Aggregate Query Answering under Uncertain Schema Mappings

Avigdor Gal [#1], Maria Vanina Martinez, Gerardo I. Simari, VS Subrahmanian [*2]

[#]*Technion – Israel Institute of Technology*
Haifa 32000 Israel
[1]avigal@ie.technion.ac.il

[*]*University of Maryland*
College Park, MD
[2]{mvm,gisimari,vs}@cs.umd.edu

***Abstract*— Recent interest in managing uncertainty in data integration has led to the introduction of probabilistic schema mappings and the use of probabilistic methods to answer queries across multiple databases using two semantics: by-table and by-tuple. In this paper, we develop three possible semantics for aggregate queries: the *range, distribution*, and *expected value* semantics, and show that these three semantics combine with the by-table and by-tuple semantics in six ways. We present algorithms to process `COUNT`, `AVG`, `SUM`, `MIN`, and `MAX` queries under all six semantics and develop results on the complexity of processing such queries under all six semantics. We show that computing `COUNT` is in PTIME for all six semantics and computing `SUM` is in PTIME for all but the by-tuple/distribution semantics. Finally, we show that `AVG`, `MIN`, and `MAX` are PTIME computable for all by-table semantics and for the by-tuple/range semantics. We developed a prototype implementation and experimented with both real-world traces and simulated data. We show that, as expected, naive processing of aggregates does not scale beyond small databases with a small number of mappings. The results also show that the polynomial time algorithms are scalable up to several million tuples as well as with a large number of mappings.**

I. Introduction

There has been intense work during the last few years on schema matching in order to answer queries over multiple databases [1], [2], [3], [4], [5], [6], [7], [8], [9], [10]. More recently, there has been a realization that methods to automatically match schemas are uncertain — when an algorithm for schema matching is executed, it might say that there are many possible mappings between one schema and another, and that a probability distribution over this set of mappings specifies the probability that a specific mapping is the correct one [11]. For example, when a company A merges with a company B, the employee databases of the two companies need to be merged. The schemas may be different, and in such a case, there may be many different ways of mapping one schema to another and a probability distribution might tell us the probability that a given schema is correct. Alternatively, a web search engine doing a product search over the databases of multiple vendors needs to find mappings between the product database of one vendor, and the product database of another vendor. Multiple ways of representing the data might lead to multiple possible schema mappings, together with a probability distribution over the set of mappings. In a residential or commercial real estate web site that aggregates information from multiple realtors across the country, there is a need to find mappings between disparate schemas — usually multiple possible mappings are found by an automated schema mapping tool [12], and a probability distribution over these mappings can specify the probability that a given schema is correct. Our paper will use such a real estate example as a motivating example.

Work to date [11], [13] on probabilistic schema mapping has studied two semantics for answering queries over multiple databases using probabilistic schema matches — a "by-table" semantics and a "by-tuple" semantics. However, when aggregate queries are considered, then in addition to whether a by-table or by-tuple semantics should be used, we need to consider the semantics of the aggregates themselves in the presence of uncertainty. Some work has been done on aggregates over uncertain data [14], [15], yet none at all w.r.t. aggregate computations under probabilistic schema mapping. Aggregates over probabilistic data can be processed in three ways (at least). In the first way, an aggregate query returns a set of possible values for the answer, together with a probability distribution over that set. We call this the "distribution" semantics. A second method returns just a *range* specifying the lowest and highest possible values for the aggregate query. We call this the "range" semantics. The third semantics returns an *expected value*. In this paper, we first propose these three semantics for aggregate computations and then show that they combine with the by-table and by-tuple semantics of [11] in six possible ways, yielding six possible semantics for aggregates in probabilistic schema mapping.

We develop algorithms to compute `COUNT`, `MIN`, `MAX`, `SUM`, and `AVG` under each of the six semantics and show that the algorithms are correct. We develop a characterization of the computational complexity of the problem of computing these six semantics. For all the above aggregate operators, we show that semantics based on the by-table semantics are PTIME computable. For the `COUNT` operator, we show that query results for all six semantics can be computed in PTIME. Computing the `SUM` operator is in PTIME for all but

the by-tuple/distribution semantics. Finally, we show that for MIN, MAX, and AVG, the only by-tuple semantics that can be efficiently computed is the range semantics.

We have developed a prototype implementation of our algorithms and tested out their efficiency on large data sets, showing that our algorithms work very efficiently in practice. Our experiments show the computational feasibility of the different semantics for each of the aggregate operators mentioned above. We show that, for each aggregate operator considered in this paper under the by-tuple semantics, the algorithms for computing the range semantics are very efficient and scalable; this is also the case for COUNT under the other two semantics. Furthermore, the expected value semantics for SUM is also very efficient since we can take advantage of the fact that it is guaranteed to be equivalent to the by-table semantics, as we show in this work. In summary, for each aggregate operator, there is at least one semantics where our experiments show that it can be computed very efficiently.

To summarize, our contributions are as follows:

- We show six possible semantics for aggregate queries with uncertain schema mappings.
- We show several cases under the by-tuple semantics where efficient algorithms exist for aggregate computation.
- We prove that for the SUM aggregate operator, by-tuple/expected value and by-table/expected value semantics yield the same answer.
- Using a thorough empirical setup, we show that the polynomial time algorithms are scalable up to several million tuples (with some even beyond 30 million tuples) and with a large number of mappings.

The rest of the paper is organized as follows. Section II provides background on aggregate query answering under uncertain schema mapping. The six semantics for aggregate query processing in the presence of uncertain schema mappings is described in detail in Section III. Section IV provides a set of algorithms for efficient computation of the various aggregates. Our empirical analysis is provided in Section V We conclude with a discussion of related work (Section VI) and directions for future work (Section VII).

II. Preliminaries

This paper focuses on the problem of aggregate query processing across multiple databases in the presence of probabilistic schema mappings. The system may contain a number of data sources and a mediated schema, as in [11]. Alternatively, a peer database system, with multiple data sources (*e.g.*, DBlife like information) and no mediated schema, as in [16], [17] may also be in place.

There are many cases where a precise schema mapping may not be available. For instance, a comparison search "bot" that tracks comparative prices from different web sites has — in real time — to determine which fields at a particular location correspond to which fields in a database at another URL. Likewise, as in the case of [11], in many cases, users querying two databases belonging to different organizations may not know what is the right schema mapping. We model this uncertainty about which schema mapping is correct by using probability theory. This robust model allows us to provide, in the case of aggregate queries, not only a ranking of the results, but also the expected value of the aggregate query outcome and the distribution of possible aggregate values.

ID	price	agentPhone	postedDate	reducedDate
1	100k	215	1/5/2008	1/30/2008
2	150k	342	1/30/2008	2/15/2008
3	200k	215	1/1/2008	1/10/2008
4	100k	337	1/2/2008	2/1/2008

TABLE I
AN INSTANCE D_{S1}

We focus on five types of aggregate queries: COUNT, MIN, MAX, SUM, and AVG. Given a mediated schema, a query Q, and a data source S, Q is reformulated according to the (probabilistic) schema mapping between S's schema and the mediated schema, and posed to S, retrieving the answers according to the appropriate semantics (to be discussed shortly).

We focus on efficient processing of aggregate queries. An orthogonal challenge in this setting involves record linkage and cleansing that relates to duplicates. We assume the presence of effective tools for solving this problem [18], [19] and focus on correct and efficient processing of the data. Also, we focus on the analysis of aggregate queries over a single table, to avoid mixing issues with joins over uncertain schema mappings. Our analysis tests the effect of executing an aggregate query over a single table or a table that is the result of any SPJ query over the non probabilistic part of the schema.

