proceeding with the description of individual operations, we define the *equality* and *equivalence* of SPOs. Two SPOs *S* and *S'* are *equal* if all their components are equal. Two SPOs are *equivalent* if their set of participating random variables, probability table, context and conditional are the same. Notice that in the case of equivalence, *paths* of two SPOs may be different. More formally,

Definition 2. (Zhao et al. 2005) Let $S=(T,V,P,C,\omega)$ and $S'=(T',V',P',C',\omega')$ be two SPOs. *S is equivalent to S'*, denoted $S\equiv S$, iff $T=T$, $V=V'$, $P=P'$ and $C=C'$.

Set Operations. Semistructured Probabilistic relations are sets of SPOs. Therefore, the definitions of union, intersection and difference of SP-relations are straightforward.

Definition 3. (Zhao et al. 2005) Let *SP* and *SP'* be two SP-relations.

- **Union:** $SP \cup SP' = \{ S \mid S \in SP$ or $S \in SP'\}$.
- **Intersection:** $SP \cap SP' = \{ S \mid S \in SP$ and $S \in SP'\}$.
- **Difference:** $SP - SP' = \{ S \mid S \in SP \text{ and } S \notin SP'\}$.

We note two features of the set operations in the SP-algebra. Classical relational algebra has a restriction on the applicability of the set operations: they are defined only on pairs of relations with matching schemas. Because SP-relations are schema-less and represent logical rather than syntactic groupings of probability distributions in an SP-database, set operations are applicable to any pair of SP-relations.

Selection. Given an SPO $S = (T, V, P, C, \omega)$, a selection query may be issued to any of its components except the path. Each part requires its own language of selection conditions. Selection on *context*, *participating random variables* and *conditionals*, when applied to an SPO, result in the SPO being selected or not in its entirety, as is the case with selection in relational algebra. Selection on *probability table* on the other hand, transforms the SPO by including in the selected object only the probability table rows that match the selection condition. For any selection operation, the path expression of the result is updated to include the selection operation. We illustrate different types of selections in the following example.

Example 1. Consider the military personnel monitoring application described in the example above. Suppose that the application database stores multiple probability distributions to be used for decision support. A human analyst working with the system may, at different times, want to see and/or use the results of the following information requests.

- `"Find all probability distributions for members of the Marine Corps."}` This is an example of selection based on context.
- `"Find all probability distributions that involve body temperature and oxygen level observations."` Body temperature and oxygen level are two of the random variables in the application domain. This is an example of selection on participating random variable.
- `"Find all probability distributions for servicemen with low oxygen levels".` Here, the analyst wants to find what is known about the probabilities of *other* random variables in the domain, when the oxygen level (a random variable in the domain) is known to be low. This is selection on conditional.
- `"What information is available about the probability of having low oxygen level and high body temperature?"` In each SPO which contains Temperature and Oxygen variables, we are interested in the row(s)[6] of the probability table which has/have values Temperature = high and Oxygen = low. This is an example of selection on probability table.
- `"What outcomes have probability over 0.4?"` This is an example of selection on probabilities. This information need should result in only the SPOs that have probability table rows with probability values of above 0.4 returned, and *only those rows* should be shown to the analyst.

Selection on Context, Participating Variables or Conditionals. We first define the three selection operations that do not alter the content of the selected objects. We start by defining the acceptable languages for selection conditions for the three

[6] If other random variables are also present in the SPO in question, there will be more than one row matching this condition.

types of selects. Recall that the universe $\boldsymbol{R}$ of context attributes consists of a finite set of attributes $A_1,\ldots, A_n$ with domains $dom(A_1),\ldots ,dom(A_n)$. With each attribute $A \in \boldsymbol{R}$ we associate a set *Predicates(A)* of allowed predicates. We assume that *equality* and *inequality* are allowed for all $A \in \boldsymbol{R}$.

Definition 4. (Zhao et al. 2005) An *atomic context selection condition* is an expression c of the form $A \bullet x$ (or $\bullet(A,x)$), where $A \in \boldsymbol{R}$, $x \in dom(A)$ and $\bullet \in Predicates(A)$.

An *atomic participation selection condition* is an expression c of the form $v \in V$, where $v \in \boldsymbol{V}$ is a random variable.[7]

An *atomic conditional selection condition* is one of the following expressions: $u =\{x_1,\ldots, x_h\}$ or $u \ni x$ where $u \in \boldsymbol{V}$ is a random variable and $x, x_1,\ldots,x_h \in dom(u)$.

Complex selection conditions can be formed as Boolean combinations of atomic selection conditions.

Definition 5. (Zhao et al. 2005) Let $S=(T,V,P,C,\omega)$ be an SPO and let $c = A \bullet x$ be an atomic context selection condition. Let $\omega' = \sigma_c(\omega)$ and let $S' = (T,V,P,C,\omega')$. Then $\sigma_c(S) =\{S'\}$ iff:

1. $A \in S.T$;
2. For some instance A^* of A in $S.T$, $S.T.A^* \bullet x$ is true.

Otherwise, $\sigma_c(S) = \emptyset$.

Definition 6. (Zhao et al. 2005) Let $S=(T,V,P,C,\omega)$ be an SPO and let $c = v \in V$ be an atomic participation selection condition. Let $\omega' = \sigma_c(\omega)$ and let $S' = (T,V,P,C,\omega')$. Then $\sigma_c(S) =\{S'\}$ if $v \in S.V$; otherwise $\sigma_c(S) = \emptyset$.

Definition 7. (Zhao et al. 2005) Let $S=(T,V,P,C,\omega)$ be an SPO. Let $\omega' = \sigma_c(\omega)$ and let $S' = (T,V,P,C,\omega')$.

1. Let $c = u =\{x_1,\ldots ,x_h\}$ be an atomic conditional selection condition. Then $\sigma_c(S) =\{S'\}$ if $S.C \ni (u, X)$ and $X = \{x_1,\ldots,x_h\}$; otherwise $\sigma_c(S) = \emptyset$.
2. Let $c = u \ni x$ be an atomic conditional selection condition. Then $\sigma_c(S) =\{S'\}$ if $S.C \ni (u, X)$ and $x \in X$; otherwise $\sigma_c(S) = \emptyset$.

The semantics of atomic selection conditions can be extended to their Boolean combinations in a straightforward manner.

$$\sigma_{c \wedge c'}(S) ::= \sigma_c(\sigma_{c'}(S));$$
$$\sigma_{c \vee c'}(S) ::= \sigma_c(S) \cup \sigma_{c'}(S),$$

except for the path component, which will become, respectively, $\sigma_{c \wedge c'}(S)$ ($\sigma_{c \vee c'}(S)$).

[7] Note that "$\in V$" is syntactic sugar here. Instances of such conditions have the form *Oxygen* $\in V$, *Condition* $\in V$ and so on.

The interpretation of *negation* in the context selection condition requires some additional explanation. In order for a selection condition of the form $\neg(A \bullet x)$ to succeed on some SPO $S=(T,V,P,C,\omega)$, attribute A must be present in $S.T$. If A is not in $S.T$, then $\sigma_{\neg(A \bullet x)}(S) = \emptyset$. Therefore, the statement $S \in \sigma_c(S) \vee S \in \sigma_{\neg c}(S)$ is not necessarily true. This also applies to conditional selection conditions.

Selection on Probability Table. The two remaining types of selection operations are more complex than the three described above. Here, the result of each operation applied to an SPO can be *a non-empty part* of the original SPO. In particular, these operations preserve the context, participating random variables and conditionals in an SPO, but may return *only a subset* of the rows of the probability table. In these operations, the selection condition will indicate which rows from the probability table are to be included and which are to be omitted. In a sense, these operations treat the probability table of an SPO as a relational table, and perform selections from it.

Definition 8. (Zhao et al. 2005) An *atomic probability table selection condition* is an expression of the form $v = x$ where $v \in \boldsymbol{V}$ and $x \in dom(v)$. *Probability table selection conditions* are Boolean combinations of atomic probability table selection conditions.

Definition 9. (Zhao et al. 2005) Let $S = (T,V,P,C,\omega)$ be an SPO, $V = \{ v_1,\ldots,v_k\}$ and let $c = v = x$ be an atomic probabilistic table selection condition. Let $\omega' = \sigma_c(\omega)$. If $v \in V$, then (assuming $v = v_i$ for some $1 \le i \le k$) the result of selection from S on c, $\sigma_c(S)$, is a semistructured probabilistic object $S' = (T,V,P',C,\omega')$, where

$$P'(y_1,\ldots,y_i,\ldots,y_k) = \begin{cases} P(y_1,\ldots,y_i,\ldots,y_k) & y_i = x; \\ \textit{undefined} & y_i \neq x. \end{cases}$$

Definition 10. An *atomic probabilistic selection condition* is an expression of the form $P \bullet \alpha$, where $\alpha \in [0,1]$ and $\bullet \in \{=, \neq, \le, \ge, <, > \}$. *Probabilistic selection conditions* are Boolean combinations of atomic probabilistic selection conditions.

Definition 11. Let $S=(T,V,P,C,\omega)$ be an SPO and let c= $P \bullet \alpha$ be an atomic probabilistic selection condition. Let $\boldsymbol{x} \in dom(V)$. The result of selection from S on c is defined as follows: $\sigma_c(S) = (T,V,P',C,\omega')$, where $\omega' = \sigma_c(\omega)$ and

$$P'(\bar{x}) = \begin{cases} P(\bar{x}) & P(\bar{x}) \bullet \alpha; \\ \textit{undefined} & \neg(P(\bar{x}) \bullet \alpha). \end{cases}$$

Different selection operations commute, as shown in the following theorem.

Theorem 1. (Zhao et al. 2005) Let c and c' be two (arbitrary) selection conditions and let SP be a semistructured probabilistic relation. Then $\sigma_c(\sigma_{c'}(SP)) \equiv \sigma_{c'}(\sigma_c(SP))$.

Projection. SPOs are complex objects consisting of four different components. Traditionally, *projection* in relational algebra is a simplification operation that removes attributes. With SPOs, there are three types of simplifications that can be performed: removal of context, removal of conditionals and removal of participating random variables. All three projection operations are introduced below.

Definition 12. (Zhao et al. 2005) Let $S = (T, V, P, C, \omega)$ be an SPO and let $L \subseteq \boldsymbol{R}$ be a set of context attributes. The projection of S onto L, denoted $\pi_L(S,)$, is an SPO $S' = (T', V, P, C, \omega')$, where $T' = \{(A,x) | (A,x) \in T, A \in L\}$ (i.e., T' contains all entries from T for attributes from the list L only), and $\omega' = \pi_L(\omega)$.

Definition 13. (Zhao et al. 2005) **Let** $S = (T, V, P, C, \omega)$ be an SPO and let $F \subseteq \boldsymbol{V}$ be a set of random variables. The projection of the conditional part of S onto F, denoted $\pi_{c:F}(S)$, is an SPO $S' = (T, V, P, C', \omega')$ where $C' = \{(u,X) | (u,X) \in T.C, u \in F\}$ and $\omega' = \pi_{c:F}(\omega)$.

