Efficient Top-k Query Evaluation on Probabilistic Data

Extended Version

TR: #2006-06-05

Christopher Ré, Nilesh Dalvi and Dan Suciu
University of Washington
Dept. of Computer Science and Engineering

E-mail: {`chrisre`,`nilesh`,`suciu`}`@cs.washington.edu`

Abstract

Modern enterprise applications are forced to deal with unreliable, inconsistent and imprecise information. Probabilistic databases can model such data naturally, but SQL query evaluation on probabilistic databases is difficult: previous approaches have either restricted the SQL queries, or computed approximate probabilities, or did not scale, and it was shown recently that precise query evaluation is theoretically hard. In this paper we describe a novel approach, which computes and ranks efficiently the top-k answers to a SQL query on a probabilistic database. The restriction to top-k answers is natural, since imprecisions in the data often lead to a large number of answers of low quality, and users are interested only in the answers with the highest probabilities. The idea in our algorithm is to run in parallel several Monte-Carlo simulations, one for each candidate answer, and approximate each probability only to the extent needed to compute correctly the top-k answers. The algorithms is in a certain sense provably optimal and scales to large databases: we have measured running times of 5 to 50 seconds for complex SQL queries over a large database (10M tuples of which 6M probabilistic). Additional contributions of the paper include several optimization techniques, and a simple data model for probabilistic data that achieves completeness by using SQL views.

1 Introduction

A number of applications today need to manage data that is imprecise. For example imprecisions arise in fuzzy object matching across multiple databases, in data extracted automatically from unstructured text, in automatic schema alignments, in sensor data, in activity recognition data. In some cases it is possible to eliminate the imprecisions completely, but this is usually very costly, like manual removal of ambiguous matches in data cleaning; in other cases complete removal is not even possible, e.g. in human activity recognition. Modern enterprise applications are forced to deal with unreliable and imprecise information, but they can often tolerate such imprecisions, especially in applications like *search* or *business intelligence*. A system that tolerates imprecisions needs to be able rank query results based on the degree of their uncertainty. Our goal is to develop techniques to manage automatically imprecisions in the data and rank query answers.

For that purpose we use a *probabilistic database*. A simplistic definition is that every tuple belongs to the database with some probability, whose value is between 0 and 1. Every tuple is thus a probabilistic event and tuples may be correlated events. The major difficulty in probabilistic databases consist of evaluating the queries correctly and efficiently. Dalvi and Suciu [4] have shown recently that most SQL queries have #P-complete data complexity, which rules out efficient algorithms for exact probabilities.

Our approach in this paper is to combine top-k style queries with approximate algorithms for computing the probabilities. When managing imprecisions in data, the most meaningful information lies not in the exact values of the output probabilities but in the ranking of the queries' answers. Thus, we shift the focus from computing the

```
AMZNReviews(asin, title, customer, rating, ...)
AMZNDirector(asin, director)
AMZNActor(asin, actor)
IMDBMovie(mid, movieTitle, genre, did, year)
IMDBDirector(did, dirName)
IMDBCast(mid, aid)
IMDBActor(aid, actorName)
TitleMatch^p(asin, mid, p)
```

Figure 1. Schema fragment of IMDB and Amazon database, and a fuzzy match table

output probabilities to finding and ordering the top k answers. Restricting to just the first k answers is justified in the context of dealing with imprecisions in data, since here many tuples in the query answer are of very low quality (probability), and users are interested in seeing only the most highly ranked answers. Under this approach, we develop a novel query evaluation algorithm for computing the top-k answers with provable theoretical guarantees.

Thus, the problem we address here is the following. We are given a SQL query and a number k, and we have to return to the user the k highest ranked answers sorted by their output probabilities. To compute the probabilities we use Luby and Karp's Monte Carlo simulation algorithm[1] [13] (MC), which can compute an approximation to any desired precision. A naive application of MC would be to run it a sufficiently large number of steps on each query answer and compute its probability with high precision, then sort the answers and return the top k. In contrast, we describe here an algorithm called *multisimulation* (MS), which concentrates the simulation steps on the top k answers, and only simulates the others a sufficient number of steps to ensure that they are not in the top k. We prove that MS is theoretically optimal in a very strong sense: it is within a factor of two of a non-deterministic optimal algorithm, which magically "knows" how many steps to simulate each answer, and *no other* deterministic algorithm can be better. There are three variations of MS: computing the set of top k answers, computing and sorting the set of top k answers, and an *any time* algorithm, which outputs the answers in the order $1, 2, 3, \ldots, k, \ldots$ and can be stopped at any time. Our experiments show that MS exploits gracefully k (the running times are essentially linear in k) and that MS is dramatically more efficient than a naive application of MC.

We make two additional contributions in this paper. The first is to complement MS with two optimizations. One pushes some of the probabilistic processing to the query engine, when it is safe to do so: for example independent probabilities can be multiplied, while disjoint probabilities can be added, and both can be done in the SQL engine. We give precise conditions under which such computations can be pushed in the engine. The other optimization prunes the number of candidate answers that need to be simulated by computing a lower and an upper bound for each probability. Both optimizations *provably correct*, i.e. they preserve the probabilistic semantics of the queries.

The second additional contribution is to show that by adding SQL views to probabilistic tables one obtains a representation formalism that is *complete* for possible worlds semantics. In this formalism the database consists of some probabilistic tables (similar to those in Barbara et al. [2]) plus some SQL views over probabilistic and/or deterministic tables. A prerequisite to such a representation system is a query processor that can evaluate efficiently complex SQL queries over probabilistic databases, because such queries arise inevitably when the views are expanded in the user queries; hence, this complete representation system is a direct application of the query evaluation method described in this paper.

1.1 Challenges

We illustrate here the challenges faced by query evaluation on probabilistic databases with an application that integrates the Internet Movie Database from `imdb.com`, with movie reviews from `amazon.com`; there are over 10M tuples in the integrated database. A simplified schema is shown in Fig. 1, and we will use it as our running example in the paper. Amazon products (DVDs in our case) are identified by a unique Amazon Standard Identification Number, `asin`, and each DVD object has several subobjects: customer reviews, actors, director, etc. The IMDB schema is self explanatory. The value of integrating the two data sources lies in combining the detailed movie data in IMDB with customers ratings in AMZN.

[1]Any other approximation algorithms can be substituted.

`TitleMatch`^{p}

	`asin`	`mid`	`p`
t_1	a282	m897 (“Twelve Monkeys”)	0.4
t_2	(“12 Monkeys”)	m389 (“Twelve Monkeys (1995)”)	0.3
t_3		m656 (“Monk”)	0.013
t_4	a845	m897 (“Twelve Monkeys”)	0.35
t_5	(“Mokey Love”)	m845 (“Love Story”)	0.27

Figure 2. Some fuzzy matches in `TitleMatch`^{p}. The table stores only the `asin` and `mid` values, but we included the review tile and movie title for readability.

From Imprecisions to Probabilities One source of imprecision in integrating the two sources is that their movie titles often don't match, e.g. `Twelve Monkeys` v.s. `12 Monkeys` or `Who Done it?` v.s. `The Three Stooges: Who Done it`. The problem of detecting when two representations denote the same object has been intensively studied, and referred to as deduplication, record linkage, or merge-purge [18, 19, 10, 7, 11, 9, 1, 3]. Perfect object matching is sometimes impossible, and when it is possible it is often very costly, since it requires specialized, domain specific algorithms. Our approach is to rely on existing domain independent methods, and change the way their outputs are used. Currently, all fuzzy match methods use a *thresholded similarity function approach* [1], which relies on a threshold value to classify objects into matches and non-matches. This is a compromise that can lead to false positives (when the threshold value is too low) or to false negatives (when it's too high). In contrast, in our approach the system retains all similarity scores and handles them as probabilistic data. We computed a similarity score between each pair of movie title and review title by comparing their sets of 3-grams: this resulted in a number `p` between 0 and 1, which we interpret as the confidence score and stored it in a table called `TitleMatch`^{p}. Fig. 2 shows a very simplified fragment of `TitleMatch`^{p}, consisting of five tuples $t_1, \dots, t_5$. Each tuple contains an `asin` value (a review in `amazon`) and a `mid` value (a movie in `IMDB`). The amazon review with `asin = a282` refers to a movie with title `12 Monkeys`, and this can be one of three movies in the IMDB database: either `Twelve Monkeys`, or to `Twelve Monkeys (1995)`, or to `Monk`. Thus, only one of the tuples t_1, t_2, t_3 can be correct, i.e. they are *exclusive*, or *disjoint*, and their probabilities are $p_1 = 0.4$, $p_2 = 0.3$, and $p_3 = 0.013$ respectively. Note that $p_1 + p_2 + p_3 \leq 1$, which is a necessary condition since the three tuples are exclusive events: we normalized the similarity scores to enforce this condition. Similarly, the movie review about `Monkey Love` can refer to one of two IMDB movies, with probabilities $p_4 = 0.35$ and $p_5 = 0.27$ respectively. We assume that any of the three matches for the first review is independent from any of the two matches of the second review. This summarizes how we mapped fuzzy object matches to probabilities. We will discuss briefly other types of imprecisions in Sec. 6.1.

Possible Worlds Semantics The table `TitleMatch`^{p} is only a representation of a probabilistic database: the superscript p indicates that it is a representation, and that it contains explicit probabilities. Its *meaning* is a probability distribution on possible instances (*possible worlds*) over a table `TitleMatch(`<u>`asin`</u>`, mid)`. A possible instance consists of a subset of the tuples in `TitleMatch`^{p} that does not contain two tuples with the same `asin`, and its probability is computed in a natural fashion. For example, the set of tuples $\{t_2, t_4\}$ is one possible world, and its probability is $\mathbf{P}(\{t_2, t_4\}) = p_2 p_4 = 0.3 \times 0.35 = 0.105$, while the probability of $\{t_2\}$ is $\mathbf{P}(\{t_2\}) = p_2(1 - p_4 - p_5) = 0.3{*}0.48 = 0.144$. In our example there are $4{*}3 = 12$ possible worlds that can be obtained from t_1, t_2, t_3, t_4, t_5. Note that `asin` is a key in `TitleMatch`, but is not a key in `TitleMatch`^{p}, since we need to store multiple possible matches for each `asin`.

SQL Queries We consider SQL queries using standard syntax but with a modified semantics. To illustrate, consider the query “find all directors that produced both a highly rated comedy and a low rated drama less than five years apart”, whose SQL expression is in Fig. 3. Since it is evaluated on a probabilistic database each of its answers has a confidence score `p`, as show in Fig. 4 which lists the 5 answers with the highest probabilities, out of a total of 1415 answers. The semantics of these probabilities is as follows. The probability of the answer `Woody Allen` is the sum of the probabilities of all possible worlds in which `Woody Allen` is an answer to the query. Thus, the probability score represents our confidence that `Woody Allen` is an answer to the query, given the imprecisions in the data. Since the input probabilities were computed using heuristics, there is very little semantics in the actual values of these probabilities, however, the *ranking* of the results is meaningful, e.g. `Woody Allen` is a more likely answer than `George Cukor`, who is a more likely answer than, say, the 1000'th ranked answer.

```
SELECT DISTINCT d.dirName AS Director
FROM AMZNReviews a, AMZNReviews b,
  TitleMatch ax, TitleMatch by,
  IMDBMovie x, IMDBMovie y,
  IMDBDirector d
WHERE a.asin=ax.asin and b.asin=by.asin
  and ax.mid=x.mid and by.mid=y.mid
  and x.did=y.did and y.did=d.did
  and x.genre='comedy' and y.genre='drama'
  and abs(x.year - y.year) <= 5
  and a.rating>4 and b.rating<2
```

Figure 3. Query retrieving all directors that produced both a highly rated comedy and a low rated drama less than five years apart.

