Data Integration with Uncertainty

Xin Dong
University of Washington
Seattle, WA 98195
lunadong@cs.washington.edu

Alon Y. Halevy
Google Inc.
Mountain View, CA 94043
halevy@google.com

Cong Yu
University of Michigan
Ann Arbor, MI 48109
congy@eecs.umich.edu

ABSTRACT

This paper reports our first set of results on managing uncertainty in data integration. We posit that data-integration systems need to handle uncertainty at three levels, and do so in a principled fashion. First, the semantic mappings between the data sources and the mediated schema may be approximate because there may be too many of them to be created and maintained or because in some domains (e.g., bioinformatics) it is not clear what the mappings should be. Second, queries to the system may be posed with keywords rather than in a structured form. Third, the data from the sources may be extracted using information extraction techniques and so may yield imprecise data.

As a first step to building such a system, we introduce the concept of probabilistic schema mappings and analyze their formal foundations. We show that there are two possible semantics for such mappings: *by-table* semantics assumes that there exists a correct mapping but we don't know what it is; *by-tuple* semantics assumes that the correct mapping may depend on the particular tuple in the source data. We present the query complexity and algorithms for answering queries in the presence of approximate schema mappings, and we describe an algorithm for efficiently computing the top-k answers to queries in such a setting.

1. INTRODUCTION

Data integration and exchange systems offer a uniform interface to a multitude of data sources and the ability to share data across multiple systems. These systems have recently enjoyed significant research and commercial success [12, 14, 16]. Current data integration systems are essentially a natural extension of traditional database systems in that queries are specified in a structured form and data is modeled in one of the traditional data models (relational, XML). In addition, the data integration system has exact knowledge of how the data in the sources map to the *mediated schema* used by the data integration system.

We argue that as the scope of data integration applications broadens, such systems need to be able to model uncertainty at their core. Uncertainty can arise for multiple reasons in data integration. First, the semantic mappings between the data sources and the mediated schema may be approximate. For example, in an application like Google Base or when mapping millions of sources on the deep web [21], we cannot imagine specifying exact mappings. In some domains (e.g., bioinformatics), we do not necessarily know what the exact mappings are. Second, if the intended users of the application are not necessarily familiar with schemas, or if the domain of the system is too broad to offer form-based query interfaces (such as web forms), we need to support keyword queries. Hence, a second source of uncertainty is the transformation between keyword queries and a set of candidate structured queries. Finally, data is often extracted from unstructured sources using information extraction techniques. Since these techniques are approximate, the data obtained from the sources may be uncertain.

Extending data integration systems to handle uncertainty is also a key step to realizing Dataspace Support Platforms [15]. The key idea of dataspaces is that data sources can be added with very little (or no) effort and the system still provides useful search, query and exploration services. Over time, the system evolves in a pay-as-you-go fashion to improve the quality of semantic mappings where they matter, therefore improving the quality of query answering. Hence, the data integration system we describe provides the underlying mechanisms that enable the dataspace to manage uncertainty about the semantic mappings, the queries, and the underlying data as the system evolves.

This paper takes a first step towards the goal of data integration with uncertainty. We first describe how the architecture of such a system differs from a traditional one (Section 2). At the core, the system models tuples and semantic mappings with probabilities associated with them. Query answering ranks answers and typically tries to obtain the top-k results to a query. These changes lead to a requirement for a new kind of adaptivity in query processing.

We then focus on one core component of data integration with uncertainty, namely probabilistic schema mappings (Section 3). Semantic mappings are the component of a data integration system that specifies the relationship between the contents of the different sources. The mappings enable the data integration to reformulate a query posed over the mediated schema into queries over the sources [18, 13]. We introduce *probabilistic schema mappings*, and describe how to answer queries in their presence.

We define probabilistic schema mapping as a set of pos-

VLDB '07, September 23-28, 2007, Vienna, Austria.

sible (ordinary) mappings between a source schema and a target schema, where each possible mapping has an associated probability. We begin by considering a simple class of mappings, where each mapping describes a set of correspondences between the attributes of a source table and the attributes of a target table. We argue that there are two possible interpretations of probabilistic schema mappings. In the first, called *by-table* semantics, we assume there exists a single correct mapping between the source and the target, but we don't know which one it is. In the second, called *by-tuple* semantics, the correct mapping may depend on the particular tuple in the source to which it is applied. In both cases, the semantics of query answers are a generalization of certain answers [1] for data integration systems.

We describe algorithms for answering queries in the presence of probabilistic schema mappings and then analyze the computational complexity of answering queries (Section 4). We show that the data complexity of answering queries in the presence of probabilistic mappings is PTIME for by-table semantics and #P-complete for by-tuple semantics. We identify a large subclass of real-world queries for which we can still obtain all the by-tuple answers in PTIME. We then describe algorithms for finding the top-k answers to a query (Section 5).

The size of a probabilistic mapping may be quite large, since it essentially enumerates a probability distribution by listing every combination of events in the probability space. In practice, we can often encode the same probability distribution much more concisely. Our next contribution (Section 6) is to identify two concise representations of probabilistic mappings for which query answering can be done in PTIME in the size of the mapping. We also examine the possibility of representing a probabilistic mapping as a Bayes Net, but show that query answering may still be exponential in the size of a Bayes Net representation of a mapping.

Finally, we consider several more powerful mapping languages, such as complex mappings, where the correspondences are between sets of attributes, and conditional mappings, where the mapping is conditioned on properties of the tuple to which it is applied (Section 7).

2. OVERVIEW OF THE SYSTEM

This section describes the requirements from a data integration system that supports uncertainty and the overall architecture of the system. We frame our specific contributions in context of this architecture.

2.1 Uncertainty in Data Integration

A data integration system needs to handle uncertainty at three levels.

Uncertain schema mappings: Data integration systems rely on schema mappings for specifying the semantic relationships between the data in the sources and the terms used in the mediated schema. However, schema mappings can be inaccurate. In many applications it is impossible to create and maintain precise mappings between data sources. This can be because the users are not skilled enough to provide precise mappings, such as in personal information management [7], because people do not understand the domain well and thus do not even know what correct mappings are, such as in bioinformatics, or because the scale of the data prevents generating and maintaining precise mappings, such as in integrating data of the web scale [21]. Hence, in practice, schema mappings are often generated by semi-automatic tools, and not necessarily verified by domain experts.

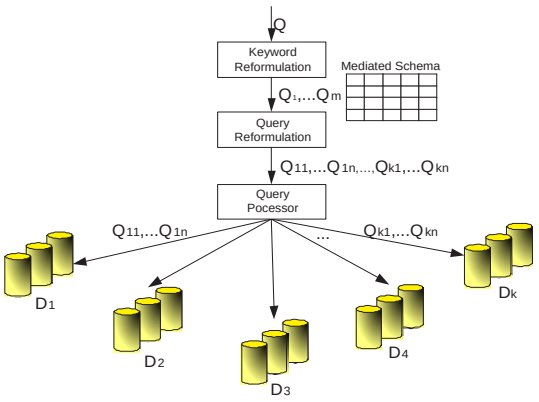

Figure 1: Architecture of a data-integration system that handles uncertainty.

Uncertain data: By nature, data integration systems need to handle uncertain data. One reason for uncertainty is that data is often extracted from unstructured or semi-structured sources by automatic methods (e.g., HTML pages, emails, blogs). A second reason is that data may come from sources that are unreliable or not up to date.

Uncertain queries: In some data integration applications, especially on the web, queries will be posed as keywords rather than as structured queries against a well defined schema. The system needs to translate these queries into some structured form so they can be reformulated with respect to the data sources. At this step, the system may generate multiple candidate structured queries and have some uncertainty about which is the real intent of the user.

2.2 System Architecture

Given the above requirements, we describe the architecture of a data integration system that manages uncertainty at its core. We describe the system by contrasting it to a traditional data integration system.

The first and most fundamental characteristic of this system is that it is based on a probabilistic data model. This means two things. First, as we process data in the system we attach probabilities to each tuple. Second, and the focus of this paper, is that schema mappings are also associated with probabilities, modeling our uncertainty about which ones are correct. We note that the probabilities associated with tuples, mappings, and answers are mostly internal to the system, and are not meant to be exposed to users. Typically, we will use these probabilities to rank answers.

Second, whereas traditional data integration systems begin by reformulating a query onto the schemas of the data sources, a data integration system with uncertainty needs to first reformulate a keyword query into a set of candidate structured queries. We refer to this step as *keyword reformulation*. Note that keyword reformulation is different from techniques for keyword search on structured data (e.g., [17, 2]) in that (a) it does not assume access to all the data in the sources or that the sources support keyword search, and (b) it tries to distinguish different structural elements in the query in order to pose more precise queries to the sources (e.g., realizing that in the keyword query "chicago weather", "weather" is an attribute label and "chicago" is an instance

name). That being said, keyword reformulation should benefit from techniques that support answering keyword search on structured data.

