Efficient Top-k Query Evaluation on Probabilistic Data

Christopher Ré Nilesh Dalvi Dan Suciu
University of Washington
Deptartment of Computer Science
{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.

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.

A recent approach to manage imprecisions is with a *probabilistic database*, which uses probabilities to represent the uncertainty about the data [5, 6, 7, 8, 21]. A simplistic definition is that every tuple belongs to the database with some probability, whose value is between 0 and 1, and, as a consequence, every tuple returned by a SQL query will have some probability, reflecting the systems' confidence in the answer.

```
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. Fragment of IMDB and Amazon schemas

A major challenge in probabilistic databases is query evaluation. Dalvi and Suciu [6] have shown recently that most SQL queries are #P-complete, which rules out efficient evaluation algorithms. Previous approaches to query evaluation on probabilistic databases have either restricted the queries [2, 5, 8], or modified the semantics [17], or were not scalable [10].

In this paper we propose a new approach to query evaluation on probabilistic databases, by combining top-k style queries with approximation algorithms with provable guarantees. More precisely, we compute and rank the top k answers (in order of their probabilities) of a SQL query. We *guarantee* the correct ranking of the top k answers, but only approximate their probabilities to the extent needed to compute their ranking. Thus, the users specifies a SQL query and a number k, and the system returns the highest ranked k answers, which are guaranteed to be correct: the probabilities of these answers are reported too, but they may be approximate. 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 probabilities to ranks, and give a new, provably optimal algorithm for computing the top k answers.

Example We illustrate with an application that integrates the Internet Movie Database from `imdb.com`, with movie reviews from `amazon.com`. Fig. 1 shows a simplified version of the real schema; the data has over 10M tuples. Amazon has data on products (DVDs), uniquely identified by an Amazon Standard Identification Number, `asin`, while

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.

IMDB has data on movies, directors, and actors.

From Imprecisions to Probabilities The movie titles in the two databases often don't match, e.g. `Twelve Monkeys` v.s. `12 Monkeys`. This is one source of imprecision, and we illustrate how it can be addressed with probabilities. The idea is to compute for each pair of movie title and review title a similarity score, which is a number between 0 and 1 and is interpret it as the probability that the two objects match. The similarity scores are stored in the table TitleMatch^p, Fig. 2. There is a rich literature on record linkage (also known as de-duplication, or merge-purge) [1, 4, 9, 11, 14, 15, 22, 23], that offers an excellent collection of techniques for computing these similarity scores. However, the traditional way of using these scores is to compare them to a threshold and classify objects into matches and non-matches [1]: this forces the user to make a difficult choice between high recall (low threshold) and high precision (high threshold). Instead, a probabilistic database keeps all potential matches, and uses them during query processing to compute a probability for each answer to a SQL query.

SQL Queries The query in Fig. 3 retrieves all directors that produced a lowly rated comedy and a highly rated drama less than five years apart. There are 1415 answers to this query: Fig. 4 shows the top 5 (in decreasing order of their probabilities). Consider one such answer, say `Stanley Donen`. The system computed its probability 0.88 by considering all low ranking reviews that may match one of his comedies, and all high ranking reviews that may match one of his dramas, and accounting for the probabilities of these matches: the exact semantics is based on *possible worlds*, and is reviewed in Sec. 2.1. Note that, unlike the threshold approach, a poor match between a movie by `Donen` and a review is not automatically discarded, but kept and used during query processing: its low probability however is taken into account when ranking `Donen` in the query's answer.