Rank	Director	p
1	Woody Allen	0.9998
2	Ralph Senensky	0.715017
3	Fred Olen Ray	0.701627
4	George Cukor	0.665626
5	Stewart Raffill	0.645483
	...	...

Figure 4. Top 5 query answers, out of 1415

Challenges Query evaluation poses two major challenges. The first is that computing the exact output probabilities is computationally hard. The data complexity for the query in Fig.3 is #P-complete (this can be shown using results in [4]), meaning that any algorithm computing the probabilities exactly essentially needs to iterate through all possible worlds. Previous work on probabilistic databases avoided this issue in several ways. Barbara [2] requires the SQL queries to include all keys in all tables, thus disallowing duplicate elimination. This rules out our query in Fig. 3, because this query does not include any of the keys of the seven tables in the `FROM` clause. If we included all these keys in the `SELECT` clause, then each director in the answer would be listed multiple times, once for each pair of movies that satisfies the criterion: in our example each of the 1415 directors would have occurred on average 234.8 times. This makes it impossible to rank the directors. Lakshmanan [14] computes probability intervals instead of exact probabilities. However, unlike Luby and Karp's algorithm which can approximate the probabilities to an arbitrary precision, the precision in [14] cannot be controlled. In fact, the more complex the query, the wider the approximation intervals end up, since the strategies in [14] have to conservatively account for a wide range of possible correlations between the input probabilities. For example, when combining the (on average) 234.8 different probabilities to compute the probability of a single director, the resulting interval degenerates to $[0, 1]$ for most directors. One can still use this method in order to rank the outputs, (by ordering them based on their intervals' midpoints) but this results in low precision. Fuhr [8] uses an exponential time algorithm that essentially iterates over all possible worlds that support a given answer. This is again impractical in our setting. Finally Dalvi [4] only considers "safe" queries, while our query is not safe.

The second challenge is that the number of potential answers for which we need to compute the probabilities is large: in our example, there are 1415 such answers. Many of them have very low probability, and exists only because of some highly unlikely matches between movies and reviews. Even if the system spends large amounts of time computing all 1415 probabilities precisely, the users is likely to end up inspecting just the first few of them.

Our Approach focuses the computation on the top k answers with the highest probabilities. A naive way to find the top k probabilities is to compute all probabilities then select the top k. Instead, we approximate probabilities only to the degree needed to guarantee that (a) the top k answers are the correct ones, and (b) the ranking of these top k answers is correct. In our running example, we will run an approximation algorithm for many steps on the, say, top $k = 10$ answers, in order to identify them and rank them, but will run only a few steps on the remaining $1415 - 10 = 1405$ answers, and approximate their probabilities only as much as needed to guarantee that they are not in the top 10. This turns out to be orders of magnitude more efficient than the naive approach. The major challenge is that we don't know which tuples are in top 10 before we know their probabilities: the solution to this is the main contribution in this paper.

Limitations In this paper we restrict to a data model where probabilities are listed explicitly. For example, if one has a `Person` table with `salary` and `age` attributes whose values are correlated probability distributions, then in our model one needs to enumerate explicitly all combinations of `salary` and `age`, e.g. `(Smith, 20, 1000, 0.3)`, `(Smith, 20, 5000, 0.1)`, `(Smith, 40, 5000, 0.6)`. This allows for correlated attributes as long as the joint distribution is represented explicitly. In contrast, Bayesian Networks, and their extensions to Probabilistic Relational Models allow such correlations to be expressed much more concisely. We also do not handle continuous attribute values, e.g. we cannot handle the case where the attribute `temperature` has a normal distribution with mean 40, as done in [6].

Organization Sec. 2 describes the basic probabilistic data model; the MS algorithm is described in Sections 3 and the two optimizations are in Sec. 4. We extend the data model in Sec. 5, report experiments in Sec. 6, and conclude in Sec. 7.

2 Background

2.1 Probabilistic Databases

We introduce here a basic probabilistic data model and will extend it in Sec. 5.

Possible Worlds Fix a relational schema S, consisting of relation names $R_1, R_2, \ldots, R_m$, a set of attributes $Attr(R_i)$ and a key $Key(R_i) \subseteq Attr(R_i)$ for each relation name R_i, $i = 1, m$. We define a probabilistic database to be a probability distribution on possible worlds over S.

Definition 2.1. *A probabilistic database over schema S is a pair $(\mathcal{W}, \mathbf{P})$ where $\mathcal{W} = \{W_1, \ldots, W_n\}$ is a set of database instances over S, and $\mathbf{P} : \mathcal{W} \to [0, 1]$ is a probability distribution (i.e. $\sum_{j=1,n} \mathbf{P}(W_j) = 1$). Each instance W_j for which $\mathbf{P}(W_j) > 0$ is called a possible world.*

The intuition is that the exact state of the database is uncertain: we have several possible instances, and for each such instance we have a probability.

Representation Of course, it is impractical to enumerate all possible worlds and their probabilities. We represent a probabilistic database by using a modified schema S^p, called *probabilistic schema*, which contains explicit probabilities (hence the superscript p). S^p consists of modified relation names $R_1^p, \ldots, R_m^p$, such that for each $i = 1, m$: (1) either R_i^p has the same schema as R_i (in this case R_i^p is deterministic), or (2) $Attr(R_i^p) = Attr(R_i) \cup \{p\}$, and $Key(R_i^p) = Attr(R_i)$. In the second case R_i^p is probabilistic, and we impose the following two constraints on its p attribute: (a) the values are real numbers in $[0, 1]$, (b) for every values $\bar{a}$ of the $Key(R_i)$ attributes, $sum(\Pi_p(\sigma_{Key(R_i)=\bar{a}}(R_i^p))) \leq 1$. We define next (following [4]) how a database instance J^p over schema S^p represents a probabilistic database over the schema S, denoted $Mod(J^p)$. To simplify the discussion we assume that S consists of a single relation name, $R(\underline{A_1, \ldots, A_m}, B_1, \ldots, B_n)$, in notation $R(\underline{\bar{A}}, \bar{B})$ (here $Key(R) = \{A_1, \ldots, A_m\} = \bar{A}$), and consider an instance J^p of the table $R^p(\bar{A}, \bar{B}, p)$. Note that the key in R^p consists of all attributes $\bar{A}, \bar{B}$, not just $\bar{A}$. We define the possible worlds $\mathcal{W} = \{W_1, \ldots, W_n\}$ to consists of all subsets W_j of $\Pi_{\bar{A},\bar{B}}(J^p)$ where the attributes $\bar{A}$ form a key. For each such W_j define its probability $\mathbf{P}(W_j) = \prod_{\bar{a} \in \Pi_{\bar{A}}(J^p)} p_{W_j}(\bar{a})$, where $p_{W_j}(\bar{a})$ is defined as follows. If there exists a tuple $(\bar{a}, \bar{b}) \in W_j$ then $p_{W_j}(\bar{a})$ is the unique probability p of that tuple in J^p (i.e. p is such that $(\bar{a}, \bar{b}, p) \in J^p$). If there is no such tuple, then $p_{W_j}(\bar{a})$ is $1 - sum(\Pi_p(\sigma_{\bar{A}=\bar{a}}(J^p)))$.

Definition 2.2. *Let J^p be a database instance over schema S^p. Then $Mod(J^p)$ is the probabilistic database $(\mathcal{W}, \mathbf{P})$ over the schema S obtained as described above.*

Example 2.3 For a simple illustration, consider the schema S^p in Fig. 1. All tables `AMZNReviews`, `IMDBDirector`, ... are deterministic, except for `TitleMatch`^{p}`(asin, mid, p)`, which represents possible worlds for `TitleMatch(`<u>`asin`</u>`, mid)`. The instance of `TitleMatch`^{p} (same as Fig. 2) and its 12 possible worlds are illustrated Fig. 5. The restrictions are:
$p_1, \ldots, p_5 \in [0, 1]$, $p_1 + p_2 + p_3 \leq 1$, $p_4 + p_5 \leq 1$.

$\texttt{TitleMatch}^p$

	asin	mid	p
t_1	a282	m897	p_1
t_2	a282	m389	p_2
t_3	a282	m656	p_3
t_4	a845	m897	p_4
t_5	a845	m845	p_5

$Mod(\texttt{TitleMatch}^p)$:

i	W_i	$\mathbf{P}(W_i)$
1	$\emptyset$	(1-p_1-p_2-p_3)(1-p_4-p_5)
2	t_1	p_1(1-p_4-p_5)
3	t_2	p_2(1-p_4-p_5)
4	t_3	p_3(1-p_4-p_5)
5	t_4	(1-p_1-p_2-p_3)p_4
6	t_1t_4	p_1p_4
7	t_2t_4	p_2p_4
8	t_3t_4	p_3p_4
9	t_5	(1-p_1-p_2-p_3)p_5
10	t_1t_5	p_1p_5
11	t_2t_5	p_2p_5
12	t_3t_5	p_3p_5

Figure 5. Illustration for Example 2.3

DNF Formulas over Tuples Let $(\mathcal{W}, \mathbf{P})$ be a probabilistic database and let $t_1, t_2, \ldots$ be all the tuples in all possible worlds. We interpret each tuple as a boolean propositional variable, and each possible world W as a truth assignment to these propositional variables, as follows: $t_i =$ true if $t_i \in W$, and $t_i =$ false if $t_i \notin W$. Consider now a DNF formula E over tuples: clearly E is true in some worlds and false in others. Define its probability $\mathbf{P}(E)$ to be the sum of $\mathbf{P}(W)$ for all worlds W where E true. Continuing our example, the expression $E = (t_1 \wedge t_5) \vee t_2$ is true in the possible worlds W_3, W_7, W_{10}, W_{11}, and its probability is thus $\mathbf{P}(E) = \mathbf{P}(W_3) + \mathbf{P}(W_7) + \mathbf{P}(W_{10}) + \mathbf{P}(W_{11})$.

2.2 Queries

Syntax We consider SQL queries over the schema S:

$$\begin{aligned} \texttt{q} \quad = \quad & \texttt{TOP k} \\ & \texttt{SELECT } \bar{\texttt{B}}, \texttt{agg}_1(\texttt{A}_1), \texttt{agg}_2(\texttt{A}_2), \ldots \\ & \texttt{FROM } \bar{\texttt{R}} \texttt{ WHERE C GROUP-BY } \bar{\texttt{B}} \end{aligned} \tag{1}$$

The aggregate operators can be `sum`, `count` (which is `sum(1)`), `min` and `max`; we do not support `avg`. We do not discuss here a HAVING clause: we study that elsewhere [16].

