Bulletin of the Technical Committee on

Data Engineering

March 2006 Vol. 29 No. 1 **IEEE Computer Society**

Letters

Special Issue on Probabilistic Data Management

Conference and Journal Notices

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems.

There are two Data Engineering Bulletin web sites: http://www.research.microsoft.com/research/db/debull and http://sites.computer.org/debull/.
The TC on Data Engineering web page is http://www.ipsi.fraunhofer.de/tcde/.

Letter from the Editor-in-Chief

The Data Engineering Conference ICDE'06

The next International Conference on Data Engineering (ICDE'06) will be held in Atlanta in April, 2006. This conference is the flagship conference of the IEEE Technical Committee on Data Engineering. Atlanta is a great venue, and April is a wonderful time to visit the city, with balmy weather and with magnolias and peach trees in bloom. This year's conference is sure to be of very high quality as submissions continue to be very very high. The result is a very selective conference with high quality papers.

Working Group on Self-Managing Database Systems

I want to draw your attention to the letter from Sam Lightstone about a new working group. The area of self-managing database systems is both interesting technically, and important in the commercial world. So I urge you to read Sam's letter.

The Current Issue

The database community mostly lives in the realm of precise information. Occasionally, that information turns out to be wrong, at which point we endeavor to correct it. So we are really addicted to precise answers for our queries.

However, the "real" world is not so precise or exact. When we measure temperature, there is an uncertainty in the reading. When we deal with polling information, there is a margin of error that needs to be considered. When looking at information, how much credence to place in it is a function of the information source. So there are several senses for the word "probabilistic" and in the current issue, several approaches involving several notions of this word are pursued. One might say that this area "welcomes database folks to the real world".

The editorial arrangement for the current issue is a bit unusual. Minos Garofalakis is a current Bulletin editor. Dan Suciu is a "future" Bulletin editor. They are both interested in probabilistic databases and have collaborated on the current issue. (They will collaborate on another before they are done.) The issue is a fine survey of work in this area, which has become one of the new areas that have attracted attention in the last couple of years. Minos and Dan have succeeded in garnering submissions from some of the leading researchers in this exciting area. I want to thank them both for their hard work and for the successful result. I believe you will almost surely "probably" enjoy this issue.

David Lomet
Microsoft Corporation

Letter from Chair of Working Group on Self-Managing Database Systems

Its my pleasure to announce to the formation of the Data Engineering Workgroup on Self-Managing Database Systems (DEW-SMDB), a formal workgroup of the IEEE Technical Committee on Data Engineering (TCDE). http://db.uwaterloo.ca/tcde-smdb/

To advance research and development in self-managing database and information management systems the workgroup will sponsor workshops in conjunction with the ICDE, foster publications, and maintain a collection of key resource links on its website. I am pleased that our initial executive includes some of the leading advocates of self-managing database technology from across geographies in both industry and academia.

Alfred Whitehead, the preeminent American mathematician once wrote that “Civilization advances by extending the number of important operations which we can perform without thinking about them.” Database technology and in particular enterprise scale relational database technology, has reached a milestone in advancement of feature rich capabilities and massive data volumes which compels a quantum improvement in the administration profile of these systems.

Since the development of the relational model by E.F. Codd at IBM, relational databases have become the de facto standard for managing and querying structured data. The rise of the Internet, online transaction processing, online banking, and the ability to connect heterogeneous systems have all contributed to the massive growth in data volumes over the past 15 years. Terabyte sized databases have become commonplace. Concurrent with this data growth, have come dramatic increases in CPU performance, spurred by Moores Law, and improvements in disk fabrication which have improved data density limits for persistent disk storage. Relational databases lie behind 70% of the world’s structured data, and much of the world’s unstructured and semi-structured data is evolving into structured management systems as well. Practically everywhere data is processed in significant volume there is an information management system. To help manage the massive volume of data, and exploit the available hardware, modern Relational Databases Systems (RDBMSs) have been enhanced with powerful features to maximize efficiency and availability, providing numerous topology options for structuring the data and the storage, for indexing and caching, and for managing system workloads. These features are much needed in environments where systems can contain hundreds of disks and CPUs, massive volumes of data, and are often required to run 24 hours a day, 7 days a week (24x7).

The powerful capabilities of modern RDBMSs add complexity to the system too however, and have given rise to an entire professional domain of experts, known as Database Administrators (DBAs), to manage these systems. The problem is not unique to RDBMSs, but is ubiquitous in technology today. Some have called this explosive growth of complexity creeping featurism. In fact many companies now spend more money recruiting administrators to manage their technology than the money they have spent on the technology itself.

The only real viable long term strategic solution is to develop systems that manage themselves. Such systems are called self-managing or autonomic systems.

The following challenges in information management require the industrys attention:

Self-Configuring refers to function that connects and configures components so that technology is operational with minimal human intervention. For database servers this can include server and storage topology configuration, installation, initial database design, and configuration of the database server (there are several dozen possible options). Self configuring also includes the ability for a database server to be easily deployed, configured and interconnected with other middleware and applications.

Self-Healing refers to the ability of a system to detect problems or problem trends (which have not yet manifested as problems) and either take corrective action, or provide advice and automated notification to an end user who can then take corrective action. With all the sophistication and redundancy in database systems, things can and will occasionally fail, which makes the automated problem determination aspect of self-healing a critical domain. In automated problem determination, a self-managing system detects problem trends (such as disk storage is filling up at a detectable rate, from which storage constraint can be projected), or detects and notifies administrators when sudden errors arise.

Self-Optimizing is the domain of system performance tuning. For databases this includes tuning of subtle I/O and caching parameters, memory distribution, and physical database design. Physical database design is the broad area of database administration that deals with the selection of attributes that structure and control the way data is physically stored. Some common physical database design problems include: determining ways to cluster (sort) data on disk, which indexes (fast lookup structures, such as B+ trees) should be created, and what subset of the data should be pre-computed or replicated (materialized views).

Trust in self-managing systems. Even as we advance the underlying technology of self-managing systems, we need to concomitantly advance the trust of the administrators and executives who depend on these new self-managing capabilities. Building trust is as much to do with emotional invest and risk assessment as much as with technology.

Benchmarking of self-managing information systems. How can we objectively assess the success of data management systems to be self-managing? Just as benchmarks have been created in other domains to help assess the maturity and performance of system capabilities (such as database query or transaction performance) the development of industry standardized benchmarks to evaluate attributes of self-management will further compel the industry to compete for TCO, just as it has done historically for system performance.

Self-Protecting systems have the ability to anticipate, identify and protect against security attacks. This scope varies from static defenses such as virus protection and data encryption to more advanced dynamic defenses such as behavior analysis where user/application activity can be analyzed for negative behaviors, tracked, and adaptively undone.

System-wide self-management. Finally but not least, the holy grail of systems administration comes through not only creating piece-wise self-managing components (such as a self-managing database management system) but also by enabling large complex stacks of hardware and software in a truly cooperative self-managing system. This cooperative notion can only be achieved through the development of standards and multi-component technology.

The future evolution of these technologies will lead to self-managing systems that are require a minimal amount of human interaction, and a maximum amount of efficiency, meaning that information management systems will become both easy to administer and resource efficient.

Its our pleasure to announce the formation of this workgroup, and we invite everyone interested to follow our web site for news and announcements.

“Any intelligent fool can make things bigger and more complex... It takes a touch of genius - and a lot of courage to move in the opposite direction.” - Albert Einstein

Sam S. Lightstone
IBM Corporation

Letter from the Special Issue Editors

Several classes of applications have recently emerged that require database systems to manage and process *probabilistic data*. For example, large-scale data integration systems have to cope with several issues of data quality that can often be expressed naturally using probabilistic information; data acquisition systems such as sensornet or RFID deployments can produce huge volumes of raw data that is often error-prone and is best described through a probabilistic model; data extracted automatically from large text corpora is often imprecise and best modeled probabilistically. There have been attempts in the past to extend database management systems with the capability of storing and processing probabilistic data, but the technical challenges involved are enormous and remain, for the most part, unaddressed.

The collection of eight articles in this special issue represents an interesting sample of different research perspectives, efforts, and novel applications in the vibrant area of large-scale *probabilistic data management*. The first four papers in the issue discuss novel algorithmic and systems aspects of probabilistic databases. The paper by Benjelloun et al. presents a prototype probabilistic database system developed at Stanford University that adds both data uncertainty and data lineage as first-class concepts. The authors give a thorough discussion of various facets of the system (e.g., data model, query language, lineage management) in a unified framework, and describe their current implementation effort. The paper by Green and Tannen offers a concise, yet very thorough, comparison of the various proposed models for both *incomplete* and *probabilistic* databases, thus providing an excellent entry point to the rich literature on both topics. In the third article, Ré et al. discuss the evaluation of SQL queries over probabilistic databases, and give interesting theoretical results demonstrating the boundary between "easy" and "hard" queries. In a nutshell, they show that while some queries can be evaluated by simple manipulation of probabilities (a process that can be easily pushed inside relational database engines), others have high theoretical complexity making evaluation intractable. Unfortunately, much of the database research on probabilistic databases has evolved independently from the much richer area of *probabilistic models* in machine learning and knowledge representation. The paper by Getoor tries to fill this gap, by providing a very accessible discussion of probabilistic graphical models and their extensions to relational databases.

The final four articles in the issue discuss four novel application domains and the associated requirements on a probabilistic data management system. Jayram et al. describe **AVATAR**, a high-precision, rule-based system for extracting information from text documents developed at IBM Almaden, that relies on probabilistic data management techniques. Choudhury et al. give a thorough survey of recent work done at the Intel Research Seattle lab on automatic human activity recognition from sensor data, and discuss desirable features for a database system that can support such applications. Garofalakis et al. discuss the novel probabilistic data management challenges faced by the *Data Furnace* project that aims to provide database support for pervasive computing (e.g., "smart home") environments. Finally, the article by Doan et al. surveys the authors' recent efforts on building a new data management platform that allows effective information sharing across web communities, arguing convincingly for probabilistic database tools.

We believe that you will find this diverse collection of research papers on probabilistic data management stimulating and thought-provoking, and hope that this special issue will spark even more interest in this exciting new area. Our sincerest thanks to all the authors for their contributions.

Minos Garofalakis, Dan Suciu
Intel Research Berkeley and University of Washington

An Introduction to ULDBs and the Trio System*

Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Jennifer Widom
Stanford University
{benjello,anish,hayworth,widom}@cs.stanford.edu

Abstract

We introduce ULDBs*: relational databases that add* uncertainty *and* lineage *of the data as first-class concepts. The ULDB model underlies the* Trio *system under development at Stanford. We describe the ULDB model, then present* TriQL*, our SQL-based query language for ULDBs. TriQL's semantics over ULDBs is defined both formally and operationally, and TriQL extends SQL with constructs for querying lineage and confidence values. We also briefly describe our initial prototype Trio implementation, which encodes ULDBs in conventional relations and automatically translates TriQL queries into SQL commands over the encoding. We conclude with research directions for ULDBs and the Trio system.*

1 Introduction

In the *Trio* project at Stanford, we are developing a new kind of database management system (DBMS): one in which *data*, *uncertainty* of the data, and data *lineage* are all first-class citizens in an extended relational model and SQL-based query language. In an earlier paper [Wid05], we motivated the need for these three aspects to coexist in one system and detailed numerous potential applications including scientific and sensor data management, data cleaning and integration, information extraction systems, and others. For examples in this paper we use a highly simplified "crime-solver" application with just two base tables: `Owns(owner,car)` and `Saw(witness,car)`, capturing (possibly uncertain) car ownership information and crime-vehicle sightings.

We call the type of relational database managed by Trio a *ULDB*, for *Uncertainty-Lineage Database*. To the best of our knowledge, ULDBs are the first database formalism to integrate uncertainty and lineage. In the rest of this section we briefly motivate the concepts and survey related work.

Uncertainty. Uncertainty is captured by tuples that may include several *alternative* possible values, with optional *confidence* values associated with each alternative. For example, if a witness saw a vehicle that was a Honda with confidence 0.5, a Toyota with confidence 0.3, or a Mazda with confidence 0.2, the sighting yields one tuple in table `Saw` with three alternative values. Furthermore, the presence of tuples may be uncertain, again with optionally specified confidence. For example, another witness may have 0.6 confidence that she saw a crime vehicle, but if she saw one it was definitely an Acura. Based on alternative tuple values and confidences, each ULDB represents multiple *possible instances* of a database.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This work was supported by the National Science Foundation under grants IIS-0324431 and IIS-1098447, and by a grant from the Boeing Corporation.

Lineage. Lineage, sometimes called *provenance*, associates with a data item information about its derivation. Broadly, lineage may be *internal*, referring to data within the ULDB, or *external*, referring to data outside the ULDB, or to other data-producing entities such as programs or devices. As a simple example of internal lineage, we may generate a table `Suspects` by joining tables `Saw` and `Owns` on attribute `car`. Lineage associated with a value in `Suspects` identifies the `Saw` and `Owns` data from which it was derived. A useful feature of internal lineage is that the confidence of a value in `Suspects` can be computed from the confidence of the data in its lineage.

As an example of external lineage, `Owns` may be populated from various car registration databases, and lineage can be used to connect the data to its original source. Many varieties of lineage are discussed in [Wid05]. In this paper we focus on simple internal lineage.

Queries. We will present a precise semantics for relational queries over ULDBs in Section 2, and in Section 3 an operational description of our SQL-based query language that conforms to the semantics. Intuitively, the result of a relational query Q on a ULDB U is a result R whose possible instances correspond to applying Q to each possible instance of U. Internal lineage connects the data in result R with the data from which it was derived, as in the `Suspects` join query introduced in the "Lineage" discussion above. Confidence values in query results are defined in a standard probabilistic fashion.

TriQL (pronounced "treacle"), Trio's query language, adapts SQL to our possible-instances semantics in a straightforward and natural manner. TriQL also includes simple but powerful constructs for querying lineage (e.g., "find all witnesses contributing to our suspicion of `Jimmy`"), querying uncertainty (e.g., "find all high-confidence sightings"), or querying both together (e.g., "find all suspects whose lineage contains low-confidence sightings or ownerships").

Prototype. In our initial prototype, Trio is built on top of a conventional relational DBMS, although as we delve more into storage, access method, and query optimization issues, we are likely to work more and more inside the system. Currently, we encode ULDBs as conventional relations, with a two-way translation layer so users can view and manipulate the ULDB model without being aware of the encoding. TriQL queries are translated automatically to SQL commands over the encoding, and the translation is not difficult or complex. Confidence and lineage are queried through simple built-in functions and predicates. Furthermore, lineage enables confidence values in query results to be computed *lazily*, a noteworthy feature of our approach. The prototype is described in more detail in Section 4.

1.1 Related Work

Our initial motivation to pursue this line of research, described in [Wid05], was followed by an exploration of the space of models for uncertainty with an emphasis on usability [DBHW06], and more recently a study of several new theoretical problems in this space [DNW05]. We subsequently added lineage to uncertainty, proposing and formalizing ULDBs in [BDHW05]. These papers all contain more extensive discussion of related work than we have room for here.

There has been a large body of previous working studying representation schemes and query answering for uncertain databases, including but certainly not limited to [AKG91, BGMP92, BP82, BP93, FR97, Gra84, Gra89, IL84, LLRS97, Var85]. A recent paper [BDM$^+$05] describes a system for handling imprecisions in data, and the same research group has made considerable progress in query answering for probabilistic databases [BDM$^+$05, DMS05, DS04, DS05].

Data lineage was introduced for a scientific data visualization system [WS97], and has been studied for conventional relational databases, e.g., [BKT01, BKT00, BKT02], and for data warehouses, e.g., [CW00, CW03, CWW00]. References [BCTV04, CTV05] describe a recent system being developed around data provenance (lineage).

2 ULDBs: Uncertainty-Lineage Databases

We present the ULDB model primarily through examples. A more formal treatment appears in [BDHW05]. ULDBs extend the standard SQL (multiset) relational model with: (1) *alternatives*, representing uncertainty about the contents of a tuple; (2) *maybe* (‘?’) annotations, representing uncertainty about the presence of a tuple; (3) numerical *confidence* values optionally attached to alternatives and ‘?’; and (4) *lineage*, connecting tuple alternatives to other alternatives from which they were derived. We next discuss each of these four constructs, then we define the semantics of relational queries on ULDBs.

2.1 Alternatives

ULDB relations are comprised of *x-tuples* (and therefore are called *x-relations*). Each x-tuple consists of one or more *alternatives*, where each alternative is a regular tuple over the schema of the relation. For example, if a witness Amy saw either a Honda, Toyota, or Mazda, then in table `Saw` we have:

(witness, car)
`(Amy,Honda)` ‖ `(Amy,Toyota)` ‖ `(Amy,Mazda)`

This x-tuple logically yields three *possible instances* for table `Saw`, one for each alternative. In general, the possible instances of an x-relation R correspond to all combinations of alternatives for the x-tuples in R. For example, if a second tuple in `Saw` had four alternatives, then there would be 12 possible instances altogether.

Clearly the x-tuple above can be represented more naturally with attribute-level instead of tuple-level uncertainty. Using set notation to denote “one of” we could write:

witness	**car**
`Amy`	{`Honda,Toyota,Mazda`}

As in [DBHW06, Wid05], we call this attribute-level construct an *or-set*. Or-sets can be a convenient compact representation for x-tuples, e.g., for readability or for space-efficient storage, and we plan to support them in Trio for these reasons. However or-sets are less expressive than x-tuples: If a tuple contains or-sets in multiple attributes, the alternatives of the x-tuple it represents are all possible combinations of the values in each of the or-sets, i.e., dependencies across attributes cannot be expressed using or-sets.

2.2 ‘?’ (Maybe) Annotations

Suppose a second witness, Betty, thinks she saw a car but is not sure. However, if she saw a car, it was definitely an Acura. In ULDBs, uncertainty about the existence of a tuple (more generally of an x-tuple) is denoted by a ‘?’ annotation on the x-tuple. Betty’s observation is thus added to table `Saw` as:

(witness, car)	
`(Amy,Honda)` ‖ `(Amy,Toyota)` ‖ `(Amy,Mazda)`	
`(Betty,Acura)`	**?**

The ‘?’ on the second x-tuple indicates that this entire tuple may or may not be present (so we call it a *maybe x-tuple*). Now the possible instances of an x-relation include not only all combinations of alternatives, but also all combinations of inclusion/exclusion for the maybe x-tuples. This `Saw` table has six possible instances: three choices for Amy’s car times two choices for whether or not Betty saw an Acura. For example, one possible instance of `Saw` is the two tuples `(Amy,Honda)`, `(Betty,Acura)`, while another instance is just `(Amy,Mazda)`.

2.3 Confidences

Numerical *confidence* values may be attached to the alternatives of an x-tuple. Suppose Amy's confidence in seeing a Honda, Toyota, or Mazda is 0.5, 0.3, and 0.2 respectively, and Betty's confidence in seeing a vehicle is 0.6. Then we have:

(witness, car)	
`(Amy,Honda):0.5` ‖ `(Amy,Toyota):0.3` ‖ `(Amy,Mazda):0.2`	
`(Betty,Acura):0.6`	**?**

In [BDHW05] we formalize an interpretation of these confidences in terms of probabilities. (Other interpretations may be imposed, but the probabilistic one is the current default for Trio.) Thus, if Σ is the sum of confidences for the alternatives of an x-tuple, then we must have $\Sigma \leq 1$, and if $\Sigma < 1$ then the x-tuple must have a '?'. Implicitly, '?' is given confidence $(1 - \Sigma)$ and denotes the probability that the tuple is not present.

Now each possible instance of an x-relation itself has a probability, defined as the product of the confidences of the alternatives and '?''s comprising the instance. It can be shown that for any x-relation: (1) The probabilities of all possible instances sum to 1; and (2) The confidence of an x-tuple alternative (respectively a '?') equals the sum of probabilities of the possible instances containing this alternative (respectively not containing any alternative from this x-tuple).

An important special case of ULDBs is when every x-tuple has only one alternative with a confidence value that may be < 1. This case corresponds to the traditional notion of probabilistic databases, as in, e.g., [BGMP92, CP87, DS04, LLRS97]. For simplicity in subsequent discussions we assume each ULDB includes confidences on all of its data or none of it, although "mixing and matching" is generally straightforward.

2.4 Lineage

Lineage in ULDBs is recorded at the granularity of tuple alternatives: lineage connects an x-tuple alternative to other x-tuple alternatives.[1] Specifically, we define lineage as a function λ over alternatives, such that $\lambda(t)$ gives the set of alternatives from which the alternative t was derived.

Consider again the join of `Saw` and `Owns` on attribute `car`, followed by a projection on `owner` to produce a table `Suspects(person)`. Let column ID contain a unique identifier for each x-tuple, and let (i, j) denote the jth alternative of the x-tuple with identifier i. Here is some sample data for all three tables, including lineage for the derived data in `Suspects`. For example, the lineage of Jimmy's presence in table `Suspects` is the second alternative of tuple 51 in table `Saw`, together with the second alternative of tuple 61 in table `Owns`.

ID	**Saw (witness, car)**
51	`(Cathy,Honda)` ‖ `(Cathy,Mazda)`

ID	**Owns (owner, car)**	
61	`(Jimmy,Toyota)` ‖ `(Jimmy,Mazda)`	**?**
62	`(Billy,Honda)`	
63	`(Hank,Honda)`	

ID	**Suspects (person)**		
71	`Jimmy`	**?**	λ (71,1) = { (51,2), (61,2) }
72	`Billy`	**?**	λ (72,1) = { (51,1), (62,1) }
73	`Hank`	**?**	λ (73,1) = { (51,1), (63,1) }

[1] Recall we are considering only internal lineage in this paper. We expect that many types of external lineage will also be recorded at the tuple-alternative granularity, although for some lineage types coarser granularity will be more appropriate [Wid05].

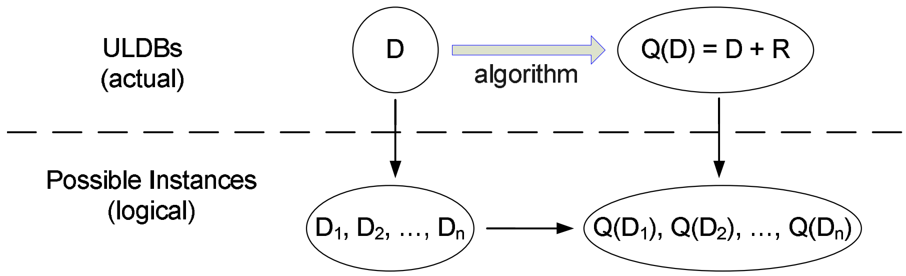

Figure 1: Relational queries on ULDBs.

An interesting and important effect of lineage is that it imposes restrictions on the possible instances of a ULDB. Consider the derived table `Suspects`. Even though there is a '?' on each of its three tuples, not all combinations are possible. If Billy is present in `Suspects` then alternative 1 must be chosen for tuple 51, and therefore Hank must be present as well. Jimmy is present in `Suspects` only if alternative 2 is chosen for tuple 51, in which case neither Billy nor Hank can be present. Note that choosing alternative $(51, 2)$ does not guarantee Jimmy's presence in `Suspects`, since tuple 61 has a '?'. Roughly speaking, a tuple alternative is present in a possible instance if and only if all of the alternatives in its lineage are present, although the actual constraints are somewhat more complex; see [BDHW05] for details.

In summary, once we add lineage, not all combinations of alternatives and '?' choices correspond to valid possible instances. The above ULDB has six possible instances, determined by the two choices for tuple 51 times the three choices (including '?') for tuple 61. Note that arbitrary lineage functions may not "work" under our model—consider for example a tuple whose lineage (directly or transitively) includes two different alternatives of the same x-tuple. In [BDHW05] we formally define *well-behaved* lineage and show that internal lineage generated by queries is always well-behaved. Under well-behaved lineage, the possible instances of an entire ULDB correspond to the possible instances of the "base" data (data with no lineage of its own), as seen in the example above. Now, with well-behaved lineage our interpretation of confidences carries over directly: combining confidences on the base data determines the probabilities of the possible instances, just as before. We will discuss confidences on derived data in Section 4.

2.5 Relational Queries

In this section we formally define the semantics of any relational query over a ULDB. Trio's SQL-based query language is presented in Section 3. The semantics for relational queries over ULDBs is quite straightforward but has two parts: (1) the possible-instances interpretation; and (2) lineage in query results.

Refer to Figure 1. Consider a ULDB D whose possible instances are $D_1, D_2, \ldots, D_n$, as shown on the left side of the figure. If we evaluate a query Q on D, the possible instances in the result of Q should be $Q(D_1), Q(D_2), \ldots, Q(D_n)$, as shown in the lower-right corner. For example, if a query Q joins x-relations `Saw` and `Owns`, then logically it should join all of the possible instances of these two x-relations. Of course we will not actually generate all possible instances and operate on them, so a query processing algorithm follows the top arrow in Figure 1, producing a ULDB query result $Q(D)$ that represents the possible instances.

In our model, a query result is more than just a set of x-tuples: $Q(D)$ contains the original x-relations of D, together with a new *result x-relation* R. Lineage from R into the x-relations of D reflects the derivation of the data in R. This approach is necessary for $Q(D)$ to represent the correct possible instances in the query result,[2] and to enable consistent further querying of the original and new x-relations. Our example in the previous subsection with `Suspects` as the result of query $Q = \pi_{\texttt{owner}}(\texttt{Saw} \bowtie \texttt{Owns})$ demonstrates the possible-instances

[2]Technically, the possible instances in the lower half of Figure 1 also contain lineage, but this aspect is not critical here; see [BDHW05] for formal details.

interpretation and lineage from query result to original data; details are left as a useful exercise for the reader.

The lineage of tuples in query results can be defined in different ways, as explored in other work [BKT01]. So far, we have considered a class of queries (subsuming SPJ queries and union, formally defined in [BDHW05]) for which the lineage is straightforward: every result tuple alternative t identifies a unique combination of base tuple alternatives from which t was derived.

3 TriQL: The Trio Query Language

In this section we describe *TriQL*, Trio's SQL-based query language. Except for built-in functions and predicates for querying confidence values and lineage, TriQL uses the same syntax as SQL. However, the interpretation of SQL queries must be modified to reflect the semantics over ULDBs discussed in the previous section.

In this paper (and in our current implementation), we limit ourselves to single-block queries without aggregation or `DISTINCT`. Thus, the general form of a TriQL query Q is:

```
SELECT attr-list [ INTO new-table ]
FROM X1, X2, ..., Xn
WHERE predicate
```

As an example, our join query producing `Suspects` is written in TriQL exactly as expected:

```
SELECT Owns.owner as person INTO Suspects
FROM Saw, Owns
WHERE Saw.car = Owns.car
```

If we execute this query as regular SQL over each of the possible instances of `Saw` and `Owns`, as in the lower portion of Figure 1, it produces the expected set of possible instances in its result. More importantly, following the operational semantics of TriQL given next, this query produces a result table `Suspects`, including lineage to tables `Saw` and `Owns`, that correctly represents those possible instances.

3.1 Operational Semantics

We provide an operational description of TriQL by specifying direct evaluation of an arbitrary TriQL query over a ULDB, corresponding to the upper arrow in Figure 1. Consider the generic TriQL query block above. Let *schema*(Q) denote the composition *schema*(`X1`) $\uplus$ *schema*(`X2`) $\uplus \cdots \uplus$ *schema*(`Xn`) of the `FROM` relation schemas, just as in SQL query processing. The `predicate` is evaluated over tuples in *schema*(Q), and the `attr-list` is a subset of *schema*(Q) or the symbol "$\star$", again just as in SQL.

The steps below are an operational description of evaluating the above query block. As in SQL database systems, a query processor would rarely execute the simplest operational description since it could be woefully inefficient, but any query plan or execution technique (such as our translation-based approach described in Section 4) must produce the same result as this description.

1. Consider every combination $x_1, x_2, \ldots, x_n$ of x-tuples in `X1`, `X2`, ..., `Xn`, one combination at a time, just as in SQL.

2. Form a "super-tuple" T whose alternatives have schema *schema*(Q). T has one alternative for each combination of alternatives in $x_1, x_2, ..., x_n$.

3. If any of $x_1, x_2, \ldots, x_n$ has a '?', add a '?' to T.

4. Set the lineage of each alternative in T to be the alternatives in $x_1, x_2, \ldots, x_n$ from which it was constructed.

5. Retain from T only those alternatives satisfying the `predicate`. If no alternatives satisfy the predicate, we're finished with T. If any alternative does not satisfy the predicate, add a '?' to T if it is not there already.

6. If we are operating on a ULDB with confidences, either compute the confidence values for T's remaining alternatives and store them (*eager confidence computation*), or set the confidence values to `NULL` (*lazy confidence computation*). See Section 4 for further discussion.

7. Project each alternative of T onto the attributes in `attr-list`, generating an x-tuple in the query result. If there is an `INTO` clause, insert T into table `new-table`.

We leave it to the reader to verify that this operational semantics produces the `Suspects` result table shown with example data in Section 2.4, and more generally that it conforms to the formal semantics given in Section 2.

3.2 Querying Confidences

TriQL provides a built-in function `conf()` for accessing confidence values. Suppose we want our `Suspects` query to only use sightings having confidence > 0.5 and ownerships having confidence > 0.8. We write:

```
SELECT Owns.owner as person INTO Suspects
FROM Saw, Owns
WHERE Saw.car = Owns.car AND conf(Saw) > 0.5 AND conf(Owns) > 0.8
```

In the operational semantics, when we evaluate the `predicate` over the alternatives in T in step 6, `conf(Xi)` refers to the confidence associated with the x_i component of the alternative being evaluated. Note that this function may trigger confidence computations in the lazy case.