Challenges Query evaluation poses two major challenges. The first is that computing the exact output probabilities is computationally hard: the query in Fig. 3 is #P-complete (based on results in [6]), meaning that any algorithm computing the output probabilities needs to iterate through all possible worlds (in this case: all possible subsets of TitleMatch^p). Previous work on probabilistic databases avoided this issue in several ways. Barbara [2] requires the SQL queries to include all keys in all tables in the `SELECT` clause, thus disallowing duplicate elimination. In Fig. 3 the user has to include the keys of all seven tables, hence each director is returned multiple times, making ranking of the directors impossible. Lakshmanan [17] computes probability intervals instead of exact probabilities. However, unlike our approach based on Luby and Karp's algorithm [16] which can approximate the probabilities to an *arbitrary precision*, the precision in [17] is *fixed*: the more complex the query, the wider the approximation intervals end up, and may become [0,1] in complex queries, making them impractical for ranking purposes. Fuhr [10] uses an exponential time algorithm that essentially iterates over all possible worlds that support a given answer. This is again impractical in our setting. Recent work [19], handles the problem of computing Top-K queries when the uncertain data is specified by "generation rules", in our case the complexity from the complexity of queries over the uncertain data. Finally, Dalvi [6] only considers "safe" queries, while our query is not safe.

```
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<2 and b.rating>4
```

Figure 3. A SQL query: Finding directors with a highly rated Drama and low rated comedy.

The second challenge is that the number of potential answers for which we need to compute a probability is large: in our example, there are $n = 1415$ 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 user is likely to end up inspecting just the first few of them.

Our Approach focuses the computation only on the top k answers, in the order of probabilities. A naive top k method is to compute all n 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

Rank	Director	p
1	Frank Capra	0.99
2	Charles Chaplin	0.98
3	Robert (I) Altman	0.93
4	Stanley Donen	0.88
5	Joseph H. Lewis	0.86
	...	...

Figure 4. Top 5 answers (out of $n = 1415$)

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 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 method turns out to be orders of magnitude more efficient than the naive approach.

Contributions This work introduces the multisimulation algorithm which enables efficient processing of probabilistic queries with provable error guarantees.

Limitations In this paper we restrict ourselves 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. Graphical models like Bayesian Networks and Probabilistic Relational Models can represent such distributions much more concisely. Also, we do not handle continuous attribute values [8, 5].

Other Related Work The statistics literature has considered the *statistical selection problem*, where the problem is to find the "best" (i.e. highest mean) of a finite set of alternatives: see [3, 12] for recent surveys: this corresponds to our setting with $k = 1$ and small n (say $2 \dots 5$). The focus of that work is on tight probabilistic guarantees for a variety of concrete probabilistic distributions; our work assumes simpler distributions but emphasizes the algorithm complexity (since our n is in the range of thousands and $k = 10 \dots 50$).

2 Preliminaries

2.1 Probabilistic Databases

We introduce here a basic probabilistic data model. It corresponds to ?-sets and or-sets in [7, 13].

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

$\texttt{TitleMatch}^p$

	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	$\emptyset$	$(1\text{-}p_1\text{-}p_2\text{-}p_3)\ (1\text{-}p_4\text{-}p_5)$
2	t_1	$p_1(1\text{-}p_4\text{-}p_5)$
3	t_2	$p_2(1\text{-}p_4\text{-}p_5)$
4	t_3	$p_3(1\text{-}p_4\text{-}p_5)$
5	t_4	$(1\text{-}p_1\text{-}p_2\text{-}p_3)p_4$
6	t_1t_4	p_1p_4
7	t_2t_4	p_2p_4
8	t_3t_4	p_3p_4
9	t_5	$(1\text{-}p_1\text{-}p_2\text{-}p_3)p_5$
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