Semantics We define now the meaning of the query q on a probabilistic database $(\{W_1, \ldots, W_n\}, \mathbf{P})$. Intuitively, the answer to query is a table like this:

$\texttt{B}_1$	$\texttt{B}_2$	...	$\texttt{agg}_1(\texttt{A}_1)$	$\texttt{agg}_2(\texttt{A}_2)$	...	p
b_{11}	b_{12}	...	e_{11}	e_{12}	...	p_1
b_{21}	b_{22}	...	e_{21}	e_{22}	...	p_2
...	...	...	...	...	...	...

Each answer is of the form $(\bar{b}, \bar{e}, p)$ and consists of (1) a tuple $\bar{b}$ of the group-by attributes; the meaning is that $\bar{b}$ occurs as answer to q in at least one possible world, (2) a tuple $\bar{e}$ of the aggregate attributes; the meaning is that these are the expected values of those aggregates over all possible worlds that return $\bar{b}$, (3) the probability p that $\bar{b}$ is an answer. For a simple illustration, the query in Fig. 3 is an instance of (1): it has no aggregates (hence the `GROUP-BY` becomes `DISTINCT`), and its answers are shown in Fig. 4. We define query semantics formally next.

Under standard SQL semantics, the answers to `q` on a possible world W_j are tuples of the form $(\bar{b}, \bar{a})$, where $\bar{b}$ are the values of the $\bar{\texttt{B}}$ attributes and $\bar{a}$ are aggregate values. Given $\bar{b}$, denote $C_{\bar{b}}(W_j)$ the predicate "$q(W_j)$ contains at least one answer of the form $(\bar{b}, \bar{a})$"; denote $F_{\bar{b}}(W_j)$ the function that returns the unique value $\bar{a}$ for which $(\bar{b}, \bar{a})$ is in $q(W_j)$ when $C_{\bar{b}}(W_j)$ is true, and which is undefined when $C_{\bar{b}}(W_j)$ is false. Recall the standard definitions of the probability of a predicate, and the conditional expected value of a function:

$$\begin{aligned} \mathbf{P}(C_{\bar{b}}) &= \sum_{j \mid C_{\bar{b}}(W_j) = \text{true}} \mathbf{P}(W_j) \\ \mathbf{E}(F_{\bar{b}} \mid C_{\bar{b}}) &= \sum_{j \mid C_{\bar{b}}(W_j) = \text{true}} F_{\bar{b}}(W_j)\mathbf{P}(W_j)/\mathbf{P}(C_{\bar{b}}) \end{aligned}$$

Definition 2.4. *For a probabilistic db $(\mathcal{W}, \mathbf{P})$ define:*

$$q(\mathcal{W}, \mathbf{P}) \quad = \quad \{(\bar{b}, \bar{e}, p) \mid \exists W_j . C_{\bar{b}}(W_j), \bar{e} = \mathbf{E}(F_{\bar{b}} \mid C_{\bar{b}}), p = \mathbf{P}(C_{\bar{b}})\}$$

For a representation J^p we define $q(J^p) = q(Mod(J^p))$.

Finally, in the context of a top-k query, we return only the tuples $(\bar{b}, \bar{e}, p)$ with the k highest probabilities p

2.3 From Queries to DNF Formulas

Our approach to query evaluation $q(J^p)$ is to compute probabilities of certain DNF formulas constructed by running a modified SQL query, called *expanded* query `qe`, over the representation J^p; `qe` is derived from `q` by replacing the `SELECT` clause with `*` and removing the `GROUP BY` clause.

```
qe = SELECT * FROM R̄ WHERE C
```

where $\bar{R} = R_1, \ldots, R_m$ and C are the same as in Eq.(1). We evaluate `qe` in the database engine, over the instance J^p, and obtain a set of answers ET. Each tuple $t \in ET$ has the form $t = (t_1, \ldots, t_m)$, where $t_1 \in R_1^p, \ldots, t_m \in R_m^p$. Define the following boolean expression associated to t:

$$t.E \quad = \quad t_1 \wedge t_2 \wedge \ldots \wedge t_m \tag{2}$$

One may compute $\mathbf{P}(t.E)$ by first eliminating duplicate tuples, then multiplying their probabilities (unless two tuples are exclusive, in which case $\mathbf{P}(t.E) = 0$).

Next, partition ET by the `GROUP-BY` attributes $\bar{\mathtt{B}}$: $ET = G_1 \cup G_2 \cup \ldots \cup G_n$. For each group $G \in \{G_1, \ldots, G_n\}$ define following DNF boolean expression:

$$G.E = \bigvee_{t \in G} t.E \tag{3}$$

Valiant [17] has shown that computing the probability $\mathbf{P}(G.E)$ of a DNF formula like (3) is #P-complete in general. For a group G, denote $G.\bar{\mathtt{B}}$ the tuple $\bar{b} = t.\bar{\mathtt{B}}$ for some $t \in G$ (it is independent on the choice of $t \in G$). We can prove:

Theorem 2.5. *$q(J^p)$ consists of all tuples $(\bar{b}, \bar{e}, p)$ s.t.:*

$$\begin{aligned}
\bar{b} &= G.\bar{\mathtt{B}} \text{ for some } G \in \{G_1, \ldots, G_n\} \\
p &= \mathbf{P}(G.E) \\
e_i &= \sum_{t \in G} \mathbf{P}(t.E) * t.A_i / p \text{ if } AGG_i = sum(A_i) \\
e_i &= \sum_{t \in G} (1 - \mathbf{P}(\bigvee_{t' \in G \mid t'.A_i \geq t.A_i} t'.E)) * t.A_i / p \text{ if } AGG_i = max(A_i)
\end{aligned}$$

Definition 2.4 gives the query's semantics in terms of possible worlds; this theorem shows how to rephrase it in terms of DNF expressions over tuples.

Example 2.6 Consider the query `q` in Fig. 3. The extended query is obtained by removing the group-by clause (removing `DISTINCT`) and replacing the `SELECT` clause with `*`:

```
SELECT * FROM (...same 7 tables...) WHERE ...
```

Thus, each answer returned by `qe` contains the seven tuple variables defined in the `FROM` clause: $(\mathtt{a}, \mathtt{b}, \mathtt{ax}^p, \mathtt{by}^p, \mathtt{x}, \mathtt{y}, \mathtt{d})$. Of these only $\mathtt{ax}^p, \mathtt{by}^p$ are probabilistic tuples, and we added the superscript p to indicate that they range over $\mathtt{TitleMatch}^p$. Thus, each row t returned by `qe` defines a boolean formula $t.E = \mathtt{ax}^p \wedge \mathtt{by}^p$, and its probability $\mathbf{P}(t.E)$ is given by:

Algorithm 2.3.1 Luby-Karp algorithm for computing the probability of a DNF formula $G.E = \bigvee_i t_i.E$ (Eq.(3)), where each $t_i.E$ is a disjunct (Eq.(2)).

```
fix an order on the disjuncts: $t_1, t_2, \ldots, t_m$
$C := 0$
repeat
  Choose randomly one disjunct $t_i \in G$
  Choose randomly a truth assignment s.t. $t_i.E = \texttt{true}$
  if forall $j < i$ $t_j.E = \texttt{false}$ then $C := C + 1$
until $N$ times
return $\tilde{p} = C/N$
```

$\mathbf{P}(t.E) = \texttt{ax}^p\texttt{.p}$ if $\texttt{ax}^p\texttt{.asin}=\texttt{by}^p\texttt{.asin} \wedge \texttt{ax}^p\texttt{.mid}=\texttt{by}^p\texttt{.mid}$
$\mathbf{P}(t.E) = 0$ if $\texttt{ax}^p\texttt{.asin}=\texttt{by}^p\texttt{.asin} \wedge \texttt{ax}^p\texttt{.mid}\neq\texttt{by}^p\texttt{.mid}$
$\mathbf{P}(t.E) = \texttt{ax}^p\texttt{.p*by}^p\texttt{.p}$ if $\texttt{ax}^p\texttt{.asin}\neq\texttt{by}^p\texttt{.asin} \wedge \texttt{ax}^p\texttt{.mid}\neq\texttt{by}^p\texttt{.mid}$ Next, we group the rows by their directors, and for each group $G = \{(\texttt{ax}_1^p, \texttt{by}_1^p), \ldots, (\texttt{ax}_m^p, \texttt{by}_m^p)\}$ construct the DNF formula: $G.E = \texttt{ax}_1^p \wedge \texttt{by}_1^p \vee \ldots \vee \texttt{ax}_m^p \wedge \texttt{bx}_m^p$. The director's probability give by $\mathbf{P}(G.E)$: this is a 2DNF, but computing its probability is still #P-complete.

In summary, we have rephrased the query evaluation problem to the problem of evaluating, for each query answer, the probability of one or more DNF formulas: $p = \mathbf{P}(G.E)$, and any DNF formula occurring in the expression for max (and similarly for min). From now on we will focus on computing $\mathbf{P}(G.E)$, where $G.E$ is given by Eq.(3).

Monte Carlo Simulation A Monte Carlo algorithm repeatedly chooses at random a possible world, and computes the truth value of the boolean expression $G.E$ (Eq.(3)); the probability $p = \mathbf{P}(G.E)$ is approximated by the frequency $\tilde{p}$ with which $G.E$ was true. Luby and Karp have described the variant shown in Algorithm 2.3.1, which has better guarantees than a naive MC. For our purposes the details of the Luby and Karp algorithm are not important: what is important is that, after running for N steps, the algorithm guarantees with high probability that p is in some interval $[a, b]$, whose width shrinks as N increases. Formally:

Theorem 2.7. *[13] Let $\delta > 0$ and define $\varepsilon = \sqrt{4m \log(2/\delta)/N}$, where $m = |G|$ represents the number of tuples in the group G (i.e. number of disjuncts in the DNF formula) and N is the number of steps executed by the Luby and Karp algorithm. Let $a = \tilde{p} - \varepsilon$ and $b = \tilde{p} + \varepsilon$. Then the value p belongs to $[a, b]$ with probability $\geq 1 - \delta$, i.e.*[2]*:*

$$\mathbf{P}(p \in [a, b]) \quad > \quad 1 - \delta \tag{4}$$

3 Top-k Query Evaluation

We now describe our algorithm. We are given a query q as in Eq.(1) and an instance J^p stored in a SQL database engine, and we have to compute the top k answers in $q(J^p)$. Evaluation has two parts: (1) evaluating the extended SQL query qe in the engine and grouping the answer tuples, (2) running a Monte Carlo simulation on each group in the middleware to compute the probabilities, then returning the top k probabilities. We describe here the basic algorithm, and discuss optimizations in the next section.

3.1 Multisimulation (MS)

We model the problem as follows. We are given a set $\mathcal{G} = \{G_1, \ldots, G_n\}$ of n objects, with unknown probabilities $p_1, \ldots, p_n$, and a number $k \leq n$. Our goal is to find a set of k objects with the highest probabilities, denoted `TopK` $\subseteq \mathcal{G}$: we discuss below how to also sort this set. The way we observe the objects' probabilities is by means of a simulation algorithm that, after running N steps on an object G, returns an approximation interval $[a^N, b^N]$ for its probability p, with $a^N < b^N$ (we assume $a^N = b^N$ can never happen). We make the following four assumptions about the simulation algorithm and about the unknown probabilities:

[2]In the original paper the bound is given as $| p - \tilde{p} | \leq \varepsilon p$. Since $p \leq 1$ this implies our bounds.