3.3 Querying Lineage

For querying lineage, TriQL introduces a built-in predicate designed to be used as a join condition. If we include predicate `lineage`(R, S) in the `WHERE` clause of a TriQL query with x-relations R and S in its `FROM` clause, then we are constraining the joined R and S tuple alternatives to be connected by lineage. For example, suppose we want to find all witnesses contributing to Hank being a suspect. We can write:

```
SELECT Saw.witness INTO AccusesHank
FROM Suspects, Saw
WHERE lineage(Suspects,Saw) AND Suspects.person = 'Hank'
```

In the `WHERE` clause, `lineage(Suspects,Saw)` evaluates to true for any pair of alternatives x_1 and x_2 from `Suspects` and `Saw` such that x_1's lineage includes x_2. Of course we could write this query directly on the base relations if we remembered how `Suspects` was computed, but the `lineage()` predicate provides a more general construct that is insensitive to query history.

With the addition of derived x-relation `AccusesHank`, we now have two layers of lineage: from `AccusesHank` to `Suspects`, then from `Suspects` to the base x-relations. In general, over time a ULDB may develop many levels of lineage, so we also have a *transitive lineage* predicate: `lineage*`(R, S) evaluates to true for any pair of alternatives x_1 and x_2 from R and S such that there is a "path" via lineage from x_1 to x_2. This predicate introduces some interesting evaluation issues. Our current prototype keeps track of lineage structure and uses it to translate `lineage*()` into joins with `lineage()` predicates.

As a final TriQL example incorporating both lineage and confidence, the following query finds persons who are suspected based on high-confidence ownership of a Honda:

```
SELECT Owns.owner INTO HighHonda
FROM Suspects, Owns
WHERE lineage(Suspects,Owns) AND Owns.car = 'Honda' AND conf(Owns) > 0.8
```

4 Current Implementation

Our first-generation Trio prototype, which is up and running, is built on top of a conventional relational DBMS: ULDBs are represented in relational tables, and TriQL queries and commands are rewritten automatically into SQL commands evaluated against the representation.

The core system is implemented in Python and presents a simple API that extends the standard Python DB 2.0 API for database access (Python's analog of JDBC). The Trio API supports TriQL queries instead of SQL, query results are cursors enumerating x-tuple objects instead of regular tuples, and x-tuple objects provide programmatic access to their alternatives, including confidences and lineage. The first application we built using the Trio API is a generic command-line interactive client, similar to that provided by most DBMS's.

We are currently building on the *Postgres* open-source DBMS. However, we intentionally rely on very few Postgres-specific features, so porting to any other DBMS providing a DB 2.0 API should be straightforward. The next two sections describe our representation of ULDBs as conventional relations, and our rewrite techniques for query processing.

4.1 Representing ULDBs as Relations

Let $R(A_1, \ldots, A_n)$ be an x-relation with lineage and possibly confidences. We store the data portion of R as a conventional table (which we also call R) with three extra attributes: R(`aid`, `xid`, `conf`, $A_1, \ldots, A_n$). Each alternative in the original x-relation is stored as its own tuple in R, with a unique identifier `aid`. The alternatives of each x-tuple have the same x-tuple identifier, `xid`. Finally, `conf` stores the confidence of the alternative, with three special values: `NULL` denotes that the alternative's confidence has not yet been computed (but can be computed from the alternative's lineage); -1 denotes that the alternative does not have a confidence value but its x-tuple has a '?'; -2 denotes that the alternative does not have a confidence value and its tuple does not have a '?'. If an alternative has `conf` $= -1$ then so must all other alternatives with the same `xid`; similarly for `conf` $= -2$.

The lineage information for each x-relation R is stored in a separate table *lin:R*(`aid1`, `table`, `aid2`), whose tuples denote that R's alternative `aid1` contains alternative `aid2` from table `table` in its lineage. Here is an example of an encoded ULDB corresponding to a subset of the data from Section 2.4, with confidence values added to `Saw` for illustrative purposes only:

Saw

aid	xid	conf	witness	car
11	51	0.8	Cathy	Honda
12	51	0.2	Cathy	Mazda

Owns

aid	xid	conf	owner	car
21	61	-1	Jimmy	Toyota
22	61	-1	Jimmy	Mazda
23	62	-2	Billy	Honda

Suspects

aid	xid	conf	person
31	71	-1	Jimmy
32	72	-2	Billy

lin:Suspects

aid1	table	aid2
31	Saw	12
31	Owns	22
32	Saw	11
32	Owns	23

The Trio implementation does not require all relations to have uncertainty or lineage—conventional relations can coexist with x-relations, and they can be queried together. Currently, a single Trio metadata catalog keeps track of which relations include uncertainty, and records the general structure of the lineage present in the database, i.e., which x-relations include lineage to which other x-relations.

4.2 Query Processing

The Trio system evaluates TriQL queries by translating them into SQL operations over the representation described in the previous section. Consider a TriQL query Q whose result is to be stored in a new x-relation R. Q is rewritten into a query Q_r executed against the actual database. Q_r's result is post-processed in order to group alternatives into the x-tuples represented in R, and to generate the lineage table *lin:R*. Confidence values for result tuples are computed separately, possibly not until they are requested by the user or application. We detail each of these phases next, then describe how Trio produces *transient* results for queries without an `INTO` clause.

Query Rewriting. Consider one last time a variant on our favorite join query producing `Suspects`:

```
SELECT Owns.owner as person INTO Suspects
FROM Saw, Owns
WHERE Saw.car = Owns.car AND conf(Saw) > 0.5
```

This query is translated to the following SQL query Q_r, shown with a sample query result:

```
CREATE TABLE Suspects AS
   SELECT newOid() as aid, NULL as xid, NULL as conf,
          owner as person,
          Saw.aid as Saw_aid, Saw.xid as Saw_xid,
          Owns.aid as Owns_aid, Owns.xid as Owns_xid
    FROM Saw, Owns WHERE Saw.car = Owns.car AND conf(Saw) > 0.5
```

aid	xid	conf	person	Saw_aid	Saw_xid	Owns_aid	Owns_xid
32	NULL	NULL	Billy	11	51	23	62

The translation of queries with `lineage()` predicates requires joining in the corresponding *lin* tables in the expected way; details are omitted.

Post-processing. Next, the Trio engine processes the result table obtained from the translated query Q_r as follows. Each step is performed by simple, automatically-generated SQL commands.

1. New xid's are generated for the `xid` column, to group the alternatives in the result into x-tuples. Recalling the operational semantics from Section 3.1, we can use a simple `GROUP BY` query on the xid's of the original tuples (`Saw_xid` and `Owns_xid` in our example) and a sequence to generate the new xid's.

2. The lineage information—`aid` values with table names—is moved to a separate newly-created table (`lin:Suspects` in our example).

3. The data needed for steps 1 and 2 (attributes `Saw_aid`, `Saw_xid`, `Owns_aid`, and `Owns_xid` in our example) are removed from the query result table by dropping the corresponding columns.

4. Metadata information is added to the Trio catalog about the new table `Suspects`, and the fact that it was derived from `Saw` and `Owns`.

Computing Confidences. Lastly, we discuss confidences on query results: their definition and their computation. For the class of queries we have considered so far, a very simple definition is enabled by lineage: The confidence of a tuple alternative t in a query result is the product of the confidences of the tuple alternatives in $\lambda(t)$. For example, if `Hank` in table `Suspects` was derived from a 0.8 confidence `Saw` alternative together with a 0.5 confidence `Owns` tuple, then `Hank` has confidence 0.4. This definition is consistent with the standard interpretation of probabilistic databases.

The fact that confidences can be computed via lineage enables us to compute and store confidence values *lazily* if we wish to do so—if an application typically does not use confidence values, or uses them very selectively. Of course we can also compute confidences *eagerly* during query execution; both options were presented in our operational semantics (Section 3.1). Many optimizations are possible for computing confidences, and new challenges arise when we extend the class of queries considered. This area constitutes an important avenue of current work, as mentioned in the next section.

Transient Query Results. So far in this section we have assumed query results are to be stored in a newly-created table. Trio also supports conventional (transient) query results, when the `INTO` clause is omitted: Translated query Q_r now sorts its result by the xid's of the original x-tuples, and is executed via a cursor instead of writing its result into a table. Trio returns a cursor for Q to its client. As the client iterates through the Q cursor, x-tuple objects (with lineage) are generated on-the-fly by iterating through the cursor for the sorted Q_r result.

5 Current and Future Directions

- **Theory:** There is a great deal of theoretical work to do on ULDBs. Effectively, nearly every topic considered in relational database theory can be reconsidered in ULDBs. Examples include (but are certainly not limited to) dependency theory and database design, query containment and rewritings, and sampling and statistics. We are attempting to address the most interesting and relevant of these problems.

- **Confidences:** We are currently exploring various algorithms and optimizations when computing confidences lazily, such as memoization and pruning the lineage traversal. We are also studying the fundamental tradeoffs between lazy and eager computation of confidences.

- **Updates:** The formal query semantics and TriQL language presented here are query-only. We are in the process of considering updates: identifying and formally defining an appropriate set of primitive update operations for ULDBs, and exposing them in SQL-like data modification commands.

- **Queries:** Our current Trio prototype supports only single-block "SPJ" queries. While some additional query constructs simply require additional programming, other constructs—such as duplicate elimination, aggregation, and negated subqueries—require more complex definitions and implementation of lineage. We also plan to add richer constructs for querying uncertainty, including "top-K" style queries based on confidence, and "horizontal" queries that aggregate or perform subqueries across tuple alternatives.

- **Storage, Auxiliary Structures, and Query Optimization:** The ULDB model introduces new options and tradeoffs in data layout, along with new possibilities for indexing, partitioning, and materialized views, and new challenges for query operators and optimizations. To fully explore many of these topics we will need to move our prototype away from its translation-based approach and begin custom implementation inside a DBMS.

- **Long-Term Goals:** There are several important features in the original Trio proposal [Wid05] that we have barely touched upon but remain on our agenda: Continuous uncertain values such as intervals and

Gaussians, incomplete relations, versioning, and various types of external lineage. In addition, the introduction of uncertainty and lineage introduces new challenges in both human and programmatic interfaces.

- **Applications:** Needless to say, a most important long-term goal is to deploy a wide variety of applications on the Trio system to test its functionality and scalability. We also may develop specialized versions of Trio with restricted and/or extended functionality suited to specific applications, such as for sensor data management or data integration.

Acknowledgments

We thank Parag Agrawal, Alon Halevy, Shubha Nabar, and the entire Trio group for many useful discussions.

References

[AKG91] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. In *Theoretical Computer Science*, pages 34–48, 1991.

[BCTV04] D. Bhagwat, L. Chiticariu, W.C. Tan, and G. Vijayvargiya. An annotation management system for relational databases. In *Proc. of Intl. Conference on Very Large Data Bases (VLDB)*, 2004.

[BDHW05] O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. The symbiosis of lineage and uncertainty. Technical report, Stanford InfoLab, December 2005. Available at http://dbpubs.stanford.edu/pub/2005-39.

[BDM+05] J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *Proc. of ACM SIGMOD Intl. Conference on Management of Data*, 2005.

[BGMP92] D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Transactions on Knowledge and Data Engineering*, 4(5):487–502, October 1992.

[BKT00] P. Buneman, S. Khanna, and W. Tan. Data provenance: Some basic issues. In *Proc. of Intl. Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 87–93, 2000.

[BKT01] P. Buneman, S. Khanna, and W.C. Tan. Why and where: A characterization of data provenance. In *Proc. of Intl. Conference on Database Theory (ICDT)*, 2001.

[BKT02] P. Buneman, S. Khanna, and W. Tan. On propagation of deletions and annotations through views. In *Proc. of ACM Intl. Conference on Principles of Database Systems (PODS)*, 2002.

[BP82] B. Buckles and F. Petry. A fuzzy model for relational databases. *International Journal of Fuzzy Sets and Systems*, 7:213–226, 1982.

[BP93] R.S. Barga and C. Pu. Accessing imprecise data: An approach based on intervals. *IEEE Data Engineering Bulletin*, 16(2):12–15, June 1993.

[CP87] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *Proc. of Intl. Conference on Very Large Data Bases (VLDB)*, 1987.

[CTV05] L. Chiticariu, W. Tan, and G. Vijayvargiya. DBNotes: a post-it system for relational databases based on provenance. *Proc. of ACM SIGMOD Intl. Conference on Management of Data*, 2005.

[CW00] Y. Cui and J. Widom. Practical lineage tracing in data warehouses. In *Proc. of Intl. Conference on Data Engineering (ICDE)*, 2000.

[CW03] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. *VLDB Journal*, 12(1):41–58, May 2003.

[CWW00] Y. Cui, J. Widom, and J. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Transactions on Database Systems*, 25(2):179–227, June 2000.

[DBHW06] A. Das Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *Proc. of Intl. Conference on Data Engineering (ICDE)*, April 2006.

[DMS05] N. Dalvi, G. Miklau, and D. Suciu. Asymptotic conditional probabilities for conjunctive queries. In *Proc. of Intl. Conference on Database Theory (ICDT)*, 2005.

[DNW05] A. Das Sarma, S. U. Nabar, and J. Widom. Representing uncertain data: Uniqueness, equivalence, minimization, and approximation. Technical report, Stanford InfoLab, 2005. Available at http://dbpubs.stanford.edu/pub/2005-38.

[DS04] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Proc. of Intl. Conference on Very Large Data Bases (VLDB)*, 2004.

[DS05] N. Dalvi and D. Suciu. Answering queries from statistics and probabilistic views. In *Proc. of Intl. Conference on Very Large Data Bases (VLDB)*, 2005.

[FR97] N. Fuhr and T. Rölleke. A probabilistic NF2 relational algebra for imprecision in databases. *Unpublished Manuscript*, 1997.

[Gra84] G. Grahne. Dependency satisfaction in databases with incomplete information. In *Proc. of Intl. Conference on Very Large Data Bases (VLDB)*, 1984.

[Gra89] G. Grahne. Horn tables - an efficient tool for handling incomplete information in databases. In *Proc. of ACM Intl. Conference on Principles of Database Systems (PODS)*, 1989.

[IL84] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, October 1984.

[LLRS97] L.V.S. Lakshmanan, N. Leone, R. Ross, and V.S. Subrahmanian. ProbView: A flexible probabilistic database system. *ACM Transactions on Database Systems*, 22(3):419–469, September 1997.

[Var85] M. Vardi. Querying logical databases. In *Proc. of ACM Intl. Conference on Principles of Database Systems (PODS)*, 1985.

[Wid05] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proc. of Conference on Innovative Data Systems Research (CIDR)*, 2005.

[WS97] A. Woodruff and M. Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *Proc. of Intl. Conference on Data Engineering (ICDE)*, 1997.

Models for Incomplete and Probabilistic Information

Todd J. Green
University of Pennsylvania
tjgreen@cis.upenn.edu

Val Tannen
University of Pennsylvania
val@cis.upenn.edu

1 Introduction

This is an abbreviated version of [13] where proofs and additional results are discussed (available also from `http://db.cis.upenn.edu`).

The representation of incomplete information in databases has been an important research topic for a long time, see the references in [12], in Ch.19 of [2], in [21], as well as the recent [22, 20, 19]. Moreover, this work is closely related to recently active research topics such as inconsistent databases and repairs [3], answering queries using views [1], and data exchange [9]. The classic reference on incomplete databases remains [14] with the fundamental concept of c-table and its restrictions to simpler tables with variables. The most important result of [14] is the query answering algorithm that defines an algebra on c-tables that corresponds exactly to the usual relational algebra ($\mathcal{RA}$). A recent paper [19] has defined a hierarchy of incomplete database models based on finite sets of choices and optional inclusion. One of our contributions consists of **comparisons** between the models [19] and the tables with variables from [14].

Two criteria have been provided for comparisons among all these models: [14, 19] discuss *closure* under relational algebra operations, while [19] also emphasizes *completeness*, specifically the ability to represent all finite incomplete databases. We point out that the latter is not appropriate for tables with variables over an infinite domain, and we contribute another criterion, $\mathcal{RA}$**-completeness**, that fully characterizes the expressive power of c-tables.

We also introduce a new idea for the study of models that are not complete. Namely, we consider combining existing models with queries in various fragments of relational algebra. We then ask how big these fragments need to be to obtain a combined model that is complete. We give a number of such **algebraic completion** results.

Early on, probabilistic models of databases were studied less intensively than incompleteness models, with some notable exceptions [5, 4, 18, 7]. Essential progress was made independently in three papers [10, 16, 23] that were published at about the same time. [10, 23] assume a model in which tuples are taken independently in a relation with given probabilities. [16] assumes a model with a separate distribution for each attribute in each tuple. All three papers attacked the problem of calculating the probability of tuples occurring in query answers. They solved the problem by developing more general models in which rows contain additional information ("event expressions","paths","traces"), and they noted the similarity with the conditions in c-tables.

We go beyond the problem of individual tuples in query answers by defining **closure** under a query language for probabilistic models. Then we develop a new model, **probabilistic c-tables** that adds *to the c-tables themselves* probability distributions for the values taken by their variables. Here is an example of such a representation that captures the set of instances in which Alice is taking a course that is Math with probability 0.3;

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Physics (0.3); or Chemistry (0.4), while Bob takes the same course as Alice, provided that course is Physics or Chemistry and Theo takes Math with probability 0.85:

Student	*Course*	*Condition*
Alice	x	
Bob	x	$x =$ phys $\vee$ $x =$ chem
Theo	math	$t = 1$

$$x = \begin{cases} \text{math} & : 0.3 \\ \text{phys} & : 0.3 \\ \text{chem} & : 0.4 \end{cases} \qquad t = \begin{cases} 0 : & 0.15 \\ 1 : & 0.85 \end{cases}$$

The concept of probabilistic c-table allows us to solve the closure problem by using the same algebra on c-tables defined in [14].

We also give a **completeness** result by showing that probabilistic boolean c-tables (all variables are two-valued and can appear only in the conditions, not in the tuples) can represent *any* probabilistic database.

An important conceptual contribution is that we show that, at least for the models we consider, the probabilistic database models can be seen, as **probabilistic counterparts** of incomplete database models. In an incompleteness model a tuple or an attribute value in a tuple may or may not be in the database. In its probabilistic counterpart, these are seen as elementary events with an assigned probability. For example, the models used in [10, 16, 23] are probabilistic counterparts of the two simplest incompleteness models discussed in [19]. As another example, the model used in [7] can be seen as the probabilistic counterpart of an incompleteness model one in which tuples sharing the same key have an exclusive-or relationship.

A consequence of this observation is that, in particular, query answering for probabilistic c-tables will allow us to solve the problem of calculating probabilities about query answers for any model that can be defined as a probabilistic counterpart of the incompleteness models considered in [14, 19].

2 Incomplete Information and Representation Systems

Our starting point is suggested by the work surveyed in [12], in Ch. 19 of [2], and in [21]. A database that provides incomplete information consists of a *set of possible instances*. At one end of this spectrum we have the conventional single instances, which provide "complete information." At the other end we have the set of *all* allowable instances which provides "no information" at all, or "zero information."

We adopt the formalism of relational databases over a fixed countably infinite domain $\mathbb{D}$. We use the unnamed form of the relational algebra. To simplify the notation we will work with relational schemas that consist of a single relation name of arity n. Everything we say can be easily reformulated for arbitrary relational schemas. We shall need a notation for the set of *all* (conventional) instances of this schema, i.e., all the finite n-ary relations over $\mathbb{D}$ namely $\mathcal{N} := \{I \mid I \subseteq \mathbb{D}^n,\ I \text{ finite}\}$

Definition 1: An **incomplete(-information) database** (**i-database** for short), $\mathcal{I}$, is a set of conventional instances, i.e., a subset $\mathcal{I} \subseteq \mathcal{N}$.

The usual relational databases correspond to the cases when $\mathcal{I} = \{I\}$. The **no-information** or **zero-information database** consists of *all* the relations: $\mathcal{N}$.

Conventional relational instances are finite. However, because $\mathbb{D}$ is infinite incomplete databases are in general infinite. Hence the interest in finite, syntactical, representations for incomplete information.

Definition 2: A **representation system** consists of a set (usually a syntactically defined "language") whose elements we call tables, and a function *Mod* that associates to each table T an incomplete database $\mathit{Mod}(T)$.

The classical reference [14] considers three representation systems: **Codd tables**, v**-tables**, and c**-tables**. v-tables are conventional instances in which variables can appear in addition to constants from $\mathbb{D}$. If T is a v-table then

$$\mathit{Mod}(T) := \{\nu(T) \mid \nu : \mathit{Var}(T) \to \mathbb{D} \text{ is a valuation for the variables of } T\}$$

Codd tables are v-tables in which all the variables are distinct. They correspond roughly to the current use of nulls in SQL, while v-tables model "labeled" or "marked" nulls. c-tables are v-tables in which each tuple is associated with a condition — a boolean combination of equalities involving variables and constants. We typically use the letter φ for conditions. The tuple condition is tested for each valuation ν and the tuple is discarded from $\nu(T)$ if the condition is not satisfied.

Example 1: Here is an example of a c-table.

$$S := \begin{array}{|ccc|c|} \hline 1 & 2 & x & \\ 3 & x & y & x = y \wedge z \neq 2 \\ z & 4 & 5 & x \neq 1 \vee x \neq y \\ \hline \end{array} \quad \mathit{Mod}(S) = \left\{ \begin{array}{|ccc|} \hline 1 & 2 & 1 \\ 3 & 1 & 1 \\ \hline \end{array}, \begin{array}{|ccc|} \hline 1 & 2 & 2 \\ 1 & 4 & 5 \\ \hline \end{array}, \ldots, \begin{array}{|ccc|} \hline 1 & 2 & 77 \\ 97 & 4 & 5 \\ \hline \end{array}, \ldots \right\}$$

Several other representation systems have been proposed in a recent paper [19]. We illustrate here three of them and we discuss several others later. A **?-table** is a conventional instance in which tuples are optionally labeled with "?," meaning that the tuple may be missing. An **or-set-table** looks like a conventional instance but or-set values [15, 17] are allowed. An or-set value $\langle 1, 2, 3\rangle$ signifies that exactly one of $1, 2$, or 3 is the "actual" (but unknown) value. Clearly, the two ideas can be combined yielding another representation systems that we might (awkwardly) call **or-set-?-tables**.(In [19] these three systems are denoted by $\mathcal{R}_?$, $\mathcal{R}^A$ and $\mathcal{R}^A_?$.)

Example 2: Here is an example of an or-set-?-table.

$$T := \begin{array}{|ccc|c|} \hline 1 & 2 & \langle 1,2\rangle & \\ 3 & \langle 1,2\rangle & \langle 3,4\rangle & \\ \langle 4,5\rangle & 4 & 5 & ? \\ \hline \end{array} \quad \mathit{Mod}(T) = \left\{ \begin{array}{|ccc|} \hline 1 & 2 & 1 \\ 3 & 1 & 3 \\ 4 & 4 & 5 \\ \hline \end{array}, \begin{array}{|ccc|} \hline 1 & 2 & 1 \\ 3 & 1 & 3 \\ \hline \end{array}, \begin{array}{|ccc|} \hline 1 & 2 & 2 \\ 3 & 1 & 3 \\ 4 & 4 & 5 \\ \hline \end{array}, \ldots, \begin{array}{|ccc|} \hline 1 & 2 & 2 \\ 3 & 2 & 4 \\ \hline \end{array} \right\}$$

3 Completeness and Closure

"Completeness" of expressive power is the first obvious question to ask about representation systems. This brings up a fundamental difference between the representation systems of [14] and those of [19]. The presence of variables in a table T and the fact that $\mathbb{D}$ is infinite means that *Mod*(T) may be infinite. For the tables considered in [19], *Mod*(T) is always finite.

[19] defines completeness as the ability of a representation system to represent "all" possible i-databases. For the kind of tables considered in [19] the question makes sense. But in the case of the tables with variables in [14] this is hopeless for trivial reasons. Indeed, in such systems there are only countably many tables while there are uncountably many i-databases (the subsets of $\mathcal{N}$, which is infinite). We will discuss separately below *finite completeness* for systems that only represent finite database. Meanwhile, we will develop a different yardstick for the expressive power of tables with variables that range over an infinite domain.

c-tables and their restrictions (v-tables and Codd tables) have an inherent limitation: the cardinality of the instances in *Mod*(T) is at most the cardinality of T. For example, the zero-information database $\mathcal{N}$ *cannot* be represented with c-tables. It also follows that among the i-databases that are representable by c-tables the "minimal"-information ones are those consisting for some m of all instances of cardinality up to m (which are in fact representable by Codd tables with m rows). Among these, we make special use of the ones of cardinality 1:

$$\mathcal{Z}_k := \{\{t\} \mid t \in \mathbb{D}^k\}.$$

Hence, $\mathcal{Z}_k$ consists of *all* the one-tuple relations of arity k. Note that $\mathcal{Z}_k = \mathit{Mod}(Z_k)$ where Z_k is the Codd table consisting of a single row of k distinct variables.

Definition 3: An i-database $\mathcal{I}$ is $\mathcal{RA}$**-definable** if there exists a relational algebra query q such that $\mathcal{I} = q(\mathcal{Z}_k)$, where k is the arity of the input relation name in q.

Theorem 4: If $\mathcal{I}$ is an i-database representable by a c-table T, i.e., $\mathcal{I} = Mod(T)$, then $\mathcal{I}$ is $\mathcal{RA}$-definable.

Hence, c-tables are in some sense "no more powerful" than the relational algebra. But are they "as powerful"? This justifies the following:

Definition 5: A representation system is **$\mathcal{RA}$-complete** if it can represent any $\mathcal{RA}$-definable i-database.

Since Z_k is itself a c-table the following is an immediate corollary of the fundamental result of [14] (see Theorem 11 below). It also states that the converse of Theorem 4 holds.

Theorem 6: c-tables are $\mathcal{RA}$-complete.

We now turn to the kind of completeness considered in [19].

Definition 7: A representation system is **finitely complete** if it can represent any finite i-database.

The finite incompleteness of ?-tables, or-set-tables, or-set-?-tables and other systems is discussed in [19] where a finitely complete representation system called $\mathcal{R}^A_{\text{prop}}$ is also given (we do not discuss $\mathcal{R}^A_{\text{prop}}$further here). Is finite completeness a reasonable question for c-tables, v-tables, and Codd tables? In general, for such tables $Mod(T)$ is infinite (all that is needed is a tuple with at least one variable and with an infinitely satisfiable condition). To facilitate comparison with the systems in [19] we define *finite-domain* versions of tables with variables.

Definition 8: A **finite-domain** c-table (v-table, Codd table) consists of a c-table (v-table, Codd table) T together with a *finite* $\text{dom}(x) \subset \mathbb{D}$ for each variable x that occurs in T.

Note that finite-domain Codd tables are equivalent to or-set tables. Indeed, to obtain an or-set table from a Codd table, one can see $\text{dom}(x)$ as an or-set and substitute it for x in the table. Conversely, to obtain a Codd table from an or-set table, one can substitute a fresh variable x for each or-set and define $\text{dom}(x)$ as the contents of the or-set.

It is easy to see that finite-domain c-tables are finitely complete. In fact, this is true even for the fragment of finite-domain c-tables which we will call *boolean c-tables*, where the variables take only boolean values and are only allowed to appear in conditions (never as attribute values).

Theorem 9: Boolean c-tables are finitely complete (hence finite-domain c-tables are also finitely complete).

If we additionally restrict boolean c-tables to allow conditions to contain only true or a single variable which appears in no other condition, then we obtain a representation system which is equivalent to ?-tables.

Definition 10: A representation system is **closed** under a query language if for any query q and any table T there is a table T' that represents $q(Mod(T))$.

This definition is from [19]. In [2], a *strong* representation system is defined in the same way, with the significant addition that T' should be *computable* from T and q. It is not hard to show, using general recursion-theoretic principles, that there exist representation systems (even ones that only represent finite i-databases) which are closed as above but not strong in the sense of [2]. However, the concrete systems studied so far are either not closed, or if they are closed, as in the theorem below, then the proof provides also the algorithm required by the definition of strong systems. Hence, we see no need to insist upon the distinction.

Theorem 11 ([14]): c-tables are closed under the relational algebra. (The same proof works for finite-domain c-tables, and even boolean c-tables.)

4 Algebraic Completion

None of the incomplete representation systems we have seen so far is closed under the full relational algebra.

Proposition 12 ([14, 19]): Codd tables and v-tables are not closed under e.g. selection. Or-set tables and finite v-tables are also not closed under e.g. selection. ?-tables are not closed under e.g. join.

We have seen that "closing" minimal-information one-row Codd tables (see before Definition 5) $\{Z_1, Z_2, \ldots\}$, by relational algebra queries yields equivalence with the c-tables. In this spirit, we will investigate "how much" of the relational algebra would be needed to complete the other representation systems considered. We call this kind of result *algebraic completion*.

Definition 13: If $(\mathcal{T}, \mathit{Mod})$ is a representation system and $\mathcal{L}$ is a query language, then the representation system *obtained by closing $\mathcal{T}$ under $\mathcal{L}$* is the set of tables $\{(T, q) \mid T \in \mathcal{T}, q \in \mathcal{L}\}$ with the function $\mathit{Mod} : \mathcal{T} \times \mathcal{L} \rightarrow \mathcal{N}$ defined by $\mathit{Mod}(T, q) := q(\mathit{Mod}(T))$.

Theorem 14 ($\mathcal{RA}$-Completion): Closing Codd tables under $SPJU$ queries and closing v-tables under SP queries produces $\mathcal{RA}$-complete systems in both cases.

We give now a set of analogous completion results for the finite case.

Theorem 15 (Finite-Completion): Closing or-set-tables under PJ queries, closing finite v-tables under PJ or S^+P queries, and closing ?-tables under $\mathcal{RA}$ queries produces finitely complete systems.

5 Probabilistic Databases and Representation Systems

Finiteness assumption For the entire discussion of probabilistic database models we will assume that *the domain of values $\mathbb{D}$ is finite.* Infinite domains of values are certainly interesting in practice; for some examples see [16, 22, 19]. Moreover, in the case of incomplete databases we have seen that they allow for interesting distinctions.[1] However, finite probability spaces are much simpler than infinite ones and we will take advantage of this simplicity. We leave for future investigations the issues related to probabilistic databases over infinite domains.