Definition 2.1. *A probabilistic database over schema S is a pair $(\mathcal{W}, \mathbf{P})$ where $\mathcal{W} = \{W_1, \dots, W_n\}$ 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. Instead, we represent a probabilistic database by using a modified schema S^p, called *probabilistic schema*, where some tables R_i are replaced with probabilistic tables R_i^p, that have an explicit probability attribute: $Attr(R_i^p) = Attr(R_i) \cup \{p\}$, and $Key(R_i^p) = Attr(R_i)$. We impose the constraints that p is in $[0, 1]$ and for every values $\bar{a}$ of the $Key(R_i)$ attributes, $sum(\Pi_p(\sigma_{Key(R_i)=\bar{a}}(R_i^p))) \leq 1$: the intuition is that the set of tuples that share the same values $\bar{a}$ represent exclusive choices (called "or-tuples" in [7]) hence their probabilities sum to ≤ 1. Each instance J^p over schema S^p represents a probabilistic database over the schema S, denoted $Mod(J^p)$, defined as follows. Assume for illustration that S has a single relation name, $R(\underline{A_1, \dots, A_m}, B_1, \dots, B_n)$, in notation $R(\underline{\bar{A}}, \bar{B})$ (here $Key(R) = \{A_1, \dots, A_m\} = \bar{A}$), hence J^p is an instance of the table $R^p(\bar{A}, \bar{B}, p)$. The possible worlds $\mathcal{W} = \{W_1, \dots, W_n\}$ are defined as all subsets W_j of $\Pi_{\bar{A},\bar{B}}(J^p)$ s.t. the attributes $\bar{A}$ form a key. For each possible world W_j its probability is defined as $\mathbf{P}(W_j) = \prod_{\bar{a} \in \Pi_{\bar{A}}(J^p)} p_{W_j}(\bar{a})$, where:

$$p_{W_j}(\bar{a}) = \begin{cases} p \text{ if } \exists \bar{b} \text{ s.t. } (\bar{a}, \bar{b}) \in W_j \text{ and } (\bar{a}, \bar{b}, p) \in J^p \\ 1 - sum(\Pi_p(\sigma_{\bar{A}=\bar{a}}(J^p))) \text{ otherwise} \end{cases}$$

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 Continuing our motivating example, consider the table $\texttt{TitleMatch}^p$: we illustrate the possible worlds semantics in Fig. 5. There are 12 possible worlds (shown on the right), namely all subsets of $\texttt{TitleMatch}^p$ (left) where asin is a key. Note that we must have $p_1, \ldots, p_5 \in [0,1]$, $p_1 + p_2 + p_3 \leq 1$, $p_4 + p_5 \leq 1$.

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 of the form:

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

The aggregate operators can be sum, count (which is sum(1)), min and max; we do not support avg.

Possible Worlds Semantics A principled semantics to a query with aggregates was given in [6], and consists of computing the query on all possible worlds, and adding up the probabilities of those worlds where a certain tuple occurs as an answer. Formally, given a probabilistic database $(\{W_1, \ldots, W_n\}, \mathbf{P})$, the answer to the query q in (1) 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
...	...	...	...	...	...	...

Consider one possible world, W_j. If we were to evaluate the SQL query q on W_j, the answer $q(W_j)$ is a set of tuples $(\bar{b}, \bar{a})$. Fix a tuple $\bar{b}$ and define the predicate $C_{\bar{b}}$ and function $F_{\bar{b}}$ on possible worlds:

$$\begin{array}{rcl} C_{\bar{b}}(W_j) & = & (\exists \bar{a}.(\bar{b}, \bar{a}) \in q(W_j)) \\ F_{\bar{b}}(W_j) & = & \begin{cases} \bar{a} & \text{if exists } \bar{a} \text{ s.t. } (\bar{b}, \bar{a}) \in q(W_j) \\ \text{undefined} & \text{otherwise} \end{cases} \end{array}$$

Recall the standard definitions of the probability of a predicate, and the conditional expected value:

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

Definition 2.4. *The "possible worlds semantics" of a query q on a probabilistic database $(\mathcal{W}, \mathbf{P})$ is given by:* $q(\mathcal{W}, \mathbf{P}) = \{(\bar{b}, \bar{e}, p) \mid \exists W_j . C_{\bar{b}}(W_j), \bar{e} = \mathbf{E}(F_{\bar{b}} \mid C_{\bar{b}}), p = \mathbf{P}(C_{\bar{b}})\}$

Semantics based on DNF Formulas The possible worlds semantics is not practical for deriving a query evaluation algorithm, so we present an equivalent semantics which reduces the query evaluation problem to computing the probabilities of certain DNF formulas. First, modify the SQL query q to obtain an *expanded* query qe, by removing the GROUP BY clause and returning *:

$$\texttt{qe} = \texttt{SELECT * FROM } \overline{\texttt{R}} \texttt{ WHERE C}$$

where $\bar{R} = R_1, \ldots, R_r$ and C are the same as in Eq.(1). Evaluate qe on the probabilistic instance J^p (using any SQL engine) and denote the answer ET. Each tuple $t \in ET$ has the form $t = (t^1, \ldots, t^r)$, where $t^1 \in R_1^p, \ldots, t^r \in R_r^p$. Define the following boolean expression associated to t:

$$t.E = t^1 \wedge t^2 \wedge \ldots \wedge t^r \quad (2)$$

Note that $\mathbf{P}(t.E)$ can be computed easily: eliminate duplicate tuples then multiply their probabilities; or, if any two tuples are exclusive, then $\mathbf{P}(t.E) = 0$.