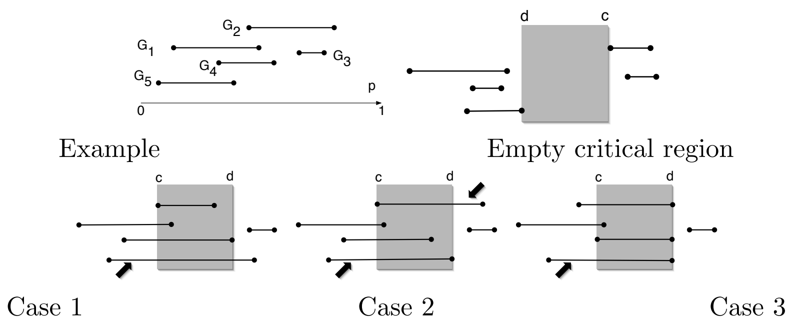


Figure 6. Illustration of MS; $k = 2$.

Convergence : $\lim_{N \to \infty} a^N = \lim_{N \to \infty} b^N$.

Precision : $\forall N. p \in [a^N, b^N]$.

Progress : $\forall N. [a^{N+1}, b^{N+1}] \subseteq [a^N, b^N]$.

Separation : $\forall i \neq j$, $p_i \neq p_j$.

By the separation assumption TopK has a unique solution, i.e. there are no ties, and by the other three assumptions the solution can be found naively by a round robin algorithm.

In our setting each object G is a group of tuples, and its probability is $p = \mathbf{P}(G.E)$ (Eq. (3)). The simulation algorithm is Luby-Karp. Only the first assumption holds strictly (convergence): we revisit below how to address the other three.

Intuition Any algorithm that computes TopK can only do this by running simulations on the objects. It initializes the intervals to $[a_1, b_1] = [a_2, b_2] = \ldots = [a_n, b_n] = [0, 1]$, then repeatedly chooses to simulate some G_i for one step. At each point in the execution, object G_i has been simulated N_i steps, and thus its interval is $[a_i^{N_i}, b_i^{N_i}] = [a_i, b_i]$ (we omit the superscript when it is clear). The total number of steps over all groups is $N = \sum_{i=1}^{n} N_i$. Consider the top left figure in Fig. 6, where for $k = 2$. Here we have already simulated each of the five groups for a while: clearly G_3 is in the top 2 (it may be dominated only by G_2), although we don't know if it is 1st or 2nd. However, it is unclear who the other object in top 2 is: it might be G_1, G_2, or G_4. It is also certain that G_5 is not among the top 2 (it is below G_2, G_3).

Given two intervals $[a_i, b_i]$, $[a_j, b_j]$, if $b_i \leq a_j$ then we say that the first is *below*, and the second is *above*. We also say that the two intervals are *separated*: in this case we know $p_i < p_j$ (even if $b_i = a_j$, due to the "separation" assumption). We say that the set of n intervals is k-separated if there exists a set $T \subseteq \mathcal{G}$ of exactly k intervals s.t. any interval in T is above any interval not in T. Any algorithm searching for the TopK must simulate the intervals until it finds a k-separation (otherwise we can prove that TopK is not uniquely determined); in that case it outputs TopK $= T$. The cost of the algorithm is the number of steps N at termination.

Our golden standard will be the following nondeterministic algorithm, OPT, which is obviously optimal. OPT "knows" exactly how many steps to simulate G_i, namely N_i^{opt} steps, such that the following holds (a) the intervals $[a_1^{N_1^{opt}}, b_1^{N_1^{opt}}]$, $\ldots$, $[a_n^{N_n^{opt}}, b_n^{N_n^{opt}}]$ are k-separated, and (b) the sum $N^{opt} = \sum_i N_i^{opt}$ is minimal. When there are multiple optimal solutions, OPT chooses one arbitrarily. Clearly such an oracle algorithm cannot be implemented in practice. Our goal is to derive a *deterministic* algorithm that comes close to OPT.

Example 3.1 To see the difficulties, consider two objects G_1, G_2 and $k = 1$ with probabilities $p_1 < p_2$, and the current intervals (say, after simulating both G_1 and G_2 for one step) are $[a_1, b_1]$, $[a_2, b_2]$ s.t. $a_1 = p_1 < a_2 < b_1 < p_2 = b_2$. The correct top-1 answer is G_2, but we don't know this until we have separated them: all we know is $p_1 \in [a_1, b_1]$, $p_2 \in [a_2, b_2]$ and it is still possible that $p_2 < p_1$. Suppose we decide to simulate repeatedly only G_2. This clearly cannot be optimal. For example, G_2 may require a huge number of simulation steps before a_2 increases above b_1, while G_1 may take only one simulation step to decrease b_1 below a_2: thus, by betting only on G_2 we may perform arbitrarily worse than OPT, which would know to choose G_1 to simulate. Symmetrically, if we bet only on G_1, then there are cases when we perform much worse than OPT. Round robin seems a more reasonable strategy, i.e. we simulate alternatively G_1 and G_2. Here, the cost is twice that of OPT, in the following case: for N steps a_2 and b_1 move very little, s.t. their relative order remains unchanged, $a_1 < a_2 < b_1 < b_2$. Then, at the $N+1$'th step, b_1 decreases dramatically, changing the order to $a_1 < b_1 < a_2 < b_2$. Round robin finishes in $2N+1$ steps. The N steps used to simulate G_2 were wasted, since the changes in a_2 were tiny and made no difference. Here OPT chooses to simulate only G_1, and its cost is $N+1$, which is almost half of round robin. In fact, no deterministic algorithm can be better than twice the cost of OPT. However, round robin is not always a good algorithm: sometime it can perform much worse than OPT. Consider n objects $G_1, \ldots, G_n$ and $k = 1$. Round robin may perform n times worse than OPT, since there are cases in which (as before) choosing the right object on which to bet exclusively is optimal, while round robin wastes simulation steps on all the n objects, hence its cost is $n \cdot N^{opt}$.

Notations and definitions Given n non-negative numbers $x_1, x_2, \ldots, x_n$, not necessarily distinct, let us define $top_k(x_1, \ldots, x_n)$ to be the k's largest value. Formally, given some permutation s.t. $x_{i_1} \geq x_{i_2} \geq \ldots \geq x_{i_n}$, top_k is defined to be x_{i_k}. We set $top_{n+1} = 0$.

Definition 3.2. *The* critical region*,* top objects*, and* bottom objects *are:*

$$\begin{aligned} (c, d) &= (top_k(a_1, \ldots, a_n), top_{k+1}(b_1, \ldots, b_n)) && (5) \\ T &= \{G_i \mid d \leq a_i\} \\ B &= \{G_i \mid b_i \leq c\} \end{aligned}$$

One can check that $B \cap \texttt{TopK} = \emptyset$ and $T \subseteq \texttt{TopK}$: e.g. $b_i \leq c$ implies (by definition of c) that there are k intervals $[a_j, b_j]$ above $[a_i, b_i]$, which proves the first claim. Fig. 6 illustrates four critical regions.

The important property of the critical region is that the intervals have a k-separation iff the critical region is empty, i.e. $c \geq d$ (proof is omitted for lack of space), in which case we can return $\texttt{TopK} = T$. This is illustrated in the upper right of Fig. 6, where the top 2 objects are clearly those to the right of the critical region. We therefore assume $c < d$ from now on. Call an object G_i a *crosser* if $[a_i, b_i]$ contains the critical region, i.e. $a_i \leq c$, $d \leq b_i$. There are always at least two crossers. Indeed, there are $k+1$ intervals $[a_i, b_i]$ s.t. $d \leq b_i$. and at most $k-1$ of them may satisfy $c < a_i$; hence, the others (at least two) satisfy $a_i \leq c$, and are crossers. Given a crosser $[a_i, b_i]$ we call it an *upper crosser* if $d < b_i$, a *lower crosser* if $a_i < c$, and a *double crosser* if it is both.

The Algorithm is shown in Algorithm 3.1.1. At each step it picks one or two intervals to simulate, according to three cases (see Fig 6). First, it tries a double crosser $[a_i, b_i]$; if there is none then it tries to find an upper crosser, lower crosser pair; if none exists then it means that either all crossers have the same left endpoint $a_i = c$ or all have the same right endpoint $d = b_i$. In either case there exists a maximal crosser, i.e. one that contains all other crossers: pick one and simulate it (there may be several, since intervals may be equal). After each iteration re-compute the critical region; when it becomes empty, stop and return the set T of intervals above the critical region. Based on our previous discussion the algorithm is clearly correct: i.e. it returns `TopK` when it terminates. From the **convergence** assumption it follows that the algorithm terminates.

Algorithm 3.1.1 The Mulisimulation Algorithm

```
**MS_TopK**($\mathcal{G}, k$) : /* $\mathcal{G} = \{G_1, \ldots, G_n\}$ */
Let $[a_1, b_1] = \ldots = [a_n, b_n] = [0, 1]$, $(c, d) = (0, 1)$
while $c \leq d$ do
  Case 1: exists a double crosser: simulate it one step
  Case 2: exists an upper crosser and a lower crosser: simulate both one step
  Case 3: otherwise: pick a maximal crosser, simulate it one step
  Update $(c, d)$ using Eq.(5)
end while
return TopK $= T = \{G_i \mid d \leq a_i\}$
```

Analysis We now prove that the algorithm is optimal within a factor of two of OPT, and, moreover, that no deterministic algorithm can be better.