Third, the query answering model is different. Instead of necessarily finding *all* answers to a given query, our goal is typically to find the top-k answers, and rank these answers most effectively.

The final difference from traditional data integration systems is that our query processing will need to be more adaptive than usual. Instead of generating a query answering plan and executing it, the steps we take in query processing will depend on results of previous steps. We note that adaptive query processing has been discussed quite a bit in data integration [19], where the need for adaptivity arises from the fact that data sources did not answer as quickly as expected or that we did not have accurate statistics about their contents to properly order our operations. In our work, however, the goal for adaptivity is to get the answers with high probabilities faster.

The architecture of the system is shown in Figure 1. The system contains a number of data sources and a mediated schema. When the user poses a query Q, which can be either a structured query on the mediated schema, or a keyword query, the system returns a set of answer tuples, each with a probability. If Q is a keyword query, the system first performs keyword reformulation to translate it into a set of candidate structured queries on the mediated schema. Otherwise, the candidate query is Q itself.

Consider how the system answers the candidate queries, and assume the queries will not involve joins over multiple sources. For each candidate structured query Q_0 and a data source S, the system reformulates Q_0 according to the schema mapping (which can be uncertain) between S's schema and the mediated schema, sends the reformulated query (or queries) to S, retrieving the answers. If the user asks for all the answers to the query, then the reformulated query is typically a query with grouping and aggregation. If S does not support grouping and aggregation, then grouping and aggregation can be processed in the integration system. If the user asks for top-k answers, then query processing is more complex. The system reformulates the query into a set of queries, and uses a middle layer to decide at runtime which queries are critical to computing the top-k answers and sends the appropriate queries to S. Note that there can be multiple iterations of deciding the promising reformulated queries and retrieving answers. Furthermore, the system can even decide which data sources are more relevant and prioritize the queries to those data sources. Finally, if the data in the sources are uncertain, then the sources will return answers with probabilities attached to them.

After receiving answers from different data sources, the system combines them to get one single set of answer tuples. For example, if the data sources are known to be independent of each other, and we obtain tuple t from n data sources with probabilities $p_1, \ldots, p_n$ respectively, then in the final answer set t has probability $1 - \Pi_{i=1}^{n}(1 - p_i)$. If we know that some data sources are duplicates or extensions of others, a different combination function needs to be used.

2.3 Handling Uncertainty in Mappings

As a first step towards developing such a data integration system, we introduce in this paper *probabilistic schema mappings*, and show how to answer queries in their presence. Before the formal discussion, we illustrate the main ideas with an example.

	Possible Mapping	Prob
$m_1 =$	{(pname, name), (email-addr, email), (current-addr, mailing-addr), (permanent-addr, home-address)}	0.5
$m_2 =$	{(pname, name), (email-addr, email), (permanent-addr, mailing-addr), (current-addr, home-address)}	0.4
$m_3 =$	{(pname, name), (email-addr, mailing-addr), (current-addr, home-addr)}	0.1

(a)

$D_S =$

*p*name	*e*mail-addr	*p*ermanent-addr	*c*urrent-addr
Alice	alice@	Mountain View	Sunnyvale
Bob	bob@	Sunnyvale	Sunnyvale

(b)

Tuple	Prob
('Sunnyvale')	0.9
('Mountain View')	0.5
('alice@')	0.1
('bob@')	0.1

(c)

Figure 2: The running example: (a) a probabilistic schema mapping between S and T; (b) a source instance D_S; (c) the answers of Q over D_S with respect to the probabilistic mapping.

EXAMPLE 2.1. *Consider a data source S, which describes a person by her email address, current address, and permanent address, and the mediated schema T, which describes a person by her name, email, mailing address, home address and office address:*

```
S=(pname, email-addr, current-addr, permanent-addr)
T=(name, email, mailing-addr, home-addr, office-addr)
```

A semi-automatic schema-mapping tool may generate three possible mappings between S and T, assigning each a probability. Whereas the three mappings all map pname *to* name*, they map other attributes in the source and the target differently. Figure 2(a) describes the three mappings using sets of attribute correspondences. For example, mapping m_1 maps* pname *to* name*,* email-addr *to* email*,* current-addr *to* mailing-addr*, and* permanent-addr *to* home-addr*. Because of the uncertainty on which mapping is correct, we consider all of these mappings in query answering.*

Suppose the system receives a query Q composed on the mediated schema and asking for people's mailing addresses:

```
Q: SELECT mailing-addr FROM T
```

Using the possible mappings, we can reformulate Q into different queries:

```
Q1: SELECT current-addr FROM S
Q2: SELECT permanent-addr FROM S
Q3: SELECT email-addr FROM S
```

If the user requires all possible answers, the system generates a single aggregation query based on Q_1, Q_2 and Q_3 to compute the probability of each returned tuple, and sends the query to the data source. Suppose the data source contains a table D_S as shown in Figure 2(b), the system will retrieve four answer tuples, each with a probability, as shown in Figure 2(c).

If the user requires only the top-1 answer (i.e., the answer tuple with the highest probability), the system decides at runtime which reformulated queries to execute. For example, after executing Q_1 and Q_2 at the source, the system can already conclude that ('Sunnyvale') is the top-1 answer and can skip query Q_3. □

2.4 Source of probabilities

A critical issue in any system that manages uncertainty is whether we have a reliable source of probabilities. Whereas obtaining reliable probabilities for such a system is one of the most interesting areas for future research, there is quite a bit to build on. For keyword reformulation, it is possible to train and test reformulators on large numbers of queries such that each reformulation result is given a probability based on its performance statistics. In the case of schema matching, it is standard practice for schema matchers to also associate numbers with the candidates they propose. The issue here is that the numbers are meant only as a ranking mechanism rather than true probabilities. However, as schema matching techniques start looking a larger number of schemas, one can imagine ascribing probabilities (or approximations thereof) to their measures. Finally, information extraction techniques are also often based on statistical machine learning methods, thereby lending their predictions a probabilistic interpretation.

3. PROBABILISTIC SCHEMA MAPPING

In this section we formally define the semantics of probabilistic schema mappings and the query answering problems we consider. Our discussion is in the context of the relational data model. A *schema* contains a finite set of relations. Each relation contains a finite set of *attributes* and is denoted by $R = \langle r_1, \ldots, r_n \rangle$. An *instance* D_R of R is a finite set of *tuples*, where each tuple associates a value with each attribute in the schema.

We consider select-project-join (SPJ) queries in SQL. Note that answering such queries is in PTIME in the size of the data.

3.1 Schema Mappings

We begin by reviewing non-probabilistic schema mappings. The goal of a schema mapping is to specify the semantic relationships between a *source schema* and a *target schema*. We refer to the source schema as $\bar{S}$, and a relation in $\bar{S}$ as $S = \langle s_1, \ldots, s_m \rangle$. Similarly, we refer to the target schema as $\bar{T}$, and a relation in $\bar{T}$ as $T = \langle t_1, \ldots, t_n \rangle$.

The common formalism for semantic mappings, GLAV, is based on expressions of the form

$$m : \forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y}))$$

In the expression, ϕ is the body of a conjunctive query over $\bar{S}$ and ψ is the body of a conjunctive query over $\bar{T}$. A pair of instances D_S and D_T *satisfies* a GLAV mapping m, if for every assignment of $\mathbf{x}$ in D_S that satisfies ϕ there exists an assignment of $\mathbf{y}$ in D_T that satisfies ψ.

We consider a limited form of GLAV mappings that involve only projection queries on a single table on each side of the mapping. These mappings have also been referred to as *schema matching* in the literature [23]. Specifically, we consider GLAV mappings where (1) ϕ (resp. ψ) is an atomic formula over S (resp. T), (2) the GLAV mapping does not include constants, and (3) each variable occurs at most once on each side of the mapping. We consider this class of mappings because they already expose many of the novel issues involved in probabilistic mappings and because they are quite common in practice. We also note that many of the concepts we define apply to a broader class of mappings. In Section 7, we briefly discuss a set of extensions to our mappings, such as complex mappings.