Next, partition the set ET by the GROUP-BY attributes $\overline{\texttt{B}}$: $ET = G_1 \cup G_2 \cup \ldots \cup G_n$. For each group $G \in \{G_1, \ldots, G_n\}$, $G = \{t_1, \ldots, t_m\}$, define the following DNF boolean expression:

$$G.E = \bigvee_{t \in G} t.E = t_1.E \vee \ldots \vee t_m.E \quad (3)$$

The query's semantics can be phrased in terms of the DNF formulas (3): namely, each group G in ET determines one probabilistic tuple in the answer, and the probability is $\mathbf{P}(G.E)$. Formally, denoting $G.\overline{\texttt{B}}$ the tuple $t.\overline{\texttt{B}}$ for some $t \in G$ (it is independent on the choice of $t \in G$), we have the following alternative semantics:

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

$$\begin{array}{rcl} \bar{b} & = & G.\overline{\texttt{B}} \text{ for some } G \in \{G_1, \ldots, G_n\} \\ p & = & \mathbf{P}(G.E) \\ e_i & = & \sum_{t \in G} \mathbf{P}(t.E) * t.A_i / p \text{ if } \texttt{AGG}_i = \texttt{sum}(\texttt{A}_i) \\ e_i & = & \sum_{t \in G} (1 - \mathbf{P}(\bigvee_{t' \in G | t'.A_i \geq t.A_i} t'.E)) * t.A_i / p \text{ if } \texttt{AGG}_i = \texttt{max}(\texttt{A}_i) \end{array}$$

Thus, we have reduced the query answering problem to the problem of evaluating probability expressions for DNF formulas, in particular $\mathbf{P}(G.E)$: the other expressions in Theorem 2.5 are either easy to compute or other DNF formulas. However, Valiant [20] has shown that computing the probability $\mathbf{P}(G.E)$ is #P-complete in general.

Example 2.6 Consider the query q in Fig. 3. The extended query qe 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: (a,b,$\mathtt{ax}^p$,$\mathtt{by}^p$,x,y,d). Of these only $\mathtt{ax}^p$,$\mathtt{by}^p$ are probabilistic tuples, and the superscript p indicates 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:

$$\mathbf{P}(t.E) = \begin{cases} \mathtt{ax}^p.\mathtt{p} & \text{if } \mathtt{ax}^p.\mathtt{asin} = \mathtt{by}^p.\mathtt{asin} \text{ and } \mathtt{ax}^p.\mathtt{mid} = \mathtt{by}^p.\mathtt{mid} \\ 0 & \text{if } \mathtt{ax}^p.\mathtt{asin} = \mathtt{by}^p.\mathtt{asin} \text{ and } \mathtt{ax}^p.\mathtt{mid} \neq \mathtt{by}^p.\mathtt{mid} \\ \mathtt{ax}^p.\mathtt{p} * \mathtt{by}^p.\mathtt{p} & \text{if } \mathtt{ax}^p.\mathtt{asin} \neq \mathtt{by}^p.\mathtt{asin} \end{cases}$$

Next, we group the rows by their directors, and for each group $G = \{(\mathtt{ax}_1^p, \mathtt{by}_1^p), \ldots, (\mathtt{ax}_m^p, \mathtt{by}_m^p)\}$ construct the DNF formula: $G.E = \mathtt{ax}_1^p \wedge \mathtt{by}_1^p \vee \ldots \vee \mathtt{ax}_m^p \wedge \mathtt{bx}_m^p$. The director's probability give by $\mathbf{P}(G.E)$: this is a 2DNF (computing its probability is still #P-hard).