At any point during the algorithm's execution we say that an interval $[a_i, b_i]$ *has slack* if $N_i < N_i^{opt}$. If it has slack then the algorithm can safely simulate it without doing worse than OPT. We have:

Lemma 3.3. *Let $[a_i, b_i]$ be a crosser. Then, in all cases below, $[a_i, b_i]$ has slack: (1) If it is an upper crosser and is not in the top k. (2) If it is a lower crosser and is in the top k. (3) If it is a double crosser. (4) If it contains all crossers (i.e. it is a maximal crosser).*

Proof. To see (1), note that OPT must find k intervals above i; but since $[a_i^{N_i}, b_i^{N_i}]$ is an upper crosser, there are at most $k-1$ $b_j^{N_j}$'s s.t. $b_j^{N_j} > b_i^{N_j}$; hence, OPT can find at most $k-1$ intervals (namely the same ones, at most) that are above $b_i^{N_i}$, i.e. $a_j^{N_j^{opt}} > b_i^{N_i}$, because $a_j^{N_j^{opt}} < b_j^{N_j}$ (due to the progress assumption). It follows that OPT must simulate i at least one more step than N_i to bring $b_i^{N_i^{opt}}$ below $b_i^{N_i}$ in order to separate it from the top k. (2) and (3) are similar. To prove (4), we assume that the interval i is in `TopK`: the other case is symmetric. Consider the $k+1$ intervals that have $b_j \geq d$: at least one, say $[a_j, b_j]$, must be not in `TopK`, and OPT must separate them by proving that $[a_j, b_j]$ is below $[a_i, b_i]$. But $a_i \leq a_j$ because either $[a_j, b_j]$ is included in $[a_i, b_i]$, or $[a_j, b_j]$ is not a crosser (hence $a_i \leq c \leq a_j$). Hence, to separate them, OPT must either reduce $[a_j, b_j]$ to a point or further simulate $[a_i, b_i]$. But since we assume that an MC algorithm cannot return a point interval (i.e. $a^N < b^N$ forall N), OPT must simulate $[a_i, b_i]$. □

Theorem 3.4. *(1) The cost of algorithm* **MS_TopK** *is $< 2N^{opt}$. (2) For any deterministic algorithm computing the top k and for any $c < 2$ there exists an instance on which its cost is $\geq cN^{opt}$.*

Proof. (Sketch) The main idea for (2) is in Example 3.1 and we omit it for lack of space. To prove (1), notice that at each step the algorithm simulates one or two intervals: it suffices to prove that at least one of them has slack[3]. There are three cases. First, a double crosser is simulated: clearly it has slack. Second, an upper and a lower crosser are simulated: in order for both not to have slack we must have one is in the top k and the other is not in the top k; but in that case OPT must simulate at least one of them, since they are not separated yet, hence one of them does have slack after all. Third, there are only upper or only lower crossers and we simulate the largest one: we have seen that this also has slack. □

Corollary 3.5. *Let* **A** *be any deterministic algorithm for finding `TopK`. Then (a) on any instance the cost of* **MS_TopK** *is at most twice the cost of* **A**, *and (b) for any $c < 1$ there exists an instance where the cost of* **A** *is greater than c times the cost of* **MS_TopK**.

3.2 Discussion

Variations and extensions In query answering we need to compute the top k answers *and* to sort them. The following variation of MS, which we call **MS_RankK**, does this. First, compute the top k, $T_k = \textbf{MS\_TopK}(\mathcal{G}, k)$. Next, compute the following sets, in this sequence:

$$\begin{aligned} T_{k-1} &= \textbf{MS\_TopK}_{ni}(T_k, k-1) \\ T_{k-2} &= \textbf{MS\_TopK}_{ni}(T_{k-1}, k-2) \\ &\cdots \\ T_1 &= \textbf{MS\_TopK}_{ni}(T_2, 1) \end{aligned}$$

At each step we have a set T_j of the top j answers, and we compute the top $j-1$: this also identifies the j'th ranked object. Thus, all top k objects are identified, in reverse order. Here $\textbf{MS\_TopK}_{ni}$ denotes the algorithm **MS_TopK** without the first line: that is, it does not initialize the intervals $[a_i, b_i]$ but continues from where the previous multisimulation left off. This algorithm is also optimal: we have proved a theorem similar to 3.4, which we omit for lack of space.

The second variation is an *any-time* algorithm, which computes and returns the top answers in order, without knowing k. The user can stop any time. The algorithm starts by identifying the top element $T_1 = \textbf{MS\_TopK}(\mathcal{G}, 1)$, then it finds the remaining groups in decreasing order: $T_{j+1} = \textbf{MS\_TopK}(B_j, 1)$, where $B_j = \mathcal{G} - (T_1 \cup \ldots \cup T_j)$.

[3]This shows that the cost is $\leq 2N^{opt}$; to prove $< 2N^{opt}$ one notices that at least one iteration simulates a single interval, with slack.

Note that for $k > 1$ this algorithm is *not* optimal in finding the top k elements; its advantage is in its any-time nature. Also, it prevents the semi-join optimization discussed below, which requires knowledge of k.

Revisiting the assumptions Precision holds for any MC algorithm, but only in a probabilistic sense. For example after running Luby-Karp's algorithm for N steps, $\mathbf{P}(p \in [a^N, b^N]) > 1 - \delta_1$. The choice of the *confidence* δ_1 affects the convergence rate: $b^N - a^N = 2\sqrt{4m\log(2/\delta_1)/N}$, where m is the size of the group. In our context the user chooses a global parameter δ and requires that *all* n groups be precise with confidence δ. Assuming equal confidences, the system sets δ_1 for each group to δ/n, since it implies $(1-\delta_1)^n \geq 1-\delta$. Sill, since it appears under log, we can choose *very small* values for δ without affecting significantly the running time (N), hence precision holds for all practical purposes. The separation assumption is more problematic, since in practice probabilities are often equal or very close to each other. Here we simply rely on a second parameter $\varepsilon > 0$: when the critical region becomes less than ε, we stop and rank the uncertain groups based on the midpoints of their intervals. Progress as stated, does not hold for our monte carlo simulation technique. Lastly, we use progress to make the statement that OPT's intervals must be contained in the intervals we see. We observe that for any interval for which precision holds, a weaker variant of progress suffices $\forall N, \exists$ infinitely many $N' > N[a^{N'}, b^{N'}] \subseteq [a^N, b^N]$. Since the limit exists (by convergence assumption), this implies our statement. By choosing δ appropriately, we can ensure only a constant number of errors with high probability. Importantly, these weaker assumptions are satisfied by alg. 2.3.1. The choice of ε also affects running time and precision/recall. We discuss the system's sensitivity on δ and ε in Sec. 6.

Finally, note that our restriction that the intervals never collapse (i.e. $a^N < b^N$ forall N) is important. This is always true in practice (for any MC algorithm). As a pure theoretical observation we note here that without this assumption the proof of Lemma 3.3 (4) fails and, in fact, no deterministic algorithm can be within a constant factor of OPT. Consider searching for the top $k = 1$ of n objects; all n intervals start from the initial configuration $[0, 1]$. OPT picks the winner object, whose interval, after one simulation step, collapses to [1,1]: OPT finishes in 1 step, while any deterministic algorithm must touch all n intervals at least one.

Further Improvements One may wonder if our adversarial model in which intervals may shrink at arbitrary, unpredictable rates is too strong. In theory it may be possible to design an algorithm that finds `TopK` by exploiting the specific rates at which the intervals shrink (see the bounds in Th. 2.7). However, note that this will result in at most a factor of 2 improvement over the MS algorithm, due to Corollary 3.5.

4 Optimizations

We present two optimizations: the first reduces the number of groups to be simulated using a simple pruning technique, the second reduces the sizes of the groups by pushing more of the processing from the middleware to the engine. Both techniques are provably correct in that they are guaranteed to preserve the query's semantics.

Pruning The following are two simple upper and lower bounds for the probability of a group G:

$$\max_{i=1}^{m} \mathbf{P}(t_i.E) \leq \quad \mathbf{P}(\bigvee_{i=1}^{m} t_i.E) \quad \leq \sum_{i=1}^{m} \mathbf{P}(t_i.E)$$

They can be computed easily and allow us to compute the *critical region* using Eq.(5) and to prune some groups before even starting MS. As an improvements, when there are no pairs of disjoint tuples in the group (which is a condition that can be checked statically) then the upper bound can be tighten to $1 - \prod_i (1 - \mathbf{P}(t_i.E))$.

Safe Subqueries Sometimes the probabilities can be pushed to the engine, by multiplying probabilities (when the tuples are independent) or by adding them (when the tuples are disjoint). This can be achieved by running a SQL query, over a subset $\overline{\mathtt{R}}' \subseteq \overline{\mathtt{R}}$ of the tables in the original query `q`, like the following (here $\overline{\mathtt{R}}' =$ `R1, R2, R2`):

```
sq = SELECT B̄', AGG(R1^p.p*R2^p.p*R3^p.p) as p
     FROM R1^p, R2^p, R3^p WHERE C GROUP-BY B̄'
```

where `AGG` is either `sum` or `prod_1_1`:

$$\begin{aligned} \texttt{sum}(p_1, \ldots, p_m) &= \sum_i p_i \\ \texttt{prod\_1\_1}(p_1, \ldots, p_m) &= 1 - \prod_i (1 - p_i) \end{aligned}$$

The optimization works like this. Given the query `q` (Eq.(1)) choose a subset of its tables $\overline{\mathtt{R}}' \subseteq \overline{\mathtt{R}}$, and some set of attributes $\overline{\mathtt{B}}'$ (which must include all attributes on which the relations $\overline{\mathtt{R}}'$ join with the other relations). Then construct a subquery like `sq` above, and use it as a subexpression in `q` as it were a normal table, with probability given by `p`, and its possible-worlds key given by a certain subset $\overline{\mathtt{S}}$ of $\overline{\mathtt{B}}'$.

Three conditions must be met for this rewriting to be correct. (1) The tuple probability `p` computed by `AGG` must be correct, (2) in the output, tuples having the same value of $\overline{\mathtt{S}}$ must be disjoint tuples and tuples having different values of $\overline{\mathtt{S}}$ must be independent tuples, and (3) each such probability must be independent of all the other tuple in the original query that it joins with. Recall that `Key(R)` denotes the set of key attributes for the possible worlds for `R`.

To check (1) we have the following:

Proposition 4.1. *Consider the query* `sq` *above. Let* `Attr(R)` *denote the attributes of relation* `R` *(does not include the* `p` *attribute, which technically belongs only to* $\mathtt{R}^p$*) and* `Attr(sq)` *denote the union of* `Attr(R)` *for all relations* `R` *in* `sq`*.*

1. *If* `AGG` *is* `sum` *then* `p` *is computed correctly iff* $\exists R \in \overline{R}'$ *s.t.* `Key(R)` $\subseteq \overline{B}'$ *and* `Attr(sq)` $- \overline{B}' \subseteq$ `Attr(R)`*.*
2. *If* `AGG` *is* `prod_1_1` *then* `p` *is computed correctly iff* $\forall R \in \overline{R}'$, `Attr(sq)` $- \overline{B}' \subseteq$ `Key(R)`*.*

To check (2) we have the following:

Proposition 4.2. *Consider the query* `sq` *above.*

1. *Two output tuples having the same values of* $\overline{S}$ *are disjoint events iff* $\exists R \in \overline{R}'$ *s.t.* `Key(R)` $\subseteq \overline{S}$ *and* $\overline{B}' - \overline{S} \subseteq$ `Attr(R)`*.*
2. *Two output tuples having different values of* $\overline{S}$ *are independent events iff* $\forall R \in \overline{R}'$, $\overline{B}' - \overline{S} \subseteq$ `Key(R)`*.*

Finally, to check (3) we need to check that the relations used by `sq` do not occur again the rest of the query `q`.