We define schema mappings between a source schema S and a target T in terms of attribute correspondences of the form $c_{ij} = (s_i, t_j)$, where s_i in S is the *source attribute* and t_i in T is the *target attribute*. For illustration purposes, we shall use the following two examples throughout the paper:

Example 1: Consider a real-estate data source S1, which describes properties for sale, their list price, an agent's contact phone, and the posting date. If the price of a property was reduced, then the date on which the most recent reduction occurred is also posted. The mediated schema T1 describes property list price, contact phone number, date of posting, and comments:

S1 = (ID, price, agentPhone, postedDate, reducedDate)
T1 = (propertyID, listPrice, phone, date, comments)

For the sake of simplicity, we assume that the mapping of ID to propertyID, price to listedPrice, and agentPhone to phone is known. In addition, there is no mapping to comments. Due to lack of background information, it is not clear whether date should be mapped to postedDate (denoted as mapping m_{11}) or reducedDate (denoted mapping m_{12}). Because of the uncertainty regarding which mapping is correct, we consider both mappings when answering queries. We can assign a probability to each such mapping (*e.g.*, m_{11} has probability 0.6 and m_{12} has probability 0.4). Such a probability may be computed automatically by algorithms to identify the correct mapping [9]. Table I shows an instance of a table D_{S1} of data source S1.

Suppose that on February 20, 2008 the system receives a query

transactionID	auctionID	time	bid	currentPrice
3401	34	0.43	195	195
3402	34	2.75	200	197.5
3403	34	2.8	331.94	202.5
3404	34	2.85	349.99	336.94
3801	38	1.16	330.01	300
3802	38	2.67	429.95	335.01
3803	38	2.68	439.95	336.30
3804	38	2.82	340.5	438.05

TABLE II
AN INSTANCE D_{S2}

Q1, composed on schema T1, asking for the number of "old" properties, those listed for more than a month:

```
Q1: SELECT COUNT(*) FROM T1
    WHERE date < '2008-1-20'
```

Using mapping m_{11}, we can reformulate Q1 into the following query:

```
Q11: SELECT COUNT(*) FROM S1
     WHERE postedDate < '2008-1-20'
```

Example 2: As another example, consider eBay auctions. These auctions have a strict end date for each auction and use a second-price model. That is, the winner is the one who places the highest bid, but the winning price is (a delta higher than) the second-highest bid. Now consider two (simplified) database schemas, S2 and T2, that keep track of auction prices:

S2 = (transactionID, auction, time, bid, currentPrice)
T2 = (transaction, auctionId, timeUpdate, price)

For simplicity, we again assume that the mappings of transactionID to transaction, auction to auctionID and the mapping of time to timeUpdate are known. The attribute price in T2 can be mapped to either the bid attribute (denoted as mapping m_{21}) or the currentPrice attribute (denoted as mapping m_{22}) in S2. Here, the source of uncertainty may be attributed to the sometimes confusing semantics of the bid and the current price in eBay auctions. Assume that m_{21} is assigned probability 0.3 and m_{22} is assigned probability 0.7. Table II contains data for two auctions (numbers 34 and 38) with four bids each. The time is measured from the beginning of the auction and therefore 0.43 means that about 10 hours (less than half a day) have passed from the opening of the auction. Suppose that the system receives a query Q2 w.r.t. schema T2, asking for the average closing price of all auctions:

```
Q2: SELECT AVG(R1.price) FROM
(SELECT MAX(DISTINCT R2.price)
   FROM T2 AS R2
   GROUP BY R2.auctionID) AS R1
```

The subquery, within the FROM clause, identifies the maximum price for each auction. Using mapping m_{21}, we can reformulate Q2 to be:

```
Q21: SELECT AVG(R1.currentPrice) FROM
(SELECT MAX(DISTINCT R2.currentPrice)
   FROM T2 AS R2
   GROUP BY R2.auction) AS R1
```

We base our model of probabilistic schema mappings on the one presented in [11], extending it to answer aggregate queries. In what follows, given relational schemas $\overline{S}$ and $\overline{T}$, S a relation in $\overline{S}$, and T a relation in $\overline{T}$, an attribute correspondence is a one-to-one mapping from the attribute names in S to the attribute names in T. Also, a one-to-one relation mapping is a mapping where each source and target attribute occurs in at most one correspondence.

Definition 1 (Schema Mapping): Let $\overline{S}$ and $\overline{T}$ be relational schemas. A relation mapping M is a triple (S, T, m), where S is a relation in $\overline{S}$, T is a relation in $\overline{T}$, and m is a set of attribute correspondences between S and T.

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

The following definition, also from [11], extends the concept of schema mapping with probabilities:

Definition 2 (Probabilistic Mapping): Let $\overline{S}$ and $\overline{T}$ be relational schemas. A *probabilistic mapping (p-mapping)* pM is a triple $(S, T, \mathbf{m})$, where $S \in \overline{S}$, $T \in \overline{T}$, and $\mathbf{m}$ is a set $\{(m_1, Pr(m_1)), ..., (m_l, Pr(m_l))\}$, such that

- for $i \in [1, l]$, m_i is a one-to-one relation 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 $\overline{S}$ and in $\overline{T}$, where every relation in either $\overline{S}$ or $\overline{T}$ appears in at most one p-mapping.

III. SEMANTICS

We now present the semantics of aggregate queries in the presence of probabilistic schema mappings. We start with a formal presentation of the by-table and by-tuple semantics, as introduced in [11] (Section III-A). Then, we move on to introduce three aggregate semantics and their combination with the by-table and by-tuple semantics (Section III-B).

A. Semantics of Probabilistic Mappings

The intuitive interpretation of a probabilistic schema mapping as presented in [11] is that there is uncertainty about which of the mappings is the *right one*. Such uncertainty may be rooted in the fact that "the syntactic representation of schemas and data do not completely convey the semantics of different databases," [20] *i.e.*, the description of a concept in a schema can be semantically misleading. As proposed in [11], there are two ways in which this uncertainty can be interpreted: either a single mapping should be applied to the entire set of tuples in the source relation, or a choice of a mapping should be made for each of these tuples. The former is referred to as the *by-table* semantics, and the latter as the *by-tuple* semantics. The *by-tuple* semantics represents a situation in which data is gathered from multiple sources, each with a potentially different interpretation of a schema.

As discussed in [11], the high complexity of query answering under the by-tuple semantics is due to the fact that all possible *sequences* of mappings (of length equal to the number of tuples in the table) must be considered in the general case.

The following examples illustrate the difference between the two semantics when considering aggregate functions.

Example 3: Consider the scenario presented in Example 1. Assume the content of table $\mathsf{D_{S1}}$ is as shown in Table I. Using the two possible mappings, we can reformulate Q1 into the following two queries, one for each possible way of mapping attribute date:

```
Q11: SELECT COUNT(*) FROM S1
       WHERE postedDate < '2008-1-20'
Q12: SELECT COUNT(*) FROM S1
       WHERE reducedDate < '2008-1-20'
```

We can adapt the procedure described for the by-table semantics in [11] to answer uncertain aggregate queries, by computing each of the two previous reformulated queries as if they were the correct mappings and the probability of the corresponding mapping is assigned to each answer. In this case, the system provides answer 3 with probability 0.6 (from query Q11) and answer 2 with probability 0.4 (from query Q12). Under the by-tuple semantics it is necessary to consider all possible sequences, *i.e.*, ways of assigning a mapping to a tuple. For instance, the sequence $s = \langle m_{11}, m_{12}, m_{12}, m_{11} \rangle$ represents the fact that tuple 1 and 4 should be interpreted under mapping m_{11}, in which case attribute date is mapped to postedDate, and tuples 2 and 3 should be interpreted using mapping m_{12} which maps date to reducedDate. Each sequence has an associated probability equal to the product of the probability of each mapping in the sequence, since mappings are independently assigned to tuples. For instance, the probability of sequence s is

$$Pr(s) = 0.6 * 0.4 * 0.4 * 0.6 = 0.0576$$

An answer in this case, as discussed in [11] for general SPJ queries, can be obtained by computing the aggregate operator for each possible sequence. The final answer is a table that contains all the different values obtained from the answers yielded by each individual computation, each with an associated probability. The probability for each value is the sum of the probabilities of all sequences that yield that value. In this example, the final answer is 1 with probability 0.16, 2 with probability 0.48, and 3 with probability 0.36.