We note that since both the context and the conditional part of an SPO can be empty, projections $\pi_{\varnothing}(S)$ (i.e., removal of all context information for an SPO) and $\pi_{c:\varnothing}(S)$ (clearing of the list of conditionals) are valid and will yield proper results. A somewhat more complicated and delicate operation is the projection on the set of participating random variables. A removal of a random variable from the SPO's participant set entails that all information related to this random variable has to be removed from the probability table as well. This essentially corresponds to removal of a random variable from consideration in a joint probability distribution, which is usually called *marginalization*. The result of this operation is a new marginal probability distribution that needs to be stored in the probability table component of the resulting SPO.

This computation is performed in two steps. First, the columns for random variables that are to be projected out are removed from the probability table. In the remainder of the table, there can now exist duplicate rows whose values for all the fields except the probability coincide. All duplicate rows of the same type are then collapsed (coalesced) into one, with the new probability value computed as the sum of the values in the constituent rows. The formal definition of this procedure is given below.

Definition 14. (Zhao et al. 2005) Let $S = (T, V, P, C, \omega)$ be an SPO, $V = \{v_1, ..., v_q\}$, $\boldsymbol{q>=1}$, and let $L \subseteq \boldsymbol{R}$ be a *non-empty* set of random variables. If $L \cap S.V = \varnothing$, then the projection of S on L, denoted $\pi_L(S)$, is an empty set. If If $L \cap S.V \neq \varnothing$, then $\pi_L(S) = \{S'\}$ where $S' = (T, L, P', C, \omega')$ and where $P' : dom(L) \rightarrow [0,1]$ and for each $\bar{x} \in dom(L)$,

$$P'(\bar{x}) = \sum_{\bar{y} \in dom(V-L); P(\bar{x},\bar{y}) \text{ is defined}} P(\bar{x},\bar{y}) \quad .$$

Notice that projection on the participating random variables is allowed only if the $S.V$ is not a singleton and if at least one random variable remains in the resulting set.

Conditionalization. Conditionalization is an operation specific to probabilistic algebras. Dey and Sarkar (Dey and Sarkar 1996) were the first to consider this operation in the context of probabilistic databases. Similarly to the variable projection operation, conditionalization reduces the probability distribution table. The difference is that the result of conditionalization is a *conditional probability distribution*. Given a joint probability distribution, conditionalization answers the general query of the form, "`What is the probability distribution of the remaining random variables if the value of some random variable` *v* `in the distribution is restricted to subset` *X* `of its values?`"

Informally, the conditionalization operation proceeds on a given SPO as follows.

The input to the operation is one participating random variable of the SPO, *v*, and a subset of its domain $X \subseteq dom(v)$. The first step of the operation consists of removal from the probability table of the SPO all rows whose *v* values are not from the set *X*. Then the *v* column is removed from the table. The remaining rows are coalesced (if needed) in the same manner as in the projection operation and afterwards, the probability values are normalized. Finally, *(v,X)* is added to the set of conditionals of the resulting SPO.

The formal definition of conditionalization is given below. Note that if the original table is incomplete, there is no meaningful way to normalize the probability distribution. The operation can still be performed, but the results may be meaningless. Thus, we restrict this operation to situations where normalization is well defined.

Definition 15. (Zhao et al. 2005) An SPO $S=(T, V,P,C,\omega)$ is *conditionalization-compatible* with an atomic conditional selection condition $v =\{x_1,\dots ,x_h\}$ iff (a) $v \in S.V$ and (b) the restriction of *S.P* on $\{ x_1,\dots ,x_h\}$ for variable *v* is a complete function.

Definition 16. (Zhao et al. 2005) Let SPO $S=(T, V,P,C,\omega)$ be an SPO which is conditionalization-compatible with an atomic conditional selection condition $c = v =\{x_1,\dots ,x_h\}$. The result of *conditionalization* of *S* by *c*, denoted $\mu_c(S)$, is defined as follows: $\mu_c(S) = (T,V',P',C',\omega)$, where

- $V' = V -\{v\}$;
- $C' = C \cup\{(v,\{x_1,\dots ,x_h)\}$;
- $P':V' \rightarrow[0,1]$ is defined as follows.

Let

$$N = \sum_{\bar{y}\in dom(V')} \sum_{x\in\{x_i,\dots,x_h\}} P(x,\bar{y}).$$

Then, for any $\bar{y} \in dom(V')$, *P'* is defined as follows:

$$P'(\bar{y}) = \frac{\sum_{x\in\{x_1,\dots,x_h\}} P(x,\bar{y})}{N};$$

- $\omega' = \mu_c(\omega)$.

Cartesian Product and Join. Sometimes an SP-database has only simple probability distributions for some random variables. In order to get a joint probability distribution, either a Cartesian product or a join operation can to be performed on the SPOs storing these distributions. Intuitively, both a Cartesian product and a join of two probabilistic distributions compute the joint probability distribution of random variables involved in both original distributions. The difference between them lies in the operation applicability. The Cartesian product can be computed only for a pair of SPOs with disjoint participating random variables. The join operation is applicable to two SPOs that share common participating random variables.

When a joint probability distribution is computed from individual (marginal) probability distributions, knowledge of the relationship between the random variables in the two marginal distributions is necessary to correctly compute the joint probability distribution. In this narrative, we restrict ourselves to the case when random variables from the two distributions are conditionally independent. This restriction allows us to represent the result as a joint probability distribution which can be explicitly computed: the joint probability is the product the marginal probabilities· Other assumptions that allow for direct computation of joint probability distributions are discussed elsewhere (Zhao et al. 2006).

Two SPOs are *compatible* for Cartesian product if their participating variables are disjoint, but their conditionals coincide.

Definition 17. (Zhao et al. 2005) Two SPOs $S = (T,V,P,C,\omega)$ and $S' =(T',V',P',C',\omega')$ are *Cartesian product-compatible (cp-compatible)* if and only if (a) $V \cap V' = \varnothing$ and (b) $C = C'$.

We can now define the Cartesian product.

Definition 18. (Zhao et al. 2005) Let $S = (T,V,P,C,\omega)$ and $S' =(T',V',P',C',\omega')$ be two cp-compatible SPOs. The result of their Cartesian product (under assumption of independence), denoted $S \times S'$, is: $S \times S' = S'' = (T'',V'',P'',C'',\omega'')$, where

- $T'' = T \cup T'$;
- $V'' = V \cup V'$;
- $P'': dom(V'') \rightarrow [0,1]$ is defined as follows. For all $\bar{z} \in dom(V'')$, where $\bar{z} = (\bar{x}, \bar{y})$, $\bar{x} \in dom(V)$, $\bar{y} \in dom(V')$: $P''(\bar{z}) = P(\bar{x}) \cdot P(\bar{y})$;
- $C'' = C = C'$;
- $\omega'' = \omega \times \omega'$.

The join operation extends the Cartesian product operation to the situation, where two SPOs being combined share random variables. If we have two probability distributions $Prob(X,Y)$ and $Prob(Y,Z)$, then a joint probability distribution $Prob(X,Y,Z)$ can be represented as $Prob(X,Y,Z) = Prob(X,Y)*Prob(Z|Y) = Prob(X|Y) * Prob(Y,Z)$. The two representations of the joint probability distribution (one, conditioning Z on Y and another, conditioning X on Y) are equal if the probability distributions are drawn from one known underlying universal probability distribution on $\boldsymbol{V}$. However, the SPO framework can store, in the same database, information from multiple universal distributions (e.g., distinguished by the context settings of the SPOs). Thus, *Prob(X,Y)*Prob(Z|Y) = Prob(X|Y)* Prob(Y,Z)* is not necessarily always true. To make sure that SPOs can be joined efficiently, we consider two separate join operations, one using the *Prob(X,Y)*Prob(Z|Y)* representation, and the other using the *Prob(X|Y)* Prob(Y,Z)*. These operations are known as *left join* and *right join.*

Definition 19. (Zhao et al. 2005) Two SPOs $S = (T,V,P,C,\omega)$ and $S' =(T',V',P',C',\omega')$ are *join-compatible* if and only if (a) $V \cup V' \neq \varnothing$ and (b) $C = C'$.

Given two join-compatible SPOs S and S', we can break the set $V \cup V'$ into three non-empty disjoint parts: $V_1 = V - V'$, $V_2 = V' - V$ and $V_c = V \cap V'$. The information about the probability distribution of random variables in V_c can be found in both S and S'. The join operation must take this into consideration when the joint probability distribution for variables in $V \cup V'$ is computed. The key to computing the joint distribution correctly is the following statement.

Lemma 1. Let $\mathbf{x} \in dom(V_1)$, $\mathbf{y} \in dom(V_c)$, $\mathbf{z} \in dom(V_2)$, and let V_1, Vc and V_2 all be disjoint. Under the assumption of independence between variables in V_1 and V_2 the following holds:

$$\Pr(\bar{x},\bar{y},\bar{z}) = \Pr(\bar{x},\bar{y}) * \Pr(\bar{y},\bar{z}) / \Pr(\bar{y}) = \Pr(\bar{x},\bar{y}) * P(\bar{z} \mid \bar{y}) = P(\bar{z},\bar{y}) * P(\bar{x} \mid \bar{y}).$$

We can now define the join operations. We want the join of S and S' to contain the joint probability distribution of the set $V_1 \cup V_c \cup V_2$. Since $\Pr(\mathbf{y})$ could be obtained either from S or from S', there exist two families of join operations, called *left join* and *right join*, with the following definitions.

Definition 20. (Zhao et al. 2005) Let $S = (T,V,P,C,\omega)$ and $S' =(T',V',P',C',\omega')$ be two join-compatible SPOs. Let $V = V_1 \cup V_c$ and $V' = V' \cup Vc$, and $Vc = V \cup V'$. We define the operations of *left join* of S and S', denoted $S \rhd\!< S'$, and right join of S and S', denoted $S >\!\lhd S'$, as follows:

- $S \rhd\!< S' ::= S'' = (T'',V'',P'',C'',\omega'')$;
- $S >\!\lhd S' ::= S''' = (T'', V'', P''',C'',\omega''')$, where
  1. $T'' = T \cup T'$;
  2. $V'' = V_1 \cup V_c \cup V_2$;
  3. $P''': dom(V'') \to [0,1]$ is computed as follows.

For all $\overline{w} \in dom(V'')$; $\overline{w} = (\bar{x},\bar{y},\bar{z})$; $\bar{x} \in dom(V_1)$, $\bar{y} \in dom(V_c)$, $\bar{z} \in dom(V_2)$:

$$P''(\overline{w}) = P(\bar{x},\bar{y}) \cdot \frac{P'(\bar{y},\bar{z})}{P'(\bar{y})}$$

$$P''(\overline{w}) = P'(\bar{y},\bar{z}) \cdot \frac{P(\bar{x},\bar{y})}{P(\bar{y})}$$

- $C'' = C = C'$.
- $\omega'' = \omega \rhd\!< \omega'$; $\omega''' = \omega >\!\lhd \omega'$.

Two *join-compatible* SPOs are *join-consistent* if probability distributions on the set of shared participating variables are identical for both SPOs.

Definition 21. (Zhao et al. 2005) Let Let $S = (T,V,P,C,\omega)$ and $S' = (T',V',P',C',\omega')$ be two join-compatible SPOs with $V \cap V' = V_c$. Then, S and S' are *join-consistent* if and only if $P(\bar{y}) = P'(\bar{x})$ for any $\bar{y} \in dom(V_c)$.

SP-algebra operations can be extended to a semistructured probabilistic relation, as described in the following proposition.