We wish to model probabilistic information using a probability space whose possible outcomes are all the conventional instances. Recall that for simplicity we assume a schema consisting of just one relation of arity n. The finiteness of $\mathbb{D}$ implies that there are only finitely many instances, $I \subseteq \mathbb{D}^n$.

By **finite probability space** we mean a probability space (see e.g. [8]) $(\Omega, \mathcal{F}, \mathbb{P}[\ ])$ in which the set of outcomes Ω is *finite* and the σ-field of events $\mathcal{F}$ consists of *all* subsets of Ω. We shall use the equivalent formulation of pairs (Ω, p) where Ω is the finite set of outcomes and where the *outcome probability assignment* $p : \Omega \rightarrow [0, 1]$ satisfies $\sum_{\omega \in \Omega} p(\omega) = 1$. Indeed, we take $\mathbb{P}[A] = \sum_{\omega \in A} p(\omega)$.

Definition 16: A **probabilistic(-information) database** (or p**-database**) is a finite probability space whose outcomes are all the conventional instances, i.e., a pair $(\mathcal{N}, p)$ where $\sum_{I \in \mathcal{N}} p(I) = 1$.

Demanding the direct specification of such probabilistic databases is unrealistic because there are 2^N possible instances, where $N := |\mathbb{D}|^n$, and we would need that many (minus one) probability values. Thus, as in the case of incomplete databases we define **probabilistic representation systems** consisting of "probabilistic tables" (prob. tables for short) and a function *Mod* that associates to each prob. table T a probabilistic database $\mathit{Mod}(T)$. Similarly, we define **completeness** (finite completeness is the only kind we have in our setting).

[1] Note however that the results remain true if $\mathbb{D}$ is finite; we just require an infinite supply of *variables*.

To define closure under a query language we face the following problem. Given a probabilistic database $(\mathcal{N}, p)$ and a query q (with just one input relation name), how do we define the probability assignment for the instances in $q(\mathcal{N})$? It turns out that this is a common construction in probability theory: image spaces.

Definition 17: Let (Ω, p) be a finite probability space and let $f : \Omega \to \Omega'$ where Ω' is some finite set. The **image** of (Ω, p) under f is the finite probability space (Ω', p') where[2] $p'(\omega') := \sum_{f(\omega)=\omega'} p(\omega)$.

Again we consider as query languages the relational algebra and its sublanguages defined by subsets of operations.

Definition 18: A probabilistic representation system is **closed** under a query language if for any query q and any prob. table T there exists a prob. table T' that represents $q(\mathit{Mod}(T))$, the image space of $\mathit{Mod}(T)$ under q.

6 Probabilistic ?-Tables and Probabilistic Or-Set Tables

Probabilistic ?-tables (p-?-tables for short) are commonly used for probabilistic models of databases [23, 10, 11, 6] (they are called "independent tuple representation in [20]). Such tables are the probabilistic counterpart of ?-tables where each "?" is replaced by a probability value. Example 3 below shows such a table. The tuples not explicitly shown are assumed tagged with probability 0. Therefore, a p-?-table is a mapping that associates to each $t \in \mathbb{D}^n$ a probability value p_t.

To define the *Mod* function we use another common construction from probability theory: product spaces.

Definition 19: Let $(\Omega_1, p_1), \ldots, (\Omega_n, p_n)$ be finite probability spaces. Their **product** is the space $(\Omega_1 \times \cdots \times \Omega_n, p)$ where[3] $p(\omega_1, \ldots, \omega_n) := p_1(\omega_1) \cdots p_n(\omega_n)$.

Given a p-?-table $T := \{p_t \| t \in \mathbb{D}^n\}$ consider the finite probability space $B_t := (\{\mathsf{true}, \mathsf{false}\}, p)$ where $p(\mathsf{true}) := p_t$ and $p(\mathsf{false}) = 1 - p_t$ and then the product space $P := \prod_{t \in \mathbb{D}^n} B_t$.

We can think of its set of outcomes (abusing notation, we will call this set P also) as the set of functions from $\mathbb{D}^n$ to $\{\mathsf{true}, \mathsf{false}\}$, in other words, predicates on $\mathbb{D}^n$. There is an obvious function $f : P \to \mathcal{N}$ that associates to each predicate the set of tuples it maps to true and this gives us a p-database, namely the image of P under f, which we define to be $\mathit{Mod}(T)$.

We define now another simple probabilistic representation system called **probabilistic or-set-tables** (p-or-set-tables for short). These are the probabilistic counterpart of or-set-tables where the attribute values are, instead of or-sets, finite probability spaces whose outcomes are the values in the or-set. p-or-set-tables correspond to a simplified version of the ProbView model presented in [16], in which plain probability values are used instead of confidence intervals.

Example 3: A p-or-set-table S, and a p-?-table T.

$S :=$

1	$\langle 2 : 0.3, 3 : 0.7 \rangle$
4	5
$\langle 6 : 0.5, 7 : 0.5 \rangle$	$\langle 8 : 0.1, 9 : 0.9 \rangle$

$T :=$

1	2	0.4
3	4	0.3
5	6	1.0

A p-or-set-table determines an instance by choosing an outcome in each of the spaces that appear as attribute values, *independently*. Recall that or-set tables are equivalent to finite-domain Codd tables. Similarly, a p-or-set-table corresponds to a Codd table T plus for each variable x in T a finite probability space $\mathrm{dom}(x)$ whose

[2] It is easy to check that the $p'(\omega')$'s do actually add up to 1.

[3] Again, it is easy to check that the outcome probability assignments add up to 1.

outcomes are in $\mathbb{D}$. This yields a p-database, again by image space construction, as shown more generally for c-tables next in section 7.

Query answering The papers [10, 23, 16] have considered, independently, the problem of calculating the probability of tuples appearing in query answers. This does *not* mean that in general $q(\mathit{Mod}(T))$ can be represented by another tuple table when T is some p-?-table and $q \in \mathcal{RA}$ (neither does this hold for p-or-set-tables). This follows from Proposition 12. Indeed, if the probabilistic counterpart of an incompleteness representation system $\mathcal{T}$ is closed, then so is $\mathcal{T}$. Hence the lifting of the results in Proposition 12 and other similar results.

Each of the papers [10, 23, 16] recognizes the problem of query answering and solves it by developing a more general model in which rows contain additional information *similar in spirit* to the conditions that appear in c-tables (in fact [10]'s model is essentially what we call probabilistic boolean c-tables, see next section). We will show that we can actually use a probabilistic counterpart to c-tables themselves together with the algebra on c-tables given in [14] to achieve the same effect.

7 Probabilistic c-tables

Definition 20: A **probabilistic c-table** (**pc-tables** for short) consists of a c-table T together with a *finite probability space* $\mathrm{dom}(x)$ (whose outcomes are values in $\mathbb{D}$) for each variable x that occurs in T.

To get a probabilistic representation system consider the product space $V := \prod_{x \in \mathit{Var}(T)} \mathrm{dom}(x)$. The outcomes of this space are in fact the *valuations* for the c-table T! Hence we can define the function $g : V \to \mathcal{N}, g(\nu) := \nu(T)$ and then define $\mathit{Mod}(T)$ as the image of V under g.

Similarly, we can talk about boolean pc-tables, pv-tables and probabilistic Codd tables (the latter related to [16], see previous section). Moreover, the p-?-tables correspond to restricted boolean pc-tables, just like ?-tables.

Theorem 21: Boolean pc-tables are complete (hence pc-tables are also complete).

The previous theorem was independently observed in [20].

Theorem 22: pc-tables (and boolean pc-tables) are closed under the relational algebra.

The proof of this theorem gives in fact an algorithm for constructing the answer as a p-database itself, represented by a pc-table. In particular this will work for the models of [10, 16, 23] or for models we might invent by adding probabilistic information to v-tables or to the representation systems considered in [19]. The interesting result of [6] about the applicability of an "extensional" algorithm to calculating answer tuple probabilities can be seen also as characterizing the conjunctive queries q which for any p-?-table T are such that the c-table $\bar{q}(T)$ is in fact equivalent to some p-?-table.

8 Conclusion

We reviewed some old and some new examples of representation systems for incomplete and probabilistic databases. We discussed notions of expressive completeness, and we gave a new notion of completeness, called $\mathcal{RA}$-completeness, which makes sense in the case of infinite domains. We introduced the concept of algebraic completion and gave some results showing that extending weaker models by various fragments of the relational algebra yields complete models. Finally, we showed how probabilistic representation systems can be seen as probabilistic counterparts of incomplete representation systems, and as an example we proposed a probabilistic representation system called pc-tables, which we showed to be closed and complete.

References

[1] S. Abiteboul and O. M. Duschka. Complexity of Answering Queries Using Materialized Views. In *PODS*, pages 254–263, 1998.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison–Wesley, Reading, MA, 1995.

[3] M. Arenas, L. E. Bertossi, and J. Chomicki. Answer sets for consistent query answering in inconsistent databases. *TPLP*, 3(4-5):393–424, 2003.

[4] D. Barbara, H. Garcia-Molina, and D. Porter. A probabilistic relational data model. In *EDBT*, 1990.

[5] R. Cavallo and M. Pittarelli. The Theory of Probabilistic Databases. In *VLDB*, pages 71–81, 1987.

[6] N. Dalvi and D. Suciu. Efficient Query Evaluation on Probabilistic Databases. In *VLDB*, 2004.

[7] D. Dey and S. Sarkar. A Probabilistic Relational Model and Algebra. *ACM TODS*, 21(3):339–369, 1996.

[8] R. Durrett. *Probability: Theory and Examples*. Duxbury Press, 3rd edition, 2004.

[9] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, pages 207–224, London, UK, 2003. Springer-Verlag.

[10] N. Fuhr and T. Rölleke. A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems. *ACM TODS*, 14(1):32–66, 1997.

[11] E. Grädel, Y. Gurevich, and C. Hirch. The Complexity of Query Reliability. In *PODS*, 1998.

[12] G. Grahne. *The Problem of Incomplete Information in Relational Databases*, volume 554 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1991.

[13] T. J. Green and V. Tannen. Models for Incomplete and Probabilistic Information. In *IIDB*, 2006.

[14] T. Imieliński and W. Lipski, Jr. Incomplete Information in Relational Databases. *J. ACM*, 31(4), 1984.

[15] T. Imieliński, S. A. Naqvi, and K. V. Vadaparty. Incomplete objects — a data model for design and planning applications. In *SIGMOD*, pages 288–297, 1991.

[16] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian. ProbView: a Flexible Probabilistic Database System. *ACM TODS*, 22(3):419–469, 1997.

[17] L. Libkin and L. Wong. Semantic representations and query languages for or-sets. *J. Computer and System Sci.*, 52(1):125–142, 1996.

[18] F. Sadri. Modeling Uncertainty in Databases. In *ICDE*, pages 122–131. IEEE Computer Society, 1991.

[19] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working Models for Uncertain Data. In *ICDE*, 2006.

[20] D. Suciu and N. Dalvi. Foundations of probabilistic answers to queries (tutorial). In *SIGMOD*, 2005.

[21] R. van der Meyden. Logical Approaches to Incomplete Information: A Survey. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*. Kluwer, 1998.

[22] J. Widom. Trio: A System for Integrated Management of Data, Accuracy, and Lineage. In *CIDR*, 2005.

[23] E. Zimányi. Query evaluation in probabilistic databases. *Theoretical Computer Science*, 171(1–2), 1997.

Query Evaluation on Probabilistic Databases

Christopher Ré, Nilesh Dalvi and Dan Suciu
University of Washington

1 The Probabilistic Data

In this paper we consider the query evaluation problem: how can we evaluate SQL queries on probabilistic databases? Our discussion is restricted to single-block SQL queries using standard syntax, with a modified semantics: each tuple in the answer is associated with a probability representing our confidence in that tuple belonging to the answer. We present here a short summary of the research done at the University of Washington into this problem.

Consider the probabilistic database in Fig. 1. $\texttt{Product}^p$ contains three products; their names and their prices are known, but we are unsure about their color and shape. Gizmo may be red and oval, or it may be blue and square, with probabilities $p_1 = 0.25$ and $p_2 = 0.75$ respectively. Camera has three possible combinations of color and shape, and IPod has two. Thus, the table defines for each product a probability distribution on its colors and shapes. Since each color-shape combination excludes the others, we must have $p_1+p_2 \leq 1$, $p_3+p_4+p_5 \leq 1$ and $p_6 + p_7 \leq 1$, which indeed holds for our table. When the sum is strictly less than one then that product may not occur in the table at all: for example Camera may be missing from the table with probability 1-p_3-p_4-p_5. Each probabilistic table is stored in a standard relational database: for example $\texttt{Product}^p$ becomes the table in Fig. 2 (a). For any two tuples in $\texttt{Product}^p$, if they have the same values of the key attributes `prod` and `price` then they are exclusive (i.e. disjoint) probabilistic events, otherwise they are independent events.

The meaning of a probabilistic database is a probability distribution on possible worlds. $\texttt{Product}^p$ has 16 possible worlds, since there are two choices for the color and shape for Gizmo, four for Camera (including removing Camera altogether) and two for IPod. Fig. 2 (b) illustrate two possible worlds and their probabilities.

$\texttt{Product}^p$

prod	price	color	shape	p
Gizmo	20	red	oval	$p_1 = 0.25$
		blue	square	$p_2 = 0.75$
Camera	80	green	oval	$p_3 = 0.3$
		red	round	$p_4 = 0.3$
		blue	oval	$p_5 = 0.2$
IPod	300	white	square	$p_6 = 0.8$
		black	square	$p_7 = 0.2$

`Order`

prod	price	cust
Gizmo	20	Sue
Gizmo	80	Fred
IPod	300	Fred

$\texttt{Customer}^p$

cust	city	p
Sue	New York	$q_1 = 0.5$
	Boston	$q_2 = 0.2$
	Seattle	$q_3 = 0.3$
Fred	Boston	$q_4 = 0.4$
	Seattle	$q_5 = 0.3$

Figure 1: Probabilistic database

Keys In this paper we impose the restriction that every deterministic attribute is part of a key. Formally, each probabilistic table `R` has a key, `R.Key`, and by definition this set of attributes must form a key in each

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Partially supported by Suciu's NSF Career Award IIS-00992955 and NSF Grants IIS-0428168, 61-2252, 61-2103, 0513877.

Product^p

prod	price	color	shape	p
Gizmo	20	red	oval	p_1
Gizmo	20	blue	square	p_2
Camera	80	green	oval	p_3
Camera	80	red	round	p_4
Camera	80	blue	oval	p_5
IPod	300	white	square	p_6
IPod	300	black	square	p_7

(a)

Product

prod	price	color	shape
Gizmo	20	blue	square
Camera	80	blue	oval
IPod	300	white	square

$p_2 p_5 p_6$

prod	price	color	shape
Gizmo	20	red	oval
IPod	300	white	square

$p_1(1\text{-}p_3\text{-}p_4\text{-}p_5)p_6$

(b)

Figure 2: Representation of a probabilistic table (a) and two possible worlds (b)

possible world. Intuitively, the attributes in `R.Key` are deterministic while the others are probabilistic. For example, in `Product(prod, price, shape, color)` the key `Product.Key` is {`prod, price`}, and one can see that it is a key in each of the two possible worlds in Fig. 2 (b). When a probabilistic table has only deterministic attributes, like in `R(A,B)`, the meaning is that each tuple occurs in the database with some probability ≤ 1, and any two tuples are independent events.

2 Easy Queries

```
Q1:  SELECT DISTINCT prod, price
     FROM Product
     WHERE shape='oval'
```

$Q_1(x, y) \;:-\; \texttt{Product}(\underline{x, y}, \texttt{'oval'}, z)$

```
Q2:  SELECT DISTINCT city
     FROM Customer
```

$Q_2(z) \;:-\; \texttt{Customer}(\underline{x, y}, z)$

```
Q3:  SELECT DISTINCT *
     FROM Product, Order, Customer
     WHERE Product.prod = Order.prod
     and Product.price = Order.price
     and Order.cust = Customer.cust
```

$Q_3(*) \;:-\; \texttt{Product}(\underline{x, y}, z), \texttt{Order}(x, y, u) \texttt{Customer}(\underline{u}, v)$

Figure 3: Three simple queries, expressed in SQL and in datalog

We start by illustrating with three simple queries in Fig. 3. The left columns shows the queries in SQL syntax, the right column shows the same queries in datalog notation. In datalog we will underline the variables that occur in the key positions. The queries are standard, i.e. they are written assuming that the database is deterministic, and ignore any probabilistic information. However, their semantics is modified: each tuple returned has an associated probability representing our confidence in that answer. For example the first query, Q_1, asks for all the oval products in the database, and it returns:

prod	price	p
Gizmo	20	p_1
Camera	80	$p_3 + p_5$

In general, given a query Q and a tuple t, the probability that t is an answer to Q is the sum of the probabilities of all possible worlds where Q returns t. For Q_1, the probability of Gizmo is thus the sum of the probabilities of the 8 possible worlds for `Product` (out of 16) where Gizmo appears as oval, and this turns out (after simplifications) to be p_1. In the case of Q_1 these probabilities can be computed without enumerating all possible worlds, directly from the table in Fig. 2 (a) by the following process: (1) Select all rows with `shape='oval'`, (2) project on `prod`, `price`, and `p` (the probability), (3) eliminate duplicates, by replacing their probabilities with the sum, because they are disjoint events. We call the operation consisting of the last two steps a disjoint project:

Disjoint Project, $\pi^{pD}_{\bar{A}}$ If k tuples with probabilities $p_1, \cdots, p_k$ have the same value, $\bar{a}$, for their $\bar{A}$ attributes, then the disjoint project will associated the tuple $\bar{a}$ with the probability $p_1 + \cdots + p_k$. The disjoint project is correctly applied if any two tuples that share the same values of the $\bar{A}$ attributes are disjoint events.

Q_1 can therefore be computed by the following plan: $Q_1 = \pi^{pD}_{\texttt{prod,price}}(\sigma_{\texttt{shape='oval'}}(\texttt{Product}^p))$. $\pi^{pD}_{\texttt{prod,price}}$ is correct, because any two tuples in $\texttt{Product}^p$ that have the same `prod` and `price` are disjoint events.

The second query asks for all cities in the `Customer` table, and its answer is:

city	p
New York	q_1
Boston	1-(1-q_2)(1-q_4)
Seattle	1-(1-q_3)(1-q_5)

This answer can also be obtained by a projection with a duplicate elimination, but now the probabilities $p_1, p_2, p_3, \ldots$ of duplicate values are replaced with 1-(1-p_1)(1-p_2)(1-p_3)..., since in this case all duplicate occurrences of the same city are independent. We call this an independent project:

Independent Project, $\pi^{pI}_{\bar{A}}$ If k tuples with probabilities $p_1, \cdots, p_k$ have the same value, $\bar{a}$, for their $\bar{A}$ attributes, then the independent project will associated the tuple $\bar{a}$ with the probability 1-(1-p_1)(1-p_2)$\cdots$(1-p_k). The independent project is correctly applied if any two tuples that share the same values of the $\bar{A}$ attributes are independent events.

Thus, the disjoint project and the independent project compute the same set of tuples, but with different probabilities: the former assumes disjoint probabilistic events, where $\mathbf{P}(t \vee t') = \mathbf{P}(t) + \mathbf{P}(t')$, while the second assumes independent probabilistic events, where $\mathbf{P}(t \vee t') = 1 - (1 - \mathbf{P}(t))(1 - \mathbf{P}(t')$. Continuing our example, the following plan computes Q_2: $Q_2 = \pi^{pI}_{\texttt{city}}(\texttt{Customer}^p)$. Here $\pi^{pI}_{\texttt{city}}$ is correct because any two tuples in $\texttt{Customer}^p$ that have the same `city` are independent events.

Finally, the third query illustrates the use of a join, and its answer is:

prod	price	color	shape	cust	city	p
Gizmo	20	red	oval	Sue	New York	p_1q_1
Gizmo	20	red	oval	Sue	Boston	p_1q_2
Gizmo	20	red	oval	Sue	Seattle	p_1q_3
Gizmo	20	blue	square	Sue	New York	p_2q_1
...	...					

It can be computed by modifying the join operator to multiply the probabilities of the input tables:

Join, $\bowtie^p$ Whenever it joins two tuples with probabilities p_1 and p_2, it sets the probability of the resulting tuple to be p_1p_2.

A plan for Q_3 is: $Q_3 = \texttt{Product} \bowtie^p \texttt{Order} \bowtie^p \texttt{Customer}$.

```
Schema:  R(A),S(A,B),T(B)
H1:  SELECT DISTINCT 'true' AS A
     FROM R, S, T
     WHERE R.A=S.A and S.B=T.B
```

$$H_1 \quad :- \quad R(\underline{x}), S(\underline{x, y}), T(\underline{y})$$

```
Schema:  R(A,B),S(B)
H2:  SELECT DISTINCT 'true' AS A
     FROM R, S
     WHERE R.B=S.B
```

$$H_2 \quad :- \quad R(\underline{x}, y), S(\underline{y})$$

```
Schema:  R(A,B),S(C,B)
H3:  SELECT DISTINCT 'true' AS A
     FROM R, S
     WHERE R.B=S.B
```

$$H_3 \quad :- \quad R(\underline{x}, y), S(\underline{z}, y)$$

Figure 4: Three queries that are #P-complete

3 Hard Queries

Unfortunately, not all queries can be computed as easily as the ones before. Consider the three queries in Fig. 4. All are boolean queries, i.e. they return either 'true' or nothing, but they still have a probabilistic semantics, and we have to compute the probability of the answer 'true'. Their schemas are kept as simple as possible: e.g. in $\texttt{H}_1$ table R has a single attribute A which forms a key (hence any two tuples are independent events). None of these queries can be computed in the style described in the previous section: for example, $\pi_{\emptyset}^{pI}(\texttt{R} \bowtie \texttt{S} \bowtie \texttt{T})$ is an incorrect plan because two distinct rows in R ⋈ S ⋈ T may share the same tuple in R, hence they are not necessarily independent events. In fact, we have:

Theorem 1: Each of the queries H_1, H_2, H_3 in Fig. 4 is #P-complete.

The complexity class #P is the counting version of NP, i.e. it denotes the class of problems that count the number of solutions to an NP problem. If a problem is #P-hard, then there is no polynomial time algorithm for it unless P = NP; in this case none of H_1, H_2, H_3 has a simple plan using the operators in Sec. 1. Both here and in the following section we assume that all relations are probabilistic, but some results extend to a mix of probabilistic and deterministic tables. For example H_1 is #P-complete even if the table S is deterministic.

4 The Boundary Between Hard and Easy Queries

We show now which queries are in PTIME and which are #P-complete. We consider a conjunctive query q in which no relation name occurs more than once (i.e. without self-joins). We use the following notations: *Head*(q) is the set of head variables in q, *FreeVar(q)* is the set of free variables (i.e. non-head variables) in q, R.Key is the set of free variables in the key position of the relation R, R.NonKey is the set of free variables in the non-key positions of the relation R, R.Pred is the predicate that q applies to R. For $x \in$ *FreeVar*(q), denote q_x a new query whose body is identical with q and where *Head*$(q_x) =$ *Head*$(q) \cup \{x\}$.

Algorithm 1 takes a conjunctive query q and produces a relational plan for q using the operators described in Sec. 2. If it succeeds, then the query is in PTIME; if it fails then the query is #P-complete.

Theorem 2:

1. Algorithm 1 is sound, i.e. if it produces a relational plan for a query q, then the plan correctly computes the output tuple probabilities for q.

Algorithm 1 FIND-PLAN(q)

If q has a single relation R and no free variables, then return $\sigma_{\texttt{R.Pred}}(\texttt{R})$.
Otherwise:

1. If there exists $x \in \mathit{FreeVar}(q)$ s.t. $x \in \texttt{R.Key}$ for every relation R in q, then return:

$$\pi^{pI}_{\mathit{Head}(q)}(\text{FIND-PLAN}(q_x))$$

2. If there exists $x \in \mathit{FreeVar}(q)$ and there exists a relation R s.t. $x \in \texttt{R.NonKey}$ and $\texttt{R.Key} \cap \mathit{FreeVar}(q) = \emptyset$, then return:

$$\pi^{pD}_{\mathit{Head}(q)}(\text{FIND-PLAN}(q_x))$$

3. If the relations in q can be partitioned into q_1 and q_2 such that they do not share any free variables, then return:

$$\text{FIND-PLAN}(q_1) \bowtie^p \text{FIND-PLAN}(q_2)$$

If none of the three conditions above holds, then q is #P-complete.

2. Algorithm 1 is complete, i.e. it does not produce a relational plan for a query only if the query is #P-hard.

As a consequence, every query that has a PTIME data complexity can in fact be evaluated using a relational plan. Any relational database engine can be used to support these queries, since the probabilistic projections and joins can be expressed in SQL using aggregate operations and multiplications.

Example 1: In the remainder of this section we illustrate with the following schema, obtained as an extension of our running example in Sec. 1.

```
Product(prod, price, color, shape)
Order(prod, price, cust)
CustomerFemale(cust, city, profession)
CustomerMale(cust, city, profession)
CitySalesRep(city, salesRep, phone)
```

All tables are now probabilistic: for example each entry in `Order` has some probability ≤ 1. The customers are partitioned into female and male customers, and we have a new table with sales representatives in each city. The following query returns all cities of male customers who have ordered a product with price 300:

$$Q(c) \quad :- \quad \texttt{Order}(\underline{x, 300, y}), \texttt{CustomerMale}(\underline{y}, c, z)$$

Here $\mathit{Head}(Q) = \{c\}$, $\mathit{FreeVar}(Q) = \{x, y, z\}$. Condition (1) of the algorithm is satisfied by the variable y, since $y \in \texttt{Order.Key}$ and $y \in \texttt{CustomerMale.Key}$, hence we generate the plan: $Q = \pi^{pI}_c(Q_y)$ where the new query Q_y is:

$$Q_y(c, y) \quad :- \quad \texttt{Order}(\underline{x, 300, y}), \texttt{CustomerMale}(\underline{y}, c, z)$$

The independence assumption needed for $\pi^{pI}_c(Q_y)$ to be correct indeed holds, since any two distinct rows in $Q_y(c, y)$ that have the same value of c must have distinct values of y, hence they consists of two independent tuples in `Order` and two independent tuples in `CustomerMale`. Now $\mathit{Head}(Q_y) = \{c, y\}$, $\mathit{FreeVar}(Q_y) = \{x, z\}$ and Q_y satisfies condition (2) of the algorithm (with $z \in \texttt{CustomerMale.NonKey}$

and $\texttt{CustomerMale.Key} = \{y\} \subseteq \mathit{Head}(Q_y)$), hence we generate the plan: $Q = \pi^{pI}_c(\pi^{pD}_{c,y}(Q_{y,z}))$ where the new query $Q_{y,z}$ is:

$$Q_{y,z}(c, y, z) \ :- \ \texttt{Order}(\underline{x, 300, y}), \texttt{CustomerMale}(\underline{y}, c, z)$$

The disjointness assumption needed for $\pi^{pD}_{c,y}(Q_{y,z})$ to be correct also holds, since any two distinct rows in $Q_{y,z}(c, y, z)$ that have the same values for c and y must have distinct values for z, hence they represent disjoint events in CustomerMale. $Q_{y,z}$ satisfies condition (3) and we compute it as a join between Order and CustomerMale. The predicate Order.Pred is $\texttt{price} =' 300'$, hence we obtain the following complete plan for Q:

$$Q = \pi^{pI}_c(\pi^{pD}_{c,y}(\sigma_{\texttt{price}='300'}(\texttt{Order}) \bowtie^p \texttt{CustomerMale}))$$

Recall the three #P-complete queries H_1, H_2, H_3 in Fig. 4. It turns out that, in some sense, these are the only #P-complete queries: every other query that is #P-complete has one of these three as a subpattern. Formally:

Theorem 3: Let q be any conjunctive query on which none of the three cases in Algorithm 1 applies (hence Q is #P-complete). Then one of the following holds:

1. There are three relations R, S, T and two free variables $x, y \in \mathit{FreeVar}(q)$ such that R.Key contains x but not y, S.Key contains both x, y, and T.Key contains y but not x. In notation:

$$\texttt{R}(\underline{x}, \ldots), \texttt{S}(\underline{x, y}, \ldots), \texttt{T}(\underline{y}, \ldots)$$

2. There are two relations R and S and two free variables $x, y \in \mathit{FreeVar}(q)$ s.t. such that x occurs in R.Key but not in S, and y occurs in R and in S.Key but not in R.Key. In notation:

$$\texttt{R}(\underline{x}, y, \ldots), \texttt{S}(\underline{y}, \ldots)$$

3. There are two relations R and S and three free variables $x, y, z \in \mathit{FreeVar}(q)$ s.t. x occurs in R.Key but not in S, x occurs in S.Key but not in R, and y occurs in both R and S but neither in R.Key nor in S.Key. In notation:

$$\texttt{R}(\underline{x}, y, \ldots), \texttt{S}(\underline{z}, y, \ldots)$$

Obviously, H_1 satisfies condition (1), H_2 satisfies condition (2), and H_3 satisfies condition (3). The theorem says that if a query is hard, then it must have one of H_1, H_2, H_3 as a subpattern.

Example 2: Continuing Example 1, consider the following three queries:

$$\begin{aligned}
HQ_1(c) \ &:- \ \texttt{Product}(\underline{x}, v, -, \texttt{'red'}), \texttt{Orders}(\underline{x, v, y}), \texttt{CustomerFemale}(\underline{y}, c, -) \\
HQ_2(sr) \ &:- \ \texttt{CustomerMale}(\underline{x}, y, \texttt{'lawyer'}), \texttt{CitySalesReps}(\underline{y}, sr, z) \\
HQ_3(c) \ &:- \ \texttt{CustomerMale}(\underline{x}, c, y), \texttt{CustomerFemale}(\underline{z}, c, y)
\end{aligned}$$

None of the three cases of the algorithm applies to these queries, hence all three are #P-complete. The first query asks for all cities where some female customer purchased some red product; it matches pattern (1). The second query asks for all sale representatives in cities that have lawyer customers: it matches pattern (2). The third query looks for all cities that have a male and a female customer with the same profession; it matches pattern (3).