Example 4: Let us now consider Table II and query Q2, presented in Example 2. Using the two possible mappings, we can reformulate Q2 into the following two queries:

```
Q21: SELECT AVG(R1.currentPrice) FROM
(SELECT MAX(DISTINCT R2.currentPrice)
   FROM T2 AS R2
   GROUP BY R2.auction) AS R1
Q22: SELECT AVG(R1.bid) FROM
(SELECT MAX(DISTINCT R2.bid)
   FROM T2 AS R2
   GROUP BY R2.auction) AS R1
```

Using the by-table semantics, the system provides the answer 345.245 with probability 0.3 and 385.945 with probability 0.7. Under the by-tuple semantics, in order to compute an answer to Q2, given that there are 8 tuples in the database instance and 2 possible mappings, we have to look at $2^8 = 256$ sequences. We need to compute the answer for each sequence and then combine the results.

B. Semantics for Aggregate Queries Under Uncertain Schema Mappings

Aggregate queries provide users with answers that are not simple cut & paste data from the database. Rather, data is processed and user expectations are also different. In many cases, users expect a simple, single answer to an aggregate query (*e.g.*, counting the number of newly posted houses). Therefore, when extended to probabilistic schema mappings, such expectations should be taken into account.

In this work, we consider three common extensions to semantics with aggregates and probabilistic information. The *range semantics* gives an interval within which the aggregate is guaranteed to lie. The *distribution semantics* specifies all possible values that the aggregate can take, and for each such value, it gives the probability that it is the correct one. Of course, we can easily derive the answer to an aggregate query under the range semantics from the answer to the same query under the distribution semantics. Finally, for those who like the answer to be a single number, we develop an *expected value* semantics which returns the expected value of the aggregate. Note that the answer to a query under the expected value semantics can also be computed from the answer to the query under the distribution semantics. In a sense, the answer according to the distribution semantics is rich, containing details that are eliminated in the other two semantics. However, as we will see below, the other two semantics may be more efficiently computable without obtaining the distribution at all.

Let $\mathbf{m} = \{(m_1, Pr(m_1)), ..., (m_l, Pr(m_m))\}$ be a set of all possible mappings from schema S to schema T, each with an associated probability $Pr(m_i)$, where $\sum_i Pr(m_i) = 1$. Let $V = \{v_1, ..., v_n\}$ be the set of results of evaluating the aggregate function for each possible mapping or a sequence of mappings. The three possible semantics for query answering with aggregate functions and multiple possible schema mappings can be formalized as follows:

1) *Range Semantics:* The result of the aggregate function under the *range semantics* is the interval $[\min(V), \max(V)]$.
2) *Probability Distribution Semantics:* Under the *probability distribution* semantics, the result of the aggregate function is a random variable X. For every distinct value $r_j \in V$, we have that

$$Pr(X = r_j) = \sum_{v_i \in V, v_i = r_j} Pr(m_i) \qquad (1)$$

3) *Expected Value Semantics:* Let $V = \{v_1, ..., v_n\}$ be the set of results of evaluating the aggregate function for each possible mapping. The result of the aggregate function under the *expected value semantics* is

$$\sum_{i=1}^{n} Pr(m_i) * v_i \qquad (2)$$

The fact that answers to queries under the range and expected value semantics can be immediately derived from the answer under the distribution semantics tells us that if the distribution semantics is PTIME computable, then the range and expected value semantics should also be PTIME computable.

Possible Combinations of Semantics. When combining the by-table and by-tuple semantics with the three aggregate semantics suggested in Section III-B, a space of six possible semantics for aggregate queries over probabilistic schema mappings is created. This space is illustrated in Table III, where for each semantics we give the query answer to query Q1.

COUNT	Range	Distribution	Exp. Value
By-Table	$[2,3]$	3 (prob 0.6), 2 (prob 0.4)	2.6
By-Tuple	$[1,3]$	see Example 3	2.2

TABLE III
THE SIX SEMANTICS OF AGGREGATE QUERIES OVER PROBABILISTIC SCHEMA MAPPING

IV. ALGORITHMS FOR AGGREGATE QUERY ANSWERING

A. By-Table Semantics

Figure 1 provides a "generic" algorithm to answer aggregate queries under the by-table semantics, extending a similar algorithm in [11]. The algorithm reformulates the input query into l new queries, one for each possible schema mapping and obtains an answer r_i to the query w.r.t. that mapping. It then outputs the result using a `CombineResults` function. `CombineResults` returns $[min(r_1, \ldots, r_m), max(r_1, \ldots, r_m)]$ when the semantics chosen is the range semantics. When the semantics chosen is the expected value semantics, it returns $\Sigma_{i=1}^{m} Pr(m_i) * r_i$ where $Pr(m_i)$ is the probability that the mapping that maps A to A_i is correct. When the semantics chosen is the distribution semantics, it returns the set of all pairs $\{(r_i, p) \mid p = \Sigma_{r_j = r_i} Pr(m_i)\}$.

B. By-Tuple Semantics

The by-tuple semantics associates a mapping with each tuple in a relational table. Hence, if we have n tuples and m different mappings, there are m^n different sequences that assign mappings to tuples. The problem of answering select, project, join queries under the by-tuple semantics is in general #P-complete in data complexity [11]. The reason for the high complexity stems from the need to assign probabilities to each tuple. Computing all by-tuple answers without returning the probabilities is in PTIME. When it comes to aggregate queries, however, merely computing all possible tuples is not enough. One also needs to know, for each possible mapping sequence, whether a tuple belongs to it or not. Therefore, in the worst case, going through all possible mapping sequences is unavoidable. To see why, consider the query

```
SELECT SUM(price) FROM T2
```

against Table II.

With 2 possible mappings and 8 tuples, there are $2^8 = 256$ possible sequences. In this case, there are 128 different possible values — in fact, there would have been 256 different possible values if the bid and currentPrice of the first tuple did not have the same value (195). Therefore, merely enumerating all possible answers may yield an exponential number of answers.

tupleID	*low*	*up*	*comment*
	0	0	initialization
1	0	1	cond. satisfied under m_{11}
2	1	2	cond. satisfied under both mappings
3	1	2	cond. satisfied under no mapping
4	1	3	cond. satisfied under m_{11}

TABLE IV
TRACE OF BYTUPLERANGE FOR QUERY Q1

The generic (naïve) algorithm discussed earlier can be greatly improved when we consider specific aggregate functions. In this section, we show how to achieve this for the `COUNT`, `SUM`, `AVG`, `MAX`, and `MIN` aggregate functions under the three alternative semantics presented in Section III. We show that in certain aggregate/semantics combinations, it is possible to compute an answer in PTIME, whereas for others PTIME algorithms could not be found.

Aggregate function COUNT. We present algorithms to compute the `COUNT` aggregate under by-tuple/range and by-tuple/distribution semantics. The answer for the expected value semantics can be computed directly from the result provided by the algorithm for distribution semantics.

We will use our running examples presented in Section II. Consider the setting from Example 1, the data in Table I, and query Q1:

```
SELECT COUNT(*) FROM T1
       WHERE date < '1-20-2008'
```

`COUNT` Under the Range Semantics. Under the range semantics, the answer to query Q1 should provide the minimum and the maximum value for the aggregate, considering any of the mappings. The algorithm is shown in Figure 2. The idea behind the algorithm is simple: each tuple, depending on the mapping that is used for it, may or may not satisfy the selection condition for the `COUNT`. Clearly, if a tuple satisfies the select condition under all mappings, then both the minimum and maximum possible values for `COUNT` should be increased. If the tuple does not satisfy the select condition under all mappings, then it is never included in the aggregate result. Finally, if there is at least one mapping under which the tuple does not satisfy the select condition, then the minimum value does not change, but the maximum does.

To see how this algorithm works, we include in Table IV the trace of how the bounds are updated with each tuple in Table I to answer query Q1. For instance, we can see that for tuple 1 only the upper bound is incremented because this tuple satisfies the select condition only for mapping m_{11}. The last row of the table shows the final answer, $[1, 3]$.