Proposition 1. (Zhao et al. 2005) Any SP-algebra operation on a semistructured probabilistic relation is equivalent to the union of the SP-algebra operation on each SPO in the SP-relation:

- Let SP be a semistructured probabilistic relation and γ be one of the three unary SP-algebra operators. Then $\gamma(SP) = \bigcup_{S \in SP} \gamma(S)$.
- Let SP_1 and SP_2 be two semistructured probabilistic relations and $\oplus$ be one of the binary SP-algebra operators. Then:

$$SP_1 \oplus SP_2 = \bigcup_{S \in SP_1} \bigcup_{S' \in SP_2} S \oplus S'.$$

Semantics of the SP-algebra Operations. The problem of determining the meaning of the results of the operations of the SP-algebra is complicated by the fact that at any moment, SP-databases can contain SPOs of two types. In the SPOs of the first type, the probabilities of all rows are exact, while in the SPOs of the second type, the probabilities of some rows may represent the lower bounds on the probability of those instances. We proceed by defining the two types of SPOs formally, discussing their properties and the effects that different SP-algebra operations have on the SPOs in light of this.

Definition 22. (Zhao et al. 2005) An SPO $S = (T,V,P,C,\omega)$ is a *Type I* SPO iff $\sum_{\mathbf{x} \in dom(V)} P(\mathbf{x}) = 1$. Otherwise, S is a *Type II* SPO.

When S is a Type I SPO, its probability table is *complete*: the probabilities of all rows add up to exactly 1. The probability table may contain a row for every instance $\mathbf{x} \in dom(V)$, or it may omit some of the instances. However, because the probabilities of the rows present in the table add up to 1, we know that the probabilities of all omitted rows are 0, and these can be added to the probability table of S. Basically, when S is a Type I SPO, we are guaranteed that *for all* $\mathbf{x} \in dom(V)$ $P(\mathbf{x})$ *is the exact point probability of instance* $\mathbf{x}$.

The nature of Type II SPOs is somewhat more complex. If the sum of probabilities in all rows of the probability table is less than 1, then that the probability table is missing some information. This can either be *missing instances*: some $\mathbf{x} \in dom(V)$ has a non-zero probability but is not included in the probability table of S, or *underestimation*: all possible instances are present, but the probabilities add up to less than 1, which means that information about the probabilities of some (or all) instances is only a *lower bound* on the true probability of the instance in the distribution.

It is important to note here that SP-algebra operations allow for Type II SPOs to occur in the SP-database, even if all original SPOs in the database were Type I. The difference in the meaning of probability values for Type I and Type II SPOs causes us to apply extra caution when interpreting the results of SP-algebra operations. In particular, when considering a specific SP-algebra operation applied to an SPO or a pair of SPOs, it is important for us to know the type of the input objects and be able to determine the type of the result. The following proposition identifies the set of "safe" operations in SP-algebra: operations that, given Type I SPOs, are guaranteed to produce Type I results.

Proposition 2. (Zhao et al. 2005) Let S and S' be two Type I SPOs. Then, the following SPOs are also Type I:

1. $\sigma_c(S)$, where c is a selection condition on *context*, *participating random variables* or *conditional.*
2. $\pi_L(S)$, $\pi_{c:F}(S)$ and $\pi_F(S)$, where L is a list of context attribute names and $F \subseteq \mathbf{V}$.
3. $\mu_c(S)$, where c is a conditional selection condition.
4. $S \times S'$.
5. $S \rhd\!< S'$ and $S >\!\lhd S'$.

Two operations missing from the list in Proposition 4 are selection on probabilities and selection on probability table. These operations can take as input Type I SPOs and produce Type II SPOs, because both operations can return incomplete probability tables. The following statements specify the semantics of the SP-algebra operations producing Type I results.

Theorem 2. (Zhao et al. 2005) Let $S = (T,V,P,C,\omega)$ be a Type I SPO and let $\emptyset \neq L \subseteq \mathbf{V}$. Let $S' = (T, L, P', C, \omega') = \pi_L(S)$. Then $S'.P'$ contains the ***correct marginal probability distribution*** of random variables in L given the probability distribution $S.P$.

Theorem 3. (Zhao et al. 2005) Let $S = (T,V,P,C,\omega)$ be a Type I SPO and let c be a conditional selection condition involving variable $v \in S.V$. Let $S' = (T,V-\{v\},P',C',\omega') = \mu_c(S)$. Then $S'.P'$ contains the ***correct conditional probability distribution*** of random variables $S.V-\{v\}$ from the distribution $S.P$ given condition c on v.

Theorem 4. (Zhao et al. 2005) Let $S = (T,V,P,C,\omega)$ and $S' =(T',V',P',C',\omega')$ be two cp-compatible SPOs and let $S'' = (T'',V'',P'',C,\omega'') = S \times S'$. Then $S''.P''$ is the correct joint probability distribution of random variables in $S.V$ and $S'.V'$ under the assumption of independence between them.

Theorem 5. (Zhao et al. 2005) Let $S = (T,V,P,C,\omega)$ and $S' =(T',V',P',C',\omega')$ be two join-compatible SPOs and let $S'' = (T'',V'',P'',C,\omega'') = S \rhd\!< S'$ and $S''' = (T''',V''',P''',C,\omega''') = S >\!\lhd S'$. Then $S''.P''$ and $S'''.P'''$ are the ***correct joint probability distributions*** of random variables in $S.V$ and $S'.V'$ under the assumption of independence between them.

Theorem 6. (Zhao et al. 2005) Let *S* and *S'* be two join-compatible SPOs. The left join S $\rhd\!<$ S' and the right join S $>\!\lhd$ S'$ are equivalent if and only if *S* and *S'* are *join-consistent.*

2.3. Extensions of the SPO Framework

Interval SPO Model. As mentioned above, the probability space $\boldsymbol{P}_{point}=[0,1]$, is not the only way to represent probabilistic information in the SPO framework. Probability intervals have been, for some time, considered the *next* natural extension of the notion of probability (Walley 1991, de Campos et al. 1994, Weichselberger 2000, Ng and Subrahmanian 1992). Because the definition of an SPO factors out the probability space, a valid Semistructured Probability Object may use probability intervals rather than point probabilities in its probability table. Such SPOs were introduced in a somewhat tongue-in-cheek manner (Goldsmith et al. 2003) as the means of representing results of political surveys. While the data representation format does not change much, the same cannot be said about the semantics of the SPOs and, consequently, the query algebra. An interval probability distribution is modeled using Nilsson's (Nilsson 1986) possible worlds semantics, (Weichselberger 2000, de Campos et al. 1994): a true probability distribution assigns point probabilities to all rows in the probability table, but is unknown. Probability intervals represent a set of linear constraints on the point probabilities. An interval probability distribution will satisfy some (possibly none) point probability distributions, termed *p-interpretations* (Ng and Subrahmanian 1992, Zhao et al. 2004), each of which is considered equally likely to be the true one.

The query algebra operations were extended to preserve the mapping between interval probability distributions and the sets of satisfying p-interpretations (Zhao et al. 2004, Zhao et al. 2003). The semantics of extended (interval) SP-algebra operations that do not alter probabilities, set operations and various selection operations, does not change much from the SP-algebra case. On the other hand, projection, Cartesian product, join, and, especially, conditionalization, operations that modify probabilities, become much more involved. With the exception of conditionalization, extended SP-algebra versions of all operations preserve the possible worlds semantics: i.e., we prove that a *p-interpretation* satisfies the interval probability distribution obtained in the result of an extended SP-algebra operation if and only if it be constructed from some *p-interpretation* (or a pair of *p-interpretations*) by applying an SP-algebra analog of the operation to it/them (Zhao et al. 2004).

For the conditionalization operation, the interval distribution obtained as result of the extended operation is guaranteed to be *tight*, i.e., some *p-interpretations* satisfying the original interval probability distribution get transformed by a conditionalization operation in a way that matches all interval boundaries. However, the resulting interval probability distribution can have satisfying *p-interpretations* that cannot be obtained from any p-interpretation satisfying the original (pre-conditionalization) interval probability. This is an instance of a general result, due to Jaffray (1992), concerning computing conditional imprecise probabilities. It represents an essential structural shortcoming of the interval probability models in general, and the extended SPO framework in particular.

SPDBMS. The SPO framework was implemented by Zhao (Zhao et al. 2005) using transformation of SP-relations in collections of relational tables on top of a relational DBMS. The Semistructured Probabilistic DBMS (SPDBMS for short) supports basic data manipulation (insert, delete, update an SPO) and provides full support for the SP-algebra. Because collections of SPOs are inherently semistructured, the translation of SPOs into relational tables is rather cumbersome. More recently, SPDBMS was re-implemented using the native XML DBMS eXist (Rosson 2008). This avoided the data translation step. Query algebra operations were implemented using XQuery, and XQuery's user-defined functions. Most of the operations behaved efficiently. However, due to the specifics of eXist's internal architecture,[8] processing Cartesian products and joins was unreasonably slow.

SPOQL. The SP-algebra provides a functional query language for querying SP-databases. Direct SP-algebra syntax was implemented in both versions of SPDBMS and used as the query language. In addition to the SP-algebra, a declarative query language for SP-databases, called SPOQL, was introduced and implemented as part of the RDBMS-based SPDBMS (Dekhtyar et al. 2006).

3 Modeling Uncertain Data

In this section, we consider databases that treat *data* as uncertain, rather than storing and managing uncertainty in terms of probability distributions. We model data uncertainty in three ways: (1) tuple uncertainty, (2) attribute uncertainty, and (3) sub-attribute uncertainty.

In *tuple uncertainty*, a probability number (sometimes called confidence) is associated with each tuple. An example is shown in Figure 5(a), which is similar to Figure 1, except that we have *mutual exclusion* correlations among tuples. Recall that it is from an application in which various sensors are embedded in the uniforms of soldiers in a battle field. The sensors send out detections of the medical conditions of the soldier that wears the uniform. The second to last column is a score that indicates how much medical attention this soldier needs. The higher the score, the more urgent it is to send medical resources to this soldier. The last column (Conf.) is the probability that the tuple exists in the table.

We may specify mutual exclusion rules, which indicate that at most one of a set of tuples can exist in the table. In this way, we can encode a discrete PMF (probability mass function) by a set of mutually exclusive tuples. In more detail, for a PMF $\{(v_1, p_1), (v_2, p_2), \ldots, (v_k, p_k)\}$, v_1 to v_k are values in a set of mutually exclusive tuples and p_1 to p_k are their probabilities. The sum of the probabilities is no more than 1. If the sum is less than 1, then with remaining probability, none of the mutually exclusive tuples exist in the table. In the example in Figure 4(a), the three highlighted tuples (T2, T4, and T7) are mutually exclusive. They are detections of the same soldier (same Soldier ID) at around the same time, and hence at

[8] The version of eXist used by Rosson (Rosson 2008) loaded user-defined functions and reinterpreted them each time they were invoked, which affected the join and Cartesian product operations .