$D_S =$

*p*name	*e*mail-addr	*p*ermanent-addr	*c*urrent-addr
Alice	alice@	Mountain View	Sunnyvale
Bob	bob@	Sunnyvale	Sunnyvale

(a)

*n*ame	*e*mail	*m*ailing-addr	*h*ome-addr	*o*ffice-addr
Alice	alice@	Mountain View	Sunnyvale	office
Bob	bob@	Sunnyvale	Sunnyvale	office

(b)

*n*ame	*e*mail	*m*ailing-addr	*h*ome-addr	*o*ffice-addr
Alice	alice@	Sunnyvale	Mountain View	office
Bob	email	bob@	Sunnyvale	office

(c)

Tuple	Prob
('Sunnyvale')	0.9
('Mountain View')	0.5
('alice@')	0.1
('bob@')	0.1

(d)

Tuple	Prob
('Sunnyvale')	0.94
('Mountain View')	0.5
('alice@')	0.1
('bob@')	0.1

(e)

Figure 3: Example 3.10: (a) a source instance D_S; (b) a target instance that is by-table consistent with D_S; (c) a target instance that is by-tuple consistent with D_S; (d) $Q^{table}(D_S)$; (e) $Q^{tuple}(D_S)$.

Given these restrictions, we can define our mappings in terms of *attribute correspondences*. An attribute correspondence is of the form $c_{ij} = (s_i, t_j)$, where s_i is a *source attribute* in the schema S and t_j is a *target attribute* in the schema T. Intuitively, c_{ij} specifies that there is a relationship between s_i and t_j. In practice, a correspondence also involves a function that transforms the value of s_i to the value of t_j. For example, the correspondence (c-degree, temperature) can be specified as temperature=c-degree$*1.8+$ 32, describing a transformation from Celsius to Fahrenheit. These functions are irrelevant to our discussion, and therefore we omit them. Formally, we define relation mappings and schema mappings as follows.

DEFINITION 3.1 (SCHEMA MAPPING). *Let $\bar{S}$ and $\bar{T}$ be relational schemas. A* relation mapping *M is a triple (S, T, m), where S is a relation in $\bar{S}$, T is a relation in $\bar{T}$, and m is a set of attribute correspondences between S and T.*

When each source and target attribute occurs in at most one correspondence in m, we call M a one-to-one relation mapping.

A schema mapping *$\overline{M}$ is a set of one-to-one relation mappings between relations in $\bar{S}$ and in $\bar{T}$, where every relation in either $\bar{S}$ or $\bar{T}$ appears at most once.* □

EXAMPLE 3.2. *Consider the mappings in Example 2.1. We can view m_1 as a GLAV mapping:*

$$\forall n, e, c, p(S(n, e, c, p) \rightarrow \exists o(T(n, e, c, p, o)))$$

The database in Figure 2 (b) (repeated in Figure 3 (a)) and the database in Figure 3 (b) satisfy m_1. □

3.2 Probabilistic Schema Mappings

Intuitively, a probabilistic schema mapping describes a probability distribution of a set of *possible* schema mappings between a source schema and a target schema.

DEFINITION 3.3 (PROBABILISTIC MAPPING). *Let $\bar{S}$ and $\bar{T}$ be relational schemas. A* probabilistic mapping (p-mapping)*, pM, is a triple $(S, T, \mathbf{m})$, where $S \in \bar{S}$, $T \in \bar{T}$, and $\mathbf{m}$ is a set $\{(m_1, Pr(m_1)), \ldots, (m_l, Pr(m_l))\}$, such that*

- *for $i \in [1, l]$, m_i is a one-to-one mapping between S and T, and for every $i, j \in [1, l]$, $i \neq j \Rightarrow m_i \neq m_j$.*
- *$Pr(m_i) \in [0, 1]$ and $\sum_{i=1}^{l} Pr(m_i) = 1$.*

A schema p-mapping*, $\overline{pM}$, is a set of p-mappings between relations in $\bar{S}$ and in $\bar{T}$, where every relation in either $\bar{S}$ or $\bar{T}$ appears in at most one p-mapping.* □

We refer to a non-probabilistic mapping as an *ordinary mapping*. Note that a schema p-mapping may contain both p-mappings and ordinary mappings. Example 2.1 shows a p-mapping (see Figure 2(a)) that contains three possible mappings.

3.3 Semantics of Probabilistic Mappings

Intuitively, a probabilistic schema mapping models the uncertainty about which of the mappings in pM is the correct one. When a schema matching system produces a set of candidate matches, there are two ways to interpret the uncertainty: (1) a single mapping in pM is the correct one and it applies to all the data in S, or (2) multiple mappings are correct and each suitable for a subset of tuples in S, though it is unknown which mapping is the right one for a specific tuple. Example 2.1 illustrates the first interpretation. For the same example, the second interpretation is equally valid: the correct mapping may depend on the particular tuple because some people use their current address as mailing address and some people use their permanent address as mailing address.

This paper analyzes query answering under both interpretations. We refer to the first interpretation as the *by-table* semantics and to the second one as the *by-tuple* semantics of probabilistic mappings. We note that we are not trying to argue for one interpretation over the other. The needs of the application should dictate the appropriate semantics. Furthermore, our complexity results, which will show advantages to by-table semantics, should not be taken as an argument in the favor of by-table semantics.

We next define the semantics of p-mappings in detail; the definitions for schema p-mappings are the obvious extensions. The semantics of p-mappings is defined as a natural extension of that of ordinary ones, which we review now. A mapping defines a relationship between instances of S and instances of T that are *consistent* with the mapping.

DEFINITION 3.4 (CONSISTENT TARGET INSTANCE). *Let $M = (S, T, m)$ be a relation mapping and D_S be an instance of S.*

An instance D_T of T is said to be consistent with *D_S and M, if D_S and D_T satisfy m.* □

For a relation mapping M and a source instance D_S, there can be an infinite number of target instances that are consistent with D_S and M. We denote by $Tar_M(D_S)$ the set of all such target instances. The set of answers to a query Q is the intersection of the answers on all instances in $Tar_M(D_S)$. The following definition is from [1].

DEFINITION 3.5 (CERTAIN ANSWER). *Let $M = (S, T, m)$ be a relation mapping. Let Q be a query over T and let D_S be an instance of S.*

A tuple t is said to be a certain answer of Q with respect to D_S and M*, if for every instance $D_T \in Tar_M(D_S)$, $t \in Q(D_T)$.* □

By-table semantics: We now generalize these notions to the probabilistic setting, beginning with the by-table semantics.

DEFINITION 3.6 (BY-TABLE CONSISTENT INSTANCE). *Let $pM = (S, T, \mathbf{m})$ be a p-mapping and D_S be an instance of S.*

An instance D_T of T is said to be by-table consistent with *D_S and pM, if there exists a mapping $m \in \mathbf{m}$ such that D_S and D_T satisfy m.* □

Given a source instance D_S and a possible mapping $m \in \mathbf{m}$, there can be an infinite number of target instances that are consistent with D_S and m. We denote by $Tar_m(D_S)$ the set of all such instances.

In the probabilistic context, we assign a probability to every answer. Intuitively, we consider the certain answers with respect to each possible mapping in isolation. The probability of an answer t is the sum of the probabilities of the mappings for which t is deemed as a certain answer. We define by-table answers as follows:

DEFINITION 3.7 (BY-TABLE ANSWER). *Let $pM = (S, T, \mathbf{m})$ be a p-mapping. Let Q be a query over T and let D_S be an instance of S.*

Let t be a tuple. Let $\bar{m}(t)$ be the subset of $\mathbf{m}$, such that for each $m \in \bar{m}(t)$ and for each $D_T \in Tar_m(D_S)$, $t \in Q(D_T)$.

Let $p = \sum_{m \in \bar{m}(t)} Pr(m)$. If $p > 0$, then we say (t, p) is a by-table answer of Q with respect to D_S and pM. □

By-tuple semantics: The key difference in the definition of by-tuple semantics is that a consistent target instance is defined by a mapping *sequence* that assigns a (possibly different) mapping in $\mathbf{m}$ to each tuple in D_S. (Without losing generality, in order to compare between such sequences, we assign some order to the tuples in the instance).

DEFINITION 3.8 (BY-TUPLE CONSISTENT INSTANCE). *Let $pM = (S, T, \mathbf{m})$ be a p-mapping and let D_S be an instance of S with d tuples.*

An instance D_T of T is said to be by-tuple consistent with *D_S and pM, if there is a sequence $\langle m^1, \ldots, m^d \rangle$ such that for every $1 \leq i \leq d$,*

- *$m^i \in \mathbf{m}$, and*
- *for the i^{th} tuple of D_S, t_i, there exists a target tuple $t'_i \in D_T$ such that t_i and t'_i satisfy m^i.* □

Given a mapping sequence $seq = \langle m^1, \ldots, m^d \rangle$, we denote by $Tar_{seq}(D_S)$ the set of all target instances that are consistent with D_S and seq. Note that if D_T is by-table consistent with D_S and m, then D_T is also by-tuple consistent with D_S and a mapping sequence in which each mapping is m.