Note that this algorithm looks at each tuple only once, and in each step it looks at most at all mappings once. Thus, if n is the number of tuples in S and m is the number of

```
Algorithm ByTableAggregateQuery
Input: Table S, T; MapList M; Attribute A; Condition C; AggregateFunction Agg;
       Semantics S
1. Let |M| = l be the number of mappings for attribute A;
2. Let A_1, ..., A_l be all the attributes to which A maps;
3. For i = 0 to l,
4.   Let r_i be the answer for the query:
     SELECT Agg(A_i) FROM T WHERE C GROUP BY B;
5. return CombineResults(r_1, ..., r_l, S);
```

Fig. 1. Generic by-table algorithm adapted from Halevy's work for Aggregate Queries

```
Algorithm ByTupleRangeCOUNT
Input: Table S, T; MapList M; Attribute A; Condition C
1. Let up an low be equal to 0;
2. For each t_i ∈ S,
3.   if for all mappings m_j ∈ M such that t_i satisfies C then
       low = low + 1; up = up + 1;
     else if there exists at least one mapping m_j ∈ M for which t_i satisfies C then
       up = up + 1;
4. return [low, up];
```

Fig. 2. Algorithm to answer `SELECT COUNT(A) FROM T WHERE C` under Range Semantics

```
Algorithm ByTuplePDCOUNT
Input: Table S, T; MapList M; Attribute A; Condition C
1. Let pd be a new probability distribution;
2. In pd set Pr(0) = 1.0;
3. For each t_i ∈ S,
4.   Let occProb be the sum of the probabilities of the mappings in M under which
     t_i satisfies C;
5.   Let notOccProb be the sum of the probabilities of the mappings in M under
     which t_i does not satisfy C;
6.   In pd set Pr(0) = Pr(0) * notOccProb;
7.   For j = 1 to i − 1,
8.     In pd set Pr(j) = (Pr(j) * notOccProb) + (Pr(j − 1) * occProb);
9.   In pd set Pr(i) = Pr(i − 1) * occProb;
10. return pd;
```

Fig. 3. Algorithm to answer `SELECT COUNT(A) FROM T WHERE C` under Distribution Semantics

possible mappings, the number of computations needed for this algorithm is in $O(n * m)$.

Theorem 1: Algorithm `ByTupleRangeCOUNT` correctly computes the result of executing a `COUNT` query under the by-tuple range semantics.[1]

`COUNT` *Under the Distribution Semantics.* A naïve way of computing an answer for a query such as Q1 under the distribution semantics is to consider all possible sequences of mappings and to compute the query for each sequence, as shown in the second part of Example 3. However, we present a more efficient algorithm that only takes polynomial time in the number of mappings and the number of tuples in the table. The pseudo-code of this algorithm is outlined in Figure 3.

Under a given mapping, a tuple can either add 0 to the `COUNT` result or 1. Hence, the probability of a tuple adding 1 to the result is that of the mapping itself, and the probability of adding nothing is the complementary probability. This reasoning can be easily extended to multiple mappings by taking the sum of the probabilities for which the tuple adds 1 to the calculation. If we look at each tuple in turn, the value of the aggregate at a certain time depends on how many tuples were taken into account. However, at each step, the count can at most be incremented by one, depending on whether the tuple at hand satisfies the selection condition. This means that if we are looking at tuple i, and the count so far is c_{i-1}, then after looking at tuple i the count will either be c_{i-1} or $c_{i-1} + 1$. Since this can be the case at each update, we must store all possible values for the result at each step. For instance, after looking at just one tuple, only two values are possible (0 and 1), and when we look at another tuple, the value 2 now becomes possible. The probabilities associated with each of these results can be easily updated at each step by looking at two values as shown in the algorithm.

Table V shows the trace of how the probability distribution is updated with each tuple in Table I to answer query Q1. For instance, consider the second row in the table, where tuple 2 is processed. This tuple has probability 0 of being part of the result because under both mappings it does not satisfy the select condition. The probability of the result being 0 is

[1]Proofs of all theorems in this paper are given in [21].

tupleID	0	1	2	3	4
1	0.4	0.6			
2	0.4	0.6	0		
3	0	0.4	0.6	0	
4	0	0.16	0.48	0.36	0

TABLE V

TRACE OF BYTUPLEPDCOUNT FOR QUERY Q1

tupleID	v_i^{min}	v_i^{max}	*low*	*up*
			0	0
1	195	195	195	195
2	197.5	200	392.5	395
3	336.3	439.95	728.8	834.95
4	340.5	438.05	1069.3	1273

TABLE VI

TRACE OF BYTUPLERANGE FOR QUERY Q2'

Sequence	SUM	p	SUM$\times p$
$(m_{21}, m_{21}, m_{21}, m_{21})$	1076.93	0.0081	8.723133
$(m_{21}, m_{21}, m_{21}, m_{22})$	1063.88	0.0189	20.107332
$(m_{21}, m_{21}, m_{22}, m_{21})$	947.49	0.0189	17.907561
$(m_{21}, m_{21}, m_{22}, m_{22})$	934.44	0.0441	41.208804
$(m_{21}, m_{22}, m_{21}, m_{21})$	1074.43	0.0189	20.306727
$(m_{21}, m_{22}, m_{21}, m_{22})$	1061.38	0.0441	46.806858
$(m_{21}, m_{22}, m_{22}, m_{21})$	944.99	0.0441	41.674059
$(m_{21}, m_{22}, m_{22}, m_{22})$	931.94	0.1029	95.896626
$(m_{22}, m_{21}, m_{21}, m_{21})$	1076.93	0.0189	20.353977
$(m_{22}, m_{21}, m_{21}, m_{22})$	1063.88	0.0441	46.917108
$(m_{22}, m_{21}, m_{22}, m_{21})$	947.49	0.0441	41.784309
$(m_{22}, m_{21}, m_{22}, m_{22})$	934.44	0.1029	96.153876
$(m_{22}, m_{22}, m_{21}, m_{21})$	1074.43	0.0441	47.382363
$(m_{22}, m_{22}, m_{21}, m_{22})$	1061.38	0.1029	109.216002
$(m_{22}, m_{22}, m_{22}, m_{21})$	944.99	0.1029	97.239471
$(m_{22}, m_{22}, m_{22}, m_{22})$	931.94	0.2401	223.758794
Expected value			975.437

TABLE VII

COMPUTING Q2′ UNDER THE BY-TUPLE/EXPECTED VALUE SEMANTICS

now 0.4; this is because the count can only be 0 if it was 0 before and tuple 2 is not part of the count ($0.4 * 1.0 = 0.4$); the probability of the result being 1 is updated in the following way: the value can only be 1 if either it was 0 before and tuple 2 satisfies the condition, or it was already 1 and tuple 2 does not satisfy the condition ($0.4 * 0 + 0.6 * 1.0 = 0.6$). Finally, 2 is a new possible value with probability 0 for now. Note that each row is a probability distribution among the values considered thus far. The final probability distribution is the same as shown in Example 3.

Theorem 2: Algorithm `ByTuplePDCOUNT` correctly computes the result of executing a `COUNT` query under the by-tuple/distribution semantics.

In Section III-A, the probability distribution for this example was computed by looking at the answer of each possible sequence of mappings assigned to individual tuples. If we have m mappings and n tuples, then the number of sequences is m^n. The algorithm presented here is polynomial in the number of mappings and tuples, and the number of computations is in $O(m * n^2)$.

Aggregate functions SUM and AVG

We now present efficient (PTIME) algorithms to compute the `SUM` aggregate under the by-tuple/range and by-tuple/expected value semantics. Computing this aggregate function under the distribution semantics does not scale, simply because the number of newly generated values may be exponential in the size of the original table, as was demonstrated at the beginning of Section IV-B.

`SUM` *Under the Range Semantics.* For the range semantics, we must compute the tightest interval in which the aggregate lies. The algorithm is presented in Figure 4 and illustrated next.