Example 4.3 Consider three probabilistic tables:

`AmazonHighReviews`^{p}`(asin, reviewer, p)`
`TitleMatch`^{p}`(asin, imdbid, p)`
`IMDBHighRatedFilms`^{p}`(imdbid, p)`

with possible worlds keys

`Key(AmazonHighReviews)` = {`asin, reviewer`}
`Key(TitleMatch)` = {`asin`}
`Key(IMDBHighRatedFilms)` = {`imdbid`}

Note that `AmazonHighReviews` and `IMDBHighRatedFilms` contain only independent tuples. Consider the query `q`:

```
q = TOP 5 SELECT DISTINCT A.reviewer
    FROM AmazonHighReviews A,
         TitleMatch T, IMDBHighRatedFilms I
    WHERE A.asin = T.asin and T.imdbid = I.imdbid
```

The query can be optimized by observing that the following subquery is a safe subquery:

```
sq = SELECT T.asin, sum(T.p * I.p) as p
     FROM TitleMatch^p T, IMDBHighRatedFilms^p I
     WHERE T.imdbid = I.imdbid
     GROUP BY T.asin
```

The output of this subquery is a table $\texttt{Tmp}^p\texttt{(asin, p)}$ that can be treated as a base probabilistic table with possible world key `asin` and probability attribute `p`. To see why, let us verify that this subquery satisfies the three conditions for safe subquery:

- For condition (1), we use Prop. 4.1(1). Here $\overline{\texttt{B}}' = \{\texttt{asin}\}$ and $\texttt{Attr(sq)} = \{\texttt{asin, imdbid}\}$. We see that $\texttt{Key(TitleMatch)} \subseteq \overline{\texttt{B}}'$ and $\texttt{Attr(sq)} - \overline{\texttt{B}}' \subseteq$ `Attr(TitleMatch)`, so the condition is met.
- For condition (2), we use Prop. 4.2. Here, $\overline{\texttt{S}} = \{\texttt{asin}\}$ since we are claiming that `asin` is the key for `Tmp`. Prop. 4.2(2) holds trivially because $\overline{\texttt{B}}' - \overline{\texttt{S}} = \emptyset$. Prop. 4.2(1) holds because $\texttt{Key(TitleMatch)} \subseteq \overline{\texttt{S}}$.
- Condition (3) holds because all event tables outside `Tmp` are distinct from those inside.

Having verified that the subquery is indeed safe, we rewrite the query `q` by making `sq` a subquery:

```
q_safe-plan = TOP 5 SELECT DISTINCT A.reviewer
              FROM AmazonHighReviews A, sq Tmp
              WHERE A.asin = Tmp.asin
```

Thus, the table $\texttt{Tmp(}\underline{\texttt{asin}}\texttt{,p)}$ is computed inside the engine, and treated like a base query by MS. The rest of MS remains unchanged. The new query has the same number of groups as the original query, but each group is much smaller since some of the probabilistic computation has been pushed in the engine.

5 A Complete Probabilistic Model

A complete representation formalism for probabilistic database is one that can represent any probabilistic data, according to Def. 2.1. The formalism we described in Sec. 2 is based on Barbara et al. [2], and is incomplete, because it represents only independent or exclusive tuples. Other formalisms described in the literature are either incomplete, or too complex to be practical. For example the model used by Dalvi and Suciu [4] is a strict subset of Barbara's (allows only for independent tuples) and thus is also incomplete. Fuhr and Roellke [8] describe a complete model, based on c-tables [12], but it is complex and impractical. Widom at al. [5] advance the hypothesis that every representation formalism is either incomplete or unintuitive (hence impractical), and proposes a two layered approach with a simple and intuitive, but incomplete model at the top and a complete, but complex model at the bottom. Here, we show that if we extend the simple representation formalism described in Sec. 2 with SQL views, then one obtains a complete representation system. Since SQL views are intuitive to database users, we believe that this system is quite intuitive and practical. However, a prerequisite to such a representation system is a query processor that can evaluate complex SQL queries over probabilistic databases, because every user query needs to be expanded with the view definitions and results in a complex expression. Thus, the complete representation system that we describe here is a direct application of the efficient query processing techniques described in Sec. 3 and 4.

Definition 5.1. *A probabilistic schema with views, S^{pv} consists of a probabilistic schema $S^p = \{R_1^p, \ldots, R_m^p\}$ associated to some deterministic schema S (see Sec. 2.1) and a set of SQL view definitions of the form $V_i := q_i$, $i = 1, k$, where V_i is a relation name and q_i is a* `SELECT-DISTINCT` *SQL query.*

An instance of S^{pv} is an instance J^p of S^p, and its meaning, denoted $Mod^v(J^p)$, is a probability distribution over possible worlds of schema $S \cup \{V_1, \ldots, V_k\}$, obtained, intuitively, by first taking the probability distribution $Mod(J^p) = (\{W_1, \ldots, W_n\}, \mathbf{P})$ (i.e. ignoring the views), then evaluating the views on each possible world W_j. Formally, we write $V(W_j)$ for the instance consisting of $m + k$ relations: the m relations $R_1, \ldots, R_m$ in W_j plus the k relations $V_1, \ldots, V_k$ obtained by evaluating the views.

Probabilistic Table Name	#Tuples	#exclusive tuples	
		Avg.	Max
MovieToAsin	339095	4	13
AmazonReviews	292680	1	1
ActorMatch	6758782	21	2541
DirectorMatch	18832	2	36
UsenetMatch	134803	5	203
UsenetReview	3159	1	3159
ActivityData	2614480	3	10
HMM	100	10	10

Figure 7. Three Case Studies of Imprecisions

Query name	# of groups (n)	Avg group size m		Max group size		# of prob. tables (m)
		no SP	SP	no SP	SP	
SS	33	20.4	8.4	63	26	2
SL	16	117.7	77.5	685	377	4
LS	3259	3.03	2.2	30	8	2
LL	1415	234.8	71.0	9088	226	4

Figure 8. Query Stats w/o and w/ S(afe) P(lan)

Definition 5.2. *$Mod^v(J^p)$ is $(\{V(W_1), \ldots, V(W_n), \mathbf{P}^v\}$ where $\mathbf{P}^v(V(W_j)) = \mathbf{P}(W_j)$, $j = 1, n$.*

Users can formulate queries q over the schema S^{pv}. Since an instance J^p of S^{pv} has possible worlds semantics, the query semantics is simply $\mathtt{q}(J^p) = \mathtt{q}(Mod^v(J^p))$. Moreover, $\mathtt{q}(J^p)$ can be computed by first expanding all the views in the query, then evaluating the rewritten query in the normal way (without views).

We show completeness of this formalism. For two schemas $S \subseteq S_0$ (i.e. S_0 consists of all relation names in S, possibly more) and an instance W of S_0, denote $\Pi_S(W)$ the instance obtained from W by dropping all tables in $S_0 - S$. Given a probability distribution $(\mathcal{W}, \mathbf{P})$ over schema S_0, denote $\Pi_S(\mathcal{W}, \mathbf{P})$ the probability distribution $(\{\Pi_S(W) \mid W \in \mathcal{W}\}, \mathbf{P}')$, $\mathbf{P}'(\Pi_S(W)) = \sum_{\Pi_S(W')=\Pi_S(W)} \mathbf{P}(W')$.

Theorem 5.3 (Completeness)**.** *Let S be any deterministic schema with m relation names. Then there exists a probabilistic schema S^{pv} with $m+1$ relations and m views s.t. for any probability distribution $(\mathcal{W}, \mathbf{P})$ over S there exists an instance J^p of S^{pv} s.t. $(\mathcal{W}, \mathbf{P}) = \Pi_S(Mod^v(J^p))$.*

Proof. We prove for $m = 1$, i.e. $S = \{\mathtt{R}(\bar{A})\}$ (the general case is similar). Define $S^p = \{\mathtt{W}^p\mathtt{(wid,\ p)},\ \mathtt{T(wid,}\ \bar{A}\mathtt{)}\}$, where the key in the corresponding deterministic counterpart `W(wid)` is defined to be empty, $\mathtt{Key(W)} = \emptyset$, while `T` is deterministic and has key $\{\mathtt{wid}\} \cup \mathtt{Key(R)}$. Define the view:

```
R := SELECT DISTINCT t.Ā FROM W x, T t WHERE x.wid = t.wid
```

To prove the claim, let $\mathcal{W} = \{W_1, \ldots, W_n\}$ be a set of possible worlds over S (each W_j is an instance of R) and let $p_1, \ldots, p_n$ be their probabilities. Construct J^p as follows. Create n distinct identifiers $w_1, \ldots, w_n$: the $\mathtt{W}^p$ table in J^p is $\{(w_1, p_1), \ldots, (w_n, p_n)\}$ (these represent the n worlds and are exclusive tuples since $\mathtt{Key(W)} = \emptyset$). The T table in J^p is $\bigcup_j \{w_j\} \times W_j$. It follows immediately that the possible worlds in $Mod^v(J^p)$ are $\{(\{w_1\}, \mathtt{T}, W_1), \ldots, (\{w_n\}, \mathtt{T}, W_n)\}$ (each world consisting of a W table, a T table, and an R table), and their probabilities are $p_1, \ldots, p_n$, which proves the theorem. □

6 Experiments

In this section we evaluate our approach experimentally. We address five questions: (1) what is the scale of probabilistic databases when modeling imprecisions; (2) how does our new query evaluation method compare to the current state of the art; (3) how effective is the multisimulation (MS) over a naive approach (4) how effective are the optimizations; and (5) how sensitive is the system's performance on the choice of δ and ε.

Setup Our experiments were run on a dual processor Intel Xenon 3GHz Machine with 8G RAM and 2 400GB disks. The operating system used was Linux with kernel version 2.6.12 high-mem build. The database was DB2 UDB Trial Edition, v. 8.2. Due to licensing restrictions DB2 was only one able to use one of the cores. Indexes and configuration parameters such as buffer pools were tuned by hand.

Methodology For each running time we perform the experiment 5 times, dropping the highest and the lowest and average the remaining three runs. The naive simulation method was capped at 20 minutes. In between each experiment, we force the database to terminate all connections. The same experiments was not run repeatedly to minimize caching effects but the cache was allowed to be warm. In the precision/recall experiments, the precision and recall are defined as the fraction of the top k answers returned by method being evaluated that overlap with the "correct" set of top k answers. In order to compute the latter we had to compute the exact tuple probabilities, which is intractable. For that we used the approximate values returned by the simulation algorithm with very low settings for ε and δ: $\varepsilon = 0.001$ and $\delta = 0.01$.

6.1 Case Studies

In an empirical study we modeled imprecisions in three application domains. The first integrates the IMDB movie database with reviews from Amazon, as described in a simplified form in Sec. 1.1, and the sources of imprecisions are fuzzy object matches (for titles, actors, and directors), and the confidence in the amazon reviews ("how many people found this review useful"). The second application integrates IMDB with reviews collected from a USENET site[4]. These reviews were in free text and we had to use information extraction techniques to retrieve for each review (a) the movie and (b) the rating. The imprecisions here were generated by information extraction tools. In the third application we used human activity recognition data obtained from body-worn sensors [15]. The data was first collected from eight different sensors (accelerometer, audio, IR/visible light, high-frequency light, barometric pressure, humidity, temperature, and compass) in a shoulder mounted multi-sensor board, collected at a rate of 4 per second, then classified into into $N = 10$ classes of human activity $A^1, A^2, \ldots, A^N$, one for each subject and each time unit. The classes where: riding elevator up or down, driving car, riding bicycle, walking stairs up or down, jogging, walking, standing, sitting. The imprecisions here come from the classification procedure, which results in probability distribution on the N activities.