We can think of every sequence of mappings $seq = \langle m^1, \ldots, m^d \rangle$ as a separate event whose probability is $Pr(seq) = \Pi_{i=1}^{d} Pr(m^i)$. (In Section 7 we relax this independence assumption and introduce *conditional mappings*.) If there are l mappings in pM, then there are l^d sequences of length d, and their probabilities add up to 1. We denote by $\mathbf{seq}_d(pM)$ the set of mapping sequences of length d generated from pM.

DEFINITION 3.9 (BY-TUPLE ANSWER). *Let $pM = (S, T, \mathbf{m})$ be a p-mapping. Let Q be a query over T and D_S be an instance of S with d tuples.*

Let t be a tuple. Let $\overline{seq}(t)$ be the subset of $\mathbf{seq}_d(pM)$, such that for each $seq \in \overline{seq}(t)$ and for each $D_T \in Tar_{seq}(D_S)$, $t \in Q(D_T)$.

Let $p = \sum_{seq \in \overline{seq}(t)} Pr(seq)$. If $p > 0$, we call (t, p) a by-tuple answer of Q with respect to D_S and pM. □

The set of by-table (resp. by-tuple) answers for Q with respect to D_S is denoted by $Q^{table}(D_S)$ (resp. $Q^{tuple}(D_S)$).

EXAMPLE 3.10. *Consider the p-mapping pM, the source instance D_S, and the query Q in the motivating example.*

In by-table semantics, Figure 3(b) shows a target instance that is consistent with D_S (repeated in Figure 3(a)) and possible mapping m_1. Figure 3(d) shows the by-table answers of Q with respect to D_S and pM. As an example, for tuple t ='Sunnyvale'), we have $\bar{m}(t) = \{m_1, m_2\}$, so the possible tuple ('Sunnyvale', 0.9) is an answer.

In by-tuple semantics, Figure 3(c) shows a target instance that is by-tuple consistent with D_S and the mapping sequence $< m_2, m_3 >$. Figure 3(e) shows the by-tuple answers of Q with respect to D_S and pM. □

4. QUERY ANSWERING COMPLEXITY

This section considers query answering in the presence of probabilistic mappings. We describe algorithms for query answering and study the complexity of query answering in terms of the size of the data (*data complexity*) and the size of the mapping (*mapping complexity*). We also consider cases in which we are not interested in the actual probability of an answer, just whether or not a tuple is a possible answer.

We show that returning all by-table answers is in PTIME for both complexity measures; whereas returning all by-tuple answers in general is #P-complete in data complexity. We show that computing the probabilities is the culprit here: even deciding the probability of a *single* answer tuple under by-tuple semantics is already #P-complete, whereas computing all by-tuple answers without returning the probabilities is in PTIME. Finally, we identify a large subclass of common queries where returning all by-tuple answers with their probabilities is still in PTIME.

4.1 By-table Query Answering

In the case of by-table semantics, answering queries is conceptually simple. Given a p-mapping $pM = (S, T, \mathbf{m})$ and an SPJ query Q, we can compute the certain answers of Q under each of the mappings $m \in \mathbf{m}$. We attach the probability $Pr(m)$ to every certain answer under m. If a tuple is an answer to Q under multiple mappings in $\mathbf{m}$, then we add up the probabilities of the different mappings.

Algorithm BYTABLE takes as input an SPJ query Q that mentions the relations $T_1, \ldots, T_l$ in the `FROM` clause. Assume that we have the p-mapping pM_i associated with the table T_i. The algorithm proceeds as follows.

Tuple	Prob
('Sunnyvale')	0.94
('Mountain View')	0.5
('alice@')	0.1
('bob@')	0.1

(a)

Tuple	Prob
('Sunnyvale')	0.8
('Mountain View')	0.8

(b)

Figure 4: (a) $Q_1^{tuple}(D)$ and (b) $Q_2^{tuple}(D)$.

Step 1: Generate the possible reformulations of Q by considering every combination of the form $(m^1, \ldots, m^l)$, where m^i is one of the possible mappings in pM_i. Denote the set of reformulations by $Q'_1, \ldots, Q'_k$. The probability of a reformulation $Q' = (m^1, \ldots, m^l)$ is $\Pi_{i=1}^{l} Pr(m^i)$.

Step 2: For each reformulation Q', retrieve each of the unique answers from the sources. For each answer obtained by $Q'_1 \cup \ldots \cup Q'_k$, its probability is computed by summing the probabilities of the Q''s in which it is returned.

Importantly, note that it is possible to express both steps as a SQL query with grouping and aggregation. Therefore, if the underlying sources support SQL, we can leverage their optimizations to compute the answers.

With our restricted form of schema mapping, the algorithm takes time polynomial in the size of the data and the mappings. We thus have the following complexity result (full proofs are given in [8]).

THEOREM 4.1. *Let $\overline{pM}$ be a schema p-mapping and let Q be an SPJ query.*

Answering Q with respect to $\overline{pM}$ is in PTIME in the size of the data and the mapping. □

GLAV mappings: It is rather straightforward to extend the above results to arbitrary GLAV mappings. We define *general p-mappings* to be triples of the form $pGM = (\bar{S}, \bar{T}, \mathbf{gm})$, where $\mathbf{gm}$ is a set $\{(gm_i, Pr(gm_i)) \mid i \in [1, n]\}$, such that for each $i \in [1, n]$, gm_i is a general GLAV mapping. The definition of by-table semantics for such mappings is a simple generalization of Definition 3.7. The following result holds for general p-mappings.

THEOREM 4.2. *Let pGM be a general p-mapping between a source schema $\bar{S}$ and a target schema $\bar{T}$. Let D_S be an instance of $\bar{S}$. Let Q be an SPJ query with only equality conditions over $\bar{T}$. The problem of computing $Q^{table}(D_S)$ with respect to pGM is in PTIME in the size of the data and the mapping.* □

4.2 By-tuple Query Answering

To extend the by-table query answering strategy to by-tuple semantics, we would need to compute the certain answers for every *mapping sequence* generated by pM (see Definition 3.8). However, the number of such mapping sequences is exponential in the size of the input data. The following example shows that for certain queries this exponential time complexity is not avoidable.

EXAMPLE 4.3. *Suppose that in addition to the tables in Example 2.1, we also have* U(city) *in the source and* V(hightech) *in the target. The p-mapping for V contains two possible mappings: ({(city, hightech)}, .8) and ($\emptyset$, .2).*

Consider the following query Q, which decides if there are any people living in a high-tech city.

```
Q: SELECT 'true'
   FROM T, V
   WHERE T.mailing-addr = V.hightech
```

One may conjecture we can answer the query by first executing the following two sub-queries Q_1 and Q_2, then joining the answers of Q_1 and Q_2 and summing up the probabilities.

```
Q1: SELECT mailing-addr FROM T
Q2: SELECT hightech FROM V
```

Now consider the source instance D, where D_S is shown in Figure 2(a), and D_U has two tuples ('Mountain View') and ('Sunnyvale'). Figure 4(a) and (b) show $Q_1^{tuple}(D)$ and $Q_2^{tuple}(D)$. If we join the results of Q_1 and Q_2, we obtain for the true *tuple the following probability:* $0.94*0.8+0.5*0.8 = 1.152$. *However, this is incorrect. By enumerating all consistent target tables, we in fact compute 0.864 as the probability. The reason for this error is that generating the tuple ('Sunnyvale') as an answer for both Q_1 and Q_2 and generating the tuple ('Mountain View') for both queries are not independent events; thus, simply adding up their probabilities leads to incorrect results.* □

In fact, we show that in general, answering SPJ queries with by-tuple semantics with respect to schema p-mappings is hard.

THEOREM 4.4. *Let Q be an SPJ query and let $\overline{pM}$ be a schema p-mapping. The problem of finding the probability for a by-tuple answer to Q with respect to $\overline{pM}$ is #P-complete with respect to data complexity and is in PTIME with respect to mapping complexity.* □

Recall that #P is the complexity class of some hard counting problems (e.g., counting the number of variable assignments that satisfy a boolean formula). It is believed that we cannot solve a #P-complete problem in polynomial time, unless $P = NP$. The lower bound in Theorem 4.4 is proved by reducing the problem of counting the number of variable assignments that satisfy a bipartite monotone 2DNF boolean formula to the problem of finding the answers to Q.

In fact, the reason for the high complexity is exactly that we are asking for the probability of the answer. The following theorem shows that if we only want to know the possible by-tuple answers, we can do so in polynomial time.