Consider Example 2, but now suppose we are interested in a simple computation of the sum of the prices for transactions whose auctionID is 34; we then use the following query:

Q2': `SELECT SUM(price) FROM T2`
`WHERE auctionID = '34'`

Table VI shows the trace of the algorithm in Figure 4 to answer query Q2'. If we look, for instance, at the second row in the table, processing tuple 2 from Table II, $v_2^{min} = 197.5$ and $v_2^{max} = 200$, thus $low = 392.5$ and $up = 395$. The answer to Q2' is thus $[1069.3, 1273]$. This algorithm is polynomial in the number of mappings and tuples, and the number of computations is in $O(m * n)$, where m is the number of mappings and n is the number of tuples.

Theorem 3: Algorithm `ByTupleRangeSUM` correctly computes the result of executing a `SUM` query under the by-tuple/range semantics.

`AVG` *Under the Range Semantics.* For the `AVG` aggregate operator, the algorithm we developed is very similar to the one in Figure 4, keeping a counter of the number of participating tuples for both the lower bound and the upper bound. The counter for the upper bound is incremented by one at each step only if there exists a maximum value for the tuple that satisfies the condition when some mapping is applied. The counter for the lower bound is incremented only if there is a minimum value for the tuple that satisfies the condition under some mapping. The answer is given by dividing each bound for `SUM` by the corresponding counter.

`SUM` *Under the Expected Value Semantics.* We now address an efficient way of computing by-tuple/expected value semantics. We do so not by giving an algorithm, but rather by showing that an answer to a `SUM` query using the by-tuple/expected value semantics is equivalent to its by-table counterpart. Before introducing this equivalence formally, we start with an illustrating example:

Example 5: Consider query Q2'. Using the by-table/expected value semantics, we consider two possible cases. Using m_{21} we map price to bid, with a query outcome of $195 + 200 + 331.94 + 349.99 = 1076.93$ and a probability of 0.3. Using m_{22} we map price to currentPrice, with a query outcome of $195 + 197.5 + 202.5 + 336.94 = 931.94$ and a probability of 0.7. Therefore, the answer to Q2', under the by-table/expected value semantics would be $1076.93 * 0.3 + 931.94 * 0.7 = 975.437$. Table VII presents

```
Algorithm ByTupleRangeSUM
Input: Table S, T; MapList M; Attribute A; Condition C
1. Let low = 0, up = 0;
2. For each t_i ∈ S,
3.    let v_i^min be the minimum value obtained by applying a mapping in M
      that satisfies condition C;
      Similarly, let v_i^max be the maximum value
      that satisfies condition C.
4.    low = low + v_i^min;
5.    up = up + v_i^max;
6. return [low, up];
```

Fig. 4. Algorithm to answer `SELECT SUM(A) FROM T WHERE C` under Range Semantics

the 16 different sequences and for each sequence it computes the query output, its probability, and the product of the two (which is a term in the summation defining expected value). The outcome of Q2' using the by-tuple/expected value semantics is identical to that of the by-table/expected value semantics. To see why, let us trace a single value, 434.99. This value appears in the fourth tuple and is used in the computation whenever a sequence contains mapping m_{21} for the fourth tuple, which is every other row in Table VII. Summing up the probabilities of all such worlds yields a probability of 0.3, which is exactly the probability of using m_{21} in the by-table semantics. The reason for this phenomenon is because the association of a mapping to one tuple is independent of the association with another tuple.

Example 5 explains the intuition underlying Theorem 4 below. It is worth noting that this solution does not extend to the AVG aggregate because it is a non-monotonic aggregate.

Theorem 4: Let $\overline{pM} = (S, T, \mathbf{m})$ be a schema p-mapping and let Q be a SUM query over attribute $A \in S$. The expected value of $Q^{tuple}(D_T)$, a *by-tuple* answer to Q with respect to $\overline{pM}$, is identical to $Q^{table}(D_T)$, a *by-table* answer to Q with respect to $\overline{pM}$.

Aggregate functions MAX and MIN

We now present an efficient algorithm to compute the MAX aggregate under the range semantics for the by-tuple semantics. The techniques presented here for MAX can be easily adapted for answering queries involving the MIN aggregate.

MAX *Under the Range semantics.* To compute MAX under the range semantics, we have to find the minimum and the maximum value of the aggregate under any possible mapping sequence, *i.e.*, the tightest interval that includes all the possible maximum values that can arise. The procedure to find this interval without the need to look at all possible sequences is outlined in Figure 5.

To see how this algorithm works, consider Example 2. We answer the subquery within the FROM clause of query Q2:

```
SELECT MAX(DISTINCT T2.price)
    FROM T2 AS R2 GROUP BY R2.auctionID
```

This subquery contains a GROUP BY auctionID, which means we will have one answer for each distinct auctionID. In this case, looking at Table II we see that the answer will consist of two different ranges, one for auctionID $= 34$ and another for auctionID $= 38$. We show how to compute the answer for auctionID $= 38$; the process to obtain the answer for auctionID $= 34$ is analogous. For tuple 5, with transactionID $= 3801$, the minimum value obtained by applying a mapping is $v_5^{min} = 300$, while the maximum is $v_5^{max} = 330.01$. For tuple 6, $v_6^{min} = 335.01$ and $v_6^{max} = 429.95$; for tuple 7, $v_7^{min} = 336.3$ and $v_7^{max} = 439.95$. Finally, for tuple 8, $v_8^{min} = 340.05$ and $v_8^{max} = 438.05$. Now, the range for the aggregator is given by $[\max_i\{v_i^{min}\}, \max_i\{v_i^{max}\}]$. Where each bound is computed as:

$$\max_i\{v_i^{min}\} = \max\{300, 335.01, 336.3, 340.05\}$$

$$\max_i\{v_i^{max}\} = \max\{330.01, 429.95, 439.95, 438.05\}$$

and thus, the final answer is $[340.05, 439.95]$. In general, it is always the case that the range yielded by the by-table semantics is a subset of the range yielded by the by-tuple semantics. This is because by-tuple has the possibility of choosing a different mapping for each tuple, which means that the algorithm has the freedom to choose sequences that are not allowed using the by-table semantics. This is true for all aggregate functions considered in this work. This algorithm also requires a polynomial number of computations in $O(m * n)$, where m is the number of mappings and n is the number of tuples.

Theorem 5: Algorithm ByTupleRangeMAX correctly computes the result of executing an MAX query under the by-tuple/range semantics.

C. Summary of Complexity Results

The tables in Figure 6 are a summary of our results for the six different kinds of semantics. The algorithms presented in this section correspond to those that require polynomial time to compute the answer.

V. Experimental Results

In order to evaluate the difference in the running times of our algorithms (both PTIME and non-PTIME), and how these are affected by changes in both the number of tuples in the database and the number of probabilistic mappings present, we carried out a series of empirical tests whose results we report in this section. The algorithms we gave for problems that were not shown to be PTIME are — as expected — inefficient. However, the algorithms we gave for problems we showed to

```
Algorithm ByTupleRangeMAX
Input: Table S, T; MapList M; Attribute A; Condition C
1. For each t_i ∈ S,
2.    let v_i^min be the minimum value obtained by applying a mapping in M
      that satisfies condition C;
      Similarly, let v_i^max be the maximum value.
3. return [max_i{v_i^min}, max_i{v_i^max}];
```

Fig. 5. Algorithm to answer `SELECT MAX(A) FROM T` under Range Semantics

COUNT	Range	Distribution	Expected Value
By-Table	PTIME	PTIME	PTIME
By-Tuple	PTIME	PTIME	PTIME

SUM	Range	Distribution	Expected Value
By-Table	PTIME	PTIME	PTIME
By-Tuple	PTIME	?	PTIME

MAX,MIN,AVG	Range	Probability Distribution	Expected Value
By-Table	PTIME	PTIME	PTIME
By-Tuple	PTIME	?	?