Tuple ID	Soldier ID	Time	Location (x, y)	Score for Medical Needs	Conf.
T1	1	10:50	(10, 20)	49	0.4
T2	2	10:49	(10, 19)	60	0.4
T3	3	10:51	(9, 25)	110	0.4
T4	2	10:50	(10, 19)	80	0.3
T5	4	10:49	(12, 7)	56	1.0
T6	3	10:50	(9, 25)	58	0.5
T7	2	10:50	(11, 19)	125	0.3

(a)

Tuple Uncertainty

Object ID	Speed
28	Normal (90, 20)
11	Normal (62, 15)
72	Normal (78, 10)

(b) **distribution**

Attribute Uncertainty

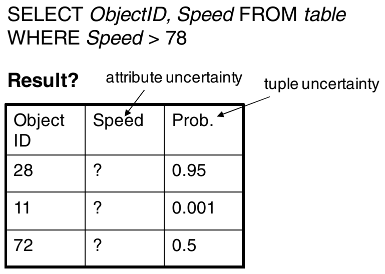
Fig. 5 Illustrating two kinds of uncertain data: tuple uncertainty (a) and attribute uncertainty (b). The last column of (a) (Conf., i.e., confidence) indicates the probability that the tuple exists in the table. The highlighted tuples are mutually exclusive (i.e., at most one of them can be true).

most one of them can have the correct score. The tuple uncertainty model can be considered as a generalization of the data model without uncertainty, in which each tuple has probability one, and there are no mutual exclusion rules.

The second type of uncertainty is called *attribute uncertainty*. In this case, an attribute is uncertain and we model each value of the attribute as a probabilistic distribution. In the example of Figure 5(b), the measurements of the *Speed* attribute can have errors and we model each speed value by a normal distribution. This is in contrast with the traditional deterministic model in which each value of an attribute is a fixed scalar value. Attribute uncertainty may also be considered as a generalization of the data model without uncertainty, in which each value in an attribute is some value with probability one (i.e., a discrete distribution).

Not only do the two kinds of uncertainty exist in the source data, but they also exist in the *query result*. Let us look at an example. We take a simple table that has attribute uncertainty as shown in Figure 5(b). We then issue a query as in Figure 6(a). What would the result be? Each of the three tuples has a non-zero probability to satisfy the predicate "*Speed* > *78*". For example, the first tuple's Speed attribute has a normal distribution with mean 90 and variance 20, and thus has a high probability (say, 0.95) of satisfying the predicate. The second tuple, on the other hand, has a normal distribution with a low mean (62) and has a tiny probability (say, 0.001) of satisfying the predicate. Thus, we have tuple uncertainty in the query result (last column in Figure 6(a)).

What about the selected "*Speed*" attribute in the result set? We know that only if the *Speed* is above 78 should the tuple be in the result at all. Hence, we can reason that the *Speed* attribute in the result should not be in its original form, but rather, a conditional distribution (conditioned on the predicate being true) based on the original distribution. We illustrate this in Figure 6(b), which shows the example for the first result tuple. We cut off the original distribution *Normal* (90, 20) at the value 78, and only take the right side of the curve. Then, we need to normalize it (by multiplying by a constant factor) so that the function still integrates to 1, as a



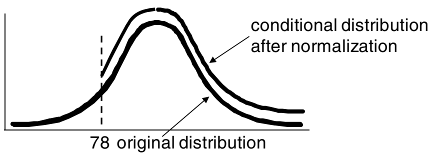
```
SELECT ObjectID, Speed FROM table
WHERE Speed > 78
```

Result?

Object ID	Speed	Prob.
28	?	0.95
11	?	0.001
72	?	0.5

attribute uncertainty
tuple uncertainty

(a)



(b)

Fig. 6 Illustrating tuple uncertainty and attribute uncertainty in a query result. We issue the query in (a) to the uncertain table in Fig. 4(b). Each of the three tuples has a non-zero probability to be in the result---this is tuple uncertainty (last column in (a)). The "Speed" in the result has attribute uncertainty – a conditional distribution shown in (b).

probability density function. We can see that the *Speed* attribute in the result is still distributions, and we have attribute uncertainty in the result.

In addition, we may have *sub-attribute* uncertainty for some data types. For instance, a text string attribute can have uncertain "characters" within it. As formalized by Jestes et al. (2010), a probabilistic string can have two models: the *string-level* model and the *character-level* model, which we define next.

Definition 23 (string-level and character-level models) (Jestes et al. 2010)**.** *Let* Σ *be an alphabet. A probabilistic string in the string-level model is represented as* $S = \{(s_1, p_1),\ (s_2, p_2), \dots, (s_m, p_m)\}$*, where* $s_i \in \Sigma^*$*,* $p_i \in (0,1]$*, and* $\sum_{i=1}^{m} p_i = 1$*. A character-level probabilistic string is* $S = S[1]S[2]\dots S[l]$*, where each character* $S[i] = \{(c_{i1}, p_{i1}), \dots, (c_{im_i}, p_{im_i})\}$*,* $c_{ij} \in \Sigma$*,* $p_{ij} \in (0,1]$*, and* $\sum_{j=1}^{m_i} p_{ij} = 1$*. That is, a string consists of independently distributed characters, some of which can be deterministic (i.e.,* $p_{i1} = 1$*).*

While a string-level probabilistic string follows the aforementioned attribute uncertainty (i.e., an attribute with a discrete distribution), a character-level model has distributions (of characters) embedded *inside* a string attribute, which is why it is termed *sub-attribute* uncertainty. Sub-attribute uncertainty has the finest granularity among the uncertainty models. Indeed, as shown by Ge and Li (2011), an index (for substring search) will point to uncertain character positions inside a string attribute, which can potentially be very long (e.g., millions, as in DNA strings).

4 Query Processing for Uncertain Data

We describe an algorithm that we devised to answer an *arbitrary* query on uncertain data (Ge and Zdonik 2008). The algorithm is called Statistical sampling for Equidepth Result distribution with Provable error-bounds, or SERP. SERP is essentially a Monte Carlo randomized algorithm.

4.1 The SERP Algorithm

The basic idea of a Monte Carlo algorithm for processing uncertain data is that we sample input data, run a query over the samples using a conventional query engine, and then learn a probability distribution for each random variable in the output, which includes any probabilistic field in a result tuple and a result tuple's existence probability in the result set.

The SERP algorithm uses a simple and consistent representation for both input data to queries and output query results, namely *equidepth histograms*. We consider probabilistic fields having continuous distributions. We can partition the domain of a probability density function (PDF) $f(x)$ into k intervals such that for each interval I, it holds that $\int_{x \in I} f(x)\, dx = \frac{1}{k}$. Thus, a distribution is "described" by k contiguous intervals and can be succinctly represented as $k + 1$ values indicating the boundaries of the k intervals: $(v_0, v_1, \ldots, v_k)$, where $[v_i, v_{i+1})$ is the ith interval. We assume a uniform distribution within an interval.

This is reminiscent of equidepth histograms widely used in query optimizers, and reflects the idea that the exact distribution of "high density areas" is more important and should be given higher "resolution". However, note the important difference that each bucket of an equidepth histogram contains a number of actual column values, whereas an equidepth distribution specifies the PDF of one attribute field. This representation is quite compact, only needing $k + 1$ values to describe a distribution.

Sampling from such a histogram representation is very simple: first pick one of the k intervals uniformly at random, and then pick a value from that interval uniformly at random. This sampling procedure will be used in the SERP algorithm. We consider the query execution as a black box that takes n input random variables (in general) and produce a number of output random variables. The n input random variables are either binary random variables indicating input tuple probability or probabilistic fields in the form of equidepth histograms as described above. Without loss of generality, we only need to consider how we obtain the distribution of one of the output probabilistic fields. Other fields are obtained in the same way. For example, for SUM or AVG, the inputs are the n uncertain fields in n tuples and the output is the result. The SERP algorithm (Ge and Zdonik 2008) is shown below.

```
Algorithm SERP($X_1, \ldots, X_n$)

   Input: $X_1, \ldots, X_n$: probabilistic fields in equidepth histogram distributions
   Output: the distribution of one output field

   // Do the main loop of the algorithm.
   // k is the number of intervals in a distribution, $\mu$ is to be determined later
1: for each $i \leftarrow 1, \ldots, k\mu$ do
2:        Sample each input $X_1, \ldots, X_n$ and get $x_1, \ldots, x_n$
3:        Run the query over $x_1, \ldots, x_n$ and let the output be $y_i$
4: end for
   // Get the output distribution
5: Sort the output values as $y_1, \ldots, y_{k\mu}$ where $y_i \leq y_{i+1}$
```

```
6: Get $k$ contiguous intervals, each containing $\mu$ output values; the first interval
   contains $y_1, \ldots, y_\mu$, the second contains $y_{\mu+1}, \ldots, y_{2\mu}$, and so on. More pre-
   cisely, let $v_0, \ldots, v_k$ be the interval boundaries, where $v_i = \frac{y_{i\mu}+y_{i\mu+1}}{2}$ $(1 \le i \le$
   $k-1)$, $v_0 = 2y_1 - y_2$, and $v_k = 2y_{k\mu} - y_{k\mu-1}$.
7: Return the $k$ contiguous intervals above as the result distribution.
```

In the algorithm, μ is a parameter that balances accuracy with performance, as we will investigate in the analysis. Note that we model all inputs as uncertain. In reality, some input values can be certain. It is straightforward to extend the algorithm to the mixed case. Also note that from one execution on the n samples to the next, to be more efficient, we can share the query plan (i.e., the query is compiled only once, and executed many times for each loop). Further, among different executions, sub-results of parts of the query plan that only refer to data without uncertainty can be shared. Another key optimization is on I/O cost. The database engine can pay the I/O cost only once, and incrementally carry out the multiple rounds of computation in parallel. It is easy to see that SERP is scalable. The cost is no more than a constant factor of that of the same operation on data without uncertainty, regardless of the number of tuples. Additionally, SERP works even if there is correlation between different inputs. We just need to carry out the sampling from the joint distribution.

4.2 Analysis of SERP

We measure the distance between the result distribution computed by some algorithm and an "ideal" one based on the same input distributions, but given as much computing resources as needed. We use a well-known distance metric: variation distance.

Definition 24 (variation distance) (Mitzenmacher and Upfal 2005). *The* variation distance *between two distributions D_1 and D_2 (each being a discrete probability distribution) on a countable state space S is given by $VD(D_1, D_2) = \frac{1}{2}\sum_{x \in S} |D_1(x) - D_2(x)|$.*

We first give some insights on the variation distance metric, as we will be using it for analysis.

Lemma 2 (Mitzenmacher and Upfal 2005). *Consider two distributions D_1 and D_2. For a state x in the state space S, if $D_2(x) > D_1(x)$, then we say D_2 overflows at x (relative to D_1) by the amount $D_2(x) - D_1(x)$. Likewise, if $D_2(x) < D_1(x)$, then we say D_2 underflows at x (relative to D_1) by an amount of $D_1(x) - D_2(x)$. We denote the total amount that D_2 overflows (and underflows, respectively) as P_{over} (and P_{under}, respectively). Then, $P_{over} = P_{under} = VD(D_1, D_2)$.*

We are now ready to present a novel proof that SERP has a nice bound on the variation distance between its result distribution and the ideal one, even though we do not know the exact form of the ideal result distribution, nor do we make any assumption on how to obtain it.