THEOREM 4.5. *Given an SPJ query and a schema p-map ping, returning all by-tuple answers without probabilities is in PTIME with respect to data complexity.* □

GLAV mappings: Extending by-tuple semantics to arbitrary GLAV mappings is a bit trickier than by-table semantics. It would involve considering Cartesian products of mapping sequences, but the length of these products depends on the number of tables in the query. We leave this extension to future work.

4.3 Two Restricted Cases

In this section we identify two restricted but common classes of queries for which by-tuple query answering takes polynomial time. We conjecture that these are the only cases where it is possible to answer a query in polynomial time.

In our discussion we refer to *subgoals* of a query. The subgoals are tables that occur in the FROM clause of a query. Hence, even if the same table occurs twice in the FROM clause, each occurrence is a different subgoal.

4.3.1 Queries with a single p-mapping subgoal

The first class of queries we consider are those that include only a single subgoal being the target of a p-mapping. Relations in the other subgoals are either involved in ordinary mappings or do not require a mapping. Hence, if we only have uncertainty with respect to one part of the domain, our queries will typically fall in this class. We call such queries *non-p-join queries*. The query Q in the motivating example is an example non-p-join query.

DEFINITION 4.6 (NON-P-JOIN QUERIES). *Let $\overline{pM}$ be a schema p-mapping and let Q be an SPJ query.*

If at most one subgoal in the body of Q is the target of a p-mapping in $\overline{pM}$, then we say Q is a non-p-join query with respect to $\overline{pM}$. □

For a non-p-join query Q, the by-tuple answers of Q can be generated from the by-table answers of Q over a set of databases, each containing a single tuple in the source table. Specifically, let $pM = (S, T, \mathbf{m})$ be the single p-mapping whose target is a relation in Q, and let D_S be an instance of S with d tuples. Consider the set of *tuple databases* $\mathbf{T}(D_S) = \{D_1, \ldots, D_d\}$, where for each $i \in [1, d]$, D_i is an instance of S and contains only the i-th tuple in D_S. The following lemma shows that $Q^{tuple}(D_S)$ can be derived from $Q^{table}(D_1), \ldots, Q^{table}(D_d)$.

LEMMA 4.7. *Let $\overline{pM}$ be a schema p-mapping between $\bar{S}$ and $\bar{T}$. Let Q be a non-p-join query over $\bar{T}$ and let D_S be an instance of $\bar{S}$. Let $(t, Pr(t))$ be a by-tuple answer with respect to D_S and $\overline{pM}$. Let $\bar{T}(t)$ be the subset of $\mathbf{T}(D_S)$ such that for each $D \in \bar{T}(t)$, $t \in Q^{table}(D)$. The following two conditions hold:*

1. $\bar{T}(t) \neq \emptyset$;
2. $Pr(t) = 1 - \Pi_{D \in \bar{T}(t), (t,p) \in Q^{table}(D)}(1 - p)$. □

In practice, answering the query for each tuple database can be expensive. We next describe Algorithm NONPJOIN, which computes the answers for all tuple databases in one step. The key idea of the algorithm is to distinguish answers generated by different source tuples. To do this, we assume there is an identifier attribute id for the source relation whose values are concatenations of values of the key columns. We now describe the algorithm in detail.

Algorithm NONPJOIN takes as input an SPJ query Q, a schema p-mapping $\overline{pM}$, and a source instance D_S, and proceeds in three steps to compute all by-tuple answers.

Step 1: Rewrite Q to Q' such that it returns T.id in addition. Revise the p-mapping such that each possible mapping contains the correspondence between S.id and T.id.

Step 2: Invoke BYTABLE with Q', $\overline{pM}$ and D_S. Note that each generated result tuple contains the id column in addition to the attributes returned by Q.

Step 3: Project the answers returned in Step 2 on Q's returned attributes. Suppose projecting $t_1, \ldots, t_n$ obtains the answer tuple t, then the probability of t is $1 - \Pi_{i=1}^{n}(1 - Pr(t_i))$.

EXAMPLE 4.8. *Consider rewriting Q in the motivating example, repeated as follows:*

```
Q: SELECT mailing-addr FROM T
```

Step 1 rewrites Q into query Q' by adding the id *column:*

```
Q': SELECT id, mailing-addr FROM T
```

In Step 2, ByTable *generates the following SQL query to compute by-table answers for Q':*

```
Qa: SELECT id, mailing-addr, SUM(pr)
    FROM (
      SELECT DISTINCT id, current-addr AS mailing-addr,
                      0.5 AS pr
      FROM S
      UNION ALL
      SELECT DISTINCT id, permanant-addr AS mailing-addr,
                      0.4 AS pr
      FROM S
      UNION ALL
      SELECT DISTINCT id, email-addr AS mailing-addr,
                      0.1 AS pr
      FROM S)
    GROUP BY id, mailing-addr
```

Step 3 then generates the results using the following query.

```
Qu:  SELECT mailing-addr, NOR(pr) AS pr
     FROM Qa
     GROUP BY mailing-addr
```

where for a set of probabilities $pr_1, \ldots, pr_n$, NOR computes $1 - \Pi_{i=1}^{n} pr_i$. □

An analysis of Algorithm NonPJoin leads to the following complexity result for non-p-join queries.

Theorem 4.9. *Let $\overline{pM}$ be a schema p-mapping and let Q be a non-p-join query with respect to $\overline{pM}$.*

Answering Q with respect to $\overline{pM}$ in by-tuple semantics is in PTIME in the size of the data and the mapping. □

4.3.2 Projected p-join queries

We now show that query answering can be done in polynomial time for a class of queries, called *projected p-join queries*, that include multiple subgoals involved in p-mappings. In such a query, we say that a join predicate is a *p-join predicate* with respect to a schema p-mapping $\overline{pM}$, if at least one of the involved relations is the target of a p-mapping in $\overline{pM}$. We define projected p-join queries as follows.

Definition 4.10 (projected p-join query). *Let $\overline{pM}$ be a schema p-mapping and Q be an SPJ query over the target of $\overline{pM}$. If the following conditions hold, we say Q is a* projected p-join query with respect to $\overline{pM}$*:*

- *at least two subgoals in the body of Q are targets of p-mappings in $\overline{pM}$.*
- *for every p-join predicate, the join attribute (or an attribute that is entailed to be equal by the predicates in Q) is returned in the* `SELECT` *clause.* □

Example 4.11. *Consider the schema p-mapping in Example 4.3. A slight revision of Q, shown as follows, is a projected-p-join query.*

```
Q': SELECT V.hightech
    FROM T, V
    WHERE T.mailing-addr = V.hightech
```
□

Note that in practice, when joining data from multiple tables in a data integration scenario, we typically project the join attributes, thereby leading to projected p-join queries.

The key to answering a projected-p-join query Q is to divide Q into multiple subqueries, each of which is a non-p-join query, and compute the answer to Q from the answers to the subqueries. We proceed by considering partitions of the subgoals in Q. We say that a partitioning $\bar{J}$ is a *refinement* of a partitioning $\bar{J}'$, denoted $\bar{J} \preceq \bar{J}'$, if for each partition $J \in \bar{J}$, there is a partition $J' \in \bar{J}'$, such that $J \subseteq J'$. We consider the following partitioning of Q, the generation of which will be described in detail in the algorithm.

Definition 4.12 (Maximal P-Join Partitioning). *Let $\overline{pM}$ be a schema p-mapping. Let Q be an SPJ query and $\bar{J}$ be a partitioning of the subgoals in Q.*

We say that $\bar{J}$ is a p-join partitioning of Q*, if (1) each partition $J \in \bar{J}$ contains at most one subgoal that is the target of a p-mapping in $\overline{pM}$, and (2) if neither subgoal in a join predicate is involved in p-mappings in $\overline{pM}$, the two subgoals belong to the same partition.*

We say that $\bar{J}$ is a maximal p-join partitioning of Q*, if there does not exist a p-join partitioning $\bar{J}'$, such that $\bar{J} \preceq \bar{J}'$.* □

For each partition $J \in \bar{J}$, we can define a query Q_J as follows. The `FROM` clause includes the subgoals in J. The `SELECT` clause includes J's attributes that occur in (1) Q's `SELECT` clause or (2) Q's join predicates that join subgoals in J with subgoals in other partitions. The `WHERE` clause includes Q's predicates that contain only subgoals in J. When J is a partition in a maximal p-join partitioning of Q, we say that Q_J is a *p-join component* of Q.

The following is the main lemma underlying our algorithm. It shows that we can compute the answers of Q from the answers to its p-join components.