Fig. 7 shows brief summaries of the probabilistic data in each of these applications. Each required between two and four base probabilistic tables, and between one to three SQL views for complex probabilistic correlations. In addition to the probabilistic data IMDB had some large deterministic tables (over 400k movies, 850k actors, and 3M casts, not shown in the figure), which are part of the query processor's input in the experiments below, hence they are important for our evaluation.

6.2 Query Performance

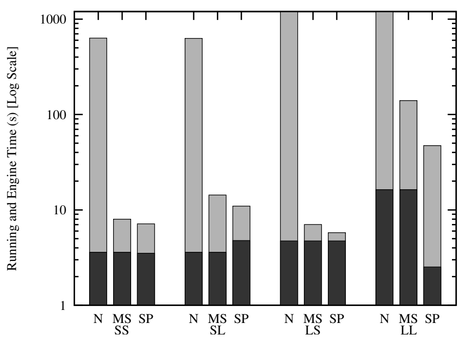
We report below measurements only from the first data set (IMDB-Amazon integration), which was the largest and richest. The processor's performance was mostly affected by two query parameters: the number of groups (denoted n in Sec. 3) and the average size of each group. In additional experiments (not shown) we noticed that the performance was less affected by the number of probabilistic tables in the query (m in Sec. 3), which roughly corresponds to the number of sources of evidence in the imprecise data.

By choosing each parameter to be small (S) or large (L) we obtained four classes of queries denote SS, SL, LS, and LL respectively; we chose one query form each class, and show it in Fig. 8. The queries are:

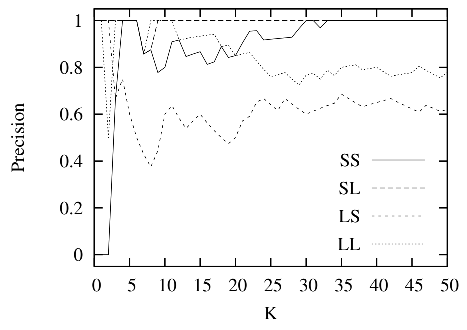
SS In which years did *Anthony Hopkins* appear in a highly rated movie? (Our system returns the top answer 2001, the year he was in Hannibal)

SL Find all actors who were in Pulp Fiction who were in two very bad movies in the five years before Pulp Fiction. (Top 2 Answers: Samuel L Jackson and Christopher Walken)

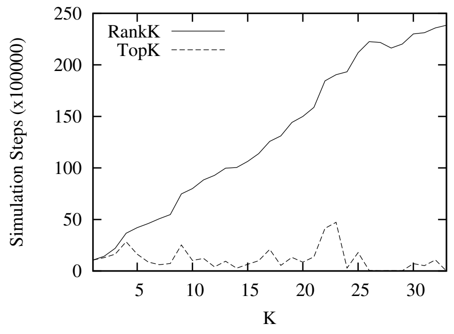
[4]`ftp://ftp.uu.net/usenet/rec.arts.movies.reviews/`



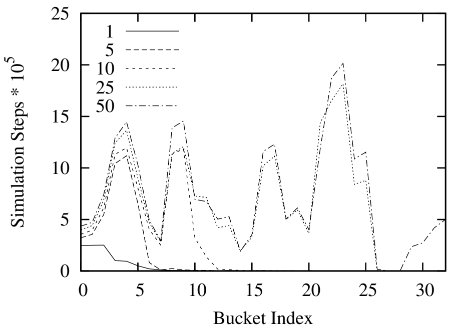
(a) Running times N(aive), MS, and S(afe) P(lan)
$[k = 10, \varepsilon = .01, \delta = .01]$



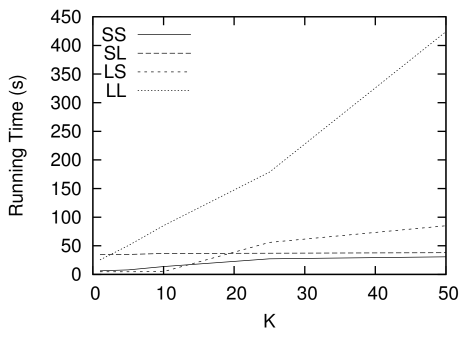
(b) Precision/Recall for naive strategies



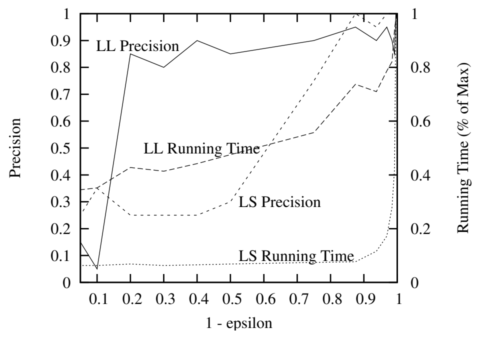
(c) Total number of simulation steps for query SS



(d) Number of simulation steps per bucket for query SS



(e) Effect of K on Running Time



(f) Effect of ϵ on Precision and Running Time

Figure 9. Experimental Evaluation

LS Find all directors who had a low rated movie between 1980 and 1989. (Top 2 Answers: Richard C. Sarafian for Gangster Wars and Tim King for Final Run)

LL Find all directors who had a low rated drama and a high rated comedy less than five years apart. (Top Answer: Woody Allen)

Unless otherwise stated, the confidence and precision parameters were $\varepsilon = .01$, $\delta = .01$, and the multisimulation algorithm run was **MS_RankK** (Sec. 3.2), which finds the top k and sorts them.

Comparison with Other Methods The state of the art in query evaluation on probabilistic databases is to either compute each query answer exactly, using a complete Monte Carlo simulation (we call this method naive (N)), or to approximate the probabilities using some strategies [14] by ignoring their correlations. The first results in much larger running times than multisimulation (MS): see Fig. 9 (a) (note the logarithmic scale): the naive method timed out for the LS and LL queries. The approximation method is much faster than MS, but results in lower precision/recall, due to the fact that it ignores correlations between imprecisions: this is shown in Fig. 9 (b). Note that, unlike a Monte Carlo simulation, where precision and recall can be improved by running longer, there is no room for further improvement in the approximate method. Note that one of the queries (LS) flattened at around 60% precision/recall. The queries that reached 100% did so only when k reached the total number of groups: even then, the answers are much worse then it looks since their order is mostly wrong. This clearly shows that one cannot ignore correlations when modeling imprecisions in data.

Analysis of Multisimulation The main idea of the multisimulation algorithm is that it tries to spend simulation steps on only the top k buckets. We tested experimentally how the total number of simulation steps varies with k, and in which buckets the simulation steps are spent. We show here the results for SS. Fig. 9 (c) shows the total number of simulation steps as a function of k, both for the `TopK` algorithm (which only finds the top k set without sorting it) and for the `RankK` algorithm (which finds *and* sorts the top k set). First, the graph clearly shows that `RankK` benefits from low values of k: the number increases linearly with k. Second, it shows that, for `TopK`, the number of steps is essentially independent on k. This is because most simulation steps are spent at the separation line between the top k and the rest. A deeper views is given by the graph in Fig. 9 (d), which shows for each group (bucket) how many simulation steps were spent, for $k = 1, 5, 10, 25$, and 50. For example, when $k = 1$ most simulation steps are spent in buckets 1 to 5 (the highest in the order of the probability). The graph illustrates two interesting things: that `RankK` correctly concentrates most simulation steps on the top k buckets, and that, once k increases beyond a given bucket's number, the number of simulation steps for that bucket does not further increase. The spikes in both graphs correspond to clusters of probabilities, where MS had to spend more simulation steps to separate them. Fig. 9 (e) shows the effect of k on the measured running time of each query. As expected, the running time scales almost linearly in k. That is, the fewer answer the user requests, the faster they can be retrieved.

Effectiveness of the Optimizations We tested both optimizations: the semijoin pruning and safe query rewriting. The semijoin pruning was always effective for the queries with a large number of buckets (LS, LL), and harmless for the other two. We performed the pruning in the middleware, and the additional cost to the total running time was negligible. The safe-plan rewriting (SP) is more interesting to study, since it is highly non-trivial. Fig. 9 (a) shows significant improvements (factors of 3 to 4) in the running times when the buckets are large (SL, LL), and modest improvements in the other cases. The query time in the engine differed, since now the queries issued are different: in one case (SL) the engine time was larger. Fig. 8 shows how the SP optimization affects the average group size: this explains the better running times.

Sensitivity to Parameters Finally, we tested the system's sensitivity to the parameters δ and ε (see Sec. 3.2). Recall that the theoretical running time is $O(1/\varepsilon^2)$ and $O(\log(1/(n\delta))$. Fig. 9 (f) shows both the precision/recall and the total running time as a function of $1-\varepsilon$, for two queries: LL and LS; $k = 20$, $\delta = 0.01$, and SP is turned off. The running time are normalized to that of our golden standard, $1-\varepsilon = 0.99$. As $1-\varepsilon$ increases, the precision/recall quickly approaches the upper values, while the running time increases too, first slowly, then dramatically. There is a price to pay for very high precision/recall (which is what we did in all the other experiments). However, there is some room to tune $1 - \varepsilon$: around 0.9 both queries have a precision/recall of 90%-100% while the running time is significantly less than the golden standard. The similar graphs for δ differ, and is much more boring: the precisions/recall reaches 1 very fast, while the running time is almost independent on δ. (The graphs look almost like two horizontal lines.) We can choose δ in a wide range without degrading either precision/recall or performance.

7 Conclusion

In this paper we described a method for answering top-k queries on probabilistic databases, with applications to imprecisions in data. We have proven our technique to be near optimal, and have validated it experimentally. Two optimization are describe that further improve performance.

References

[1] Rohit Ananthakrishna, Surajit Chaudhuri, and Venkatesh Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, pages 586–596, 2002.

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

[3] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*, pages 313–324, 2003.

[4] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, Toronto, Canada, 2004.

[5] A. Das Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, 2006.

[6] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, pages 588–599, 2004.

[7] I. P. Fellegi and A. B. Sunter. A theory for record linkage. In *Journal of the American Statistical Society*, volume 64, pages 1183–1210, 1969.

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

[9] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian-Augustin Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, pages 371–380, 2001.

[10] Lifang Gu, Rohan Baxter, Deanne Vickers, and Chris Rainsford. Record linkage: Current practice and future directions. In *CMIS Technical Report No. 03/83*, 2003.

[11] Mauricio Hernandez and Salvatore Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995.

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

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

[14] L. Lakshmanan, N. Leone, R. Ross, and V.S. Subrahmanian. Probview: A flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3), 1997.

[15] J. Lester, T. Choudhury, N. Kern, G. Borriello, and B. Hannaford. A hybrid discriminative/generative approach for modeling human activities. In *IJCAI*, pages 766–772, 2005.

[16] C. Re, N.Dalvi, and D.Suciu. Evaluation of having queries on probabilistic databases, 2006. submitted.

[17] L. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8:410–421, 1979.

[18] Stephen E. Fienberg William W. Cohen, Pradeep Ravikumar. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.

[19] William Winkler. The state of record linkage and current research problems. In *Technical Report, Statistical Research Division, U.S. Bureau of the Census*, 1999.