Theorem 7 (Ge and Zdonik 2008). *In the SERP algorithm, let* k *and* μ *be parameters as described in the algorithm. Then, with probability at least* $1 - k \cdot \left[\left(\frac{e^{2\delta}}{(1+2\delta)^{1+2\delta}}\right)^{\mu} + \left(\frac{e^{-2\delta}}{(1-2\delta)^{1-2\delta}}\right)^{\mu}\right]$, *the variation distance between the result distribution and the ideal one is no more than* δ $(0 < \delta < 0.5)$.

Proof. Consider any one interval I of the ideal distribution. Define $k\mu$ random variables X_i $(1 \le i \le k\mu)$ as follows:

$$X_i = \begin{cases} 1, & if\ y_i \in I\ in\ line\ 3\ of\ SERP \\ 0, & if\ y_i \notin I\ in\ line\ 3\ of\ SERP \end{cases}.$$

Because I is an interval of the ideal distribution, from the definition of the equi-depth partition, we have $\Pr[X_i = 1] = \frac{1}{k}$, and hence $E[X_i] = \frac{1}{k}$. We define a random variable $X = \sum_{i=1}^{k\mu} X_i$, indicating the number of output y_i's that fall in I. From the linearity of expectation, we have $E[X] = k\mu \cdot \frac{1}{k} = \mu$. As X is the sum of independent 0/1 random variables, we can apply Chernoff bounds that for any $0 < \delta < 0.5$, we have $\Pr[X > (1+2\delta)\mu] < \left(\frac{e^{2\delta}}{(1+2\delta)^{1+2\delta}}\right)^{\mu}$ and $\Pr[X < (1-2\delta)\mu] < \left(\frac{e^{-2\delta}}{(1-2\delta)^{1-2\delta}}\right)^{\mu}$. Then from the union bound,

$$\Pr[X > (1+2\delta)\mu\ or\ X < (1-2\delta)\mu] < \left(\frac{e^{2\delta}}{(1+2\delta)^{1+2\delta}}\right)^{\mu} + \left(\frac{e^{-2\delta}}{(1-2\delta)^{1-2\delta}}\right)^{\mu}.$$

Now consider all k intervals and apply the union bound again:

$$\Pr[\exists interval\ st. |X - \mu| > 2\delta\mu] < k \cdot \left[\left(\frac{e^{2\delta}}{(1+2\delta)^{1+2\delta}}\right)^{\mu} + \left(\frac{e^{-2\delta}}{(1-2\delta)^{1-2\delta}}\right)^{\mu}\right].$$

Hence,

$$\Pr[\forall interval, |X - \mu| \le 2\delta\mu] \ge 1 - k \cdot \left[\left(\frac{e^{2\delta}}{(1+2\delta)^{1+2\delta}}\right)^{\mu} + \left(\frac{e^{-2\delta}}{(1-2\delta)^{1-2\delta}}\right)^{\mu}\right].$$

Thus, with probability at least $1 - k \cdot \left[\left(\frac{e^{2\delta}}{(1+2\delta)^{1+2\delta}}\right)^{\mu} + \left(\frac{e^{-2\delta}}{(1-2\delta)^{1-2\delta}}\right)^{\mu}\right]$, all intervals contain sample result points whose number differs from the expected value by no more than $2\delta\mu$. As each such point carries weight $\frac{1}{k\mu}$ into the probability, and there are either no more than $\frac{k}{2}$ overflow intervals (holding more than μ points) or no more than $\frac{k}{2}$ underflow intervals, from Lemma 2, we get that the variation distance is no more than $2\delta\mu \cdot \frac{1}{k\mu} \cdot \frac{k}{2} = \delta$. □

To get a numerical sense about the bound, we take $k = 5$, $\delta = 0.2$, and $\mu = 60$. Then from Theorem 7, using 300 sample points (rounds), with probability at least 0.91, the variation distance between the result of the SERP algorithm and the ideal distribution is no more than 0.2. This is a (rather conservative) theoretical guarantee, and our experiments (Ge and Zdonik 2008) showed that, in practice, one can obtain a small variation distance with significantly fewer rounds. On the other hand, theoretical guarantees are important as they hold for any dataset while the result of a particular experiment depends on its data.

4.3 Join Query Semantics

We now focus on an important kind of query, namely join queries, on uncertain attributes. We show that there are two useful types of join operations specific to uncertain attributes: *value join* (v-join) and *distribution join* (d-join) (Ge 2011). V-join is a natural extension of the join operation on deterministic data. Let us first look at an example.

Table R

ID	Temperature
1	N (78, 5)
2	U (70, 75)
3	N (86, 10)
⋮	⋮

Table S

ID	Temperature
1	N (85, 6)
2	U (92, 94)
3	N (77, 8)
4	hist(...)
⋮	⋮

Fig. 7 Illustrating v-join between two uncertain attributes

Example 2 (v-join). *In Figure 7, we would like to examine the temperature attributes in table R and in table S, and find pairs that are very close. Note that both temperature attributes are uncertain and contain distributions, which appear in various forms. For instance, N(78, 5) denotes a normal distribution with mean 78 and variance 5, while U(70, 75) is a uniform distribution in the range [70, 75] and "hist(...)" indicates a histogram representation whose details we omit for clarity. The query is:*

SELECT *R.ID, S.ID* **FROM** *R, S*
WHERE $R.temperature \underset{1.0,0.8}{=} S.temperature$

This is called probabilistic threshold join query *in previous work* (*Cheng et al. 2006*). *The interpretation of the join predicate is that with probability at least* 0.8, *the difference between the two join attributes is no more than* 1.0 *degree, i.e.,* $|R.temperature - S.temperature\ | \leq 1.0$.

For uncertain attributes (either numerical or categorical), there is a special kind of join, which we call d-join. The idea of d-join is to treat probability distributions as "objects" and the join operation is based on the *similarity* of two distributions. We now look at some examples.

Example 3 (sensor fusion). *For high availability, five sensors redundantly measure the same environmental physical property (e.g., temperature) in a sensor network deployment on Great Duck Island (off the coast of Maine)* (Szewczyk et al.

2004). *Due to the harsh environment and the unreliable nature of the sensors, the readings can have large errors. A central database system performs a sensor fusion and uses machine learning techniques (e.g., kernel methods)* (Bishop 2007) *to obtain a temperature distribution from the five sensors. We record the temperature distributions at various times within two months in two tables (one for each month). We want to query for two time instances (one from each month) that have close temperatures.*

Example 4 (data integration). *Consider data integration from several sources. We need to perform schema matching and record linkage to combine different versions of the same data entity. However, due to schema and format inconsistencies, a data entity can have a lot of uncertainty. In the integrated database, we model the uncertainty with distributions (for either numerical or categorical values)* (Dong et al. 2009). *If two entities have similar distributions, then they are likely to be close. It is useful to find out this information.*

Example 5 (prediction queries). *We use different statistical models to predict the stock prices of a large number of companies one week from now* (Brockwell and Davis 2002). *Different models gave different results and again, by using techniques such as kernel methods* (Bishop 2007), *we can get a distribution of the predicted price of each company, which is stored in relational tables. The query is to ask for pairs of two companies that are likely to have very close stock prices at that time.*

In all these three examples, if we were to use v-join, even if two distributions are exactly the same, the probability that the join predicate is satisfied might still be *insignificant*. Here is a simple example. Suppose in Example 3, the five sensors give readings that are quite different (the difference is more than the v-join value difference parameter ε). Thus, the integrated temperature distribution has approximately five buckets, each with the same probability (1/5). Even if we were to do a v-join on two *identical* distributions as such, the probability that they are within ε apart would be only about $\sum_{i=1}^{5}\left(\frac{1}{5}\times\frac{1}{5}\right)=\frac{1}{5}$ (i.e., when both random variables fall into the same bucket). The observation here is that whether v-join is satisfied or not heavily depends on the "width" of the two distributions (i.e., the uncertainty, or, the entropy). V-join does not compare the two distributions *themselves*: two identical distributions may still fail to match. However, in all these examples, the fact that two *distributions* are close is also useful: it tends to indicate a special relationship of the two tuple entities that are being joined; i.e., their uncertain attributes are likely to be close in spite of the uncertainty. Essentially, we treat probability distributions themselves as *objects* and we are joining such objects. We are now ready to formalize v-joins and d-joins.

Definition 25 (domain partition scheme) (Ge 2011). *The* domain partition scheme *for an uncertain attribute is a many-to-one mapping of values in the domain of the uncertain attribute to a countable number of states.*

Example 6 (domain partition scheme). *If the domain of an uncertain attribute is all positive real numbers, then one possible domain partition scheme is based on a parameter* step*: we map all attribute values in the interval (0, step] to state 1, all values in (step, 2×step] to state 2, and so on.*

Consider two relations R and S that have uncertain attributes $R.A$ and $S.B$. In R, each record's A attribute is a probability distribution, rather than a single value, as in deterministic databases. The distribution can be encoded in various ways, including well-known distributions (e.g., a normal distribution) and histograms. The same is true for $S.B$.

We denote a join operation between R and S on attributes $R.A$ and $S.B$ as $R \bowtie_{A\theta B,\varepsilon,p} S$, where ε and p are optional parameters. There are two types of join: *value join* (*v-join*) and *distribution join* (*d-join*). A v-join has a join predicate that is an (approximate) equality or an inequality with some probability threshold. For example, a v-join predicate can be $R.A \underset{\varepsilon,p}{=} S.B$, which means $\Pr(|R.A - S.B| < \varepsilon) \geq p$. This is a probabilistic version of a *band join* (DeWitt et al. 1991); for deterministic data, when the predicate is $|R.A - S.B| < \varepsilon$, it is a band-join. Another example is $R.A \underset{p}{<} S.B$, which means $\Pr(R.A < S.B) \geq p$ (ε is not present here). ε usually denotes a small value and p is a probability threshold. Note that, when it is clear from the context, we often use $R.A$ to denote the random variable that represents the A attribute of *a tuple* in R, and likewise for $S.B$. A d-join predicate is denoted as $R.A \underset{\varepsilon}{\sim} S.B$. It is equivalent to $VD(R.A, S.B) \leq \varepsilon$, where $VD(R.A, S.B)$ denotes the *variation distance* (Definition 24) between a distribution in $R.A$ and a distribution in $S.B$, and $R.A$ and $S.B$ have a common set of states resulting from their domain partition schemes (Definition 25).

4.4 Efficiently Processing V-joins

4.4.1 Using the First Two Moments

Our query processing techniques for v-join are based on probability theory. Specifically, the kth *moment* of a random variable X is defined as $E[X^k]$. The moments are a concise way to describe the nature of the distribution of a random variable. The first moment is the expectation of the random variable while the first two moments determine the variance of the random variable: $\mathrm{Var}[X] = E[X^2] - (E[X])^2$. In fact, all moments of a variable together uniquely define its distribution (Mitzenmacher and Upfal 2005). Simply computing and storing the first two moments (or equivalently, the *expectation* and *variance*) of a random variable (in our context, an *uncertain attribute of a record* is a *random variable*) incurs little overhead but, as we show, is very useful in making quick decisions during v-join in order to improve the speed. In some well-known distributions, such as Gaussian, the expectation and variance come for free, since they are part of the description of the distribution.