Lemma 4.13. *Let $\overline{pM}$ be a schema p-mapping. Let Q be a projected p-join query with respect to $\overline{pM}$ and let $\bar{J}$ be a maximal p-join partitioning of Q. Let $Q_{J1}, \ldots, Q_{Jn}$ be the p-join components of Q with respect to $\bar{J}$.*

For any instance D_S of the source schema of $\overline{pM}$ and result tuple $t \in Q^{tuple}(D_S)$, the following two conditions hold:

1. *For each $i \in [1, n]$, there exists a single tuple $t_i \in Q_{Ji}^{tuple}(D_S)$, such that $t_1, \ldots, t_n$ generate t when joined together.*
2. *Let $t_1, \ldots, t_n$ be the above tuples. Then, $Pr(t) = \Pi_{i=1}^{n} Pr(t_i)$.* □

Lemma 4.13 entails to the query-rewriting algorithm ProjectedPJoin, which takes as input a projected-p-join query Q, a schema p-mapping $\overline{pM}$, and a source instance D_S, outputs all by-tuple answers:

Step 1: Generate maximum p-join partitions $J_1, \ldots, J_n$ as follows. First, initialize each partition to contain one subgoal in Q. Then, for each join predicate with subgoals S_1 and S_2 that are not involved in p-mappings in $\overline{pM}$, merge the partitions that S_1 and S_2 belong to. Finally, for each partition that contains no subgoal involved in $\overline{pM}$, merge it with another partition.

Step 2: For each p-join partition $J_i, i \in [1, n]$, generate the p-join component Q_{Ji}, and invoke Algorithm NonPJoin with Q_{Ji}, $\overline{pM}$ and D_S to compute answers for Q_{Ji}.

Step 3: Join the results of $Q_{J1}, \ldots, Q_{Jn}$. If an answer tuple t is obtained by joining $t_1, \ldots, t_n$, then the probability of t is computed by $\Pi_{i=1}^{n} Pr(t_i)$.

We illustrate the algorithm using the following example.

EXAMPLE 4.14. *Continue with Example 4.11. Its two p-join components are Q_1 and Q_2 shown in Example 4.3. Suppose we compute Q_1 with query Q_u (shown in Example 4.8) and compute Q_2 with query Q'_u. We can compute by-tuple answers of Q' as follows:*

```
SELECT Qu'.hightech, Qu.pr*Qu'.pr
FROM Qu, Qu'
WHERE Qu.mailing-addr = Qu'.hightect
```

□

Since the number of p-join components is bounded by the number of subgoals in a query, and for each of them we invoke Algorithm NONPJOIN, query answering for projected p-join queries takes polynomial time.

THEOREM 4.15. *Let $\overline{pM}$ be a schema p-mapping and let Q be a projected-p-join query with respect to $\overline{pM}$.*

Answering Q with respect to $\overline{pM}$ in by-tuple semantics is in PTIME in the size of the data and the mapping. □

4.3.3 Other SPJ queries

A natural question is whether the two classes of queries we have identified are the only ones for which query answering is in PTIME for by-tuple semantics. As Example 4.3 shows, if Q contains multiple subgoals that are involved in a schema p-mapping, but Q is not a projected-p-join query, then Condition 1 in Lemma 4.13 does not hold, and query answering needs to proceed by enumerating all mapping sequences.

We believe that the complexity of the border case, where a query joins two relations involved in p-mappings but does not return the join attribute, is #P-hard, but currently it remains an open problem.

5. TOP-K QUERY ANSWERING

In this section, we consider returning the top-k query answers, which are the k answer tuples with the top probabilities. The main challenge in designing the algorithm is to only perform the necessary reformulations at every step and halt when the top-k answers are found. We first describe our algorithm for by-table semantics. We then show the challenges for by-tuple semantics and outline our solution, but defer the details to the full version of the paper.

5.1 Returning Top-K By-table Answers

Recall that in by-table query answering, the probability of an answer is the sum of the probabilities of the reformulated queries that generate the answer. Our goal is to reduce the number of reformulated queries we execute. Our algorithm proceeds in a greedy fashion: we execute queries in descending order of probabilities. For each tuple t, we maintain the upper bound $p_{max}(t)$ and lower bound $p_{min}(t)$ of its probability. This process halts when we find k tuples whose p_{min} values are higher than p_{max} of the rest of the tuples.

TOPKBYTABLE takes as input an SPJ query Q, a schema p-mapping $\overline{pM}$, an instance D_S of the source schema, and an integer k, and outputs the top-k answers in $Q^{table}(D_S)$. The algorithm proceeds in three steps.

Step 1: Rewrite Q according to $\overline{pM}$ into a set of queries $Q_1, \ldots, Q_n$, each with a probability assigned in a similar way as stated in Algorithm BYTABLE.

Step 2: Execute $Q_1, \ldots, Q_n$ in descending order of their probabilities. Maintain the following measures:

- The highest probability, $PMax$, for the tuples that have not been generated yet. We initialize $PMax$ to 1; after executing query Q_i and updating the list of answers (see third bullet), we decrease $PMax$ by $Pr(Q_i)$;
- The threshold th determining which answers are potentially in the top-k. We initialize th to 0; after executing Q_i and updating the answer list, we set th to the k-th largest p_{min} for tuples in the answer list;
- A list L of answers whose p_{max} is no less than th, and bounds p_{min} and p_{max} for each answer in L. After executing query Q_i, we update the list as follows: (1) for each $t \in L$ and $t \in Q_i(D_S)$, we increase $p_{min}(t)$ by $Pr(Q_i)$; (2) for each $t \in L$ but $t \notin Q_i(D_S)$, we decrease $p_{max}(t)$ by $Pr(Q_i)$; (3) if $PMax \geq th$, for each $t \notin L$ but $t \in Q_i(D_S)$, insert t to L, set p_{min} to $Pr(Q_i)$ and $p_{max}(t)$ to $PMax$.
- A list T of k tuples with top p_{min} values.

Step 3: When $th > PMax$ and for each $t \notin T$, $th > p_{max}(t)$, halt and return T.

EXAMPLE 5.1. *Consider Example 2.1 where we seek the top-1 answer. We answer the reformulated queries in order of Q_1, Q_2, Q_3. After answering Q_1, for tuple ("Sunnyvale") we have $p_{min} = .5$ and $p_{max} = 1$, and for tuple ("Mountain View") we have the same bounds. In addition, $PMax = .5$ and $th = .5$.*

In the second round, we answer Q_2. Then, for tuple ("Sunnyvale") we have $p_{min} = .9$ and $p_{max} = 1$, and for tuple ("Mountain View") we have $p_{min} = .5$ and $p_{max} = .6$. Now $PMax = .1$ and $th = .9$.

Because $th > PMax$ and th is above the p_{max} for the ("Mountain View") tuple, we can halt and return ("Sunnyvale") as the top-1 answer. □

The next theorem states the correctness of BYTABLETOPK.

THEOREM 5.2. *For any schema mapping $\overline{pM}$, SPJ query Q, instance D_S of the source schema of $\overline{pM}$, and integer k, Algorithm* BYTABLETOPK *correctly computes the top-k answers in $Q^{table}(D_S)$.* □

Note that in the worst case we need to execute all of $Q_1, \ldots, Q_n$ in Step 2, so the time complexity of TOPKBYTABLE is the same as BYTABLE. However, in many cases we can return top-k answers without executing all of the reformulated queries, and thus the greedy strategy can improve the efficiency of query answering.

Our algorithm differs from previous top-k algorithms in the literature in two aspects. First, we execute the reformulated queries only when necessary, so we can return the top-k answers without executing all reformulated queries thereby leading to significant performance improvements. Fagin et al. [9] have proposed several algorithms for finding instances with top-k scores, where each instance has m attributes and the score of the instance is an aggregation over values of these m attributes. However, these algorithms assume that for each attribute there exists a sorted list on its values, and they access the lists in parallel. In our context, this would require executing all reformulated queries upfront. Li et al. [20] have studied computing top-k answers for aggregation and group-by queries and optimizing query answering by generating the groups incrementally. Although we can

also compute by-table answers using an aggregation query, this query is different from those considered in [20] in that the WHERE clause contains a set of sub-queries rather than database tables. Therefore, applying [20] here also requires evaluating all reformulated queries at the beginning.

Second, whereas maintaining upper bounds and lower bounds for instances has been explored in the literature, such as in Fagin's NRA (Non-Random Access) algorithm and in [20], our algorithm is different in that it keeps these bounds only for tuples that have already been generated by an executed reformulated query and that are potential top-k answers (by judging if the upper bound is above the threshold th). For unseen result tuples, we compute the upper bound of their probability by exploiting the fact that the upper bound of each probability score is 1.