Fig. 6. Summary of complexity for the different aggregates

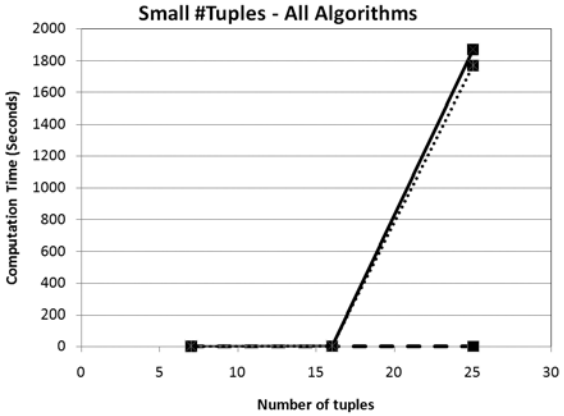

Fig. 7. Running times for variation of #tuples using the eBay data; #attributes = 7, #mappings = 2, results are averages over 5 runs on the eBay auction data. *Solid line*: ByTuplePDMAX. *Dotted line*: ByTupleExpValAVG, ByTuplePDAVG, ByTuplePDSUM, and ByTupleExpValMAX. *Dashed line (touching the x axis)*: ByTupleRangeMAX, ByTupleRangeCOUNT, ByTuplePDCOUNT, ByTupleExpValCOUNT, ByTupleRangeSUM, ByTupleExpValSUM, and ByTupleRangeAVG.)

be in PTIME are quite efficient when we vary both the number of tuples and the numbers of mappings — but clearly there are limits that vary from one algorithm to another. We will discuss these limits below.

The programs to carry out these tests consist of about 3,300 lines of Java code. All computations were carried out on a quad-processor computer with Intel Xeon 5140 dual core CPUs at 2.33GHz each, 4GB of RAM, under the CentOS GNU/Linux OS (distribution 2.6.9-55.ELsmp). The database engine we used was PostgreSQL version 7.4.16.

Experimental Setup. We carried out two sets of experiments. The first set used *real-world data* of 1,129 eBay 3-day auctions with a total of 155,688 bids for Intel, IBM, and Dell laptop computers. The data was obtained from an RSS feed for

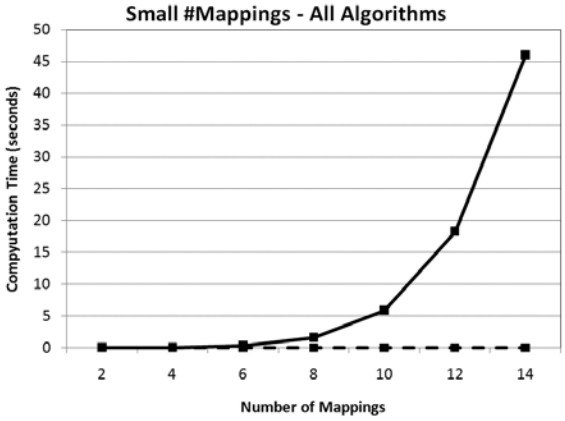

Fig. 8. Running times for variation of #mappings; #attributes = 20, #tuples = 6, results are averages over 5 runs on synthetic data. *Solid line*: ByTupleExpValAVG, ByTuplePDAVG, ByTuplePDSUM, ByTupleExpValMAX, and ByTuplePDMAX. *Dashed line (touching the x axis)*: ByTupleRangeMAX, ByTupleRangeCOUNT, ByTuplePDCOUNT, ByTupleExpValCOUNT, ByTupleRangeSUM, ByTupleExpValSUM, and ByTupleRangeAVG.

a search query on eBay.[2] The database schema is the one presented in Example 2. The sole point of uncertainty lies in the two price attributes where a reference to Price could mean either the bid price or the current price. We therefore defined two mappings: bid mapped to Price with probability 0.3 and currentPrice mapped to Price with probability 0.7. We have applied the inner query of query Q2 and also a set of queries that cover four different operators discussed in this work (all except `MIN`).

The second set of experiments was done on synthetic, randomly generated data in order to be able to evaluate configurations not possible with the eBay data (in particular, larger numbers of attributes, tuples, and mappings). The tables consist of attributes of type `real`, plus one column of type `int` used as id (not included in the number of attributes reported in the results). Mappings were also randomly generated by selecting an attribute at random and then a set of attributes that are mapped to it, also with a randomly chosen probability distribution. Each experiment was repeated several times.

Results. We now present and analyze the experiment results for small, medium, and large instances.

Small instances. We ran a set of experiments on small relations to compare the performance of all possible semantics, including those for which there are no PTIME algorithms.

[2] http://search.ebay.com/ws/search/

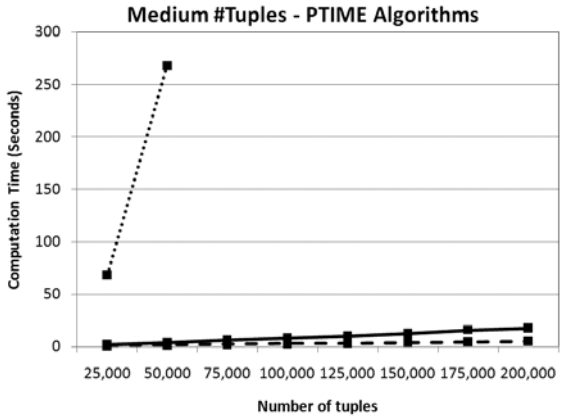

Fig. 9. Running times for variation of #tuples; #attributes = 50, #mappings = 20, results are averages over 5 runs on synthetic data. *Solid line*: ByTupleRangeAVG, ByTupleRangeSUM, ByTupleRangeCOUNT, and ByTupleRangeMAX. *Dashed line*: ByTupleExpValSUM. *Dotted line*: ByTuplePDCOUNT and ByTupleExpValCOUNT.

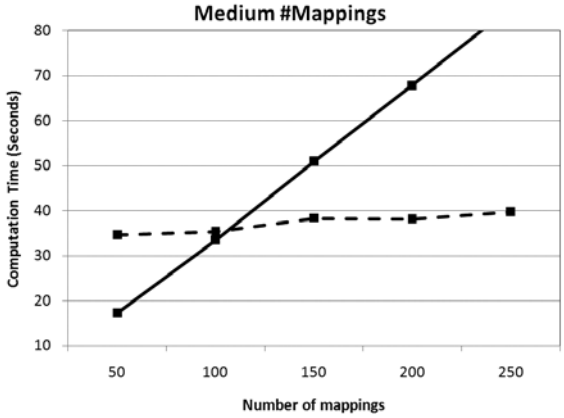

Fig. 10. Running times for variation of #mappings; #attributes = 500, #tuples = 50,000, results are averages over 2 runs on synthetic data. *Solid line*: ByTupleExpValSUM. *Dashed line*: ByTupleRangeMAX, ByTupleRangeCOUNT, ByTupleRangeSUM, and ByTupleRangeAVG.

Figures 7 and 8 show the running times of all algorithms on small instances (#mappings fixed at 2 in the former, #tuples fixed at 6 in the latter). The former corresponds to runs using the eBay auction data (results shown on a scatterplot, since each point corresponds to adding all tuples from an auction), while the latter reports results from runs on synthetic data.

As we can see, running times climb exponentially for algorithms we did not show to be in PTIME; the sharp increase in Figure 7 continues when more auctions are included, with a completion time of more than 10 days for 4 auctions (36 tuples). On the other hand, the running times of the other algorithms are negligible. When we varied #tuples, the by-table algorithms running times lay between 0.07 and 0.13 seconds. When we varied #mappings, the by-table algorithms took between 0.03 and 0.26 seconds. We also ran experiments varying #tuples using synthetic data which yielded the same trends in running times as those in Figure 7. These results are included in [21].

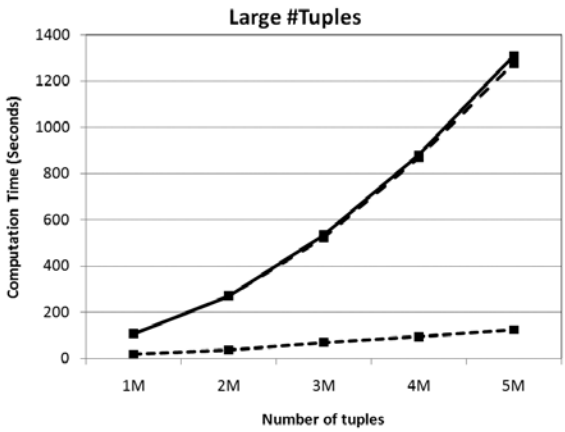

Fig. 11. Running times for variation of #tuples; #attributes = 50, #mappings = 20, results are averages over 2 runs on synthetic data. *Solid line*: ByTupleRangeMAX and ByTupleRangeAVG. *Dashed line*: ByTupleRangeSUM and ByTupleRangeCOUNT. *Dotted line*: ByTupleExpValSUM.