Finally, note that the three patterns are a necessary condition for the query to be #P-complete, but they are sufficient conditions only after one has applied Algorithm 1 until it got stuck. In other words, there are queries that have one or more of the three patterns, but are still in PTIME since the algorithm eliminates free variables in a way in which it makes the patterns disappear. For example:

$$Q(v) \quad :- \quad R(\underline{x}), S(\underline{x, y}), T(\underline{y}), U(\underline{u}, y), V(\underline{v}, u)$$

The query contains the subpattern (1) (the first three relations are identical to H_1), yet it is in PTIME. This is because it is possible to remove variables in order u, y, x and obtain the following plan:

$$Q = \pi_v^{pD}(V \bowtie^p \pi_{v,u}^{pD}(U \bowtie^p T \bowtie^p \pi_y^{pI}(R \bowtie^p S)))$$

Theorem 3 has interesting connections to several existing probabilistic systems. In Cavallo and Pittarelli's system [2], all the tuples in a table `R` represent disjoint events, which corresponds in our model to `R.Key` $= \emptyset$. None of the three patterns of Theorem 3 can occur, because each pattern asks for at least one variable to occur in a key position, and therefore all the queries in Cavallo and Pittarelli's model have PTIME data complexity. Barbara et al. [1] and then Dey et al. [4] consider a system that allows arbitrary tables, i.e. `R.Key` can be any subset of the attributes of `R`, but they consider restricted SQL queries: all key attributes must be included in the `SELECT` clause. In datalog terminology, `R.Key` $\subseteq$ `Head`(q) for every table `R`, hence none of the three patterns in Theorem 3 can occur since each looks for at least one variable in a key position that does *not* occur in the query head. Thus, all queries discussed by Barbara et al. are in PTIME. Theorem 3 indicates that a much larger class of queries can be efficiently supported by their system. Finally, in our previous work [3], we consider a system where `R.Key` is the set of all attributes. In this case only case (1) of Theorem 3 applies, and one can check that now the pattern is a sufficient condition for #P-completeness: this is precisely Theorem 5.2 of [3].

5 Future Work

We identify three future research problems. (1) Self joins: we currently do not know the boundary between PTIME and #P-complete queries when some relation name occurs two ore more times in the query (i.e. queries with self-joins). (2) Query optimization: the relational operators π^{pD}, π^{pI}, $\bowtie^p$ and σ^p do not follow the same rules as the standard relational algebra. A combination of cost-based optimization and safe-plan generation is needed. (3) Queries that are #P-hard require simulation based techniques [5], which are expensive. However, often there are subqueries that admit safe-plans: this calls for investigations of mixed techniques, combining safe plans with simulations.

References

[1] Daniel Barbará, Hector Garcia-Molina, and Daryl Porter. The management of probabilistic data. *IEEE Trans. Knowl. Data Eng.*, 4(5):487–502, 1992.

[2] Roger Cavallo and Michael Pittarelli. The theory of probabilistic databases. In *VLDB*, pages 71–81, 1987.

[3] Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.

[4] Debabrata Dey and Sumit Sarkar. A probabilistic relational model and algebra. *ACM Trans. Database Syst.*, 21(3):339–369, 1996.

[5] Richard Karp and Michael Luby. Monte-carlo algorithms for enumeration and reliability problems. In *STOC*, 1983.

An Introduction to Probabilistic Graphical Models for Relational Data

Lise Getoor
Computer Science Department/UMIACS
University of Maryland
College Park, MD 20740
getoor@cs.umd.edu

Abstract

We survey some of the recent work on probabilistic graphical models for relational data. The models that we describe are all based upon 'graphical models' [12]. The models can capture statistical correlations among attributes within a single relational table, between attributes in different tables, and can capture certain structural properties, such as the expected size of a join between tables. These models can then be used for a variety of tasks including filling in missing values, data summarization and anomaly detection. Here we describe two complementary semantics for the models: one approach suited to making probabilistic statements about individuals and the second approach suited to making statements about frequencies in relational data. After describing the semantics, we briefly describe algorithms for automatically constructing the models from an existing (non-probabilistic) database.

1 Introduction

There has been a growing interest in probabilistic databases. This growing interest is largely due to expanding data collection opportunities and capabilities. In addition, recent advances in probabilistic modeling, both in terms of efficient learning and estimation algorithms and practical inference algorithms have made the use of these models for large-scale domains practical. In this survey paper, we review some of the work developed in the probabilistic reasoning and machine learning communities commonly refered to as *probabilistic relational models* (PRMs) [11]. PRMs are based on existing probabilistic models (typically Bayesian networks) and extend them to relational and first-order domains.

Halpern [9] identified two distinct semantics for first-order logics of probability, refered to as the *possible-worlds* approach and the *domain-frequency* approach. In the possible-worlds approach, we put a distribution over possible worlds. The distribution can be used to answer degree of belief questions such as 'What is the probability that a particular person, Joe Smith, has purchased a particular tractor part, Widget Supreme?'. In this case, the semantics of the formula is defined by summing up the probability over all of the possible worlds in which Joe Smith has bought Widget Supremes. However, as many have noted [2, 1], there are difficulties with making statistical assertions, such as '75% of the purchases were for widgets'. The second approach, refered to

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

as the domain-frequency approach, is appropriate for giving semantics to statistical queries such as, ‘What is the probability that a randomly chosen individual has purchased a widget?’. In this case the semantics are defined in terms of a particular world, and are the result of an experiment performed in this world. In this approach, statements such as ’80% of the purchases are made by men’ make sense. Whereas questions such as, ‘What is the probability Joe Smith purchased a Widget Supreme?’ do not make sense; the probability of the purchase, in a particular world, is either 0 or 1.

Here we survey some of the recent work on probabilistic graphical models for relational data and show how the two semantics are supported. We begin by introducing the syntax for probabilistic graphical models for relational data. We then describe the two alternate semantics for the models, and describe how they support the ability to make inferences about missing values and estimate the expected result set size for certain queries. Then, we briefly describe algorithms for automatically constructing the models.

2 Graphical Models for Relational Data: Syntax

First, we describe common syntactic elements of probabilistic relational models: the relational schema and the probabilistic schema. After these are introduced, we describe alternate semantics for the models.

2.1 The Relational Language

The relational schema defines the structure of our domain; it defines the tables and the attributes of the tables in the database. We assume a simple model, which obeys certain basic integrity constraints. In our relational model each table has a primary key, some descriptive attributes and possibly some foreign keys which are references to keys into other tables. For simplicity we assume primary keys are a single attribute, but this requirement can easily be lifted.

Definition 1: A *relation schema* $\mathcal{R}$ consists of a set of tables $\mathcal{R} = \{R_1, \ldots, R_m\}$. Each table R is associated with attributes of three types: a primary key $R.K$, a set $\mathcal{F}(R)$ of foreign keys, and a set $\mathcal{A}(R)$ of descriptive attributes. Each foreign key $R.F$ references a table $S \in \mathcal{R}$ and its domain, $\mathcal{V}(R.F)$, is a subset of $\mathcal{V}(S.K)$, $\mathcal{V}(R.F) \subseteq \mathcal{V}(S.K)$. Each descriptive attribute $R.A$ is associated with a domain of possible values $\mathcal{V}(R.A)$. ∎

Definition 2: A database instance $\mathcal{D}$ over $\mathcal{R}$ consists of a set of tuples $\mathcal{D}(R)$ for each table R. For each $t \in \mathcal{D}(R)$: (1) The primary key $t.K$ is unique within R; (2) For each $F \in \mathcal{F}(R)$, if F refers to tuples from S, $t.F$ is the primary key of some tuple in $\mathcal{D}(S)$; and, (3) For each $A \in \mathcal{A}(R)$, $t.A$ is a value in $\mathcal{V}(R.A)$. ∎

Note that, for simplicity, we assume referential integrity, but our model can be extended to accommodate cases where the foreign key may be *null*, indicating that there is no related tuple.

2.2 The Probabilistic Schema

In a probabilistic schema, we model the conditional dependence of descriptive attributes on other attribute values. To do this, we use a graphical dependence structure within which we can model local conditional probability distributions (CPDs) between descriptive attributes. The probabilistic schema gives a template which describes both the graphical dependence structure of the attributes and the parameters of the local conditional probability distribution for an attribute. Here, we define a simple generic syntax for the probabilistic schema; as we will see in the next sections, alternate probabilistic semantics can be defined using the same basic syntax.

Definition 3: A *probabilistic schema* $\mathcal{PS}$ for a relational schema $\mathcal{R}$ is a pair $(\mathcal{S}, \boldsymbol{\Theta})$. $\mathcal{S}$ is a directed graph with a node for each descriptive attribute $R.A$ in each of the tables in $\mathcal{R}$. For each $R.A$, $\mathcal{S}$ specifies the parents of each node, denoted Pa$(R.A)$, and $\boldsymbol{\Theta}$ specifies a CPD, written $P(R.A \mid \text{Pa}(R.A))$. ∎

The parents of a node $R.A$ are defined via a query over the relational schema. In the simplest case, a parent may be an attribute $R.B$ in the same table, in which case we annotate the edge with $\sigma_B(R)$. Another simple case is one in which a parent is defined via a foreign-key join. Suppose that $R.F$ is a reference to $S.K$ and the values of attribute $R.A$ are correlated with the corresponding attributes of $S.C$. In this case, we make $S.C$ to be a parent of $R.A$, and annotate the edge with $\sigma_C(R \bowtie_{R.F=S.K} S)$. By default, each edge represents the dependence on a single value. If the relationship between R and S is one-many, we compute an aggregate γ over C, where γ is any single-valued deterministic function over the multiset. The edge is then annotated with $\gamma_C(R \bowtie_{R.F=S.K} S)$. More complex chains of joins can be used to define parents. Furthermore, rather than specifying each parent as an independent query, it is also possible, though less common, to define a set of parents via a relational query and make use of arbitrary user-defined aggregates [6].

Each $R.A$ has an associated CPD; $\boldsymbol{\Theta}$ is the entire collection of CPDs. The CPD describes the generic statistical relationship between each node and its parents. The CPD specifies the distribution over the values of $R.A$ given any possible assignment of values to its parents.[1] Let $\mathcal{V}(R.A)$ denote the space of possible values for $R.A$ and $\mathcal{V}(\text{Pa}(R.A))$ denote the space of possible values for the parents of $R.A$. The CPD is *legal* if all of the conditional probabilities are non-negative, and if, for any particular instantiation of $R.A$, $a \in \mathcal{V}(R.A)$, the sum over all possible instantiations of $R.A$, for any particular instantiation, $\boldsymbol{u}$, of $R.A$'s parents is 1, i.e.,

$$\sum_{a \in \mathcal{V}(R.A)} P(R.A = a \mid \text{Pa}(R.A) = \boldsymbol{u}) = 1.$$

The CPD is a function that may be represented in a number of ways. A common representation is as a table of multinomials, in which we have a multinomial distribution for each possible instantiation of parents.. Alternatively, the CPD can be represented as a tree [3], where the interior vertices represent splits on the value of some parent of $R.A$, and the leaves contain distributions over the values of $R.A$. In this representation, we find the conditional distribution over $R.A$ given a particular choice of values $A_{k_1} = a_1, \ldots, A_{k_\ell} = a_\ell$ for its parents by following the appropriate path in the tree down to a leaf; when we encounter a split on some variable A_{k_j}, we go down the branch corresponding to the value of a_j; we then use the distribution stored at that leaf. There are a number of other possible compact functional forms for the distribution such as noisy-or and noisy-max.

3 Semantics

The previous section gave syntax for probabilistic relational models. In this section, we define two distinct semantics for the models. One, which we call the *possible worlds* approach, is based on defining a distribution over possible database instances. We will see that this approach is useful when we are attempting to generalize from observed data. This is useful for example, if we are trying to infer values for missing attributes in an incomplete database. The other semantics, which we call the *domain frequency* approach, is based on the defining a distribution over a randomly chosen tuple (or a collection of randomly chosen tuples). We will see that this approach is useful when we are attempting to represent a compact statistical summary of a particular database instance. The two semantics correspond, respectively, to the Type II and Type I approaches described in Halpern [9].

3.1 Probabilistic Relational Models: Possible Worlds Semantics

PRMs, as originally introduced [11], can be seen as extending directed graphical models, commonly refered to as Bayesian networks, to relational domains. The PRM syntax specifies a template for a probability distribution over a database. The template includes the relational component, that describes the relational schema for our

[1] Most often we assume that the attribute domains are discrete and finite; continuous domains can also be supported in which case we must specify conditional density functions.

domain, and the probabilistic component, that describes, in an abstract manner, the probabilistic dependencies that hold in the domain.

The possible worlds semantics for PRMs, which we will denote as PRM_{pw}, provides a coherent formal semantics in terms of probability distributions over sets of relational logic interpretations. Given a set of ground objects, a PRM_{pw} specifies a probability distribution over a set of interpretations involving these objects. A probabilistic schema, together with a partial database describing tuples and relations, defines a probability distribution over the unspecified attributes of the tuples. As such, it can be used effectively for representing *degree of belief*, which for example can be used when reasoning over missing values is required.

We will refer to a database instance $\mathcal{D}$ with no missing or unknown values as a *complete instantiation* of a schema. Each of these complete instantiations is considered a possible world, and a PRM_{pw} defines a probability distribution over database instances. In order to make this probability space well-defined, we need to constrain the space in some manner. Several different ways of specifying the probability space have been studied, with varying representational power.

Attribute uncertainty is the simplest way of defining the probability space. Intuitively, we assume that the set of objects and the relations between them are fixed, i.e., external to the probabilistic model. We specify the tuples and relations using a relational skeleton. Then, the PRM_{pw} defines a probability distribution over the assignments to the attributes of the objects in the model.

Definition 4: A *relational skeleton* $\mathcal{D}_s$ of a relational schema is a partial specification of a database instance. It specifies the value for the primary key $R.K$ and all of the foreign keys $F \in \mathcal{F}(R)$; however, it leaves the values of the attributes unspecified.

A PRM_{pw} defines a distribution over the possible worlds consistent with the relational skeleton. The relational skeleton implicitly defines the random variables in our domain; we have a random variable for each attribute of each object in the skeleton. A PRM_{pw} then specifies a probability distribution over *completions* $\mathcal{D}$ of the skeleton.

The probabilistic schema defines the qualitative dependency structure, $\mathcal{S}$. As we saw, the dependency structure associates with each attribute $R.A$ a set of parents $\text{Pa}(R.A)$. These correspond to *formal* parents; for different objects, the relational skeleton $\mathcal{D}_s$ determines the *actual* parents. In other words, the formal parents are defined by the probabilistic schema, however the relational skeleton provides the interpretation. The interpretation tells us which tuples join; this in turn tells us which attributes depend on each other. We will typically use capital letters to denote the generic attributes, $R.A$, and use lowercase letters to denote tuple variables, so that $r.A$ refers to the attribute of a specific tuple.

Definition 5: A *probabilistic relational model* Π_{pw} for a relational schema $\mathcal{R}$ and a probabilistic schema $\mathcal{PS} = (\mathcal{S}, \mathbf{\Theta})$, with *possible worlds semantics* is a triple $(\mathcal{S}, \mathbf{\Theta}, \mathcal{D}_s)$. The probabilistic schema together with the relational skeleton induces a "ground" Bayesian network:

- There is a random variable for every attribute of every object $r \in \mathcal{D}_s$, $r.A$.
- Each $r.A$ depends probabilistically on its parents according to the probabilistic schema and the relational skeleton, $\text{Pa}_{\mathcal{D}_s}(r.A)$.
- The CPD for $r.A$ is $P(R.A \mid \text{Pa}(R.A))$, as specified by the probabilistic schema.

The PRM_{pw} defines the following distribution:

$$P(\mathcal{D} : \mathcal{S}, \mathbf{\Theta}, \mathcal{D}_s) = \prod_{R_i \in \mathcal{R}} \prod_{r \in \mathcal{D}_s(R_i)} \prod_{A \in \mathcal{A}(R_i)} P(r.A \mid \text{Pa}_{\mathcal{D}_s}(r.A))$$

∎

This expression is very similar to the chain rule for Bayesian networks. There are three primary differences. First, our random variables correspond to the attributes of the tuples defined by the relational skeleton. Thus, a different relational skeleton will result in a distribution defined over a different set of random variables. Second, the set of parents of a random variable can vary according to the relational context of the object — the set of objects to which it is related. These are also determined by the relational skeleton. Third, the parameters are shared; the parameters of the local probability models for attributes of objects in the same class are identical.

As in any definition of this type, we must take care that the resulting function from instances to numbers defines a coherent probability distribution, i.e., the probability of each instance is between 0 and 1, and the sum of the probability of all instances is 1. In Bayesian networks, where the joint probability is also a product of CPDs, this requirement is satisfied if the dependency graph is acyclic: a variable is not an ancestor of itself. A similar condition is sufficient to ensure coherence in PRM$_{pw}$s as well. Additional details are given in [4].

In the discussion so far, we have assumed that the relational skeleton is external to the probabilistic model. The PRM$_{pw}$ framework can be extended to accommodate uncertainty about the structural relationships between objects as well as about their properties [5]. One approach to structural uncertainty we have developed is called *existence uncertainty*. Existence uncertainty is a simple approach to modeling the probability that a relationship exists between any two entities. Suppose we have the relation $Purchase(Key, Person, Item, Price)$. We add a Boolean attribute, the *Exists* attribute, and build a probabilistic model for it, just as any other attribute in our domain. We can model the probability that a person purchases a particular item. This can depend on both properties of the person and properties of the item. The approach is described more fully in [5]. We have also investigated the representation and learning of PRMs with class hierarchies. A discussion of these issues can be found in [4].

3.2 Probabilistic Relational Models: Domain Frequency Semantics

An alternative semantics for PRMs allows them to capture domain frequency information. These semantics describe a statistical model of a *particular* database instantiation. The model captures the tuple frequencies in the database, and in particular it captures the frequencies with which tuples *join*. We will refer to this flavor of probabilistic relational models as a PRM$_{df}$. These semantics are useful for describing a compact statistical model of a database. This compact model can then be used to efficiently answer question about the *expected* number of tuples that will satisfy a query. What makes PRM$_{df}$s unusual is they model correlations across tuples and they can flexibly answer a wide collection of queries. This has many possible uses, including for use by a query optimizer in choosing the appropriate query plan, and for use in computing approximate aggregate query answers. These semantics were first introduced in [8] and their utility for selectivity estimation was shown. Here, we give an introduction to the semantics, but for full details see [4].

As a simple illustration of the domain frequency semantics that we hope to capture, consider two tables R and S such that R has a foreign key, $R.F$, that points to $S.K$. We define a joint probability space over R and S using an imaginary sampling process that randomly samples a tuple r from R and independently samples a tuple s from S. The two tuples may or may not join with each other. We introduce a new *join indicator variable* to model this event. This variable, J_F, is binary valued; it is true when $r.F = s.K$ and false otherwise.

This sampling process induces a distribution

$$P_{\mathcal{D}}(J_F, A_1, \ldots, A_n, B_1, \ldots, B_m)$$

over the values of the join indicator J_F, the descriptive attributes of R, $\mathcal{A}(R) = \{A_1, \ldots, A_n\}$, and the descriptive attributes of S, $\mathcal{A}(S) = \{B_1, \ldots, B_m\}$.

Now, consider any query Q over R and S of the form: $r.\mathbf{A} = \mathbf{a}, s.\mathbf{B} = \mathbf{b}, r.F = s.K$ (where we abbreviate a multidimensional select using vector notation). The probability that this query is satisfied by a randomly chosen tuple r from R and s from S is simply:

$$P_{\mathcal{D}}(\mathbf{A} = \mathbf{a}, \mathbf{B} = \mathbf{b}, J_F = \textit{true}).$$

In other words, we can estimate the result for any query of this form using the joint distribution $P_{\mathcal{D}}$ defined using our sampling process.

Generalizing from this example to more general select-join queries, let Q be a query over tuple variables $r_1, \ldots, r_k$ (which may or may not refer to the same tables). Let σ_Q be the set of equality select clauses in Q; i.e., $\sigma_Q = \{r_{i_1}.A_{i_1} = a_{i_1}, \ldots, r_{i_l}.A_{i_l} = a_{i_l}\}$ and let $\bowtie_Q$ be the set of foreign-key joins in Q; i.e., $\bowtie_Q = \{r_{j_1}.F_{j_1} = s_{j_1}.K, \ldots, r_{j_m}.F_{j_m} = s_{j_m}.K\}$. We can write Q as follows:

$$\bowtie_Q (\sigma_Q(R_1 \times \ldots \times R_k)).$$

We introduce an indicator variable $\mathcal{I}_Q$ indicating when the equalities in Q hold.

Definition 6: For any query Q over database $\mathcal{D}$, the joins in Q induce a distribution $P_{\mathcal{D}}(Q)$ over the attributes of the tuple variables in Q, $R_1 \times \ldots \times R_k$. The probability that the query is satisfied is:

$$P_{\mathcal{D}}(\mathcal{I}_Q) = \frac{| \bowtie_Q (\sigma_Q(R_1 \times \ldots \times R_k))|}{|R_1| \times \ldots \times |R_k|} . \blacksquare$$

We can also view this joint distribution as generated by an imaginary process, where we uniformly and independently sample a sequence of tuples $r_1, \ldots, r_k$ from $R_1, \ldots, R_k$, and then select as the values of $A_1, \ldots, A_n$ the values of $r.A_1, \ldots, r.A_n$.

While the expression in Definition 6 is computable for any select-join query, we will find it more useful if we can define a unique distribution induced by our database. To achieve this, we restrict our attention to a finite, acyclic collection of foreign-key joins. If there exists a partial ordering $\prec$ over the tables in our schema $\mathcal{R}$, such that, for any foreign key $R.F$ with $\mathcal{V}(R.F) \subseteq \mathcal{V}(S.K)$, $S \prec R$, then we say that $\mathcal{R}$ is *table-stratified*.

Now we can define a new relation $\mathcal{U}$, which is the *universal foreign-key closure* of a database $\mathcal{D}$ with respect to a table stratification. This relation is never actually materialized, we merely use it as a tool in defining a unique distribution induced by our database. Intuitively, the query that we construct introduces a tuple variable for each table in our schema, and has a unique tuple variable for each foreign-key in the schema.

Definition 7: Let $\mathcal{D}$ be a database with relational schema $\mathcal{R}$ and let $\prec$ be a table stratification of $\mathcal{D}$. The *universal foreign-key closure* of $\mathcal{D}$ is defined by the query $\mathcal{U}$ we construct below. $\mathcal{T}(\mathcal{U})$ will be the set of tuples variables in our query. Initially, $\mathcal{T}(\mathcal{U})$ has one tuple variable for each of the tables that are leaves in $\prec$. Each tuple variable is initially marked unprocessed.

We will construct the full set of tuple variables in the query as follows. While there are tuple variables in $\mathcal{T}(\mathcal{U})$ that have not been processed:

- Let r be an unprocessed tuple variable in $\mathcal{T}(\mathcal{U})$.
- Suppose r is a tuple from R. For each $F \in \mathcal{F}(R)$, where $R.F$ refers to S, add a new unique tuple variable s to $\mathcal{T}(\mathcal{U})$. This tuple variable is marked unprocessed. We say that s is the tuple variable *associated* with $r.F$. We also add the join $r.F = s.K$ to $\bowtie_{\mathcal{U}}$.

Let $\mathcal{A}(\mathcal{U}) = \{A_1, \ldots, A_m\}$ be the attributes in $\mathcal{U}$ (in order to avoid ambiguity, assume attributes are prefixed by their associated tuple variable), and let $\mathcal{T}(\mathcal{U}) = \{t_1, \ldots, t_j\}$ be the tuple variables in the query. $\mathcal{U}$ is simply a query over the cross product of the relations with a new copy of the relation introduced for each tuple variable that we add. $\blacksquare$

Given this query $\mathcal{U}$, we can define the probability distribution $P_{\mathcal{U}}$. It is the distribution induced by the occurrence frequency of different value combinations for the attributes of the tuple variables in $\mathcal{U}$ and join events among the tuple variables. PRM$_{df}$s allow us to compactly model $P_{\mathcal{U}}$. A probabilistic schema $\mathcal{PS}$ describes a set of independence assumptions and the local distributions of attributes given their parents. Intuitively, we say that a database $\mathcal{D}$ is a model of a probabilistic schema $\mathcal{PS}$, $\mathcal{D} \models \mathcal{PS}$, if the conditional independence assumptions made in $\mathcal{PS}$ hold in $\mathcal{D}$ and if the local distributions match the frequencies in $\mathcal{D}$. This is made more precise in [4].

Further, PRM$_{df}$s allow us to efficiently answer certain frequency queries by constructing a small query-specific Bayesian network that can be used to compute the desired frequencies. We begin by restricting attention to a form of select-join queries over multiple tables we call *inverted-tree foreign-key-join* queries. These queries are over a subset of the tuple variables of the universal foreign-key closure. Intuitively, they are over an upwardly closed fragment of the forest defined via the foreign-key joins in $\mathcal{U}$, and may themselves form a forest. We refer to these as *legal* queries.

Given a probabilistic schema $\mathcal{PS}$ and a legal query Q, the domain frequency semantics are defined as follows:

Definition 8: Suppose PRM$_{df}$ consists of $\mathcal{PS} = (\mathcal{S}, \mathbf{\Theta})$ over $\mathcal{D}$ and $\mathcal{D} \models \mathcal{PS}$. Let Q be a legal query. $Vars(Q)$ is a set of random variables which includes, for each attribute $r.A$ of every tuple variable $r \in Q$, $r.A$, and for each $r.F = s.K$, it has a random variable $r.J_F$. For every node $r.V$ introduced, $r.V$ has parents $\text{Pa}(r.V)$ defined by $\mathcal{S}$ and the CPD of $r.V$ is as specified in $\mathbf{\Theta}$. Then, $\mathcal{PS}$ defines the following distribution:

$$P_{\mathcal{D}}(Q) = \prod_{A_i \in Vars(Q)} P(A_i \mid \text{Pa}A_i, J^*_{\mathcal{U}}(A_i) = \mathbf{T})$$

$J^*_{\mathcal{U}}(A_i)$ are the join indicator variables that are ancestors of A_i in $\mathcal{U}$. ∎

4 PRM Construction

Now that we have seen the different semantics for PRMs, we have left the question of how the models can be automatically constructed. It turns out that although the semantics for the models are very different, the learning algorithms are largely the same and are closely related to the work in learning Bayesian networks [10]. The input to the construction algorithm is the relational schema, including the possible foreign-key joins between tuples; and a database instance. Our goal is to find a probabilistic schema $\mathcal{PS}(\mathcal{S}, \mathbf{\Theta})$ that optimizing some scoring criteria.

We can set this problem up as an optimization problem. There are three important components that need to be defined: the **hypothesis space** which specifies which structures are candidate hypotheses that our learning algorithm can return; a **scoring function** that evaluates the "goodness" of different candidate hypotheses relative to the data; and the **search algorithm**, a procedure that searches the hypothesis space for a structure with a high score.

Hypothesis Space. A hypothesis $\mathcal{PS}$ specifies a set of parents for each attribute $R.A$. We must restrict attention to probabilistic schemas which will generate a consistent probability model for any skeleton or query we are likely to see. We can do this by constructing a dependency graph for the candidate structure and ensuring that the class graph is acyclic. We maintain the graph during learning, and consider only models whose dependency structure passes the appropriate test; see [4] for more details.

Scoring. The second key component is the ability to evaluate different structures in order to pick one that fits the data well. We adapt Bayesian *model selection* methods to our framework. Bayesian model selection utilizes a probabilistic scoring function. In line with the Bayesian philosophy, it ascribes a prior probability distribution over any aspect of the model about which we are uncertain. In this case, we have a prior $P(\mathcal{S})$ over structures, and a prior $P(\mathbf{\Theta} \mid \mathcal{S})$ over the parameters given each possible structure. The *Bayesian score* of a structure $\mathcal{S}$ is defined as the *posterior* probability of the structure given the data $\mathcal{D}$:

$$P(\mathcal{S} \mid \mathcal{D}) \propto P(\mathcal{D} \mid \mathcal{S}) P(\mathcal{S})$$

where the denominator, which is the marginal probability $P(\mathcal{D})$ is a normalizing constant that does not change the relative rankings of different structures. This score is composed of two main parts: the prior probability of the structure, and the probability of the data given that structure. The marginal likelihood is a crucial component, which has the effect of penalizing models with a large number of parameters. Thus, this score automatically balances the complexity of the structure with its fit to the data. In the case where $\mathcal{D}$ is a complete assignment, and we make certain reasonable assumptions about the structure prior, there is a closed form solution for the score.

Search Algorithm. The simplest heuristic search algorithm is greedy hill-climbing search, using our score as a metric. We maintain our current candidate structure and iteratively improve it. At each iteration, we consider a set of simple local transformations to that structure, score all of them, and pick the one with highest score. We restrict attention to simple transformations such as adding, deleting or reversing an edge. We stop when we have reached a local optimum in the model space. We can make our search procedure more robust by incorporating any of the standard extensions to greedy search such as random restarts, maintaining a tabu list or using a simulated annealing form of search algorithm. While this is not guaranteed to find the optimal solution, it is quite efficient and works surprisingly well in practice.

5 Conclusion

We have presented two approaches to structured statistical models: a possible worlds semantics and a domain frequency semantics. Experimental results using these models have been presented elsewhere: we have shown that they are useful for classifying web pages and bibliographic citations [5], for discovering patterns in tuberculosis epidemiology [7]; and for selectivity estimation in a variety of real-world domains [8]. Here we have attempted to present a concise summary of the two approaches, leaving out some of the technical details, in an effort to make the methods more accessible and to highlight the commonalities and differences between the approaches, and contrast with some of the other approaches described in this special issue.