5.2 By-tuple Top-K Query Answering

We next consider returning top-k answers in by-tuple semantics. In general, we need to consider each mapping sequence and answer the query on the target instance that is consistent with the source and the mapping sequence. Algorithm TopKByTable can be modified to compute top-k by-tuple answers by deciding at runtime the mapping sequence to consider next. However, for non-p-join queries and projected-p-join queries, we can return top-k answers more efficiently. We outline our method for answering non-p-join queries here, and leave the details of projected-p-join queries to the full paper.

For non-p-join queries the probability of an answer tuple t to query Q cannot be expressed as a function of t's probabilities in executing reformulations of Q; rather, it is a function of t's probabilities in answering Q on each tuple database of the source table. However, retrieving answers on a tuple base is expensive. Algorithm NonPJoin provides a method that computes by-tuple answers on the tuple databases in batch mode by first rewriting Q into Q' by returning the id column and then executing Q''s reformulated queries. We find top-k answers in a similar fashion. Here, after executing each reformulated query, we need to maintain two answer lists, one for Q and one for Q', and compute p_{min} and p_{max} for answers in different lists differently.

6. P-MAPPING REPRESENTATIONS

Thus far, a p-mapping was represented by listing each of its possible mappings, and the complexity of query answering was polynomial in the size of that representation. Such a representation can be quite lengthy since it essentially enumerates a probability distribution by listing every combination of events in the probability space. Hence, an interesting question is whether there are more concise representations of p-mappings and whether our algorithms can leverage them.

We consider three representations that can reduce the size of the p-mapping exponentially. In Section 6.1 we consider a representation in which the attributes of the source and target tables are partitioned into groups and p-mappings are specified for each group separately. We show that query answering can be done in time polynomial in the size of the representation. In Section 6.2 we consider probabilistic correspondences, where we specify the marginal probability of each attribute correspondence. However, we show that such a representation can only be leveraged in limited cases. Finally, we consider Bayes Nets, the most common method for concisely representing probability distributions, in Section 6.3, and show that even though some p-mappings can be represented with them, query answering does not necessarily benefit from the representation.

Mapping	Prob
{(a,a'), (b,b'), (c,c')}	0.72
{(a,b'), (c,c')}	0.18
{(a,a'), (b,b')}	0.08
{(a,b')}	0.02

(a)

Mapping	Prob
{(a,a'), (b,b')}	0.8
{(a,b')}	0.2

(b)

Mapping	Prob
{(c,c')}	0.9
∅	0.1

(c)

Figure 5: The p-mapping in (a) is equivalent to the 2-group p-mapping in (b) and (c).

6.1 Group Probabilistic Mapping

In practice, the uncertainty we have about a p-mapping can often be represented as a few localized choices, especially when schema mappings are created by semi-automatic methods. To represent such p-mappings more concisely, we can partition the source and target attributes and specify p-mappings for each partition.

Definition 6.1 (Group P-Mapping). *An n-group p-mapping gpM is a triple $(S, T, \overline{pM})$, where*

- *S (resp. T) is a source (resp. target) relation schema and $\{S_1, \ldots, S_n\}$ (resp. $\{T_1, \ldots, T_n\}$) is a set of disjoint subsets of attributes in S (resp. T);*
- *$\overline{pM}$ is a set of p-mappings $\{pM_1, \ldots, pM_n\}$, where for each $1 \leq i \leq n$, pM_i is a p-mapping between S_i and T_i.* □

The semantics of an n-group p-mapping $gpM = (S, T, \overline{pM})$ is a p-mapping that includes the Cartesian product of the mappings in each of the pM_i's. The probability of the mapping composed of $m_1 \in pM_1, \ldots, m_n \in pM_n$ is $\Pi_{i=1}^{n} Pr(m_i)$.

Example 6.2. *Figure 5(a) shows p-mapping pM between the schemas $S(a, b, c)$ and $T(a', b', c')$. Figure 5(b) and (c) show two independent mappings that together form a 2-group p-mapping equivalent to pM.* □

Note that a group p-mapping can be considerably more compact than an equivalent p-mapping. Specifically, if each pM_i includes l_i mappings, then a group p-mapping can describe $\Pi_{i=1}^{n} l_i$ possible mappings with $\sum_{i=1}^{n} l_i$ sub-mappings. The important feature of n-group p-mappings is that query answering can be done in time polynomial in their size.

Theorem 6.3. *Let $\overline{gpM}$ be a schema group p-mapping and let Q be an SPJ query. The mapping complexity of answering Q with respect to $\overline{gpM}$ in both by-table semantics and by-tuple semantics is PTIME.* □

Theorem 6.4. *Given a p-mapping pM, we can find in polynomial time in the size of pM the maximal n and an n-group p-mapping gpM, such that gpM is equivalent to pM.* □

Mapping	Prob
{(a,a'), (b,b'), (c,c')}	0.8
{(a,b'), (c,c')}	0.1
{(a,b')}	0.1

(a)

Corr	Prob
{(a,a')}	0.8
{(a,b')}	0.2
{(b,b')}	0.8
{(c,c')}	0.9

(b)

Figure 6: The p-mapping in (a) corresponds to the p-correspondence in (b).

6.2 Probabilistic Correspondences

The second representation we consider, *probabilistic correspondences*, represents a p-mapping with the marginal probabilities of attribute correspondences. This representation is the most compact one as its size is proportional to the product of the schema sizes of S and T.

DEFINITION 6.5 (PROBABILISTIC CORRESPONDENCES). *A probabilistic correspondence mapping (p-correspondence) is a triple $pC = (S, T, \mathbf{c})$, where $S = \langle s_1, \ldots, s_m \rangle$ is a source relation schema, $T = \langle t_1, \ldots, t_n \rangle$ is a target relation schema, and*

- $\mathbf{c}$ *is a set $\{(c_{ij}, Pr(c_{ij}))|i \in [1,m], j \in [1,n]\}$, where $c_{ij} = (s_i, t_j)$ is an attribute correspondence, and $Pr(c_{ij}) \in [0,1]$;*
- *for each $i \in [1,m]$, $\sum_{j=1}^{n} Pr(c_{ij}) \leq 1$;*
- *for each $j \in [1,n]$, $\sum_{i=1}^{m} Pr(c_{ij}) \leq 1$.* □

Note that for a source attribute s_i, we allow $\sum_{j=1}^{n} Pr(c_{ij}) < 1$. This is because in some of the possible mappings, s_i may not be mapped to any target attribute. The same is true for target attributes.

¿From each p-mapping, we can infer a p-correspondence by calculating the marginal probabilities of each attribute correspondence. Specifically, for a p-mapping $pM = (S, T, \mathbf{m})$, we denote by $pC(pM)$ the p-correspondence where each mar ginal probability is computed as follows:

$$Pr(c_{ij}) = \sum_{c_{ij} \in m, m \in \mathbf{m}} Pr(m)$$

However, the relationship between p-mappings and p-correspondences is many-to-one. For example, The p-correspondence in Figure 6(b) is the one computed both for the p-mapping in Figure 6(a) and for the p-mapping in Figure 5(a).

Given the many-to-one relationship, the question is when it is possible to compute the correct answer to a query based only on the p-correspondence. That is, we are looking for a class of queries $\bar{Q}$, called *p-mapping independent queries*, such that for every $Q \in \bar{Q}$ and every database instance D_S, if $pC(pM_1) = pC(pM_2)$, then the answer of Q with respect to pM_1 and D_S is the same as the answer of Q with respect to pM_2 and D_S. Unfortunately, this property holds for a very restricted class of queries, defined as follows:

DEFINITION 6.6 (SINGLE-ATTRIBUTE QUERY). *Let $pC = (S, T, \mathbf{c})$ be a p-correspondence. An SPJ query Q is said to be a* single-attribute *query with respect to pC if T has one single attribute occurring in the* `SELECT` *and* `WHERE` *clauses of Q.* □

THEOREM 6.7. *Let $\overline{pC}$ be a schema p-correspondence, and Q be an SPJ query. Then, Q is p-mapping independent with respect to $\overline{pC}$ if and only if for each $pC \subseteq \overline{pC}$, Q is a single-attribute query with respect to pC.* □

6.3 Bayes Nets

Bayes Nets are a powerful mechanism for concisely representing probability distributions and reasoning about probabilistic events [22]. The following example shows how Bayes Nets can be used in our context.