Medium-size instances. Figure 9 shows the running times of all our PTIME algorithms when the number of tuples is increased into the tens and hundreds of thousands (#mappings fixed at 20). As we can see, the `ByTuplePDCOUNT` and `ByTupleExpValCOUNT` algorithms' performance is well differentiated from the rest, as they become intractable at about 50,000 tuples. This is due to the fact that these algorithms must update the probability distribution for the possible values in each iteration, leading to a running time in $O(m*n^2)$ as shown in Section IV-B. In this case, the by-table algorithms' running times varied between 0.96 seconds and 5.49 seconds.

Figure 10 shows how the running times increase with the number of mappings (#tuples fixed at 50,000, #attributes = 500). It is interesting to note that `ByTupleExpValSUM` is more affected by the increase in number of mappings than the other four algorithms, with its running time climbing to almost 90 seconds for 250 mappings. This is because it is a by-table algorithm, and it must issue as many queries as mappings and then combine the answers. The other four, on the other hand, only slightly increase their running times at these numbers of mappings. The by-table algorithms' running times in this case lie between 16.49 and 86.49 seconds.

Large instances. Figure 11 shows how our most scalable by-tuple algorithms perform when the number of tuples is increased into the millions, showing that `ByTupleRangeMAX`, `ByTupleRangeCOUNT`, `ByTupleRangeAVG`, and `ByTupleRangeSUM` take about 1,300 seconds (about 21 minutes) to answer queries with 5 million tuples and 20 mappings. This figure also shows the running time of `ByTupleExpValSUM`, which is much lower than the others because it is actually equivalent to the by-table algorithm, as seen in Section IV-B. The corresponding running times for the by-table algorithms varied between 15.73 and 125.63 seconds. We also ran experiments for 15 to 30 million tuples, the results of which are shown in Figure 12. For these runs, the by-table algorithms took between 65.17 seconds and 124.76 seconds.

It should be noted that the greater scalability of the by-table

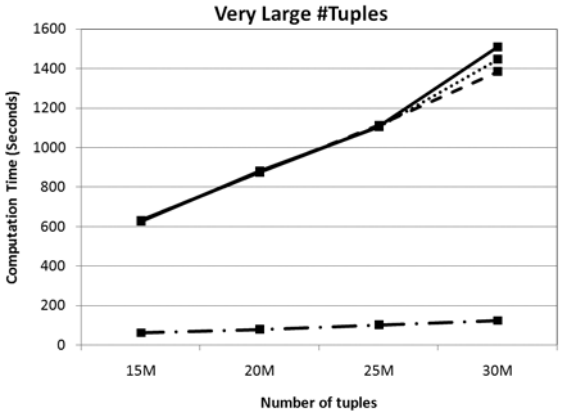

Fig. 12. Running times for variation of #tuples; #attributes = 20, #mappings = 5, results are averages over 2 runs on synthetic data. *Solid line*: ByTupleRangeCOUNT. *Dotted line*: ByTupleRangeSUM and ByTupleRangeAVG. *Dashed line*: ByTupleRangeMAX. *Dashed and Dotted line*: ByTupleExpValSUM.

algorithms with respect to the efficient by-tuple algorithms presented here is in large part due to the fact that the former are taking advantage of the optimizations implemented by the DBMS when answering queries.

VI. Related Work

Uncertainty in the process of data integration is gaining attention in the research community. [22] argues for the need "to incorporate *inaccurate* mappings and handle *uncertainty* about mappings. Inaccuracy arises because in many contexts there is no precise mapping... mappings may be inaccurate [since] the mappings language is too restricted to express more accurate mappings." [23] went even further, arguing philosophically that even if two schemata fully agree on the semantics and the language is rich enough, schemata may still not convey the same meaning, due to some hidden semantics, beyond the scope of the schemata. Therefore, [22] argues that "when no accurate mapping exists, the issue becomes one of choosing the *best* mapping from the viable ones." The latter approach was later extended to handle multiple schema matchings in parallel [24], [12], [25], [11].

One aspect of such uncertainty stems from the quality of the underlying schema matchings. Uncertainty management arose at the core of data integration with the advent of new approaches to data management, such as dataspaces [26]. Researchers argue for moving to fully-automatic (that is, unsupervised) schema matching in numerous emerging applications triggered by the vision of the Semantic Web and machine-understandable Web resources.

Several recent works were devoted to the parallel use of alternative schema matchings [12], [27], [11]. In this work, we extend the work in [11] to handle aggregates. In [13], a mechanism for generating a set of probabilistic mediator schemas was proposed. [28] provides a mechanism for generating the top-K probabilistic schema matchings. Here, we assume a set of probabilistic schema matchings is given through an existing algorithm such as one of those mentioned above.

Aggregate queries over imprecise and uncertain data have been studied in [29], [30], [31], [32], [33]. The work of Afrati and Kolaitis [34] focuses on answering aggregate queries in data exchange, which is closely related to our setting of data integration. The authors provide a different range semantics, based on glb and lub operators, under which aggregates can be computed in PTIME, which is a similar result to our version of the range semantics. The authors also propose a possible values semantics that is similar to our distribution semantics but is not probabilistic. Their NP-completeness result for a value existence decision problem fits well with our assertion that in general, aggregates under the by tuple/probability distribution semantics cannot be computed in PTIME. Our work is unique in its setting, involving probabilistic schema matching and the by-table and by-tuple semantics. For example, in [33], tuples are assigned an allocation measure, which is similar to our probability assignment based on probabilistic schema matchings. They also promote the use of expected value as a measure that maintains "faithfulness." However, under their setting, computing an expected value of the `AVG` operator can be done in PTIME using a polynomial number of passes on the set of tuples. Under the by-tuple/expected semantics such a computation is not feasible.

[14] introduces the concept of probabilistic aggregates without independence assumptions based on the distribution semantics of this paper. They provide a linear programming approach for computing aggregates over probabilistic DBMSs, develop algorithms under this approach, analyze their complexity, and introduce several families of approximation algorithms that run in polynomial time. [15] studies the problem of computing aggregate operators on probabilistic data in an I/O efficient manner. Under their assumptions, the algorithms for `SUM` and `COUNT` are simple, whereas we show that for the by-tuple/distribution semantics computing `SUM` may yield an answer that is exponential in the original table size. For `AVG` and `MIN/MAX` the authors provide stream approximations. We show efficient algorithms for the by-table semantics and the by-tuple/range semantics.

VII. Conclusions and Future Work

Probabilistic schema matching is emerging as a paradigm for integrating information from multiple databases. In past work, [11] has proposed two semantics called the by-table and by-tuple semantics for selection, projection and join query processing under probabilistic schema mapping.

In this paper, we study the problem of answering aggregate queries in such an environment. We present three semantics for aggregates — a range semantics in which a range of possible values for an aggregate query is returned, a probability distribution semantics in which all possible answer values are returned together with their probabilities, and an expected value semantics. These three semantics combine together with the semantics of [11] to provide six possible semantics for aggregates. Given this setting, we provide algorithms to answer `COUNT`, `SUM`, `AVG`, `MIN`, and `MAX` aggregate queries. We develop algorithms for each of these five aggregate operators

under each of the six semantics. The good news is that for every aggregate operator, at least one (and sometimes more) semantics are PTIME computable.

We also report on a prototype implementation and experiments with two data sets — a real world eBay data set, and a synthetic data set. We experimentally show that each aggregate operator listed above can be computed efficiently in at least one of these six semantics, even when the data set is large and there are many different possible schema mappings.