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, $p = \mathbf{P}(G.E)$, given by Eq.(3). We need to rank the tuples by p, and return the top k.

Monte Carlo (MC) Simulation An MC 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 2.2.1, 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 $p \in [a^N, b^N]$ that shrinks as N increases. Formally:

Theorem 2.7. *[16] Let $\delta > 0$, m = number of disjuncts (see Eq.(3)) and N the number of steps executed by the Luby and Karp algorithm. Define:*

$$\varepsilon = \sqrt{4m\log(2/\delta)/N} \quad a^N = \tilde{p} - \varepsilon \quad b^N = \tilde{p} + \varepsilon$$

Algorithm 2.2.1 Luby-Karp algorithm for computing the probability of a DNF formula $G.E$ (Eq.(3)).

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

Then[1]*:*

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

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. The goal is to minimize the total number of simulation steps.

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 $\mathtt{TopK} \subseteq \mathcal{G}$: 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:

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

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

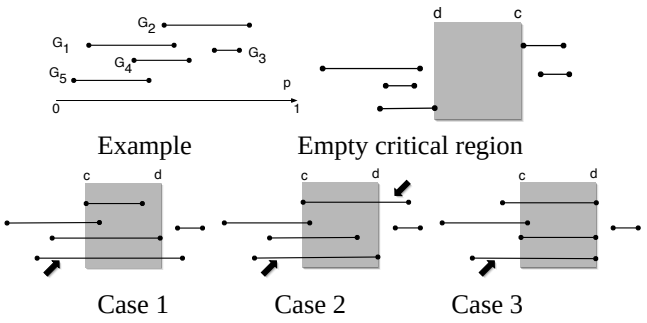


Figure 6. Illustration of MS; $k = 2$. Intervals represent uncertainty about the value of a tuple's probability score.

The separation assumption implies that TopK has a unique solution (no ties) and the other three imply that the solution can be found naively by a round robin algorithm. In our setting each object G is a group of tuples, its probability is $p = \mathbf{P}(G.E)$ (Eq. (3)), and the simulation algorithm is Luby-Karp. Only convergence holds: we revisit below the other three assumptions.

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 $\mathsf{TopK} = T$. 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 $[a_1^{N_1^{opt}}, b_1^{N_1^{opt}}]$, ..., $[a_n^{N_n^{opt}}, b_n^{N_n^{opt}}]$ are k-separated, and (b) the sum $N^{opt} = \sum_i N_i^{opt}$ is minimal. 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 $p_1 < p_2$. 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:*

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

Fig. 6 illustrates four critical regions.