References

[1] F. Bacchus. On probability distributions over possible worlds. In *Proceedings of 4th Workshop on Uncertainty in Artificial Intelligence*, 1988.

[2] F. Bacchus. *Representing and Reasoning with Probabilistic Knowledge*. MIT Press, 1990.

[3] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 1996.

[4] L. Getoor. *Learning Probabilistic Models of Relational Data*. PhD thesis, Stanford University, 2001.

[5] L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models with link uncertainty. *Journal of Machine Learning Research*, 3:679—707, 2002.

[6] L. Getoor and J. Grant. PRL: A logical approach to probabilistic relational models. *Machine Learning Journal*, 26, 2006.

[7] L. Getoor, J. Rhee, D. Koller, and P. Small. Understanding tuberculosis epidemiology using probabilistic relational models. *AI in Medicine Journal*, 30:233—256, 2004.

[8] L. Getoor, B. Taskar, and D. Koller. Using probabilistic models for selectivity estimation. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 2001.

[9] J. Halpern. An analysis of first-order logics for reasoning about probability. *Artificial Intelligence*, 46:311–350, 1990.

[10] D. Heckerman. A tutorial on learning with Bayesian networks. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 301–354. MIT Press, 1998.

[11] D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proceedings of the 15th National Conference on Artificial Intelligence*, 1998.

[12] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

Avatar Information Extraction System

T.S. Jayram, Rajasekar Krishnamurthy, Sriram Raghavan,
Shivakumar Vaithyanathan, Huaiyu Zhu

IBM Almaden Research Center
{jayram,rajase,rsriram,vaithyan,huaiyu}@us.ibm.com

Abstract

The **AVATAR** *Information Extraction System (IES) at the IBM Almaden Research Center enables high-precision, rule-based, information extraction from text-documents. Drawing from our experience we propose the use of probabilistic database techniques as the formal underpinnings of information extraction systems so as to maintain high precision while increasing recall. This involves building a framework where rule-based annotators can be mapped to queries in a database system. We use examples from* **AVATAR** *IES to describe the challenges in achieving this goal. Finally, we show that deriving precision estimates in such a database system presents a significant challenge for probabilistic database systems.*

1 Introduction

Text analytics is a mature area of research concerned with the problem of automatically analyzing text to extract structured information. Examples of common text analytic tasks include *entity identification* (e.g., identifying persons, locations, organizations, etc.) [1], *relationship detection* (e.g., person X works in company Y)[9] and *co-reference resolution* (identifying different variants of the same entity either in the same document or different documents) [8]. Text analytic programs used for information extraction are called **annotators** and the objects extracted by them are called **annotations**. Traditionally, such annotations have been directly absorbed into applications. Increasingly, due to the complex needs of today's enterprise applications (such as Community Information Management [3]), there is a need for infrastructure that enables information extraction, manages the extracted objects and provides an easy interface to applications. Moreover, a very important pre-requisite for the use of annotations in enterprise applications is *high precision*.

At the IBM Almaden Research Center we are currently building the **AVATAR** Information Extraction System (IES) to tackle some of these challenges. Drawing from our experience in building the **AVATAR** IES infrastructure, in this paper, we make a case for the use of probabilistic database techniques as the formal underpinnings of such information extraction systems.

Annotations and Rules. Annotators in **AVATAR** IES are classified into two categories based on their input:

- **Base annotators** operate over the document text, independent of any other annotator.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

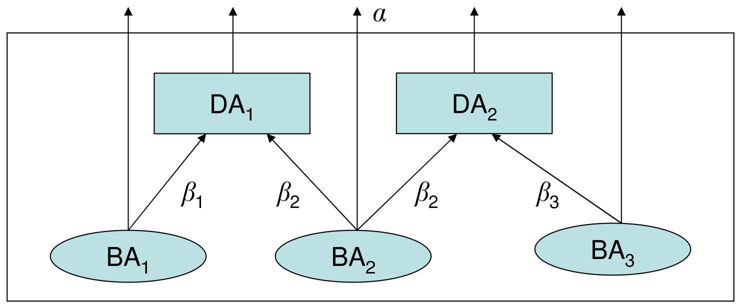

Figure 1: Thresholding annotator precisions

- **Derived annotators** operate on document text as well as the annotation objects produced by other annotators.

Every annotation object produced by an annotator is characterized by a well-defined set of operations. We refer to each such operation as a *rule*. A set of rules within an annotator that together produce an annotation object is called a *meta-rule*. As an example, consider a simple base annotator that identifies occurrences of person names in the text of a document. An example of a meta-rule used by such an annotator would be (informally) R_1: *look for the presence of a salutation such as Dr. or Mr. followed by a capitalized word*. Meta-rule R_1 would identify "Dr. Stonebraker" as a candidate Person annotation. Since a derived annotation depends on other annotation objects, the concept of meta-rule history is useful:

Definition 1 (Meta-Rule History): The *meta-rule history* $\mathcal{H}(a)$ of an annotation object a is defined as follows: If a is a base annotation produced by meta-rule R, then $\mathcal{H}(a) = \langle R \rangle$; if a is a derived annotation object produced by a meta-rule R that operates on previously defined (base or derived) annotation objects $a_1, \ldots, a_k$, then $\mathcal{H}(a) = \langle R, \mathcal{H}(a_1), \ldots, \mathcal{H}(a_k) \rangle$.

The confidence in the accuracy of an annotation is related to its meta-rule history. For example, the person annotator mentioned above may use a different meta-rule R_2 that looks for capitalized words that may or may not be person names (e.g., to identify "Bill" as a candidate Person). Intuitively, the annotator has higher confidence in R_1 and therefore, higher confidence in the accuracy of the objects produced by R_1. To formalize this intuition, we characterize the precision of individual annotation objects as follows:

Definition 2 (Annotation Object Precision): The precision $\mathrm{prec}(a)$ of an annotation object a is defined as the confidence value in $[0, 1]$ given by the annotator to all objects that can be produced with the same meta-rule history as that of a.

Definition 3 (High-precision Information Extraction (HPIE) System): An information-extraction system in which the precision of all annotation objects are above a threshold α (α close to 1.0)[1] is a high-precision information-extraction system.

In accordance with Definitions 1 and 2, the precision for derived annotations is computed using the entire meta-rule history of the corresponding base and derived annotators. This approach has two severe drawbacks. First, the combinatorial explosion in the space of meta-rules renders information-extraction systems of any reasonable size to be intractable. Second, there is a sparsity problem arising out of the fact that there may not be enough evidence (i.e., not enough annotation objects) to obtain precision estimates for all meta-rule histories.

[1]The choice of α will be driven by application requirements.

Current Implementation in AVATAR IES. The current implementation of **AVATAR** IES uses the UIMA [6] workflow engine to execute annotators. Annotations produced in **AVATAR** IES are stored in an annotation store implemented using a relational database (DB2). The store allows multiple derived annotators to be executed without having to re-execute the base annotators.

To overcome problems associated with maintaining meta-rule history, **AVATAR** IES implicitly assumes that all α-filtered input objects to a derived annotator are of equal precision. However, this approach has several problems that we explain below using Figure 1. In this figure, derived annotator (DA_1) has inputs from two base annotators BA_1 and BA_2. For an application that requires an HPIE system with $\alpha = 0.9$, the naive approach of α-filtering the objects of BA_1 and BA_2 may not be sufficient for DA_1 to produce derived objects with $\alpha \geq 0.9$.[2] The reason is that a derived annotator might require input annotations to be filtered at thresholds different from α in order to produce derived annotations above α. To account for this, annotators are thresholded differently (the β's in Figure 1) for consumption by derived annotators. This process can become complicated if multiple β's are required for a single base annotator whose output is consumed by different derived annotators. To minimize the need to tune a large number of β's, in the current implementation of **AVATAR** IES, we only allow two β settings for each annotator, namely, *high* and *medium*.

Motivation for Probabilistic Databases. Even under the simplistic assumptions made in our implementation, two problems remain. First, as **AVATAR** IES scales to a large number of annotators, the task of setting β's can quickly become intractable.[3] Second, the choice of β has a significant impact on the recall of derived annotators. As an example, consider a derived annotator PersonPhone (see Section 2.2) that uses Person annotations produced by a base annotator. In **AVATAR** IES, by switching the β for Person from *medium* to *high*, the number of annotation objects produced by PersonPhone over the Enron email data set [4] drops from 910 to 580.

Below, we motivate the use of probabilistic database techniques [2, 7] as a potential solution to these problems. Our first step in this direction is to view the precision of an annotation object in probabilistic terms.

Assumption 4: Consider an object a of type $\mathbf{type}(a)$. The precision $\mathrm{prec}(a)$ of a can be interpreted as a probability as follows: let $\widetilde{a}$ be drawn at random from the set of annotation objects whose meta-rule histories are identical to that of a. Then $\mathrm{prec}(a)$ equals the probability that $\widetilde{a}$ is *truly* an object of type $\mathbf{type}(a)$[4]. Formally, $\mathrm{prec}(a) = P(\mathbf{tt}(\widetilde{a}) = \mathbf{type}(a) \mid \mathcal{H}(a) = \mathcal{H}(\tilde{a}))$.

Assumption 5: Let R be a meta-rule in a derived annotator that takes as input k objects of specific types $T_1, \ldots, T_k$. Let the derived annotation object a be produced by R using annotation objects $a_1, a_2, \ldots, a_k$ of types $T_1, \ldots, T_k$, respectively, i.e., $a = R(a_1, \ldots, a_k)$. Let $\widetilde{a}$ and $\widetilde{a}_i$ correspond to a and a_i, for $i = 1 \ldots k$, as defined in Assumption 4. Then, $\mathbf{tt}(\widetilde{a}) = T \implies \forall\, i :\ \mathbf{tt}(\widetilde{a}_i) = T_i$.

Proposition 6: Using Assumptions 4 and 5, we can express the precision of an annotation object produced by a derived annotator as

$$\begin{aligned}
\mathrm{prec}(a) &= P(\mathbf{tt}(\widetilde{a}) = T \mid \mathcal{H}(a) = \mathcal{H}(\widetilde{a})) \\
&= P(\mathbf{tt}(\widetilde{a}) = T \mid \widetilde{a} = R(\widetilde{a}_1, \ldots, \widetilde{a}_k), \{\mathbf{tt}(\widetilde{a}_i) = T_i\}, \{\mathcal{H}(a_i) = \mathcal{H}(\widetilde{a}_i)\}) && \text{(meta-rule-prec)} \\
&\quad \cdot P(\{\mathbf{tt}(\widetilde{a}_i) = T_i\} \mid \widetilde{a} = R(\widetilde{a}_1, \ldots, \widetilde{a}_k), \{\mathcal{H}(a_i) = \mathcal{H}(\widetilde{a}_i)\}) && \text{(input-prec)}
\end{aligned}$$

[2] Indeed for very high α a derived annotator may produce no objects.

[3] Today **AVATAR** IES has about a hundred annotators. However, the infrastructure for annotator development is sufficiently powerful that we expect this number to go up dramatically (potentially to tens of thousands).

[4] This is the result of an experimental procedure wherein an expert examines each object and determines whether it is truly of the intended type.

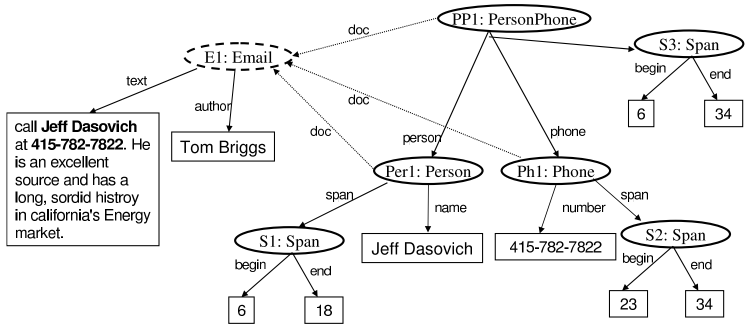

Figure 2: Annotation Store

In the proposition above, the expression (meta-rule-prec) represents meta-rule precision while the expression (input-prec) represents the overall input precision (see Section 4.3 for details and examples). While Assumption 4 has allowed us to reduce the problem of computing precision of derived object to one of computing probabilities for meta-rule precision and overall input precision, the expressions in Proposition 6 appear intractable. To obtain a practical solution, we believe that a better understanding of annotators and their dependencies is essential.

In the rest of the paper we describe the internals of **AVATAR** IES and connections to probabilistic databases as appropriate. In Section 2.2, we describe a generic template for **AVATAR** rule-based annotators. In Section 3, we consider a simple probability model for base annotators. Finally, in Section 4, we discuss briefly efficiency issues related to derived annotators.

2 Rule-based Annotators

2.1 Annotator Data Model

Each annotation produced by an IES can be viewed as a structured object with a well-defined type. The overall output of an IES, called an annotation store, is a collection of such objects as defined below:

Definition 7 (annotation store): An annotation store $\mathcal{S} = (\mathcal{T}, \mathcal{O})$ consists of a set of types $\mathcal{T}$, a set of objects $\mathcal{O}$, and two distinguished types $D, S \in \mathcal{T}$, such that :

- there is a special attribute text for type D
- $\forall x \in \mathcal{O}\ \mathbf{type}(x) \in \mathcal{T}$
- $\forall x \in \mathcal{O}$ such that $\mathbf{type}(x) \neq D, \mathbf{type}(x) \neq S$, there exist attributes doc and span with $\mathbf{type}(x.\mathsf{doc}) = D$ and $\mathbf{type}(x.\mathsf{span}) = S$.

In this definition, D is called the *document type*, S is called the *span type*, and every other type in $\mathcal{T}$ is an *annotation type*. The special attribute text refers to the raw text of each document. The doc attribute of an annotation object A points to the source document from which A was extracted. Similarly, the span attribute of A describes the portion of A.doc.text where A was mentioned. Figure 2 shows a simple annotation store with one document object of type Email and three annotation objects. Each oval in the figure represents one object and is labeled as A:B where A is the ID of the object and B is the type. The rectangular boxes represent atomic attribute values. In this example the *span type* S contains a pair of integers begin and end that store the character offsets of the piece of the text corresponding to the annotation.

For the purposes of this paper, we assume that every annotation is extracted from a single document and that the span is interpreted relative to the text of this document.

procedure Annotator(d**,** $\mathcal{A}_d$)
R_f *is a set of rules to generate features*
R_g *is a set of rules to generate candidate annotations*
R_c *is a set of rules to consolidate the annotations produced by* R_g

1. *Features* = Compute_Features(R_f,d)
2. **foreach** $r \in R_g$
   Candidates = *Candidates* $\cup$ Apply_Rule(r,*Features*,$\mathcal{A}_d$)
3. *Results* = Consolidate(R_c,*Candidates*)
   return *Results*

Figure 3: **Generic Template for a Rule-based Annotator**

2.2 Rule-based Annotator Template

Figure 3 is a high-level template that describes the steps involved in a rule-based annotator. The input to the annotator is a document d and a set of annotation objects $\mathcal{A}_d$ such that $\forall x \in \mathcal{A}_d, x.\mathsf{doc} = d$ ($\mathcal{A}_d = \emptyset$ for base annotators). A rule-based annotator uses three kinds of rules: *feature extraction rules* R_f, *candidate generation rules* R_g, and *consolidation rules* R_c. Every rule operates on one or more annotation objects to produce other annotation objects. However, to distinguish between the objects produced by these different kinds of rules, we use the terms *features*, *candidates*, and *result annotations* respectively.

In the first step, a rule-based annotator uses the rules in R_f to produce a set of features. Note that all of the rules in R_f operate on the document and do not involve input annotations. In the second step, each rule in R_g is independently applied to the features and the input annotations. The output of each such rule is a set of candidates. Finally, the rules in R_c are used to consolidate candidates to produce a set of *Results*. Consolidation rules typically fall into two broad categories: (i) *discard rules* that discard some candidates, and (ii) *merge rules* that merge a set of candidates to produce a result annotation.

In **AVATAR** IES, the feature extraction rules are deterministic operations over the document that do not impact the precision of the result annotations. Therefore, the meta-rule for a result annotation A is the set of candidate generation and consolidation rules (rules from R_g and R_c) that are involved in producing A.

As concrete instantiations of the template in Figure 3, we describe below a base annotator called SimplePerson and a derived annotator called PersonPhone.

Base Annotator SimplePerson. Base annotator SimplePerson identifies persons using two dictionaries – D_u (containing unambiguous first or last names such as "michael", "stonebraker", etc.) and D_a (containing ambiguous first or last names such as "bill", "gray"). Expressed in terms of the template in Figure 3, SimplePerson works as follows:

- In the Compute_Features step, the input document D is tokenized and each individual token is added to the set of features *Features*. Further, for each feature F, an attribute dictU (resp. dictA) is set to true if the corresponding token matches an entry in D_u (resp. D_a).
- The set of rules R_g for identifying person names are: (i) r_{uu}: a pair of features that are adjacent to each other in the document text and both of which are labeled with D_u (e.g., michael stonebraker), (ii) r_{ua}: a pair of features that are adjacent to each other in the document text and are labeled with D_u and D_a respectively (e.g., james gray), (iii) r_u: a feature that is labeled with D_u (e.g., michael), and (iv) r_a: a feature that is labeled with D_a (e.g., gray).
- Step 3 consists of a single consolidation rule r_d that executes the following logic: *If two candidates o_1 and o_2 are such that the o_1.span contains o_2.span, discard o_2.* This rule is applied repeatedly to produce *Results*.

Note that SimplePerson is a simple but powerful annotator that we use in this paper to illustrate some key ideas. In reality, a person annotator will use a significantly larger set of rules that exploit feature attributes such as capitalization, the presence of salutations (e.g., Dr., Mr.), etc.

Derived annotator PersonPhone. Derived annotator PersonPhone identifies people's phone numbers from documents. This annotator takes in as input the document D and a set of Person, Phone and ContactPattern annotations identified by *Base* annotators. Here (i) Person annotations are produced by a person annotator such as SimplePerson, (ii) Phone annotations are produced by an annotator that identifies telephone numbers, and (iii) ContactPattern annotations are produced by an annotator that looks for occurrences of phrases such as *"at"*, *"can be (reached|called) at"* and *"'s number is"*. Given these inputs, the PersonPhone annotator is fully described by the following two rules:

- Candidate generation rule r_{seq}: *If a triple $\langle$Person,ContactPattern,Phone$\rangle$ appear sequentially in the document, create a candidate PersonPhone annotation*. An example instance identified using this rule is shown in Figure 2.
- Consolidation rule r_d as described earlier.

3 Probability Model for Base Annotators

Let us revisit the rules of the SimplePerson annotator that was described in Section 2.2. Since the only consolidation rule in this annotator r_d is a *discard rule*, it does not affect the probability assigned to the result objects. Therefore, each generation rule in SimplePerson fully describes a meta-rule. For each candidate generation rule in the SimplePerson annotator, let us assume the corresponding precision values are available to us (e.g., we are given $\mathrm{prec}(r_{uu})$, $\mathrm{prec}(r_{ua})$, etc.). The precision value associated with a rule is a measure of the confidence that the annotator writer has in the accuracy of that rule. An experimental procedure to compute such precision values would entail running the annotator on a labeled document collection and setting $\mathrm{prec}(r)$ to be the fraction of objects produced by rule r that indeed turned out to be persons. Irrespective of how such precision values are computed, we make the same assumption about candidate generation rules as we did in Assumption 4 for meta-rules.

For example, if o is an object of type Person produced by rule r_{ua} upon examining the text "James Gray", we have $P(\mathtt{tt}(o) = \mathsf{Person}) = \mathrm{prec}(r_{ua})$. Thus, for annotators with discard-only consolidation rules, knowing the precision values for generation rules is sufficient to assign probabilities to the result annotations. However, more complex base annotators use a mixture of *merge rules* and *discard rules* to perform consolidation. The task of computing annotation probabilities for such annotators is significantly more complicated.

To illustrate, let us consider a more sophisticated person annotator ComplexPerson that uses an additional dictionary D_s containing salutations (such as Dr., Mr., Prof., etc.). Besides the generation and consolidation rules of SimplePerson, the extended ComplexPerson annotator uses a generation rule r_s and a merge rule r_m where:

Rule r_s: generate a candidate annotation if there is a pair of features in the document text that are adjacent to each other and are such that the first one is labeled with D_s and the second one is labeled with either D_u or D_a (e.g., "Dr. Michael" and "Dr. Stonebraker" would both be matched by this rule).

Rule r_m: given two candidate o_1 and o_2 such that o_1.span overlaps with o_2.span, produce a new candidate object by merging the two spans into a single larger span[5]

[5]To fully specify ComplexPerson, we must now specify an order in which the consolidation rules r_m and r_d are executed. However, these details are omitted as they are not relevant for the discussion in this section.

For instance, given the piece of text "Dr. Michael Stonebraker", rule r_s would produce object o_1 corresponding to "Dr. Michael", rule r_{uu} would produce object o_2 corresponding to "Michael Stonebraker", and rule r_m would merge o_1 and o_2 to produce o_3 corresponding to "Dr. Michael Stonebraker". From our earlier assumption, we know that $P(\mathtt{tt}(o_1) = \textsf{Person}) = \mathrm{prec}(r_s)$ and $P(\mathtt{tt}(o_2) = \textsf{Person}) = \mathrm{prec}(r_{uu})$. However, to assign a probability to object o_3, we need a meaningful probability model to compute $P(\mathtt{tt}(o_3) = \textsf{Person})$, given the above two probabilities. In general, designing a probability model to handle consolidation rules that involve merging of candidates to produce new annotation objects is an open problem. Today, in **AVATAR**, without such a model, we are forced to manually tune base annotators until we obtain desired precision levels.

4 Derived Annotators

Based on our experiences with **AVATAR** IES on various applications such as email, IBM intranet and Internet blogs, the real power is in the extraction of domain-specific derived annotations which are usually very large in number. As part of our attempts to make **AVATAR** IES more efficient we try to describe the rules in annotator template 2.2 as queries over the **AVATAR** annotation store. Such a view opens the door to exploit appropriate indices, evaluate alternate plans and in general perform cost-based optimization of *Derived* annotators.[6] We begin by describing some of our current challenges in mapping rules to queries. We then briefly discuss several issues that arise in computing precision of *Derived* annotations in the context of probabilistic databases.

4.1 Rules as Queries

In this section, we give examples that demonstrate that candidate generation and consolidation rules can be expressed as queries (using the standard Object Query Language (OQL) [5] syntax). For ease of exposition, we only consider discard rules in the consolidation step.

As our first example, the candidate generation rule r_{seq} for the PersonPhone annotator can be written as shown below:

Query q_1**.**

```
CREATE   P as person, Ph as phone, span as SpanMerge(P.span, CP.span, Ph.span), doc as P.doc
FROM     Person P, ContactPattern CP, Phone Ph
WHERE    ImmSucc(P.span,CP.span) AND ImmSucc(CP.span,Ph.span) AND P.doc = CP.doc AND CP.doc = Ph.doc
```

In the above, ImmSucc(span1,span2) is a user-defined function that returns true if "span2" occurs immediately after "span1" in the original document text. SpanMerge is another function that produces a new span by concatenating the set of spans provided as input. Note that we use a CREATE statement to indicate that a new annotation object is produced that contains the features P and Ph as attributes "person" and "phone" respectively. Similarly, rule r_d can be written as:

Query q_2**.**

```
Candidates -   (SELECT O2
                FROM Candidates O1, Candidates O2
                WHERE SpanContains(O1.span, O2.span))
```

where SpanContains(span1,span2) is a user-defined function that returns true if "span1" contains "span2". Notice how the recursive application of rule r_d is captured using set difference.

4.2 Challenges in Modeling Complex Rules as Queries

While the example candidate generation rules presented above map easily to queries - **AVATAR** IES uses several more complex rules for which the mapping is not obvious. We present an example below from our suite of rule-based annotators to extract reviews from blogs - specifically, MusicReview annotator that identifies an

[6]Note that such queries can also be used to represent base annotator rules but the benefit is most significant for derived annotators.

informal music review from a blog. A simplistic version of this annotator works as follows. $\mathcal{A}_d$ consists of *MusicDescription* phrases identified by an earlier annotator (e.g., "lead singer sang well", "Danny Sigelman played drums"). The MusicReview annotator identifies contiguous occurrences of MusicDescription (based on some proximity condition across successive entries), groups them into blocks and marks each block as a MusicReview.

We now present the candidate generation and consolidation rules for a simplistic version of this annotator, which is interested in block size up to two, i.e., it identifies occurrences of the pattern MD and $\langle MD, MD\rangle$ in the document (MD is the input MusicDescription annotations).

$R_g = \{r_{md}, r_{2md}\}$ is a pair of rules that identifies occurrences of MD and $\langle MD, MD\rangle$ respectively. The corresponding queries are given below.

Query q_3**.**

CREATE	MD as first, MD as second
FROM	MusicDescription MD

Query q_4**.**

CREATE	MD_1 as first, MD_2 as second
FROM	MusicDescription MD_1, MusicDescription MD_2
WHERE	Window(MD_1.span, MD_2.span, m) AND BeginsBefore(MD_1.span, MD_2.span)

where Window(span1,span2,m) is a user-defined function that returns true if the two spans are within a distance of m, and BeginBefore(span1, span2) returns true if "span1" begins before "span2".

There is a single consolidation rule r_{d1} given below. This rule discards candidates that are completely contained in some other candidate, or have another MusicDescription inside them.

Query q_5**.**

Candidates - (SELECT O_2
FROM *Candidates* O_1, *Candidates* O_2
WHERE BeginsBefore(O_1.first.span, O_2.first.span) AND EndsAfter(O_1.second.span, O_2.second.span)
UNION
SELECT O_2
FROM *Candidates* O_1, *Candidates* O_2
WHERE BeginsBefore(O_1.first.span,O_2.first.span) AND O_1.second = O_2.second
UNION
SELECT O_2
FROM *Candidates* O_1, *Candidates* O_2
WHERE O_1.first = O_2.first and EndsAfter(O_1.second.span, O_2.second.span))

For larger block sizes, note that the both the rules become significantly more complex. The following challenges arise in modeling the rules as queries: (i) ability to express proximity conditions across objects based on their location in the document (e.g., identify two MD's appearing adjacent to each other within m characters and no other MD between them) (ii) ability to group multiple such objects together, where each pair satisfies some proximity conditions and (iii) ability to retain groups in decreasing order of size.

4.3 Computing Precision of Derived Annotations

The precision of derived annotation objects is given by Proposition 6 in Section 1. In the expression given in Proposition 6, the precision of a derived annotation is the product of the rule-precision and the precision of the input objects. The computation of the rule precision for derived annotators is similar to that for base annotators, therefore all the issues discussed in Section 3 are relevant to derived annotators as well. We now turn to computing the second term involving the precision of the input objects. Without any additional information, one reasonable assumption is the following:

Assumption 8: The overall input precision of the input annotations in Proposition 6 is a product of the precisions of individual input objects and is independent of the rule R. In other words,

$$P(\{\mathtt{tt}(\widetilde{a}_i) = T_i\} \mid \widetilde{a} = R(\widetilde{a}_1, \ldots, \widetilde{a}_k), \{\mathcal{H}(a_i) = \mathcal{H}(\widetilde{a}_i)\}) = \prod_i P(\mathtt{tt}(\widetilde{a}_i) = T_i \mid \mathcal{H}(a_i) = \mathcal{H}(\widetilde{a}_i))$$

Although the above assumption allows us to account for the overall input precision, this assumption is invalid for most derived annotators. In particular, we believe that most derived annotator rules enhance our confidence in the precision of the input annotations. For example, let us revisit the PersonPhone annotator described in Section 2.2. This annotator has a candidate generation rule r_{seq} that operates on annotations generated by three base annotators, namely, Person, ContactPattern and Phone. Consider the example annotation identified by this rule shown in Figure 2. The regular expressions used in the ContactPattern annotator and the sequencing condition (ImmSucc) in the rule r_{seq} have a strong correlation with the average precision of the Person annotation. In fact, in the current implementation of this annotator in **AVATAR** IES, the precision of the Person annotations that satisfy this additional constraint is 97.5% – significantly higher than the overall precision of Person annotations under β = *medium* setting. Since Assumption 8 assumes that $P(\mathtt{tt}(Per1) = \text{Person})$ is independent of $\{\mathcal{H}(PP1)\}$ and the rule r_{seq}, this significantly lowers the computed precision for PersonPhone objects, and results in lowering the recall of the PersonPhone annotator.

The combination of mapping rules to queries and accounting for the dependencies as described above presents a significant challenge for probabilistic database systems. This is the focus of our ongoing work in this area. We believe that this requires enhancing the existing semantics for probabilistic database query evaluation, and will lead to a fresh set of open problems for efficient query evaluation under these enhanced semantics.

References

[1] H. Cunningham. Information extraction - a user guide. Technical Report CS-97-02, University of Sheffield, 1997.

[2] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[3] A. Doan, R. Ramakrishnan, F. Chen, P. DeRose, Y. Lee, R. McCann, M. Sayyadian, and W. Shen. Community information management. In *IEEE Data Engineering Bulletin*, March 2006.

[4] Enron Dataset. http://www-2.cs.cmu.edu/ enron/.

[5] O. Schadow et. al. *The Object Data Standard: ODMG 3.0*. Morgan Kauffman, 2000.

[6] D. Ferrucci and A. Lally. UIMA: An architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, June 2004.

[7] N. Fuhr and T. Roelleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[8] J. F. McCarthy and W. G. Lehnert. Using decision trees for coreference resolution. In *IJCAI*, pages 1050–1055, 1995.

[9] K. Nanda. Combining lexical, syntactic and semantic features with maximum entropy models for extracting relations. In *Proc. of the 42nd Anniversary Meeting of the Association for Computational Linguistics (ACL04)*, 2004.