In our future work we intend to extend our system to support nested aggregate queries by interpreting the results on inner queries in terms of probablistic databases. We shall also investigate methods for optimizing some of our algorithms, including the by-tuple/range semantics of `COUNT` and `SUM`. Finally, we shall investigate sampling methods to provide efficient answers to `MIN`, `MAX`, and `AVG` under the by-tuple/distribution semantics.

VIII. Acknowledgments

The authors gratefully acknowledge funding support for this work provided by the AFOSR through the Laboratory for Computational Cultural Dynamics (LCCD) under grants FA95500610405 and FA95500510298, and the NSF under grant 0540216. Any opinions, findings or recommendations in this document are those of the authors and do not necessarily reflect the views of sponsors.

References

[1] W.-S. Li and C. Clifton, "SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks," *Data & Knowledge Engineering*, vol. 33, no. 1, pp. 49–84, 2000.

[2] E. Mena, V. Kashayap, A. Illarramendi, and A. Sheth, "Imprecise answers in distributed environments: Estimation of information loss for multi-ontological based query processing," *International Journal of Cooperative Information Systems*, vol. 9, no. 4, pp. 403–425, 2000.

[3] R. Miller, M. Hernàndez, L. Haas, L.-L. Yan, C. Ho, R. Fagin, and L. Popa, "The Clio project: Managing heterogeneity," *SIGMOD Record*, vol. 30, no. 1, pp. 78–83, 2001.

[4] A. Doan, N. Noy, and A. Halevy, "Introduction to the special issue on semantic integration." *SIGMOD Record*, vol. 33, no. 4, pp. 11–13, 2004.

[5] H. He, W. Meng, C. Yu, and Z. Wu, "Wise-integrator: A system for extracting and integrating complex web search interfaces of the deep web." in *Proceedings of the International conference on Very Large Data Bases (VLDB)*, 2005, pp. 1314–1317.

[6] M. Ehrig, S. Staab, and Y. Sure, "Bootstrapping ontology alignment methods with apfel." in *Proceedings of ISWC 2005, 4th International Semantic Web Conference, ISWC 2005*, 2005, pp. 186–200.

[7] A. Gal, "Why is schema matching tough and what can we do about it?" *SIGMOD Record*, vol. 35, no. 4, pp. 2–5, 2007.

[8] A. Bonifati, G. Mecca, A. Pappalardo, S. Raunich, and G. Summa, "Schema mapping verification: The spicy way," in *EDBT*, 2008.

[9] X. Chai, M. Sayyadian, A. Doan, A. Rosenthal, and L. Seligman, "Analyzing and revising mediated schemas to improve their matchability," in *Proceedings of the International conference on Very Large Data Bases (VLDB)*, Auckland, New Zealand, Aug. 2008.

[10] N. Bozovic and V. Vassalos, "Two-phase schema matching in real world relational databases," in *Proceedings of ICDE 2008*, 2008, pp. 290–296.

[11] X. L. Dong, A. Y. Halevy, and C. Yu, "Data integration with uncertainty," in *VLDB*, C. Koch, J. Gehrke, M. N. Garofalakis, D. Srivastava, K. Aberer, A. Deshpande, D. Florescu, C. Y. Chan, V. Ganti, C.-C. Kanne, W. Klas, and E. J. Neuhold, Eds. ACM, 2007, pp. 687–698.

[12] A. Gal, "Managing uncertainty in schema matching with top-k schema mappings," *Journal of Data Semantics*, vol. 6, pp. 90–114, 2006.

[13] A. D. Sarma, X. Dong, and A. Halevy, "Bootstrapping pay-as-you-go data integration systems," in *Proceedings of the ACM-SIGMOD conference on Management of Data (SIGMOD)*, 2008, pp. 861–874.

[14] R. Ross, V. Subrahmanian, and J. Grant, "Aggregate operators in probabilistic databases," *J. of the ACM*, vol. 52, no. 1, pp. 54–101, 2005.

[15] T. Jayram, S. Kale, and E. Vee, "Efficient aggregation algorithms for probabilistic data," in *Proceedings of SODA 2007*, New Orleans, Louisiana, USA, Jan. 2007, pp. 346–355.

[16] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. Miller, and J. Mylopoulos, "The hyperion project: From data integration to data coordination," *SIGMOD Record*, vol. 32, no. 3, 2003.

[17] A. Halevy, Z. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov, "The Piazza peer data management system," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 16, no. 7, pp. 787–798, 2004.

[18] L. Gravano, P. Ipeirotis, N. Koudas, and D. Srivastava, "Text joins for data cleansing and integration in an rdbms," in *Proceedings of the IEEE CS International Conference on Data Engineering*, 2003, pp. 729–731.

[19] A. Inan, M. Kantarcioglu, E. Bertino, and M. Scannapieco, "A hybrid approach to private record linkage," in *Proceedings of the IEEE CS International Conference on Data Engineering*, 2008, pp. 496–505.

[20] R. Miller, L. Haas, and M. Hernández, "Schema mapping as query discovery," in *Proceedings of the International conference on Very Large Data Bases (VLDB)*, A. E. Abbadi, M. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K.-Y. Whang, Eds. Morgan Kaufmann, 2000, pp. 77–88.

[21] A. Gal, M. V. Martinez, G. I. Simari, and V. Subrahmanian, "Aggregate query answering under uncertain schema mappings," University of Maryland College Park, Technical Report 2008-06-27, 2008, http://www.cs.umd.edu/∼mvm/GalTechReport.pdf.

[22] J. Madhavan, P. Bernstein, P. Domingos, and A. Halevy, "Representing and reasoning about mappings between domain models," in *Proceedings of AAAI/IAAI 2002*, 2002, pp. 80–86.

[23] M. Benerecetti, P. Bouquet, and S. Zanobini, "Soundness of schema matching methods," in *Proceedings of ESWC 2005*, 2005, pp. 211–225.

[24] A. Gal, A. Anaby-Tavor, A. Trombetta, and D. Montesi, "A framework for modeling and evaluating automatic semantic reconciliation," *VLDB Journal*, vol. 14, no. 1, pp. 50–67, 2005.

[25] C. Domshlak, A. Gal, and H. Roitman, "Rank aggregation for automatic schema matching," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 19, no. 4, pp. 538–553, 2007.

[26] M. Franklin, A. Halevy, and D. Maier, "From databases to dataspaces: a new abstraction for information management." *SIGMOD Record*, vol. 34, no. 4, pp. 27–33, 2005.

[27] Y. Qi, K. Candan, and M. Sapino, "FICSR: *F*eedback-based *I*n*C*on*S*istency *R*esolution and query processing on misaligned data sources," in *Proceedings of the ACM-SIGMOD conference on Management of Data (SIGMOD)*, 2007, pp. 151–162.

[28] H. Roitman, A. Gal, and C. Domshlak, "Providing top-k alternative schema matchings with ontomatcher," in *Proceedings of the International Conference on Conceptual Modeling (ER)*, 2008.

[29] E. Rundensteiner and L. Bic, "Evaluating aggregates in possibilistic relational databases," *Data & Knowledge Engineering*, vol. 7, pp. 239–267, 1991.

[30] A. Chen, J.-S. Chiu, and F.-C. Tseng, "Evaluating aggregate operations over imprecise data," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 8, no. 2, pp. 273–284, 1996.

[31] S. McClean, B. Scotney, and M. Shapcott, "Aggregation of imprecise and uncertain information in databases," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 13, no. 6, pp. 902–912, 2001.

[32] R. Cheng, D. Kalashnikov, and S. Prabhakar, "Evaluating probabilistic queries over imprecise data," in *Proceedings of the ACM-SIGMOD conference on Management of Data (SIGMOD)*, 2003, pp. 551–562.

[33] D. Burdick, P. Deshpande, T. Jayram, R. Ramakrishnan, and S. Vaithyanathan, "Olap over uncertain and imprecise data," in *Proceedings of the International conference on Very Large Data Bases (VLDB)*, 2005, pp. 970–981.

[34] F. Afrati and P. G. Kolaitis, "Answering aggregate queries in data exchange," in *PODS '08: Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. New York, NY, USA: ACM, 2008, pp. 129–138.