One can check that always $B \cap \mathtt{TopK} = \emptyset$ and $T \subseteq \mathtt{TopK}$. Moreover, there is a k-separation iff the critical region is empty, i.e. $c \geq d$, in which case $\mathtt{TopK} = T$. 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, and call G_i a **crosser** if $[c, d] \subseteq [a_i, b_i]$. Further:

G_i **is a double crosser** if $a_i < c$, $d < b_i$

G_i **is a lower(upper) crosser** if $a_i < c$ ($d < b_i$)

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. We prove in [18]:

Algorithm 3.1.1 The Multisimulation Algorithm

$\mathbf{MS\_TopK}(\mathcal{G}, k)$: /* $\mathcal{G} = \{G_1, \ldots, G_n\}$ */
Let $[a_1, b_1] = \ldots = [a_n, b_n] = [0, 1]$, $(c, d) = (0, 1)$
while $c \leq d$ **do**
  Case 1: choose a double crosser to simulate
  Case 2: choose upper and lower crosser to simulate
  Case 3: choose a maximal crosser to simulate
  Update (c, d) using Eq.(5)
end while
return $\mathtt{TopK} = T = \{G_i \mid d \leq a_i\}$

Theorem 3.3. *(1) The algorithm always terminates and returns the correct* TopK*. (2) Its cost 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* $\geq cN^{opt}$*.*

Corollary 3.4. *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 We show now how to extend MS to compute *and rank* the top k answers: we call the extension **MS_RankK**. First, compute the top k, $T_k = \mathbf{MS\_TopK}(\mathcal{G}, k)$. Next, compute the following sets, in this sequence[2]:

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

After step j, T_j is the set of top j answers; by computing next the top $j-1$ objects in this set, we have identified the j's ranked object. Thus, the ranks of the top k objects are found in reverse order. This algorithm is also "optimal" i.e. within a factor of two of an optimal oracle. A useful variation is an *any-time* algorithm, which computes and returns the top answers in order 1, 2, 3, ..., and can be stopped at any time: this algorithm is *not* optimal for finding the top k.

Reviewing the assumptions Precision holds in a probabilistic sense at each step: $\mathbf{P}(p \in [a^N, b^N]) > 1 - \delta$. However, the user chooses some δ_0 and wants a guarantee that precision holds throughout the execution of the simulation algorithm with probability $> 1 - \delta_0$: we call this *global precision*. Let N be an upper bound on the total number of steps required by the algorithm. Define δ s.t. $(1-\delta)^N \geq 1 - \delta_0$, hence $\delta_0 \approx \delta_0/N$ and use δ in the MC algorithm: now global precision holds. Note that this affects only slightly the width of the intervals (and, hence, the convergence ratio), since $b^N - a^N = 2\sqrt{4m\log(2/\delta)/N}$.

Progress fails in general: after step N of the Monte Carlo algorithm the midpoint $\tilde{p}$ can move (left or right) by $\approx 1/N$, while the width of the interval shrinks by only $O(1/\sqrt{N} - 1/\sqrt{N+1}) = O(N^{-3/2})$ and $[a^{N+1}, b^{N+1}]$ is not contained in $[a^N, b^N]$ because $N^{-1} > N^{-3/2}$. Our solution here is to run the Monte Carlo algorithm for $\sqrt{N}$ iteration at each step of the multisimulation algorithm, instead of just one step: we show in [18] that $\forall \alpha, \tau \in (0,1), \exists N_0$ s.t. $\forall N \geq N_0$ the midpoint $\tilde{p}$ moves between steps N and $N + N^\alpha$ by $\leq N^{\tau-1}$ with very high probability. At the same time the interval shrinks by $O(1/\sqrt{N} - 1/\sqrt{N+N^\alpha}) = O(N^{\alpha-3/2})$: thus, we have progress for $\alpha \geq 1/2$, and we show that the progress is global progress (at all steps). The resulting multisimulation algorithm runs in at most $2(N^{opt} + \sqrt{N^{opt}})$ steps, which asymptotically has a competitive ratio of 2.

The separation assumption also fails in practice since probabilities are often equal or very close to each other. Here we simply rely on a second parameter $\varepsilon > 0$ and stop the simulation when the critical region becomes less than ε.

Optimizations We have added two optimizations to the multisimulation algorithm. We present them here only at a very high level for lack of space, and refer the reader to [18]

[2] $\mathbf{MS\_TopK}_{ni}$ refers to **MS_TopK** without the initialization on the first line.