Towards Activity Databases: Using Sensors and Statistical Models to Summarize People's Lives

Tanzeem Choudhury[1,2], Matthai Philipose,[1] Danny Wyatt,[2] Jonathan Lester[1,2]
[1]Intel Research Seattle, 1100 NE 45th Street (6th Floor), Seattle, WA 98105.
[2]Univ. of Washington, Seattle, WA 98195
{tanzeem.choudhury,matthai.philipose}@intel.com
danny@cs.washington.edu, jlester@ee.washington.edu

Abstract

Automated reasoning about human behavior is a central goal of artificial intelligence. In order to engage and intervene in a meaningful way, an intelligent system must be able to understand what humans are doing, their goals and intentions. Furthermore, as social animals, people's interactions with each other underlie many aspects of their lives: how they learn, how they work, how they play and how they affect the broader community. Understanding people's interactions and their social networks will play an important role in designing technology and applications that are "socially-aware". This paper introduces some of the current approaches in activity recognition which use a variety of different sensors to collect data about users' activities, and probabilistic models and relational information that are used to transform the raw sensor data into higher-level descriptions of people's behaviors and interactions. The end result of these methods is a richly structured dataset describing people's daily patterns of activities and their evolving social networks. The potential applications of such datasets include mapping patterns of information-flow within an organization, predicting the spread of disease within a community, monitoring the health and activity-levels of elderly patients as well as healthy adults, and allowing "smart environments" to respond proactively to the needs and intentions of their users.

1 Introduction

For computers to become increasingly useful and capable of independently assisting human beings, they need to be given a richer understanding of how humans behave "in the world." The more a computer knows about the environment in which its user exists, the better it will be able to respond to and meet a user's needs. Example uses of such new understanding cover a wide range of applications, from a messaging application that does not interrupt its user when she is a giving talk, to a surgical assistant application that follows a doctor's motions and suggest diagnoses and actions.

Even if a system cannot fully model a user's beliefs, desires, and intentions, it can still be useful if it can simply recognize her activities. The recognition of human activities is becoming a central component to a many of the pervasive computing usage models and applications, such as activity-aware actuation in smart environments, embedded health assessment, assistive technologies for elder-care, task monitoring and prompting in the workplace, enhancing workplace efficiency and information flow, surveillance and anomaly detection, etc.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

For these applications to be practical, the underlying activity recognition module often needs to detect a wide variety of activities (people may routinely perform dozens to hundreds of relevant activities a day, for instance) performed in many different manners, under many different environmental conditions, and across many different individuals. The particular aspects of the activity that are of interest also vary widely across applications (e.g., user motion, whom the user interacts with, task progress, object usage or space usage). Hence, robust recognition across a variety of activities and individuals and their variations has proved to be difficult to engineer.

The current methods available for tracking activities are time and resource consuming manual tasks, relying on either paid trained observers (i.e., a job coach who periodically monitors an individual performing their job or a nurse monitoring an elderly patient) or on self-reporting, namely, having people complete an activity report at the end of the day. However, these methods have significant deficiencies in cost, accuracy, scope, coverage, and obtrusiveness. Paid observers such as job coaches and nurses must typically split their time among several clients at different locations. Also, extensive observation causes fatigue in observers and resentment in those being observed; in addition the constant involvement of humans makes the process very expensive. Self-reporting is often inaccurate and of limited usefulness due to patient forgetfulness and both unintentional and intentional misreporting, such as a patient reporting more fitness activities than they actually completed.

An automatic activity recognition system would help not only to reduce the errors that arise from self-reporting and sparse observational sampling, but also to improve the quality of service that coaches and caregivers can provide, as they would spend less of their time performing bookkeeping duties. In addition, unobtrusive monitoring enables people to go about their daily lives in an unimpeded manner, while providing their caregivers with a more accurate assessment of their real life activities, rather than of a small sample. An accurate automated system does has another clear benefit over existing methods such as surveys, in that it provides a continuous activity log along with times and durations for a wide range of activities.

Activity recognition is also an important component for modeling group-level behavior and social dynamics. Large businesses have long been interested in the flow of information within their organization, as the difference between success and bankruptcy can depend on how well information flows between different groups of employees. Although people heavily rely on email, telephone and other virtual means of communication, highly complex information is primarily exchanged through face-to-face interactions [1]. An understanding of these face-to-face interactions and the social networks in which they take place would enable businesses to determine bottlenecks and breakdowns in communication before they become serious problems. Another real-world problem in which social networks play a central role is the spread of disease. An infectious outbreak in a self-contained village community would exhibit a completely different propagation pattern than an outbreak in a busy metropolitan city. Knowing the social networks in these communities can have enormous practical benefits, from predicting the rate of propagation of a given disease to determining where it will spread to next. This information would enable doctors to curb the further spread of a disease and begin treatment of those likely to be infected, long before they might be aware of their illness. Wearable sensing combined with statistical reasoning techniques can play an important role in discovering and modeling face-to-face interactions.

2 Building an Activity Recognition System

An activity recognition system typically has three main subcomponents: (i) A low-level sensing module that gathers relevant information about activities, e.g., camera, microphone, acceleration, RFID etc. (ii) A feature-processing and feature-selection module that processes the raw sensor data into features that can help discriminate between activities. Features can be low-level information such frequency content or correlation coefficients, or higher level information such as objects detected or the number of people present in a scene. The third subcomponent, (iii), is a computational model that uses these various features to infer the activity that an individual or a group of individuals are engaged in, e.g., walking, talking, making tea, having a conversation etc.

Because human activities are complex and sensor signals have varying amounts of noise, these activity models are almost always probabilistic.

One has also to consider and specify the requirements for an activity recognition system that may determine the choice of sensors, form-factor, and the complexity of the models needed for inference. The aspects to consider are (i) functionality: whether the system will be used for logging and classifying activities (e.g., for a doctor or to understand the usage of space) or for taking an action based on inference (e.g., an application that reminds someone to take their medicine), (ii) speed: real-time inference is necessary for prompting but not necessary for logging, (iii) resolution and timescale, e.g., whether the system needs to detect number of steps a person or takes or how long a person spends at work, and (iv) accuracy: how well the inference system has to perform in order to be useful will depend on the application (e.g., a trade off might exist between allowing more false alarms but preventing more potentially harmful false negatives in medical domains). For an activity recognition system to be widely deployable and useable, it will need to support queries that are meaningful to the user of the system, and to provide the user with the right level of summarization of a person's life.

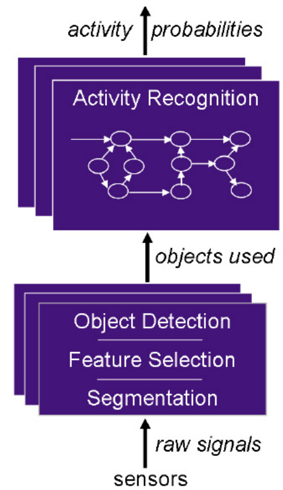

Figure 1: A typical activity recognition system.

3 Sensing

The most common sensing approach is to use a few very rich sensors (typically one per room or user) such as cameras and microphones, which can record very large quantities of data about the user and their environment. For example, originally most of the research in activity recognition was done using vision and audio sensors [2, 3]. Although in principle the data captured by these sensors should be as useful as that captured by the key human senses of sight and hearing, in practice the task of extracting features from rich low-level representations has proved to be challenging in unstructured environments [4, 5].

An increasingly popular alternative approach is to use personalized sensors (one set of sensors per user) such as accelerometers and location beacons to get precise information about a particular small set of features related to the user, such as limb-movement and user location. The majority of research using wearable devices has concentrated on using multiple sensors of a single modality, typically accelerometers on several locations on the body [6, 7]. The placement of sensors in multiple pre-defined locations can be quite obtrusive and is one of the limitations of such an approach. As a result, a single sensing device that can be integrated into existing mobile platforms, such as a cell phone, would be more appealing to users and is likely to garner greater user acceptance. In our work, we have shown that incorporating multiple sensor modalities (e.g., accelerometer, audio, light, barometric pressure, humidity, temperature, and compass) will offset the information lost by using a single sensing device. Furthermore, multiple modalities will be better suited to record the rich perceptual cues that are present in the environment, cues that a single modality often fails to capture.

Recent advancements in miniaturization and wireless communication have seen the emergence of a third approach to sensing that may be termed *dense* sensing. In this approach, sensors are directly attached to many objects of interest. These sensors are either battery-free wireless stickers called Radio Frequency Identification (RFID) tags [8] or small wireless sensor nodes powered by batteries [9]. The sensors transmit to ambient readers the usage of the objects they are attached to by detecting either motion or hand-proximity to the object. Since each sensor has a unique identifier, fixed metadata about the object (such as its color, weight or even ownership), which would conventionally have to be discerned by sensors, can be easily associated in a directly machine-readable way with the object. The reliable sensing of detailed object-use that is enabled by dense sensing has several advantages: (i) for many day-to-day activities, the objects used serve as a good indicator of the activity being performed (ii) objects used remain fairly invariant across the different manners of performing these activities (iii) since the sensors detect the features quite well regardless of most environmental conditions, activity recognition can be robust to changes in the environment or individual. Finally, knowing the class of objects being used can serve as a powerful cue to constrain the search space of possible activities.

4 Feature Extraction

For an automated system to recognize people's behavior accurately, the choice of features is critical. The usefulness of certain features will depend on the application and the activities that need to be inferred. For example, frequency information from acceleration is important in determining activities such as walking, running and related gait. The periodicity of the auditory signal is useful in determining speech and whether someone is talking or not. The overall visual shape of objects appearing in an image can be used to detect the presence of a person. Some of the features may be deterministic transformations of the raw sensor data (e.g., frequency content), while others can be a probability measure (e.g., the likelihood that an image contains a human shaped blob or likelihood of a person being in a certain location). The time-scale at which features are computed also impacts recognition, e.g., human speech is usually analyzed at millisecond resolution whereas a variety of physical activity models use features computed at 0.1 to 10Hz, and contextual information about behavior is often computed over minutes or even hours.

It is conceivable that in the near future many people will be logging information about their activities and interactions continuously, for a variety of different purposes. The need for generating reliable databases that store features and support various types of queries over time, space and other sensor attributes will be increasingly important. Example queries may be of the form, "*get audio features from all the people who were in the computer science building at time t*" or "*get camera information from a specific location when there are more than 5 people present with at least 80% certainty*."

5 Models

The two main approaches that are used for classification in machine learning are: (i) generative techniques that model the underlying joint probability distribution P(X,Y) of the classes/activities (Y) and features (X), e.g., Naïve Bayesian models, Hidden Markov models, Dynamic Bayesian networks etc. and (ii) discriminative techniques that focus on learning the class boundaries [10] or only the class posterior probability P(Y|X), e.g., support vector machines, logistic regression and conditional random fields. Both of these approaches have been used extensively for recognizing various human behaviors and activities. Although discriminative techniques sometimes outperform generative approaches in classification tasks, generative models are necessary for synthesis (e.g., if a robot has to perform an instance of an activity), in anomaly detection, or in circumstances where the discovery of the underlying process is a goal. Another recently developed class of models, called probabilistic relational models and relational Markov networks, incorporate relational structure within the probabilistic framework [11, 12]. In relational models, the properties of a certain entity can depend probabilistically on the properties of other entities (e.g., a person's role can depend on the roles of other individuals in his social network as well as his own attributes). As a result these models have been successfully applied to activity recognition and social network modeling tasks. Another dimension along which techniques vary is the manner in which the models are learned. A conventional approach is to label traces of sensor data collected during the performance of a set of activities one wants to recognize, and use the labeled examples to learn the structure and parameters of the model; this is referred to as supervised learning. The other approach is unsupervised, where the underlying structure and associated parameters are learned automatically given only the sensor traces[13]. Semi-supervised techniques use sparsely labeled data to seed the parameters of unsupervised models [14].

No matter what kinds of model and learning techniques are used, sensor data has significant variability across instances and individuals, and it is nearly impossible for automated activity classification systems to recognize every event with absolute certainty. Thus, most of the time classifiers are probabilistic or have confidence values associated with them, especially when the activities being modeled are complex or the number of activities being recognized is large. Consequently, activity databases will also need to support probabilistic entries and queries. For example, a typical query may be, "*what is the expected time spent walking for person A on a weekday?*."

6 Sensing and Modeling Activities at Different Granularities

Human activities naturally fall into two categories: (i) activities that an individual does by himself (e.g., brushing teeth) and (ii) activities that he engages in with others (e.g., conversation). When it comes to probabilistic representation, joint activities require explicit modeling of the relationships between individuals and how they affect each other. Another aspect of activities that influences the choice of models is the time-scale, we currently break them into (a) short time-scale activities, where the pattern or regularity in the sensor data is present within a short time window (on the order of seconds, e.g., walking) and (b) long time-scale activities which have regularities at a longer time window (on the order of minutes and hours, e.g., attending a meeting). For short-time scale activities, static models are often sufficient (i.e., no temporal constraints), whereas for longer time-scale events temporal models are usually required. Most approaches to constructing models suffer from what may be termed as the *model completeness* problem: models have observations that are either missing or that have inappropriate probabilities. Incomplete models can, of course, result in faulty inference. For example a model for making tea may have probabilities of various objects being used, e.g., "kettle", "teabag", teacup" and "sugar", but may mention neither "coffee cup" nor "honey". Similarly, given the inconvenience of generating labeled examples of all (or most) possible ways to execute an activity, it is likely that probabilities associated with certain observations will be under-represented. Below we give a brief overview of the work done by our group in the following areas: (i) representation of individual-level activities using multi-modal sensing (ii) automated approaches to handling model incompleteness (iii) multi-person sensing and modeling of group interactions.

Representation of individual-level activities: Advances in the development of multi-modal wearable sensors enable us to gather rich datasets of human activities. However, the problem of automatically identifying the most useful features for modeling such activities remains largely unsolved. We have developed a discriminative approach based on boosting [15] to select the most useful features and learn a weighted set of static classifiers (decision stumps) that can be additively combined to recognize different activities. During training we provide a set of labeled examples to the system, which it uses to learn the most discriminative features and the model parameters. A trained system will then use the selected features to output a probability score that a given data point or data sequence is generated by a specific activity. To capture the temporal smoothness of activities, a first-order probabilistic Markov-model (a hidden Markov model, or HMM [16]) is trained using the output of the static classifiers, where the observation sequence of the HMM consists of the probabilities from the static classification step. This combination lever-

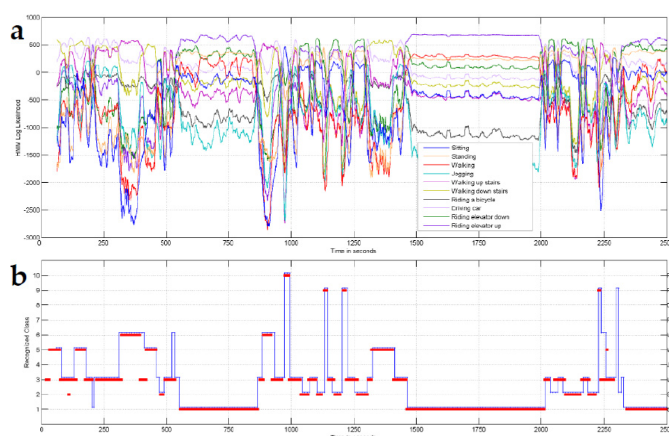

Figure 2: For a forty minute segment of data (a) the likelihood of the different activities over time and (b) the final output of the activity classification system in blue (based on the class that has maximum likelihood) and the ground truth in red.

ages the good discriminative qualities of the decision stumps with the temporal smoothness of the HMM. By automatically inferring the features that were most useful, we discovered that two modalities in particular (out of seven) yielded the most discriminative information for our activities: the audio and accelerometer sensors. These two modalities provide complementary information about the environment and the wearer. The audio captures the sounds produced during the various activities, whereas the accelerometer data is sensitive to the movement of the body. In addition, other sensors yielded more specific types of information for certain activities, such as the barometric pressure sensor being sensitive enough to detect the activity of riding in an elevator[17, 18].

Automated approaches to handling model incompleteness: Learning from data requires labeling, and since the amount of data is large it is impractical to expect an appreciable portion of it to be labeled. However, although activities are varied and idiosyncratic, they have common features that most people recognize, i.e., they have a generic "common sense" aspect that often suffices to recognize them. Furthermore, many daily activities are performed using objects that can be easily recognized if they have RFID tags on them. We have developed techniques for mining from the web simple but useful discriminative models of numerous object-based activities, which can be applied to segment and label object-use traces (thereby avoiding the need for hand-labeling). These segments can then be used to effectively bootstrap the learning of better model parameters for activities.

Given a set of activities A, we mine from the web a set of objects O used for each activity a in A and their associated usage probabilities $p(o \in O \mid a \in A)$. The mining process proceeds in four distinct steps. First, for each activity in A, we identify web pages that describe that activity being performed. Second, having identified these pages, we extract phrases from them that describe the objects used during the performance of the activity. Third, once the set of pages and phrases have been found, co-occurrence statistics for the pages and phrases are used to estimate the object-use probabilities. Finally, we use the mined information to assemble a Hidden Markov Model (HMM) capable of recognizing activities in traces of object data; the hidden states of the HMM correspond to the various activities, and the observation probabilities of the HMM are the object-use probabilities. Now, given a set E of unlabeled traces (a trace is a sequence of sensed objects), we use the mined models as a basis for learning an improved or more customized model. To train this customized model from the generic mined model, we first apply the most probable labeling for the traces E (using the Viterbi algorithm [16]) given the model. We then re-estimate the model parameters according to the labeled trace. If certain parts of the model are not observed then their parameters are not changed, and remain set to the mined probabilities [8].

The use of objects as the underlying features being modeled suggests another simple approach to countering models with missing information. Intuitively, we can exploit common-sense information about which objects are functionally similar. If the model ascribes very different probabilities to two very similar objects, we can "smooth" these probabilities into more similar values. As a degenerate case, if the model omits an object while incorporating very similar ones, we can postulate that the omitted object is likely to be observed in the model. We have developed a completely unsupervised approach to realizing this idea. By using auxiliary information, called an ontology, about the functional similarities between objects, we mitigate the problem of model incompleteness. The similarity information is extracted automatically from WordNet, a large, relevant ontology of lexical reference system for the English language, and incorporated into our models by using a statistical smoothing technique, called shrinkage [19].

Multi-person sensing and modeling of group interactions: The structure and dynamics of face-to-face social networks are of critical importance to many social phenomena, ranging from organizational efficiency to the spread of knowledge and disease. Research in face-to-face networks has an abundance of interesting and important questions, but has been faced with a paucity of data rich enough to answer many of these questions. We believe better models of social network and organizational dynamics will facilitate efficient means of collaboration and information propagation. Virtually all of the datasets are collected manually by human observers or via surveys, which are very labor intensive and yield only a small number of observations, sparsely spread over time. In our work, we have demonstrated the feasibility of learning social interactions from raw sensor data. We collected a large repository of wearable sensor data that includes auditory features (raw audio signal is not stored for privacy reasons) for several hours everyday over several weeks from multiple individuals (more than twenty people). We have developed a framework for automatic modeling of face-to-face interactions, starting from data processing and going all the way up to capturing the structure and dynamics of social networks, by analyzing whom we talk to and how we talk to them. The micro-level inter-relationship between individuals is modeled via a coupled probabilistic model of turn-taking during conversations. The coupled model allows us to estimate how much influence an individual has in the overall turn-taking that occurs during conversations. Furthermore, we were able to show how this measure of "influence" correlates significantly with betweenness centrality [20], an independent measure of an individual's importance in a social network. This result suggests that micro-level measures such as conversational influence can be predictive of more macro-level social influence [21, 22].

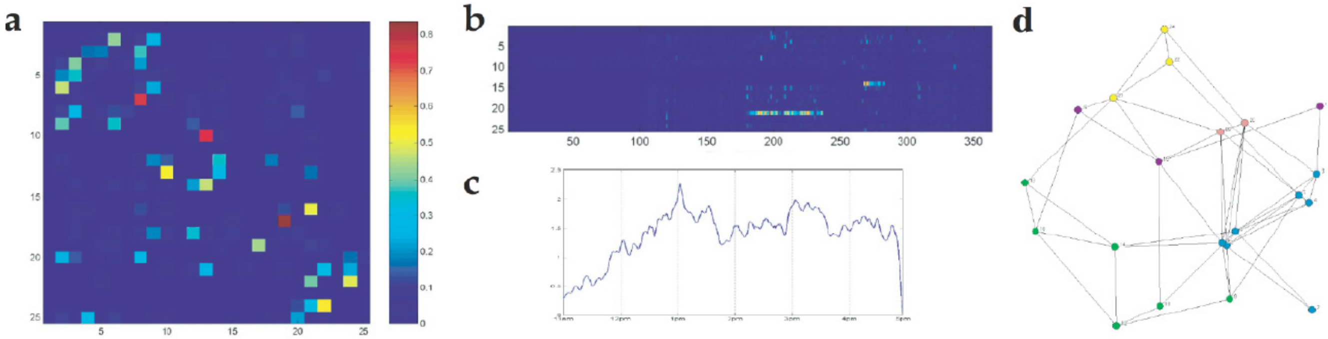

Figure 3: Some information of interest about social networks. (a) Interaction matrix, I, of a face-to-face network. Each row corresponds to a different person. I(i,j) is the fraction of person i's total interaction with person j. (b) A given person's interaction likelihood with other people in the network. The x-axis is time in minutes (6 hours) and the y-axis numbers are the IDs of people in the network. (c) Speech activity over the course of the day, averaged over all subjects. (d) Interaction network diagram, based on multi-dimensional scaling of geodesic distances. Node numbers represent the subject IDs.

7 Conclusion

This paper provides a brief introduction to the sensing and statistical reasoning techniques used in building systems that reason about human activities and interactions. The approaches outlined here are common to many systems, although the illustrative examples given in the paper have been drawn mostly from our own work.

Databases for storing the output of activity inference systems need to meet several challenges. They must be able to to support a variety of activity-based queries, while also protecting raw sensor data and sensitive private information. It is important that different users can be given different levels of access privilege to specific types of query. For example, the raw sensor data should be accessible only to a minimal set of individuals, whereas a broader set of users may be able to compute deterministic features or issue probabilistic queries. Privacy protection is even more important when answers to a query require access to the data from multiple individuals (e.g., "Did A and B have a conversation today?"). Social network information or any relational data can often destroy anonymity, so queries need to support varying levels of anonymity.

An activity database will certainly be probabilistic, as both entries and answer to queries will often be probabilistic. Such databases will also need to be able to deal with sporadically missing data, and with combining data from sensors that record at varying rates — the data from real-world activity recognition systems are all too rarely uniform or complete. The statistical techniques used in activity-recognition modeling already offer potential methods for handling missing information. Such tools may be applicable in ranking queries and in dealing with inconsistent data in probabilistic databases. Finally, these databases will also need to deal with temporal queries about people's behavior. For example, a query may ask not what an individual was doing at a particular instant, but instead what their pattern of behavior was over the course of a day.

Sensors are being embedded more and more into the everyday objects around us: phones and watches contain cameras, microphones and GPS, and objects in shops, factories and hospitals are being tagged with RFID. Powerful computer processors are being incorporated into previously "dumb" consumer products. However, such technology will do little to improve usability if it is not sensitive to people's needs, and these needs vary as a function of the activities that people are engaged in. We therefore believe that the need for activity recognition, and for the management and retrieval of information about activities, will present important new research challenges for a long time to come.

References

1. Allen, T., Architecture and communication among product development engineers. 1997, Sloan School of Management, MIT: Cambridge. p. 1-35.
2. Pentland, A., *Smart Rooms.* Scientific American, 1996. **274**(4): p. 68-76.
3. Gavrila, G., *The visual analysis of human movement: A survey.* Computer Vision & Image Understanding, 1999, **75**(1).
4. Moore, D.J., I. Essa, and M.H. Hayes. Exploiting object context for recognition tasks. In *Proceedings of the Conference on Computer Vision*. 1999: IEEE Press.
5. Davis, J.W. and A.F. Bobick. The representation and recognition of action using temporal templates. In the *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1997: IEEE Press.
6. Bao, L. and S. Intille. Activity recognition from user-annotated acceleration data. In *Pervasive* 2004.
7. Kern, N., B. Schiele, and A. Schmidt. Multi-sensor activity context detection for wearable computing. in the *Proc. EUSAI, LNCS*. 2003. Eindhoven, The Netherlands.
8. Wyatt, D., M. Philipose, and T. Choudhury. Unsupervised activity recognition using automatically Mined common sense. In *the Proc. of Twentieth National Conference on Artificial Intelligence (AAAI)*. 2005.
9. Munguia-Tapia, E., et al., MITes: Wireless portable sensors for studying behavior. In the *Proceedings of Extended Abstracts Ubicomp 2004: Ubiquitous Computing*. 2004: Vienna, Austria.
10. Rubinstein, Y.D. and T. Hastie. Discriminative vs. informative learning. In *the Proceedings of Knowledge Discovery and Data Mining*. 1997.
11. Getoor, L., et al., *Learning probabilistic models of relational structure.* Journal of Machine Learning Research, 2002. **3**: p. 679-707.
12. Taskar, B., P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Uncertainty in Artificial Intelligence (UAI)*. 2002.
13. Duda, R., P. Hart, and D. Stork, *Pattern Classification*. 2000: Wiley-Interscience.
14. Zhu, J., Semi-supervised learning literature survey. CS TR 1530, Univ. of Wisconsin-Madison, 2006.
15. Schapire, R.E. A brief introduction to boosting. In *16th Intl. Joint Conf. on Artificial Intelligence*. 1999.
16. Rabiner, L., A tutorial on hidden Markov models and selected applications in speech recognition. In the *Proceedings of IEEE*, 1989. **77**(2): p. 257-286.
17. Lester, J., Choudhury, T., et al. A hybrid discriminative-generative approach for modeling human activities. In *the Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*. 2005.
18. Lester, J., T. Choudhury, and G. Borriello. A practical approach to recognizing physical activities. In *Pervasive 2006*. Dublin, Ireland.
19. Munguia-Tapia, E., T. Choudhury, and M. Philipose. Building reliable activity models using hierarchical shrinkage and mined ontology. In *Pervasive 2006*. Dublin, Ireland.
20. Freeman, L.C., *A set of measures of centrality based on betweenness.* Sociometry, 1977. **40**: p. 35-41.
21. Choudhury, T., Sensing and Modeling Human Networks, *Doctoral Thesis*. 2003, MIT, Cambridge, MA.
22. Choudhury, T. and S. Basu. Modeling Conversational Dynamics as a Mixed Memory Markov Process.In *Neural Information Processing Systems (NIPS)*. 2004. Vancouver, BC.

Probabilistic Data Management for Pervasive Computing: The *Data Furnace* Project

Minos Garofalakis[†], Kurt P. Brown[†], Michael J. Franklin[*], Joseph M. Hellerstein[*], Daisy Zhe Wang[*]
Eirinaios Michelakis[*], Liviu Tancau[*], Eugene Wu[*], Shawn R. Jeffery[*], Ryan Aipperspach[*]

[†]Intel Research Berkeley and [*]University of California, Berkeley

Abstract

The wide deployment of wireless sensor and RFID (Radio Frequency IDentification) devices is one of the key enablers for next-generation pervasive computing applications, including large-scale environmental monitoring and control, context-aware computing, and "smart digital homes". Sensory readings are inherently unreliable and typically exhibit strong temporal and spatial correlations (within and across different sensing devices); effective reasoning over such unreliable streams introduces a host of new data management challenges. The Data Furnace *project at Intel Research and UC-Berkeley aims to build a probabilistic data management infrastructure for pervasive computing environments that handles the uncertain nature of such data as a first-class citizen through a principled framework grounded in probabilistic models and inference techniques.*

1 Introduction

Pervasive Computing is an area that has seen significant interest since it was first identified by the late Mark Weiser over a decade ago, with research contributions from a breadth of computer science disciplines including distributed and mobile systems, machine learning, and human-computer interaction. The broad challenge in pervasive computing is the creation of environments that embed computation and communication in a way that organically interacts with humans to enhance or ease their daily behavior. One typical scenario for pervasive computing is in the design of "smart digital homes", which are instrumented to observe, learn, and facilitate the typical behaviors of occupants. As one concrete example, smart homes can automate control of utilities like lighting, heating and cooling, with the goal of minimizing energy usage without adversely impacting occupant comfort.

Recent advances in distributed sensing and wireless communication enable pervasive applications that are quite information-rich, capturing and utilizing large numbers of potentially high-bandwidth streams of data. This raises a number of interesting research opportunities at the nexus of database management, stream query processing, sensor networks, machine learning, and human-computer interaction. We highlight some of the challenges of data-intensive pervasive computing in what follows.

• *Diverse Data Sources.* A wide variety of sensors can be reasonably deployed in pervasive applications, from richly semantic, high-bandwidth sensors like video cameras, to simple boolean sensors like door-ajar switches, with a variety of sensing modalities in between (audio, motion, temperature, etc.).

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

• *The Realities of Sensory Data.* Data from sensors is different from what is typically seen in enterprise databases. First, sensor data is typically a very noisy, *uncertain* representation of the phenomena it is intended to capture, due to issues like miscalibration and sensitivity to environmental factors. This can make sensor data hard to "trust". On the other hand, many physical phenomena represented by sensor data can exhibit significant spatial and temporal correlation, and this structure in the data can be exploited both for efficiency and for better understanding the true properties of the underlying physical phenomena.
• *Streams and Storage.* Pervasive applications typically have a "real-time" nature, since they are intended to interact with human behaviors. On the other hand, in order for pervasive applications to build good temporal models of human behavior and use those models for prediction, they may need to track significant volumes of archival data. Hence, the underlying data-management infrastructure needs to support both rich, continuous queries/triggers over streams, as well as offline mining/learning of useful patterns from archival data.
• *Integration of Probabilities and Logic.* Traditional interfaces and languages for data management have strong roots in logic. By contrast, pervasive applications that deal with sensory data most often operate with probabilistic methods, which must reason about uncertainty for a variety of reasons: to model the true phenomena generating the sensor readings, to model future states of the environment, to predict human desires or behaviors, and so on. As is well known in the database literature, data-intensive applications perform better by pushing computation into data access, rather than extracting data and shipping it to computations. Hence, the uncertain, probabilistic nature of the data needs to be a first-class citizen across all layers of the data-management system: the data model (both logical and physical), the query processing techniques, triggering engines, pattern detection, etc.
• *Complex Modeling.* There is often a significant semantic gap between sensed data (e.g., a sequence of plumbing usage, power consumption, and certain sound signatures) and meaningful human activities (e.g., "making a cup of tea"). This gap needs to be bridged by a modeling framework that can capture specifications of high-level "complex" events in terms of lower-level sensed events. These specifications may be generated manually, automatically, or semi-automatically, but in any case there needs to be a modeling framework to capture them. Moreover, this modeling framework has to interact meaningfully with the uncertainty and correlations present in the base sensory data, as well as with other, more conventional data sources.