8 Appendix A

The high level goal of this section is to establish that after a negligible number of steps with respect to the optimal, the progress assumption holds. The result contained here, allows us to show that with extremely high probability our algorithm takes $2(OPT + o(OPT))$ steps, thus achieving our stated competitive ratio of 2.

Progress holds when our estimator jumps a smaller distance than the difference in our bounds. We will see that with high probability our estimator jumps $O(\frac{1}{n})$ (corrolary 8.3) even after a relatively large number of steps. If we take n^{α} simulation steps, the difference in our confidence bounds are like $n^{\alpha-3/2}$ (lemma 8.1), thus progress holds so long as the following equation holds

$$\frac{1}{n} < n^{\alpha-3/2} \text{ iff } 1 < n^{\alpha-1/2} \tag{6}$$

This tells us, if $\alpha > \frac{1}{2}$ with high probability progress holds.

Remark 8.1. *As a technical aside, we do not assume anything about the underlying distribution of true probabilities, thus keeping our competitive result free of distributional assumptions. Specifically, in what follows, all random choices are on the results of the experiment with the underlying probabilities chosen adversarially.*

We first introduce some notation and define our random variables, an estimation lemma and prove our main result.

8.1 Notation and Random Variables

We let n be the number of iterations on the current interval, and consider for a fixed δ what is the probability that progress is violated after we take n_0 steps. We denote the (fixed) true probability p.

Interval Size Function Let $f(n, \delta)$ be the bounding function,

$$f(n, \delta) = \frac{2m}{\sqrt{n}} \log \frac{2}{\delta}$$

Random Variables We have two main random variables, $\rho(n)$ and $X(n_0)$. $\rho(n)$ represents the value of our estimator after n draws. $X(n_0)$ represents the value of n_0 random draws after the n. We observe that all though we have described them as sequential, they are actually independent random variables because they are sums of independent trials. We summarize the properties of these random variables below.

r.v. α	$\mathbf{E}[\alpha]$	As Sums of Indicators
$\rho(n)$	p	$\frac{1}{n}\sum_{i=1}^{n_0} y_i$
$X(n_0)$	$n_0 p$	$\sum_{j=1}^{n} x_j$

Notation An obvious extension is $\rho(n+n_0)$, the estimate after $n+n_0$ steps. This random variable can be written in terms of the other two (independent) rvs as,

$$\rho(n+n_0) = \frac{\rho(n)*n + X(n_0)}{n+n_0} \tag{7}$$

And the following decompositions are helpful later

$$\rho(n+n_0) - \rho(n) = \frac{X(n_0) - \rho(n)n_0}{n+n_0} \tag{8}$$

and,

$$\rho(n) - \rho(n+n_0) = \frac{\rho(n)n_0 - X(n_0)}{n+n_0} \tag{9}$$

Progress Conditions Progress is violated when either of two conditions are met

1. $\rho(n) < \rho(n+n_0)$ and
$$\rho(n) + f(n, \delta) < \rho(n+n_0) + f(n+n_0, \delta')$$
2. $\rho(n) > \rho(n+n_0)$ and
$$\rho(n) - f(n, \delta) > \rho(n+n_0) - f(n+n_0, \delta')$$

8.2 How the bounds shrink

Lemma 8.1. *Let $n_0 = n^\alpha$ and β be a constant such that $\beta > 1$. Then the following holds,*

$$\frac{1}{\sqrt{n}} - \frac{1}{\sqrt{n+n_0}} \in \Theta(n^{\alpha-3/2}) \tag{10}$$

Proof. We apply Taylor's theorem (with remainder) in the form $f(x) = f(y) + f'(y)(x-y) + f''(c)(x-y)^2/2$ with $c \in [x, y]$ on $f = \frac{1}{\sqrt{n}} = n^{-1/2}$ and $x = n$, $y = n + n_0$.

$$\begin{aligned} n^{-1/2} = &\quad (n+n_0)^{-1/2} + n_0(n+n_0)^{-3/2} + \tfrac{(c)^{-5/2}{n_0}^2}{2} \\ n^{-1/2} - (n+n^\alpha)^{-1/2} = &\quad n^{\alpha-3/2} + O(n^{2\alpha-5/2}) \end{aligned}$$

□

8.3 Chernoff Version of Bounds

Our technical trick is to exponentiate the bounds, i.e. (e^{tX}) and then apply a markov inequality. We then optimize our choice of t to lower the probability.

$$\begin{aligned}
\mathbf{P}[\text{Progress is violated}](n, n_0, \delta) \leq\ & \mathbf{P}[(f(n,\delta) - f(n+n_0,\delta)) < (\rho(n+n_0) - \rho(n))] + \\
& \mathbf{P}[(f(n,\delta) - f(n+n_0,\delta)) < (\rho(n) - \rho(n+n_0))] && \text{Union Bound} \\
\leq\ & 2\mathbf{P}[-e^{t(f(n,\delta)-f(n+n_0,\delta))} < e^{t(\rho(n+n_0)-\rho(n))^2}] && \text{Justified below} \\
\leq\ & 2\mathbf{E}[e^{t(\rho(n+n_0)-\rho(n))}]e^{-t(f(n,\delta)-f(n+n_0,\delta))} && \text{Markov after } e^t
\end{aligned}$$

Our job is now to select a value of t that minimizes this expression.

8.3.1 The Expectation term

We prove the following lemma, which shows that the terms we wish to control are infact $O(1)$. In other words, for even a value of $t << n$ we have that the likely error is bounded.

Lemma 8.2. *If $t << n + n_0$ then $\boldsymbol{E}[e^{t(\rho(n+n_0)-\rho(n))}]$ and $\boldsymbol{E}[e^{t(\rho(n)-\rho(n+n_0))}]$ are both $O(1)$.*

Proof. We do the calculation for $\mathbf{E}[e^{t(\rho(n+n_0)-\rho(n))}]$,

$$\begin{aligned}
\mathbf{E}[e^{t(\rho(n+n_0)-\rho(n))}] =\ & \mathbf{E}[e^{t(\frac{X(n_0)}{n+n_0} - \frac{n_0\rho(n)}{n+n_0})}] && \text{Decomposition in eq. 8} \\
=\ & \mathbf{E}[e^{\frac{tX(n_0)}{n+n_0}}]\mathbf{E}[e^{-\frac{tn_0\rho(n)}{n+n_0}}] && \text{independence of } X(n_0), \rho(n)
\end{aligned}$$

Thus we only need toestablish that if $t << n + n_0$ then

1. $\mathbf{E}[e^{\frac{tX(n_0)}{n+n_0}}] \in O(1)$
2. $\mathbf{E}[e^{\frac{tn_0\rho(n)}{n+n_0}}] \in O(1)$

First term: $\mathbf{E}[e^{\frac{tX(n_0)}{n+n_0}}]$

$$\begin{aligned}
\mathbf{E}[e^{\frac{tX(n_0)}{n+n_0}}] =\ & \mathbf{E}[e^{\sum_{i=1}^{n_0}\frac{tx_i}{n+n_0}}] && \text{Definition of } X(n_0) \\
=\ & \textstyle\prod_{i=1}^{n_0} \mathbf{E}[e^{\frac{tx_i}{n+n_0}}] && \text{independence of } \{x_i\} \\
=\ & \textstyle\prod_{i=1}^{n_0} (e^{\frac{t}{n+n_0}}p + 1 - p) && \text{Expectation} \\
=\ & \textstyle\prod_{i=1}^{n_0} ((e^{\frac{t}{n+n_0}} - 1)p + 1) && \text{factoring } p \\
<\ & \textstyle\prod_{i=1}^{n_0} e^{e^{\frac{t}{n+n_0}}-1}p && 1 + x < e^x \\
=\ & e^{n_0 p(e^{\frac{t}{n+n_0}}-1)} && \text{exp rules}
\end{aligned}$$

We observe that for $t << n + n_0$, then this term goes to $e^0 = 1$ as desired.

Second Term: $\mathbf{E}[e^{\frac{tn_0\rho(n)}{n+n_0}}]$ We deal with the second expectation term.

$$\begin{aligned}
\mathbf{E}[e^{\rho(n)\frac{t}{(n+n_0)}}] =\ & \mathbf{E}[e^{\sum_{i=1}^{n} y_i \frac{t}{n(n+n_0)}}] \\
=\ & exp(np(e^{\frac{t}{n(n+n_0)}} - 1)) && \text{Same argument}
\end{aligned}$$

Now, we observe that for $t << n(n + n_0)$ this term will also go to 1. □

Corollary 8.3. *Let $\tau > 0$. If $t << n + n_0$ then with high probability*

$$\rho(n + n_0) - \rho(n) \le n^{-1+\tau}$$

Proof. We know from above that $\mathbf{E}[e^{t(\rho(n+n_0)-\rho(n))}] \in O(1)$ for all $t << n + n_0$ thus choose $t = n^{1-\tau/2}$, using a markov bound we have

$$\begin{aligned} \mathbf{P}[\rho(n+n_0) - \rho(n) \le n^{-1+\tau}] \le & \quad \mathbf{E}[e^{n^{1-\tau/2}(\rho(n+n_0)-\rho(n))}]e^{-n^{1-\tau/2}n^{-1+\tau}} \\ = & \quad O(e^{-n^{\tau/2}}) \end{aligned}$$

□

8.3.2 The Denominator

Lemma 8.4. *For any choice of t,*

$$e^{-t(f(n,\delta)-f(n+n_0,\delta))} \le e^{2tm\log\frac{2}{\delta}n^{\alpha-3/2}}$$

Proof.

$$\begin{aligned} e^{-t(f(n,\delta)-f(n+n_0,\delta))} = & \quad e^{2tm\log\frac{2}{\delta}(n^{-1/2}-(n+n_0)^{-1/2})} \\ \le & \quad e^{2mt\log\frac{2}{\delta}n^{\alpha-3/2}} \qquad \text{Lemma 8.1} \end{aligned}$$

□

8.3.3 The Punchline

Combining lemma 8.4 and 8.2, we have

$$t << n + n_0 \implies \mathbf{P}[\text{Progress is violated}](n, n_0, \delta) \le 2e^{2mt\log\frac{2}{\delta}n^{\alpha-3/2}}$$

We take $t = n_0 = n^\alpha$ for $\alpha < 1$, this satisfies $t << n + n_0$ and substitution yields,

$$\mathbf{P}[\text{Progress is violated}](n, n_0, \delta) \le 2e^{2m\log\frac{2}{\delta}n^{2\alpha-3/2}}$$

If we take $t = n^\alpha = \frac{3}{4} + \alpha'$, for $\alpha' \in [0, \frac{1}{4})$ then our probability simplifies to:

$$\mathbf{P}[\text{violation}](n, \alpha', \delta) \le 2e^{-2m\log\frac{2}{\delta}n^{\alpha'}} \tag{11}$$

Since this is exponentially small probability, the number and variance of errors are exponentially small. Thus by applying a Chebyshev inequality we can conclude with exponentially high probability there are no errors after n^α steps from any fixed n.

Theorem 8.5. *The probability that progres holds after some negligble number of steps tends to 1 exponentially quickly.*

Remark 8.2. *Also as long as $\alpha < 1$, our optimality ratio is preserved.*