Probabilistic Table Name	#Tuples	#exclusive tuples	
		Avg.	Max
MovieToAsin	339095	4	13
AmazonReviews	292680	1	1
ActorMatch	6758782	21	2541
DirectorMatch	18832	2	36

Figure 7. Data used in experiments

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)

for details. The first optimization initializes the intervals $[a_i, b_i]$ to better estimates than [0,1]: this eliminates many low ranking objects from the start. In the second optimization, *safe plan rewriting*, we identify subqueries for which the probabilities can be computed inside of the SQL engine (this expands ideas in [6]). For example if a group of the subquery consists of events with probabilities $p_1, \ldots, p_n$, then the probability of the entire group is $p_1 + \ldots + p_n$ (when all events are disjoint) or $1 - (1 - p_1)(1 - p_2) \ldots (1 - p_n)$ (when all events are independent), and both expressions can be computed in SQL. We refer to [18] for details.

4 Experiments

In this section we evaluate our approach experimentally. We address four questions: (1) how does our new query evaluation method compare to the current state of the art query processing on probabilistic databases; (2) how effective is the multisimulation (MS) over a naive application of Monte Carlo (3) how effective are the optimizations; and (4) how sensitive is the system's performance on the choice of δ_0 and ε (Sec. 3)

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, 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.

Data consists of an integration of the IMDB movie database with reviews from Amazon, as described in a simplified form in Sec. 1. 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"). Some statistics are shown in Fig. 7.

Queries We report experiments on four queries that illustrate different scales for the number of groups and the average size of each group (n and m in Sec. 3): each of n and m can be small (S) or large (L) resulting in the four queries below, whose statistics are shown in Fig. 8:

SS In which years did *Anthony Hopkins* appear in a highly rated movie?

SL Find all actors who were in Pulp Fiction who were in two very bad movies in the five years before Pulp Fiction.

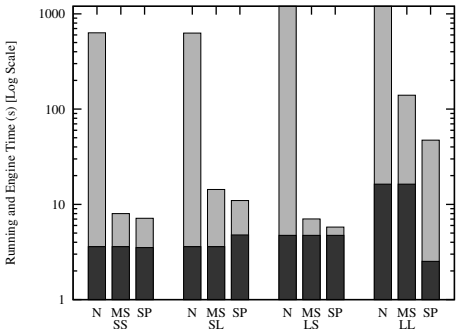
LS Find all directors who had a low rated movie between 1980 and 1989.

LL Find all directors who had a low rated drama and a high rated comedy less than five years apart.

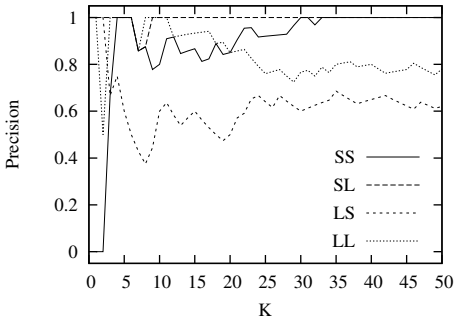
Methodology For each running time we perform the experiment 5 times, dropping the highest and the lowest and average the remaining three runs. The naive simulation method was capped at 20 minutes. In between each experiment, we force the database to terminate all connections. The same experiments was 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 Luby and Karp algorithm with very low settings for ε and δ: $\varepsilon = 0.001$ and $\delta = 0.01$.