Probabilistic Band Join. Perhaps the most often used v-join is the probabilistic band join; i.e., when the join predicate is $R.A \underset{\varepsilon, p}{=} S.B$. The basic method for evaluating this predicate is by computing a double integral of the form $p' = \int_{-\infty}^{\infty} f_1(x) \int_{x-\varepsilon}^{x+\varepsilon} f_2(y) dy\, dx$, where x is a random variable in $R.A$, y is a random variable in $S.B$ and $f_1(x)$ and $f_2(y)$ are the density functions of x and y, respectively. The result p' is the probability that $R.A$ and $S.B$ (of two tuples) are at most ε apart. Note that the v-join predicate is satisfied if and only if $p' \geq p$.

The problem with the above solution is that it is very CPU expensive. Therefore, we wish to use probability bounds to improve the speed of evaluating such a predicate. Define a random variable $X = R.A - S.B$. Then the predicate is equivalent to:

$$\Pr[|x| < \varepsilon] > p. \tag{1}$$

Let $E(X) = E(R.A) - E(S.B) = \lambda$. Then we have the following three cases, as shown in Figure 8.

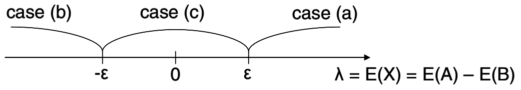


Fig. 8 Illustrating the three cases of λ

Case (a): $\lambda > \varepsilon$. We would like to know if (1) must be false, in which case we can exclude the tuple pair. From the *Cantelli's inequality* (Grimmett and Stirzaker 2001), we have:

$$\Pr[|X| < \varepsilon] \leq \Pr[X < \varepsilon] = \Pr[\lambda - X > \lambda - \varepsilon] < \frac{Var(X)}{Var(X)+(\lambda-\varepsilon)^2}.$$

If $\frac{Var(X)}{Var(X)+(\lambda-\varepsilon)^2} \leq p$, then condition (1) must be false.

Case (b): $\lambda < -\varepsilon$. Similar to case (a), we would like to use Cantelli's inequality to see if we can determine that (1) must be false and rule out the tuple pair:

$$\Pr[|X| < \varepsilon] \leq \Pr[X > -\varepsilon] = \Pr[X - \lambda > -\varepsilon - \lambda] < \frac{Var(X)}{Var(X)+(\lambda+\varepsilon)^2}.$$

If $\frac{Var(X)}{Var(X)+(\lambda+\varepsilon)^2} \leq p$, then condition (1) must be false.

Case (c): $-\varepsilon < \lambda < \varepsilon$. In contrast to the previous two cases, we would like to see if (1) must be *true* and hence the tuple pair satisfies the v-join condition. From Cantelli's inequality, we have:

$$\Pr[X > \varepsilon] = \Pr[X - \lambda > \varepsilon - \lambda] < \frac{Var(X)}{Var(X)+(\varepsilon-\lambda)^2},$$
$$\Pr[X < -\varepsilon] = \Pr[\lambda - X > \varepsilon + \lambda] < \frac{Var(X)}{Var(X)+(\varepsilon+\lambda)^2},$$
$$\Pr[|X| < \varepsilon] = 1 - \Pr[X > \varepsilon \text{ or } X < -\varepsilon] > 1 - \frac{Var(X)}{Var(X)+(\varepsilon-\lambda)^2} - \frac{Var(X)}{Var(X)+(\varepsilon+\lambda)^2}.$$

The last inequality is due to the union bound. Thus, if $1 - \frac{Var(X)}{Var(X)+(\varepsilon-\lambda)^2} - \frac{Var(X)}{Var(X)+(\varepsilon+\lambda)^2} \geq p$, then condition (1) must be true.

Now suppose we are *only* given the moments of the two fields being joined. For clarity, we write A for $R.A$ and B for $S.B$. For the above methods to work, we need to express λ and Var(X) using the moments of A and B. From the linearity of expectation, λ = E(X) = E(A) – E(B). With the typical assumption that A and B are independent, we have Var(X) = Var(A) + Var(B) = E(A^2) – E^2(A) + E(B^2) – E^2(B).

Therefore, only using the first two moments of A and B, we can quickly exclude the (A, B) pair from the join result (Cases a and b) or include it in the result (Case c) if the conditions in those cases are met. If the pair is neither excluded nor included, we need to resort to the “old-fashioned” way of computing the actual probability that $|A – B| < \varepsilon$ by a double integral (or summation if they are discrete) as in (1) to see if it is greater than p. We can save a great deal of computational cost by using moments and probabilistic bounds to make quick judgments.

Other Inequality V-joins. Thus far we have only considered probabilistic band join; we now turn to other inequality v-joins. We only demonstrate $R.A \underset{p}{<} S.B$; we can apply similar techniques to other inequalities. Again we define a random variable $X = R.A – S.B$. Let E(X) = λ. We now examine two cases:

Case (a): λ < 0. Then,

$$\Pr[R.A \geq S.B] = \Pr[X \geq 0] = \Pr[X - \lambda \geq -\lambda] < \frac{Var(X)}{Var(X)+\lambda^2}.$$

If $\frac{Var(X)}{Var(X)+\lambda^2} \leq 1 - p$, it must be true that $\Pr\,[R.A < S.B] \geq p$ and the predicate is satisfied.

Case (b): λ ≥ 0. Then we see if we can exclude the tuple pair:

$$\Pr[R.A < S.B] = \Pr[X < 0] = \Pr[\lambda - X > \lambda] < \frac{Var(X)}{Var(X)+\lambda^2}.$$

If $\frac{Var(X)}{Var(X)+\lambda^2} \leq p$, it must be true that $\Pr[R.A < S.B] < p$ and the tuple pair is excluded from the result. Details such as obtaining λ and Var(X) from the moments of $R.A$ and $S.B$ are the same as in the discussions for probabilistic band join.

We also devise indexing techniques for v-join queries. For additional details, we refer the reader to Ge (2011).

4.5 Efficiently Processing D-joins

4.5.1 The Condensed D-join Algorithm

In this section, we examine how we can process d-join queries efficiently. We can perform a d-join on two uncertain attributes if their domain partition schemes (Definition 3) result in a common set of states. Let the size of the state space S resulting from the domain partition schemes be n. Then the "features" of an uncertain distribution with respect to S can be described as $(p_1, p_2,\dots, p_n)$, meaning that the uncertain field has probability p_i of being in state s_i. By taking this vector, we can map an uncertain distribution to a point in the n-dimensional space. It is then easy to verify that the variation distance between two distributions exactly maps to half of the L1 distance between the two corresponding points in the n-dimensional space.

This discussion leads us to the direction that a d-join can be reduced to a *similarity join,* which is well studied in the database literature (e.g., Koudas and Sevcik 2000). There are many competing algorithms that can do similarity join. However, there is a common phenomenon among the algorithms: due to the "curse of dimensionality": as dimensionality increases, performance deteriorates significantly. For example, as shown in Koudas and Sevcik's Figure 17 (2000), the response time grows 17fold as the dimensionality increases from 3 to 20 with the same number of data points for both algorithms, as shown by Koudas and Sevcik (2000). We therefore propose an algorithm called *condensed d-join*, as shown below. The algorithm starts by reducing the dimensionality by a procedure called a *condensation scheme,* as we now define.

Definition 26 (condensation scheme) (Ge 2011). A *condensation scheme* for an uncertain attribute is an onto function f: $S \rightarrow S'$, where S is the original state space determined by the domain partition scheme of the attribute and S' is a state space with a smaller cardinality, i.e., $|S'| < |S|$. The space S' is called the *condensed state space*.

Algorithm CONDENSED-D-JOIN $(R.A, S.B, \varepsilon)$

```
Input: Two uncertain attributes R.A and S.B whose domain partition schemes have the same states; and value ε.
Output: Pairs of R.A and S.B that satisfy $R.A \underset{\varepsilon}{\sim} S.B$.
1: Precomputation step: Determine the best condensation scheme for either R.A or S.B using the algorithm in Section 4.5.2.
2: In the condensed state space determined in line 1, we get the new distributions for all fields in R.A and S.B. If a new state is the merge of a number of previous states, then its probability is the sum of the probabilities of the original states.
3: Phase 1: Use any existing similarity join algorithm to compute the join result based on the smaller state space, using the L1 distance metric and the distance parameter 2ε.
4: Phase 2: Among the qualified tuple pairs selected in line 3, further refine the selections by computing the variation distance (VD) over the original state space.
```

Line 1 of the algorithm is to determine the optimal condensation scheme according to any one side of the join and is typically pre-computed. The condensation algorithm combines a number of neighboring states into one and sums up their probabilities. We thus get the new probability distributions in line 2. Phase 1 of the condensed d-join is performed in line 3, where we essentially reduce the d-join problem to the similarity join of multidimensional points (with the parameter value 2ε). Because of the reduced dimensionality, it is much faster. Over the qualified tuple pairs, we perform the second phase, which is a post-processing as shown in line 4. The goal of the two phase approach is to avoid the slow performance caused by high dimensionality. The quality of the condensation scheme is of a critical role here since it impacts the number of false positives that must be filtered out in the post-processing phase. We study the optimal condensation scheme in detail in Section 4.5.2.

We show the correctness of the CONDENSED-D-JOIN algorithm.

Lemma 3 (Ge 2011). *Over the original state space determined by the domain partition scheme, let distribution D_1 come from R.A and D_2 come from S.B. After the condensation, let the distributions (in step 2) be D_1' and D_2'. Then $VD(D_1', D_2') \leq VD(D_1, D_2)$.*

Proof. Suppose the condensation scheme merges k states $s_1, s_2, \ldots, s_k$ into a single state s'. Let D_1 have probabilities $p_{11}, p_{12}, \ldots, p_{1k}$ and D_2 have probabilities $p_{21}, p_{22}, \ldots, p_{2k}$ in those states, respectively. Then step (2) of the algorithm indicates that D_1' has probability $p_1' = p_{11} + p_{12} + \ldots + p_{1k}$ in state s' while D_2' has probability $p_2' = p_{21} + p_{22} + \ldots + p_{2k}$ in state s'. It holds that:

$$\begin{aligned} |p_1' - p_2'| &= |(p_{11} - p_{21}) + (p_{12} - p_{22}) + \ldots + (p_{1k} - p_{2k})| \\ &\leq |p_{11} - p_{21}| + |p_{12} - p_{22}| + \ldots + |p_{1k} - p_{2k}|. \end{aligned}$$

Thus, iterating this over all states of D_1' and D_2', summing up the inequalities as produced above, and finally dividing both sides of the resulting inequality by 2, we get $VD(D_1', D_2') \leq VD(D_1, D_2)$, which directly follows from the definition of *VD*. □

Theorem 8 (Ge 2011). *The* CONDENSED-D-JOIN *algorithm gives the correct result.*

Proof. From Lemma 3, we know that if, in the original state space, the *VD* between two distributions is less than ε, then it must also be true after the condensation. Thus, phase 1 of the d-join (step 3) will not miss any result tuples that should be returned. Finally, the second phase of the algorithm filters out all false positives. □

4.5.2 The Optimal Condensation Scheme

Consider an uncertain attribute and a condensation scheme that reduces the number of its states from n to k ($k < n$). The question now is how we should merge the states in the original state space S. Let us look at a motivating example. Figure 9(a) shows the distributions of an uncertain attribute. The solid vertical lines describe the domain partition scheme: an interval between two neighboring lines is a state. Suppose the (blue) dotted lines indicate a potential condensation scheme: there are three condensed states: the interval $[v_1, v_2)$ is the first condensed state, $[v_2, v_3)$ is the second, and $[v_3, v_4]$ is the third. It appears to be a fair condensation scheme as each condensed state contains about the same number of the original states. However, let us suppose that 1000 distributions fall in the middle range, i.e., between v_2 and v_3, while there is only one distribution in the first and third condensed states, respectively.