EXAMPLE 6.8. *Consider two schemas $S = (s_1, \ldots, s_n, s'_1, \ldots, s'_n)$ and $T = (t_1, \ldots, t_n)$. Consider the p-mapping $pM = (S, T, \mathbf{m})$, which describes the following p-mapping: if s_1 maps to t_1 then it is more likely that $\{s_2, \ldots, s_n\}$ maps to $\{t_2, \ldots, t_n\}$, whereas if s'_1 maps to t_1 then it is more likely that $\{s'_2, \ldots, s'_n\}$ maps to $\{t_2, \ldots, t_n\}$.*

We can represent the p-mapping using a Bayes Net as follows. Let c be an integer constant. Then,

1. *$Pr((s_1, t_1)) = Pr((s'_1, t_1)) = 1/2$;*
2. *for each $i \in [1,n]$, $Pr((s_i, t_i)|(s_1, t_1)) = 1 - \frac{1}{c}$ and $Pr((s'_i, t_i)|(s_1, t_1)) = \frac{1}{c}$;*
3. *for each $i \in [1,n]$, $Pr((s_i, t_i)|(s'_1, t_1)) = \frac{1}{c}$ and $Pr((s'_i, t_i)|(s'_1, t_1)) = 1 - \frac{1}{c}$.*

Since the p-mapping contains 2^n possible mappings, the original representation would take space $O(2^n)$; however, the Bayes-Net representation takes only space $O(n)$. □

Although the Bayes-Net representation can reduce the size exponentially for some p-mappings, this conciseness may not help reduce the complexity of query answering. We formalize this result in the following theorem.

THEOREM 6.9. *There exists a schema p-mapping $\overline{pM}$ and a query Q, such that answering Q with respect to $\overline{pM}$ in by-table semantics takes exponential time in the size of $\overline{pM}$'s Bayes-Net representation.* □

7. BROADER CLASSES OF MAPPINGS

In this section we briefly show how our results can be extended to capture two common practical extensions to our mapping language.

Complex mappings: Complex mappings map a set of attributes in the source to a set of attributes in the target. For example, we can map the attribute address to the concatenation of street, city, and state.

Formally, a *set correspondence* between S and T is a relationship between a subset of attributes in S and a subset of attributes in T. Here, the function associated with the relationship specifies a single value for each of the target attributes given a value for each of the source attributes. Again, the actual functions are irrelevant to our discussion. A *complex mapping* is a triple (S, T, cm), where cm is a set of set correspondences, such that each attribute in S or T is involved in at most one set correspondence. A *probabilistic complex mapping* is of the form $\overline{pCM} = \{(cm_i, Pr(cm_i)) \mid i \in [1,n]\}$, where $\sum_{i=1}^{n} Pr(cm_i) = 1$.

THEOREM 7.1. *Let $\overline{pCM}$ be a schema probabilistic complex mapping between schemas $\bar{S}$ and $\bar{T}$. Let D_S be an instance of $\bar{S}$. Let Q be an SPJ query over $\bar{T}$. The data complexity and mapping complexity of computing $Q^{table}(D_S)$ w.r.t. $\overline{pCM}$ are PTIME. The data complexity of computing $Q^{tuple}(D_S)$ w.r.t. $\overline{pCM}$ is #P-complete. The mapping complexity of computing $Q^{tuple}(D_S)$ w.r.t. $\overline{pCM}$ is PTIME.* □

Conditional mappings: In practice, our uncertainty is often conditioned. For example, we may want to state that daytime-phone maps to work-phone with probability 60% if age ≤ 65, and maps to home-phone with probability 90% if age > 65.

We define a *conditional p-mapping* as a set $\overline{cpM} = \{(pM_1, C_1), \ldots, (pM_n, C_n)\}$, where $pM_1, \ldots, pM_n$ are p-mappings, and $C_1, \ldots, C_n$ are pairwise disjoint conditions. Intuitively, for each $i \in [1, n]$, pM_i describes the probability distribution of possible mappings when condition C_i holds. Conditional mappings make more sense for by-tuple semantics. The following theorem shows that our results carry over to such mappings.

THEOREM 7.2. *Let $\overline{cpM}$ be a schema conditional p-mapping between $\bar{S}$ and $\bar{T}$. Let D_S be an instance of $\bar{S}$. Let Q be an SPJ query over $\bar{T}$. The problem of computing $Q^{tuple}(D_S)$ with respect to $\overline{cpM}$ is in PTIME in the size of the mapping and #P-complete in the size of the data.* □

8. RELATED WORK

We are not aware of any previous work studying the semantics and properties of probabilistic schema mappings. Gal [11] used the top-K schema mappings obtained by a semi-automatic mapper to improve the precision of the top mapping, but did not address any of the issues we consider. Florescu et al. [10] were the first to advocate the use of probabilities in data integration. Their work used probabilities to model (1) a mediated schema with overlapping classes (e.g, DatabasePapers and AIPapers), (2) source descriptions stating the probability of a tuple being present in a source, and (3) overlap between data sources. Although these are important aspects of many domains and should be incorporated into a data integration system, our focus here is different. De Rougement and Vieilleribiere [6] considered approximate data exchange in that they relaxed the constraints on the target schema, which is a different approach from ours.

There has been a flurry of activity around probabilistic and uncertain databases lately [4, 24, 5, 3]. Our intention is that a data integration system will be based on a probabilistic data model, and we leverage concepts from that work as much as possible. We also believe that uncertainty and lineage are closely related, in the spirit of [4], and that relationship will play a key role in data integration. We leave exploring this topic to future work.

9. CONCLUSIONS AND FUTURE WORK

This paper took the first step towards data integration systems that handle uncertainty about queries, mappings and data in a principled fashion. We introduced probabilistic semantic mappings, which play a key role in such a system. We presented query answering algorithms for by-table and by-tuple semantics and for the case where the user is interested in the top-k answers. We also considered concise encoding of probabilistic mappings.

Many other problems need to be solved to realize the data integration systems we envision. Aside from the connection between uncertainty and lineage in such systems, we also want to incorporate uncertainty about the results of keyword reformulation and about possibly dirty data. In addition, we want to establish a formal connection between schema mapping tools and data integration systems that support uncertainty. In one direction this means extracting probabilities about schema mappings from the results of schema mapping tools. In the reverse direction, our goal is to reason globally about the uncertainty in a data integration system and its effect on query answers, in order to decide where it is most beneficial to expand more resources (human or otherwise) to improve schema mappings.

10. REFERENCES

[1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *PODS*, 1998.

[2] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *ICDE*, 2002.

[3] L. Antova, C. Koch, and D. Olteanu. World-set decompositions: Expressiveness and efficient algorithms. In *ICDT*, 2007.

[4] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, 2006.

[5] R. Cheng, S. Prabhakar, and D. V. Kalashnikov. Querying imprecise data in moving object environments. In *ICDE*, 2003.

[6] M. de Rougemont and A. Vieilleribiere. Approximate data exchange. In *ICDT*, 2007.

[7] X. Dong and A. Halevy. A platform for personal information management and integration. In *CIDR*, 2005.

[8] X. Dong, A. Halevy, and C. Yu. Probabilistic schema mapping. Technical Report 2006-12-01, Univ. of Washington, 2006. http://www.cs.washington.edu/homes/lunadong/publication/pmapping_techReport.pdf.

[9] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[10] D. Florescu, D. Koller, and A. Levy. Using probabilistic information in data integration. In *VLDB*, 1997.

[11] A. Gal. Managing uncertainty in schema matching with Top-K schema mappings. *Journal on Data Semantics*, 6, 2006.

[12] L. Haas. The theory and practice of information integration. In *ICDT*, pages 28–43, 2007.

[13] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4), 2001.

[14] A. Y. Halevy, N. Ashish, D. Bitton, M. J. Carey, D. Draper, J. Pollock, A. Rosenthal, and V. Sikka. Enterprise information integration: successes, challenges and controversies. In *SIGMOD*, 2005.

[15] A. Y. Halevy, M. J. Franklin, and D. Maier. Principles of dataspace systems. In *PODS*, 2006.

[16] A. Y. Halevy, A. Rajaraman, and J. J. Ordille. Data integration: The teenage years. In *VLDB*, 2006.

[17] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *VLDB*, 2002.

[18] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.

[19] A. Y. Levy. Special issue on adaptive query processing. *IEEE Data Eng. Bull.*, 23(2), 2000.

[20] C. Li, K. C.-C. Chang, and I. F. LLyas. Supporting ad-hoc ranking aggregates. In *SIGMOD*, 2006.

[21] J. Madhavan, S. Cohen, X. Dong, A. Halevy, S. Jeffery, D. Ko, and C. Yu. Navigating the seas of structured web data. In *CIDR*, 2007.

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

[23] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[24] D. Suciu and N. N. Dalvi. Foundations of probabilistic answers to queries. In *SIGMOD*, 2005.