Unless otherwise stated, the confidence and precision parameters were $\varepsilon = .01$, $\delta_0 = .01$, 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 [17] 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. Also note that one of the queries

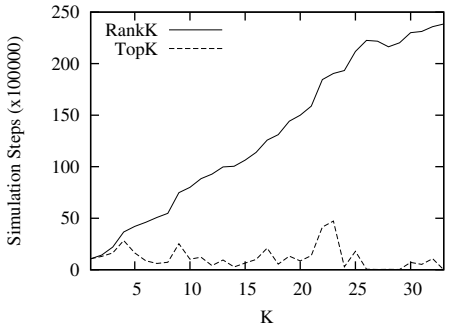


(a) Running times N(aive), MS, and S(afe) P(lan)

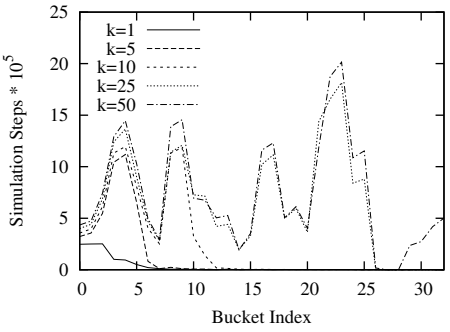


(b) Precision/Recall for naive strategies

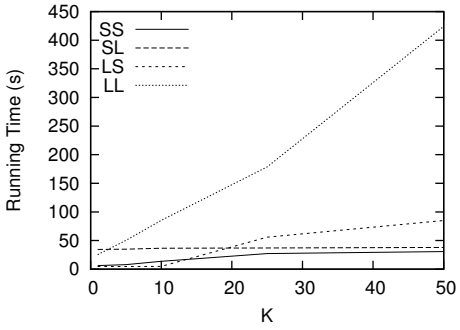
$[k = 10, \varepsilon = .01, \delta_0 = .01]$



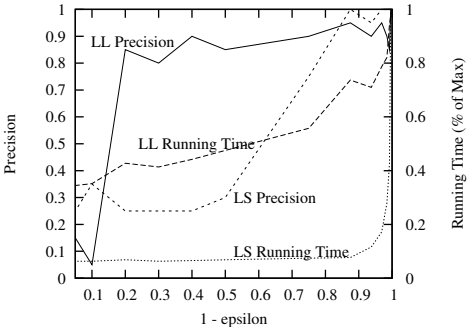
(c) Total number of simulation steps for query SS



(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) 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 than 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. The fewer answers the user requests, the faster they are retrieved.

Effectiveness of the Optimizations The more important optimization is the safe-plan rewriting (SP), 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_0 and ε (see Sec. 3.2). Recall that the theoretical running time is $O(1/\varepsilon^2)$ and $O(\log(1/(n\delta_0))$. 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; $k = 20$, $\delta_0 = 0.01$, 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_0 differ, and is much more boring: the precisions/recall reaches 1 very fast, while the running time is almost independent on δ_0. (The graphs look almost like two horizontal lines.) We can choose δ_0 in a wide range without degrading either precision/recall or performance.

5 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.

Acknowledgements The authors would like to thank the anonymous referees for their comments. This work was partially supported by Suciu's NSF CAREER grant IIS-00992955 and the NSF grants IIS-0428168 and 0513877.

References

[1] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, pages 586–597, 2002.

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

[3] J. Branke, S. Chick, and C. Schmidt. New developments in ranking and selection: an empirical comparison of three main approaches. In *Proceedings of the Winter Simulation Conference*, pages 708–717, 2005.

[4] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *ACM SIGMOD*, San Diego, CA, 2003.

[5] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, pages 551–562, 2003.

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

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

[8] 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.

[9] I. Felligi and A. Sunter. A theory for record linkage. *Journal of the American Statistical Society*, 64:1183–1210, 1969.

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

[11] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, pages 371–380, 2001.

[12] D. Goldsman and B. Nelson. Satistical selection of the best system. In *Proceedings of the Winter Simulation Conference*, pages 139–146, 2001.

[13] T. J. Green and V. Tannen. Models for incomplete and probabilistic information. *IEEE Data Engineering Bulletin*, 29(1):17–24, March 2006.

[14] L. Gu, R. Baxter, D. Vickers, and C. Rainsford. Record linkage: Current practice and future directions. In *CMIS Technical Report No. 03/83*, 2003.

[15] M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995.

[16] R. Karp and M. Luby. Monte-carlo algorithms for enumeration and reliability problems. In *STOC*, 1983.

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

[18] C. Ré, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data (extended version). Technical Report 2006-06-05, University of Washington, Dept. Computer Science, 2006. manuscript.

[19] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Top-k query processing in uncertain databases. In *Proceedings of ICDE*, 2007.

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

[21] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.

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

[23] W. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Bureau of the Census, 1999.