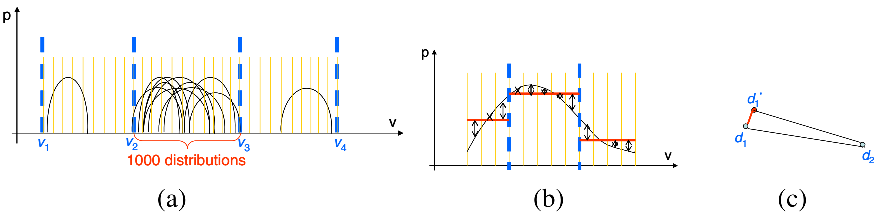


Fig. 9 (a) The necessity of a good condensation scheme, (b & c) illustrating the concept of "minimum disturbance" of a condensation scheme as measured by variation distance

Then this condensation scheme loses a lot of information: all of the 1000 distributions have the same distribution (0, 1, 0) in the condensed state space (each number is the probability of one state). In other words, it is not discriminative. In the condensed space, if one of the 1000 distributions matches with a distribution in another column for d-join, so will all other 999 distributions. We therefore need a principled algorithm to make the condensation scheme more *discriminative*.

But how to make it discriminative? The idea is to make the new distributions after applying the condensation scheme as *faithfully* as possible to the original ones. The faithfulness is again measured by variation distance. Clearly condensation would lose some information about the distributions. Thus, we would prefer a scheme that would result in new distributions that have the minimum distance from the original ones, which we call *minimum disturbance*. It is quantified by the sum of the variation distances between each new distribution and its original one, in the *original* state space. Since a new distribution can be considered as a *lossy compression* of the original one, when we convert the new distribution back to its original state space, we simply divide the probability of a condensed state by the number of the original states that map to it. This is because we do not distinguish between those states in the condensed space.

Therefore, when computing the variation distance between an original distribution and the new one, we compare the probabilities of the original states with their averages in each group, where each group corresponds to a condensed state. We illustrate this in Figure 9(b). In Figure 9(c), the original distribution is mapped to a point d_1 in the multi-dimensional space. For d-join we need to compute the distance between d_1 and a point d_2 that represents a distribution in another column. The condensation step brings d_1 to another point d_1'. The point d_1' corresponds to the conversion of the new distribution back to its original state space, which we describe in the previous paragraph. Even though d_2's position is not known *a priori*, by minimizing the distance between d_1 and d_1', the distance between d_1 and d_2 is optimally approximated by the distance between d_1' and d_2. Moreover, this optimization problem is over *all* distributions in an uncertain column.

We first formalize the problem (Ge 2011). An uncertain column has N probability distributions. The column follows a domain partition scheme that consists of n states in some serial order (e.g., n small buckets in value order). The goal of our condensation scheme is to merge some neighboring states in order to reduce them to k states ($k < n$). The scheme is chosen in such a way that the variation distance between the new distribution and the original one, summing over all records of the column, is minimized.

Let us denote the optimal (i.e., minimum) sum value (over the whole column) of the *variation distances* between the new distributions and the original ones as $D(k, n)$, where k is the target number of condensed states and n is the original number of states. We then have the following recursion:

$$D(k,n) = \min_{k \le i \le n} [\, D(k-1, i-1) + \sum_{r=1}^{N} C^{(r)}(i,n) \,] \tag{2}$$

where $C^{(r)}(i, n)$ is the "cost" of merging states from i to n into a single new state for the distribution in record r. This cost is just the part of the variation distance between the two distributions at states from i to n. More precisely (recall Figure 8 b & c),

$$C^{(r)}(i,n) = \frac{1}{2}\sum_{j=i}^{n} |\, p_j^{(r)} - a_i^{(r)} \,|, \; where \; a_i^{(r)} = \frac{1}{n-i+1}\sum_{j=i}^{n} p_j^{(r)} \quad .$$

(3)

Here $p_j^{(r)}$ is the probability of the jth state in the rth distribution. We also note the boundary condition that

$$D(1,i) = \sum_{r=1}^{N} C^{(r)}(1,i), \quad for\, 1 \le i \le n-k+1 \quad . \tag{4}$$

We then can have an efficient dynamic programming algorithm for this problem, as illustrated in Figure 10(a). The figure shows a "D table" for values of the D function in Equation (1). The row numbers of the D table (1 to k) are the first parameter of the function while the column numbers (1 to n) are the second parameter. Our target value is $D(k, n)$, which is indicated by the red "?" at the bottom right corner of the D table. From the recursion in Equation (1), the target value can be obtained from the values in the row above, assuming we already have all the C values. The whole process can be recursively applied for each cell in the table. We therefore have a top-down procedure to fill in the shaded region in Figure 10(a) row by row, starting from the boundary condition as described in Equation (4).

In the above algorithm, we assume that we have all the C values. We now describe how to obtain them. We simply do a scan of the whole uncertain column and compute the aggregation of the C values as described in Equation (3). Figure 10(b) illustrates the C table. It is not hard to verify from Equations (2) and (4) that we only need to fill in the shaded region of the C table (C is a two-dimensional array). As we scan the column and get each distribution, we obtain the C values of the shaded region using Equation (3). Because in Equations (2) and (4) we require a sum of the C values over all distributions, we do the aggregation (sum) for each cell of the shaded region of the C table as we scan each distribution of the column

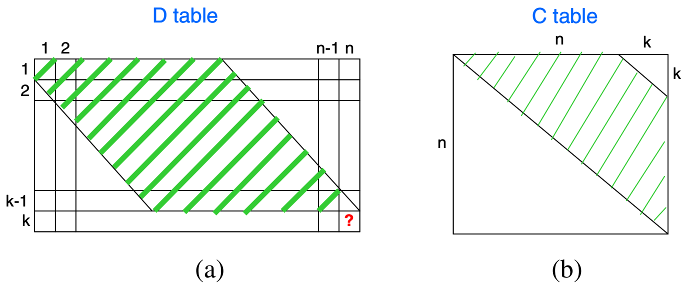


Fig. 10 A dynamic programming algorithm to get the D function (a) and a column scan to obtain the aggregated C values (b)

one by one. Eventually, when we finish scanning the uncertain column, each cell of the C table contains a sum value. Combining the above two algorithms (i.e., getting the C table followed by getting the D table), we have an efficient method to obtain the optimal condensation scheme.

5 Related Work

There has been substantial work on managing uncertain data and information in recent years due to the rise of new applications that demand this capability.

Nilsson's seminal paper on probabilistic logic (Nilsson 1986) introduced the notion of reasoning with probabilities to the field of artificial intelligence. The possible worlds semantics proposed by Nilsson has become a *de facto* standard for interpreting statements about probabilities both in the field of AI and, a bit later, in the field of databases. The key idea expressed by Nilsson is that in a world with multiple discrete random variables with finite domains, each random variable must take an exact value, and the uncertainty essentially expresses the lack of information an observer has. Each assignment of values to all random variables in the model (universe) is known as a *possible world.* If a probability is associated with each possible world, then a probability of a random variable taking a specific value is computed as the sum of probabilities of all possible worlds in which this assignment occurs.

The first work on probabilistic databases did not directly use the possible world semantics. However, the proposed frameworks were consistent with it. Cavallo and Pittarelli (1987) were perhaps the first who studied probabilistic data in the context of databases. They proposed a framework in which a probabilistic relation represented a single probability distribution. Tuples in such relations represented the probabilities of specific outcomes. Cavallo and Pittarelli defined two query algebra operations for working with such data: join, which produced a joint probability distribution for a pair of probabilistic relations, and selection, which returned the probabilities of specific outcomes.

Most of the work that followed the work of Cavllo and Pittarelli (1987), however, adopted a different view about what should be represented by a probabilistic relational table. Just as individual tuples in classical relational tables represent *independent* statements of fact, in these approaches each tuple in a probabilistic table represents a single probability distribution. The first to propose this approach in early 1990s were Barbara, Garcia-Molina and Porter (1992). In their framework a relation had a set of *certain* attributes that jointly formed a primary key, and a collection of *uncertain* attributes, over which a probability distribution was defined. Dey and Sarkar (1996) built a 1-NF representation of the probabilistic relations of Barbara et al. (1992) and described an extensive query algebra, which included such probabilistic database-specific operations as data compaction/ coalescence and conditionalization operations. The former, given two or more probability distributions for the same event, produces a *consensus* probability distribution. The latter computes the conditional probability distribution conditioned on a specific value of one or more of the uncertain attributes.

Zimanyi explicitly introduced possible worlds semantics of Nilsson to probabilistic databases (Zimanyi 1997). His framework uses the language of first-order probabilistic logic introduced by Halpern (1990). Zimanyi treats each probabilistic relation as a formula in the first-order probabilistic logic. He then defines full query algebra on by specifying how the formulas describing the probabilistic relations change when the operation is performed. While this approach is not very practical, it provides a clear semantics for query algebra operations.

In the mid- to late 1990s, a number of research groups extended probabilistic relational database frameworks. The work on the Semistructured Probabilistic Databases model (SPO) described in this chapter takes its roots from two such directions. The first is the work of Kornatzky and Shimony who proposed the first object-oriented probabilistic database framework (Kornatzky and Shimony 1994). This was, to our knowledge, the first extension of the work on probabilistic databases that extended beyond relational database model. In the framework of Kornatzky and Shimony, uncertainty was associated not just with the specific values of attributes, but also with the hierarchical structure of the objects themselves.

The second precursor to the work on the SPO model was ProbView, a relational probabilistic database management system which was the first framework to introduce *interval probabilities* to represent uncertainty in data. (Lakshmanan et al. 1997). ProvView stored the data in a compact, non-1-NF form, similar to the approach of Barbara et al. (1992). The semantics of the data though was defined using the 1-NF *annotated* probabilistic relations similar to those considered by Dey and Sarkar (1996). The query algebra operated on the annotated relations, so to answer a query ProbView translated the data into annotated form and performed the requisite operations.

In late 1990s, research on specific types of uncertainty in data appeared. Dyreson and Snodgrass (1998) considered temporal indeterminancy and introduced a probabilistic temporal database framework. Dekhtyar, Ross and Subrahmanian (2001) adopted and improved the ProbView approach to management of probabilities in temporal databases.

The SPO model (Dekhtyar et al. 2001, Zhao et al. 2005) came out of the observations that most of the probabilistic database frameworks at the time were not designed to store arbitrary probability distributions of multiple discrete random variables (with finite domains). A semistructured probabilistic object can store any probability distribution regardless of how many and which random variables are in it, what meta-data about the variables is present and known, and whether the probability distribution is conditional. The SPO framework was further relaxed by introducing interval probabilities to represent uncertainty (Goldsmith et al. 2003, Zhao et al. 2003, Zhao et al. 2004).

In parallel with our work on the SPO model, a number of alternative frameworks for management of uncertainty using semistructured data models and XML emerged. Hung et al. (Hung et al. 2003, Hung et al. 2003A) proposed Probabilistic XML framework (PIXML) to encode information about probability distribution. Nierman and Jagadish (2002) introduced ProTDB framework for the same purpose.