The *Data Furnace*. In our work at Intel Research and UC Berkeley, we are embarking on an effort to develop information-rich infrastructure for pervasive computing, suitable to applications like the Digital Home. In that spirit, we refer to our system as a *Data Furnace*: a low-maintenance, low-profile, but critical piece of infrastructure that provides general-purpose data management infrastructure for pervasive computing applications. Briefly, the *Data Furnace* aims to manage data uncertainty as a first-class citizen through a principled probabilistic framework, and provide a uniform, declarative means for higher-level applications to store, query, and learn from such probabilistic data. In this paper, we outline some of the unique research challenges that arise in the context of the *Data Furnace*, and discuss some of the basic ideas underlying our approach.

2 A Motivating Application Scenario: The "Smart Home"

Harper [9] defines the term "smart home" as "a residence equipped with computing and information technology which anticipates and responds to the needs of the occupants, working to promote their comfort, convenience, security and entertainment through the management of technology within the home and connections to the world beyond". The recent revolutions in personal digital media and sensing technologies has rendered such "futuristic" visions much more credible, and the home is currently one of the most frequently targeted markets for these new technologies.

Data management plays a critical role in the smart-home vision. Overwhelming amounts of data float around in our home today, such as music, movies, contact lists, calendars, photos, financial data, news articles, web pages, e-mail messages, and so on. The smart home of the future will only exacerbate the situation with the addition of many more devices and systems, including hundreds of sensors (of different modalities) and RFID readers/tags for sensing several different physical phenomena and activities. Through such sensory (input) devices as well as additional control (output) mechanisms, smart homes will be able to support *richer, real-time interactions* with their users; for instance, energy management and demand-response applications (`http://dr.berkeley.edu`) can employ motion sensors to track context information for Alice and appropriately actuate electric devices like lighting and water heating. A different application might require correlating sensory readings and actuation with more conventional information sources in the home or even historical patterns of user behavior; for example, a security subsystem sensing motion patterns and activities that do not match any known historical patterns for the home users, might choose to take some precautionary action (e.g., notify Bob or start video-recording the "suspect" individual).

Allowing future smart homes to support such rich user-interaction models over streaming, uncertain sensor data (of various modalities) raises a number of novel data-management challenges. Providing persistent storage and querying (that, of course, satisfies all other traditional database requirements of availability, consistency, security/privacy, etc.) for diverse, heterogeneous data-types has received attention recently through different research projects, such as MyLifeBits [8]. Still, as our discussion above indicates, this is only part of the equation: data-management infrastructures for the smart home will also have to effectively combine and correlate large streams of low-level, uncertain sensory data (possibly, in real time) to accurately track and react to higher-level events and activities. We use motivating examples from the smart-home setting in the remainder of this paper.

3 The *Data Furnace* System: Architecture and Challenges

We envision our *Data Furnace* system as the centerpiece of the data-centric architecture for pervasive applications. The *Data Furnace* serves as the central repository for application data and metadata, and offers a number of diverse services at different layers of abstraction (ranging from device and web connectivity, to data archiving, to pattern learning and probabilistic reasoning). The high-level logical architecture of the *Data Furnace* is depicted in Figure 1. In a nutshell, the *Data Furnace* architecture comprises three broad layers: (1) The *Hardware Layer* manages physical system resources, such as processing, storage, and communication. (2) The *Metadata Layer* serves as the repository for environment metadata, including, for instance, "schema" information for the application context (e.g., home architecture, floorplans, wiring and pipe layouts), data on the users and (possibly) their routines, as well as information on devices, events, and API definitions; this layer defines the basic *Data Furnace* interface with the physical world and higher-level applications. (3) The *Service Layer* is the heart of the *Data Furnace* engine, essentially providing the key information-management functionality for our target application scenarios, including query processing and optimization, data archiving, complex event processing, pattern and model learning, probabilistic reasoning, and so on.

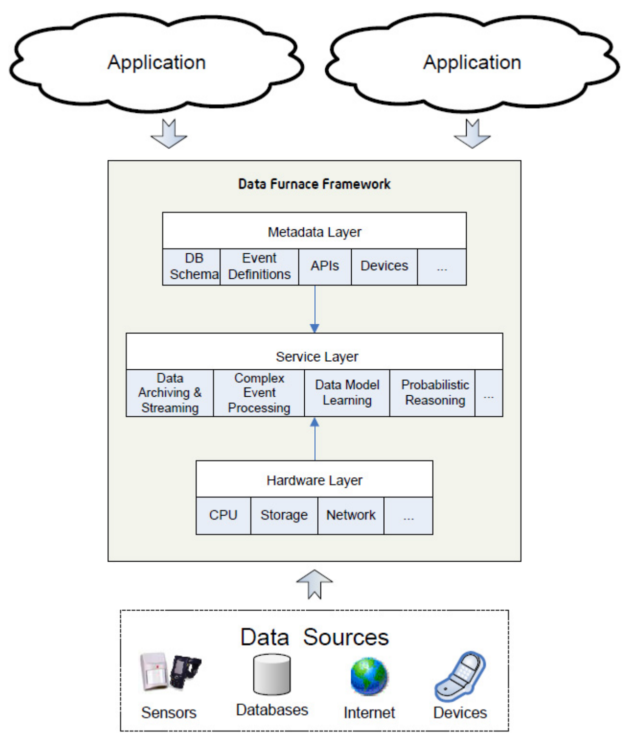

Figure 1: The *Data Furnace* Architecture.

Given the inherently uncertain nature of several of its key data sources, as well as the complexity of human behavioral patterns and automated recognition of higher-level activities, it is clear that uncertain data and probabilistic information will play a central role in our *Data Furnace* system. We now outline some of the key probabilistic data management challenges for the *Data Furnace* engine, and briefly discuss some of the basic ideas underlying our approach. (Pervasive computing environments such as the smart home also raise several other challenging database and system-design issues with respect to multi-device integration, availability, privacy, self-management, longevity, unobtrusiveness/invisibility, and so on; while acknowledging their importance, we do not focus on these concerns in this short paper.)

Voluminous Streams of Uncertain, Low-Level, Correlated Sensory Data. Raw readings from sensing and RFID devices are inherently unreliable and subject to various physical-world phenomena, such as battery-power fluctuations, noise, and interference. As a result, the streams of readings emanating from such devices are *uncertain* (to various degrees), with several missing and/or inaccurate values. For instance, the observed read rates (i.e., percentage of tags in a reader's vicinity that are actually reported) in real-world RFID deployments is often in the $60-70\%$ range [10, 12] (i.e., over 30% of the tag readings are routinely dropped); even higher drop rates are possible depending on environmental characteristics (e.g., in the presence of metal [5]).

Sensor and RFID readings also tend to exhibit strong *correlation patterns*, a fact that has already been exploited in the area of acquisitional query processing in sensornet deployments [1, 4]. Such correlation patterns include both (1) *spatial correlations*, where sensing devices in close physical proximity capture the same physical event or highly-correlated readings (for instance, temperature and light readings in the same room); and, (2) *temporal correlations*, where correlated sequences of events are often sensed together (for instance, turning on the stove is typically followed by turning on the hood fan). *Causality* relationships are also quite common in sensor data streams; for example, detecting a person at the door can be causally connected to the event "Bob returning home from work".

Due to their highly unreliable nature, raw, low-level readings streams are often useless for the purposes of providing useful knowledge to higher-level applications (e.g., accurate inventory or people tracking). Instead, such applications are interested in more concise, *semantically-rich events* that can be *inferred* (with some level of *probabilistic confidence*) from the base sensing data. This implies that a probabilistic data-management framework is needed in the *Data Furnace* engine. And, of course, capturing the (possibly, complex) correlation and causality patterns in such data is a crucial requirement for both efficient and accurate probabilistic inference — base-tuple independence assumptions typically employed in earlier work on probabilistic database systems (e.g., [2, 3]) are rarely valid in our setting and almost certainly will lead to poor probabilistic estimates for higher-level events. Continuing with our earlier example, it is clear that the conditional probability $\mathbf{Pr}\,[\text{Bob returning from work}|\text{person at the door}]$ is likely to be very different from $\mathbf{Pr}\,[\text{Bob returning from work}]$ (while the two are equal assuming independence); similarly, probabilistic computations across, say, sensors in the same room, also need to account for such strong correlations. Thus, effectively capturing probabilistic (base and derived) data correlations is an important requirement for the *Data Furnace* engine.

Our basic approach here is to accurately model both the probabilistic nature of the sensor data and the underlying correlation/causality patterns by incorporating *statistical learning techniques* and *probabilistic models* [11, 13, 14] as first-class citizens in the *Data Furnace* engine. The *Data Furnace* can learn such models either incrementally (over the input data streams) or off-line (from archived data), and employ probabilistic-inference methods to efficiently query these models at run-time. Model learning is a computationally-intensive process [11], so maintaining our probabilistic models over continuous streams from possibly hundreds of sensory streams poses difficult technical challenges; we intend to exploit semantic knowledge (such as floorplan and sensor-layout information) to enable scalable incremental learning.

Definition and Real-Time Tracking of Complex, Hierarchical Probabilistic Events. Effective real-time monitoring and management of the target pervasive-computing environment (e.g., smart home or supply chain) is an important requirement, and mechanisms for composing and tracking complex events (in real time) can address this need. Such events typically require correlating (e.g., through joins or aggregation) multiple uncertain sensing streams under intricate notions of time, space, ordering, and Boolean connections.

In addition to the uncertain nature of *Data Furnace*'s sensor inputs, in most real-life scenarios, meaningful definitions of high-level, semantically-rich events over sensor readings are inherently *probabilistic* in nature. For instance, a "making tea" event might be connected with "stove usage" 70% of the time and "microwave usage" 30% of the time. In general, such events can be thought of as *"occurrences of significance"* defined in a *hierarchical manner* using appropriate (probabilistic) *rules* for composing several streams of real-time sensing data, and perhaps other (lower-level) events. Detecting such an occurrence (with some appropriate level of confidence) can, in turn, trigger higher-level events or appropriate actions. As an example, detecting Bob in the living room and room-temperature readings below 60^oF with confidence, say, above 95% can lead to the system automatically turning on the heat in the living room; similarly, stationary readings from Alice in the TV room over a window of time combined with a "TV on" event can trigger a higher-level event "Alice is watching TV", with some level of confidence.

Abstractly, at any point in time, the state of the system can be seen as a *probability distribution* over possible states and high-level events, and the *Data Furnace* engine needs to support effective mechanisms for (1) defining hierarchies of complex probabilistic events that are of interest to users/applications, and (2) accurately tracking such events (and corresponding confidences) in real time over numerous sensor and RFID streams. Obviously, accurate probabilistic estimations are needed to avoid erroneous inferences that can potentially result in event false positives/negatives and improper actions; for instance, the environment might overreact in response to low-confidence sensed user activity, thus defeating the purpose of *calm computing*. And, again, principled techniques for modeling existing correlation/causality patterns within and across base data and events are needed for correct probabilistic reasoning. Semantically-rich probabilistic events also provide a fairly clean, natural abstraction for higher-level applications that wish to employ *Data Furnace* services (Figure 1), without worrying about the details and uncertainties of raw sensor readings. Our initial design of the *Probabilistic Complex Event Triggering (PCET)* subsystem (Section 4) addresses several of these challenges.

Efficient Querying and Learning over both Probabilistic and Deterministic Information. Uncertain and probabilistic information needs to co-exist in the *Data Furnace* engine with more traditional (relational and non-relational) data. Event and query processing techniques must be able to effectively reason with both types of data and provide reasonable performance to higher-level applications. For instance, both complex event tracking and ad-hoc queries over streams from smart-home sensors and RFID readers may require direct correlation with home metadata, such as floorplans and users' daily schedules. In addition to continuous monitoring and ad-hoc query processing, it is also important to effectively *mine* the information collected from both probabilistic and conventional data sources for longer-term trends and patterns at different time and/or space granularities; for example, learning users' daily or weekly routines is critical for effective energy management or detecting "suspicious" behavior in the smart home.

Effective querying and mining of uncertain, probabilistic data (perhaps in combination with other, deterministic information) raises a host of new challenges for the *Data Furnace*. Efficiency, in particular, is an important concern, given the recent negative results of Dalvi and Suciu for general query processing over probabilistic data tuples [2]. We believe that, through the effective use of rich probabilistic models (which can, in a sense, be seen as concise approximations of the *possible-worlds distribution* [2, 3]), the *Data Furnace* query processor can avoid such inherently intractable problems. The *granularity of the probabilistic information* is a key parameter here — while earlier probabilistic-database work that focuses on integration and/or lineage problems [2, 3, 15] connects probabilities with individual tuples or even attribute values, it is not clear that such fine-grain information is needed in our settings. For instance, it may be possible to associate probabilistic noise or dropped-readings models for individual sensing devices, which can essentially be tied to all readings (tuples) from a given physical device. This leads to more concise probabilistic representations and, hence, more efficient (albeit, approximate) techniques for probabilistic query processing and event tracking through model inference. Of course, for sizeable data collections and complex probabilistic models incorporating notions of time and space, the *Data Furnace* needs to support efficient methods for *approximate probabilistic inference* (e.g., particle filtering [14]) and *approximate query processing* (e.g., histograms, wavelets, sketches [6, 7]). Designing appropriate *query and data-definition languages* for such rich, diverse data collections also raises several interesting research challenges.

System Support for Managing, Maintaining, and Reasoning over a Multitude of Probabilistic Models. Managing large collections of probabilistic models (perhaps built for different inferencing tasks) is a crucial requirement for our *Data Furnace* architecture. Supporting a variety of such models as first-class citizens brings up a number of interesting systems issues, including appropriate declarative interfaces and data structures (e.g., indexes) for probabilistic-model maintenance and inference (querying). *Data Furnace* models can also be viewed as *probabilistic views* built over the base data, and can be supported (as either virtual or materialized structures) to model the state of the pervasive-computing environment at potentially different levels of abstraction (e.g., at the sensor-readings layer or at the user-activities layer). As with traditional view materialization, the key tradeoff lies in the view maintenance cost vs. its query-processing benefits; of course, the probabilistic nature of our model views makes the problem somewhat unique. Similarly, multiple inferencing tasks over probabilistic models can possibly share processing costs, leading to interesting (probabilistic) multi-query optimization issues. This is only a small sample of the interesting optimization questions arising in the query/event-processing engine of the *Data Furnace*.

4 The Probabilistic Complex Event Triggering (*PCET*) Subsystem

We are currently building a first incarnation of the *PCET* subsystem of the *Data Furnace*, which aims to provide a general probabilistic event triggering service over uncertain sensory data. In this section, we briefly touch upon some of the key elements of the *PCET* logical architecture (Figure 2).

The Probabilistic Inference Engine (PIE) essentially employs statistical learning techniques to build, maintain, and run inferences over probabilistic models that capture correlations across base-level events (i.e., individual sensor readings), as discussed in Section 3. For our initial implementation, we plan to use parametric (e.g., Gaussian) models to model sensor noise and a variant of *dynamic Bayesian networks* [14] as our basic probabilistic model; we also plan to explore ways of capturing and exploiting *spatial information* (e.g., home floorplans) in *PCET*'s learning and modeling tools. *PCET*'s Application Layer Interface employs *probabilistic events*, perhaps with associated *confidence thresholds* (e.g., Bob is in the living room with probability $\geq 90\%$), as the key abstraction for communicating with higher-level applications. Through this simple interface, applications can hierarchically define and subscribe to new complex events based on either base-level events (e.g., sensor readings exceeding a certain value) or other event-tracking "services" already offered within or on top of *PCET*. For instance, basic smart-home services, such as a "People Tracker" that tracks the location of individual users in the home can be provided as part of the *Data Furnace* distribution with an easy-to-use customization GUI; other, higher-level applications like a "Personalized Environment Controller" can be built using both base sensor readings and "People Tracker" events.

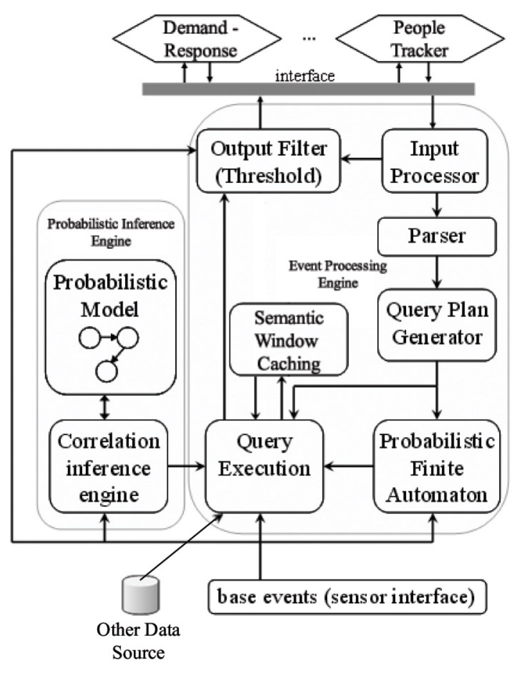

Figure 2: The *PCET* Subsystem.

PCET supports the definition of new complex events through a small, yet expressive composite-event algebra that includes: sequencing $(e_1; e_2)$, conjunction $(e_1 \& e_2)$, disjunction $(e_1|e_2)$, negation $(!e)$, and temporal $(\{e\}t)$ as well as location $(e@loc)$ constraints. Of course, events at different levels of abstraction can be defined by referencing previously-defined events, as shown in the following simple example:

```
LightOn := LightSensor1='ON'
LightSwitchedOn := !LightOn ; LightOn
PersonEntry := {Motion@Outside ; DoorOpen ; DoorClose ; Motion@Hall}10s
Cooking := {StoveOn | MicrowaveOn}10m
```

PCET's Event Processing Engine (EPE) is responsible for (1) compiling event definitions into self-contained execution plans that are continuously run over the input data streams; and, (2) possibly tracking the current "state" of the environment (based on observed events) using what we term a Probabilistic Finite Automaton representation. Naturally, EPE employs the probabilistic inference services provided by PIE to ensure that probabilistic beliefs are correctly propagated through the composite-event definitions.

5 Conclusions

We have discussed some of the unique probabilistic data management challenges arising in pervasive-computing environments, and provided a quick overview of our proposed approach for the *Data Furnace* system, which aims to provide a general data-management infrastructure for pervasive applications. Our current research agenda includes building a first prototype of the probabilistic-event tracking subsystem and exploring different alternatives and algorithms for capturing and querying probabilistic information in the *Data Furnace*.

References

[1] David Chu, Amol Deshpande, Joseph M. Hellerstein, and Wei Hong. "Approximate Data Collection in Sensor Networks using Probabilistic Models". In *ICDE*, 2006.

[2] Nilesh Dalvi and Dan Suciu. "Efficient Query Evaluation on Probabilistic Databases". In *VLDB*, 2004.

[3] Nilesh Dalvi and Dan Suciu. "Answering Queries from Statistics and Probabilistic Views". In *VLDB*, 2005.

[4] Amol Deshpande, Carlos Guestrin, Samuel R. Madden, Joseph M. Hellerstein, and Wei Hong. "Model-Driven Data Acquisition in Sensor Networks". In *VLDB*, 2004.

[5] Kenneth P. Fishkin, Bing Jiang, Matthai Philipose, and Sumit Roy. "I sense a disturbance in the force: Unobtrusive detection of interactions with RFID-tagged objects". In *UbiComp*, 2004.

[6] Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. "Querying and Mining Data Streams: You Only Get One Look". Tutorial in *VLDB*, 2002.

[7] Minos Garofalakis and Phillip B. Gibbons. "Approximate Query Processing: Taming the Terabytes". Tutorial in *VLDB*, 2001.

[8] Jim Gemmel, Gordon Bell, and Roger Lueder. "MyLifeBits: A Personal Database for Everything". *Comm. of the ACM*, 49(1), 2006.

[9] Richard Harper. *"Inside the Smart Home"*. Springer, 2000.

[10] Shawn R. Jeffery, Gustavo Alonso, Michael J. Franklin, Wei Hong, and Jennifer Widom. "A Pipelined Framework for Online Cleaning of Sensor Data Streams". In *ICDE*, 2006.

[11] Michael I. Jordan. *"An Introduction to Probabilistic Graphical Models"*. (Book, in preparation), 2006.

[12] Laurie Sullivan. "RFID Implementation Challenges Persist, All This Time Later". *Information Week*, October 2005.

[13] Judea Pearl. *"Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference"*. Morgan Kaufmann Publishers, 1988.

[14] Sumit Sanghai, Pedro Domingos, and Daniel Weld. "Relational Dynamic Bayesian Networks". *Journal of AI Research*, 24:1–39, 2005.

[15] Jennifer Widom. "Trio: A System for Integrated Management of Data, Accuracy, and Lineage". In *CIDR*, 2005.

Community Information Management

AnHai Doan[1], Raghu Ramakrishnan[2], Fei Chen[2], Pedro DeRose[1],
Yoonkyong Lee[1], Robert McCann[1], Mayssam Sayyadian[1], Warren Shen[1]
[1] University of Illinois, [2] University of Wisconsin

Abstract

We introduce CIMple*, a joint project between the University of Illinois and the University of Wisconsin.* CIMple *aims to develop a software platform that can be rapidly deployed and customized to manage data-rich online communities. We first describe the envisioned working of such a software platform and our prototype,* DBLife*, which is a community portal being developed for the database research community. We then describe the technical challenges in* CIMple *and our solution approach. Finally, we discuss managing uncertainty and provenance, a crucial task in making our software platform practical.*

1 Introduction

There are many *communities* on the Web, each focusing on a specific set of topics. Examples of communities based on common interests include communities of movie goers, football fans, database researchers, and bioinformaticians. Other common examples include communities with a shared purpose, such as organization intranets and online technical support groups. Community members often want to query, monitor, and discover information about various *entities* and *relationships* in the community. For example, database researchers might be interested in questions such as these:

- Is there any interesting connection between two researchers X and Y (e.g., sharing same advisor)?
- In which course is this paper cited?
- Find all citations of this paper in the past one week on the Web.
- What is new in the past 24 hours in the database research community?

Answering such questions often requires retrieving the raw, largely unstructured data from multiple disparate sources (e.g., home pages, conferences, DBLP, mailing lists), then inferring and monitoring semantic information from the data. Examples of such inference and monitoring include recognizing entity mentions (e.g., "J. Gray", "SIGMOD-04"), deciding if two mentions (e.g., "J. Gray" and "Jim Gray") refer to the same real-world entity, recognizing that a certain relationship (e.g., co-authoring, advising, giving a talk) exists between two entities, detecting that a new entity (e.g., workshop) has appeared, and inferring that a current relationship (e.g., affiliation with a university) has ceased to exist.

The above inference and monitoring tasks are recognized as being challenging [10, 14, 29]. As communities proliferate, the problem of developing effective solutions to support their information needs is becoming increasingly important. We call this problem *community information management*, or *CIM* for short.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

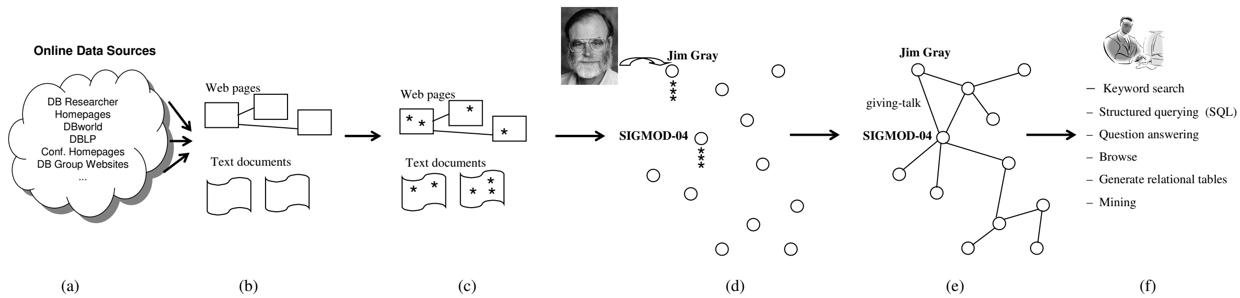

Figure 1: How CIMple infers a semantic entity-relationship graph from the raw unstructured data of the database research community, then leverages the inferred graph to provide a host of user services.

In this paper we describe CIMple[1], a joint project between the University of Illinois and the University of Wisconsin, that addresses the CIM problem. Our goal is to develop a software platform that a specific online community can quickly deploy and customize to effectively manage its data. CIMple can be valuable for communities in a broad range of domains, ranging from scientific data management, government agencies, and enterprise intranets, to those on the World-Wide Web.

We now briefly explain the envisioned working of CIMple, using the community of database researchers as an example (see Figure 1). First, a community expert provides CIMple with a set of relevant data sources (e.g., home pages of database researchers, DBworld mailing list, conference pages, etc.; see Figure 1.a), domain knowledge about entities and relationships of interest, and possibly hints on extracting relevant mentions from the listed data sources. CIMple then crawls the sources at regular intervals to obtain data pages (Figure 1.b), then marks up mentions of relevant entities (denoted by * in Figure 1.c). Examples of mentions include people names (e.g., "J. Gray", "James N. Gray"), conference names, and paper titles. Next, CIMple matches mentions and groups them into entities (Figure 1.d). CIMple then discovers relationships among the entities, in effect transforming the raw data into a semantic entity-relation (ER) data graph (Figure 1.e). CIMple then provides a broad variety of user services over the ER data graph, including browsing, keyword search, structured querying, summarization, and mining. It also maintains and tracks the ER graph over time, as the underlying raw data evolves. Such temporal tracking allows CIMple to provide interesting notification services. Finally, CIMple employs a novel approach called *mass collaboration* to evolve and maintain the inferred ER graph and the provided services. Specifically, it leverages the entire user community, by providing carefully crafted functionalities that are valuable to users, then learning from users' interactions to identify and address a broad range of problems.

Developing the CIMple platform is a long-term goal that we are working toward. As a concrete first step, we are building a prototype community of the kind that CIMple is eventually intended to support. The prototype system, called DBLife, is aimed at the database research community, and the features being developed include monitoring and reporting of interesting events, such as "SIGMOD-06 has posted the list of accepted papers," "researcher X is giving an invited talk at institution Y," and "student S has graduated and moved to department D." To this end, each day we crawl a set of data sources in the database community (e.g., researcher homepages, group pages, conferences, etc.; currently we have collected 1,158 such sources and are adding more). We retrieve Web pages from the sources (on average 11 pages per source, for 20+ MB of data per day), and parse the data to mark up mentions of interesting entities (e.g., researchers, publications, conferences), using hand-crafted rules. We then match the mentions, and group matching mentions into entities. For example, we match roughly 114,400 people mentions per day, and group them into 5,400+ entities. If we see a mention of an entity X on the seminar page of a department Y, then we can infer that X probably is giving a talk at Y. DBLife will be released as an extension of the current DBworld service from the University of Wisconsin.

In the rest of this paper, we briefly describe related work, discuss the technical challenges in developing CIMple and then consider the problem of managing uncertainty and provenance in the CIM context.

[1]The acronym stands for CIM PLatform. The final "e" is an acronym-friendly mistake that we expect our readers to catch.

Related Work: Our work is related to the wealth of research on information extraction and integration (e.g., [1, 22, 16, 4], see [10, 29, 21] for recent tutorials), mention matching, a.k.a. record linkage, entity resolution, fuzzy tuple matching (e.g., [15, 8, 34], see [14] for a brief recent survey), relationship discovery [28], uncertainty and provenance [3, 30, 35, 12, 11, 5]. Many more works exist on these topics than our limited space can list here. Many recent works have also considered the problem of inferring and exploiting semantic information on the World-Wide Web [16], the Semantic Web [33], in personal information management [15], business intelligence (with significant efforts in the AVATAR project [20] and the UIMA framework [17]), text management [19], and most recently, in building data space support platforms (DSSPs), a new unifying data management abstraction for diverse applications [18].

CIMple differs from this body of work in several important ways. First, we focus on *community* settings, where we often have significant domain knowledge about the entities and relationships involved, and hence can address the extraction and integration challenges sufficiently well to enable useful solutions. Second, we seek to build an end-to-end solution to CIM, which requires us to address several important problems that have not received much attention so far. One example is the need to maintain extracted information over time, as the underlying raw data evolves. Third, given a particular community, we want to rapidly deploy and customize our end-to-end solution. Toward this goal, we aim to make our solutions as *declarative* as possible, so that the community builder and users can rapidly enter, debug, and modify domain knowledge that guides the extraction and integration process. Fourth, we must address scalability issues to make our solution practical. To do so, we place a strong emphasis on being able to quickly compile declarative and procedural knowledge (as supplied by a builder and potentially users) into an optimized execution plan, taking cues from optimization technologies in relational contexts. Finally, we add a novel "people" aspect to CIM in particular and information management in general with our use of mass collaboration to improve and maintain data extraction and integration.

2 Technical Challenges

We now discuss four key challenges for CIM: how to extract structure, exploit structure, maintain structure, and provide effective mass collaboration. Our discussion also highlights the sources of uncertainty in CIM contexts. Section 3 then discusses managing uncertainty and the related topic of provenance in more detail.