Some work addresses imprecise and uncertain data in sensor networks (Cheng et al. 2003, Deshpande et al. 2004, Tran et al. 2010, among others). The work presented in Sections 3 and 4 differs from the sensor network-based work in that we can process arbitrary query types based on the possible world semantics, but are not restricted to specific query operators. Note, however, that some of these approaches were shown to be more efficient (Tran et al. 2010). Independently, the MCDB project at University of Florida and IBM (Jampani et al. 2008) also employed Monte Carlo query processing for uncertain data, and focused on efficient integration of their techniques into a system. Previous work that is based on tuple uncertainty includes that by Dalvi and Suciu (2004) and by Benjelloun et al. (2006). In this chapter, our proposed techniques focus on attribute uncertainty, which is common in a number of application domains that we present.

Cheng et al. (2006) proposed probabilistic threshold join, which is similar to our v-join semantics. However, their query processing is based on x-bounds, which are a number of bounds for some data structure (e.g., each data page), unlike our dynamic filtering bounds using probability theory. In addition, we study indexing for efficient joins and a second type of semantics, d-join, which is useful for different applications.

Similarity join on data points in multidimensional space is well studied (e.g., Koudas and Sevcik 2000). The connection between this line of work and our work on d-join is due to the fact that we can reduce a d-join to a similarity join. However, when the dimensionality is high, with any existing technique, there is invariably a significant performance penalty. Our design of the condensed d-join and the optimal condensation scheme are a novel contribution. Dimensionality reduction is also studied for indexing time series databases (e.g., Keogh et al. 2001). However, a salient difference in discrete probability distributions than time series features is the constraint that probability values are between 0 and 1 and sum to 1. We take advantage of this and devise a simple, efficient, and optimal condensation algorithm. Finally, band-join on deterministic data was studied by DeWitt et al. (1991). V-join can deal with a variant of band-join on uncertain data, where the old techniques cannot be applied.

6 Conclusions

We have introduced several approaches to computation with probabilities, and given introductions to databases to support these approaches. The first approach, the Semistructured Probabilistic Objects framework, treats joint and conditional probability distributions as fundamental data objects. This supports reasoning with Bayesian networks, hidden Markov models (HMMs), and other probabilistic graphical models. We have discussed the queries possible with standard probability distributions, and mentioned some of the issues that arise when probability intervals are used. We also mention two implementations of the SPDBMS. The second approach, the tuple and attribute uncertainty framework, takes a data-centric approach and more tightly couples probability distributions with "data" itself. That is, either entities (tuples) or their properties (attributes) are extended with probabilities. We have discussed the semantics of general SQL queries, including joins, in this framework, and proposed some efficient query processing techniques.

We have presented a number of application scenarios in which significant uncertainty is present, and in which each of our approaches would be useful. We believe that a broad range of applications, albeit all containing uncertain information, would require database techniques tailored to their specific requirements.

As future work, it would be interesting to seamlessly integrate our approaches and produce an even more powerful system that can meet the diverse needs of modern applications.

References

Barbará, D., Garcia-Molina, H., Porter, D.: The Management of Probabilistic Data. IEEE Trans. Knowl. Data Eng. 4(5), 487–502 (1992)

Benjelloun, O., Das Sarma, A., Halevy, A., Widom, J.: ULDBs: Databases with Uncertainty and Lineage. In: VLDB (2006)

Bishop, C.: Pattern Recognition and Machine Learning. Springer (2007)

Block, C., Collins, J., Ketter, W.: Agent-based competitive simulation: Exploring future retail energy markets. In: Twelfth International Con-ference on Electronic Commerce, ICEC 2010, pp. 67–76. ACM (August 2010)

Brockwell, P., Davis, R.: Introduction to Time Series and Forecasting, 2nd edn. Springer Texts in Statistics (2002)

Burton, P., et al.: Size matters: just how big is BIG? – Quanti-fying realistic sample size requirements for human genome epidemiology. International Journal of Epidemiology 38, 263–273 (2009)

Cavallo, R., Pittarelli, M.: The Theory of Probabilistic Databases. In: VLDB, pp. 71–9 (1987)

de Campos, L.M., Huete, J.F., Moral, S.: Uncertainty Management Using Probability Intervals. In: Proc. International Conference on Information Processing and Management of Uncertainty (IPMU 1994), pp. 190–199 (1994)

Cheng, R., Kalashnikov, D., Prabhakar, S.: Evaluating probabilistic queries over imprecise data. In: SIGMOD (2003)

Cheng, R., Singh, S., Prabhakar, S., Shah, R., Vitter, J., Xia, Y.: Efficient Join Processing over Uncertain Data. In: CIKM (2006)

Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. In: VLDB (2004)

Dekhtyar, A., Goldsmith, J., Hawkes, S.R.: Semistructured Probalistic Databases. In: Proc. SSDBM, pp. 36–45 (2001)

Dekhtyar, A., Ross, R.B., Subrahmanian, V.S.: Probabilistic temporal databases, I: algebra. ACM Trans. Database Syst. 26(1), 41–95 (2001)

Dekhtyar, A., Kevin Mathias, K., Gutti, P.: Structured Que-ries for Semistructured Probabilistic Data. In: Proc. 2nd Twente Data Manage-ment Workshop (TDM), pp. 11–18 (June 2006)

Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J.M., Hong, W.: Model-driven data acquisition in sensor networks. In: VLDB (2004)

DeWitt, D., Naughton, J., Schneider, D.: An Evaluation of Non-Equijoin Algorithms. In: VLDB (1991)

Dey, D., Sarkar, S.: A Probabilistic Relational Model and Algebra. ACM Trans. Database Syst. 21(3), 339–369 (1996)

Dong, X., Halevy, A., Yu, C.: Data integration with uncer-tainty. The VLDB Journal (April 2009)

Dyreson, C.E., Snodgrass, R.T.: Supporting Valid-Time Indeterminacy. ACM Trans. Database Syst. 23(1), 1–57 (1998)

Ge, T.: Join Queries on Uncertain Data: Semantics and Efficient Processing. In: The Proceedings of the IEEE 27th International Conference on Data Engineering (ICDE 2011), Hannover, Germany (April 2011)

Ge, T., Li, Z.: Approximate Substring Matching over Uncertain Strings. The Proceedings of the VLDB Endowment (PVLDB Journal) 4(11), 772–782 (2011)

Ge, T., Zdonik, S.: Handling Uncertain Data in Array Database Systems. In: Proceedings of the IEEE 24th International Conference on Data Engineering (ICDE 2008), Cancun, Mexico (April 2008)

Goldsmith, J., Dekhtyar, A., Zhao, W.: Can Probabilistic Databases Help Elect Qualified Officials? In: Proceedings FLAIRS 2003 Conference, pp. 501–505 (2003)

Grimmett, G., Stirzaker, D.: Probability and Random Processes, 3rd edn. Oxford (2001)

Halpern, J.: An Analysis of First-order Logic of Probability. Artificial Intelligence 46(3), 311–350 (1990)

Hung, E., Getoor, L., Subrahmanian, V.S.: PXML: A Probabilistic Semistructured Data Model and Algebra. In: ICDE (2003)

Hung, E., Getoor, L., Subrahmanian, V.S.: Probabilistic Interval XML. In: ICDT 2003, pp. 358–374 (2003)

Jaffray, J.: Bayesian Updating and Belief Functions. IEEE Trans. on Systems, Man and Cybernetics 22(5), 1144–1152 (1992)

Jampani, R., Xu, F., Wu, M., Perez, L., Jermaine, C., Haas, P.: MCDB: A Monte Carlo Approach to Managing Uncertain Data. In: SIGMOD (2008)

Jestes, J., Li, F., Yan, Z., Yi, K.: Probabilistic String Similarity Joins. In: SIGMOD, pp. 327–338 (2010)

Keogh, E., Chakrabarti, K., Mehrotra, S., Pazzani, M.: Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. In: SIGMOD (2001)

Komatsu, K., et al.: Gene expression profiling following constitutive activation of MEK1 and transformation of rat intestinal epithelial cells. Molecular Cancer 5, 63 (2006)

Kornatzky, Y., Shimony, S.E.: A Probabilistic Object-Oriented Data Model. Data Knowl. Eng. 12(2), 143–166 (1994)

Koudas, N., Sevcik, K.: High Dimensional Similarity Joins: Algorithms and Performance Evaluation. In: TKDE (2000)

Lakshmanan, L.V.S., Leone, N., Ross, R.B., Subrahmanian, V.S.: ProbView: A Flexible Probabilistic Database System. ACM Trans. Database Syst. 22(3), 419–469 (1997)

Mann, M., Hendrickson, R., Pandey, A.: Analysis of Proteins and Proteomes by Mass Spectrometry. Annu. Rev. Biochem. 70, 437–473 (2001)

McDonald, M.: To Build a Better Grid. NY Times. July 28 (2011)

Mitzenmacher, M., Upfal, E.: Probability & Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge U. Press (2005)

Nierman, A., Jagadish, H. V.: ProTDB: Probabilistic Data in XML. In: VLDB 2002, pp. 646–657 (2002)

Nilsson, N.J.: Probabilistic Logic. Artificial Intelligence 28(1), 71–87 (1986)

Ng, R., Subrahmanian, V.S.: Probabilistic Logic Programming. Inf. Comput. 101(2), 150–201 (1992)

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

Rosson, E.: Native XML Support for Semistructured Probabilistic Data Management, M.S. Thesis, Department of Computer Science, California Polytechnic State University (May 2008)

Szewczyk, R., et al.: An analysis of a large scale habitat monitoring application. In: SenSys (2004)

Tatbul, N., Buller, M., Hoyt, R., Mullen, S., Zdonik, S.: Confidence-based Data Management for Personal Area Sensor Networks. In: DMSN (2004)

Thiagarajan, A., Ravindranath, L., LaCurts, K., Mad-den, S., Balakrishnan, H., Toledo, S., Eriksson, J.: VTrack: Accurate, Energy-Aware Road Traffic Delay Estimation Using Mobile Phones. In: SenSys (2009)

Tran, T., Peng, L., Li, B., Diao, Y., Liu, A.: PODS: A New Model and Processing Algorithms for Uncertain Data Streams. In: SIGMOD (2010)

Walley, P.: Statistical Reasoning with Imprecise Probabilities. Chapman and Hall (1991)

Weichselberger, K.: The theory of interval-probability as a unifying concept for uncertainty. Int. J. Approx. Reasoning 24(2-3), 149–170 (2000)

Zhao, W., Dekhtyar, A., Goldsmith, J.: Query algebra operations for interval probabilities. In: Mařík, V., Štěpánková, O., Retschitzegger, W. (eds.) DEXA 2003. LNCS, vol. 2736, pp. 527–536. Springer, Heidelberg (2003)

Zhao, W., Dekhtyar, A., Goldsmith, J.: Databases for interval probabilities. Int. J. Intell. Syst. 19(9), 789–815 (2004)

Zhao, W., Dekhtyar, A., Goldsmith, J.: A Framework for Management of Semistructured Probabilistic Data. J. Intell. Inf. Syst. 25(3), 293–332 (2005)

Zimányi, E.: Query Evaluation in Probabilistic Relational Databases. Theor. Comput. Sci. 171(1-2), 179–219 (1997)