2.1 Extracting Structure

To deploy CIMple, we propose that the community builder supply a set of seed *Web data sources*, an ER-like *semantic schema* and a set of *extractors*. Since the builder is an expert on the target community, he or she can quickly assemble a set of community data sources to serve as a seed. In the database community, for example, data sources include researcher home pages, DBworld, DBLP, conferences, project pages, etc. CIMple can "bootstrap" from the seed sources to discover other related Web sources.

The builder must also specify an ER-like *semantic schema* whose entities and relationships capture the underlying semantic structure of the domain. For example, in the database domain, entities include person, paper, and conference, and relationships include advise, co-author, and write. A person entity instance has attributes such as name, affiliation, and email. This schema can be specified in its entirety, or in a piecemeal fashion, e.g., some entities and relationships are described first, then some more are added later.

Finally, the builder supplies a set of *extractors*, each of which specifies a way to extract instances of entities or relations of the ER schema from the raw data. Many extraction techniques have been developed (in the areas of information extraction, named entity recognition, wrapper construction, and text segmentation [10, 29, 21]), and many implementations are available commercially, or publicly. The builder thus can create extractors that either directly implement the techniques or adapt off-the-shelf "blackbox" implementations to the target community.

Using the above extraction-related knowledge, CIMple crawls the specified data sources at regular intervals (e.g., daily) to retrieve data pages, then applies extractors to these pages to construct an ER graph of entity and relationship instances that conform to the given ER schema. Conceptually this graph is generated as follows. First, CIMple applies the extractors to the retrieved data pages to extract *mentions* of entities. Example mentions are "J. Gray" and "James N. Gray" for persons and "SIGMOD-04" and "The ACM SIGMOD Conf" for conferences. These mentions are often *ambiguous* in that different ones can refer to the same real-world entity, and conversely the same mention can refer to different real-world entities. CIMple therefore must disambiguate and partition the mentions such that all mentions in a partition refer to the same real-world entity. These entities, denoted as $E_1, \ldots. E_n$, form the nodes of the ER graph. In the next step, CIMple applies relation extractors to discover possible relations between the entities. If a relation R exists between two nodes E_i and E_j, then CIMple adds to the ER graph an edge that connects these nodes and corresponds to R.

As described, constructing the ER graph poses two major challenges. First, we must solve the mention disambiguation problem outlined above. Numerous solutions have been proposed to variants of this problem, but they suffer from limited accuracy in the CIM context (as we recently found out when experimenting with the prototype DBLife system), because they typically employ just a single matching technique and thus fail to exploit the varying degree of semantic ambiguity in the data. In recent work [32], we have proposed a solution to this problem, which builds on the observation that different data sources within a community often vary significantly in their level of semantic ambiguity, thus requiring different matching methods.

The second challenge in ER graph construction is that CIMple can apply the mention extractors and mention matchers in many different ways, each of which forms an *execution plan* (analogous in a sense to execution plans in relational databases). These plans often vary significantly in run time and matching accuracy. Hence, finding an optimal or near-optimal execution plan is critical. We are studying the space of such plans, and developing optimizers to find good execution plans, drawing from query optimization insights in relational databases. Our initial work on this topic is described in [32].

2.2 Exploiting Extracted Structure

Once the ER graph has been constructed, CIMple can exploit it to provide many useful services. Services that we are developing or plan to develop include:

- *Keyword search:* Given a keyword query, we return matching data pages, entity instances, relation instances, as well as matching fragments of the ER graphs. We are building on our recent work in keyword search over structured databases [31] to develop this search service.
- *Entity profiling in "Super Homepages":* We create a "super homepage" for each entity E, which displays all information CIMple gathers about that entity, including all mentions of E together with brief explanations. For example, it may say that E's name appeared recently in a conference homepage because E authored a paper in that conference, on a database group homepage because E gave an invited talk, and so on.
- *Notification:* A related functionality is to monitor mentions of an entity, and alert the user when interesting mentions appear. For example, E may want to monitor when a certain paper P will be read by a class or cited in a new paper.
- *ER graph browsing:* Users can browse the ER graph easily, in a manner similar to browsing the citation graph of CiteSeer. For example, starting with a researcher's super homepage, they can follow a paper to a journal in which the paper is published, or follow a community service, e.g., "PC member, SIGMOD-06", to a conference, and so on.
- *Community daily newsletter:* Every morning CIMple will generate an "executive summary" of what interesting events happened in the past 24 hours in the community. For example, the summary may state that X just mentioned on his or her homepage that he or she will serve on SIGMOD-07, or that the list of papers accepted to ICDE-06 has just been posted on DBworld.
- *Structured querying:* We are studying how to formulate SQL-like structured queries over the extracted ER graph, what the syntax and semantics of the results should be, and how to execute such queries efficiently. As

a first step, [24] describes our recent work on interactive SQL querying of Web data generated from structured templates (e.g., Amazon.com and DBLP pages).

• *Temporal keyword search and structured querying:* We will develop temporal query capabilities. For example, the user can pose a query over only the data crawled in the past two weeks, or ask CIMple to rank answers in chronological order. Note that this is a key advantage of CIMple over general search engines. Such search engines cover the *entire* World-Wide Web and hence cannot archive it, e.g., on a daily basis, in order to provide effective temporal querying.

• *Service modules for one-click capabilities in context:* We will allow community builders to use the full range of search, query, alert and reporting capabilities to associate stored versions (of searches, queries, etc.) with different entity and relationship types. When a page displays entities or relationship instances of these types, the associated capabilities are then made available to users. Thus, even casual users can ask sophisticated queries through a single click in context.

2.3 Maintaining Extracted Structures

After creating an ER graph and associated services for a community, we must monitor and adjust the ER graph as the underlying data sources evolve. Sources on the Web are often highly dynamic [23, 9]. Consequently, maintaining the extracted ER graph is a labor intensive undertaking, and developing techniques to reduce the maintenance cost is critical.

The first maintenance challenge is how to *efficiently update* the extracted ER graph. We can simply re-crawl the sources at regular intervals (e.g., daily), then apply the methods discussed previously to rebuild the ER graph from scratch. Indeed, we apply this solution in the current DBLife prototype. However, this solution has two major limitations (as confirmed painfully by our experience with the prototype). First, rebuilding the ER graph from scratch is quite time intensive. We must extract afresh all entity mentions, match the mentions, and rediscover the various relations. All of these tasks are time consuming [10, 29, 21]. Thus, for highly dynamic communities (e.g., auction or finance) that require updating the ER graph every few hours, this approach will probably not scale. Second, if we rebuild the ER graph from scratch at every re-crawl, we loose the temporal dimension associated with entities and relationships. Consider an entity E that we have inferred from the raw data on Day 1. If we re-crawl and rebuild the ER graph again on Day 2, which of the newly created entities correspond to E? Establishing the correspondence is crucial for tracking E over time, and for answering temporal queries. To remove these limitations, we are currently developing a solution to *incrementally* update the ER graph.

The second challenge that we must address is how to detect and repair broken extractors. Consider for example a price extractor that always returns as a price the third number in the first italics line of a page. This extractor will break if data pages change the formatting rule used to display prices. Given the central role extractors play in CIMple it is important that we detect and repair broken extractors, or adjust to them in some way when repair has not been carried out. We have developed an initial solution called Maveric for detecting broken extractors [23], and are developing a solution that attempts to repair a broken extractor or makes repair suggestions to the builder.

2.4 Mass Collaboration

We have described how a builder can use CIMple to quickly develop and deploy a first-cut data portal for a community. Next, we propose a set of novel techniques that leverages *community users*—instead of the *builder*—to help refine the extraction and integration logic used by CIMple. We consider three specific techniques: personalized data spaces, reputation incentives, and “payment” schemes.

• *Personalized data spaces:* To illustrate the idea of leveraging personalized data spaces for the common good, consider the Internet Movie Database (IMDB, at *imdb.com*). When a new movie is added to IMDB, within a few days it receives a score (say 7.6 out of 10), averaged over scores given by thousands of movie goers. How can IMDB convince so many people to score the movie? It turns out that the vast majority of scores come from *registered users*. These users maintain *private* movie collections in their IMDB accounts, i.e., personalized versions of the public IMDB movie collection. Many users then score the movies in their collections only so that they can search later, e.g., "list all action movies in my collection that I gave a score of 7 or higher", using the IMDB search engine. IMDB, however, can aggregate these private scores to provide public scores for movies. Our idea is to similarly allow users to personalize and manage private versions of the public data space in their CIMple accounts, then learn from the private actions (in a way that protects individual privacy and has their consent) to improve the public data space. We will consider a range of "personalization" actions that users can carry out in their private CIMple accounts, design simple interfaces to facilitate them, then develop techniques to leverage personalization activities to improve the public data portal.

• *Reputation incentives:* There were cases where researchers corrected mistakes in their DBLP homepages by contacting the DBLP owners, because they felt that these homepages form an important part of their public image. This is similar to the experience of many community portals; e.g., at QUIQ [26], people answered hundreds (in some cases, thousands) of technical support questions posed by other users solely to gain greater visibility in the community through a variety of mechanisms. We therefore plan to design CIMple such that users will have sufficiently strong "reputation incentives" to correct mistakes in the publicly viewable (extracted and integrated, and therefore possibly incorrect) information that they believe may adversely affect them. This raises several challenges that we plan to examine: First, how can we authenticate a user vis a vis the corresponding person entity? Second, if different users edit related data (e.g., co-authors edit the same paper), their changes may conflict. How should we reconcile the changes? Finally, how can we provide sufficiently strong incentives?

• *Payment schemes:* For certain CIM services, we can make users "pay" for using them, by answering relatively simple questions. We then leverage the answers to improve the services. For example, suppose that CIMple has compiled a query processing bibliography B, and that employing learning techniques, it has found a paper P that may be relevant to B. Then when a user U wants to access B, CIMple may show U the paper P and ask "Is P about query processing?" Once the user has answered, he or she is allowed to access B. If a sufficient number of users answer yes, then CIMple may decide to include P in bibliography B, thereby expanding B. To realize the above idea, we must address several challenges—the most important is merging multiple, often noisy, user answers into a single answer. Users may disagree and there may not even be a single correct answer. We discuss these challenges and preliminary solutions in [25, 27, 26]. The QUIQ experience [27, 26] and experiments on small user communities (10-130 users) [25] suggest the potential of the mass collaboration approach.

3 Managing Uncertainty and Provenance

We now discuss uncertainty and provenance problems in CIMple, and sketch our solutions. We highlight in particular the ideas of making CIM services interactive and leveraging mass collaboration to reduce uncertainty.

3.1 Uncertainty

All major CIM steps—extraction and integration, data evolution, and mass collaboration—generate uncertainty. Techniques for extracting mentions, tagging them (e.g., as people names, conferences, etc.), and matching them are well-known to be imperfect. In addition, the evolution of the data often invalidates the assumptions made by extractors, causing errors, thus adding yet another source of uncertainty. Finally, mass collaboration generates uncertainty since people do not contribute in a perfect fashion, and malicious or ignorant users often provide incorrect feedback.

To handle such uncertainties, we consider three directions: (a) understand the nature of the uncertainty involved, (b) model and reason using uncertain data, and (c) reduce uncertainty.

Understanding Uncertainty: In CIM contexts, we have encountered primarily two types of uncertainty: *confidence score* and *multi-value*, though additional types are certainly possible. The first type arises when an extractor (or a mention matcher) operates on domain knowledge or heuristics that are typically (though not always) true. In such cases the CIM module can make qualified predictions, such as "the mention Madison in a text document U is a last name with confidence 0.8" or "mentions Madison and J. Madison match with confidence 0.9." Most current extractors generate this type of uncertainty.

The second type of uncertainty, multi-value, arises when an extractor operates on *correct, but underspecified* domain knowledge. For instance, given a text document U and the knowledge "there is a publication year in U and it is of numeric type" (e.g., supplied by the user [24]), an extractor may extract the set of all numeric values from U, say $\{23, 4, 2001, 1998, 135\}$, and specify that publication year takes a value in this set. As another example, if values are organized into a domain hierarchy (e.g., time might be specified as days, weeks, and years), we might not always know a value at the finest granularity. Thus, we might know that a sale occurred in 2006, but not know the specific week or day. Multi-value uncertainty, specifically the form arising from domain hierarchies, is referred to as *imprecision* in [6].

Reasoning with Uncertainty: We are developing methods to provide services such as keyword search and SQL querying over extracted uncertain data. We have studied confidence-score uncertainty in the context of keyword search over multiple disparate and heterogeneous structured data sets. Given a user query Q, the goal is to return a ranked list of answers, where each answer "glues" together a set of data fragments taken from the same or different data sets [31]. Since the structured data sets are heterogeneous, matching mentions (e.g., "D. Smith" vs. "David Smith") and matching schema elements (e.g., pname vs. researcher-name) can be predicted only with some confidence score. Our solution is to incorporate such scores directly into the overall score of each answer for the keyword query. This work suggests that IR-style ranking can be a very natural tool for handling multiple types of uncertainties.

For the second type of uncertainty (i.e., multi-value), we have developed a solution to provide SQL querying over extracted data with multi-value uncertainty [24]. Our solution provides a *superset semantics*, i.e., it always produces a superset of the correct results. A key idea underlying the solution is that it is *interactive*: it allows the user to quickly pose SQL queries, obtain initial results, then iterate to get increasingly better results. In each iteration, the system asks the user 1-2 relatively simple questions, designed to solicit structural information to *reduce* the uncertainty associated with multiple values. It then leverages the user answers to refine the query results.

In joint work with the Avatar project at IBM [20], we have also developed a principled approach to defining semantics for OLAP queries over imprecise data [6] based on *allocation* of imprecise facts, and are developing efficient implementation techniques for allocation [7].

Reducing Uncertainty: We reduce uncertainty via two mechanisms: user interaction and mass collaboration. For many CIM services (e.g., keyword search, SQL querying), we consider how to make them interactive, so that the service can learn from the user, and provide increasingly better results (in a sense this strategy can be viewed as a variant of relevance user feedback, see [31, 24]). Our preliminary work on this topic in the context of keyword search is described in [31], and an interactive solution for SQL querying over multi-valued data is described in [24].

In CIMple, we also apply mass collaboration to reduce uncertainty, a solution that, to the best of our knowledge, has not been considered in the context of data extraction or integration. As discussed in Section 2.4, we will develop this solution in two steps. First, we allow each user to take extracted data provided by automatic solutions as a starting point, then manually correct the data or interact with CIMple to improve the data. Second, we develop solutions to learn from what each user is doing, and apply the results to help other users.

3.2 Provenance

In CIM contexts, we found that provenance serves two goals. First, it helps the user understand and reduce uncertainty. Second, it provides a way for the user to express certain information needs using provenance-related criteria (e.g., find only answers that originate from sources X and Y).

Toward the first goal, we are designing CIMple to provide a *justification* for any answer it produces in response to a user query. The justification lists all data pages that contribute to the answer, and all operations (extraction, mention matching, etc.) that have been invoked along the path from the raw data to the answer. Thus, in effect we provide a "derivation (a.k.a., provenance) tree" that explains how the answer was produced. The user can follow the derivation tree to the original data pages for further verification. We also plan to develop a "what-if" facility. When examining the derivation tree, the user can ask hypothetical questions, such as "What if I state that these two mentions do not match?" The idea is that CIMple will show how the answers look if the proposed assumption holds. Such a facility would help evaluate the robustness of the original answers, thereby increasing user confidence in those answers. We have developed justification mechanisms previously for schema matching [13] and logic program inference [2]. We are building upon this work, as well as results on managing provenance (e.g., [3, 12, 5]) to develop justification and "what-if" mechanisms for CIMple. Toward the second goal of serving provenance-related information needs, we will adopt results from current work on languages for querying provenance (e.g., [3, 5]) to the CIM context, on an "as-needed" basis.

4 Concluding Remarks

We have introduced the CIMple project that develops a software platform to extract and integrate information for online communities. In general, information extraction is taking on an increasingly larger role in how we seek to organize and analyze text corpora. By its nature, extracted information has associated uncertainty, and the CIM setting offers many novel opportunities and challenges for dealing with this uncertainty, as we have tried to illustrate in this brief overview of CIMple.

References

[1] A. Arasu and H. Garcia-Molina. Extracting structured data from Web pages. In *SIGMOD-03*.

[2] T. Arora, R. Ramakrishnan, W. Roth, P. Seshadri, and D. Srivastava. Explaining program execution in deductive systems. In *Int. Conf. on Deductive and Object-Oriented Databases*, 1993.

[3] O. Benjelloun, A. Das Sarma, C. Hayworth, and J. Widom. An introduction to ULDBs and the Trio system. *IEEE Data Engineering Bulletin, Special Issue on Probabilistic Databases*, 29(1), 2006.

[4] L. Bertossi, J. Chomicki, P. Godfrey, P. Kolaitis, A. Thomo, and C. Zuzarte. Exchange, integration, and consistency of data: Report on the ARISE/NISR workshop. *SIGMOD Record*, 34(3):87–90, 2005.

[5] P. Buneman, S. Khanna, K. Tajima, and W. Tan. Archiving scientific data. *ACM Trans. on Database Systems*, 29:2–42, 2004.

[6] D. Burdick, P. Deshpande, T. Jayram, R. Ramakrishnan, and S. Vaithyanathan. OLAP over uncertain and imprecise data. In *VLDB-05*.

[7] D. Burdick, P. Deshpande, T. Jayram, R. Ramakrishnan, and S. Vaithyanathan. Efficient allocation algorithms for OLAP over imprecise data. *IBM Almaden Tech. Report*, 2006.

[8] S. Chaudhuri, V. Ganti, and R. Motwani. Robust identification of fuzzy duplicates. In *ICDE-05*.

[9] W. Cohen. Some practical observations on integration of Web information. In *WebDB-99*.

[10] W. Cohen. Information extraction. *Tutorial, www.cs.cmu.edu/∼wcohen/ie-survey.ppt*, 2003.

[11] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB-04*.

[12] N. Dalvi and D. Suciu. Answering queries from statistics and probabilistic views. In *VLDB-05*.

[13] R. Dhamankar, Y. Lee, A. Doan, P. Domingos, and A. Halevy. iMAP: Learning complex matches between database schemas. In *SIGMOD-04*.

[14] A. Doan and A. Halevy. Semantic integration research in the database community: A brief survey. *AI Magazine, Special Issue on Semantic Integration*, Spring 2005.

[15] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD-05*.

[16] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Web-scale information extraction in KnowItAll. In *WWW-04*.

[17] D. Ferrucci and A. Lally. Building an example application with the unstructured information management architecture. *IBM Systems Journal 43, No. 3, 455-475*, 2004.

[18] A. Halevy, M. Franklin, and D. Maier. Principles of dataspace systems (invited paper). In *PODS-06*.

[19] P. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano. To search or to crawl? Towards a query optimizer for text-centric tasks. In *SIGMOD-06*.

[20] R. Krishnamurthy, S. Raghavan, J. Thathachar, S. Vaithyanathan, and H. Zhu. AVATAR information extraction system. *IEEE Data Engineering Bulletin, Special Issue on Probabilistic Databases*, 29(1), 2006.

[21] A. Laender, B. Ribeiro-Neto, A. da Silva, and J. Teixeira. A brief survey of Web data extraction tools. *SIGMOD Record*, 31(2), 2002.

[22] I. Mansuri and S. Sarawagi. A system for integrating unstructured data into relational databases. In *ICDE-06*.

[23] R. McCann, B. AlShelbi, Q. Le, H. Nguyen, L. Vu, and A. Doan. Mapping maintenance for data integration systems. In *VLDB-05*.

[24] R. McCann, P. DeRose, A. Doan, and R. Ramakrishnan. SLIC: On-the-fly extraction and integration of Web data. *UW-Madison Tech. Report TR-1558, available as http://anhai.cs.uiuc.edu/public/slic-tr06.pdf*, 2006.

[25] R. McCann, A. Doan, A. Kramnik, and V. Varadarajan. Building data integration systems via mass collaboration. In *WebDB-03*.

[26] R. Ramakrishnan. Mass collaboration and data mining, 2001. Keynote address, KDD-01, www.cs.wisc.edu/ raghu/Kddrev.ppt.

[27] R. Ramakrishnan, A. Baptist, V. Ercegovac, M. Hanselman, N. Kabra, A. Marathe, and U. Shaft. Mass collaboration: A case study. In *IDEAS-04*.

[28] D. Roth and W. Yih. Probabilistic reasoning for entity and relation recognition. In *Int. Conf. on Computational Linguistics (COLING)*, 2002.

[29] S. Sarawagi. Graphical models for structure extraction and information integration. *Keynote talk and tutorial at ICDM-05, www.it.iitb.ac.in/ sunita/talks/graphical2.ppt*, 2005.

[30] A. Das Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE-06*.

[31] M. Sayyadian, A. Doan, and L. Gravano. Keyword search across heterogeneous relational databases. In *UIUC Tech. Report 03-2006-02, available as http://anhai.cs.uiuc.edu/public/kite-tr06.pdf*, 2006.

[32] W. Shen, P. DeRose, L. Vu, A. Doan, and R. Ramakrishnan. Source-aware entity matching: A compositional approach. *UW-Madison Tech. Report TR-1559, available as http://anhai.cs.uiuc.edu/public/soccer-tr06.pdf*, 2006.

[33] Frank van Harmelen. *A Semantic Web Primer*. MIT Press, 2004.

[34] M. Weis and F. Naumann. Dogmatix tracks down duplicates in XML. In *SIGMOD-05*.

[35] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR-05*.

CALL FOR PAPERS

2007 IEEE 23rd International Conference on Data Engineering (ICDE 2007)

April 16- 20, 2007

The Marmara Hotel, İstanbul, Turkey

http://www.srdc.metu.edu.tr/webpage/icde/

Sponsored by the IEEE Computer Society

Data Engineering deals with the use of engineering techniques and methodologies in the design, development and assessment of information systems for different computing platforms and application environments. The 23rd International Conference on Data Engineering provides a premier forum for sharing and exchanging research and engineering results to problems encountered in today's information society.

We especially encourage submissions that make efforts (1) to expose practitioners to how the research results and tools can contribute to their everyday problems in practice and to provide them with an early opportunity to evaluate these tools; (2) to raise awareness in the research community of the problems of practical applications of data engineering and to promote the exchange of data engineering technologies and experience among researchers and practitioners; (3) to identify new issues and directions for future research and development in the data engineering field.

AWARDS

An award will be given to the best paper. A separate award will be given to the best student paper. Papers eligible for this award must have a (graduate or undergraduate) student listed as the first and contact author, and the majority of the authors must be students. Such submissions must be marked as student papers at the time of submission.

INDUSTRIAL PROGRAM

Industrial track submissions should also be prepared in the 8/5"x11" IEEE camera-ready format, with a 12-page limit, and submitted electronically via the industrial track submissions website.

PANELS

Panel proposals must include an abstract, an outline of the panel format, and relevant information about the proposed panelists. Send your proposals electronically by the submission deadline to the Panel Chair, Meral Özsoyoğlu at <meral[AT]case[.]edu>.

DEMONSTRATIONS

Demonstration proposals should focus on new technology advances in applying databases or new techniques. Proposals must be no more than four double-columned pages, and should give a short description of the demonstrated system, explain what is going to be demonstrated, and state the significance of the contribution to database technology, applications or techniques. Send your proposals electronically to the Demonstration Chairs, Jose Blakeley and Mario Nascimento at <joseb[AT]microsoft[.]com> and <mn[AT]cs[.]ualberta[.]ca>.

ADVANCED TECHNOLOGY SEMINARS

Seminar proposals must include an abstract, an outline, a description of the target audience, duration (1.5 or 3 hours), and a short bio of the presenter(s). Send your proposals electronically to the Seminar Chairs, Karl Aberer and Gültekin Özsoyoğlu at <karl[.]aberer[AT]epfl[.]ch> and <tekin[AT]eecs[.]cwru[.]edu>.

WORKSHOPS

We invite proposals for workshops to be held in conjunction with ICDE 2007. We solicit proposals for workshops related to the conference topics. Proposals for workshops should stress how they intend to provide more insight into the proposed topics with respect to the main conference. The scope definition of the proposal should encompass only some of the topics of the main conference. Workshop proposals should be submitted to the workshop chairs, Ahmed Elmagarmid and Fred Lochovsky (at <ake[AT]cs[.]purdue[.]edu> and <fred[AT]cs[.]ust[.]hk>) with the title, a description of the topic, and the names of key organizers such as program chair(s). Prior contact with the workshop chairs is encouraged. The organizer(s) of approved workshops will be responsible for advertising their workshops, and for the reviewing process for their workshop's papers, and for the collection of camera-ready copies of accepted papers.

SUBMISSION INFORMATION

Research papers must be prepared in the 8/5"x11" IEEE camera-ready format, with a 12-page limit, and submitted electronically at http://www.srdc.metu.edu.tr/webpage/icde/. All accepted papers will appear in the Proceedings published by the IEEE Computer Society.

IMPORTANT DATES

Abstract Deadline: July 5, 2006
Paper submission Deadline: July 12, 2006
Panel submission deadline: July 26, 2006
Demo submission deadline: July 26, 2006
Seminar submission deadline: July 26, 2006
Workshop proposal submission deadline: July 26, 2006
Industrial paper submission deadline: July 12, 2006
Notification: October 7, 2006

GENERAL CHAIRS
Ling Liu, Georgia Institute of Technology, USA
Adnan Yazıcı, Middle East Technical University, Turkey

PROGRAM CHAIRS
Asuman Dogac, Middle East Technical University, Turkey
Tamer Özsu, University of Waterloo, Canada
Timos Sellis, National Technical University of Athens, Greece

PUBLICITY CHAIR
İbrahim Körpeoğlu, Bilkent University, Turkey

TREASURER
I. Hakki Toroslu, Middle East Technical University, Turkey

INDUSTRIAL PROGRAM CHAIRS
Fatma Özcan, IBM Almaden, USA
Patrick Valduriez, INRIA, France

PANEL CHAIR
Meral Özsoyoğlu, Case Western Reserve University, USA

SEMINAR CHAIRS
Karl Aberer, EPFL Lausanne, Switzerland
Gültekin Özsoyoğlu, Case Western Reserve University, USA

WORKSHOP CHAIRS
Ahmed Elmagarmid, Purdue University, USA
Fred Lochovsky, HKUST, China

DEMONSTRATION CHAIRS
Jose Blakeley, Microsoft Corporation, USA
Mario Nascimento, University of Alberta, Canada

LOCAL ARRANGEMENTS
Mehmet Altınel, IBM Almaden, USA
Zehra Çataltepe, İstanbul Technical University, Turkey
Ahmet Coşar, METU, Turkey
Yıldıray Kabak, METU, Turkey
Gökçe Banu Laleci, METU, Turkey
Şule G. Öğüdücü, Istanbul Technical University, Turkey
Ahmet Saçan, METU, Turkey
Yücel Saygın, Sabancı University, Turkey
Pınar K. Şenkul, METU, Turkey
Nesime Tatbul, Brown University, USA
Pınar Yolum, Boğaziçi University, Turkey

DEMONSTRATION PROGRAM COMMITTEE MEMBERS
Walid Aref, Purdue University, USA
Malu Castellanos, HP Labs, USA
Anand Deshpande, Persistent Systems, India
Ralf Hartmut Güting, University of Hagen, Germany
Bettina Kemme, McGill University, Canada
Nick Koudas, University of Toronto, Canada
Bill McKenna, NCR/Teradata, USA
Sergey Melnik, Microsoft, USA
Glenn Paulley, Sybase, Canada
Kian-Lee Tan, NUS, Singapore
Caetano Traina Jr, USP, Brazil
Kyu-Young Whang, KAIST, Korea

INDUSTRIAL COMMITTEE MEMBERS
Kevin Beyer, IBM Almaden, USA
Mike Carey, BEA, USA
Daniela Florescu, Oracle, USA
Marcus Fontoura, Yahoo Research, USA
Piero Fraternali, Politecnico di Milano, Italy
Stéphane Gançarski, University Paris 6, France
James Hamilton, Microsoft, USA
Mary Holstege, Mark Logic, USA
Francois Llirbat, Business Objects, France
Qiong Luo, HKUST, China
Jussi Myllmaki, Google, USA
Marta Patino-Martinez, Technical University of Madrid, Spain
Mikael Ronstrom, MySQL AB, Sweden
Oded Shmueli, Technion, Israel
Vishal Sikka, SAP, USA

ICDE 2007 invites research submissions on all topics related to data engineering, including but not limited to those listed below:

1. Distributed, parallel, P2P and mobile DB's
2. Web Data Management, Web Information Systems
3. Data Warehousing, OLAP, and Statistical Databases
4. Mining Data, Text, and the Web
5. Query processing (standard and adaptive) and query optimization
6. Middleware, workflow, and Web services
7. Security and Privacy
8. Database applications and experiences
9. Stream processing, continuous queries, and sensor DB's
10. Indexing, access methods, data structures
11. Semi-structured data and XML
12. Scientific and Biological DB's and Bioinformatics
13. Temporal, Spatial and Multimedia DB's
14. Data Integration, Interoperability, and Metadata
15. Imprecision, Uncertainty and fuzziness in databases

AREA PC VICE-CHAIRS
Özgür Ulusoy, Bilkent University, Turkey
Juliana Freire, University of Utah, USA
Joachim Hammer, University of Florida, USA
Kyuseok Shim, Seoul National University, Korea
Tore Risch, Uppsala University, Sweden
Krithi Ramamritham, IIT Bombay, India
Marianne Winslett, University of Illinois, USA
Masaru Kitsuregawa, University of Tokyo, Japan
Mitch Cherniack, Brandeis University, USA
Yannis Manolopoulos, Aristotle University, Greece
Frank W. Tompa, University of Waterloo, Canada
Carol Goble, The University of Manchester, UK
Nihan Çicekli, Middle East Technical University, Turkey
Renée J. Miller, University of Toronto, Canada
Gabriella Pasi, Università degli Studi di Milano Bicocca, Italy

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398